A GUI INTERFACE FOR REUSABLE COMPONENTS STORAGE AND RETRIEVAL IN THE CAPS SOFTWARE BASE

by

Gregory L. Meckstroth

June 1998

Thesis Advisor: Luqi
Co-Advisor: V. Berzins

Approved for public release; distribution is unlimited.
**Title:** A GUI Interface for Reusable Components Storage and Retrieval in the CAPS Software Base

**Abstract:**
With the increase in size and complexity of software component repositories, the need for an easy-to-use search and retrieval process becomes a necessity. Multilevel filtering shows great promise as a quick accurate search algorithm. This approach applies a series of filters starting with high recall, low precision syntactic techniques, moving through a range of more computationally expensive high precision syntactic filters.

The goal of this thesis is to develop a graphical user interface, using multilevel filtering, to make searching the CAPS component repository a less tedious task. The interface will make the retrieval process less error prone. The user would not need to be an expert in how the software base works thus increasing the ease of use and productivity. The current prototype system has a limited user interface capability. This research will add a graphical user interface for both retrieval and maintenance.

**Subject Terms:**
- Software Reuse
- User Interface
- Multilevel Filtering
- Profile Matching
- Signature Matching
A GUI INTERFACE FOR REUSABLE COMPONENTS STORAGE
AND RETRIEVAL IN THE CAPS SOFTWARE BASE

Gregory L. Meckstroth
B.S., San Diego State University, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1998

Author: Gregory L. Meckstroth

Approved by: Luqi, Thesis Advisor

V. Berzins, Co-Advisor

Dan Boger, Chairman
Department of Computer Science
ABSTRACT

With the increase in size and complexity of software component repositories, the need for an easy to use search and retrieval process becomes a necessity. Multilevel filtering shows great promise as a quick accurate search algorithm. This approach applies a series of filters starting with high recall, low precision syntactic techniques, moving through a range of more computationally expensive high precision syntactic filters.

The goal of this thesis is to develop a graphical user interface, using multilevel filtering, to make searching the CAPS component repository a less tedious task. The interface will make the retrieval process less error prone. The user would not need to be an expert in how the software base works thus increasing the ease of use and productivity. The current prototype system has a limited user interface capability. This research will add a graphical user interface for both retrieval and maintenance.
TABLE OF CONTENTS

I. INTRODUCTION
   A. FORMS OF REUSE ................................................................. 1
   B. CURRENT SOFTWARE DEVELOPMENT PRACTICES ..................... 2

II. BACKGROUND ................................................................. 5
   A. CAPS .................................................................................. 5
   B. PSDL .................................................................................. 6
   C. TEXT SEARCH ...................................................................... 8
   D. SYNTACTIC MATCHING ....................................................... 8
   E. PROFILE FILTERING ........................................................... 9
   F. SIGNATURE MATCHING ....................................................... 10
   G. SOFTWARE BASE .............................................................. 11

III. DESIGN AND CONCEPTS ................................................. 13
   A. MULTI-LEVEL FILTERING IMPLEMENTATION ....................... 13
   B. SOFTWARE BASE ORGANIZATION ......................................... 14
   C. DIALOG INTERFACE CONNECTIONS .................................... 15
   D. TEST PROGRAM ............................................................... 16
   E. SYNTAX DIRECTED EDITOR ................................................ 16
   F. SEARCH DISPLAY ............................................................. 19
   G. PROGRESS DISPLAY .......................................................... 24
   H. MAINTENANCE DISPLAY ..................................................... 25

IV. CONCLUSIONS AND FUTURE RESEARCH ...................... 31
   A. ACCOMPLISHMENTS .......................................................... 31
   B. FUTURE RESEARCH ............................................................ 32

LIST OF REFERENCES .......................................................... 33

APPENDIX A C++ GUI SOURCE CODE ................................ 35
   A. CALLBACKS.H .................................................................... 35
   B. CALLBACKS.C ................................................................... 36
   C. DISPLAYPROGRESS.H ......................................................... 42
   D. DISPLAYPROGRESS.C ........................................................ 43
   E. GUI.H ............................................................................... 45
   F. GUI.C ............................................................................... 46
   G. HELPMESSAGES.H ............................................................ 48
   H. MAINTENANCEDIALOG.H .................................................. 50
   I. MAINTENANCEDIALOG.C .................................................... 50
   J. MENU.H ............................................................................. 65
   K. MENU.C ............................................................................. 66
   L. PROMPTDIALOG.H ............................................................. 69
   M. PROMPTDIALOG.C ........................................................... 69
   N. SDE.H ............................................................................... 72
   O. SDE.C ............................................................................... 73
   P. SEARCHDIALOG.H ............................................................ 79
I. INTRODUCTION

As the size and complexity of the software base increases the need for an easy to use query entry, along with quick and accurate search algorithms, becomes a necessity. The goal of this thesis is to develop a graphical user interface to make searching the CAPS software base, or component repository, a less tedious task. It will use profile filtering and signature matching as proposed in [1] Improving Syntactic Matching For Multi-Level Filtering. The current prototype system has a limited user interface capability. This research will add a graphical user interface for the system. The interface will make the retrieval process less error prone. The user would not need to be an expert in how the software base works thus increasing the ease of use and productivity. Search metrics could be collected to aid future improvements to the search algorithms.

A. FORMS OF REUSE

When developing a software project many forms of reuse are commonly practiced. Most often they are in the form of libraries some are include with the compiler, or purchased separately like graphics, static and engineering packages. Finding specific modules in the package is accomplished using a simple text based search engine and a requirements list. This can be adequate for well-understood utilities like the transcendental functions supplied with the compiler. Searching for some intricate engineering functions, where lexical descriptions may vary widely, can be very time consuming. Libraries that are developed in-house, for use on a specific project, will have descriptions that are clear and concise to the author while other members of the team may
find them indistinct. The user can form a search string using terms that are familiar to them but not get a match on a module that would suit their needs.

On large projects, common tasks could benefit from reuse if the source code could be found but it is often easier to rewrite than to search a large software repository for something that is not there or have the search miss the component that is sought.

B. CURRENT SOFTWARE DEVELOPMENT PRACTICES

As the size and complexity of software projects increase and budgets decrease the need for software reuse becomes critical. Software development project managers will specify that software reuse will be practiced as a way of decreasing the cost and increasing the reliability of the software to be developed as part of the proposal process. However, if reuse was not part of the development process in the past, during the actual development they will find that they are not able to benefit from reuse. Software developed without reuse as one of the design criteria will be difficult to transform into a reusable module making it less likely to be reused in future projects. Those with reuse as one of the design goals will be ready for reuse in future projects. One form of designing for reuse is a generic module that can be used in a variety of instances without modification. An example is a generic sorting routine that can sort any standard data type, while a non-generic routine would sort one data type. The need to make even simple modifications to a module may eliminate it from being reused in future projects. When modifications are required to reuse a module, the possibility of introducing errors is very likely. Assumptions can be made by the original author in writing the module that are not
known or understood by those making modifications, thus introducing errors. As the module evolves over time, the module becomes a disaster waiting to happen.
II. BACKGROUND

Some previous works in software engineering and search techniques are presented here as a background for the research presented in this thesis.

A. CAPS

Computer Aided Prototyping System (CAPS) automates the early design phases of developing embedded systems that have strict real time constraints. CAPS represents a working environment consisting of development tools that help systems analysts and programmers to automate the design and implementation of rapid prototypes for hard real time embedded systems [2]. These tools include an execution support system; syntax directed editor, graphical editor, and automatic constructors for scheduling and control code, automated integration of ADA modules, and the framework for the inclusion of components from a software base.

Prototyping in CAPS consists of creating the PSDL description of the system design. This is accomplished with the graphical editor [3], used to create the PSDL skeleton, and the syntax directed editor, that is used to flesh out the skeleton. The PSDL description is then translated into an ADA package that is a driver for the atomic operators. A static scheduler finds a schedule for the time critical operators and produces an ADA package that contains the schedule for the time critical operators that is represents as an ADA task. The dynamic scheduler produces an ADA package for the non-time critical operators. The software base is a repository for reusable components that can be used for the atomic operators. If a reusable component cannot be found the
developer either writes it or decomposes it in an effort to find a reusable component. CAPS will compile and execute the prototype. The prototype is then modified in response to the users’ input. After the users accept the prototypes demonstrated functionality the developer will port the prototype to the target hardware and operating system.

There are two places in CAPS that can benefit from reuse, the construction of the atomic ADA operators in the prototype and the final optimized versions that will make up the delivered product.

B. PSDL

The Prototyping Description Language (PSDL) [4] is used to specify the prototypes in CAPS. This language has data flow like semantics containing operators that communicate via data streams. PSDL programs have two kinds of objects, abstract data types and abstract state machines. The data streams carry values of a fixed abstract data type. Formally, the PSDL model is that of an augmented graph \( G = (V, E, T(v), C(v)) \) where \( V \) is the set of vertices, \( E \) is the set of edges, \( T(v) \) is the maximum execution time for each vertex \( v \), and \( C(v) \) is the set of control constraints for each vertex \( v \). Each vertex is an operator and each edge is a data stream.

An Operator is either a function or a state machine. When an operator fires, it reads the input data stream and writes zero or one date object to its output streams. The output depends only on the current set of input values. For state machines, the output depends on the current set of input values and a finite number of internal state variables.
Operators are either atomic or composite. Atomic operators cannot be decomposed any further and are implemented in a programming language. Composite operators are constructed from PSDL components with a lower level of abstractions. This continues until the atomic operator is reached. Figure 1 illustrates the decomposition of the PSDL operator into two atomic operators.

A data stream is a communications link between two operators. Each stream carries a sequence of data values from the produce to the consumer. There are two types of data streams, data flow streams and sampled streams. The data flow stream can be thought of as a FIFO queue where data values are neither lost nor replicated. A sampled stream can be thought of as a cell that contains one value that is updated as the producer generates new values.
The PSDL specification for a component contains the information needed for analyzing and finding reusable independent objects that are contained in the software base. A PSDL specification is independent of the computer language that is used to implement a component making it ideal for query and tag specification.

C. TEXT SEARCH

Text queries are based on keyword matching. The query is specified as a set of keywords. The software base is searched for the given keywords. Any components that match are returned as candidates for the query. For a software base with a large number of components, if the user includes too few keywords they are overwhelmed by the number of candidates. If they use too many keywords, they miss the component because an exact match is not found. An improvement to this technique is to use a faceted approach [5] where keywords are selected from predefined keywords in a faceted list.

The faceted list is a predefined set of keywords that are constructed by experts and are designed to best describe the component. This list must be continually updated as new components are added to the software base. To facilitate the maintenance of the software base on large projects a full time librarian is required. This is an added cost for the project. The fidelity of the faceted list is a function of how well the users and librarian associate the functionality of a component with the keywords that are selected.

D. SYNTACTIC MATCHING

Syntactic matching uses non-behavioral component information, such as a keyword list, package declaration or PSDL specifications [6]. If a query component has
the same interface specification as one from the software base then the components may match. Components with dissimilar interface specifications will not match. These components can be eliminated from consideration as a candidate. This method can be used to quickly eliminate candidates from the search that can not be a match. This leads to a multi-level filtering [7] approach which is organized as a series of increasingly stringent filters that pass only candidates that are an approximate match to the query.

E. PROFILE FILTERING

A profile [6] is a sequence of numbers that describes how the types\(^1\) associated with an operation are organized. For a query to match a component in the software base they must have matching profiles. The profile of an operation is a sequence [6] of integers, defined as follows:

1. The first integer is the total number of occurrences of types.
2. If the total number of type groups, N, is greater than 0, then the second to 
   \((1 + N)^{th}\) integers are the cardinalities of the type groups, in descending order.
3. The \((2 + N)^{th}\) integer is the cardinality of the unrelated sort group.
4. The \((3 + N)^{th}\) integer is:
   
   0 if the value type is different from any of the argument types; and
   
   1 if the value type belongs to some type group.

\(^1\) Note that the dissertation used the words sort and type interchangeably. For clarity, only type will be used.
By calculating, a profile for each component in the software base, like components can be placed into a common partition [6]. When searching for a query it is only necessary to search the partition that contains components with profiles that match the queries profile. All components in other partitions have been eliminated as candidates. This is a fast process that is well suited for the early stages of multi-level filtering where low precision is acceptable and the main goal is to prune the number of candidates for latter high precision high cost filters.

Improvements to increase the resolution of profile filtering have been suggested [1] such as adding more properties to the profile using properties that can be measured with more possible values. By increasing the resolution, we also increase the number of profiles, thus reducing the number of candidates that pass this lever of filtering.

F. SIGNATURE MATCHING

The signature of a module [8] [6] is a triple \((S, N, X)\) where \(S\) is the set of types\(^2\) that appear in the operation signature in \(X\), \(N\) is the set of operation names that appear in the operation signature in \(X\), and \(X\) is a set of operation signatures. An operation signature is a triple containing an operation name, a sequence of input types, and an output type. This definition assumes each operation signature has exactly one output.

A signature match exits if there is a mapping between the query operations and types to the candidate operations and types. A partial signature map [6] is one that does

\(^2\) Note that the class notes use the term sort instead of type. In the interest of continuity sort was changed to type.
not map all of the queries operations. A full signature map successfully maps all of the operations.

G. SOFTWARE BASE

The software base is the repository for the reusable components in CAPS. The software base must be structured [9] so that it will support the automatic retrieval of components based on their specifications. The architecture must support syntactic matching and handle variable numbers of type attributes.
III. DESIGN AND CONCEPTS

The design goal is to have a Graphical User Interface to the program proposed and written in [1] Improving Syntactic Matching For Multi-Level Filtering that will handle interactions between the user and the software base. This will include searches as well as maintenance of the software base.

A. MULTI-LEVEL FILTERING IMPLEMENTATION

Multi-Level Filtering was implemented [1] in ADA using the foreach extension that was add, as a preprocessor, by the Naval Postgraduate School Computer Science Department. It was assumed that all queries and component specifications would be written in PSDL. Extensive use of the CAPS PSDL library was used in handling the query as well as the software base component. All components will be stored in separate directories in the software base. A header file will be used to identify [1] all of the components that comprise the software base. Input to the filtering program was through a file that contains the query. The standard output was used to display the results. This placed restrictions on the implementation of the user interface that uses C++, the X11R6 libraries, and the Motif graphs libraries in how the input and output are performed. One goal was to minimize the modifications to the existing ADA code so input and output will be buffered to a temporary disk file. The internal representation of the software base components was recalculated each time that filtering was done. A modification was made to split the program into two distinct parts maintenance, which saves the initialized data
structures, and searching, which reinitializes the data structure from the those saved during maintenance.

B. SOFTWARE BASE ORGANIZATION

The software base is organized into component modules. All files for a component will reside in a separate component directory. No other directories are allowed in the software base. Each component will have a PSDL specification irrespective of the language of implementation. The PSDL specifications will be used in the filtering process.

Figure 2 Software Base Directory Structure

Figure 2 is an example of the structure of the software base. Set, Tan, and Sequence are component directories in the software base. The header file, sb_header.dat, is used to identify all of the components that comprise the software base. An example is shown in Figure 3.
Each entry contains a unique ID [1] followed by the component directory name. The ID will be used to identify the component in the data structure that internally represents the software base. The ID saves space and is easier to manipulate than a character-based component name. In the original implementation, the component directories could be spread across networked file systems. In the interest of maintainability, the software base will be restricted to a single file system that can be exported to other machines. The pre-computed data structures for the internal representations of the software base components are stored in files that reside in the software base directory.

**C. DIALOG INTERFACE CONNECTIONS**

The graphical user interface to the software base was split into two dialogs. One dialog will be used for maintenance, allowing the addition of new components and the initialization of the data structures. The other dialog will search for components in the software base.

Two conventions were implemented for starting the dialogs, one in the Motif tradition that is non-blocking and uses a callback routine to pass results on completion.
The other a normal "C++" routine blocks until the user is finished. The subroutine returns a pointer to the results.

```c
void SearchDialog(Widget parent, XtCallbackProc callback);
char* ModalSearchDialog(Widget parent);
void MaintenanceDialog(Widget parent);
void ModalMaintenanceDialog(Widget parent);
```

**Figure 4: GUI Interface Routines**

Figure 4 shows the dialog subroutine calls where parent is the widget that the dialog will be display in and callback is the routine that will be called upon completion. The modal versions are blocking. A modeless version of the maintenance dialog was included for completeness.

**D. TEST PROGRAM**

A simple test program was written to test the graphical interface. The only functionality was to initiate the maintenance or search dialog and display the results.

![Test Display](image)

**Figure 5: Test Display**

Pressing one of the buttons (Figure 5) will initiate a call to the appropriate dialog entry routine. The output is displayed on the standard output.

**E. SYNTAX DIRECTED EDITOR**

At the time that this thesis was written the CAPS Syntax Directed Editor (SDE) was unavailable. Creating a true SDE was beyond the scope of this thesis. A simple editor
was created for testing purposes to be used in making changes to the PSDL query/spec. The functionality that is supplied by this editor will be replaced by the CAPS Syntax Directed Editor. The interface to the CAPS SDE was unknown at the time that this editor was written so some modifications to other routines may be necessary. The subroutine SDE.C can be removed or used as the connection point for the CAPS SDE.

```c
void SDE(Widget parent, char **filename, char **string);
```

**Figure 6: SD Editor Calling Convention**

Figure 6 is an example of the SDE subroutine entry point. Parent is the widget that the dialog will be display in. Filename is a pointer to the user-selected file name for the query. If the user canceled the editor or did not select a file name, then file name will be set to the NULL pointer. String is a pointer to the PSDL query that will be used in filtering. If the user canceled the editor or did not enter a query then the string will be set to the NULL pointer.
Figure 7: SD Editor

Figure 7 shows the simple SD Editor display. The user can type the query directly into the edit window. Through the file menu (Figure 8) an existing query can be opened, saved, or saved under a different file name. The close menu entry will close the SD Editor and exit will exit the program. Simple editing capabilities are implemented through the edit menu (Figure 9). Cut copies the selected text to the buffer then deletes the selection. Copy puts the selected text in the buffer. Paste copies the buffer in to the current cursor location.
F. SEARCH DISPLAY

The filtering process has three distinct parts, the PSDL query, profile filtering results, and the signature matching results. The dialog is partitioned into these three areas with their associated input. Figure 10 illustrates the search dialog box and its components.
CAPS Software Base Search

**PSDL Query:** /home/greg/Thesis/NewQuery.pSDL

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SPECIFICATION</th>
<th>GENERIC</th>
<th>OPERATOR</th>
<th>SPECIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td></td>
<td>Item: DISCRETE_TYPE</td>
<td>Copy</td>
<td></td>
</tr>
</tbody>
</table>

**INPUT**
- From_The_Set : Set
- To_The_Set : Set

**OUTPUT**
- To_The_Set : Set

**EXCEPTIONS**
- Item_Is_In_Set, Item_Is_Not_In_Set

---

**Profile Filtering Minimum Rank:** 1.00

Found 34 components in 26 partitions.
Their are 34 possible candidates.

<table>
<thead>
<tr>
<th>Component ID</th>
<th>Profile Rank</th>
<th>Signature Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>3001</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4002</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4003</td>
<td>0.77</td>
<td>1.00</td>
</tr>
<tr>
<td>4004</td>
<td>0.77</td>
<td>1.00</td>
</tr>
<tr>
<td>5006</td>
<td>0.38</td>
<td>1.00</td>
</tr>
<tr>
<td>5007</td>
<td>0.38</td>
<td>1.00</td>
</tr>
<tr>
<td>5011</td>
<td>0.38</td>
<td>1.00</td>
</tr>
<tr>
<td>5013</td>
<td>0.38</td>
<td>1.00</td>
</tr>
<tr>
<td>1002</td>
<td>0.31</td>
<td>1.00</td>
</tr>
<tr>
<td>4001</td>
<td>0.31</td>
<td>1.00</td>
</tr>
<tr>
<td>5008</td>
<td>0.31</td>
<td>1.00</td>
</tr>
<tr>
<td>5009</td>
<td>0.31</td>
<td>1.00</td>
</tr>
<tr>
<td>5010</td>
<td>0.31</td>
<td>1.00</td>
</tr>
</tbody>
</table>

**Signature Matching Minimum Rank:** 1.00

2 candidates have profile rank >= 1.00

/home/greg/Thesis/SoftwareBase/ran/PSDL_SPEC

Component ID: 3001
Profile Rank: 1.00
Number of signature match solutions: 1

Solution 1

- Signature Rank: 1.00
- Semantic Rank: unknown
- Valid, Not Expanded
- Op Map: [Extent_OE]

---

**Figure 10 Search Dialog**
The query file name input box is placed above the query display. A file name can be entered; when the return key is pressed the file is opened and displayed in the query window. The query menu, Figure 11, gives the user the means to enter a new query, open an existing query, edit the current one, or save the current query. The search can be started by pressing the search button or through the search menu. Figure 12 illustrates the search menu. Usage information is provided through a help menu (Figure 13).

<table>
<thead>
<tr>
<th>New</th>
<th>Ctrl+N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Ctrl+O</td>
</tr>
<tr>
<td>Edit</td>
<td>Ctrl+E</td>
</tr>
<tr>
<td>Save</td>
<td>Ctrl+S</td>
</tr>
<tr>
<td>Save As</td>
<td>Ctrl+A</td>
</tr>
<tr>
<td>Close</td>
<td>Ctrl+W</td>
</tr>
<tr>
<td>Edit</td>
<td>Ctrl+Q</td>
</tr>
</tbody>
</table>

Figure 11: Query Menu

The minimum profile ranking that must be exceeded to pass filtering is displayed above the profile filtering results. This window is where the user will input the desired value. The user may adjust the value up or down to tailor the filtering process to increase or decrease the number of components that pass profile filtering.

Figure 12: Search Menu

Figure 13: Help Menu
<table>
<thead>
<tr>
<th>Component ID</th>
<th>Rank</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>3001</td>
<td>1.00</td>
<td>{}</td>
</tr>
<tr>
<td>4002</td>
<td>1.00</td>
<td>{}</td>
</tr>
<tr>
<td>4003</td>
<td>0.77</td>
<td>{}</td>
</tr>
<tr>
<td>4004</td>
<td>0.77</td>
<td>{}</td>
</tr>
<tr>
<td>5006</td>
<td>0.38</td>
<td>{}</td>
</tr>
<tr>
<td>5007</td>
<td>0.38</td>
<td>{}</td>
</tr>
<tr>
<td>5011</td>
<td>0.38</td>
<td>{}</td>
</tr>
<tr>
<td>5013</td>
<td>0.38</td>
<td>{}</td>
</tr>
<tr>
<td>1002</td>
<td>0.31</td>
<td>{}</td>
</tr>
<tr>
<td>4001</td>
<td>0.31</td>
<td>{}</td>
</tr>
<tr>
<td>5008</td>
<td>0.31</td>
<td>{}</td>
</tr>
<tr>
<td>5009</td>
<td>0.31</td>
<td>{}</td>
</tr>
<tr>
<td>5010</td>
<td>0.31</td>
<td>{}</td>
</tr>
<tr>
<td>5012</td>
<td>0.31</td>
<td>{}</td>
</tr>
<tr>
<td>5014</td>
<td>0.31</td>
<td>{}</td>
</tr>
<tr>
<td>5015</td>
<td>0.31</td>
<td>{}</td>
</tr>
<tr>
<td>1003</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>1004</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5016</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5017</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5018</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5019</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5020</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5021</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5022</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5023</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5024</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5025</td>
<td>0.23</td>
<td>{}</td>
</tr>
<tr>
<td>5001</td>
<td>0.15</td>
<td>{}</td>
</tr>
<tr>
<td>1001</td>
<td>0.08</td>
<td>{}</td>
</tr>
<tr>
<td>5002</td>
<td>0.08</td>
<td>{}</td>
</tr>
<tr>
<td>5003</td>
<td>0.08</td>
<td>{}</td>
</tr>
<tr>
<td>5004</td>
<td>0.08</td>
<td>{}</td>
</tr>
<tr>
<td>5005</td>
<td>0.08</td>
<td>{}</td>
</tr>
</tbody>
</table>

Figure 14: Profile Filtering Results Example

Like the profile display, the signature match has both a rank and results window. The results window is also where the user will select the component that best matches their needs. Figure 15 illustrates the results of a query. Filtering found two candidates with a rank greater than or equal to that set by the user, one in this case. The component name is tan with component ID 3001. To select this component the user will press the button next to the name. To view the other candidates the user can scroll down. If neither of the candidates meet the users needs they can reduce the rank and search again. The operator map is shown in Figure 16.
Figure 15: Search Results Example

As the multilevel filtering technology evolves and levels are added, the modular design of the search dialog will allow it to encompass them by adding new windows for input and output.
G. PROGRESS DISPLAY

While the filtering process is running a progress display keep the user informed as to what is happening. Figure 17 is an example of the search progress.
H. MAINTENANCE DISPLAY

Maintenance is the addition of components to the software base or the initialization of the multilevel filtering representation of the components in the software base. The maintenance dialog aids the librarian in keeping the software base current. Figure 18 shows the maintenance dialog.
The librarian enters a new PSDL spec with the SD Editor or opens one that was previously created. Figure 19 show the spec menu. New opens the SD Editor for input of a new PSDL specification. Open starts a file dialog to open an existing PSDL specification. Edit opens the current PSDL specification in the SD Editor. After the user has finished with the PSDL spec, it will be displayed in the top window. An alternate way of opening the PSDL file is to enter its name in the PSDL file window; a return is needed update the display.
The directory that the component will be stored in will be named after the PSDL specification file name. This will be created during the initialization process. The component files include any file that is needed to implement the component or document its function. These files will be copied to the component directory during initialization.

Figure 20 is the component menu. Included in the menu are the following items: Add component file adds a file to the component list. Clear component files, removes all files from the component list. Clear all files clears both the PSDL spec and component list.

There are two steps to the initialization process. First the component directory is created and the components files are copied into it. Then the header file is created using all of the directories found in the software base. The header file is used as input to the initialization of the data structures. Figure 21 is the initialization menu. Initialize directory sets up the component directory but does not initialize the data structures. Initialize software base will initialize the component directory, if needed, and the data structures. User information is provided through the help menu (Figure 22).
The following steps illustrates adding a component to the software base:

1. Setup the PSDL Spec.
2. Add component implementation files.
3. Initialize the component directory.
4. Initialize the data structures.

Note that steps one though three can be repeated for multiple components to set up the directories before initializing the data structures.

While initialization is running a progress display keeps the user informed as to what is happening. Figure 23 is an example of the initialization progress display.
Doing software base initialization

Initializing software base
Preparing 1000 /home/greg/Thesis/SoftwareBase/pstacksbm ... 4 comp
Preparing 2000 /home/greg/Thesis/SoftwareBase/pstorages ... 0 comp
Preparing 3000 /home/greg/Thesis/SoftwareBase/tan ... 1 components
Preparing 4000 /home/greg/Thesis/SoftwareBase/vbagsebmi ... 4 comp
Preparing 5000 /home/greg/Thesis/SoftwareBase/vmapensbm ... 25 comp
inserting 1001
inserting 1002
inserting 1003
inserting 1004
inserting 3001
inserting 4001
inserting 4002
inserting 4003
inserting 4004
inserting 5001
inserting 5002
inserting 5003

Figure 23: Maintenance Progress Display
IV. CONCLUSIONS AND FUTURE RESEARCH

A. ACCOMPLISHMENTS

The goal of software reuse will only be attained if the tools that insulate the user from the tedious job of finding suitable components are created. The tools must be easy to use, accurate, fast, and display results in an ordered manner. Tools are also needed for the addition of components to the repository. If the task of adding a component is too complex developers will not contribute to the repository. The Graphical User Interface, to Multilevel Filtering, developed in this thesis will aid the user in finding the reusable components that they are looking for and making additional components available for others to use.

The Following Tools, running on the Linux operating system, have been used to implement the Graphical User Interface.

- Linux version 2.0.27
- Gnu C++ compiler version
- GNAT ADA 95 version
- GNU Source-Level Debugger
- Data Display Debugger
- XFree86 X11R6
- MetroLinks Motif version 2.0
All of the development tools are available on the Internet for a variety of development platforms, with the exception of MetroLinks Motif. During the development of the interface, a suitable Motif replacement was unavailable.

B. FUTURE RESEARCH

As multilevel filtering evolves and grows, the additions should be incorporated into the underling search program. The method of passing data between the C++ and ADA code needs to be refined.
LIST OF REFERENCES


33

APPENDIX A C++ GUI SOURCE CODE

Source code for the C++ for the Graphical User Interface.

A. CALLBACKS.H

/*
 * $Id: Callbacks.h,v 1.3 1998/01/16 00:17:34 greg Exp $
 * Callbacks.h -- Software Base Search Interface
 * Header file for the common callbacks.
 * Naval Postgraduate School
 * January 13, 1998
 * Written by Gregory L. Meckstroth
 */

/*
 * FILEDLG_DATA structure is used to pass data to the file callback.
 * parent: widget to use a parent for dialog construction.
 * fname_text: text widget for file name display.
 * text: text widget for query/spec display.
 */

struct FILEDLG_DATA
{
    Widget parent;
    Widget fname_text;
    Widget text;
};

void genericCB(Widget widget, XtPointer client_data, XtPointer call_data);

void newCB(Widget widget, XtPointer client_data, XtPointer call_data);

void editCB(Widget widget, XtPointer client_data, XtPointer call_data);

void openCB(Widget widget, XtPointer client_data, XtPointer call_data);

void saveCB(Widget widget, XtPointer client_data, XtPointer call_data);

void saveasCB(Widget widget, XtPointer client_data, XtPointer call_data);

void textchangedCB(Widget widget, XtPointer client_data,
B. CALLBACKS.C

/*
* Callbacks.C -- Software Base Search Interface
* Source code that implements the functionality of the file menu, in the search
* and maintenance dialogs, through the use of callback routines.
* Entry points: newCB, editCB, openCB, saveCB, saveasC, textchangedCB
* Naval Postgraduate School
* January 11, 1998
* Written by Gregory L. Meckstroth
*/
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <Xm/Xm.h>
#include <Xm/Text.h>
#include <Xm/FileSB.h>
#include "Gui.h"
#include "Utils.h"
#include "SDE.h"
#include "PromptDialog.h"
#include "Callbacks.h"

/*
* Declarations for local functions.
*/
static void
fileokCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void
filecancelCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void
saveasokCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void
editlefile(FILEDLG_DATA * data, char *filename, char **string);

/*
* Global storage for the user selected directory.
*/
static char *save_directory = NULL;

/*
* Generic callback assumes client_data has a text string. This callback is for
* testing only
*/
void
genericCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    if (client_data != NULL)
    {
        printf("Generic callback [%s]\n", (char*)client_data);
    }
    
/*

36
void openCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    FILEDLG_DATA *data = (FILEDLG_DATA *) client_data;
    static FILEDLG_DATA ok_data;
    Widget fsdialog;
    Arg args[20];
    Cardinal n;
    XmString title = XmStringCreateSimple("Open Query Specification");
    XmString dirmask = XmStringCreateSimple("*.psdl");

    n = 0;
    SETARG(args[n], XmNdirMask, dirmask, n);
    SETARG(args[n], XmNdialogTitle, title, n);
    fsdialog = XmCreateFileSelectionDialog(data->parent, "sbsdialog", args, n);
    if (save_directory != NULL)
        Set_Res_String(fsdialog, XmNdirectory, save_directory);
    XmStringFree(title);
    XmStringFree(dirmask);

    ok_data.parent = fsdialog;
    ok_data.fname_text = data->fname_text;
    ok_data.text = data->text;

    XtAddCallback(fsdialog, XmNokCallback,
                  (XtCallbackProc) fileokCB,
                  (XtPointer) & ok_data);
    XtAddCallback(fsdialog, XmNcancelCallback,
                  (XtCallbackProc) filecancelCB,
                  (XtPointer) fsdialog);
    XtManageChild(fsdialog);
}

void saveasCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    FILEDLG_DATA *data = (FILEDLG_DATA *) client_data;
    static FILEDLG_DATA ok_data;
    Widget fsdialog;
    Arg args[20];
    Cardinal n;
XmString title = XmStringCreateSimple("Save Query Specification");
XmString dirmask = XmStringCreateSimple("*.psdl");

/*
 * Create the file selection dialog using a directory mask so that the user
 * only sees the PSDL specification files.
 */

n = 0;
SETARG(args[n], XmNdirMask, dirmask, n);
SETARG(args[n], XmNdialogTitle, title, n);

fsdialog = XmCreateFileSelectionDialog(data->parent, "Save Query Specification as:",
args, n);

XmStringFree(title);
XmStringFree(dirmask);

/*
 * Change the parent widget to the file selection dialog so we can close it
 * and not its parent. The rest of the data is pasted through.
 */

ok_data.parent = fsdialog;
ok_data.fname_text = data->fname_text;
ok_data.text = data->text;

XtAddCallback(fsdialog, XmNokCallback,
(XtCallbackProc) saveasokCB,
(XtPointer) ok_data);

XtAddCallback(fsdialog, XmNcancelCallback,
(XtCallbackProc) filecancelCB,
(XtPointer) fsdialog);

XtManageChild(fsdialog);
}

/*
 * Save file callback. The data for this callback is passed through the client
 * data as a pointer to a FILEDLG_DATA structure. The query/spec text will be
 * saved into the currently open file. If a file has not been opened the save
 * as callback is used to solicit a file name from the user.
 */

void
saveCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
 FILEDLG_DATA *data = (FILEDLG_DATA *) client_data;
 char *string = XmTextGetString(data->text);
 char *filename = gettext(data->fname_text);

 if (filename == NULL)
 {
  saveasCB(widget, client_data, call_data);
 }
 else
 {
  if (string != NULL)
   savetext(widget, filename, string);
  XtFree(string);
  }

  XtFree(filename);
}

/*
 * The user changed the filename in the text window. This is activated when the
 * user types a carriage return and the text window has the focus. The data for
 * this callback is passed through the client data as a pointer to a
 * FILEDLG_DATA structure. The specified file will be opened and loaded into
 * the query/spec text widget.
 */

38
void
textchangedCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    FILEDLG_DATA *data = (FILEDLG_DATA *) client_data;
    char *string;
    char *filename = gettext(data->fname_text);
    /*
     * If user entered a file name try to display it in the text window. If the
     * user cleared the file name clear the text window.
     */
    if (filename != NULL)
    {
        if (LoadFile(filename, &string) != -1 && string != NULL)
        {
            XmTextSetString(data->text, string);
            XtFree(string);
        }
        XtFree(filename);
    } else
    {
        XmTextSetString(data->text, "");
    }
}

/*
 * Create a new query spec callback. The data for this callback is passed
 * through the client data as a pointer to a FILEDLG_DATA structure. Open the
 * Syntax Directed Editor for the user to input a new query/spec. After the
 * editor is closed the file is saved in the file query.psdl. The query/spec
 * text display is updated.
 */
void
newCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    FILEDLG_DATA *data = (FILEDLG_DATA *) client_data;
    /*
     * Pass null for the file name and text string so that the editor will
     * start with an empty specification.
     */
    char *string = NULL;
    char *filename = NULL;
    editfile(data, &filename, &string);
}

/*
 * Edit existing query spec callback. The data for this callback is passed
 * through the client data as a pointer to a FILEDLG_DATA structure. Open the
 * Syntax Directed Editor for the user to input a new query/spec. After the
 * editor is closed the file is saved in the specified file or query.psdl if
 * one is not specified. The query/spec text display is updated.
 */
void
editCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    FILEDLG_DATA *data = (FILEDLG_DATA *) client_data;
    /*
     * get the file name and text string so that they will be loaded into the
     * editor when it starts.
     */
}
char *string = XmTextGetString(data->text);
char *filename = XmTextGetString(data->fname_text);
editfile(data, filename, string);
}
/*
* OK callback for the file selection dialog. The data for this callback is
* passed through the client data as a pointer to a FILEDLG_DATA structure and
* the call_data as pointer to a XmFileSelectionBoxCallbackStruct. The file
* name is take from the call data and the file name text is updated and the
* query/spec text display is loaded with the files contents.
*/
static void
fileokCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
  FILEDLG_DATA *data = (FILEDLG_DATA *) client_data;
  XmFileSelectionBoxCallbackStruct *ptr;
  char *string;
  char *filename;
  /* Convert the file name to a character string. */
  ptr = (XmFileSelectionBoxCallbackStruct *) call_data;
  XmStringGetLtoR(ptr->value, XmSTRING_DEFAULT_CHARSET, filename);
  /* If the user did not select a file we are done */
  if (filename != NULL)
  {
    /* Save the directory for the next time the user wants to open a file.
     * This will make the task of entering multiple files from the same
     * directory easier. */
    if (save_directory != NULL)
      XtFree(save_directory);
    save_directory = getdirname(filename);
    /* Display the file name in its text widget. This is also the place we
     * save the file name for later use. */
    XmTextSetString(data->fname_text, filename);
    /* Display the specification in its text widget. If their was an error
     * loading the file tell the user. */
    if (LoadFile(filename, &string) == -1)
      ModalWarningDialog(data->parent, "Error", "Error loading file");
  if (string != NULL)
  {
    XmTextSetString(data->text, string);
    XtFree(string);
  }
  }
  /* Take the dialog box off of the screen and free up its data */
}
XtUnmanageChild(data->parent);
XtDestroyWidget(data->parent);
}

/*
 * Cancel callback for the file selection dialog.
 */
static void
filecancelCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
  XtUnmanageChild((Widget) client_data);
  XtDestroyWidget((Widget) client_data);
}

/*
 * OK callback for the save as file selection dialog. The data for this
 * callback is passed through the client data as a pointer to a FILEDLG_DATA
 * structure and the call data as pointer to a XmFileSelectionBoxCallbackStruct.
 * The file name is take from the call data and the query/spec text is save to
 * the file.
 */
static void
saveasokCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
  FILEDLG_DATA *data = (FILEDLG_DATA *) client_data;
  XmFileSelectionBoxCallbackStruct *ptr;
  char *string = XmTextGetString(data->text);
  char *filename;

  /*
   * Convert the file name to a character string.
   */
  ptr = (XmFileSelectionBoxCallbackStruct *) call_data;
  XmStringGetLtoR(ptr->value, XmSTRING_DEFAULT_CHARSET, &filename);

  /*
   * Take the dialog box off of the screen and free up its data
   */
  XtUnmanageChild(data->parent);
  XtDestroyWidget(data->parent);

  /*
   * If the user did not select a file we are done
   */
  if (filename == NULL)
    return;

  /*
   * Save the directory for the next time the user wants to open a file. This
   * will make the task of entering multiple files from the same directory
   * easier.
   */
  if (save_directory != NULL)
    XtFree(save_directory);
  save_directory = getdirname(filename);

  /*
   * Display the file name in its text widget. This is also the place we save
   * the file name for later use.
   */
  XmTextSetString(data->fname_text, filename);

  /*
   * If the specification text is not empty save it into the file.
   */
if (string != NULL) {
  savetext(data->parent, filename, string);
  XtFree(string);
}

/*
* Edit new or existing query spec. This will be changed to call the syntax
* directed editor (SDE) from CAPS.
*/
static void editfile(FILEDLG_DATA * data, char **filename, char **string) {
  /*
   * Open the editor with the file name and specification so the use can make
   * changes.
   */
  SDE(data->parent, filename, string);
  /*
   * If the user did not save the specification we are done
   */
  if (*string == NULL) {
    if (*filename != NULL)
      XtFree(*filename);
    return;
  }
  /*
   * If the user did not enter a file name use the default
   */
  if (*filename == NULL) {
    *filename = "query.psd1";
    savetext(data->parent, *filename, *string);
    XmTextSetString(data->fname_text, *filename);
  } else {
    savetext(data->parent, *filename, *string);
    XmTextSetString(data->fname_text, *filename);
    XtFree(*filename);
  }
  /*
   * Display the specification in its text widget.
   */
  XmTextSetString(data->text, *string);
  XtFree(*string);
}

C. DISPLAYPROGRESS.H

/*
* $Id: DisplayProgress.h,v 1.4 1998/01/18 17:40:48 greg Exp $
* DisplayProgress.h -- Software Base Search Interface
* Header file for the progress display.
* Naval Postgraduate School
* January 13, 1998
* Written by Gregory L. Meckstroth
*/
void
initialize_display(Widget parent);

void
clear_display(char *title);

void
home_display(void);

void
display_message(char *message);

D. DISPLAYPROGRESS.C

/*
 * DisplayProgress.C -- Software Base Search Interface
 * Source code for the progress display.
 * Entry points: initialize_display clear_display display_message
 * Naval Postgraduate School
 * January 11, 1998
 * Written by Gregory L. Meckstroth
 */

#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/ScrolledW.h>
#include <Xm/PushB.h>
#include "Gui.h"
#include "Utils.h"
#include "DisplayProgress.h"

/*
 * Declarations for local functions.
 */

static void
dismissCB(Widget widget, XtPointer client_data, XtPointer call_data);

/*
 * The following widgets are global, to this routine, to minimize the effort in
 * interfacing the display routines to the ADA search code.
 */

static Widget display_window = NULL;
static Widget text_display = NULL;

/*
 * Initialize the display widgets
 */

void
initialize_display(Widget parent)
{
    Widget dismiss_btn;
    XmString xmstring;

if (text_display != NULL)
return;

/*
 * Setup the display dialog
 */
n = 0;
SETARG(args[n], XmNautoUnmanage, False, n);
SETARG(args[n], XmNwidth, DIALOG_WIDTH, n);
SETARG(args[n], XmNheight, 1.5 * INFO_WINDOW_HEIGHT, n);
SETARG(args[n], XmNnoResize, False, n);

display_window = XmCreateFormDialog(parent, "sbsdialog", args, n);

/*
 * Add button to the bottom of the dialog box
 */

xmstring = XmStringCreateSimple("Dismiss");
dismiss_btn = XtVaCreateManagedWidget("sbslabel",
 XmPushButtonWidgetClass, display_window,
 XmNlabelString, xmstring,
 XmNwidth, BUTTON_WIDTH,
 XmNheight, BUTTON_HEIGHT,
 XmNleftAttachment, XmATTACH_FORM,
 XmNleftOffset, 20,
 XmNrightAttachment, XmATTACH_NONE,
 XmNtopAttachment, XmATTACH_NONE,
 XmNbottomAttachment, XmATTACH_FORM,
 XmNbottomOffset, 10,
 NULL);
XmStringFree(xmstring);

XtAddCallback(dismiss_btn,
 XmNactivateCallback,
 (XtCallbackProc) dismissCB,
 (XtPointer) display_window);

XtManageChild(dismiss_btn);

/*
 * Add the scrolled text window. This is attached to the top of the dialog
 * box and the top of the dismiss button.
 */

n = 0;
SETARG(args[n], XmNeditMode, XmMULTI_LINE_EDIT, n);
SETARG(args[n], XmNeditable, False, n);
SETARG(args[n], XmNtraversalOn, False, n);
SETARG(args[n], XmNcursorPositionVisible, False, n);
SETARG(args[n], XmNleftAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNleftOffset, 5, n);
SETARG(args[n], XmNrightAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNrightOffset, 5, n);
SETARG(args[n], XmNtopAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNtopOffset, 5, n);
SETARG(args[n], XmNbottomAttachment, XmATTACH_WIDGET, n);
SETARG(args[n], XmNbottomOffset, 5, n);
SETARG(args[n], XmNresizable, True, n);

text_display = XmCreateScrolledText(display_window, "sbstext", args, n);

XtManageChild(text_display);
}

/*
 * Clear the display and change the dialog title.
 */
void
clear_display(char *title) {
    if (display_window != NULL && title != NULL) {
        Set_Res_String(display_window, XmNdialogTitle, title);
    }
    if (text_display != NULL) 
        XmTextSetString(text_display, "");
    if (IXtlsManaged(display_window)) 
        XtManageChild(display_window);
}

/*
 * Scroll display to so first line is at the top of window
 */
void home_display(void) {
    if (text_display != NULL)
        XmTextScroll(text_display, -1000);
}

/*
 * Display progress messages and help information.
 */
void display_message(char *message) {
    XtAppContext app_context;
    XtInputMask mask;
    XmTextInsert(text_display, 10000, message);
    app_context = XtWidgetToApplicationContext(text_display);
    while ((mask = XtAppPending(app_context)))
        XtAppProcessEvent(app_context, mask);
}

/*
 * Callback to take the progress display off of the screen.
 */
static void dismissCB(Widget widget, 
    XtPointer client_data, 
    XtPointer call_data) {
    if (XtIsManaged((Widget) client_data)) 
        XtUnmanageChild((Widget) client_data);
}

E. GUI.H

/* $Id: Gui.h,v 1.3 1998/01/16 17:29:38 greg Exp $ 
 * Gui.h -- Software Base Search Interface 
 * Header file of the common GUI definitions. 
 * Naval Postgraduate School 
 * January 13, 1998 
 * Written by Gregory L. Meckstroth 
 */
Define the location of the software base. It is assumed that the location
will be static and that the CAPS system will be rebuilt for different
systems.
*/
#define SBROOT "/home/greg/Thesis/SoftwareBase/"
/*
 * Define the size of the common widgets and buffers.
 */
#define MAXBUFSIZE 1024
#define DIALOG_WIDTH 600
#define DIALOG_HEIGHT 550
#define INFO_WINDOW_WIDTH (DIALOG_WIDTH-100)
#define INFO_WINDOW_HEIGHT 250
#define BUTTON_WIDTH 77
#define BUTTON_HEIGHT 33

/*
 * Define a macro to set the Motif arguments.
 * Note: indent wants to move the increment of n to a separate line defeating
 * the standard Motif style.
 */
#define SETARG(arg,name,value,n) XtSetArg(arg,name,value);n++;

F. GULC

/*
 *
 * Menu.h -- Software Base Search Interface
 *
 * Header file for the GUI menus.
 *
 * Naval Postgraduate School
 * January 13, 1998
 * Written by Gregory L. Meckstroth
 *
 */

/*
 * The first string is used as the help dialog title
 */
static const char *maintenance_help_message[] =
{
  "Maintenance Help",
  "CAPS SOFTWARE BASE SEARCH \n",
  "\n",
  "NAVAL POSTGRADUATE SCHOOL \n",
  "MONTEREY, CALIFORNIA \n",
  "January 11, 1998 \n",
  "\n",
  "Initialize components in the CAPS software base. This dialog box \n",
  "displays the PSDL specification and component file list. The input \n",
  "consists of the component specification in PSDL and the component \n",
  "file list. The spec file name can be changed by typing the new name \n",
  "in the PSDL Spec window. After entering the file name press the enter \n",
  "key to update the spec display. \n",
  "\n",
  "Each component in the Software Base is stored in a separate directory. \n",
  "After entering all files for a component the user can initialize the \n",
  "component directory structure by pressing the 'Init DIR' button or \n",
  "through the Init menu. After the component directory is initialized \n",
  "the user can enter another component. This can be repeated until all \n",
  "components have been entered. Then the Software Base must be \n",
  "initialized by pressing the 'Init SB' button or through the Init menu. \n",
  "\n",
};
"All component files are copied to the component directory no user 
files will be deleted. \n",
"The Spec menu allows the user to manage the PSDL query. \n",
"  New: Starts the Syntax Directed Editor to input a new query. \n",
"  Open: Open an existing query file. \n",
"  Edit: Starts the Syntax Directed Editor to edit the current query. \n",
"  Save: Save the current query. \n",
"  SaveAs: Save the current query in a user specified file. \n",
"  Close: Close the search dialog. \n",
"  Exit: Exit the program. \n",
"The component menu allows the user to manage the component file list. \n",
"  Add Component files: \n",
"  Clear Component file list: \n",
"  Clear all: \n",
"Buttons. \n",
"Dismiss: Close the maintenance dialog. \n",
"Init DIR: Initialize the component directory. \n",
"Init SB: Run the ADA initialization for the Software Base. \n",
"Clear FL: Clear the file list. \n",
"Clear ALL: Clear file list and Component specification. \n",
(char *)NULL
};

/*
 * The first string is used as the help dialog title
 */

static const char *search_help_message[] =
{
    "Search Help",
    "CAPS SOFTWARE BASE SEARCH \n",
    "NAVAL POSTGRADUATE SCHOOL \n",
    "MONTEREY, CALIFORNIA \n",
    "January 11, 1998 \n",
    "Search and retrieval of components from the CAPS software base. This \n", 
    "dialog box displays the PSDL query, profile filtering and signature \n", 
    "matching results. The input consists of the query specification, \n", 
    "minimum profile rank and minimum signature rank. The user can change \n", 
    "the minimum rank by entering a new value in the appropriate text \n", 
    "window. The query file name can be changed typing the new name in \n", 
    "the PSDL Query window. After entering the file name press the enter \n", 
    "key to update query display. The query must be stored in a file for \n", 
    "the search routines to work. If the user does not supply a file \n", 
    "name the default query.psdl will be used. \n",
    "After running the search the user can view the results and select \n", 
    "the appropriate component by pushing the toggle button next to the \n", 
    "component name. Pushing the OK button will return the selected \n", 
    "component. \n",
    "The Query menu allows the user to manage the PSDL query. \n",
    "  New: Starts the Syntax Directed Editor to input a new query. \n",
    "  Open: Open an existing query file. \n",
    "  Edit: Starts the Syntax Directed Editor to edit the current \n", 
    "  Save: Save the current query. \n"}
The Search menu allows the user to run the software base search.

- **Start**: Start the software base search.
- **Close**: Close the search dialog.
- **Exit**: Exit the program.
- **SaveAs**: Save the current query in a user specified file.
- **Cancel**: Exit the dialog.
- **OK**: Exits the dialog and returns the selected component name.
- **Search**: Start the software base search.

Buttons:

- **OK**: Exits the dialog and returns the selected component name.
- **Search**: Start the software base search.
- **Cancel**: Exit the dialog.

G. HELPMESSAGES.H

/*
 * The first string is used as the help dialog title
 */

static const char *version_message[] =
{
    "Version",
    "CAPS SOFTWARE BASE SEARCH \n",
    "NAVAL POSTGRADUATE SCHOOL \n",
    "MONTEREY, CALIFORNIA \n",
    "January 11, 1998 \n",
    "$Revision: 1.3 $\n",
    (char *)NULL
};

G. HELPMESSAGES.H

/*
 * The first string is used as the help dialog title
 */

static const char *maintenance_help_message[] =
{
    "Maintenance Help",
    "CAPS SOFTWARE BASE SEARCH \n",
    "NAVAL POSTGRADUATE SCHOOL \n",
    "MONTEREY, CALIFORNIA \n",
    "January 11, 1998 \n",
    "\n",
    "Initialize components in the CAPS software base. This dialog box \n",
    "displays the PSDL specification and component file list. The input \n",
    "consists of the component specification in PSDL and the component \n",
    "file list. The spec file name can be changed by typing the new name \n",
    "in the PSDL Spec window. After entering the file name press the enter \n",
    "key to update the spec display. \n",
    "\n",
    "Each component in the Software Base is stored in a separate directory. \n",
    "After entering all files for a component the user can initialize the \n",
    "component directory structure by pressing the 'Init DIR' button or \n",
    "through the Init menu. After the component directory is initialized \n",
    "the user can enter another component. This can be repeated until all \n",
    "$Id: HelpMessages.h,v 1.3 1998/01/25 22:49:08 greg Exp $\n", (char *)NULL
};
components have been entered. Then the Software Base must be initialized by pressing the 'Init SB' button or through the Init menu.

All component files are copied to the component directory no user files will be deleted.

The Spec menu allows the user to manage the PSDL query.

New: Starts the Syntax Directed Editor to input a new query.

Open: Open an existing query file.

Edit: Starts the Syntax Directed Editor to edit the current query.

Save: Save the current query.

SaveAs: Save the current query in a user specified file.

Close: Close the search dialog.

Exit: Exit the program.

The component menu allows the user to manage the component file list.

Add Component files:

Clear Component file list:

Clear all:

Buttons.

Dismiss: Close the maintenance dialog.

Init DIR: Initialize the component directory.

Init SB: Run the ADA initialization for the Software Base.

Clear FL: Clear the file list.

Clear ALL: Clear file list and Component specification.

char *)NOLL

The first string is used as the help dialog title

static const char *search_help_message[] =
{
"Search Help",
"CAPS SOFTWARE BASE SEARCH",
"NAVAL POSTGRADUATE SCHOOL",
"MONTEREY, CALIFORNIA",
"January 11, 1998",
"Search and retrieval of components from the CAPS software base. This dialog box displays the PSDL query, profile filtering and signature matching results. The input consists of the query specification, minimum profile rank and minimum signature rank. The user can change the minimum rank by entering a new value in the appropriate text window. The query file name can be changed typing the new name in the PSDL Query window. After entering the file name press the enter key to update query display. The query must be stored in a file for the search routines to work. If the user does not supply a file name the default query.psdl will be used. After running the search the user can view the results and select the appropriate component by pushing the toggle button next to the component name. Pushing the OK button will return the selected component."
}
"Edit: Starts the Syntax Directed Editor to edit the current \n",  
"query. \n",  
"Save: Save the current query. \n",  
"SaveAs: Save the current query in a user specified file. \n",  
"Close: Close the search dialog. \n",  
"Exit: Exit the program. \n",  
"\n",  
"The Search menu allows the user to run the software base search. \n",  
"Start: Start the software base search. \n",  
"\n",  
"Buttons. \n",  
"OK: Exits the dialog and returns the selected component name. \n",  
"Search: Start the software base search. \n",  
"Cancel: Exit the dialog. \n",  
(char *)NULL
};

/* The first string is used as the help dialog title */
static const char *version_message[] =
{
  "Version",
  "\n",
  "CAPS SOFTWARE BASE SEARCH \n",
  "\n",
  "NAVAL POSTGRADUATE SCHOOL \n",
  "MONTEREY, CALIFORNIA \n",
  "January 11, 1998 \n",
  "$Revision: 1.3 $\n",
  (char *)NULL
};

H. MAINTENANCEDIALOG.H

/* $Id: MaintenanceDialog.h,v 1.3 1998/01/25 22:49:08 greg Exp $ */
/* MaintenanceDialog.h -- Software Base Search Interface*/
/* Header file for the initialization dialog. */
/* Naval Postgraduate School */
/* January 13, 1998 */
/* Written by Gregory L. Meckstroth */
*/

void MaintenanceDialog(Widget parent);
void ModalMaintenanceDialog(Widget parent);

I. MAINTENANCEDIALOG.C

/* MaintenanceDialog.C -- Software Base Search Interface*/
/* Source code for the CAPS software base maintenance dialog. This dialog setups*/
/* the files needed by the ADA routines and runs the initialization. It is */
/* assumed that the components are in separate directories and that these are */
/* the only directories in the software base. All of the directory names in the */
/* software base are written to the sb_header.dat file. This file is then used */
/* as input to the initialization. */
/* Entry points: MaintenanceDialog, ModalMaintenanceDialog. */
/* Naval Postgraduate School  
* January 13, 1998  
* Written by Gregory L. Meckstroth  
*/

#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>

#include <Xm/Xm.h>
#include <Xm/Text.h>
#include <Xm/Label.h>
#include <Xm/ScrolledW.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <Xm/Separator.h>
#include <Xm/PushB.h>
#include <Xm/FileSB.h>

#include "Gui.h"
#include "Menu.h"
#include "SearchDialog.h"
#include "SDE.h"
#include "Callbacks.h"
#include "DisplayProgress.h"
#include "PromptDialog.h"
#include "MaintenanceDialog.h"

/* The INIT_DATA structure is used to pass data to the callback routines.
* parent: widget to use as the parent for dialog construction.
* flist_text: text widget for component file list.
* fname_text: text widget for file name display.
* spec_text: text widget for specification display.
*/

struct INIT_DATA
{
    Widget parent;
    Widget flist_text;
    Widget fname_text;
    Widget spec_text;
};

/* Declaration for local functions. */

static void clear(INIT_DATA * data);
static int initdirectory(char *sfname, INIT_DATA * data);
static void closeCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void exitCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void initdirCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void initsbCB(Widget widget, XtPointer client_data, XtPointer call_data);

51
static void
addokCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void
addcancelCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void
addCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void
clearflCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void
clearallCB(Widget widget, XtPointer client_data, XtPointer call_data);

/*
* Declaration for the "ADA export" initialization routine.
*/
extern "C" sb_init(char *sb_root_directory, char *sb_header_file_name);

/*
* Global storage for the user selected spec file and component directory.
*/
static char *save_spec_filename = NULL;
static char *save_component_dir = NULL;

/*
* Used by the modal search dialog. done_with_dialog while true the event loop
* will continue running. Setting this to false allows the event loop to exit.
*/
static int done_with_dialog;

/* Display the software base maintenance modal dialog. */

void
ModalMaintenanceDialog(Widget parent)
{
    MaintenanceDialog(parent);

    /*
    * Wait for a response to the dialog. When the Dismiss or cancel button is
    * pressed done_with_dialog will be set true causing the loop to terminate.
    */
    done_with_dialog = False;
    while (!done_with_dialog)
        XtAppProcessEvent(XtWidgetToApplicationContext(parent), XtIMAll);
}

/*
* Display the software base maintenance modal dialog.
*/

void
MaintenanceDialog(Widget parent)
{
    Widget spec_fname_text;
    Widget dialog_window;
    Widget menubar;
    Widget specmenu;
    Widget componentmenu;
    Widget initmenu;
    Widget helpmenu;
    Widget previous;
    Widget spec_label;
    Widget scrolled_spec_window;
    Widget spec_text;
    Widget initSB_btn;
    Widget initdir_btn;
Widget clearfl_btn;
Widget clearall_btn;
Widget dismiss_btn;
Widget caps_label;
Widget filelist_label;
Widget scrolled_filelist_window;
Widget filelist_text;

Arg args[20];
Cardinal n;
char *title;
XmString xmstring;

/*
 * The following static variables are used by callbacks after
 * MaintenanceDialog returns.
 */
static FILEDLG_DATA filedlg_data;
static INIT_DATA init_data;

/*
 * Create the dialog box.
 */
title = "Software Base Maintenance";
n = 0;
SETARG(args[n], XmNautoUnmanage, False, n);
SETARG(args[n], XmNWidth, DIALOG_WIDTH, n);
SETARG(args[n], XmNHeight, DIALOG_HEIGHT - INFO_WINDOW_HEIGHT, n);
SETARG(args[n], XmNNoResize, True, n);
SETARG(args[n], XmNtitle, title, n);
SETARG(args[n], XmNIconName, title, n);
dialog_window = XmCreateFormDialog(parent, "sbsdialog", args, n);

/*
 * Save the dialog widget for later use.
 */
filedlg_data.parent = dialog_window;
init_data.parent = dialog_window;

/*
 * Create menubar.
 */

n = 0;
SETARG(args[n], XmNtopAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNleftAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNrightAttachment, XmATTACH_FORM, n);
menubar = XmCreateMenuBar(dialog_window, "sbsmenubar", args, n);
previous = menubar;
XtManageChild(menubar);

/*
 * Create spec menu. Callbacks for this menu are in Callbacks.C.
 */
specmenu = CreateMenu(menubar, "sbsmenu", "Spec", 'S');

(void)AddMenuItem(specmenu, "new", "New",
"Ctrl+N", "Ctrl<Key>n", 'N',
(XtCallbackProc) newCB,
(XtPointer) & filedlg_data);

(void)AddMenuItem(specmenu, "open", "Open",
"Ctrl+O", "Ctrl<Key>o", 'O',
(XtCallbackProc) openCB,
(XtPointer) & filedlg_data);
(void)AddMenuItem(specmenu, "edit",
"Edit",
"Ctrl+E", CtrlKey>e", 'E',
(XtCallbackProc) editCB,
(XtPointer) & filedlg_data);
(void)XtVaCreateManagedWidget("sep", xmSeparatorWidgetClass, specmenu, NULL);
(void)AddMenuItem(specmenu, "save",
"Save",
"Ctrl+S", CtrlKey>s", 'S',
(XtCallbackProc) saveCB,
(XtPointer) & filedlg_data);
(void)AddMenuItem(specmenu, "saveas",
"Save As",
"Ctrl+A", CtrlKey>a", 'A',
(XtCallbackProc) saveasCB,
(XtPointer) & filedlg_data);
(void)XtVaCreateManagedWidget("sep", xmSeparatorWidgetClass, specmenu, NULL);
(void)AddMenuItem(specmenu, "close",
"Close",
"Ctrl+W", CtrlKey>w", 'C',
(XtCallbackProc) closeCB,
(XtPointer) & init_data);
(void)AddMenuItem(specmenu, "exit",
"Exit",
"Ctrl+Q", CtrlKey>q", 'x',
(XtCallbackProc) exitCB,
(XtPointer) NULL);

/* * Create the component menu. Callbacks are implemented locally */

componentmenu = CreateMenu(menubar, "sbsmenu", "Component", 'C');
(void)AddMenuItem(componentmenu, "addfile",
"Add Component File",
"Ctrl+F", CtrlKey>f", 'F',
(XtCallbackProc) addCB,
(XtPointer) & init_data);
(void)AddMenuItem(componentmenu, "clearfile",
"Clear Component Files",
"Ctrl+C", CtrlKey>c", 'C',
(XtCallbackProc) clearfileCB,
(XtPointer) & init_data);
(void)AddMenuItem(componentmenu, "clearall",
"Clear All Files",
"Ctrl+L", CtrlKey<l", 'l',
(XtCallbackProc) clearallCB,
(XtPointer) & init_data);

/* * Create the init menu. Callbacks are implemented locally */

initmenu = CreateMenu(menubar, "sbsmenu", "Init", 'I');
(void)AddMenuItem(initmenu, "initsb",
"Initialize Software Base",
"Ctrl+I", CtrlKey>i", 'I',
(XtCallbackProc) initsbCB,
(XtPointer) & init_data);
(void)AddMenuItem(initmenu, "initdir",
"Initialize Directory"
"Ctrl+D", "Ctrl<Key>d", 'D',
(XtCallbackProc) initdirCB,
(XtPointer) & init_data);

/*
 * Create Help menu.
 */

helpmenu = CreateHelpMenu(menubar, "sbsmenu", "Help", 'H');

/*
 * Label the dialog to let the user know what it is for.
 */

xmstring = XmStringCreateSimple("CAPS Software Base Maintenance");
caps_label = XtVaCreateManagedWidget("sbslabel",
xmLabelWidgetClass, dialog_window,
xmLabelString, xmstring,
xmNtopAttachment, XmATTACH_WIDGET,
xmNtopWidget, previous,
xmNtopOffset, 5,
xmNbottomAttachment, XmATTACH_NONE,
xmNleftAttachment, XmATTACH_POSITION,
xmNleftPosition, 35,
xmNrightAttachment, XmATTACH_NONE,
NULL);
XmStringFree(xmstring);

previous = caps_label;

/*
 * Spec_label and spec_fname_text display the current spec file name for
 * the user
 */

xmstring = XmStringCreateSimple("PSDL Spec:");
spec_label = XtVaCreateManagedWidget("sbslabel",
xmLabelWidgetClass, dialog_window,
xmLabelString, xmstring,
xmNtopAttachment, XmATTACH_WIDGET,
xmNtopWidget, previous,
xmNtopOffset, 5,
xmNleftAttachment, XmATTACH_POSITION,
xmNleftPosition, 15,
NULL);
XmStringFree(xmstring);
spec_fname_text = XtVaCreateManagedWidget("sbstext",
xmTextWidgetClass, dialog_window,
xmNeditMode, XmSINGLE_LINE_EDIT,
xmNeditable, True,
xmNshadowThickness, 1,
xmNmarginHeight, 2,
xmNcursorPositionVisible, True,
xmNtopAttachment, XmATTACH_WIDGET,
xmNtopWidget, previous,
xmNtopOffset, 5,
xmNleftAttachment, XmATTACH_WIDGET,
xmNleftWidget, spec_label,
xmNleftOffset, 5,
xmNrightAttachment, XmATTACH_FORM,
xmNrightOffset, 10,
NULL);

/*
 * Save the widget so that the file and init callbacks can change the name
 * when the user enters a new one.
 */

filedlg_data.fname_text = spec_fname_text;
init_data.fname_text = spec_fname_text;

/*
 * This callback will update the file name and spec text if the user types
 * a return in the fname text box.
 */
XtAddCallback(spec_fname_text, XmNactivateCallback,
    (XtCallbackProc) textchangedCB,
    (XtPointer) & filedlg_data);
XtManageChild(spec_fname_text);

/*
 * Setup scrolled window to display spec text.
 */

scrolled_spec_window = XtVaCreateManagedWidget("sbswindow",
    xmScrolledWindowWidgetClass, dialog_window,
    XmNwidth, INFO_WINDOW_WIDTH,
    XmNheight, INFO_WINDOW_HEIGHT,
    XmNleftAttachment, XmATTACH_FORM,
    XmNrightAttachment, XmATTACH_FORM,
    XmNtopAttachment, XmATTACH_WIDGET,
    XmNtopWidget, spec_label,
    XmNtopOffset, 5,
    XmNbottomAttachment, XmATTACH_NONE,
    XmNresizeable, True,
    XmNleftOffset, 5,
    XmNrightOffset, 5,
    XmNbottomOffset, 5,
    XmNscrollingPolicy, XmAUTOMATIC,
    XmNscrollBarDisplayPolicy, XmSTATIC,
    XmNscrolledWindowMarginWidth, 5,
    XmNscrolledWindowMarginHeight, 5,
    XmNscrollVertical, False,
    XmNscrollHorizontal, True,
    NULL);

spec_text = XtVaCreateManagedWidget("sbstext",
    xmTextWidgetClass, scrolled_spec_window,
    XmNwidth, INFO_WINDOW_WIDTH,
    XmNheight, INFO_WINDOW_HEIGHT,
    XmNeditMode, XmMULTI_LINE_EDIT,
    XmNshadowThickness, 0,
    XmNmarginHeight, 0,
    XmNscrollHorizontal, False,
    XmNeditable, False,
    XmNtraversalOn, False,
    XmNcursorPositionVisible, False,
    NULL);

/*
 * Save the widget so that the spec text can be updated when the user
 * changes the file name or edits that text
 */

filedlg_data.text = spec_text;
init_data.spec_text = spec_text;

XmScrolledWindowSetAreas(scrolled_spec_window,
    (Widget) NULL,
    (Widget) NULL,
    spec_text);
XtManageChild(spec_text);

/*
 * If the user is restarting the dialog try to use the file name that was
 * entered
 */

if (save_spec_filename != NULL)
{
    char *text;
    XmTextSetString(spec_fname_text, save_spec_filename);
    if (LoadFile(save_spec_filename, &text) != -1 & text != NULL)
{  XmTextSetString(spec_text, text);
  XtFree(text);
}
XtFree(save_spec_filename);
save_spec_filename = NULL;
}  
/*
 * Filelist_label and filelist_text display the component file list that
 * will be copied into the component directory
 */
xmstring = XmStringCreateSimple("Component Files:");
filelist_label = XtVaCreateManagedWidget("sbslabel",
xmLabelWidgetClass, dialog_window,
xmNewLabelString, xmstring,
xmNtopAttachment, XmATTACH_WIDGET,
xmNtopWidget, scrolled_spec_window,
xmNtopOffset, 5,
xmNleftAttachment, XmATTACH_POSITION,
xmNleftPosition, 15,
NULL);
XmStringFree(xmstring);
scrolled_filelist_window = XtVaCreateManagedWidget("sbswindow",
xmScrolledWindowWidgetClass, dialog_window,
xmNwidth, INFO_WINDOW_WIDTH,
xmNheight, INFO_WINDOW_HEIGHT,
xmNleftAttachment, XmATTACH_FORM,
xmNrightAttachment, XmATTACH_FORM,
xmNtopAttachment, XmATTACH_WIDGET,
xmNtopWidget, filelist_label,
xmNtopOffset, 5,
xmNbottomAttachment, XmATTACH_NONE,
xmNresizable, True,
xmNleftOffset, 5,
xmNrightOffset, 5,
xmNbottomOffset, 5,
xmNscrollingPolicy, XmAUTOMATIC,
xmNscrollBarDisplayPolicy, XmSTATIC,
xmNscrollWindowMarginWidth, 5,
xmNscrollWindowMarginHeight, 5,
xmNscrollVertical, False,
xmNscrollHorizontal, True,
NULL);
previous = scrolled_filelist_window;
filelist_text = XtVaCreateManagedWidget("sbstext",
xmTextWidgetClass, scrolled_filelist_window,
xmNwidth, INFO_WINDOW_WIDTH,
xmNheight, INFO_WINDOW_HEIGHT,
xmNeditMode, XmMULTI_LINE_EDIT,
xmNshadowThickness, 0,
xmNmarginHeight, 0,
xmNmarginHeight, 0,
xmNscrollHorizontal, True,
xmNeditable, True,
xmNtraversalOn, True,
xmNcursorPositionVisible, True,
NULL);
/
/*
 * Save the widget so that the init callbacks can update the file list when
 * the user enters changes.
 */
init_data.flist_text = filelist_text;
XmScrolledWindowSetAreas(scrolled_filelist_window,
(Widget) NULL,
(Widget) NULL,
filelist_text);
XtManageChild(filelist_text);

/* Create the push buttons at the bottom of the dialog box. */

xmstring = XmStringCreateSimple("Dismiss") ;
dismiss_btn = XtVaCreateManagedWidget("sbslabel",
  xmPushButtonWidgetClass, dialog_window,
  XmLabelString, xmstring,
  XmWidth, BUTTON_WIDTH,
  XmHeight, BUTTON_HEIGHT,
  XmLeftAttachment, XmATTACH_FORM,
  XmLeftOffset, 20,
  XmRightAttachment, XmATTACH_NONE,
  XmTopAttachment, XmATTACH_NONE,
  XmBottomAttachment, XmATTACH_FORM,
  XmBottomOffset, 10,
  XmStringFree(xmstring);
XtAddCallback(dismiss_btn,
  XmNactivateCallback,
  (XtCallbackProc) closeCB,
  (XtPointer) & init_data);
XtManageChild(dismiss_btn);

xmstring = XmStringCreateSimple("Init DIR");
initdir_btn = XtVaCreateManagedWidget("sbslabel",
  xmPushButtonWidgetClass, dialog_window,
  XmLabelString, xmstring,
  XmWidth, BUTTON_WIDTH,
  XmHeight, BUTTON_HEIGHT,
  XmLeftAttachment, XmATTACH_WIDGET,
  XmLeftWidget, dismiss_btn,
  XmLeftOffset, 10,
  XmRightAttachment, XmATTACH_NONE,
  XmTopAttachment, XmATTACH_NONE,
  XmBottomAttachment, XmATTACH_FORM,
  XmBottomOffset, 10,
  XmStringFree(xmstring);
XtAddCallback(initdir_btn,
  XmNactivateCallback,
  (XtCallbackProc) initdirCB,
  (XtPointer) & init_data);
XtManageChild(initdir_btn);

xmstring = XmStringCreateSimple("Init SB");
initsb_btn = XtVaCreateManagedWidget("sbslabel",
  xmPushButtonWidgetClass, dialog_window,
  XmLabelString, xmstring,
  XmWidth, BUTTON_WIDTH,
  XmHeight, BUTTON_HEIGHT,
  XmLeftAttachment, XmATTACH_WIDGET,
  XmLeftWidget, initdir_btn,
  XmLeftOffset, 10,
  XmRightAttachment, XmATTACH_NONE,
  XmTopAttachment, XmATTACH_NONE,
  XmBottomAttachment, XmATTACH_FORM,
  XmBottomOffset, 10,
  XmStringFree(xmstring);
XtAddCallback(initsb_btn,
  XmNactivateCallback,
  (XtCallbackProc) initsbCB,
  (XtPointer) & init_data);
XtManageChild(initsb_btn);
xmstring = XmStringCreateSimple("Clear FL");
clearfl_btn = XtVaCreateManagedWidget("sbslabel",
    XmPushButtonWidgetClass, dialog_window,
    XmLabelString, xmstring,
    XmWidth, BUTTON_WIDTH,
    XmHeight, BUTTON_HEIGHT,
    XmLeftAttachment, XmATTACH_WIDGET,
    XmLeftWidget, initsb_btn,
    XmLeftOffset, 10,
    XmRightAttachment, XmATTACH_NONE,
    XmTopAttachment, XmATTACH_NONE,
    XmBottomAttachment, XmATTACH_FORM,
    XmBottomOffset, 10,
    NULL);
XmStringFree(xmstring);

XtAddCallback(clearfl_btn,
    XmNactivateCallback,
    (XtCallbackProc) clearflCB,
    (XtPointer) & init_data);
XtManageChild(clearfl_btn);

xmstring = XmStringCreateSimple("Clear All");
clearall_btn = XtVaCreateManagedWidget("sbslabel",
    XmPushButtonWidgetClass, dialog_window,
    XmLabelString, xmstring,
    XmWidth, BUTTON_WIDTH,
    XmHeight, BUTTON_HEIGHT,
    XmLeftAttachment, XmATTACH_WIDGET,
    XmLeftWidget, clearfl_btn,
    XmLeftOffset, 10,
    XmRightAttachment, XmATTACH_NONE,
    XmTopAttachment, XmATTACH_NONE,
    XmBottomAttachment, XmATTACH_FORM,
    XmBottomOffset, 10,
    NULL);
XmStringFree(xmstring);

XtAddCallback(clearall_btn,
    XmNactivateCallback,
    (XtCallbackProc) clearallCB,
    (XtPointer) & init_data);
XtManageChild(clearall_btn);
XtManageChild(dialog_window);

/*
* Initialize the progress display this is done one time from the search
dialog or init dialog.
*/
initialize_display(parent);
}

/*
* Clear all of the init dialog text displays.
*/
static void
clear(INIT_DATA * data)
{
    XmTextSetString(data->fname_text, "");
    XmTextSetString(data->spec_text, "");
    XmTextSetString(data->flist_text, "");
}

/*
* Initialize the component directory. Create the directory using the spec file
* name with out the extension as the directory name. Copy the spec file and
* all files in the file list into the component directory.
*/
static int

59
initdirectory(char *sfname, INIT_DATA * data) {
    char *sbfname;
    char *component;
    char *extension;
    char *file;
    char IObuffer[MAXBUFSIZE];
    char *flist;
    char *path;
    char *newline;
    char *format = "Copying: %s
    To: %s\n\n";

    /*
    * Show the user what is happening in the progress display. The display was
    * cleared by the calling routine.
    */
    display_message("Doing Directory Initialization\n\n");

    /*
    * Create the component directory using the spec file name without the
    * extension as the directory name.
    */
    sbfname = getfilename(sfname);
    component = catstrings(SBROOT, sbfname);
    extension = strrchr(component, '.');
    if (extension != NULL)
        {  
        strcpy(extension, "/");
        }
    sprintf(IObuffer, "Creating directory: %s\n\n", component);
    display_message(IObuffer);

    if (mkdir(component, 0755) == -1)
        {  
        sprintf(IObuffer, "error creating component directory %s\n\n", component);
        ModalWarningDialog(data->parent, "Error", IObuffer);
        return True;
        }

    /*
    * Copy the spec file into PSDL_SPEC which is used by the ADA search
    * routines. For convenience also copy the file into the component
    * directory.
    */
    file = catstrings(component, "PSDL_SPEC");
    sprintf(IObuffer, format, sfname, file);
    display_message(IObuffer);
    CopyFile(sfname, file);
    XtFree(file);

    file = catstrings(component, sbfname);
    sprintf(IObuffer, format, sfname, file);
    display_message(IObuffer);
    CopyFile(sfname, file);
    XtFree(file);

    XtFree(sbfname);

    /*
    * Now copy all of the files in the file list into the component directory
    */
    flist = XmTextGetString(data->flist_text);
    newline = strchr(flist, '\n');

    for (path = flist; newline != NULL; newline = strchr(path, '\n'))
        {  
        *newline = '\0';
        }
sbfname = getfilename(path);
file = catstrings(component, sbfname);
sprintf(IObuffer, format, path, file);
display_message(IObuffer);
CopyFile(path, file);
XtFree(sbfname);
XtFree(file);
path = newline + 1;
}
XtFree(flist);
XtFree(component);
display_message("Done\n");
return False;

/*
 * Callback closes the dialog and saves the spec file name for later. The data
 * for this callback is passed through the client data as a pointer to a
 * INIT_DATA structure.
 */
static void
closeCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    INIT_DATA *data = (INIT_DATA *) client_data;
    if (save_spec_filename != NULL)
        XtFree(save_spec_filename);
    save_spec_filename = gettext(data->fname_text);
    if (XtIsManaged(data->parent))
        XtUnmanageChild(data->parent);
    done_with_dialog = True;
}

/*
 * Callback exits program.
 */
static void
exitCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    exit(O);
}

/*
 * Callback to initialize the component directory. The data for this callback is
 * passed through the client data as a pointer to a INIT_DATA structure.
 */
static void
initdirCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    INIT_DATA *data = (INIT_DATA *) client_data;
    char *sbfname = gettext(data->fname_text);
    /*
     * Clear the progress display and change its title
     */
    clear_display("Initializing SB Directory");
/ * If the user had not entered a spec file name no need to init directory */
if (sfname != NULL) {
  /* If the directory initialization errors out don't clear the dialog. */
  if (!initdirectory(sfname, data))
    clear(data);
  XtFree(sfname);
}

/*
 * Callback to initialize the software base. The data for this callback is
 * passed through the client data as a pointer to a INIT_DATA structure. If the
 * user has entered a spec initialize the component directory first. Then create
 * sb_header.dat file before running the initialization routine.
 */

static void
initsbCB(Widget widget,
         XtPointer client_data,
         XtPointer call_data)
{
  INIT_DATA *data = (INIT_DATA *) client_data;
  FILE *fp;
  struct dirent *entry;
  DIR *dirp;
  struct stat file_info;
  long ID = 1000;
  char *sfname = gettext(data->fname_text);
  char *file;
  char *header_file_name;

  /* Let the user know what is happening */
  clear_display("Initializing Software Base");
  display_message("Doing software Base Initialization \n\n");

  /* Initialize the component directory if their is a spec name */
  if (sfname != NULL) {
    /* If the initialization errors out stop and give the user a chance to
     * fix the problems */
    if (initdirectory(sfname, data))
      return;
  }

  /* Create the header file in the software base root directory */
  header_file_name = catstrings(SBROOT, "sb_header.dat");
  fp = fopen(header_file_name, "w");

  /* Loop through the software base root directory adding the component
   * directories to the header file */
dirp = opendir(SBROOT);
for (entry = readdir(dirp); entry != NULL; entry = readdir(dirp))
{
  if (entry->d_name[0] == '.
      continue;
/*
 * Only add directory names to the header file.
 */
file = catstrings(SBROOT, entry->d_name);
if (stat(file, &file_info) != 0)
{
  closedir(dirp);
  XtFree(file);
  break;
}
if (S_ISDIR(file_info.st_mode))
{
  fprintf(fp, "%ld %s\n", ID, file);
  ID += 1000;
}
XtFree(file);
}
closedir(dirp);
close(fp);
/*
 * Run the ADA software base initialization. Progress messages are displayed
 * to let the user know what is happening
 */
sb_init(SBROOT, header_file_name);
XtFree(header_file_name);
/*
 * Clear the dialog so that the user can enter another spec if needed.
 */
clear(data);
display_message("Done\n");
*/
/*
 * OK callback for adding components to the file list. The data for this
 * callback is * passed through the client data as a pointer to an INIT_DATA
 * structure and * the call data is a pointer to a
 * XmFileSelectionBoxCallbackStruct. The file * name is take from the call data
 * and added to the file list text.
 */
static void
addokCB(Widget widget,
        XtPointer client_data,
        XtPointer call_data)
{
  INIT_DATA *data = (INIT_DATA *) client_data;
  XmFileSelectionBoxCallbackStruct *ptr;
  char *filename;

  /* Convert the file name to a character string.
   */
  ptr = (XmFileSelectionBoxCallbackStruct *) call_data;
  XmStringGetLtoR(ptr->value, XmSTRING_DEFAULT_CHARSET, &filename);

  /* Take the dialog box off of the screen and free up its data
   */
XtUnmanageChild(data->parent);
XtDestroyWidget(data->parent);

/*
 * If the user did not select a file we are done
 */
if (filename == NULL)
    return;

if (save_component_dir != NULL)
    XtFree(save_component_dir);

/*
 * Save the directory for the next time the user wants to open a file. This
 * will make the task of entering multiple files from the same directory
 * easier.
 */
save_component_dir = getdirname(filename);

/*
 * Display the file list in its text widget. This is also the place we save
 * the file list for later use.
 */
XmTextInsert(data->flist_text, 10000, filename);
XmTextInsert(data->flist_text, 10000, "\n");
XtFree(filename);
}

/*
 * Cancel callback for the file list selection dialog.
 */
static void
addcancelCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    XtUnmanageChild((Widget) client_data);
    XtDestroyWidget((Widget) client_data);
}

/*
 * Add file dialog callback. Displays a file selection dialog to the user.
 * The data for this callback is passed through the client data as a pointer
 * to an INIT_DATA structure. User input is returned through the addokCB
 * callback. The selected file will be added to the filelist
 */
static void
addCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    INIT_DATA *data = (INIT_DATA *) client_data;
    static INIT_DATA ok_data;
    Widget fsdialog;
    Arg args[20];
    Cardinal n;
    XmString title = XmStringCreateSimple("Add to file list");

    /*
     * Create the file selection dialog.
     */
    n = 0;
    SETARG(args[n], XmNdialogTitle, title, n);
    fsdialog = XmCreateFileSelectionDialog(data->parent, "sbsdialog", args, n);
    XmStringFree(title);
    if (save_component_dir != NULL)
Set_Res_String(fsdialog, XmNdirectory, save_component_dir);

/*
 * Change the parent widget to the file selection dialog so we can close it
 * and not its parent. The rest of the data is pasted through.
 */

ok_data.parent = fsdialog;
ok_data.flist_text = data->flist_text;
ok_data.fname_text = data->fname_text;

/*
 * Add the callbacks to the OK and cancel buttons.
 */

 XtAddCallback(fsdialog, XmNokCallback,
    (XtCallbackProc) addokCB,
    (XtPointer) & ok_data);

 XtAddCallback(fsdialog, XmNcancelCallback,
    (XtCallbackProc) addcancelCB,
    (XtPointer) fsdialog);

 XtManageChild(fsdialog);
}

/*
 * Callback to clear the component file list. The data for this callback is
 * passed through the client data as a pointer to a INIT_DATA structure.
 */

 static void
clearflCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    INIT_DATA *data = (INIT_DATA *) client_data;

    XmTextSetString(data->flist_text, "");
}

/*
 * Callback to clear the init dialog. The data for this callback is passed
 * through the client data as a pointer to a INIT_DATA structure.
 */

 static void
clearallCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    INIT_DATA *data = (INIT_DATA *) client_data;

    clear(data);
}

J. MENU.H

/*
 * $Id: Menu.h,v 1.4 1998/01/18 16:35:45 greg Exp $
 * Menu.h -- Software Base Search Interface
 * Header file for the dialogs menus.
 * Naval Postgraduate School
 * January 13, 1998
 * Written by Gregory L. Meckstroth
 */
Widget
CreateMenu(Widget parent,
    char *name,
    char *label,
    char mnemonic);

Widget
CreateHelpMenu(Widget parent,
    char *name,
    char *label,
    char mnemonic);

Widget
AddMenuItem(Widget parent,
    char *name,
    char *label,
    char *acc_text,
    char *acc_key,
    char mnemonic,
    XtCallbackProc callback,
    XtPointer client_data);

K. MENU.C

/*
 *
 * Menu.C -- Software Base Search Interface
 *
 * Source code for the creation of the menus used in the dialogs. These routine
 * create cascade style menus, add menu items, and implement the cascade style
 * help menu.
 *
 * Entry points: CreateMenu, CreateHelpMenu, AddMenuItem.
 *
 * Naval Postgraduate School
 * January 13, 1998
 *
 * Written by Gregory L. Meckstroth
 *
*/

#include <stdio.h>
#include <stdlib.h>
#include <Xm/Xm.h>
#include <Xm/CascadeB.h>
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>
#include <Xm/Separator.h>
#include "Gui.h"
#include "Menu.h"
#include "HelpMessages.h"
#include "DisplayProgress.h"

/*
 * Declarations for local functions.
 */

static void
helpCB(Widget widget, XtPointer client_data, XtPointer call_data);

/*
 * Create a cascade style menu with a widget name, label and mnemonic.
 */

Widget
CreateMenu(Widget parent,
    char *name,    // Parent for the menu.
    char *name,    // Widget name.
    char *label,   // Menu label.
    char *label,
    char mnemonic,
    XtCallbackProc callback,
    XtPointer client_data);
char mnemonic) // Shortcut key.
{
    Widget menu;
    Widget cascade;
    XmString xmstring;
    Arg args[20];
    Cardinal n;

    /* Create pull down menu. Engage tear-off menu. */
    xmstring = XmStringCreateSimple(label);
    n = 0;
    SETARG(args[n], XmNtearOffModel, XmTEAR_OFF_ENABLED, n);
    menu = XmCreatePulldownMenu(parent, name, args, n);

    /* Create cascade button and connect to menu. */
    cascade = XtVaCreateWidget(name,
                               xmCascadeButtonWidgetClass, parent,
                               XmNlabelString, xmstring,
                               XmNsubMenuId, menu,
                               XmNmnemonic, mnemonic,
                               NULL);
    XtManageChild(cascade);

    return menu;
}

/* Create the help menu to the far right with items maintenance, search and */
/* version. The menu will have a keyboard shortcut */

Widget CreateHelpMenu(Widget parent, // Parent for the menu.
                      char *name,    // Widget name.
                      char *label,  // Menu label. 
                      char mnemonic) // Shortcut key.
{
    Widget menu;
    Widget cascade;
    XmString xmstring;
    Arg args[20];
    Cardinal n;

    /* Create the pull down menu. */
    xmstring = XmStringCreateSimple(label);
    n = 0;
    SETARG(args[n], XmNtearOffModel, XmTEAR_OFF_ENABLED, n);
    menu = XmCreatePulldownMenu(parent, name, args, n);

    /* Create cascade button and connect to menu. */
    cascade = XtVaCreateWidget(name,
                               xmCascadeButtonWidgetClass, parent,
                               XmNlabelString, xmstring,
                               XmNsubMenuId, menu,
                               XmNmnemonic, mnemonic,
                               NULL);
/* Make this the help menu which will place it at the far right. */
XtVaSetValues(parent, XmNmenuHelpWidget, cascade, NULL);

/* Set up menu choices for the Help menu. */

(void)AddMenuItem(menu, "maintenancehelp",
"Maintenance",
NULL, NULL, 'I',
(XtCallbackProc) helpCB,
(XtPointer) & maintenance_help_message);

(void)AddMenuItem(menu, "searchhelp",
"Search",
NULL, NULL, 'S',
(XtCallbackProc) helpCB,
(XtPointer) search_help_message);

(void) XtVaCreateManagedWidget("sep", xmSeparatorWidgetClass, menu, NULL);

(void)AddMenuItem(menu, "version",
"Version",
NULL, NULL, 'V',
(XtCallbackProc) helpCB,
(XtPointer) version_message);

XtManageChild(cascade);

return menu;
}

/* Add a menu item to the parent, which is assumed to be a menu widget. */

Widget
AddMenuItem(Widget parent,
// Parent for item. char *name,
// Widget name.
char *label, // Item label.
char *acc_text, // Accelerator text. char *acc_key, // Accelerator key. char mnemonic, // Shortcut key.
XtCallbackProc callback, // Callback for item
XtPointer client_data) // Client data for callback.
{
Widget button;
XmString stl = XmStringCreateSimple(label);
XmString st2 = XmStringCreateSimple(acc_text);

/* Create the push button. the label and accelerator text are strings not */
/* chars */

button = XtVaCreateManagedWidget(name, xmPushButtonWidgetClass, parent,
XmNlabelString, st1,
XmNacceleratorText, st2,
XmNaccelerator, acc_key,
XmNmnemonic, mnemonic,
NULL);

XtAddCallback(button, XmNactivateCallback, callback, client_data);

XmStringFree(st1);
XmStringFree(st2);

XtManageChild(button);

return button;
)
Callback for the help menu items opens the progress display and add the help message. The data for this callback is passed through the client data as a pointer to an array of character strings. The first string is used as the dialog title. The last string is NULL.

```c
static void helpCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    char **message = (char **)client_data;
    /* Make sure that we have a message if not we are done. */
    if (message != NULL)
    {
        /* Clear the display and set the dialog title. */
        clear_display(*message);
        /* Loop through the message array adding the strings to the display */
        for (char **c = message + 1; *c != (char *)NULL; c++)
        {
            display_message(*c);
        }
    /* Make sure that we are at the top of the messages */
    home_display();
    }
}

L. PROMPTDIALOG.H

/* $Id: PromptDialog.h,v 1.1.1.1 1998/01/14 03:40:05 greg Exp $ */
/* PromptDialog.h -- Software Base Search Interface */
/* Header file for the GUI prompt dialogs. */
/* Naval Postgraduate School */
/* January 13, 1998 */
/* Written by Gregory L. Meckstroth */

int ModalPromptDialog(Widget parent, char *title, char *message);
void ModalWarningDialog(Widget parent, char *title, char *message);

M. PROMPTDIALOG.C

/* $Id: PromptDialog.C,v 1.2 1998/01/18 16:35:45 greg Exp $ */
/* PromptDialog.C -- Software Base Search Interface */
Source code for the modal prompt dialogs.

Entry points: ModalWarningDialog and ModalPromptDialog.

Naval Postgraduate School
January 13, 1998

Written by Gregory L. Meckstroth

#include <Xm/Xm.h>
#include <Xm/MessageB.h>
#include <Xm/Text.h>

#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include "Gui.h"
#include "Utils.h"

The PROMPT_DATA structure is used to pass data to the callback routines.

done with dialog: While true the event loop will continue running. Setting
this to false allows the event loop to exit. return_value: Set by the
Callback routine to the modal dialog return value.

struct PROMPT_DATA
{
    int done_with_dialog;
    int return_value;
};

Declarations for local functions.

static void
okCB(Widget widget, XtPointer client_data, XtPointer call_data);

static void
cancelCB(Widget widget, XtPointer client_data, XtPointer call_data);

Display a modal dialog with a message for the user then wait for the user to
respond before allowing further program execution. Returns false if the ok
button is pressed and false for the cancel button.

int
ModalPromptDialog(Widget parent,
    char *title,
    char *message)
{
    Widget dialog;
    Widget button;
    Arg args[20];
    Cardinal argcount;
    PROMPT_DATA cdata;

    cdata.done_with_dialog = False;

    /* Create a question dialog box to display the message to the user.
     * Customize the title, ok and cancel buttons, remove the help button, and
     * add callbacks for the ok and cancel buttons.
     */

    argcount = 0;
    SETARG(args[argcount], XmdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
        argcount);

    ...
dialog = XmCreateQuestionDialog(parent, "sbstext", args, argcount);
Set_Res_String(dialog, XmNmessageString, message);
Set_Res_String(dialog, XmNdialogTitle, title);
Set_Res_String(dialog, XmNokLabelString, "Yes");
Set_Res_String(dialog, XmNcancelLabelString, "No");
button = XmMessageBoxGetChild(dialog, XmDIALOG_HELP_BUTTON);
XtUnmanageChild(button);

/*
* Set the callbacks for the OK and cancel buttons so we will get
* control when the button is pressed.
*/
XtAddCallback(dialog, XmNokCallback, (XtCallbackProc) okCB, (XtPointer) & cdata);
XtAddCallback(dialog, XmNcancelCallback, (XtCallbackProc) cancelCB, (XtPointer) &
cdata);
XtManageChild(dialog);

/*
* Wait for a response to the dialog. When the OK or cancel button is
* pressed done_with_dialog will be set true causing the loop to terminate.
* Return_value is also set to the appropriate value in the callback.
*/
while (!cdata.done_with_dialog)
XtAppProcessEvent(XtWidgetToApplicationContext(dialog), XtIMAll);
return cdata.return_value;

void
ModalWarningDialog(Widget parent, char *title, char *message)
{
Widget dialog;
Widget button;
Arg args[20];
Cardinal argcount;
PROMPT_DATA cdata;
cdata.done_with_dialog = False;

/*
* Create a warning dialog box to display the message to the user.
*/
argcount = 0;
SETARG(args[argcount], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL, argcount);
dialog = XmCreateWarningDialog(parent, "", args, argcount);
Set_Res_String(dialog, XmNmessageString, message);

/*
* Customize the title, remove the ok and help buttons. Change the name of
* the cancel button to dismiss.
*/
Set_Res_String(dialog, XmNdialogTitle, title);
Set_Res_String(dialog, XmNcancelLabelString, "Dismiss");
button = XmMessageBoxGetChild(dialog, XmDIALOG_OK_BUTTON);
XtUnmanageChild(button);
button = XmMessageBoxGetChild(dialog, XmDIALOG_HELP_BUTTON);
XtUnmanageChild(button);
XtAddCallback(dialog, XmNcancelCallback, (XtCallbackProc) cancelCB, (XtPointer) & cdata);
XtManageChild(dialog);

while (!cdata.done_with_dialog)
    XtAppProcessEvent(XtWidgetToApplicationContext(dialog), XtIMAll);

static void
okCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    PROMPT_DATA *cptr = (PROMPT_DATA *) client_data;
    cptr->done_with_dialog = True;
    cptr->return_value = False;
}

static void
cancelCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    PROMPT_DATA *cptr = (PROMPT_DATA *) client_data;
    cptr->done_with_dialog = True;
    cptr->return_value = True;
}

N. SDE.H

void SDE(Widget parent, char **filename, char **string);
O. SDE.C

/*
 * $Id: SDE.C,v 1.3 1998/01/18 16:35:45 greg Exp $
 * SDE.C -- Software Base Search Interface
 * Source code for the temporary "Syntax Directed Editor". The functionality
 * that is supplied by SDE will be replaced by the Syntax directed Editor in
 * CAPS. SDE is a simple editor and not a real syntax directed editor.
 * Creating a functional Syntax Directed Editor is beyond the scope of this
 * Thesis.
 * Entry points: SDE.
 * Naval Postgraduate School
 * January 11, 1998
 * Written by Gregory L. Meckstroth
 */

#include <Xm/Xm.h>
#include <Xm/Text.h>
#include <Xm/Label.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/Separator.h>
#include "Gui.h"
#include "SearchDialog.h"
#include "Menu.h"
#include "Callbacks.h"
#include "Util.h"
#include "PromptDialog.h"
#include "SDE.h"

#define EDITOR_RUNNING 'r'
#define EDITOR_DONE 'd'
#define EDITOR_CANCELED 'c'

/*
 * Declarations for local functions.
 */

static void cutCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void copyCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void pasteCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void okCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void cancelCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void closeCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void exitCB(Widget widget, XtPointer client_data, XtPointer call_data);

/*
 * A simple text editor for making changes to the PSDL query/spec. The
 * functionality that is supplied by this editor will be replaced by the CAPS
 * Syntax Directed Editor. The interface to the CAPS SDE was unknown at the
 * time that this editor was written so some modifications to other routines
 */
void SDE(Widget parent, char **filename, char **string)
{
    Widget dialog;
    Widget ok_btn;
    Widget cancel_btn;
    Widget query_text;
    Widget menubar;
    Widget filemenu;
    Widget editmenu;
    Widget filename_text;
    Widget label;

    Arg args[20];
    Cardinal n;
    XmString xmstring;
    char *title;

    /*
     * The variable editor_state is used to determine when to exit the event
     * loop. It has three states running, done, and canceled. The initial state
     * is running. When the user presses the OK or cancel button the state is
     * changed and the event loop will exit.
     */
    static char editor_state;

    /*
     * The variable callback_data is used to pass data to the file callbacks.
     */
    static FILEDLG_DATA callback_data;

    /*
     * Create the editors dialog box
     */
    title = "SD Editor";
    n = 0;
    SETARG[args[n], XmNautoUnmanage, False, n];
    SETARG[args[n], XmNwidth, DIALOG_WIDTH, n];
    SETARG[args[n], XmNheight, DIALOG_HEIGHT / 2, n];
    SETARG[args[n], XmNtitle, title, n];
    SETARG[args[n], XmNiconName, title, n];
    dialog = XmCreateFormDialog(parent, "sbsdialog", args, n);

    /*
     * save the dialog widget for the file callback.
     */
    callback_data.parent = dialog;

    /*
     * Create menubar with file and edit menus.
     */
    n = 0;
    SETARG[args[n], XmNtopAttachment, XmATTACH_FORM, n];
    SETARG[args[n], XmNleftAttachment, XmATTACH_FORM, n];
    SETARG[args[n], XmNrightAttachment, XmATTACH_FORM, n];
    menubar = XmCreateMenuBar(dialog, "sbsmenu", "File", 'F');
    XtManageChild(menubar);

    filemenu = CreateMenu(menubar, "sbsmenu", "File", 'F');
(void)AddMenuItem(filemenu, "open", "Open", "Ctrl+O", "Ctrl<Key>O", 'O', (XtCallbackProc) openCB, (XtPointer) & callback_data);

(void)AddMenuItem(filemenu, "edit", "Edit", "Ctrl+E", "Ctrl<Key>e", 'E', (XtCallbackProc) editCB, (XtPointer) & callback_data);

(void)XtVaCreateManagedWidget("sep", xmSeparatorWidgetClass, filemenu, NULL);

(void)AddMenuItem(filemenu, "save", "Save", "Ctrl+S", "Ctrl<Key>s", 'S', (XtCallbackProc) saveCB, (XtPointer) & callback_data);

(void)AddMenuItem(filemenu, "saveas", "Save As", "Ctrl+A", "Ctrl<Key>a", 'A', (XtCallbackProc) saveasCB, (XtPointer) & callback_data);

(void)XtVaCreateManagedWidget("sep", xmSeparatorWidgetClass, filemenu, NULL);

(void)AddMenuItem(filemenu, "close", "Close", "Ctrl+W", "Ctrl<Key>w", 'C', (XtCallbackProc) closeCB, (XtPointer) dialog);

(void)AddMenuItem(filemenu, "exit", "Exit", "Ctrl+Q", "Ctrl<Key>Q", 'x', (XtCallbackProc) exitCB, (XtPointer) dialog);

editmenu = CreateMenu(menubar, "edit", "Edit", 'E');

(void)AddMenuItem(editmenu, "cut", "Cut", "Ctrl+X", "Ctrl<Key>x", 'T', (XtCallbackProc) cutCB, (XtPointer) & callback_data);

(void)AddMenuItem(editmenu, "copy", "Copy", "Ctrl+C", "Ctrl<Key>c", 'C', (XtCallbackProc) copyCB, (XtPointer) & callback_data);

(void)AddMenuItem(editmenu, "paste", "Paste", "Ctrl+V", "Ctrl<Key>v", 'P', (XtCallbackProc) pasteCB, (XtPointer) & callback_data);

/*
 * Create a text box for the file name so that the user will know which
 * file they are editing.
 */

xmstring = XmStringCreateSimple("File name:");
label = XtVaCreateManagedWidget("sbslabel", XmLabelWidgetClass, dialog,
XmNlabelString, xmstring, XmNLabelString, xmstring,
XmNtopAttachment, XmATTACH_WIDGET, XmNtopWidget, menubar,
XmNtopOffset, 5, XmNleftAttachment, XmATTACH_FORM,
XmNleftOffset, 5,
XmNrightAttachment, XmATTACH_NONE,
        NULL);
XtManageChild(label);
XmStringFree(xmstring);

filename_text = XtVaCreateManagedWidget("sbstext",
        xmTextWidgetClass, dialog,
        XmNeditMode, XmSINGLE_LINE_EDIT,
        XmNeditable, True,
        XmNshadowThickness, 1,
        XmNmarginHeight, 2,
        XmNcursorPositionVisible, True,
        XmNtopAttachment, XmATTACH_WIDGET,
        XmNtopWidget, menubar,
        XmNtopOffset, 5,
        XmNleftAttachment, XmATTACH_WIDGET,
        XmNleftWidget, label,
        XmNleftOffset, 5,
        XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5,
        NULL);

/* * Save the filename text widget for use in the file callbacks. */
callback_data.fname_text = filename_text;

/* * If there is an initial file name display it for the user to see. */
if (*filename != NULL)
    XmTextSetString(filename_text, *filename);
XtManageChild(filename_text);

/* * Setup the ok button before the query text widget so that the bottom of
 * the query text widget can be attached to the ok button. */
ok_btn = XtVaCreateManagedWidget("OK",
        xmPushButtonWidgetClass, dialog,
        XmNwidth, BUTTON_WIDTH,
        XmNheight, BUTTON_HEIGHT,
        XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 20,
        XmNrightAttachment, XmATTACH_NONE,
        XmNtopAttachment, XmATTACH_NONE,
        XmNbottomAttachment, XmATTACH_FORM,
        XmNbottomOffset, 10,
        NULL);
XtAddCallback(ok_btn,
        XmNactivateCallback,
        (XtCallbackProc) okCB,
        (XtPointer) & editor_state);
XtManageChild(ok_btn);

cancel_btn = XtVaCreateManagedWidget("Cancel",
        xmPushButtonWidgetClass, dialog,
        XmNwidth, BUTTON_WIDTH,
        XmNheight, BUTTON_HEIGHT,
        XmNleftAttachment, XmATTACH_WIDGET,
        XmNleftOffset, ok_btn,
        XmNleftOffset, 10,
        XmNrightAttachment, XmATTACH_NONE,
        XmNtopAttachment, XmATTACH_NONE,
        XmNbottomAttachment, XmATTACH_FORM,
        XmNbottomOffset, 10,
XtAddCallback(cancel_btn, XmNactivateCallback, (XtCallbackProc) cancelCB, (XtPointer) & editor_state);

XtManageChild(cancel_btn);

/*
 * Create the text edit widget
 */

n = 0;
SETARG(args[n], XmNeditMode, XmMULTI_LINE_EDIT, n);
SETARG(args[n], XmNtopAttachment, XmATTACH_WIDGET, n);
SETARG(args[n], XmNtopWidget, filename_text, n);
SETARG(args[n], XmNtopOffset, 5, n);
SETARG(args[n], XmNleftAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNleftOffset, 5, n);
SETARG(args[n], XmNrightAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNrightOffset, 5, n);
SETARG(args[n], XmNbottomAttachment, XmATTACH_WIDGET, n);
SETARG(args[n], XmNbottomWidget, ok_btn, n);
SETARG(args[n], XmNbottomOffset, 5, n);

query_text = XmCreateScrolledText(dialog, "sbstext", args, n);

/*
 * Save the text widget for the file callbacks.
 */

callback_data.text = query_text;

XtManageChild(query_text);

/*
 * If we have an initial file name try to load the text from it.
 */

if (*filename)
{
    char *text;
    if (LoadFile(*filename, &text) != -1)
    {
        XmTextSetString(query_text, text);
        XtFree(text);
    }
}

XtManageChild(dialog);

/*
 * Wait for the editor to exit.
 */

editor_state = EDITOR_RUNNING;

while (editor_state == EDITOR_RUNNING)
    XtAppProcessEvent(XtWidgetToApplicationContext(dialog), XtIMAll);

/*
 * If the user did not cancel the editor return the text and file
 * name
 */

if (editor_state == EDITOR_DONE)
{
    *string = XmTextGetString(query_text);
    if (**string == '\0')
    {
        XtFree(*string);
        *string = NULL;
    }
    *filename = XmTextGetString(filename_text);
}
if (**filename == \0')
{
    XtFree(*filename);
    *filename = NULL;
}
else
{
    *string = NULL;
}

if (XtIsManaged(dialog))
    XtUnmanageChild(dialog);

/*
 * Cut callback that copies the primary selected text to the clipboard and then
 * deletes the primary selected text.
*/
static void
cutCB(Widget widget,
       XtPointer client_data,
       XtPointer call_data)
{
    Widget text = ((FILEDLG_DATA *) client_data)->text;
    if (text != NULL)
    {
        XmTextCut(text,
                   XtLastTimestampProcessed(XtDisplay(text)));
    }
}

/*
 * Copy callback that copies the primary selected text to the clipboard
*/
static void
copyCB(Widget widget,
        XtPointer client_data,
        XtPointer call_data)
{
    Widget text = ((FILEDLG_DATA *) client_data)->text;
    if (text != NULL)
    {
        XmTextCopy(text,
                   XtLastTimestampProcessed(XtDisplay(text)));
    }
}

/*
 * Paste callback that inserts the clipboard selection at the insertion cursor.
*/
static void
pasteCB(Widget widget,
        XtPointer client_data,
        XtPointer call_data)
{
    Widget text = ((FILEDLG_DATA *) client_data)->text;
    if (text != NULL)
    {
        XmTextPaste(text);
    }
}

/*
 * Ok callback closes the editor and returns the text to the calling program.
*/
static void okCB(Widget widget, 
    XtPointer client_data, 
    XtPointer call_data)
{
    char *running = (char *)client_data;
    *running = EDITOR_DONE;
}

/*
 * Cancel callback closes the editor, the text is lost.
 */
static void cancelCB(Widget widget, 
    XtPointer client_data, 
    XtPointer call_data)
{
    char *running = (char *)client_data;
    *running = EDITOR_CANCELED;
}

/*
 * Close the editor but give the user a chance to go back to the editor
 */
static void closeCB(Widget widget, 
    XtPointer client_data, 
    XtPointer call_data)
{
    Widget parent = (Widget) client_data;
    
    if (ModalPromptDialog(parent, "Warning", "Close editor?"))
        return;
    
    if (XtIsManaged(parent))
        XtUnmanageChild(parent);
}

/*
 * Exit the program but give the user a chance to go back to the editor
 */
static void exitCB(Widget widget, 
    XtPointer client_data, 
    XtPointer call_data)
{
    Widget parent = (Widget) client_data;
    
    if (ModalPromptDialog(parent, "Warning", "Exit program?"))
        return;
    
    exit(0);
}

P. SEARCHDIALOG.H

/*
 * $Id: SearchDialog.h,v 1.2 1998/01/18 18:50:09 greg Exp $
 */
* SearchDialog.h -- Software Base Search Interface
* Header file for the search dialog.
* Naval Postgraduate School
* January 13, 1998
* Written by Gregory L. Meckstroth
*/
void
SearchDialog(Widget parent, XtCallbackProc callback);

char *
ModalSearchDialog(Widget parent);

Q. SEARCHDIALOG.C

/*
 * $Id: SearchDialog.C,v 1.6 1998/01/18 18:50:09 greg Exp $
 * SearchDialog.C -- Software Base Search Interface
 * Source code for the CAPS software base search dialog. This dialog is the
 * interface to the ADA software base search that uses profile filtering and
 * signature matching. The dialog facilities the input of a PSDL specification
 * that is used as a query in the search. The output is displayed and the user
 * can select the desired component.
 * Entry points: SearchDialog, ModalSearchDialog.
 * Naval Postgraduate School
 * January 13, 1998
 * Written by Gregory L. Meckstroth
 */

#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>

#include <sys/types.h>
#include <sys/stat.h>

#include <Xm/Xm.h>
#include <Xm/Text.h>
#include <Xm/Label.h>
#include <Xm/ScrolledW.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/ToggleB.h>
#include <Xm/PanedW.h>
#include <Xm.RowColumn.h>
#include <Xm/Separator.h>
#include <Xm/FileSB.h>

#include "Gui.h"
#include "SearchDialog.h"
#include "PromptDialog.h"
#include "Menu.h"
#include "SDE.h"
#include "DisplayProgress.h"
#include "Callbacks.h"
#include "Utils.h"

/*
 * Names for files used as buffers to pass the search output to the interface.
 */
#define PF_FILE_BUFFER "../IO_Profile_Filtering.buf"
#define SM_FILE_BUFFER "../IO_Signature_Matching.buf"

/*
 * A simple linked list to keep track of which file name is associated with
 * which toggle button. This makes it easier to free memory when it is no
 * longer needed.
 */

struct MATCH_LIST
{

char *filename;
Widget tbutton;
MATCH_LIST *next;
);

/*
 * The SEARCH_DATA structure is used to pass data to the callback routines.
 * *parent*: widget to use as the parent for dialog construction.
 * *fname_text*: text widget for file name.display.
 * *prank_text*: text widget of the profile rank.
 * *frank_text*: text widget of the signature rank.
 * *sw_parent*: signature window parent.
 * *callback*: Callback to pass the selected component name to.
 * *matches*: linked list of toggle buttons and file names that the user can
 * choose from.
 */

struct SEARCH_DATA
{
    Widget parent;
    Widget fname_text;
    Widget prank_text;
    Widget frank_text;
    Widget sw_parent;
    XtCallbackProc callback;
    MATCH_LIST *matches;
};

/*
 * The toggle button uses selected_component as the currently selected button.
 * This is also the return value for the ok callback. So to make things less
 * complicated just make it global.
 */

static MATCH_LIST *selected_component = NULL;

/*
 * The ADA code will call profile_filtering_doneCB when the profile filtering
 * is completed. Trying to pass profile_text as an argument complicated the ADA
 * interface so make it global to the search dialog.
 */

static Widget profile_text;

/*
 * Save these to keep the users changes for next time.
 */

static char *save_query_filename = NULL;
static float min_profile_rank = 1.00;
static float min_signature_rank = 1.00;

/*
 * Used by the modal search dialog. done_with_dialog while true the event loop
 * will continue running. Setting this to false allows the event loop to exit.
 * return_value is set by the callback routine to the modal dialog return
 * value.
 */

static int done_with_dialog;
static char *return_value;

/*
 * Declarations for local functions.
 */

static int
remove_nl(char *cptr);

static Widget
fgetlabel(Widget parent, Widget previous, FILE * stream);

static void
clear_search(SEARCH_DATA * sdata);
static void closeCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void exitCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void cancelCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void okCB(Widget widget, XtPointer client_data, XtPointer call_data);
static void togglebuttonCB(Widget widget, XtPointer client_data, XtPointer call_data);
void profile_filtering_doneCB();
static void do_searchCB(Widget widget, XtPointer client_data, XtPointer call_data);

/*
 * Declaration for the "ADA export" search routine.
 */
extern "C" sb_search(char *sbroot,
        char *qfname,
        float *mprank,
        float *msrank,
        char *pf_name,
        char *sm_name);

/*
 * Display the software base search modal dialog. Returns selected component
 * name.
 */
char *
ModalSearchDialog(Widget parent)
{
    done_with_dialog = False;
    SearchDialog(parent, NULL);
    
    /* Wait for a response to the dialog. When the OK or cancel button is
     * pressed done_with_dialog will be set true causing the loop to terminate.
     * Return_value is also set to the appropriate value in the callback.
     */
    done_with_dialog = False;
    while (!done_with_dialog)
        XtAppProcessEvent(XtWidgetToApplicationContext(parent), XtlMAll);
    return return_value;
}

/*
 * Display the software base search dialog.
 */
void
SearchDialog(Widget parent,
            XtCallbackProc callback)
{
    Widget dialog_window;
    Widget scrolled_query_window;
    Widget query_text;
    Widget query_fname_text;
    Widget profile_rank_text;
    Widget scrolled_profile_window;
    Widget profile_fname_text;
    Widget profile_sm_name;
/* The following static variables are used by callbacks after SearchDialog returns. */

static FILEDLG_DATA filedlg_data;       // file callbacks
static SEARCH_DATA search_data;         // Search results

Widget caps_label;
Widget profile_label;
Widget signature_label;
Widget previous;
Widget query_label;

Widget search_btn;
Widget ok_btn;
Widget cancel_btn;

Widget menubar;
Widget querymenu;
Widget searchmenu;
Widget helpmenu;

char IOBuffer[256];
char *title;

Arg args[20];
Cardinal n;
XmString xmstring;

/* Start with no match */

search_data.matches = NULL;
search_data.callback = callback;

/* Create the dialog box. */

title = "Software Base Search";
n = 0;
SETARG(args[n], XmNautoUnmanage, False, n);
SETARG(args[n], XmNwidth, DIALOG_WIDTH, n);
SETARG(args[n], XmNheight, DIALOG_HEIGHT, n);
SETARG(args[n], XmNnoResize, True, n);
SETARG(args[n], XmNtitle, title, n);
SETARG(args[n], XmNiconName, title, n);
dialog_window = XmCreateFormDialog(parent, "sbsdialog", args, n);

/* Save the dialog widget for later use. */

search_data.parent = dialog_window;
filedlg_data.parent = dialog_window;

initialize_display(parent);

/* Create menubar */

n = 0;
SETARG(args[n], XmNtopAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNleftAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNrightAttachment, XmATTACH_FORM, n);
menubar = XmCreateMenuBar(dialog_window, "sbsmenubar", args, n);
previous = menubar;

XtManageChild(menubar);

/*
 * Create query menu. Callbacks for this menu are in Callbacks.C.
 */

querymenu = CreateMenu(menubar, "sbsmenu", "Query", 'Q');
(void)AddMenuItem(querymenu, "new",
"New",
"Ctrl+N", "Ctrl<Key>n", 'N',
(XtCallbackProc) newCB,
(XtPointer) & filedlg_data);

(void)AddMenuItem(querymenu, "open",
"Open",
"Ctrl+O", "Ctrl<Key>o", 'O',
(XtCallbackProc) openCB,
(XtPointer) & filedlg_data);

(void)AddMenuItem(querymenu, "edit",
"Edit",
"Ctrl+E", "Ctrl<Key>e", 'E',
(XtCallbackProc) editCB,
(XtPointer) & filedlg_data);

(void)XtVaCreateManagedWidget("sep", xmSeparatorWidgetClass, querymenu, NULL);

(void)AddMenuItem(querymenu, "save",
"Save",
"Ctrl+S", "Ctrl<Key>s", 'S',
(XtCallbackProc) saveCB,
(XtPointer) & filedlg_data);

(void)AddMenuItem(querymenu, "saveas",
"Save As",
"Ctrl+A", "Ctrl<Key>a", 'A',
(XtCallbackProc) saveasCB,
(XtPointer) & filedlg_data);

(void)XtVaCreateManagedWidget("sep", xmSeparatorWidgetClass, querymenu, NULL);

(void)AddMenuItem(querymenu, "close",
"Close",
"Ctrl+W", "Ctrl<Key>w", 'C',
(XtCallbackProc) closeCB,
(XtPointer) & search_data);

(void)AddMenuItem(querymenu, "exit",
"Exit",
"Ctrl+Q", "Ctrl<Key>Q", 'x',
(XtCallbackProc) exitCB,
(XtPointer) NULL);

/*
 * Create the search menu. Callbacks are implemented locally
 */

searchmenu = CreateMenu(menubar, "sbsmenu", "Search", 'S');
(void)AddMenuItem(searchmenu, "search",
"Start",
"Ctrl+T", "Ctrl<Key>t", 't',
(XtCallbackProc) do_searchCB,
(XtPointer) & search_data);

/*
 * Create Help menu.
 */
helpmenu = CreateHelpMenu(menubar, "sbsmenu", "Help", 'H');

xmstring = XmStringCreateSimple("CAPS Software Base Search");
caps_label = XtVaCreateManagedWidget("sbslabel",
  XmLabelWidgetClass, dialog_window,
  XmNlabelString, xmstring,
  XmNtopAttachment, XmATTACH_WIDGET,
  XmNtopWidget, previous,
  XmNtopOffset, 5,
  XmNbottomAttachment, XmATTACH_NONE,
  XmNleftAttachment, XmATTACH_NONE,
  XmNleftOffset, 0,
  XmNrightAttachment, XmATTACH_NONE,
  XmNrightOffset, 0,
  NULL);
XmStringFree(xmstring) ;

/ * Query_label and query_fname_text display the current query file name for
* the user
*/

previous = caps_label;

xmstring = XmStringCreateSimple("FSDL Query:");
query_label = XtVaCreateManagedWidget("sbslabel",
  xmLabelWidgetClass, dialog_window,
  XmNlabelString, xmstring,
  XmNtopAttachment, XmATTACH_WIDGET,
  XmNtopWidget, previous,
  XmNtopOffset, 5,
  XmNleftAttachment, XmATTACH_WIDGET,
  XmNleftOffset, 5,
  XmNrightAttachment, XmATTACH_NONE,
  XmNrightOffset, 10,
  NULL);
XmStringFree(xmstring);

query_fname_text = XtVaCreateManagedWidget("sbstext",
  xmTextWidgetClass, dialog_window,
  XmNeditMode, XmSINGLE_LINE_EDIT,
  XmNeditable, True,
  XmNshadowThickness, 1,
  XmNmarginHeight, 2,
  XmNcursorPositionVisible, True,
  XmNtopAttachment, XmATTACH_WIDGET,
  XmNtopWidget, previous,
  XmNtopOffset, 5,
  XmNleftAttachment, XmATTACH_WIDGET,
  XmNleftWidget, query_label,
  XmNleftOffset, 5,
  XmNrightAttachment, XmATTACH_NONE,
  XmNrightOffset, 10,
  NULL);

/ * Save the widget so that the file and search callbacks can change the name
* when the user enters a new one.
*/

tfidf_data.fname_text = query_fname_text;
search_data.fname_text = query_fname_text;

/ * This callback will update the file name and query text if the user types
* a return in the fname text box.
*/

 XtAddCallback(query_fname_text, XmNactivateCallback,
   (XtCallbackProc) textchangedCB,
   (XtPointer) & tfidf_data);

previous = query_label;

XtManageChild(query_fname_text);

/*
* Setup scrolled window to display query text.
* /

scrolled_query_window = XtVaCreateManagedWidget("sbswindow",
    XmScrolledWindowWidgetClass, dialog_window,
    XmNwidth, INFO_WINDOW_WIDTH,
    XmNheight, INFO_WINDOW_HEIGHT,
    XmNleftAttachment, XmATTACH_FORM,
    XmNrightAttachment, XmATTACH_FORM,
    XmNtopAttachment, XmATTACH_WIDGET,
    XmNtopWidget, query_label,
    XmNtopOffset, 5,
    XmNbottomAttachment, XmATTACH_NONE,
    XmNresizeable, True,
    XmNleftOffset, 5,
    XmNrightOffset, 5,
    XmNbottomOffset, 5,
    XmNscrollingPolicy, XmAUTOMATIC,
    XmNscrollBarDisplayPolicy, XmSTATIC,
    XmNscrolledWindowMarginWidth, 5,
    XmNscrolledWindowMarginHeight, 5,
    XmNscrollVertical, False,
    XmNscrollHorizontal, True,
    NULL);

previous = scrolled_query_window;

query_text = XtVaCreateManagedWidget("sbstext",
    xmTextWidgetClass, scrolled_query_window,
    XmNwidth, INFO_WINDOW_WIDTH,
    XmNheight, INFO_WINDOW_HEIGHT,
    XmNeditMode, XmMULTI_LINE_EDIT,
    XmNshadowThickness, 0,
    XmNmarginHeight, 0,
    XmNscrollHorizontal, False,
    XmNeditable, False,
    XmNtraversalOn, False,
    XmNcursorPositionVisible, False,
    NULL);

/*
* Save the widget so that the query text can be updated when the user
* changes the file name or edits that text
*/

filedlg_data.text = query_text;

XmScrolledWindowSetAreas(scrolled_query_window,
    (Widget) NULL,
    (Widget) NULL,
    query_text);

XtManageChild(query_text);

/*
* If the user is restarting the dialog try to use the file name that was
* entered
*/

if (save_query_filename != NULL)
    {
    char *text;
    XmTextSetString(query_fname_text, save_query_filename);
    if (LoadFile(save_query_filename, &text) != -1 && text != NULL)
        {
            XmTextSetString(query_text, text);
            XtFree(text);
        }
    XtFree(save_query_filename);
    }
*/
Create a text box for the profile rank. This is where the rank is stored so that the user can make changes before running the search.

```
xmstring = XmStringCreateSimple("Profile Filtering Minimum Rank: ");
profile_label = XtVaCreateManagedWidget("sbslabel",
    xmLabelWidgetClass, dialog_window,
    XmNlabelString, xmstring,
    XmNtopAttachment, XmATTACH_WIDGET,
    XmNtopWidget, previous,
    XmNtopOffset, 5,
    XmNleftAttachment, XmATTACH_POSITION,
    XmNleftPosition, 15,
    NULL);
XmStringFree(xmstring);
```

```
sprintf(IOBuffer, ".2f", min_profile_rank);
profile_rank_text = XtVaCreateManagedWidget("sbstext",
    xmTextWidgetClass, dialog_window,
    XmNeditMode, XmSINGLE_LINE_EDIT,
    XmNeditable, True,
    XmNshadowThickness, 1,
    XmNmarginHeight, 2,
    XmNcolumns, 5,
    XmNtopAttachment, XmATTACH_WIDGET,
    XmNtopWidget, previous,
    XmNtopOffset, 5,
    XmNleftAttachment, XmATTACH_WIDGET,
    XmNleftWidget, profile_label,
    XmNleftOffset, 5,
    NULL);
```

```
* Save the widget so that the search callbacks can get the profile rank when the user runs the search.

search_data.prank_text = profile_rank_text;
```

```
previous = profile_label;
XmTextSetString(profile_rank_text, IOBuffer);
XtManageChild(profile_rank_text);
```

```
* Create a scrolled window for the profile filtering results.

scrolled_profile_window = XtVaCreateManagedWidget("sbswindow",
    xsScrolledWindowWidgetClass, dialog_window,
    XmNwidth, INFO_WINDOW_WIDTH,
    XmNheight, INFO_WINDOW_HEIGHT,
    XmNleftAttachment, XmATTACH_FORM,
    XmNrightAttachment, XmATTACH_FORM,
    XmNtopAttachment, XmATTACH_WIDGET,
    XmNtopWidget, profile_label,
    XmNtopOffset, 5,
    XmNleftAttachment, XmATTACH_NONE,
    XmNscrollPolicy, XmAUTOMATIC,
    XmNscrollBarDisplayPolicy, XmSTATIC,
    XmNscrollWindowMarginWidth, 5,
    XmNscrollWindowMarginHeight, 5,
    XmNscrollVertical, False,
    XmNscrollHorizontal, True,
    NULL);
previous = scrolled_profile_window;
```

```
profile_text = XtVaCreateManagedWidget("sbstext", 87
```
Create a text box for the signature rank. This is where the rank is stored so that the user can make changes before running the search.

/*
 * Create a text box for the signature rank. This is where the rank is stored so that the user can make changes before running the search.
 */

xmstring = XmStringCreateSimple("Signature Matching Minimum Rank: ");
signature_label = XtVaCreateManagedWidget("sbslabel",
xmLabelWidgetClass, dialog_window,
xmLabelString, xmstring,
xmNtopAttachment, XmATTACH_WIDGET,
xmNtopWidget, previous,
xmNtopOffset, 5,
xmNleftAttachment, XmATTACH_WIDGET,
xmNleftWidget, signature_label,
xmNleftOffset, 5,
NULL);
XmStringFree(xmstring);

sprintf(IOBuffer, "%.2f", min_signature_rank);

signature_rank_text = XtVaCreateManagedWidget("sbstext",
xmTextWidgetClass, dialog_window,
xmNeditMode, XmSINGLE_LINE_EDIT,
xmNeditable, True,
xmNshadowThickness, 1,
xmNmarginHeight, 2,
xmNcolumns, 5,
xmNcursorPositionVisible, True,
xmNtopAttachment, XmATTACH_WIDGET,
xmNtopWidget, previous,
xmNtopOffset, 5,
xmNleftAttachment, XmATTACH_WIDGET,
xmNleftWidget, signature_label,
xmNleftOffset, 5,
NULL);

/*
 * Save the widget so that the search callbacks can get the signature rank when the user runs the search.
 */

search_data.srank_text = signature_rank_text;
previous = signature_label;

XmTextSetString(signature_rank_text, IOBuffer);
XtManageChild(signature_rank_text);

/*
 * Create a scrolled window for the profile filtering results.
 */

scrolled_signature_window = XtVaCreateManagedWidget("sbswindow",
xmScrolledWindowWidgetClass, dialog_window,
xmNwidth, INFO_WINDOW_WIDTH,
xmNheight, INFO_WINDOW_HEIGHT,
xmNleftAttachment, XmATTACH_FORM,
xmNshadowThickness, 0,
xmNmarginHeight, 0,
xmNscrollHorizontal, False,
xmNeditable, False,
xmNtraversalOn, False,
xmNcursorPositionVisible, False,
NULL);
XmNrightAttachment, XmATTACH_FORM,
XmNtopAttachment, XmATTACH_WIDGET,
XmNtopOffset, 5,
XmNbottomAttachment, XmATTACH_NONE,
XmNresizable, True,
XmNleftOffset, 5,
XmNrightOffset, 5,
XmNbottomOffset, 5,
XmNscrollingPolicy, XmAUTOMATIC,
XmNscrollBarDisplayPolicy, XmSTATIC,
XmNscrolledWindowMarginWidth, 5,
XmNscrolledWindowMarginHeight, 5,
XmNscrollVertical, False,
XmNscrollHorizontal, True,
(NULL);

/*
 * Save the widget so that the search callbacks can create signature match
 * results widgets in it.
 */

search_data.sw_parent = scrolled_signature_window;
previous = scrolled_signature_window;
signature_window = XtVaCreateManagedWidget("sbsform",
xmFormWidgetClass, scrolled_signature_window,
XmNwidth, INFO_WINDOW_WIDTH,
XmNheight, INFO_WINDOW_HEIGHT,
(NULL);
XmScrolledWindowSetAreas(scrolled_signature_window,
(NULL),
(NULL),
signature_window);

/*
 * Create the push buttons at the bottom of the dialog box.
 */

xmstring = XmStringCreateSimple("OK");
ok_btn = XtVaCreateManagedWidget("sbslabel",
xmPushButtonWidgetClass, dialog_window,
XmNlabelString, xmstring,
XmNwidth, BUTTON_WIDTH,
XmNheight, BUTTON_HEIGHT,
XmNleftAttachment, XmATTACH_FORM,
XmNleftOffset, 20,
XmNrightAttachment, XmATTACH_NONE,
XmNtopAttachment, XmATTACH_NONE,
XmNbottomAttachment, XmATTACH_NONE,
XmNbottomOffset, 10,
(NULL);
XmStringFree(xmstring);
XtAddCallback(ok_btn,
XmNactivateCallback,
(XtCallbackProc) okCB,
(XtPointer) & search_data);
XtManageChild(ok_btn);

xmstring = XmStringCreateSimple("Search");
search_btn = XtVaCreateManagedWidget("sbslabel",
xmPushButtonWidgetClass, dialog_window,
XmNlabelString, xmstring,
XmNwidth, BUTTON_WIDTH,
XmNheight, BUTTON_HEIGHT,
XmNleftAttachment, XmATTACH_WIDGET,
XmNleftWidget, ok_btn,
XmNleftOffset, 10,
XmNrightAttachment, XmATTACH_NONE,
XmNtopAttachment, XmATTACH_NONE,
XmNbottomAttachment, XmATTACH_FORM,
XmStringFree(xmstring);
XtAddCallback(search_btn,
XmNactivateCallback,
(XtCallbackProc) do_searchCB,
(XtPointer) & search_data);
XtManageChild(search_btn);

xmstring = XmStringCreateSimple("Cancel");
cancel_btn = XtVaCreateManagedWidget("sbslabel",
  XmPushButtonWidgetClass, dialog_window,
  XmNlabelString, xmstring,
  XmNwidth, BUTTON_WIDTH,
  XmNheight, BUTTON_HEIGHT,
  XmNleftAttachment, XmATTACH_WIDGET,
  XmNleftWidget, search_btn,
  XmNleftOffset, 10,
  XmNrightAttachment, XmATTACH_NONE,
  XmNstopAttachment, XmATTACH_NONE,
  XmNbottomAttachment, XmATTACH_FORM,
  XmNbottomOffset, 10,
  NULL);
XmStringFree(xmstring);
XtAddCallback(cancel_btn,
XmNactivateCallback,
(XtCallbackProc) cancelCB,
(XtPointer) & search_data);
XtManageChild(cancel_btn);

XtManageChild(dialog_window);

/*
 * Find new line and null it out. Returns length of string.
 */
static int
remove_nl(char *cptr)
{
  char *first = cptr;
  while (*cptr != '\n')
  {
    if (*cptr == ' ') break;
    cptr++;
  }
  *cptr = '\0';
  return cptr - first;
}

/*
 * Create a label from a line of text that is read from the input stream. The
 * label will be placed in the parent widget after the previous widget. Leading
 * spaces will be skipped.
 */
static Widget
fgetlabel(Widget parent, Widget previous, FILE * stream)
{
  Widget widget;
  Xmstring xmstring;
  char IObuf[MAXBUFSIZE];
  char *ptr = IObuf;
  fgets(IObuf, MAXBUFSIZE, stream);
  remove_nl(IObuf);
  while (*ptr == ' ')
    ptr++;

xmstring = XmStringCreateSimple(ptr);
widget = XtVaCreateManagedWidget("sbstext",
xmLabelWidgetClass, parent,
xmLabelString, xmstring,
xmNtopAttachment, XmATTACH_WIDGET,
xmNtopWidget, previous,
xmNtopOffset, 5,
xmNleftAttachment, XmATTACH_FORM,
xmNleftOffset, 25,
NULL);
XtManageChild(widget);
XmStringFree(xmstring);
return widget;

/*
 * Clear the search results displays and free memory used by data structures.
 */

static void
clear_search(SEARCH_DATA * sdata)
{
  MATCH_LIST *match;
  MATCH_LIST *next_match;
  if (sdata->matches == NULL)
    return;
  XmTextSetString(profile_text, "") ;
  for (match = sdata->matches; match != NULL; match = next_match)
    {
      next_match = match->next;
      XtFree(match->filename);
      XtFree((char *)match);
    }
}

/*
 * Callback quits program without returning a component selection.
 */

static void
closeCB(Widget widget,
         XtPointer client_data,
         XtPointer call_data)
{
  SEARCH_DATA *data = (SEARCH_DATA *) client_data;
  char *warning = "A component was selected\nexit without saving?";
  if (selected_component != NULL && ModalPromptDialog(data->parent, "Warning", warning))
    return;
  cancelCB(widget, client_data, call_data);
}

/*
 * Callback exits program.
 */

static void
exitCB(Widget widget,
       XtPointer client_data,
       XtPointer call_data)
{
  exit(0);
}

/*
 * Cancel callback
 */

static void
cancelCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    SEARCH_DATA *data = (SEARCH_DATA *) client_data;
    /*
    * Save the query filename in case the used restarts the search dialog.
    */
    save_query_filename = gettext(data->fname_text);
    if (XtIsManaged(data->parent))
        XtUnmanageChild(data->parent);
    /*
    * The modal search dialog set the call back to NULL. So set its return
    * value
    */
    if (data->callback != NULL)
        (data->callback)(widget, (XtPointer) NULL, (XtPointer) NULL);
    else
        return_value = NULL;
    done_with_dialog = True;
    clear_search(data);
}

/*
* Ok callback passes the selected component to the applications callback then
* exits the dialog.
*/
static void
okCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    char *warning = "No module selected\nexit anyway?";
    SEARCH_DATA *data = (SEARCH_DATA *) client_data;
    /*
    * If the user did not select a component give them a chance to return to
    * the dialog before exiting.
    */
    if (selected_component == NULL)
    {
        if (ModalPromptDialog(data->parent, "Warning", warning))
            return;
    }
    else
    {
        cancelCB(widget, client_data, call_data);
        return;
    }
    /*
    * Save the query filename in case the used restarts the search dialog.
    */
    save_query_filename = gettext(data->fname_text);
    if (XtIsManaged(data->parent))
        XtUnmanageChild(data->parent);
    /*
    * Pass the component to the callback or if modal set the return value.
    */
if (data->callback != NULL)
    (data->callback) (widget, (XtPointer) selected_component->filename, (XtPointer) NULL);
else
    return_value = selected_component->filename;

done_with_dialog = True;

clear_search(data);
)

/*
* Toggle button callback turns off previously selected button and saves new
* component.
*/

callback void
togglebuttonCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    if (selected_component)
    {
        XtVaSetValues(selected_component->tbutton,
            XmNset, XmUNSET,
            NULL);
    }
    selected_component = (MATCH_LIST *) client_data;
}

/*
* Show the user the results of the profile filtering part of the search. This
* callback is made from the ADA search routine.
*/
callback void
profile_filtering_doneCB()
{
    char *string;
    
    /*
    * PF_FILE_BUFFER is the temporary file used to pass the results from the
    * ADA search routines to the GUI interface.
    */
    if (LoadFile(PF_FILE_BUFFER, &string) > 0)
    {
        XmTextSetString(profile_text, string);
        XtFree(string);
        remove(PF_FILE_BUFFER);
    }
    else
        XmTextSetString(profile_text, "Profile filtering ERROR!.");
}

/*
* Callback searches the software base for matches to the user specified query.
*/
callback void
do_searchCB(Widget widget,
    XtPointer client_data,
    XtPointer call_data)
{
    SEARCH_DATA *data = (SEARCH_DATA *) client_data;
    
    Widget signature_text;
    Widget candidates;
    Widget previous_widget;
    Widget component_id;
    Widget solution;
    XmString xmstring;
    Cardinal n;

Arg args[20];
FILE *fp;
char *soltext;
char *cptr;
char PTbuffer[MAXBUFSIZE];
char IObuffer[MAXBUFSIZE];
char *string;
char *filename;
int length;
int candidates_offset = 5;
int nmatches;
int position;
MATCH_LIST *match;

static Widget signature_window = NULL;

if (client_data == NULL)
{
    cout << "Error no data\n";
    return;
}

/*
 * We have to pass a query file to the ADA search routines.
 */
filename = gettext(data->fname_text);
if (filename == NULL)
{
    ModalWarningDialog(data->parent, "Error", "Query not specified");
    return;
}

/*
 * Clear any old search results.
 */
clear_search(data);

/*
 * If the signature window exists destroy it to clear out the old search
 * results.
 */
if (signature_window != NULL && XtIsManaged(signature_window))
{
    XtUnmanageChild(signature_window);
    XtDestroyWidget(signature_window);
}

signature_window = XmVaCreateManagedWidget("sbsform",
xmFormWidgetClass, data->sw_parent,
    NULL);
XmScrolledWindowSetAreas(data->sw_parent,
    (Widget) NULL,
    (Widget) NULL,
    signature_window);

/*
 * Get the current minimum rank from their text widgets.
 */
string = XmTextGetString(data->prank_text);
min_profile_rank = atof(string);
XtFree(string);

string = XmTextGetString(data->srank_text);
min_signature_rank = atof(string);
XtFree(string);

/*
 * Clear the progress display and set the title. The display will be
 * updated by the ADA search routines as the search progresses.
 */
clear_display("Search Progress");

sb_search(SBROOT,
filename,
&min_profile_rank,
&min_signature_rank,
PF_FILE_BUFFER,
SM_FILE_BUFFER);

XtFree(filename);

selected_component = NULL;

data->matches = NULL;
match = NULL;

/*
Assume that an error occurred if we can't open the results buffer.
*/

fp = fopen(SM_FILE_BUFFER, "r");
if (fp == NULL)
{
    xmstring = XmStringCreateSimple("Signature matching ERROR!");
    (void)XtVaCreateManagedWidget("sbstext",
        xmLabelWidgetClass, signature_window,
        XmNlabelString, xmstring,
        XmNuseState, XmUNSET,
        XmNtopAttachment, XmATTACH_FORM,
        XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5,
        NULL);
    XmStringFree(xmstring);
    XtManageChild(signature_window);
}

/*
We remove the new lines from labels so that they will not cause problems.
The first line has information about the candidates found.
*/

fgets(IObuffer, MAXBUFSIZE, fp);
remove_nl(IObuffer);

xmstring = XmStringCreateSimple(IObuffer);
candidates = XtVaCreateManagedWidget("sbstext",
        xmLabelWidgetClass, signature_window,
        XmNlabelString, xmstring,
        XmNuseState, XmUNSET,
        XmNtopAttachment, XmATTACH_FORM,
        XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5,
        NULL);
XmStringFree(xmstring);

previous_widget = candidates;

XtManageChild(candidates);

/*
Loop through the components that were found display informaiton about
each one and set up a toggle button for the use to select one.
*/

while (fgets(IObuffer, MAXBUFSIZE, fp))
{
    /*
    * Build the data structure that will have the toggle buttons and file
    * names.
    */
    if (data->matches == NULL)
{  
data->matches = (MATCH_LIST *) XtMalloc((Cardinal) sizeof(MATCH_LIST));  
match = data->matches;  
}  
else  
{  
match->next = (MATCH_LIST *) XtMalloc((Cardinal) sizeof(MATCH_LIST));  
match = match->next;  
}  
match->next = NULL;  
/*  
* Add the file name of the component  
*/  
length = remove_nl(IObuffer);  
match->filename = (char *)XtMalloc((Cardinal) (length + 1));  
strcpy(match->filename, IObuffer);  
/*  
* Add the toggle button  
*/  
xmstring = XmStringCreateSimple(IObuffer);  
match->button = XtVaCreateManagedWidget("sbslabel",  
XmToggleButtonWidgetClass, signature_window,  
XmNlabelString, xmstring,  
XmNtopAttachment, XmATTACH_WIDGET,  
XmNtopWidget, previous_widget,  
XmNtopOffset, candidates_offset,  
XmNleftAttachment, XmATTACH_FORM,  
XmNleftOffset, 5,  
NULL);  
previous_widget = match->button;  
XmStringFree(xmstring);  
candidates_offset = 20;  
XtAddCallback(match->button,  
XmNarmCallback,  
(XtCallbackProc) togglebuttonCB,  
(XtPointer) match);  
XtManageChild(match->button);  
/*  
* Show the user the component information  
*/  
component_id = fgetlabel(signature_window, previous_widget, fp);  
previous_widget = component_id;  
profile_rank = fgetlabel(signature_window, previous_widget, fp);  
previous_widget = profile_rank;  
nmatchs = 1;  
/*  
* The ADA routines put in end of sections, %%End_Signature%% and  
* %%End_Component%%, that are used to control the parsing of the  
* results. If we find an end of component then their are not signature  
* matches for this candidate.  
*/  
fgets(IObuffer, MAXBUFSIZE, fp);  
continue;  
/*  
* Loop through the solutions for this component.  
*/  
do  
{  
/*  
*/
96
Create a label for this solution

```c
sprintf(PTbuffer, "Solution %d", nmatches++);

xmstring = XmStringCreateSimple(PTbuffer);

solution = XtVaCreateManagedWidget("sbstext",
    xmlLabelWidgetClass, signature_window,
    XmNlabelString, xmstring,
    XmNtopAttachment, XmATTACH_WIDGET,
    XmNtopWidget, previous_widget,
    XmNtopOffset, 5,
    XmNleftAttachment, XmATTACH_FORM,
    XmNleftOffset, 45,
    NULL);

XtManageChild(solution);

XmStringFree(xmstring);

previous_widget = solution;
```

Save the text of the solutions.

```c
position = strlen(IObuffer);

length = 1;
soltext = (char *)XtMalloc((Cardinal) (position + 1));

strcpy(soltext, IObuffer);

while (fgets(IObuffer, MAXBUF SIZE, fp))
{
        break;

    cptr = soltext;
    position += strlen(IObuffer);
    soltext = (char *)XtMalloc((Cardinal) (position + 1));
    strcpy(soltext, cptr);
    strcat(soltext, IObuffer);

    XmTextSetString(signature_text, soltext);
    XtFree(soltext);

    length++;
}
```

Create a scrolled text widget for the solution it may be quite long.

```c
n = 0;

SETARG(args[n], XmNeditMode, XmMULTI_LINE_EDIT, n);
SETARG(args[n], XmNeditable, False, n);
SETARG(args[n], XmNwidth, INFO_WINDOW_WIDTH - 25, n);
SETARG(args[n], XmNheight, INFO_WINDOW_HEIGHT / 2, n);
SETARG(args[n], XmNcursorPositionVisible, False, n);
SETARG(args[n], XmNscrollPolicy, XmAUTOMATIC, n);
SETARG(args[n], XmNscrollBarDisplayPolicy, XmSTATIC, n);
SETARG(args[n], XmNscrolledWindowMarginWidth, 5, n);
SETARG(args[n], XmNscrolledWindowMarginHeight, 5, n);
SETARG(args[n], XmNtopAttachment, XmATTACH_WIDGET, n);
SETARG(args[n], XmNtopWidget, previous_widget, n);
SETARG(args[n], XmNtopOffset, 5, n);
SETARG(args[n], XmNleftAttachment, XmATTACH_FORM, n);
SETARG(args[n], XmNleftOffset, 45, n);

signature_text = XmCreateScrolledText(signature_window, "sbstext",
    args, n);

previous_widget = signature_text;

XmTextSetString(signature_text, soltext);
XtFree(soltext);
```
```c
XtManageChild(signature_text);
fgets(IObuffer, MAXBUFFSIZE, fp);
} while (IObuffer[6] == 'S');
}
fclose(fp);
remove(SM_FILE_BUFFER);
XtManageChild(signature_window);
}

R. UTILS.H

/*
 * $Id: Utilities.h,v 1.3 1998/01/18 16:35:46 greg Exp $
 * Utilities.h -- Software Base Search Interface
 * Header file for the interface utilities.
 * Naval Postgraduate School
 * January 13, 1998
 * Written by Gregory L. Meckstroth
 */

char *
getdirname(char *path);

char *
getfilename(char *path);

char *
catstrings(char *stl, char *st2);

void
Set_Res_String(Widget widget, String name, char *text);

char *
gettext(Widget string);

void
savetext(Widget parent, char *filename, char *text);

long
LoadFile(char *filename, char **ptraddr);

Boolean
CopyFile(char *filein, char *fileout);

void
Reformat(const char *fname, const int nskipcommas);

S. UTILS.C

/*
 * Utilities.C -- Software Base Search Interface
 * Source code for the common utilities used in the software base search
 * interface.
 * Entry points: getdirname, getfilename, catstrings, Set_Res_String, gettext,
 * savetext, LoadFile, CopyFile, Reformat,
 *
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <sys/stat.h>
#include <Xm/Xm.h>
#include <Xm/Text.h>
#include "PromptDialog.h"

/*
 * Returns a character pointer to a copy of the directory part of the path to a
 * file. The application is responsible for freeing the storage associated with
 * the string by calling XtFree.
 */
char *
getdirname(char *path)
{
    char *cptr = strrchr(path, '/');
    char *rv;
    Cardinal length;

    /*
     * If the path did not have a separator its local so return a pointer to
     * the local directory. Other wise make a copy of the path up to and
     * including the separator.
     */
    if (cptr == NULL)
    {
        rv = XtMalloc((Cardinal) 3);
        strcpy(rv, "./");
    } else
    {
        length = cptr - path + 1;
        rv = XtMalloc(length + 1);
        strncpy(rv, path, length);
        *(rv + length) = '\0';
    }

    return rv;
}

/*
 * Returns a character pointer to a copy of the file name part of the path to a
 * file. The application is responsible for freeing the storage associated with
 * the string by calling XtFree.
 */
char *
getfilename(char *path)
{
    char *cptr = strrchr(path, '/');
    char *rv;
    int length;

    /*
     * If the path did not have a separator then its the file name. Other wise
     * step past it.
     */
    if (cptr == NULL)
    {
        cptr = path;
    }

    return rv;
}
else
{
    cptr++;
}

/*
 * Make a copy of the file name and return it pointer.
 */

    length = strlen(cptr);
    rv = XtMalloc(length + 1);
    strcpy(rv, cptr);
    return rv;
}

/*
 * Returns a character pointer to a copy of the concatenation of the stl and
 * st2. The application is responsible for freeing the storage associated with
 * the string by calling XtFree.
 */
char *
catstrings(char *st1, char *st2)
{
    unsigned int length = strlen(st1) + strlen(st2);
    char *rv = XtMalloc(length + 1);
    char *cptr = rv;

    while (*st1 != '\0')
        *cptr++ = *st1++;

    while (*st2 != '\0')
        *cptr++ = *st2++;

    *cptr = '\0';
    return rv;
}

/*
 * Set a resource string of a widget
 */

void
Set_Res_String(Widget widget,          // Widget containing resource.
                String name,            // Motif name of widget.
                char *text)            // Text to set resource to.
{
    XmString tmp = XmStringCreateLtoR(text, XmSTRING_DEFAULT_CHARSET);
    Arg args[1] =
    {
        {name, (XtArgVal) 0}
    };

    args[0].value = (XtArgVal) tmp;
    XtSetValues(widget, args, 1);

    XmStringFree(tmp);
}

/*
 * Returns a character pointer to the string value of the text widget or NULL
 * if its empty. The application is responsible for freeing the storage
 * associated with the string by calling XtFree.
 */
char *
gettext(Widget text)
{
    if (text == NULL)
        return NULL;
    char *string = XmTextGetString(text);

    return string;
}
if (*string == '\0')
{
XtFree(string);
string = NULL;
}
return string;

void
savetext(Widget parent, char *filename, char *text)
{
FILE *fp;
size_t length;
size_t written;
fp = fopen(filename, "w");
/*
 * If the open failed prompt the user so they will know
 */
if (fp == NULL)
{
char IObuffer[256];
sprintf(IObuffer, "Error opening file %s", filename);
ModalWarningDialog(parent, "Error", IObuffer);
return;
}
/*
 * Write the text to the file
 */
length = strlen(text);
written = fwrite(text, 1, length, fp);
fclose(fp);
/*
 * If their was an error writing the file prompt the user so they will know
 */
if (written != length)
{
char IObuffer[256];
sprintf(IObuffer, "Error writing file %s", filename);
ModalWarningDialog(parent, "Error", IObuffer);
}

long
LoadFile(char *filename, char **string)
{
FILE *fp;
struct stat file_info;
char *buffer;
long bytes_read;
Open the file and return if there is an error.
```
fp = fopen(filename, "r");
if (fp == NULL)
{
    *string = NULL;
    return -1;
}
```

Get file size. Return if there is an error.
```
if (stat(filename, &file_info) != 0)
{
    *string = NULL;
    fclose(fp);
    return -1;
}
```

Get memory to hold the file's text string. Return if there is an error.
```
*string = (char *)XtMalloc(file_info.st_size + 1);
buffer = *string;
if (buffer == (char *)NULL)
{
    fclose(fp);
    return -1;
}
```

Read in file. Return if there is an error writing the text.
```
bytes_read = fread(buffer, 1, file_info.st_size, fp);
fclose(fp);
if (bytes_read < file_info.st_size)
{
    XtFree(buffer);
    *string = NULL;
    return -1;
}
buffer[file_info.st_size] = '\0';  // Terminate string.
return bytes_read;
```

Copy filein to fileout. Returns true if the copy was successful and fails otherwise.
```
Boolean CopyFile(char *filein, char *fileout)
{
    FILE *fp;
    struct stat file_info;
    char *buffer;
    long bytes_read;

    // Get input file size. Return if we can't stat file.
    if (stat(filein, &file_info) != 0)
{ return False;
}

/*
 * Open the input file. Return if we can't open the file for reading.
 */
fp = fopen(filein, "r");
if (fp == NULL)
 return False;

/*
 * Allocate memory to hold file contents. Return if there was an error.
 */
buffer = (char *)XtMalloc(file_info.st_size + 1);
if (buffer == (char *)NULL)
{
fclose(fp);
 return False;
}

/*
 * Read in the input file's contents. Return if there was an error reading
 * the file.
 */
bytes_read = fread(buffer, 1, file_info.st_size, fp);
fclose(fp);
if (bytes_read < file_info.st_size)
{
 XtFree(buffer);
 return False;
}

/*
 * Open the output file. Return if we can't open the file for writing.
 */
fp = fopen(fileout, "w");
if (fp == NULL)
 return False;

/*
 * Write the contents of the input file to the output file. Return if there
 * was an error writing the file.
 */
bytes_read = fwrite(buffer, 1, file_info.st_size, fp);
fclose(fp);
if (bytes_read < file_info.st_size)
{
 XtFree(buffer);
 return False;
}

return True;
}

/*
 * Reformat the ADA search/maintenance output so we can display it in the
 * progress dialog. The ADA has very long lines that needed to be folded. The
 * profile filtering is slightly different from the signature matching. So we
 * use the skip commas to distinguish between the types.
 */
void
Reformat(const char *fname, const int nskipcommas)
FILE *fpin;
FILE *fpout;
char *ftemp;
int brace;
char tab[256];
char *ptab;
char c;
int commas = 0;

/* Return if the calling program gave us a null file name. */
if (fname == NULL)
    return;

/* We create a temporary file by adding .tmp to the input file name. Then
rename the input file so we can reformat into the correct file. */
ftemp = (char *)XtMalloc((Cardinal) (strlen(fname) + 5));
strcpy(ftemp, fname);
strcat(ftemp, " .tmp");
if (rename(fname, ftemp) == -1)
    return;
brace = False;
ptab = tab;
fpin = fopen(ftemp, "r");
fpout = fopen(fname, "w");

/* Try to fold the line after a comma so that the braces line up */
while ((c = fgetc(fpin)) != EOF)
{
    if (!brace)
    {
        /* If we have not found a brace look for one. */
        switch (c)
        {
        /* brace found start looking for end of line. */
        case '{':
            case '{':
                brace = True;
                break;
        /* If we find a end of line before brace restart tab string */
        case '\n':
            ptab = tab;
            *ptab = '\0';
            break;
        /* Other wise add a space for each character. */
        default:
            *ptab++ = ' ';
            *ptab = '\0';
            break;
        }
fclose(fpin);
fclose(fpout);
remove(ftemp);
XtFree(ftemp);
}

fputc(c, fpout);
)
else
{
  /* brace found now look for the end of the line. */
  fputc(c, fpout);
  switch (c)
  {
    /*
    * If we find a comma put rest of text on next line after
    * tabbing over to line up text with brace. We may need to skip
    * so of the commas first
    */
    case ',':
      commas++;
      if (commas > nskipcommas)
      {
        commas = 0;
        fputc('n', fpout);
        fputs(tab, fpout);
      }
      break;
    /*
    * Found a new line so start looking for the first brace on the
    * next line. Also reset the amount to tab over.
    */
    case 'n':
      brace = False;
      ptab = tab;
      *ptab = '\0';
      commas = 0;
      break;
      default:
        break;
    }
  }
  /* We are done reformating so cleanup */
  fclose(fpin);
  fclose(fpout);
  remove(ftemp);
  XtFree(ftemp);
}
APPENDIX B GNAT GENERATED C SOURCE CODE

Source code generated by the GNAT ADA compiler to initialize the environment.

A. B_ADA_SYSTEM.H

/*
 * $Id: b_ada_system.h,v 1.1.1.1 1998/01/14 03:40:05 greg Exp $
 * b_ada_system.h -- Software Base Search Interface
 * Header file for the ADA systems initialization routine.
 * Naval Postgraduate School
 * January 13, 1998
 * Written by Gregory L. Meckstroth
 */

extern "C" b_ada_init(int argc, char **argv, char **envp);
extern "C" b_ada_final(void);

B. B_ADA_SYSTEM.C

/*
 * $Id: b_ada_system.c,v 1.2 1998/01/18 17:40:49 greg Exp $
 * b_ada_system.c -- Software Base Search Interface
 * Source code for the ADA systems initialization. This was generated by the
 * GNAT Compiler.
 * Naval Postgraduate School
 * January 13, 1998
 * Written by Gregory L. Meckstroth
 */

extern int gnat_argc;
extern char **gnat_argv;
extern char **gnat_envp;
extern int gnat_exit_status;
void adafinal();
void adainit()
{
   __gnat_set Globals{
      /*
       * Main_Priority
       */
      -1,
      /*
       * Time_Slice_Value
       */
      -1,
      /*
       * Locking_Policy
       */
      ',
      /*
       * Queuing_Policy
       */
      ',
      /*
       * Tasking_Dispatching_Policy
       */
}
adafinal);

* ada__elabs ();
*/
* ada__characters__elabs ();
*/
* ada__characters__handling__elabs ();
*/
* ada__characters__latin_l__elabs ();
*/
* gnat__elabs ();
*/
* gnat__characters__elabs ();
*/
* gnat__case_util__elabs ();
*/
* gnat__case_util__elabb ();
*/
* gnat__htable__elabs ();
*/
* gnat__htable__elabb ();
*/
* gnat__io__elabs ();
*/
* gnat__io__elabb ();
*/
* interfaces__elabs ();
*/
  system__elabs();
*/
* system__exn_gen__elabs ();
*/
* system__exn_gen__elabb ();
*/
* system__exn_lli__elabs ();
*/
* system__img_bool__elabs ();
*/
* system__img_int__elabs ();
*/
* system__img_lli__elabs ();
*/
* system__img_real__elabs ();
*/
* system__parameters__elabs ();
*/
* system__parameters__elabb ();
*/
  interfaces__c_streams__elabs();
*/
  interfaces__c_streams__elabb ();
*/
  system__powten_table__elabs();
*/
  system__standard_library__elabs ()
*/
108
/*
* system_exception_table__elabs ()
*/
 system_exception_table__elabb();
 ada__exceptions__elabs();
 ada__strings__elabs();
/*
* io_exceptions__elabs ()
*/
/*
* system__storage_elements__elabs ()
*/
* system__storage_elements__elabb ()
/*
* system__secondary_stack__elabs ()
*/
* system__img_lli__elabb ()
/*
* system__img_int__elabb ()
*/
/*
* system__img_bool__elabb ()
*/
 ada__tags__elabs();
 ada__tags__elabb();
 ada__streams__elabs();
 ada__exceptions__elabs();
/*
* system__string_ops__elabs ()
*/
/*
* system__string_ops__elabb ()
*/
/*
* system__task_specific_data__elabs ()
*/
/*
* system__tasking_soft_links__elabs ()
*/
 system__tasking_soft_links__elabb();
/*
* system__task_specific_data__elabb ()
*/
* system__secondary_stack__elabb();
/*
* system__standard_library__elabb ()
*/
/*
* system__exn_lli__elabs ()
*/
/*
* system__unsigned_types__elabs ()
*/
 ada__strings__maps__elabs();
 ada__strings__maps__constants__elabs();
 ada__characters__handling__elabb();
/*
* system__bit_ops__elabs ()
*/
/*
* system__bit_ops__elabb ()
*/
/*
* ada__strings__maps__elabb ()
*/
/*
* system__fat_lli__elabs ()
*/
/*
* system__img_biu__elabs ()
*/
system img_biu_elabb();
*/
* system_img_lbu_elabs();
*/
* system_img_lbu_elabb();
*/
* system_img_lub_elabs();
*/
* system_img_lub_elabb();
*/
* system_img_lsu_elabs();
*/
* system_img_lsu_elabb();
*/
* system_img_llw_elabs();
*/
* system_img_llw_elabb();
*/
* system_img_uns_elabs();
*/
* system_img_uns_elabb();
*/
* system_img_real_elabb();
*/
* system_img_wiu_elabs();
*/
* system_img_wiu_elabb();
*/
* system_stream_attributes_elabs();
*/
* system_stream_attributes_elabb();
*/
* ada_exceptions_elabb();
*/
* system_finalization_root_elabs();
*/
* system_finalization_root_elabb();
*/
* system_finalization_implementation_elabs();
*/
* system_finalization_implementation_elabb();
*/
adas_finalization_elabs();
*/
adas_finalization_elabb();
*/
adas_finalization_list_controller_elabs();
*/
adas_finalization_list_controller_elabb();
*/
* system_file_control_block_elabs();
*/
* system_file_io_elabs();
*/
* system_file_io_elabb();
* ada_text_io_elabs();
* ada_text_io_elabb();
*/
adafloat_text_io_elabs();
*/
adainteger_text_io_elabs();
*/
/*
* ada_long_long_integer_text_io__elabs ();
*/
* ada_text_io_enumeration_aux__elabs ();
*/
* ada_text_io_float_aux__elabs ();
*/
* ada_float_text_io__elabb ();
*/
* ada_text_io_generic_aux__elabs ();
*/
* ada_text_io_generic_aux__elabb ();
*/
* ada_text_io_enumeration_aux__elabb ();
*/
* ada_text_io_integer_aux__elabs ();
*/
* ada_long_long_integer_text_io__elabb ();
*/
* ada_integer_text_io__elabb ();
*/
* system_val_bool__elabs ();
*/
* system_val_enum__elabs ();
*/
* system_val_int__elabs ();
*/
* system_val_lli__elabs ();
*/
* ada_text_io_integer_aux__elabb ();
*/
* system_val_llu__elabs ();
*/
* system_val_real__elabs ();
*/
* ada_text_io_float_aux__elabb ();
*/
* system_val_uns__elabs ();
*/
* system_val_util__elabs ();
*/
* system_val_util__elabb ();
*/
* system_val_uns__elabb ();
*/
* system_val_real__elabb ();
*/
* system_val_llu__elabb ();
*/
*/
111
* system_val_lli_elabb ();
  */
* system_val_int_elabb ();
  */
* system_val_enum_elabb ();
  */
* system_val_bool_elabb ();
  */
text_io_elabs ();
  */
  * max_elabb ();
  */
  * min_elabb ();
    a_strings_elabs();
    a_strings_elabb();
  */
  * bool_io_elabs ();
  */
  * bool_io_elabb();
  */
  * delimiter_pkg_elabs ();
  */
  * delimiter_pkg_elabb();
*/
* generic_buffered_allocation_elabs ();
  */
  * generic_set_pkg_elabs();
  */
  * int_io_elabs();
  */
  * int_io_elabb();
  */
  * lookahead_pkg_elabs();
  */
  * lookahead_pkg_elabb();
  */
  * millisec_pkg_elabs();
  */
  * millisec_pkg_elabb();
  */
  * ordered_set_pkg_elabs();
  */
  * parser_goto_elabs();
  */
  * parser_lex_dfa_elabs();
    parser_lex_dfa_elabb();
    parser_lex_io_elabs();
    parser_lex_io_elabb();
    parser_shift_reduce_elabs();
  */
  * sb_utils_elabs();
  */
  * sb_utils_elabb();
  * shared_free_list_elabs();
  * shared_free_list_elabb();
  * generic_buffered_allocation_elabb();
  * ordered_set_pkg_elabb();
  * square_root_pkg_elabs();
  * square_root_pkg_elabb();
  * generic_set_pkg_elabb();
/*
 * generic_map_pkg.elabs();
 * generic_map_pkg.elabb();
 * natural_set_pkg.elabs();
 * natural_set_pkg.elabb();
 */

* generic_sequence_pkg.elabs();
* generic_sequence_pkg.elabb();

* ordered_map_pkg.elabs();
* ordered_map_pkg.elabb();
* profile_types.elabs();
* profile_calc.elabs();
* profile_calc.elabb();

* text_pkg.elabs();
* text_pkg.elabb();

* ada_id_pkg.elabs();

* psdl_id_pkg.elabs();
* psdl_id_pkg.elabb();
* op_id_pkg.elabs();
* op_id_pkg.elabb();
* excep_id_pkg.elabs();
* excep_id_pkg.elabb();
* op_id_set_inst_pkg.elabs();
* op_id_set_inst_pkg.elabb();

* output_id_pkg.elabs();
* output_id_pkg.elabb();
* timing_map_inst_pkg.elabs();
* timing_map_inst_pkg.elabb();
* type_name_pkg.elabs();
* type_name_pkg.elabb();

* psdl_id_set_subtype_pkg.elabs();
* expression_pkg.elabs();
* expression_pkg.elabb();
* excep_trigger_map_inst_pkg.elabs();
* excep_trigger_map_inst_pkg.elabb();
* exec_guard_map_inst_pkg.elabs();
* exec_guard_map_inst_pkg.elabb();
* init_map_inst_pkg.elabs();
* init_map_inst_pkg.elabb();
* init_map_inst_pkg.elabb();
* out_guard_map_inst_pkg.elabs();
* out_guard_map_inst_pkg.elabb();

* psdl_type_set_subtype_pkg.elabs();
*/
void adafinal()
{
    system_finalization_implementation_finalize_global_list();
}

int b_ada_init(argc, argv, envp)

    int argc;
char **argv;
char **envp;

gnat_argc = argc;
gnat_argv = argv;
gnat_envp = envp;
__gnat_initialize();
adainit();
/*
 * _ada_sb_init (
*/

void
b_ada_final()
{
adafinal();
__gnat_finalize();
exit(gnat_exit_status);
}

unsigned sb_initB = 0x0164A2F2;
unsigned system_standard_libraryB = 0x522691A4;
unsigned system_standard_libraryS = 0x79B018CE;
unsigned a_stringsB = 0x4A10E5BB;
unsigned a_stringsS = 0x224334F3;
unsigned system_secondary_stackB = 0x5EAFC39A;
unsigned system_secondary_stackS = 0x36DFD40;
unsigned systemS = 0x08FBD4A7E;
unsigned system_task_specific_datab = 0x3FC34A96;
unsigned system_task_specific_dataS = 0x47178527;
unsigned system_tasking_soft_linkSB = 0x06D05994;
unsigned system_tasking_soft_linkssS = 0x63160326;
unsigned system_storage_elementsB = 0x6F7DFD62;
unsigned system_storage_elementsS = 0x5B2FF7B1;
unsigned system_string_opsB = 0x6E258F4E;
unsigned system_string_opsS = 0x260A1023;
unsigned system_exception_tableB = 0x2A7F6590;
unsigned system_exception_tableS = 0x192C2E08;
unsigned gnats = 0x156A40CF;
unsigned gnat_hetableB = 0x138A54C1;
unsigned gnathtableS = 0x463AD2F7;
unsigned adaS = 0x2359F9ED;
unsigned ada_float_text_ioB = 0x5AF53864;
unsigned ada_float_text_ios = 0x6CB3990E;
unsigned ada_text_obi = 0x67C3B9C4;
unsigned ada_text_ios = 0x56ACDF93;
unsigned ada_streamsS = 0x7C25DE96;
unsigned ada_tagss = 0x07DC57C0;
unsigned ada_tags = 0x0A72F2E2;
unsigned interfacesS = 0x0357D00A;
unsigned interfaces_c_streamsB = 0x0915C508;
unsigned interfaces_c_streamsS = 0x16327F6B;
unsigned system_parametersB = 0x1D5A020;
unsigned system_parametersS = 0x00004206;
unsigned system_file_ioB = 0x5640C74A;
unsigned system_file_ios = 0x3504FCF0;
unsigned ada_finalizationB = 0x4F01B4F2;
unsigned ada_finalizations = 0x0A0D66D8;
unsigned system_finalization_rootsB = 0x26610631;
unsigned system_finalization_rootsS = 0x1E9694A4;
unsigned system_stream_attributesB = 0x3E43967C;
unsigned system_stream_attributesS = 0x55C61A60;
unsigned ada_io_exceptionsS = 0x34054F96;
unsigned system_unsigned_typesS = 0x362290AA;
unsigned system_finalization_implementationB = 0x24B3332D;
unsigned system_finalization_implementationS = 0x3CC4D947;
unsigned ada_exceptionsB = 0x3016363D;
unsigned ada_exceptionsS = 0x4C07D4A0;
unsigned system_file_control_blockS = 0x7B3BF0FA;
unsigned ada_finalization_list_controllerB = 0x358E9753;
unsigned ada_finalization_list_controllerS = 0x34B32999;
APPENDIX C PSEUDO ADA SOURCE CODE

ADA interface between the C++ GUI and multilevel filtering routines.

A. SB_INTERFACE.ADS

```ada
-- $Id: sb_interface.ads,v 1.2 1998/01/18 17:05:14 greg Exp $
-- sb_interface.ads -- Software Base Search Interface.
-- Package Spec for the CAPS software base ADA interface.
-- Naval Postgraduate School
-- January 11, 1998
-- Written by Gregory L. Meckstroth

with sb_utils;
package sb_interface is
  type AString is access string;
  procedure sb_init(sbroot: in sb_utils.String_Pointer; hfname: in sb_utils.String_Pointer);
  pragma export (C, sb_init, "sb_init");
  procedure sb_search(sbroot: in sb_utils.String_Pointer; qfname: in sb_utils.String_Pointer; min_profile_rank: in out float; min_signature_rank: in out float; pf_file_buffer: in sb_utils.String_Pointer; sm_file_buffer: in sb_utils.String_Pointer);
  pragma export (C, sb_search, "sb_search");
end sb_interface;
```

B. SB_INTERFACE.G

```ada
-- $Id: sb_interface.g,v 1.2 1998/01/18 17:05:14 greg Exp S$
-- sb_interface.g -- Software Base Search Interface.
-- Package Body for the CAPS software base ADA interface.
-- Entry points: IntPut, Get_Char_Line, Get_Char_Word, C_to_Ada_String, Ada_Strcpy, Display_Message_line, Display_Message, reformat.
-- Naval Postgraduate School
-- January 11, 1998
-- Written by Gregory L. Meckstroth

with text_io; use text_io;
with ada.float_text_io; use ada.float_text_io;
with ada.integer_text_io;
with software_base;
with sb_utils;
with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
with profile_calc; use profile_calc;
```
package body sb_interface is

  -- Procedure: sb_init
  -- Description: "C" language Interface for software base initialization.
  --
  procedure sb_init(sbroot: in sb_utils.String_Pointer;
  hfname: in sb_utils.String_Pointer) is
    sb_dir_name: AString;
    header_name: AString;
    nst: a_string;
  begin
    sb_utils.Display_Message_line("Initializing Software Base");
    header_name := new string(sb_utils.C_to_Ada_String(hfname));
    sb_dir_name := new string(sb_utils.C_to_Ada_String(sbroot));
    software_base.initialize(sb_dir_name.all, header_name.all);
    sb_utils.Display_Message_line("finished.");
    nst := to_a("Found");
    nst := nst & integer'image(software_base.numComponents);
    nst := nst & " components in";
    nst := nst & integer'image(software_base.numOccupiedPartitions);
    nst := nst & " partitions."
    sb_utils.Display_Message_line(nst.s);
  end sb_init;

  -- Procedure: sb_search
  -- Description: "C" language Interface for software base search.
  --
  procedure sb_search(sbroot: in sb_utils.String_Pointer;
  qfname: in sb_utils.String_Pointer;
  min_profile_rank: in out float;
  min_signature_rank: in out float;
  pf_file_buffer: in sb_utils.String_Pointer;
  sm_file_buffer: in sb_utils.String_Pointer) is
    min_pr: float;
    min_sr: float;
    the_candidates: CandidateSet;
    sn, the_branch, another_branch: SigMatchNode;
    q_ops, c_ops: OpWithProfileSeq;
    batch_file: file_type;
    queries_dir, results_dir: a_string;
    query_filename, sm_filename, p_hist_filename, sm_hist_filename: a_string;
    candidate_filename: string(1..256);
    length: integer;
    temp_candidate: Candidate;
    query_name, pf_file, sm_file: AString;
    sb_dir_name: AString;
    IO_buffer_file: file_type;
    procedure Cfiltering_doneCB;
    pragma Import(C, Cfiltering_doneCB, "profile_filtering_doneCB_Fv");
  begin
    sb_dir_name := new string(sb_utils.C_to_Ada_String(sbroot));
    query_name := new string(sb_utils.C_to_Ada_String(qfname));
    pf_file := new string(sb_utils.C_to_Ada_String(pf_file_buffer));
    sm_file := new string(sb_utils.C_to_Ada_String(sm_file_buffer));
    min_pr := min_profile_rank - 0.0001;
    min_sr := min_signature_rank - 0.0001;
    -- Reinitialize the search database from the save initialization data.
sb_utils.Display_Message_line("Reinitializing Software Base");
software_base.reinitialize(sb_dir_name.all);
sb_utils.Display_Message_line("finished.");

-- First do the profile filtering to reduce the number of components that
-- we have to do signature mating on.

query_filename := to_a(query_name.all);
sb_utils.Display_Message("PROCESSING ");
sb_utils.Display_Message_line(convert(text(query_filename)));

sb_utils.Display_Message("Profile Filtering...");
the_candidates := software_base.profileFilter(convert(text(query_filename)));
sb_utils.Display_Message_line("finished.");

-- Buffer the results to disk so that the "C++" interface can display it
-- to the user.

create(IO_buffer_file, out_file, pf_file.all);
set_output(IO_buffer_file);

put("Found ");
ada.integer_text_io.put(software_base.numComponents, 0);
put(" components in ");
ada.integer_text_io.put(software_base.numOccupiedPartitions, 0);
put_line(" Partitions.");

put("There are ");
ada.integer_text_io.put(candidate_set_pkg.size(the_candidates), 0);
put_line(" possible candidates.");
candidateSetPut(the_candidates);
new_line;
set_output (standard_output);
close(IO_buffer_file);
sb_utils.reformat(pf_file.all, 0);

-- This is the "C++" callback that will display the results.
Cfiltering_doneCB;

the_candidates := profileSkim(min_pr, the_candidates);

-- Now do the signature matching on the components that pass profile
-- filtering. Buffer the results to disk for the "C++" interface.

create(IO_buffer_file, out_file, sm_file.all);
set_output(IO_buffer_file);

if candidate_set_pkg.size(the_candidates) <= 0 then
  put("0 candidates have profile rank >= ");
  ada.float_text_io.put(min_pr, 2, 2, 0);
  set_output (standard_output);
close(IO_buffer_file);
return;
end if;

sb_utils.Display_Message("Signature Matching...");
ada.integer_text_io.put(candidate_set_pkg.size(the_candidates), 0);
ada.float_text_io.put(min_pr, 2, 2, 0);
new_line;
foreach((c: Candidate), candidate_set_pkg.scan, (the_candidates),
temp_candidate := software_base.signatureMatch(convert(text(query_filename)),
c, min_sr);
software_base.getCandidateFilename(temp_candidate.component_id,
candidate_filename, length);
for i in 1..length loop
  put(candidate_filename(i));
end loop;
new_line;
candidatePrint(temp_candidate);
)
set_output (standard_output);
C. SB_UTILS.ADS

-- $Id: sb_utils.ads,v 1.2 1998/01/18 17:05:14 greg Exp $
-- sb_utils.ads -- Software Base Search Interface.
-- Package Spec for the CAPS software base ADA utilities.
-- Naval Postgraduate School
-- January 11, 1998
-- Written by Gregory L. Meckstroth

with system;

package sb_utils is
  -- Pointer to C char* arguments
  subtype String_Pointer is system.address;

  -- Put an integer to the output stream (used to save search data)
  procedure IntPut (int: Integer);

  -- Get an integer from the input stream (used to read search data)
  procedure IntGet (int: in out Integer);

  -- Get a line that is terminated with a ';' (used to read search data)
  procedure Get_Char_Line (cline: in out String; last: in out Integer);

  -- Get a word from the input stream that has separator characters ']', '}', ',', ';
  procedure Get_Char_Word (cword: in out String; last: in out Integer);

  -- Convert a "C" string (char*) to an ADA string.
  function C_to_Ada_String (the_cstring: in String_Pointer) return String;

  -- Copy an ADA string to a "C" string
  procedure Ada_Strcpy (the_cstring: in String_Pointer; the_adastring : string);

  -- Display a software base message on the GUI with new line
  procedure Display_Message_line (message: string);

  -- Display a software base message on the GUI
  procedure Display_Message (message: string);

  -- Reformat the search output for display on the GUI
  procedure reformat (filename: string; nskipcommas: integer);
end sb_utils;

D. SB_UTILS.G

-- $Id: sb_utils.g,v 1.2 1998/01/18 17:05:14 greg Exp $
-- sb_utils.g -- Software Base Search Interface
-- Package Body for the CAPS software base ADA utilities.
-- Entry points: IntPut, Get_Char_Line, Get_Char_Word, C_to_Ada_String,
-- Ada_Strcpy, Display_Message_line, Display_Message, reformat.
with lookahead_pkg;
with text_io;
with ada.integer_text_io;

package body sb_utils is

package Int_IO is new Text_IO.Integer_IO (Num => Integer);

-- Put an integer to the output stream (used to save search data)
procedure IntPut(int: integer) is
begin
    Int_IO.Put(item => int, width => 4);
end IntPut;

-- Get an integer from the input stream (used to read search data)
-- The input can have separators that confuse the ADA IO package so
-- only read in integers numbers.
procedure IntGet(int: in out Integer) is
    use lookahead_pkg;
    -- Assume that a number will be less that 256 characters
    io_buffer: String(1..256);
    last_char: Integer;
    last_int: Integer;
    begin
        last_char := 0;
        while (lookahead_pkg.Token in '-'..'9')
            loop
                last_char := last_char + 1;
                lookahead_pkg.Get_Char(io_buffer(last_char));
            end loop;
        ada.integer_text_io.Get(io_buffer(1..last_char), int, last_int);
    end IntGet;

-- Get a line that is terminated with a ';' (used to read search data)
procedure Get_Char_Line(cline: in out String; last: in out Integer) is
begin
    last := 0;
    while lookahead_pkg.Token /= ';'
        loop
            last := last+1;
            lookahead_pkg.Get_Char(cline(last));
        end loop;
    lookahead_pkg.Skip_Char;
end Get_Char_Line;

-- Get a word from the input stream that is followed by a separator character
function Is_Seperator(c: in Character) return Boolean is
 begin
    if c = ',' or c = ascii.r_bracket or c = ';' or c = '}'
    then return True;
    end if;
 return False;
end Is_Seperator;
begin
    last := 0;
    while not Is_Seperator(lookahead_pkg.Token)
        loop
            last := last + 1;
            lookahead_pkg.Get_Char(cword(last));
        end loop;
end Get_Char_Word;
--- Convert a "C" string (char*) to an ADA string.

function C_to_Ada_String(the_cstring: in String_Pointer) return String is
    function Strlen(s : String_Pointer) return Integer;
    pragma Import (C, Strlen, "strlen");
    nlen: integer := Strlen(the_cstring);
    the_string: String(1..nlen);
    for the_string use at the_cstring;
begin
    return the_string;
end C_to_Ada_String;

--- Copy an ada string to a "C" string

procedure Ada_Strcpy(the_cstring: in String_Pointer; the_adastring : string) is
    last: integer := the_adastring'last;
    cstring: String(1..last+1);
    for cstring use at the_cstring;
begin
    for i in L.last loop
        cstring(i) := the_adastring(i);
    end loop;
    cstring(last+1) := character'first;
end Ada_Strcpy;

--- Display a message on the GUI with new line

procedure Display_Message_line(message: string) is
    lf: string(1..1);
begin
    Display_Message(message);
    lf := ascii.'lf';
    Display_Message(lf);
end Display_Message_line;

--- Display a message on the GUI

type AString is access string;
procedure Display_Message(message: string) is
    procedure Cdisplay(Cmsg: in out AString);
    pragma Import(C, Cdisplay, "display_message_FPC");
    Cmessage: AString;
begin
    Cmessage := new string(1..message'last+1);
    for i in 1..message'last loop
        Cmessage(i) := message(i);
    end loop;
    Cmessage(message'last+1) := character'first;
    Cdisplay(Cmessage);
end Display_Message;

procedure reformat(filename: string; nskipcommas: integer) is
    procedure CReformat(fname: in out AString; skip: in out integer);
    pragma Import(C, CReformat, "Reformat_FPC");
    Cfname: AString;
    Cskip: integer;
begin
    Cfname := new string(1..filename'last+1);
    Cskip := nskipcommas;
    for i in 1..filename'last loop
        Cfname(i) := filename(i);
    end loop;
    Cfname(filename'last+1) := character'first;
    CReformat(Cfname, Cskip);
end reformat;

end sb_utils;
APPENDIX D MISCELLANEOUS SOURCE FILES

Help.txt is the source for the HelpMessages. This was created with Microsoft word then saved as HelpMessages.h then converted to a C++ header file.

A. HELP.TXT

Search and retrieval of components from the CAPS software base. This dialog box displays the PSDL query, profile filtering and signature matching results. The input consists of the query specification, minimum profile rank and minimum signature rank. The user can change the minimum rank by entering a new value in the appropriate text window. The query file name can be changed by typing the new name in the PSDL Query window. After entering the file name press the enter key to update query display. The query must be stored in a file for the search routines to work. If the user does not supply a file name the default query.psdl will be used.

After running the search the user can view the results and select the appropriate component by pushing the toggle button next to the component name. Pushing the OK button will return the selected component.

The Query menu allows the user to manage the PSDL query.
New: Starts the Syntax Directed Editor to input a new query.
Open: Open an existing query file.
Edit: Starts the Syntax Directed Editor to edit the current query.
Save: Save the current query.
SaveAs: Save the current query in a user specified file.
Close: Close the search dialog.
Exit: Exit the program.

The Search menu allows the user to run the software base search.
Start: Start the software base search.
OK: Exits the dialog and returns the selected component name.
Search: Start the software base search.
Cancel: Exit the dialog.

Initialize components in the CAPS software base. This dialog box displays the PSDL specification and component file list. The input consists of the component specification in PSDL and the component file list. The spec file name can be changed by typing the new name in the PSDL Spec window. After entering the file name press the enter key to update the spec display.

Each component in the Software Base is stored in a separate directory. After entering all files for a component the user can initialize the component directory structure by pressing the 'Init DIR' button or through the Init menu. After the component directory is initialized the use can enter another component. This can be repeated until all components have been entered. Then the Software Base must be initialized by pressing the 'Init SB' button or through the Init menu.

All component files are copied to the component directory no user files will be deleted.

The Spec menu allows the user to manage the PSDL query.
New: Starts the Syntax Directed Editor to input a new query.
Open: Open an existing query file.
Edit: Starts the Syntax Directed Editor to edit the current query.
Save: Save the current query.
SaveAs: Save the current query in a user specified file.
Close: Close the search dialog.
Exit: Exit the program.

The component menu allows the user to manage the component file list.
Add Component files:
Clear Component file list:
Clear all:

Buttons.
Dismiss: Close the init dialog.
Init DIR: Initialize the component directory.
Init SB: Run the ADA initialization for the Software Base.
Clear FL: Clear the file list.
Clear ALL: Clear file list and Component specification.

E. MAKEFILE

PSDL_TYPE_ROOT = /home/greg/PSDL_TYPE-May97
GEN = m4 generator.m4
DEL = /bin/rm -f
INCLUDES = -I$(PSDL_TYPE_ROOT)/GNAT \
     -I$(PSDL_TYPE_ROOT)/GENERIC_TYPES/GNAT \
     -I$(PSDL_TYPE_ROOT)/INSTANTIATIONS/GNAT

COMPILE.ada = gcc $(ADAFLAGS) $(INCLUDES)

.SUFFIXES: .g .adb

.g.o:
 $(GEN) $< > $*.adb
 $(COMPILE.ada) -c $*.adb $(OUTPUT_OPTION)
 $(DEL) $*.adb $*.ali

.g.adb:
 $(GEN) $< > $@

ADAFLAGS = -g

WARNINGS = -Werror -Wimplicit -Wreturn-type -Wunused -Wswitch -Wcomment -Wformat \   
    -Wchar-subscripts -Wparentheses -Wtemplate-debugging -Wpointer-arith \  
    -Wcast-align -Wstrict-prototypes -Wmissing-prototypes -Wnested-externs \   
    -Woverloaded-virtual -Winline -Wconversion -Wmissing-declarations

CXXFLAGS = -g $(WARNINGS) -I/usr/include/g++

# Don't use warnings on the "ADA" generated "C" code

CFLAGS = -g

LDFLAGS = -L/usr/XllR6/lib -L/usr/gnat/lib/gcc-lib/386-linux/2.7.2.1/adalib \
    -L/usr/lib/gcc-lib/386-linux/2.7.2.1/adalib/ -L/home/greg/psdl_lib
LOADLIBES = -lXm -lXpm -lXt -lSM -lICE -lXext -lpsdl -lgnat


XGui_OBJS = Gui.o SearchDialog.o MaintenanceDialog.o PromptDialog.o Menu.o SDE.o \ DisplayProgress.o Callbacks.o Utils.o

SB_OBJS = sb_interface.o candidate_types.o component_id_types.o haase_diagram.o \ profile_calc.o profile_types.o psdl_profile.o sb_utils.o sig_match.o \ sig_match_types.o software_base.o profile_filter_pkg.o b_ada_system.o

XGui: $(XGui_OBJS) $(SB_OBJS) $(LINK.C) -o XGui $(XGui_OBJS) $(SB_OBJS) $(LOADLIBES)

all: XGui

clean:
 $(DEL) XGui *.o *.ali *.adb *.bak *.bak - core

depend:
 makedepend -I/usr/include -I/usr/include/g++ $(XGui_SRCS)

install:
 /bin/cp -f XGui /home/greg/bin/demoXGui

# DO NOT DELETE THIS LINE -- make depend depends on it.
APPENDIX E ADA FILTERING SOURCE CODE

The following appendix is the ADA source code for Jeff Herman's [1] Profile Filtering and Signature Matching. It is included here because some of the routines were modified for use in this thesis project. The following files were modified:

- candidate_types.g
- component_id_types.ads
- component_id_types.g
- haase_diagram.ads
- haase_diagram.g
- profile_types.ads
- profile_types.g
- psdl_profile.ads
- psdl_profile.g
- sig_match_types.g
- software_base.ads
- software_base.g

A. CANDIDATE_TYPES.ADS

```ada
-- Package Spec: candidate_types

with generic_sequence_pkg;
with ordered_set_pkg;
with component_id_types; use component_id_types;
with sig_match_types; use sig_match_types;

package candidate_types is

  RANK_UNKNOWN: constant := -1.0;

  -- Candidate
  type Candidate is record
```
profile rank: float;
keyword rank: float;
signature matches: SigMatchNodePtrSet;
component id: ComponentID;
end record;

function candidateEqual(c1: in Candidate; c2: in Candidate) return boolean;
function candidateLessThan(c1: in Candidate; c2: in Candidate) return boolean;
procedure candidateAssign(c1: in out Candidate; c2: in Candidate);
procedure candidatePut(the_candidate: in Candidate);
procedure candidatePrint(the_candidate: in Candidate);

function newCandidate return Candidate;
procedure generateSigMatchHistogram(filename: in string; c: in Candidate);

-- CandidateSequence
-- Note: should use addCandidate to add a candidate to the CandidateSequence.
-- addCandidate keeps the CandidateSequence sorted.
package candidate_sequence_pkg is new generic_sequence_pkg(
  t => Candidate, average_size => 4);
subtype CandidateSequence is candidate_sequence_pkg.sequence;

function candidateSequenceEqual is
  new candidate_sequence_pkg.generic_equal(eq => candidateEqual);
function candidateSequenceMember is
  new candidate_sequence_pkg.generic_member(eq => candidateEqual);
procedure candidateSequenceRemove is
  new candidate_sequence_pkg.generic_remove(eq => candidateEqual);
function candidateSequenceSort is
  new candidate_sequence_pkg.generic_sort("<" => candidateLessThan);
procedure candidateSequencePut is
  new candidate_sequence_pkg.generic_put(put => candidatePut);
procedure addCandidate(c: in Candidate; cs: in out CandidateSequence);

-- CandidateSet
package candidate_set_pkg is new ordered_set_pkg(t => Candidate,
  eq => candidateEqual, "<" => candidateLessThan);
subtype CandidateSet is candidate_set_pkg.set;

procedure candidateSetPut is
  new candidate_set_pkg.generic_put(put => candidatePut);
function profileSkim(profile_threshold: in float; the_candidates: in CandidateSet) return CandidateSet;
procedure generateProfileHistogram(filename: in string; the_candidates: in CandidateSet);

end candidate_types;

B. CANDIDATE_TYPES.G

-- Package Body: candidate_types

with ada.text_io;
with ada.float_text_io;
with ada.integer_text_io;
with component_id_types; use component_id_types;

130
package body candidate_types is

-- Function: candidateEqual
function candidateEqual(cl: in Candidate; c2: in Candidate) return boolean is
  begin
    return cl.component_id = c2.component_id;
  end candidateEqual;

-- Description: sort candidates in rank descending order (highest rank first).
function candidateLessThan(cl: in Candidate; c2: in Candidate) return boolean is
  begin
    -- TODO
    if cl.profile_rank > c2.profile_rank then
      return true;
    -- the following test for less-than is just being paranoid
    -- about potential float equality problems
    elsif cl.profile_rank < c2.profile_rank then
      return false;
    else
      return cl.component_id < c2.component_id;
    end if;
  end candidateLessThan;

-- Procedure: candidateAssign
procedure candidateAssign(cl: in out Candidate; c2: in Candidate) is
  begin
    cl.profile_rank := c2.profile_rank;
    cl.keyword_rank := c2.keyword_rank;
    cl.component_id := c2.component_id;
    sig_match_node_ptr_set_pkg.assign(cl.signature_matches,
                                       c2.signature_matches);
  end candidateAssign;

-- Procedure: candidatePut
procedure candidatePut(the_candidate: in Candidate) is
  begin
    ada.text_io.put(" ( ");
    ada.integer_text_io.put(the_candidate.component_id, 0);
    ada.text_io.put(" I ");
    ada.float_text_io.put(the_candidate.profile_rank, 1, 2, 0);
    ada.text_io.put(" I ");
    sigMatchNodePtrSetPut(the_candidate.signature_matches);
    ada.text_io.put(" )");
  end candidatePut;

-- Procedure: candidatePrint
procedure candidatePrint(the_candidate: in Candidate) is
  begin
    ada.text_io.put("  Component ID: ");
    ada.integer_text_io.put(the_candidate.component_id, 0);
    ada.text_io.new_line;
    ada.text_io.put("  Profile Rank: ");
    ada.float_text_io.put(the_candidate.profile_rank, 1, 2, 0);
    ada.text_io.put(" Number of signature match solutions: ");
    ada.integer_text_io.put(sig_match_node_ptr_set_pkg.size(
                             the_candidate.signature_matches), 0);
    ada.text_io.new_line;
    sigMatchNodePtrSetPrint(the_candidate.signature_matches);
  end candidatePrint;
end candidatePrint;

-- Function: newCandidate
--
function newCandidate return Candidate is
  return_val: Candidate;
  begin
    return_val.profile_rank := RANK_UNKNOWN;
    return_val.keyword_rank := RANK_UNKNOWN;
    return_val.signature_matches := sig_match_node_ptr_set_pkg.empty;
    return return_val;
  end newCandidate;

-- generateSigMatchHistogram
--
-- Description: generates histogram data of the signature ranks for the
-- set of signature matches and saves it to a file so it can be
-- read by a charting program. The format is one line
-- for each pair where the first item of the pair is the
-- profile rank and the second item is the number of
-- candidates with that rank.

procedure generateSigMatchHistogram(filename: in string; c: in Candidate) is
  ft: ada.text_io.file_type;
  last_rank: float;
  count: natural := 0;
  temp_snp: SigMatchNodePtr;

  procedure putPair(the_rank: float; the_count: natural) is
    begin
      ada.float_text_io.put(ft, the_rank, 1, 2, 0) ;
      ada.text_io.put(ft, " ");
      ada.integer_text_io.put(ft, the_count);
      ada.text_io.new_line(ft);
    end putPair;

  begin
    ada.text_io.create(ft, ada.text_io.out_file, filename);
    if sig_match_node_ptr_set_pkg.size(c.signature_matches) = 0 then
      ada.text_io.close(ft);
      return;
    end if;
    temp_snp := sig_match_node_ptr_set_pkg.fetch(c.signature_matches, 1);
    last_rank := temp_snp.signature_rank;
    foreach((snp: SigMatchNodePtr), sig_match_node_ptr_set_pkg.scan,
      (c.signature_matches),
      if snp.signature_rank /= last_rank then
        putPair(last_rank, count);
        last_rank := snp.signature_rank;
        count := 1;
      else
        count := count + 1;
      end if;
    )
    putPair(last_rank, count);
    ada.text_io.close(ft);
  end generateSigMatchHistogram;

-- Procedure: addCandidate

procedure addCandidate (c: in Candidate; cs: in out CandidateSequence) is
  begin
    candidate_sequence_pkg.add(c, cs);
    cs := candidateSequenceSort(cs);
  end addCandidate;

  -- Function: profileSkim (for CandidateSet)
function profileSkim(profile_threshold: in float; the_candidates: in CandidateSet) return CandidateSet is
return_val: CandidateSet;
begin
return_val := candidate_set_pkg.empty;
foreach((c: Candidate), candidate_set_pkg.scan, (the_candidates),
if c.profile_rank >= profile_threshold then
candidate_set_pkg.add(c, return_val);
end if;
)
return return_val;
end profileSkim;

procedure generateProfileHistogram(filename: in string; the_candidates: CandidateSet) is
ft: ada.text_io.file_type;
last_rank: float;
count: natural := 0;
temp_candidate: Candidate;

procedure putPair(the_rank: float; the_count: natural) is
begin
ada.float_text_io.put(ft, the_rank, 1, 2, 0) ;
ada.text_io.put(ft, " ");
ada.integer_text_io.put(ft, the_count);
ada.text_io.new_line(ft) ;
end putPair;
begin
ada.text_io.create(ft, ada.text_io.out_file, filename);
if candidate_set_pkg.size(the_candidates) = 0 then
ada.text_io.close(ft);
return;
end if;
temp_candidate := candidate_set_pkg.fetch(the_candidates, 1);
last_rank := temp_candidate.profile_rank;
foreach((c: Candidate), candidate_set_pkg.scan, (the_candidates),
if c.profile_rank /= last_rank then
putPair(last_rank, count);
last_rank := c.profile_rank;
count := 1;
else
count := count + 1;
end if;
)
putPair(last_rank, count);
ada.text_io.close(ft);
end generateProfileHistogram;
end candidate_types;

C. COMPONENT_ID_TYPES.ADS

-- Package Spec: component_id_types
with gnat.io; use gnat.io;
with generic_map_pkg;
with generic_set_pkg;
with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
with sb_utils;
with psdl_profile; use psdl_profile;

package component_id_types is
  --
  -- ComponentID
  --
  subtype ComponentID is integer;
  procedure componentIDPut(c_id: ComponentID);

  -- Component
  --
  -- Note: Make sure to use createComponent to instantiate a new Component.
  -- This will ensure that generics_mapping is initialized.
  --
  type Component is record
    psdl_filename: text;
    generics_mapping: GenericsMap;
  end record;

  function createComponent return Component;

  procedure addGenericsMapping(generic_type_id: psdl_id;
    actual_type_id: psdl_id; the_component: in out Component);

  function componentEqual(cl: in Component; c2: in Component) return boolean;

  procedure componentPut(the_component: in Component);

  procedure componentGet(the_component: out Component);

  --
  -- ComponentIDMap
  --
  package component_id_map_pkg is new generic_map_pkg(
    key => ComponentID,
    result => Component,
    eq_key => "=",
    eq_res => ComponentEqual,
    average_size => 8);
  subtype ComponentIDMap is component_id_map_pkg.map;
  -- These raise an exception
  -- procedure componentIDMapPut is new component_id_map_pkg.generic_put(
  -- key_put => put, res_put => componentPut);
  --
  -- procedure componentIDMapFilePut is new component_id_map_pkg.generic_file_put(
  -- key_put => put, res_put => componentPut);

  procedure componentIDMapPut(Comp_IDMap: ComponentIDMap);

  --
  -- ComponentIDSet
  --
  package component_id_set_pkg is new generic_set_pkg(
    t => ComponentID,
    average_size => 8,
    eq => "=");
  subtype ComponentIDSet is component_id_set_pkg.set;

  procedure componentIDSetPut is
    new component_id_set_pkg.generic_put(put => sb_utils.IntPut);

  procedure componentIDSetFilePut is
    new component_id_set_pkg.generic_file_put(put => componentIDPut);
procedure componentIDSetGet is
  new component_id_set_pkg.generic_input(input => sb_utils.IntGet); -- gnat.io.put);
end component_id_types;

D. COMPONENT_ID_TYPES.G

-----------------------------------
Package Body: component_id_types
-----------------------------------
with text_io;
with ada.integer_text_io;
with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
package body component_id_types is
  package Int_IO is new Text_IO.Integer_IO (Num => Integer);
  -- Procedure: componentIDPut
  -- procedure componentIDPut(c_id: ComponentID) is
  --   begin
  --     ada.integer_text_io.put(c_id, 0);
  --   end componentIDPut;
  --
  -- Procedure: componentIDMapPut
  -- procedure componentIDMapPut(Comp_IDMap: ComponentIDMap) is
  --   map_size: integer;
  --   begin
  --     foreach((the_comp_id: ComponentID; the_component: Component),
  --       component_id_map_pkg.scan, (Comp_IDMap),
  --       Int_IO.Put(the_comp_id, 0);
  --       text_io.put("; ");
  --       text_io.put(convert(the_component.psdl_filename));
  --       text_io.put("; ");
  --       map_size := generics_map_pkg.size(the_component.generics_mapping);
  --       if map_size < 1 then
  --         text_io.put("; ");
  --       else
  --         genericsMapPut(the_component.generics_mapping);
  --       end if;
  --       text_io.new_line;
  --   )
  -- end componentIDMapPut;
  --
  -- Procedure: createComponent
  -- function createComponent return Component is
  --   return_val: Component;
  -- begin
  --   generics_map_pkg.create(empty, return_val.generics_mapping);
  --   return return_val;
  -- end createComponent;
  --
  -- Procedure: addGenericsMapping
  -- procedure addGenericsMapping(generic_type_id: psdl_id;
  -- actual_type_id: psdl_id; the_component: in out Component) is
  -- begin
  --   generics_map_pkg.bind(generic_type_id, actual_type_id,
  --     the_component.generics_mapping);
  -- end addGenericsMapping;
  --
  -- Function: componentEqual
function componentEqual(cl: in Component; c2: in Component) return boolean is
begin
    if not eq(cl.psdl_filename, c2.psdl_filename) then
        return false;
    end if;

    return generics_map_pkg.equal(cl.generics_mapping, c2.generics_mapping);
end componentEqual;

-- Procedure: componentPut
procedure componentPut(the_component: in Component) is
begin
    text_io.put(convert(the_component.psdl_filename));
    text_io.put(" | ");
    genericsMapPut(the_component.generics_mapping);
end componentPut;

-- Procedure: componentGet
procedure componentGet(the_component: out Component) is
    filename: string(1..256);
begin
    text_io.get(filename);
    --the_component.psdl_filename := text(filename);
    --genericsMapGet(the_component.generics_mapping);
end componentGet;
end component_id_types;

E. HAASE_DIAGRAM.ADS

with generic_map_pkg;
with profile_types; use profile_types;
with component_id_types; use component_id_types;
package haase_diagram is

-- Types
-- type HaaseNode is private;
-- type HaaseDiagram is private;

-- HaaseNode
type HaaseNode is record
    key: ComponentProfile;
    components: ComponentIDSet;
    children: ComponentProfileSet;
end record;

function haaseNodeEqual(hn1: in HaaseNode; hn2: in HaaseNode) return boolean;
procedure haaseNodeAssign(hn1: in out HaaseNode; hn2: in HaaseNode);
procedure haaseNodePut(the_haase_node: in HaaseNode);
procedure haaseNodeGet(the_haase_node: in out HaaseNode; last_node: in out boolean);
procedure haaseNodePrint(the_haase_node: HaaseNode);
package haase_node_map_pkg is new generic_map_pkg(
  key => ComponentProfile,
  result => HaaseNode,
  eq_key => componentProfileEqual,
  eq_res => haaseNodeEqual,
  average_size => 8);
subtype HaaseDiagram is haase_node_map_pkg.map;

procedure haaseDiagramPut is new haase_node_map_pkg.generic_put(
  key_put => componentProfilePut, res_put => haaseNodePut);

procedure haaseDiagramFilePut is new haase_node_map_pkg.generic_file_put(
  key_put => componentProfilePut, res_put => haaseNodePut);

procedure haaseDiagramPut(the_haase_diagram: HaaseDiagram);

procedure haaseDiagramGet is new haase_node_map_pkg.generic_input(
  key_input => componentProfileGet, res_input => haaseNodeGet);

procedure haaseDiagramGet(diagram: in out HaaseDiagram);

procedure haaseDiagramPrint(the_haase_diagram: HaaseDiagram);

procedure generateGML(the_haase_diagram: in HaaseDiagram;
  filename: in string);

function createHaaseNode(key: in ComponentProfile) return HaaseNode;
function createHaaseDiagram return HaaseDiagram;

procedure addComponent(the_comp_id: in ComponentID;
  the_haase_node: in out HaaseNode);

procedure addChild(the_child_key: in ComponentProfile;
  the_haase_node: in out HaaseNode);

procedure addHaaseNode(the_haase_node: in HaaseNode;
  the_haase_diagram: in out HaaseDiagram);

procedure addBaseNodes(the_haase_diagram: in out HaaseDiagram);

procedure connectNodes(the_haase_diagram: in out HaaseDiagram);

package body haase_diagram is

end haase_diagram;

package haase_diagram is

end haase_diagram;

F. HAASE_DIAGRAM.G

package body haase_diagram is

end haase_diagram;
Function: createHaaseNode

Description: create and initialize a HaaseNode for use.

function createHaaseNode(key: in ComponentProfile) return HaaseNode is
    return_val: HaaseNode;
    begin
        profile_id_sequence_pkg.assign(return_val.key, key);
        return_val.components := component_id_set_pkg.empty;
        return_val.children := component_profile_set_pkg.empty;
        return return_val;
    end createHaaseNode;

Function: createHaaseDiagram

Description: create and initialize a HaaseDiagram for use.

function createHaaseDiagram return HaaseDiagram is
    begin
        return haase_node_map_pkg.create(
            createHaaseNode(profile_id_sequence_pkg.empty));
    end createHaaseDiagram;

Function: addComponent

Description: add a ComponentID to the HaaseNode.

procedure addComponent(the_comp_id: in ComponentID;
    the_haase_node: in out HaaseNode) is
    begin
        component_id_set_pkg.add(the_comp_id, the_haase_node.components);
    end addComponent;

Function: addChild

Description: add a ComponentProfile that represents the key to a child HaaseNode to the HaaseNode.

procedure addChild(the_child_key: in ComponentProfile;
    the_haase_node: in out HaaseNode) is
    begin
        component_profile_set_pkg.add(the_child_key, the_haase_node.children);
    end addChild;

Function: addHaaseNode

Description: add a HaaseNode to the HaaseDiagram.

procedure addHaaseNode(the_haase_node: in HaaseNode;
    the_haase_diagram: in out HaaseDiagram) is
    temp_key: ComponentProfile;
    begin
        profile_id_sequence_pkg.assign(temp_key, the_haase_node.key);
        haase_node_map_pkg.bind(temp_key, the_haase_node, the_haase_diagram);
    end addHaaseNode;

Procedure: addBaseNodes

Description: add base nodes for the nodes already in the diagram.
   This is done by adding a node for each profile in the key for each node in the diagram. Note, duplicates will not be added.

procedure addBaseNodes(the_haase_diagram: in out HaaseDiagram) is
    new_diagram: HaaseDiagram;
    new_node: HaaseNode;
    new_key: ComponentProfile;
    begin
       
new_diagram := createHaaseDiagram;
haase_node_map_pkg.assign(new_diagram, the_haase_diagram);
new_key := profile_id_sequence_pkg.empty;

foreach((p_id: ProfileID),
  profile_lookup_table_pkg.res_set_pkg.scan,
  (software_base.getProfileIDs),
  addProfileID(p_id, new_key);
if not haase_node_map_pkg.member(new_key, the_haase_diagram) then
  new_node := createHaaseNode(new_key);
  addHaaseNode(new_node, new_diagram);
end if;
new_key := profile_id_sequence_pkg.empty;
)

haase_node_map_pkg.assign(the_haase_diagram, new_diagram);
haase_node_map_pkg.recycle(new_diagram);
end addBaseNodes;

-- Procedure: connectNodes
-- Description: connect nodes in diagram. Invariant:
-- n2 is nl's child iff subbag(nl.key, n2.key) and
-- there is no node n3 such that subbag(nl.key, n3.key)
-- and subbag(n3.key, n2.key).
-- Note, an entirely new diagram is constructed because
-- scan returns copies of the nodes in the_haase_diagram,
-- not the actual nodes.

procedure connectNodes(the_haase_diagram: in out HaaseDiagram) is
new_node: HaaseNode;
new_diagram: HaaseDiagram;
found_n3: boolean;
begin
new_diagram := createHaaseDiagram;
foreach((nl_key: ComponentProfile; nl: HaaseNode),
  haase_node_map_pkg.scan, (the_haase_diagram),
  new_node := createHaaseNode(nl_key);
  haaseNodeAssign(new_node, nl);
  foreach((n2_key: ComponentProfile; n2: HaaseNode),
    haase_node_map_pkg.scan, (the_haase_diagram),
    if not haaseNodeEqual(nl,n2) then
      if subbag(nl_key, n2_key) then
        found_n3 := false;
        foreach((n3_key: ComponentProfile; n3: HaaseNode),
          haase_node_map_pkg.scan, (the_haase_diagram),
          if not found_n3 then
            if (not haaseNodeEqual(nl,n3)) and
              (not haaseNodeEqual(n2,n3)) then
              if subbag(nl_key, n3_key) and
                subbag(n3_key, n2_key) then
                found_n3 := true;
              end if;
            end if;
          end if;
        end foreach;
        if not found_n3 then
          addChild(n2_key, new_node);
        end if;
      end if;
    end if;
  end foreach;
  end if;
end connectNodes;

-- Function: haaseNodeEqual
-- Description: checks for equality of two haase nodes by
-- comparing the keys.
--
function haaseNodeEqual(hnl: in HaaseNode; hn2: in HaaseNode)
return boolean is
begin
  return componentProfileEqual(hnl.key, hn2.key);
end haaseNodeEqual;

-- Procedure: haaseNodeAssign
--
-- Description: creates a duplicate of hn2.
--
procedure haaseNodeAssign(hnl: in out HaaseNode; hn2: in HaaseNode) is
begin
  profile_id_sequence_pkg.assign(hnl.key, hn2.key);
  component_id_set_pkg.assign(hnl.components, hn2.components);
end haaseNodeAssign;

-- Procedure: haaseNodePut
--
procedure haaseNodePut(the_haase_node: in HaaseNode) is
begin
  componentProfilePut(the_haase_node.key);
  componentIDSetPut(the_haase_node.components);
  componentProfileSetPut(the_haase_node.children);
end haaseNodePut;

-- Procedure: haaseNodeGet
--
procedure haaseNodeGet(the_haase_node: in out HaaseNode; last_node: in out boolean) is
begin
  first_key := profile_id_sequence_pkg.fetch(temp_comp_profile, 1);
  if first_key >= 0 then
    last_node := False;
    the_haase_node := createHaaseNode(temp_comp_profile);
    componentIDSetGet(the_haase_node.components);
    componentProfileSetGet(the_haase_node.children);
  else
    last_node := True;
  end if;
end haaseNodeGet;

-- Procedure: haaseNodePrint
--
procedure haaseNodePrint(the_haase_node: in HaaseNode) is
begin
  put("Key: ");
  componentProfilePut(the_haase_node.key);
  new_line;
  put("Components: ");
  componentIDSetPut(the_haase_node.components);
  new_line;
  put("Children: ");
  componentProfileSetPut(the_haase_node.children);
  new_line;
end haaseNodePrint;

-- Procedure: haaseDiagramPrint
--
procedure haaseDiagramPrint(the_haase_diagram: in HaaseDiagram) is
begin
  foreach(node_key: ComponentProfile; node: HaaseNode),
    haase_node_map_pkg.scan, (the_haase_diagram),
    haaseNodePrint(node);
end haaseDiagramPrint;
procedure haaseDiagramPut(the_haase_diagram: in HaaseDiagram) is
begin
foreach((node_key: ComponentProfile; node: HaaseNode),
haase_node_map_pkg.scan, (the_haase_diagram),
haaseNodePut(node);
new_line;
)
new_line;
end haaseDiagramPut;

procedure haaseDiagramGet(diagram: in out HaaseDiagram) is
node: HaaseNode;
last_node: boolean;
begin
loop
haaseNodeGet(node, last_node);
if last_node then
exit;
end if;
addHaaseNode(node, diagram);
end loop;
end haaseDiagramGet;

procedure generateGML(the_haase_diagram: in HaaseDiagram;
filename: in string) is
id: natural := 0; /* unique ID counter */
the_id: natural;
gml_file: file_type;
function new_id return natural is
begin
id := id + 1;
return id;
end new_id;
package temp_map_pkg is new generic_map_pkg(
key => ComponentProfile,
result => natural,
eq_key => componentProfileEqual,
eq_res => =,
average_size => 8);
subtype tempMap is temp_map_pkg.map;
temp_map: tempMap;
begin
create(gml_file, out_file, filename);
put(gml_file, "graph t id ");
put(gml_file, integer'image(new_id));
put_line(gml_file, " directed 1");
temp_map_pkg.create(id, temp_map);
— make the nodes
foreach((node_key: ComponentProfile; node: HaaseNode),
haase_node_map_pkg.scan, (the_haase_diagram),
put(gml_file, "node ");
new_line;
)
new_line;
end haaseDiagramPrint;
the_id := new_id;
put(gml_file, " integer'image(the_id));
put(gml_file, " label ");
componentProfileFilePut(gml_file, node.key);
put_line(gml_file, "]");

temp_map_pkg.bind(node.key, the_id, temp_map);
}

-- make the edges
foreach((node_key: ComponentProfile; node: HaaseNode),
haase_node_map_pkg.scan, (the_haase_diagram),
foreach((child_key: ComponentProfile),

component_profile_set_pkg.scan, (node.children),

put(gml_file, " edge ");
put(gml_file, integer'image(new_id));
put(gml_file, " source ");
put(gml_file, integer'image(temp_map_pkg.fetch(temp_map, node.key)));
put(gml_file, " target ");
put(gml_file, integer'image(temp_map_pkg.fetch(temp_map, child_key)));
put_line(gml_file, "]");
}

put_line(gml_file, "]");
close(gml_file);
temp_map_pkg.recycle(temp_map);
end generateGML;
end haase_diagram;

G. PROFILE_CALC.ADS

with profile_types; use profile_types;

package profile_calc is

-- Package Spec: profile_calc
--
-- This package contains functions and types that support the computation
-- of profiles from numeric representations of signatures.
--
-- Description of numeric signatures: Positive integers represent
-- instances of non-generic types in the signature. Negative integers
-- represent instances of generic types in the signature. Finally,
-- a 0 is used to terminate the array of integers representing the
-- signature.
--
-- Examples of numeric signatures:
-- [integer, char, float -> integer] ==> [1,2,3,1,0]
-- [integer, generic, float -> float] ==> [1,-1,2,3,0]
-- [generic1, generic2 -> generic2] ==> [-1,-2,-2,0]
--
-- Profiles are sequences of integers.
--
-- Generic Types:
-- Generic types cause more than one profile to be generated for a
-- single signature. Hence, computeArrayProfileWithGenerics returns an
-- array of ArrayProfiles, ProfileValues, bound by NumProfiles.
--
-- ArrayProfiles are terminated with PROFILE TERMINATOR. For example,
-- the profile [3,1,1,2] is returned as [3,1,1,2,-99].
--
-- Eventually a different method for handling generic types will be
-- employed and will likely do away with the ArrayProfile data type.
--
-- Types
--
MAX_SIG_LENGTH: constant := 100;
MAX_PROFILE_LENGTH: constant := 100;
MAX_PROFILE_VARIATIONS: constant := 100; -- for generic types
PROFILE_TERMINATOR: constant := -99;

subtype SignatureLengthRange is Positive range 1..MAX_SIG_LENGTH;
subtype ProfileLengthRange is Positive range 1..MAX_PROFILE_LENGTH;
subtype ProfileVariationRange is Positive range 1..MAX_PROFILE_VARIATIONS;

type Signature is array (SignatureLengthRange) of Integer;
type ArrayProfile is array (ProfileLengthRange) of Integer;
type ArrayProfiles is array (ProfileVariationRange) of ArrayProfile;

--
-- Functions
--
function computeProfile(T: in Signature) return Profile;
function computeArrayProfile(T: in Signature) return ArrayProfile;

-- note NumProfiles should be 0..MAX_PROFILE_VARIATIONS, not Natural
procedure computeArrayProfileWithGenerics(
  T: in Signature;
  ProfileValues: out ArrayProfiles;
  NumProfiles: out Natural);

function printSignature(sig: Signature) return SignatureLengthRange;
function printArrayProfile(prof: ArrayProfile) return ProfileLengthRange;

end profile_calc;

H. PROFILE_CALC.G

package body profile_calc is

-- Function: convertToSequence
--
-- Description: helper function to convert an ArrayProfile (an
-- array of ints terminated with PROFILE_TERMINATOR)
-- to a Profile (a sequence of ints).
--
function convertToSequence(Prof: ArrayProfile) return Profile is
  return_val: Profile;
  i, count: ProfileLengthRange;
begin
  count := 1;
  while Prof(count) /= PROFILE_TERMINATOR and count <= MAX_PROFILE_LENGTH loop
    count := count + 1;
  end loop;
  count := count - 1;

  return_val := 0;
  for i in 1..count loop
    return_val := return_val + (long_long_integer(Prof(i)) *
                              (10 ** (count-i)));
  end loop;

  return return_val;
end convertToSequence;

function printSignature(Sig: Signature) return SignatureLengthRange is
  Num: SignatureLengthRange;
begin
  Num := 0;
end printSignature;
Num := 1;
Puts ("");
while Sig (Num + 1) /= 0 loop
  Put (Sig (Num));
  if Sig (Num + 2) /= 0 then
    Put (", ");
  end if;
  Num := Num + 1;
end loop;
Puts (" -> ");
Puts (Sig (Num));
return Num;
end printSignature;

function printArrayProfile (Prof: ArrayProfile) return ProfileLengthRange is
  Num: ProfileLengthRange;
begin
  Num := 1;
Puts ("");
  while Prof (Num) /= PROFILE TERMINATOR and Num < MAX PROFILE LENGTH loop
    Put (Prof (Num));
    if Prof (Num + 1) /= PROFILE TERMINATOR then
      Put (", ");
    end if;
    Num := Num + 1;
  end loop;
  return Num;
end printArrayProfile;

function computeProfile (T: Signature) return Profile is
begin
  return convertToSequence (computeArrayProfile (T));
end computeProfile;

function computeArrayProfile (T: Signature) return ArrayProfile is
  Result: ArrayProfile;
  Result_Count : Integer;
  NumResSort: Integer;
  NumOneSorts: Integer;
  I, J: Integer;
  L: SignatureLengthRange;
  SortValues: array (SignatureLengthRange) of Integer;
  SortNums: array (SignatureLengthRange) of Integer;
  NumSorts: Integer;
  Found: Boolean;
begin
  -- Compute Profile[1], Total Number of Sorts.
  Result_Count := 1;
  J := 0;

  -- set L to number of elements in T
  -- note, this is the first number in the profile
  I := 1;
  while (T (I) /= 0 and I <= MAX SIG_LENGTH) loop
    I := I + 1;
  end loop;
  L := I - 1;

  Result (Result_Count) := L;

  -- Compute Profile[2], Number of Times Result Sort in Signature.
  -- note, Nguyen's thesis just uses 0 or 1 to indicate if the
  -- result sort is used in the input arguments. Representing
  -- the number of times the result sort is used is finer resolution,
  -- which should partition the software base better.
  NumResSort := 0;
  for I in 1..L loop
    if T (I) = T (L) then
      NumResSort := NumResSort + 1;
    end if;
  end loop;
  Result_Count := Result_Count + 1;
Herman

-- Herman Improvement Profile[3]
-- Add the number of occurrences of the type being defined by the
-- component (if the component is a type).
-- Result_Count := Result_Count + 1;
-- Result(Result_Count) := T(L+2);

-- Herman Improvement Profile[4..8]
-- Add the number of occurrences of types in the basic sort groups
Result_Count := Result_Count + 1;
Result(Result_Count) := T(L+3);
Result_Count := Result_Count + 1;
Result(Result_Count) := T(L+4);
-- Result_Count := Result_Count + 1;
-- Result(Result_Count) := T(L+5);
Result_Count := Result_Count + 1;
Result(Result_Count) := T(L+6);
Result_Count := Result_Count + 1;
Result(Result_Count) := T(L+7);

-- Generate Helper Arrays
-- SortValues: an ordered SET of sort values
-- e.g. If the signature input T was [1, 1, 2, 1, 0]
-- SortValues would be [1, 2]
-- NumSorts: the cardinality of the ordered set SortValues
-- e.g. in the above example, NumSorts would be 2
-- Add the number of occurrences of each sort in SortValues
-- e.g. in the above example, SortValues would be [3, 1]
for I in 1..L loop
    SortNums(I) := 0;
end loop;
SortValues(1) := T(1);
NumSorts := 1;
SortNums(1) := 1;
for I in 2..L loop
    Found := False;
    for J in 1..NumSorts loop
        if T(I) = SortValues(J) then
            SortNums(J) := SortNums(J) + 1;
            Found := True;
            end if;
    end loop;
    if not Found then
        NumSorts := NumSorts + 1;
        SortValues(NumSorts) := T(I);
        SortNums(NumSorts) := 1;
    end if;
end loop;

-- Becomes Profile[9]
-- Compute Profile[3], Number of Sort Groups of Size One.
NumOneSorts := 0;
for I in 1..NumSorts loop
    if SortNums(I) = 1 then
        NumOneSorts := NumOneSorts + 1;
    end if;
end loop;
Result_Count := Result_Count + 1;
Result(Result_Count) := NumOneSorts;

-- Becomes Profile[10..N]
-- Compute Profile[4..N], Sequence of Sizes of the Sort Groups that
-- Have Size Greater than One.
for I in 0..L-2 loop
for J in 1..NumSorts loop
    if SortNums(J) = L-I then
        Result_Count := Result_Count + 1;
        Result(Result_Count) := L-I;
    end if;
end loop;

-- Terminate the ArrayProfile
Result(Result_Count+1) := PROFILE_TERMINATOR;
return Result;
end computeArrayProfile;

procedure computeArrayProfileWithGenerics(
    T: in Signature;
    ProfileValues: out ArrayProfiles;
    NumProfiles: out Natural) is
    I, G, J, K: Integer;
    L: SignatureLengthRange;
    NewSig: Signature;
    NumGenerics: Integer;
    NumDiffGenerics: Integer;
    Found: Boolean;
    Valj: Integer;
    GenericPos: array (SignatureLengthRange) of Integer;
    ProfileVal: ArrayProfile;
begin
    NumGenerics := 0;
    NumProfiles := 0;
    Valj:=0;
    NumDiffGenerics := 0;
    G := 0;
    J := 0;
    K := 0;

    -- set L to number of elements in T
    I := 1;
    while (T(I) /= 0 and I <= MAX_SIG_LENGTH) loop
        I := I + 1;
    end loop;
    L := I - 1;

    for I in 1..L loop
        if T(I) < 0 then
            if T(I) < NumDiffGenerics then
                NumDiffGenerics := T(I);
            end if;
            NumGenerics := NumGenerics + 1;
            GenericPos(NumGenerics) := I;
        end if;
    end loop;

    NumDiffGenerics := -1 * NumDiffGenerics;
    if NumGenerics = 0 then
        NumProfiles := 1;
        ProfileValues(1) := computeArrayProfile(T);
    else
        for G in 1..NumDiffGenerics loop
            for I in 1..L loop
                if T(GenericPos(I)) >= -1 * G then
                    NewSig(GenericPos(I)) := T(J);
                end if;
            end loop;
        end for;
    end if;

    -- These following lines are good for debugging.
    -- They print out all the combinations of signatures computed
    Valj:= printSignature(NewSig);
    New_Line;
end computeArrayProfileWithGenerics;
ProfileVal := computeArrayProfile(NewSig);
if NumProfiles = 0 then
    NumProfiles := 1;
    ProfileValues(l) := ProfileVal;
else
    Found := False;
    for K in 1..NumProfiles loop
        if ProfileValues(K) = ProfileVal then
            Found := True;
            end if;
        end loop;
    end if;
end loop;
if not Found then
    NumProfiles := NumProfiles + 1;
    ProfileValues(NumProfiles) := ProfileVal;
    end if;
end if;
end loop;
end loop;
end if;
end computeArrayProfileWithGenerics;

I. PROFILE_FILTER_PKG.ADS

------------- -----------------------------------------------
-- Package Spec: profile_filter
------------- -----------------------------------------------

with haase_diagram; use haase_diagram;
with candidate_types; use candidate_types;
with profile_types; use profile_types;

package profile_filter_pkg is

    function findCandidates(query_profile: in ComponentProfile;
           the_haase_diagram: in HaaseDiagram) return CandidateSet;

end profile_filter_pkg;

J. PROFILE_FILTER_PKG.G

------------- -----------------------------------------------
-- Package Body: profile_filter
------------- -----------------------------------------------

with haase_diagram; use haase_diagram;
with candidate_types; use candidate_types;
with component_id_types; use component_id_types;

package body profile_filter_pkg is

    -- Function: findCandidates
    -- Description: for each profile in query_profile start at the base-node
    -- that represents that profile and perform a depth-first
    -- search on the haase-diagram. At each node calculate the
    -- profile rank, create a Candidate with that rank and the
    -- components in that node, and add it to return_val.
    function findCandidates(query_profile: in ComponentProfile;
           the_haase_diagram: in HaaseDiagram) return CandidateSet is
        return_val: CandidateSet;
        base_node: HaaseNode;
        base_node_key: ComponentProfile;
        num_matches: natural;
        i, j: natural;

        procedure DFSFW(hn: in HaaseNode) is
            temp_candidate: Candidate;
begin
  -- count the number of profiles in the node that
  -- are also in the query
  num_matches := 0;
  i := 1;
  j := 1;
  while i <= profile_id_sequence_pkg.length(query_profile) and
    j <= profile_id_sequence_pkg.length(hn.key) loop
    if profile_id_sequence_pkg.fetch(query_profile, i) =
      profile_id_sequence_pkg.fetch(hn.key, j) then
      num_matches := num_matches + 1;
      i := i + 1;
      j := j + 1;
    elsif profileIDLessThan(profile_id_sequence_pkg.fetch(query_profile, i),
      profile_id_sequence_pkg.fetch(hn.key, j)) then
      i := i + 1;
    else
      j := j + 1;
    end if;
  end loop;

  -- add the node's components to return val
  foreach((comp_id: ComponentID), component_id_set_pkg.scan,
    (hn.components),
    temp_candidate := newCandidate;
  temp_candidate.profile_rank :=
    float(num_matches) / float(profile_id_sequence_pkg.length(query_profile));
  temp_candidate.component_id := comp_id;
  candidate_set_pkg.add(temp_candidate, return_val);
}

  -- recursively call DFSFW on each child
  foreach((child: ComponentProfile), component_profile_set_pkg.scan,
    (hn.children),
    DFSFW(haase_node_map_pkg.fetch(the_haase_diagram, child));
end DFSFW;
begin
  return_val := candidate_set_pkg.empty;
  foreach((p_id: ProfileID), profile_id_sequence_pkg.scan, (query_profile),
    base_node_key := profile_id_sequence_pkg.empty;
    addProfileID(p_id, base_node_key);
    if haase_node_map_pkg.member(base_node_key, the_haase_diagram) then
      base_node :=
        haase_node_map_pkg.fetch(the_haase_diagram, base_node_key);
      DFSFW(base_node);
      end if;
  }

  return return_val;
end findCandidates;
end profile_filter_pkg;

K. PROFILE_TYPES.ADS

package profile_types is
  procedure myIntPut(i: integer);
-- Profile
--
-- package int_sequence_pkg is new generic_sequence_pkg(
t => integer, average_size => 4);
-- subtype Profile is int_sequence_pkg.sequence;

-- function profileEqual is new int_sequence_pkg.generic_equal(eq => "=");
-- function profileLessThan is new int_sequence_pkg.generic_less_than("<" => "<");
-- procedure profilePut is new int_sequence_pkg.generic_put(put => gnat.io.put);
-- procedure profileFilePut is new int_sequence_pkg.generic_put(put => myIntPut);

subtype Profile is long_long_integer;

function profileEqual(pi, p2: Profile) return boolean;
function profileLessThan(pi, p2: Profile) return boolean;
procedure profilePut(p: Profile);
procedure profileFilePut(p: Profile);
procedure profileGet(p: in out Profile);

-- ProfileID
--
-- subtype ProfileID is integer;

function profileIDLessThan(pi, p2: ProfileID) return boolean;
procedure profileIDPut(p_id: ProfileID);
procedure profileIDFilePut(p_id: ProfileID);
procedure profileIDGet(p_id: in out ProfileID);

-- ProfileLookupTable
--
DEFAULT_PROFILE_ID: constant := -1;
package profile_lookup_table_pkg is new ordered_map_pkg(
key => Profile,
result => ProfileID,
eq_key => profileEqual,
eq_res => "=",
"<" => profileLessThan);
subtype ProfileLookupTable is profile_lookup_table_pkg.map;

procedure profileLookupTablePut is new profile_lookup_table_pkg.generic_put(
key_put => profilePut, res_put => profileIDPut);
procedure profileLookupTableGet is new profile_lookup_table_pkg.generic_input(
key_input => profileGet, res_input => profileIDGet);
procedure profileLookupTableFilePut is new profile_lookup_table_pkg.generic_file_put(
key_put => profilePut, res_put => profileIDPut);

-- ComponentProfile
--
-- Note: should use addProfileID to add a profile id to the ComponentProfile.
-- addProfileID keeps the ComponentProfile sorted which is important
-- for equality and subbag (multiset subset) testing.

package profile_id_sequence_pkg is new generic_sequence_pkg(
t => ProfileID, average_size => 4);
subtype ComponentProfile is profile_id_sequence_pkg.sequence;

function componentProfileEqual is
new profile_id_sequence_pkg.generic_equal(eq => "=");
function componentProfileMember is
new profile_id_sequence_pkg.generic_member(eq => "=");
procedure componentProfileRemove is
new profile_id_sequence_pkg.generic_remove(eq => "=");
function componentProfileSort is
new profile_id_sequence_pkg.generic_sort("<" => "<");
function componentProfileLessThan is
    new profile_id_sequence_pkg.generic_less_than("<" => profileIDLessThan);
procedure componentProfilePut is
    new profile_id_sequence_pkg.generic_put(put => profileIDPut);
procedure componentProfileFilePut is
    new profile_id_sequence_pkg.generic_file_put(put => profileIDFilePut);
procedure componentProfileGet is
    new profile_id_sequence_pkg.generic_input(input => profileIDGet);
function subbag is
    new profile_id_sequence_pkg.generic_subsequence(eq => "=");
package component_profile_set_pkg'g is new generic_set_pkg(t => ComponentProfile, eq => componentProfileEqual, average_size => 8);
subtype ComponentProfileSet is component_profile_set_pkg.set;
procedure componentProfileSetPut is
    new component_profile_set_pkg.generic_put(put => componentProfilePut);
procedure componentProfileSetGet is
    new component_profile_set_pkg.generic_input(input => componentProfileGet);
procedure addProfileID(p_id: in ProfileID; cp: in out ComponentProfile) is
begin
    profile_id_sequence_pkg.add(p_id, cp);
    cp := componentProfileSort(cp);
end addProfileID;
procedure addProfiles(new_profiles: in ComponentProfile; target: in out ComponentProfile) is
begin
    componentProfileSetPut(target);
    addProfiles(new_profiles, target);
end addProfiles;
end profile_types;

L. PROFILE TYPES.G

-----------
-- Package Body: profile_types
-----------

with text_io;
with ada.long_long_integer_text_io;
with ada.integer_text_io;
with software_base;
with sb_utils;
package body profile_types is
    package Int_IO is new Text_IO.Integer_IO (Num => Integer);
    --
    -- Procedure: myIntPut
    --
    procedure myIntPut(i: integer) is
    begin
        ada.integer_text_io.put(i, 0);
    end myIntPut;
    --
    -- Procedure: addProfileID
    --
    -- Description: adds a ProfileID to a ComponentProfile by adding the
    -- ProfileID to the sequence then sorting the sequence.
    --
    procedure addProfileID(p_id: in ProfileID; cp: in out ComponentProfile) is
    begin
        profile_id_sequence_pkg.add(p_id, cp);
        cp := componentProfileSort(cp);
    end addProfileID;
    --
    -- Procedure: addProfiles
    --
    -- Description: appends the profiles from new_profiles to target then
    -- sorts target.
-- procedure addProfiles(newProfiles: in ComponentProfile;
  target: in out ComponentProfile) is
begin
  target := profile_id_sequence_pkg.append(target, new_profiles);
  target := componentProfileSort(target);
end addProfiles;

-- -- Function: profileEqual
--
function profileEqual(pi, p2: Profile) return boolean is
begin
  return pi = p2;
end profileEqual;

-- -- Function: profileLessThan
--
function profileLessThan(pi, p2: Profile) return boolean is
begin
  return pi < p2;
end profileLessThan;

-- -- Function: profilePut
--
procedure profilePut(p: Profile) is
begin
  ada.long_long_integer_text_io.put(p, 0);
end profilePut;

-- -- Function: profileGet
--
procedure profileGet(p: in out Profile) is
begin
  ada.long_long_integer_text_io.get(p);
end profileGet;

-- -- Function: profileFilePut
--
procedure profileFilePut(p: Profile) is
begin
  profilePut(p);
end profileFilePut;

-- -- Function: profileIDLessThan
--
function profileIDLessThan(p1, p2: ProfileID) return boolean is
begin
  return software_base.getProfile(p1) < software_base.getProfile(p2);
end profileIDLessThan;

-- -- Procedure: profileIDPut
--
procedure profileIDPut(p_id: ProfileID) is
begin
  ada.integer_text_io.put(p_id, 0);
end profileIDPut;

-- -- Procedure: profileIDGet
--
procedure profileIDGet(p_id: in out ProfileID) is
  j: integer;
begin
  sb_utils.IntGet(j);
  p_id := j;
end profileIDGet;
-- Function: profileIDFilePut
--
procedure profileIDFilePut(p_id: ProfileID) is
begin
  profileIDPut(p_id);
end profileIDFilePut;

--
-- Function: createProfileLookupTable
--
function createProfileLookupTable return ProfileLookupTable is
begin
  return profile_lookup_table_pkg.create(0);
end createProfileLookupTable;

end profile_types;

M. PSDL_PROFILE.ADS

-------------------------------------------------------------------------------
-- Package Spec: psl_profile
--
-- This package contains functions and types that support the collection
-- of operation profiles from a component specified in PSDL.
-------------------------------------------------------------------------------
with text_io;
with generic_sequence_pkg;
with generic_map_pkg;
with generic_set_pkg;
with ordered_set_pkg;
with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
with psdl_component_pkg; use psdl_component_pkg;
with profile_types; use profile_types;
package psdl_profile is
  -- Types
  --
  -- OpWithProfile
  --
  type OpWithProfile is record
    op: operator;
    op_profile: ProfileID;
  end record;

  function opWithProfileEqual(owpl: in OpWithProfile; owp2: in OpWithProfile) return boolean;
  function opWithProfileLessThan(owpl: in OpWithProfile; owp2: in OpWithProfile) return boolean;
  procedure opWithProfilePut(owp: in OpWithProfile);

  --
  -- OpWithProfileSeq
  --
  -- Note: should use addOpWithProfile to add an OpWithProfile to the sequence.
  --    addOpWithProfile keeps the sequence sorted.
  --
  package owp_sequence_pkg is new generic_sequence_pkg(
    t => OpWithProfile, average_size => 4);
  subtype OpWithProfileSeq is owp_sequence_pkg.sequence;

  function opWithProfileSeqEqual is
new owp_sequence_pkg.generic_equal(eq => opWithProfileEqual);

function opWithProfileSeqMember is
  new owp_sequence_pkg.generic_member(eq => opWithProfileEqual);

procedure opWithProfileSeqRemove is
  new owp_sequence_pkg.generic_remove(eq => opWithProfileEqual);

function opWithProfileSeqSort is
  new owp_sequence_pkg.generic_sort("<" => opWithProfileLessThan);

procedure opWithProfileSeqPut is
  new owp_sequence_pkg.generic_put(put => opWithProfilePut);

procedure opWithProfileSeqPut is
  new owp_sequence_pkg.generic_put(put => opWithProfilePut);

procedure addOpWithProfile(owp: in OpWithProfile;
  owp_seq: in out OpWithProfileSeq);

-- OpWithProfileSet
package owp_set_pkg is new ordered_set_pkg(
  t => OpWithProfile, eq => opWithProfileEqual,
  "<" => opWithProfileLessThan);
subtype OpWithProfileSet is owp_set_pkg.set;

procedure opWithProfileSetPut is
  new owp_set_pkg.generic_put(put => opWithProfilePut);

procedure OpWithProfileSetPrint(owp_set: in OpWithProfileSet);

-- GenericsMap
package generics_map_pkg is new generic_map_pkg(
  key => psdl_id,
  result => psdl_id,
  eq_key => eq,
  eq_res => eq,
  average_size => 8);
subtype GenericsMap is generics_map_pkg.map;

procedure psdl_idPut(the_id: in psdl_id);
procedure psdl_idGet(the_id: in out psdl_id);

procedure genericsMapPut is new generics_map_pkg.generic_put(
  key_put => psdl_idPut, res_put => psdl_idPut);

procedure genericsMapGet is new generics_map_pkg.generic_input(
  key_input => psdl_idGet, res_input => psdl_idGet);

-- GenericsMapSet
package generics_map_set_pkg is new generic_set_pkg(
  t => GenericsMap, eq => generics_map_pkg.equal);
subtype GenericsMapSet is generics_map_set_pkg.set;

procedure genericsMapSetPut is
  new generics_map_set_pkg.generic_put(put => genericsMapPut);

-- Functions

function getGenericsMaps(filename: in string) return GenericsMapSet;

function getComponentProfile(filename: in string;
  generics_mapping: in GenericsMap) return ComponentProfile;

function getOpsWithProfiles(filename: in string;
  generics_mapping: in GenericsMap) return OpWithProfileSeq;

function getOpsWithProfiles(filename: in string;
  generics_mapping: in GenericsMap) return OpWithProfileSet;

end psdl_profile;

N. PSDL_PROFILE.G

package body psdl_profile is

package signature_seq_pkg is new generic_sequence_pkg(
  t => Signature, average_size => 2);
subtype SignatureSequence is signature_seq_pkg.sequence;

-- Function: opWithProfileEqual

function opWithProfileEqual(owpl: in OpWithProfile; owp2: in OpWithProfile) return boolean is
begin
  -- if not profileEqual(owpl.op_profile, owp2.op_profile) then
  if owpl.op_profile /= owp2.op_profile then
    return false;
  end if;
  return eq(owpl.op, owp2.op);
end opWithProfileEqual;

-- Function: opWithProfileLessThan

function opWithProfileLessThan(owpl: in OpWithProfile; owp2: in OpWithProfile) return boolean is
begin
  return profileLessThan(owpl.op_profile, owp2.op_profile);
end opWithProfileLessThan;
return profileLessThan(software_base.getProfile(owpl.op_profile), software_base.getProfile(owp2.op_profile));
end opWithProfileLessThan;

-- Function: opWithProfilePut

procedure opWithProfilePut(owp: in OpWithProfile) is
begin
  put("(");
  put(convert(name(owp.op)));
  put("":");
  foreach((the_id: psdl_id; the_tn: type_name),
    type_declaration_pkg.scan, (inputs(owp.op)),
    put(convert(the_tn.name));
  put(" ");
  put("--> ");
  foreach((the_id: psdl_id; the_tn: type_name),
    type_declaration_pkg.scan, (outputs(owp.op)),
    put(convert(the_tn.name));
  put(" ");
  profilePut(software_base.getProfile(owp.op_profile));
  put(")");
end opWithProfilePut;

-- Function: opWithProfileSeqPrint

procedure opWithProfileSeqPrint(owp_seq: in OpWithProfileSeq) is
begin
  foreach((owp: OpWithProfile), owp_sequence_pkg.scan, (owp_seq),
    put(convert(name(owp.op)));
    put(" ");
  foreach((the_id: psdl_id; the_tn: type_name),
    type_declaration_pkg.scan, (inputs(owp.op)),
    put(convert(the_tn.name));
    put(" ");
  put("--> ");
  foreach((the_id: psdl_id; the_tn: type_name),
    type_declaration_pkg.scan, (outputs(owp.op)),
    put(convert(the_tn.name));
    put(" ");
  put(" ");
  profilePut(software_base.getProfile(owp.op_profile));
  new_line;
end opWithProfileSeqPrint;

-- Function: opWithProfileSetPrint

procedure opWithProfileSetPrint(owp_set: in OpWithProfileSet) is
begin
  foreach((owp: OpWithProfile), owp_set_pkg.scan, (owp_set),
    put(convert(name(owp.op)));
    put(" ");
  foreach((the_id: psdl_id; the_tn: type_name),
    type_declaration_pkg.scan, (inputs(owp.op)),
    put(convert(the_tn.name));
    put(" ");
  put("--> ");
  foreach((the_id: psdl_id; the_tn: type_name),
    type_declaration_pkg.scan, (outputs(owp.op)),
    put(convert(the_tn.name));
    put(" ");
  new_line;
  profilePut(software_base.getProfile(owp.op_profile));
end opWithProfileSetPrint;
new_line;
end opWithProfileSetPrint;

-- Function: addOpWithProfile
procedure addOpWithProfile(owp: in OpWithProfile;
                          owp_seq: in out OpWithProfileSeq) is
begin
  owp_sequence_pkg.add(owp, owp_seq);
  owp_seq := opWithProfileSeqSort(owp_seq);
end addOpWithProfile;

-- Function: createNumericSignatures
-- Description: helper function to create numeric signatures for
-- an operator.
function createNumericSignatures(op: in operator;
                                  generics_mapping: GenericsMap; type_id: psdl_id)
return SignatureSequence is
  type_map_pkg is
    new generic_map_pkg(
      key => type_name,
      result => integer,
      eq_key => equal,
      eq_res => "=",
      average_size => 2);
  subtype type_map is type_map_pkg.map;

  the_inputs: type_declaration := inputs(op);
  the_outputs: type_declaration := outputs(op);
  the_type_map: type_map;
  i, t: natural;
  sort_group_num: integer;
  gen_set: psdl_id_set_subtype_pkg.psdl_id_set;
  temp_signature: Signature;
  temp_tn: type_name;
  return_val: SignatureSequence;
  type_occurrence_count: natural;
  bool_count, char_count, string_count, int_count, float_count: natural;

  procedure update_additional_counts(the_tn: type_name) is
    begin
      if eq(temp_tn.name, type_id) then
        type_occurrence_count := type_occurrence_count + 1;
      elsif same_sort_group(the_tn, boolean_type) then
        bool_count := bool_count + 1;
      elsif same_sort_group(the_tn, character_type) then
        char_count := char_count + 1;
      elsif same_sort_group(the_tn, string_type) then
string_count := string_count + 1;
elsif same_sort_group(the_tn, integer_type) then
  int_count := int_count + 1;
elsif same_sort_group(the_tn, float_type) then
  float_count := float_count + 1;
end if;
end;

begin

  type_map_pkg.create(0, the_type_map);
  -- for each output
  foreach((o_id: psdl_id; o_tn: type_name),
          type_declaration_pkg.scan, (the_outputs),
          type_map_pkg.recycle(the_type_map);
    t := 0;
    i := 0;
    type_occurrence_count := 0;
    bool_count := 0;
    char_count := 0;
    string_count := 0;
    int_count := 0;
    float_count := 0;
    -- for each input
    foreach((i_id: psdl_id; i_tn: type_name),
            type_declaration_pkg.scan, (the_inputs),
            type_occurrence_count := 0;
    if generics_map_pkg.member(i_tn.name, generics_mapping) then
      temp_tn := create(generics_map_pkg.fetch(generics_mapping, i_tn.name),
                         psdl_id_sequence_pkg.empty,
                         type_declaration_pkg.create(null_type));
    else
      -- could probably use i_tn as is rather than create
      -- a copy but we're being safe in case i_tn has some
      -- residue in its formals and gen_pars
      temp_tn := create(i_tn.name,
                         psdl_id_sequence_pkg.empty,
                         type_declaration_pkg.create(null_type));
    end if;
    update_additional_counts(temp_tn);
    if not type_map_pkg.member(temp_tn, the_type_map) then
      sort_group_num := getSortGroupNum(temp_tn, the_type_map);
      if sort_group_num = 0 then
        t := t + 1;
        type_map_pkg.bind(temp_tn, t, the_type_map);
      end if;
    end if;
    -- add the input's sort group number
    i := i + 1;
    temp_signature(i) := getSortGroupNum(temp_tn, the_type_map);
  )

  -- handle the output
  -- check if type is a generic type or a regular type
  if generics_map_pkg.member(o_tn.name, generics_mapping) then
    temp_tn := create(generics_map_pkg.fetch(generics_mapping, o_tn.name),
                      psdl_id_sequence_pkg.empty,
                      type_declaration_pkg.create(null_type));
  else
    -- could probably use o_tn as is rather than create
    -- a copy but we're being safe in case o_tn has some
    -- residue in its formals and gen_pars
    temp_tn := create(o_tn.name,
                      psdl_id_sequence_pkg.empty,
                      type_occurrence_count := 0;
    -- ensure that type_map_pkg contains the type
    if not type_map_pkg.member(temp_tn, the_type_map) then
      sort_group_num := getSortGroupNum(temp_tn, the_type_map);
      if sort_group_num = 0 then
        t := t + 1;
        type_map_pkg.bind(temp_tn, t, the_type_map);
      end if;
    end if;
    -- add the output's sort group number
    i := i + 1;
    temp_signature(i) := getSortGroupNum(temp_tn, the_type_map);
  )
case declaration_pkg.create(null_type));
end if;

update_additional_counts(temp_tn);
-- if the type isn't in the map yet then put it in
if not type_map_pkg.member(temp_tn, the_type_map) then
  sort_group_num := getSortGroupNum(temp_tn, the_type_map);
  if sort_group_num = 0 then
    t := t + 1;
    type_map_pkg.bind(temp_tn, t, the_type_map);
  end if;
end if;
-- add the output's sort group number
i := i + 1;
temp_signature(i) := getSortGroupNum(temp_tn, the_type_map);
-- mark end of signature
i := i + 1;
temp_signature(i) := 0;
-- add the type_occurrence_count to the signature
i := i + 1;
temp_signature(i) := type_occurrence_count;
-- add basic type counts in
i := i + 1;
temp_signature(i) := bool_count;
i := i + 1;
temp_signature(i) := char_count;
i := i + 1;
temp_signature(i) := string_count;
i := i + 1;
temp_signature(i) := int_count;
i := i + 1;
temp_signature(i) := float_count;
i := i + 1;
temp_signature(i) := 0;
-- add the signature to the sequence of signatures
signature_seq_pkg.add(temp_signature, return_val);
)
return return_val;
end createNumericSignatures;

-- Function: getOperatorProfiles
-- Description: helper function to collect the profiles for
-- an operator. A ComponentProfile (sequence of
-- profiles) is used because if an operator has
-- more than one output it is treated as if there
-- is a separate operator for each output.
function getOperatorProfiles(op: operator;
  generics_mapping: in GenericsMap; type_id: psl_id)
return ComponentProfile is

  return_val: ComponentProfile;
  numeric_sigs: SignatureSequence;
begin
  -- convert the operator's signature to numeric signatures
  -- (see the comments in the specification of profile_calc)
  numeric_sigs := createNumericSignatures(op, generics_mapping, type_id);

  -- compute the profile for each signature
  foreach((sig: Signature), signature_seq_pkg.scan, (numeric_sigs),
    addProfileID(software_base.getProfileID(computeProfile(sig)),
    return_val);

  return return_val;
end getOperatorProfiles;
Function: getComponentProfile

Description: this function will return the ComponentProfile for a component specified in PSDL in the PSDL file filename.

function getComponentProfile(filename: in string;
generics_mapping: in GenericsMap) return ComponentProfile is
the_file: file_type;
the_prog: psdl_program;
return_val: ComponentProfile;
begins
-- parse the psdl file to create a psdl_program
open(the_file, IN_FILE, filename);
assign(the_prog, psdl_program_pkg.empty_psdl_program);
psdl_io.get(the_file, the_prog);
close(the_file);

-- if the program contains more than one component
-- then just get the first one since the program
-- is only supposed to have one (a requirement of
-- this implementation)
foreach((c_id: psdl_id; c: psdl_component),
   psdl_program_map_pkg.scan, (the_prog),
   -- if the component is a single operator then just
   -- get the profile for that operator
   if component_category(c) = psdl_operator then
      addProfiles(getOperatorProfiles(c, generics_mapping, empty),
      return_val);
   -- otherwise the component is a type so get the profiles
   -- for each of its operators
   else
      foreach((id: psdl_id; o: operator),
         operation_map_pkg.scan, (operations(c)),
         addProfiles(getOperatorProfiles(o, generics_mapping,
         psdl_id_pkg.Upper_To_Lower(c_id)), return_val);
   end if;
-- TODO: need to break out of this loop so that only the
-- first component is processed.
return return_val;
end getComponentProfile;

Function: splitOp

Description: helper function to split an operator with more
than one output into a sequence of operators
where each operator has one of the outputs.
When splitting, instances of the operator's generic
types in the inputs and the outputs are converted to
their mapped types according to the generics_mapping.
Each split operator's profile is then calculated.

function splitOp(op: operator; generics_mapping: in GenericsMap;
type_id: psdl_id) return OpWithProfileSeq is
return_val: OpWithProfileSeq;
temp_owp: OpWithProfile;
temp_output_name: psdl_id;
temp_output_type: type_name;
numeric_sigs: SignatureSequence;

begin
  -- for each output
  foreach((o_id: psdl_id; o_tn: type_name),
    type_declaration_pkg.scan, (outputs(op)),
    -- make a copy of op but with only the current output
    temp_owp.op := make_atomic_operator(
      psdl_name => name(op),
      ada_name => ada_name(op),
      gen_par => generic_parameters(op),
      keywords => keywords(op),
      axioms => axioms(op),
      state => states(op));
    -- add the inputs
    foreach((i_id: psdl_id; i_tn: type_name),
      type_declaration_pkg.scan, (inputs(op)),
      if generics_map_pkg.member(i_tn.name, generics_mapping) then
        add_input(i_id, create!
          generics_map_pkg.fetch(generics_mapping, i_tn.name),
          psdlid_sequence_pkg.empty,
          type_declaration_pkg.create(null_type)),
          temp_owp.op);
      else
        add_input(i_id, i_tn, temp_owp.op);
      end if;
    )
    -- add the output
    if generics_map_pkg.member(o_tn.name, generics_mapping) then
      add_output(o_id, create(
        generics_map_pkg.fetch(generics_mapping, o_tn.name),
        psdl_id_sequence_pkg.empty,
        type_declaration_pkg.create(null_type)),
        temp_owp.op);
    else
      add_output(o_id, o_tn, temp_owp.op);
    end if;
    -- Convert the new operator's signature to numeric signatures
    numeric_sigs :=
      createNumericSignatures(temp_owp.op, generics_map_pkg.create(empty), type_id);
    -- compute the new operator's profile
    temp_owp.op_profile := software_base.getProfileID(computeProfile(
      signature_seq_pkg.fetch(numeric_sigs, 1)));
    -- add the new operator-with-profile to return_val
    addOpWithProfile(temp_owp, return_val);
  )
  return return_val;
end splitOp;

-- Function: getOpsWithProfiles
-- Description: constructs a sequence of OpWithProfiles (a PSDL operator
-- and its corresponding profile) representing the operators
-- in the PSDL component specified in filename.
function getOpsWithProfiles(filename: in string;
generics_mapping: in GenericsMap) return OpWithProfileSeq is

the_file: file_type;
the_prog: psdl_program;
return_val, foo: OpWithProfileSeq := owp_sequence_pkg.empty;

begin
  -- parse the psdl file to create a psdl_program
  open(the_file, IN_FILE, filename);
  assign(the_prog, psdl_program_pkg.empty_psdl_program);
  psdl_io.get(the_file, the_prog);
  close(the_file);

  -- if the program contains more than one component
  -- then just get the first one since the program
  -- is only supposed to have one (a requirement of
  -- this implementation). Generic maps need a method
  -- that allows the user to fetch a single mapping
  -- in the map.
  foreach((c_id: psdl_id; c: psdl_component),
    psdl_program_map_pkg.scan, (the_prog),
    -- if the component is a single operator then just
    -- get that operator
    if component_category(c) = psdl_operator then
      foreach((owp: OpWithProfile), owp_sequence_pkg.scan,
        (splitOp(c, generics_mapping, empty)),
        addOpWithProfile(owp, return_val);
    )
    -- otherwise the component is a type so get
    -- each of its operators
    else
      foreach((id: psdl_id; o: operator),
        operation_map_pkg.scan, (operations(c)),
        foreach((owp: OpWithProfile), owp_sequence_pkg.scan,
          (splitOp(o, generics_mapping,
            psdl_id_pkg.Upper_To_Lower(c_id))),
          addOpWithProfile(owp, return_val);
        )
        -- in the above statement we
        -- temporally pass the generic parameters for the whole
        -- type, c. Should really just pass the generic
        -- parameters for the operation, o, only. This will
        -- happen when generics get reworked.
      )
    end if;

  -- TODO: need to break out of this loop so that only the
  -- first component is processed.
  )

  return return_val;
end getOpsWithProfiles;

--
-- Function: getOpsWithProfiles
--
-- Description: constructs a set of OpWithProfiles (a PSDL operator
-- and its corresponding profile) representing the operators
-- in the PSDL component specified in filename.
--
function getOpsWithProfiles(filename: in string;
  generics_mapping: in GenericsMap) return OpWithProfileSet is

  the_file: file_type;
  the_prog: psdl_program;
  return_val: OpWithProfileSet;

begin
  -- parse the psdl file to create a psdl_program

  open(the_file, IN_FILE, filename);
  assign(the_prog, psdl_program_pkg.empty_psdl_program);
  psdl_io.get(the_file, the_prog);
  close(the_file);

  -- if the program contains more than one component
  -- then just get the first one since the program
  -- is only supposed to have one (a requirement of
  -- this implementation). Generic maps need a method
  -- that allows the user to fetch a single mapping
  -- in the map.
  foreach((c_id: psdl_id; c: psdl_component),
    psdl_program_map_pkg.scan, (the_prog),
    -- if the component is a single operator then just
    -- get that operator
    if component_category(c) = psdl_operator then
      foreach((owp: OpWithProfile), owp_sequence_pkg.scan,
        (splitOp(c, generics_mapping, empty)),
        addOpWithProfile(owp, return_val);
    )
    -- otherwise the component is a type so get
    -- each of its operators
    else
      foreach((id: psdl_id; o: operator),
        operation_map_pkg.scan, (operations(c)),
        foreach((owp: OpWithProfile), owp_sequence_pkg.scan,
          (splitOp(o, generics_mapping,
            psdl_id_pkg.Upper_To_Lower(c_id))),
          addOpWithProfile(owp, return_val);
        )
        -- in the above statement we
        -- temporally pass the generic parameters for the whole
        -- type, c. Should really just pass the generic
        -- parameters for the operation, o, only. This will
        -- happen when generics get reworked.
      )
    end if;

  -- TODO: need to break out of this loop so that only the
  -- first component is processed.
  )

  return return_val;
end getOpsWithProfiles;
open(the_file, IN_FILE, filename);
assign(the_prog, psdl_program_pkg.empty_psdl_program);
psdl_io.get(the_file, the_prog);
close(the_file);

-- if the program contains more than one component
-- then just get the first one since the program
-- is only supposed to have one (a requirement of
-- this implementation). Generic maps need a method
-- that allows the user to fetch a single mapping
-- in the map.
foreach((c_id: psdl_id; c: psdl_component),
    psdl_program_map_pkg.scan, (the_prog),
    -- if the component is a single operator then just
    -- get that operator
    if component_category(c) = psdl_operator then
        foreach(owp: OpWithProfile), owp_sequence_pkg.scan,
            (splitOp(c, generics_mapping, empty)),
            owp_set_pkg.add(owp, return_val);
    else
        foreach((id: psdl_id; o: operator),
            operation_map_pkg.scan, (operations(c)),
                foreach(owp: OpWithProfile), owp_sequence_pkg.scan,
                    (splitOp(o, generics_mapping,
                        psdl_id_pkg.Opper_To_Lower(c_id))),
                    owp_set_pkg.add(owp, return_val);
        )
        -- in the above statement we
        -- temporally pass the generic parameters for the whole
        -- type, c. Should really just pass the generic
        -- parameters for the operation, o, only. This will
        -- happen when generics get reworked.
    end if;
    -- TODO: need to break out of this loop so that only the
    -- first component is processed.
) return return_val;
end getOpsWithProfiles;

-- Procedure: psdl_idPut
procedure psdl_idPut(the_id: in psdl_id) is
    put(convert(the_id));
end psdl_idPut;

-- Procedure: psdl_idGet
procedure psdl_idGet(the_id: in out psdl_id) is
    IO_buffer: string(1..256);
    last: integer;
begin
    sb_utils.get_char_word(IO_buffer, last);
    the_id := psdl_id(a_strings.to_a(IO_buffer(1..last))); end psdl_idGet;

-- Function: getGenericsMap
-- Description: generates all the possible mappings of generic types
-- to actual types for all the generic parameters in
-- the component specified in the PSDL file, filename.
See description of GenericsMap in psdl_profile.ads.

This is done by collecting all the types used in the
operations of the component (note we are only processing
type components, not operator components) into a set
and then performing the cross-product of this set with
the set of generic parameters.

function getGenericsMaps(filename: in string) return GenericsMapSet is

the_file: file_type;
the_prog: psdl_program;
return_val: GenericsMapSet;
gen_set: psdl_id_set;
type_set: psdl_id_set;
temp_map: GenericsMap;

cross_product(g_set, t_set: psdl_id_set; gens_map: GenericsMap) is

temp_set: psdl_id_set;
g: psdl_id;
local_map: GenericsMap;

begin
    generics_map_pkg.assign(local_map, gens_map);
    if psdl_id_set_pkg.size(g_set) > 0 then
        psdl_id_set_pkg.assign(temp_set, g_set);
        g := psdl_id_set_pkg.choose(g_set);
        foreach((the_type_id: psdl_id), psdl_id_set_pkg.scan, (t_set),
            generics_map_pkg.bind(g, the_type_id, local_map);
        psdl_id_set_pkg.remove(g, temp_set);
        cross_product(temp_set, t_set, local_map);
        generics_map_pkg.assign(local_map, gens_map);
    )
    generics_map_pkg.recycle(temp_map);
    else
        generics_map_set_pkg.add(local_map, return_val);
    end if;
end cross_product;

begin
    return_val := generics_map_set_pkg.empty;

    -- parse the psdl file to create a psdl_program
    open(the_file, IN_FILE, filename);
    assign(the_prog, psdl_program_pkg.empty_psdl_program);
    psdl_io.get(the_file, the_prog);
    close(the_file);

    -- if the program contains more than one component
    -- then just get the first one since the program
    -- is only supposed to have one (a requirement of
    -- this implementation).  Generic maps need a method
    -- that allows the user to fetch a single mapping
    -- in the map.
    foreach((c_id: psdl_id; c: psdl_component), psdl_program_map_pkg.scan, 
        (the_prog),
            -- collect the names of the generic parameters
            foreach((the_id: psdl_id; the_tn: type_name),
                type_declaration_pkg.scan, (generic_parameters(c)),
                if eq(psdl_id_pkg.Upper_To_Lower(the_tn.name),
                    convert("private_type")) then
                    psdl_id_set_pkg.add(psdl_id_pkg.Upper_To_Lower(the_tn.name),
                        gen_set);
                end if;
            )
        -- collect the types used in all the operators
        if component_category(c) = psdl_type then
            foreach((o_id: psdl_id; o: operator),
                operation_map_pkg.scan, (operations(c)),
                    -- inputs
                    foreach((the_id: psdl_id; the_tn: type_name),
                        type_declaration_pkg.scan, (inputs(o)),
                        psdl_id_set_pkg.add(163
psdl_id_pkg.Upper_To_Lower(the_tn.name), type_set);

-- outputs
foreach((the_id: psdl_id; the_tn: type_name),
type_declaration_pkg.scan, (outputs(o)),
psdl_id_set_pkg.add(
  psdl_id_pkg.Upper_To_Lower(the_tn.name), type_set);
)
end if;

-- TODO: need to break out of this loop so that only the
-- first component is processed.
}
generics_map_pkg.create(empty, temp_map);
cross_product(gen_set, type_set, temp_map);
return return_val;
end getGenericsMaps;
end psdl_profile;

O. SIG_MATCH.ADS

-- Package Spec: sig_match
with psdl_profile; use psdl_profile;
with sig_match_types; use sig_match_types;
package sig_match is
  procedure match_ops(query, candidate: in OpWithProfileSeq;
    root_sn: in out SigMatchNode);
  procedure sigMatchStatsReset;
  procedure sigMatchStatsPut(filename: string);
end sig_match;

P. SIG_MATCH.PLG

-- Package Body: sig_match
with text_io; use text_io;
with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
with psdl_component_pkg; use psdl_component_pkg;
with profile_types; use profile_types;
with psdl_profile; use psdl_profile;
with sig_match_types; use sig_match_types;
package body sig_match is
  failed_outputs: natural := 0;
  passed_outputs: natural := 0;
  failed_basics: natural := 0;
  passed_basics: natural := 0;
  duplicates: natural := 0;
  total_inputs: natural := 0;
  failed_inputs: natural := 0;

-- Function: get_basics
-- Description: removes any user-defined types from the inputs argument,
  thereby returning a type_declaration with predefined
function get_basics(inputs: in type_declaration) return type_declaration is
return_val: type_declaration;
begin
  type_declaration_pkg.assign(return_val, inputs);
  foreachf(the_id: psdl_id; the_tn: type_name), type_declaration_pkg.scan,
  (inputs),
    if not is_predefined(the_tn) then
      type_declaration_pkg.remove(the_id, return_val);
    end if;
  )
return return_val;
end get_basics;

function get_user_defined(inputs: in type_declaration) return type_declaration is
return_val: type_declaration;
begin
  type_declaration_pkg.assign(return_val, inputs);
  foreachf(the_id: psdl_id; the_tn: type_name), type_declaration_pkg.scan,
  (inputs),
    if is_predefined(the_tn) then
      type_declaration_pkg.remove(the_id, return_val);
    end if;
  )
return return_val;
end get_user_defined;

function match_basics(q_basics, c_basics: in type_declaration) return boolean is
the_q_basics: type_declaration;
the_c_basics: type_declaration;
new_q_basics: type_declaration;
new_c_basics: type_declaration;
found_match, found_c2, return_val: boolean;
begin
  type_declaration_pkg.assign(new_q_basics, q_basics);
  type_declaration_pkg.assign(new_c_basics, c_basics);

  -- cannot match if query has different number of basics then
  -- the candidate
  if type_declaration_pkg.size(q_basics) /=
    type_declaration_pkg.size(c_basics) then
    return false;
  end if;

  -- filter out the basics that match exactly
  type_declaration_pkg.assign(the_c_basics, new_c_basics);
  foreachf((q_id: psdl_id; q_tn: type_name), type_declaration_pkg.scan,
  (q_basics),
    if is_predefined(q_tn) then
      type_declaration_pkg.remove(q_id, return_val);
    end if;
  )
return return_val;
end match_basics;
foreach((c_id: psdl_id; c_tn: type_name), type_declaration_pkg.scan, (new_c_basics),
if not found_match then
  if equal(q_tn, c_tn) then
    type_declaration_pkg.remove(q_id, new_q_basics);
    type_declaration_pkg.remove(c_id, the_c_basics);
    found_match := true;
  end if;
end if;
-- TODO: would rather break out of the inner for loop when a
-- match is found rather than do this found_match stuff.
end if;

type_declaration_pkg.assign(new_c_basics, the_c_basics);

-- Filter out the remaining basics that can match to supertypes.
-- This is done by temporally mapping each query input type to a
-- supertype in the candidate that is closest in the partial ordering
-- of basic types.

type_declaration_pkg.assign(the_q_basics, new_q_basics);
foreach((q_id: psdl_id; q_tn: type_name), type_declaration_pkg.scan, (the_q_basics),
found_match := false;

foreach((c_id: psdl_id; c_tn: type_name), type_declaration_pkg.scan, (the_c_basics),
if not found_match then
  if subtype_of(q_tn, c_tn) then
    found_c2 := false;
    foreach((c2_id: psdl_id; c2_tn: type_name),
      type_declaration_pkg.scan, (the_c_basics),
    if not found_c2 then
      if not equal(c_tn, c2_tn) then
        if subtype_of(q_tn, c2_tn) and
          subtype_of(c2_tn, c_tn) then
          found_c2 := true;
        end if;
      end if;
    end if;
  end if;
end if;

if not found_c2 then
  type_declaration_pkg.remove(q_id, new_q_basics);
  type_declaration_pkg.remove(c_id, new_c_basics);
  found_match := true;
end if;
end if;
end if;
end if;

end if;

-- if there are any basics left over than match is not possible since
-- basics cannot be matched to non-basics
return_val := type_declaration_pkg.size(new_q_basics) = 0;

-- recycle local variables

type_declaration_pkg.recycle(new_q_basics);
type_declaration_pkg.recycle(new_c_basics);
type_declaration_pkg.recycle(the_q_basics);
type_declaration_pkg.recycle(the_c_basics);

return return_val;
end match basics;

-- Procedure: match_outputs
--
-- Description: This function serves two purposes: 1. to determine if
the outputs of the matched operations can match, and

2. if they can match, add the type mappings to $sn.V.TM$.

```
procedure match_outputs(sn: in out SigMatchNode; success: out boolean) is
    q_output_type, c_output_type: type_name;
begin
    success := true;
    foreach((q_op: operator; c_op: operator), op_map_pkg.scan, (sn.V.OM),
        if success then
            -- get q_op's one-and-only output type
            q_output_type := type_declaration_pkg.res_set_pkg.choose(
                type_declaration_pkg.map_range(outputs(q_op)));
            -- get c_op's one-and-only output type
            c_output_type := type_declaration_pkg.res_set_pkg.choose(
                type_declaration_pkg.map_range(outputs(c_op)));
            if is_predefined(q_output_type) or
                is_predefined(c_output_type) then
                if not subtype_of(c_output_type, q_output_type) then
                    success := false;
                end if;
            elsif type_map_pkg.member(q_output_type, sn.V.TM) then
                if not equal(c_output_type,
                    type_map_pkg.fetch(sn.V.TM, q_output_type)) then
                    success := false;
                end if;
            else
                type_map_pkg.bind(q_output_type, c_output_type, sn.V.TM);
            end if;
        end if;
    end foreach;
end match_outputs;
```

---

---

```
procedure match_inputs(root_sn: in out SigMatchNode; success: out boolean) is
    procedure match(q_inputs, c_inputs: in type_declaration;
        root_sn: in out SigMatchNode; success: out boolean) is
    new_q_inputs, new_c_inputs: type_declaration;
    temp_q_inputs, temp_c_inputs: type_declaration;
    ci: type_name;
    temp_sn: SigMatchNodePtr;
    temp_id: psdl_id;
    found_temp_id: boolean;
    got_first_qi: boolean;
    return_val: SigMatchNode;
begin
    return_val := createSigMatchNode;
    sigMatchNodeAssign(return_val, root_sn);
    type_declaration_pkg.assign(new_q_inputs, q_inputs);
    type_declaration_pkg.assign(new_c_inputs, c_inputs);
    success := true;
    foreach((q_id: psdl_id; qi: type_name),
        type_declaration_pkg.scan, (q_inputs),
        if success then
            if not type_map_pkg.member(qi, root_sn.V.TM) then
                ci := type_map_pkg.fetch(root_sn.V.TM, qi);
                -- if the current query input type is already mapped
                -- then make sure it is mapped to an existing type in
                -- the candidate's inputs. Note to test this we must
                -- look at the type_declaration's range (the types)
                -- not it's domain (the psdl_ids).
                if not type_declaration_pkg.res_set_pkg.member(ci,
                    type_declaration_pkg.map_range(c_inputs)) then
                    success := false;
                else
                    -- remove qi from new_q_inputs
                    type_declaration_pkg.remove(q_id, new_q_inputs);
                end if;
            end if;
        end if;
    end foreach;
end match_inputs;
```
-- remove ci from new_c_inputs
found_temp_id := false;
if not found_temp_id then
    foreach((c_id: psdl_id; c_tn: type_name),
        type_declaration_pkg.scan, (new_c_inputs),
        if equal(ci, c_tn) then
            temp_id := c_id;
            found_temp_id := true;
        end if;
    end if;
end if;
if found_temp_id then
    type_declaration_pkg.remove(temp_id, new_c_inputs);
else
    if this else block gets called
        there is something wrong
    put_line("there is something wrong");
    success := false;
end if;
end if;
end if;
end if;
end if;
end if;
if success then
    got_first_qi := false;
    foreach((q_id: psdl_id; qi: type_name),
        type_declaration_pkg.scan, (q_inputs),
        if not got_first_qi then
            got_first_qi := true;
            foreach((c_id: psdl_id; c_tn: type_name),
                type_declaration_pkg.scan, (c_inputs),
                temp_sn := new SigMatchNode'(createSigMatchNode);
                sigMatchNodeAssign(temp_sn.all, root_sn);
                temp_sn.expanded_for_inputs := false;
                type_map_pkg.bind(qi, c_tn, temp_sn.V.TM);
                type_declaration_pkg.assign(temp_q_inputs,
                    new_q_inputs);
                type_declaration_pkg.assign(temp_c_inputs,
                    new_c_inputs);
                type_declaration_pkg.remove(q_id, temp_q_inputs);
                type_declaration_pkg.remove(c_id, temp_c_inputs);
                match(temp_q_inputs, temp_c_inputs, temp_sn.all,
                    success);
                if success then
                    addBranch(temp_sn, return_val);
                end if;
            end if;
        end if;
    end if;
sigMatchNodeAssign(root_sn, return_val);
end match;
q_inputs, c_inputs: type_declaration;
begin
success := true;
foreach((q_op: operator; c_op: operator), op_map_pkg.scan, (root_sn.V.OM),
    if success then
        -- Remove the input types that have already been mapped.
        type_declaration_pkg.assign(q_inputs, inputs(q_op));
        type_declaration_pkg.assign(c_inputs, inputs(c_op));
        -- query
        foreach((the_id: psdl_id; the.tn: type_name),
            type_declaration_pkg.scan, (inputs(q_op)),
            if type_map_pkg.key_set_pkg.member(the.tn,
                type_map_pkg.map_domain(root_sn.V.TM)) then

If the type was mapped make sure it was mapped to
a type in the candidate operator. This is necessary
because inputs are mapped for one operator at a time.
if type_declaration_pkg.res_set_pkg.member(
    type_map_pkg.fetch(root_sn.V.TM, the tn),
    type_declaration_pkg.map_range(c_inputs)) then
    type_declaration_pkg.remove(the id, q_inputs);
else
    success := false;
end if;
end if;

-- candidate
foreach((the_id: psdl_id; the_n: type_name),
    type_declaration_pkg.scan, (inputs(c_op)) ,
    if type_map_pkg.res_set_pkg.member(the tn,
        type_map_pkg.map_range(root_sn.V.TM)) then
        type_declaration_pkg.remove(the_id, c_inputs);
    end if;
)

-- if the number of remaining inputs types for the query and
the candidate are not equal then the operations cannot match

if success then
    if type_declaration_pkg.size(q_inputs) /=
        type_declaration_pkg.size(c_inputs) then
        success := false;
    else
        if the node has already been expanded for inputs then
            all of its operators' inputs must already be mapped
        -- otherwise the node fails.
            if root_sn.expanded_for_inputs then
                success := type_declaration_pkg.size(q_inputs) = 0;
            else
                match(get_user_defined(q_inputs),
                    get_user_defined(c_inputs), root_sn, success);
            end if;
        end if;
    end if;
end if;
end if;
end match_inputs;

-- Function: verify_subtypes
-- Description:
function verify_subtypes(root_sn: in SigMatchNode) return boolean is
begin
    -- TODO
    return true;
end verify_subtypes;

-- Procedure: match_ops
-- Description: this is the main procedure for signature matching.
-- Given the operations and their profiles for a query and a
candidate, this method will return a SigMatchNode whose
branches contain valid operation and type mappings.
procedure match_ops(query, candidate: in OpWithProfileSeq;
    root_sn: in out SigMatchNode) is
return_val: SigMatchNode;
    temp_sn: SigMatchNodePtr;
    success, pruned: boolean;
    temp_query, temp_candidate: OpWithProfileSeq;
    temp_char: character;
begin
return_val := createSigMatchNode;
sigMatchNodeAssign(return_val, root_sn);

owp_sequence_pkg.assign(temp_query, query);
owp_sequence_pkg.assign(temp_candidate, candidate);
foreach((q_owp: OpWithProfile), owp_sequence_pkg.scan, (query),
  foreach((c_owp: OpWithProfile), owp_sequence_pkg.scan, (candidate),
    if q_owp.op_profile = c_owp.op_profile then
      temp_sn := new SigMatchNode'(createSigMatchNode);
      sigMatchNodeAssign(temp_sn.all, root_sn);
      op_map_pkg.bind(q_owp.op, c_owp.op, temp_sn.V.OM);
      if not validPairingExists(temp_sn.V.OM, return_val) then
        match_outputs(temp_sn.all, success);
        if success then
          passed_outputs := passed_outputs + 1;
          if match_basics(get_basics(inputs(q_owp.op)),
            get_basics(inputs(c_owp.op))) then
            opWithProfileSeqRemove(q_owp, temp_query);
            opWithProfileSeqRemove(c_owp, temp_candidate);
            match_ops(temp_query, temp_candidate, temp_sn.all);
            addBranch(temp_sn, return_val);
            passed_basics := passed_basics + 1;
          else
            failed_outputs := failed_outputs + 1;
          end if;
        else
          failed_outputs := failed_outputs + 1;
        end if;
      else
        duplicates := duplicates + 1;
      end if;
    end if;
  end foreach;
end foreach;

— prune leaf nodes until all leaves are valid solutions
—
pruned := true;
while pruned loop
  pruned := false;
sigMatchNodeAssign(root_sn, return_val);
foreach((leaf_snp: SigMatchNodePtr), sig_match_node_ptr_seq_pkg.scan,
  (getLeafNodePtrs(root_sn)),
  if leaf_snp.validation = UNKNOWN then
    match_inputs(leaf_snp.all, success);
    total_inputs := total_inputs + 1;
    if not success then
      leaf_snp.validation := INVALID;
      elsif not verify_subtypes(leaf_snp.all) then
        leaf_snp.validation := INVALID;
      else
        if sig_match_node_ptr_seq_pkg.length(leaf_snp.branches) = 0 then
          leaf_snp.validation := VALID;
        else
          leaf_snp.expanded_for_inputs := true;
        end if;
      end if;
    end if;
  end if;
  if leaf_snp.validation = INVALID then
    removeBranch(leaf_snp, return_val);
    removeAllMatchingBranches(leaf_snp, return_val);
    failed_inputs := failed_inputs + 1;
    pruned := true;
  end if;
end foreach;
end loop;

— recycle local variables
—
owp_sequence_pkg.recycle(temp_query)

170
owp_sequence_pkg.recycle(temp_candidate);

sigMatchNodeAssign(root_sn, return_val);
end match_ops;

procedure sigMatchStatsReset is
begin
failed_outputs := 0;
passed_outputs := 0;
failed_basics := 0;
passed_basics := 0;
duplicates := 0;
total_inputs := 0;
failed_inputs := 0;
end sigMatchStatsReset;

procedure sigMatchStatsPut(filename: string) is
the_file: file_type;
begin
create(the_file, out_file, filename);
put(the_file, "Duplicates: ");
put_line(the_file, integer'image(duplicates));
put(the_file, "Passed Output Matching: ");
p_put_line(the_file, integer'image(passed_outputs));
put(the_file, "Failed Output Matching: ");
p_put_line(the_file, integer'image(failed_outputs));
put(the_file, "Passed Predefined Type Matching: ");
p_put_line(the_file, integer'image(passed_basics));
put(the_file, "Failed Predefined Type Matching: ");
p_put_line(the_file, integer'image(failed_basics));
put(the_file, "Total Inputs: ");
p_put_line(the_file, integer'image(total_inputs));
put(the_file, "Failed Inputs: ");
p_put_line(the_file, integer'image(failed_inputs));
close(the_file);
end sigMatchStatsPut;
end sig_match;

Q. SIG_MATCH_TYPES.ADS

-- Package Spec: sig_match_types
---------------------------------------------------------------------------------------------
with text_io; use text_io;
with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
with psdl_component_pkg; use psdl_component_pkg;
with generic_map_pkg;
with generic_sequence_pkg;
with generic_set_pkg;
with ordered_set_pkg;
package sig_match_types is

-- Types

-- TypeMap

package type_map_pkg is new generic_map_pkg(
  key => type_name,
  result => type_name,
  eq_key => equal,
  eq_res => equal,
  average_size => 4);
subtype TypeMap is type_map_pkg.map;
procedure typeNamePut(the_tn: type_name);
procedure typeMapPut is new type_map_pkg.generic_put(
    key_put => typeNamePut, res_put => typeNamePut);

procedure typeMapFilePut is new type_map_pkg.generic_file_put(
    key_put => typeNamePut, res_put => typeNamePut);

-- OpMap
package op_map_pkg is new generic_map_pkg(
    key => operator,
    result => operator,
    eq_key => eq,
    eq_res => eq,
    average_size => 4);
subtype OpMap is op_map_pkg.map;

procedure opPut(the_op: operator);

procedure opMapPut is new op_map_pkg.generic_put(
    key_put => opPut, res_put => opPut);

procedure opMapFilePut is new op_map_pkg.generic_file_put(
    key_put => opPut, res_put => opPut);

-- SignatureMap

type SignatureMap is record
    TM: TypeMap;
    OM: OpMap;
end record;

function createSignatureMap return SignatureMap;

procedure addTypeMapping(tnl: in type_name; tn2: in type_name;
    sm: in out SignatureMap);

procedure addOpMapping(opl: in operator; op2: in operator;
    sm: in out SignatureMap);

function signatureMapEqual(sml: in SignatureMap; sm2: in SignatureMap)
    return boolean;

procedure signatureMapPut(sm: in SignatureMap);

-- SignatureMapSet
package sig_map_set_pkg is new generic_set_pkg(
    t => SignatureMap, eq => signatureMapEqual);
subtype SignatureMapSet is sig_map_set_pkg.set;

procedure signatureMapSetPut is
    new sig_map_set_pkg.generic_put(put => signatureMapPut);

-- SigMatchNodePtr

procedure sigMatchPut(smnp: in SigMatchNodePtr);

function sigMatchNodePtrEqual(smnpl: in SigMatchNodePtr;
    smnp2: in SigMatchNodePtr) return boolean;

function sigMatchNodePtrLessThan(smnpl: in SigMatchNodePtr;
    smnp2: in SigMatchNodePtr) return boolean;

procedure sigMatchNodePtrPut(smnp: in SigMatchNodePtr);
SigMatchNodePtrSeq

package sig_match_node_ptr_seq_pkg is new generic_sequence_pkg(
  t => SigMatchNodePtr, average_size => 4);
subtype SigMatchNodePtrSeq is sig_match_node_ptr_seq_pkg.sequence;

function sigMatchNodePtrSeqEqual is
  new sig_match_node_ptr_seq_pkg.generic_equal(eq => SigMatchNodePtrEqual);
function sigMatchNodePtrSeqMember is
  new sig_match_node_ptr_seq_pkg.generic_member(eq => sigMatchNodePtrEqual);
procedure sigMatchNodePtrSeqRemove is
  new sig_match_node_ptr_seq_pkg.generic_remove(eq => sigMatchNodePtrEqual);
procedure sigMatchNodePtrSeqPut is
  new sig_match_node_ptr_seq_pkg.generic_put(put => sigMatchNodePtrPut);

SigMatchNodePtrSet

package sig_match_node_ptr_set_pkg is new ordered_set_pkg(
  t => SigMatchNodePtr, eq => sigMatchNodePtrEqual,
  "<" => sigMatchNodePtrLessThan);
subtype SigMatchNodePtrSet is sig_match_node_ptr_set_pkg.set;
procedure sigMatchNodePtrSetPut is
  new sig_match_node_ptr_set_pkg.generic_put(put => SigMatchNodePtrPut);
procedure sigMatchNodePtrSetPrint(the_set: SigMatchNodePtrSet);

SigMatchNode

type ValidationType is (UNKNOWN, VALID, INVALID);
type SigMatchNode is record
  id: natural;
  signature_rank: float;
  semantic_rank: float;
  V: SignatureMap;
  validation: ValidationType;
  expanded_for_inputs: boolean;
  branches: SigMatchNodePtrSeq;
end record;

function createSigMatchNode return SigMatchNode;
procedure addBranch(the_branch: in SigMatchNodePtr;
  the_node: in out SigMatchNode);
procedure removeBranch(the_branch: in SigMatchNodePtr;
  the_node: in out SigMatchNode);
procedure removeAllMatchingBranches(the_branch: in SigMatchNodePtr;
  the_node: in out SigMatchNode);

function sigMatchNodeEqual(smnl: in SigMatchNode; smn2: in SigMatchNode)
  return boolean;
function sigMatchNodeLessThan(smnl: in SigMatchNode; smn2: in SigMatchNode)
  return boolean;

procedure sigMatchNodeAssign(smnl: in out SigMatchNode;
  smn2: in SigMatchNode);
procedure sigMatchNodePut(the_node: in SigMatchNode);
procedure sigMatchNodePrint(the_node: SigMatchNode);
procedure generateGML(the_node: in SigMatchNode; filename: in string);

function getLeafNodePtrs(the_node: in SigMatchNode) return SigMatchNodePtrSeq;
function getLeafNodePtrs(the_node: in SigMatchNode) return SigMatchNodePtrSet;

173
function getValidLeafNodePtrs(the_node: in SigMatchNode) return SigMatchNodePtrSet;

function validPairingExists(pairing: in OpMap; the_node: in SigMatchNode) return boolean;

end sig_match_types;

R. SIG_MATCH_TYPES.G

-- Package Body: sig_match_types
-- Package Body: sig_match_types
with text_io; use text_io;
with ada.float_text_io;

with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
with psdl_component_pkg; use psdl_component_pkg;

with candidate_types;
package body sig_match_types is

-- Procedure: typeNamePut
-- Description: outputs the type_name's name
-- procedure typeNamePut(the_tn: type_name) is
begin
  if not equal(the_tn, null_type) then
    put(convert(the_tn.name));
  end if;
end typeNamePut;

-- Procedure: opPut
-- Description: outputs the operator's name
-- procedure opPut(the_op: operator) is
begin
  if the_op /= null_component then
    put(convert(name(the_op)));
  end if;
end opPut;

-- Function: createSignatureMap
-- Description: create and initialize a SignatureMap for use.
-- function createSignatureMap return SignatureMap is
begin
  return_val.TM := type_map_pkg.create(null_type);
  return_val.OM := op_map_pkg.create(null_component);
  return return_val;
end createSignatureMap;

-- Procedure: addTypeMapping
-- Description: binds two types together and adds them to the
-- SignatureMap's TypeMap.
-- procedure addTypeMapping(tnl: in type_name; tn2: in type_name;
-- sm: in out SignatureMap) is
begin
  type_map_pkg.bind(tnl, tn2, sm.TM);
end addTypeMapping;

174
-- Procedure: addOpMapping
-- Description: binds two operators together and adds them to the SignatureMap's OpMap.
procedure addOpMapping(op1: in operator; op2: in operator; sm: in out SignatureMap) is
begin
    op_map_pkg.bind(op1, op2, sm.OM);
end addOpMapping;

-- Function: signatureMapEqual
function signatureMapEqual(sml: in SignatureMap; sm2: in SignatureMap) return boolean is
begin
    return type_map_pkg.equal(sml.TM, sm2.TM) and op_map_pkg.equal(sml.OM, sm2.OM);
end signatureMapEqual;

-- Procedure: signatureMapPut
procedure signatureMapPut(sm: in SignatureMap) is
begin
    put("OM: ");
    opMapPut(sm.OM);
    put(" | TM: ");
    typeMapPut(sm.TM);
end signatureMapPut;

-- Function: sigMatchNodePtrEqual
function sigMatchNodePtrEqual(smnpl: in SigMatchNodePtr; smnp2: in SigMatchNodePtr) return boolean is
begin
    return sigMatchNodeEqual(smnpl.all, smnp2.all);
end sigMatchNodePtrEqual;

-- Function: sigMatchNodePtrLessThan
function sigMatchNodePtrLessThan(smnpl: in SigMatchNodePtr; smnp2: in SigMatchNodePtr) return boolean is
begin
    return sigMatchNodeLessThan(smnpl.all, smnp2.all);
end SigMatchNodePtrLessThan;

-- Procedure: sigMatchNodePtrPut
procedure sigMatchNodePtrPut(smnp: in SigMatchNodePtr) is
begin
    sigMatchNodePut(smnp.all);
end sigMatchNodePtrPut;

-- Function: sigMatchNodeEqual
function sigMatchNodeEqual(smnl: in SigMatchNode; smn2: in SigMatchNode) return boolean is
begin
    if smnl.signature_rank /= smn2.signature_rank then
        return false;
    end if;
    if smnl.semantic_rank /= smn2.semantic_rank then
        return false;
    end if;
end sigMatchNodeEqual;
if smnl.validation /= smn2.validation then
  return false;
end if;

if smnl.expanded_for_inputs /= smn2.expanded_for_inputs then
  return false;
end if;

if not signatureMapEqual(smnl.V, smn2.V) then
  return false;
end if;

return sigMatchNodePtrSeqEqual(smnl.branches, smn2.branches);
end sigMatchNodeEqual;

-- Function: sigMatchNodeLessThan

function sigMatchNodeLessThan(smnl: in SigMatchNode;
                               smn2: in SigMatchNode) return boolean is
begin
  if smnl.signature_rank > smn2.signature_rank then
    return true;
  -- the following test for less-than is just being paranoid
  elsif smnl.signature_rank < smn2.signature_rank then
    return false;
  elsif smnl.semantic_rank > smn2.semantic_rank then
    return true;
  -- the following test for less-than is just being paranoid
  elsif smnl.semantic_rank < smn2.semantic_rank then
    return false;
  else
    return smnl.id < smn2.id;
  end if;
end if;
end sigMatchNodeLessThan;

-- Procedure: sigMatchNodeAssign

procedure sigMatchNodeAssign(smnl: in out SigMatchNode;
                            smn2: in SigMatchNode) is
begin
  smnl.signature_rank := smn2.signature_rank;
  smnl.semantic_rank := smn2.semantic_rank;
  smnl.validation := smn2.validation;
  smnl.expanded_for_inputs := smn2.expanded_for_inputs;
  type_map_pkg.assign(smnl.V.TM, smn2.V.TM);
  op_map_pkg.assign(smnl.V.OM, smn2.V.OM);
  -- TODO: might have to do the deep copy myself here
  --       rather than call assign
  sig_match_node_ptr_seq_pkg.assign(smnl.branches, smn2.branches);
end sigMatchNodeAssign;

-- Procedure: sigMatchNodePut

procedure sigMatchNodePut(the_node: in SigMatchNode) is
begin
  put("(Signature Rank: ");
  if the_node.signature_rank = candidate_types.RANK_UNKNOWN then
    put("unknown");
  else
    ada.float_text_io.put(the_node.signature_rank, 1, 2, 0);
  end if;
  put(" | ");
  if the_node.semantic_rank = candidate_types.RANK_UNKNOWN then
    put("unknown");
  else
    ada.float_text_io.put(the_node.semantic_rank, 1, 2, 0);
  end if;
  put(" | ");
case the_node.validation is
  when UNKNOWN => put("Validation Unknown");
  when VALID => put("Valid");
  when INVALID => put("Invalid");
end case;
put(" | ");
if the_node.expanded_for_inputs then
  put("Expanded");
else
  put("Not Expanded");
end if;
put(" | ");
opMapPut(the_node.V.OM);
put(" | ");
typeMapPut(the_node.V.TM);
put(" | ");
sigMatchNodePtrSeqPut(the_node.branches);
end sigMatchNodePut;

-- Procedure: sigMatchNodePrint
--
procedure sigMatchNodePrint(the_node: SigMatchNode) is
begin
  put(" Signature Rank: ");
  if the_node.signature_rank = candidate_types.RANK_UNKNOWN then
    put("unknown");
  else
    ada.float_text_io.put(the_node.signature_rank, 1, 2, 0);
  end if;
  new_line;
  put(" Semantic Rank: ");
  if the_node.semantic_rank = candidate_types.RANK_UNKNOWN then
    put("unknown");
  else
    ada.float_text_io.put(the_node.semantic_rank, 1, 2, 0);
  end if;
  new_line;
  put(" |");
case the_node.validation is
  when UNKNOWN => put("Validation Unknown");
  when VALID => put("Valid");
  when INVALID => put("Invalid");
end case;
put(" |");
if the_node.expanded_for_inputs then
  put_line("Expanded");
else
  put_line("Not Expanded");
end if;
put(" Op Map: ");
opMapPut(the_node.V.OM);
new_line;
put(" Type Map: ");
typeMapPut(the_node.V.TM);
new_line;
put(" Branches: ");
sigMatchNodePtrSeqPut(the_node.branches);
new_line;
end sigMatchNodePrint;

--
-- Function: createSigMatchNode
--
-- Description: create and initialize a SigMatchNode for use.
-- Note: a unique node id is maintained to facilitate
-- sorting when two nodes have equal signature and
-- semantic ranks.
unique_node_id := 0;

function createSigMatchNode return SigMatchNode is
    return_val: SigMatchNode;
begin
    return_val.id := unique_node_id;
    unique_node_id := unique_node_id + 1;
    return_val.signature_rank := candidate_types.RANK_UNKNOWN;
    return_val.semantic_rank := candidate_types.RANK_UNKNOWN;
    return_val.validation := UNKNOWN;
    return_val.expanded_for_inputs := false;
    return_val.V := createSignatureMap;
    return_val.branches := sig_match_node_ptr_seq_pkg.empty;
    return return_val;
end createSigMatchNode;

-- Function: addBranch
-- Description: add a branch (a child SigMatchNode) to the SigMatchNode.
-- A branch represents a superset of the node it belongs to.
-- What this really means is the branch node contains all the
-- type and operator mappings plus of the node it belongs to
-- plus more.
procedure addBranch(the_branch: in SigMatchNodePtr;
the_node: in out SigMatchNode) is
begin
    sig_match_node_ptr_seq_pkg.add(the_branch, the_node.branches);
end addBranch;

-- Function: removeBranch
-- Description:
procedure removeBranch(the_branch: in SigMatchNodePtr;
the_node: in out SigMatchNode) is
begin
    sigMatchNodePtrSeqRemove(the_branch, the_node.branches);
end removeBranch;

-- Function: removeAllMatchingBranches
-- Description:
procedure removeAllMatchingBranches(the_branch: in SigMatchNodePtr;
the_node: in out SigMatchNode) is
begin
    sigMatchNodePtrSeqRemove(the_branch, the_node.branches);
    foreachf(branch: SigMatchNodePtr), sig_match_node_ptr_seq_pkg.scan,
    (the_node.branches),
    removeAllMatchingBranches(thejbranch, branch.all);
end removeAllMatchingBranches;

-- Procedure: generateGML
-- Description: generate a GML file to graphically represent the
-- SigMatchNode's relationship with its branches.
procedure generateGML(the_node: in SigMatchNode; filename: string) is
    id: natural := 0; -- unique ID counter
    the_id: natural; -- place holder for call to put_node_gml
    gml_file: file_type;
function new_id return natural is
begin
    id := id + 1;
    return id;
end new_id;
procedure put_node_gml(sn: in SigMatchNode; my_id: out natural) is
child_id: natural;
begin
  my_id := new_id;
  put(gml_file, "node [ id ");
  put(gml_file, integer'image(my_id));
  put(gml_file, " label ");
  opMapFilePut(gml_file, sn.V.OM);
  put_line(gml_file, "");
  typeMapFilePut(gml_file, sn.V.TM);
  put_line(gml_file, "");
  case sn.validation is
    when UNKNOWN => put(gml_file, "Validation Unknown");
    when VALID => put(gml_file, "Valid");
    when INVALID => put(gml_file, "Invalid");
  end case;
  put_line(gml_file, "");
  if sn.expanded_for_inputs then
    put(gml_file, "Expanded");
  else
    put(gml_file, "Not Expanded");
  end if;
  put_line(gml_file, "");
-- recursively call put_node_gml for each of its branches
foreach((branch: SigMatchNodePtr), sig_match_node_ptr_seq_pkg.scan,(sn.branches),
  put_node_gml(branch.all, child_id);
-- make the edge to the branch
put(gml_file, "edge [ id ");
put(gml_file, integer'image(new_id));
put(gml_file, " source ");
put(gml_file, integer'image(my_id));
put(gml_file, " target ");
put(gml_file, integer'image(child_id));
put_line(gml_file, "");
end put_node_gml;
begin
create(gml_file, out_file, filename);
put(gml_file, "graph [ id ");
put(gml_file, integer'image(new_id));
put_line(gml_file, " directed 1");
put_node_gml(the_node, the_id);
put_line(gml_file, "");
close(gml_file);
end generateGML;
-- Function: getLeafNodePtrs
-- Description: collect the leaf nodes of the_node into a sequence.
function getLeafNodePtrs(the_node: in SigMatchNode) return SigMatchNodePtrSeq is
return_val: SigMatchNodePtrSeq;
procedure processNode(smnp: in SigMatchNodePtr) is
begin
  if sig_match_node_ptr_seq_pkg.length(smnp.branches) = 0 then
    sig_match_node_ptr_seq_pkg.add(smnp, return_val);
    return;
  end if;
foreach((branch: SigMatchNodePtr), sig_match_node_ptr_seq_pkg.scan,(smnp.branches),
  processNode(branch);
end processNode;
begin
return_val := sig_match_node_ptr_seq_pkg.empty;
foreach((branch: SigMatchNodePtr), sig_match_node_ptr_seq_pkg.scan,
begin
  return_val := sig_match_node_ptr_set_pkg.empty;
  foreach (branch: SigMatchNodePtr, sig_match_node_ptr_set_pkg.scan,
    (the_node.branches),
    processNode(branch);
  )
  return return_val;
end getValidLeafNodePtrs;

-- Function: validPairingExists
--
-- Description: gets all the valid leaf nodes and checks if the pairing
--
exists in any of them

function validPairingExists(pairing: in OpMap; the_node: in SigMatchNode)
return boolean is
return_val: boolean;
begin
return_val := false;
foreach((sn: SigMatchNodePtr), sig_match_node_ptr_set_pkg.scan,
  (getValidLeafNodePtrs(the_node)),
  if not return_val then
    return_val := op_map_pkg.submap(pairing, sn.V.OM);
    -- TODO: if return_val is true then should immediately return
    -- but for each doesn't let me do this
    end if;
  )
return return_val;
end validPairingExists;

-- Procedure: sigMatchNodePtrSetPrint

-- procedure sigMatchNodePtrSetPrint(the_set: sigMatchNodePtrSet) is
begin
foreach((the_node: SigMatchNodePtr), sig_match_node_ptr_set_pkg.scan,
  (the_set),
  sigMatchNodePrint(the_node.all);
  put_line("%%%End_Signature%%");
  )
  put_line("%%%End_Component%%");
end sigMatchNodePtrSetPrint;

end sig_match_types;

S. SOFTWARE_BASE.ADS

-- Package Spec: software_base

with gnat.io;
with component_id_types; use component_id_types;
with haase_diagram; use haase_diagram;
with candidate_types; use candidate_types;
with profile_types; use profile_types;
with a_strings;

package software_base is
  procedure initialize(sb_root: in string; header_filename: in string);
  procedure reinitialize(sb_root: in string);
  function numComponents return natural;
  function numPartitions return natural;
  function numOccupiedPartitions return natural;
  procedure generateGML(gml_filename: in string);
  procedure getCandidateFilename(component_id: in integer; filename: out string;
    filename_length: out integer);
  function profileFilter(query_filename: in string) return CandidateSet;
  function signatureMatch(query_filename: in string;
    the_candidate: in Candidate;
    srank: in float) return Candidate;
  function getProfileID(p: Profile) return ProfileID;
  function getProfile(p_id: ProfileID) return Profile;

181
function getProfileIDs return profile_lookup_table_pkg.res_set;

private
  -- the_component_id_map
  the_component_id_map: ComponentIDMap;
  -- the_haase_diagram
  the_haase_diagram: HaaseDiagram;
  -- the_profile_lookup_table
  the_profile_lookup_table: ProfileLookupTable;
end software_base;

T. SOFTWARE_BASE.G

with text_io; use text_io;
with ada.integer_text_io; use ada.integer_text_io;
with lookahead_pkg;
with sb_utils;
with a_strings;
with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
with component_id_types; use component_id_types;
with haase_diagram; use haase_diagram;
with candidate_types; use candidate_types;
with profile_types; use profile_types;
with psdl_profile; use psdl_profile;
with sig_match_types; use sig_match_types;
with profile_filter_pkg;
with sig_match;

package body software_base is
  -- Procedure: initialize
  -- Description: reads the header file to construct the_component_id_map
  -- and the_haase_diagram.
  procedure initialize(sb_root: in string; header_filename: in string) is
    use a_strings;
    header_file: file_type;
    comp_id: ComponentID;
    dir_name: a_string;
    sb_file_name: a_string;
    input_line: string(1..256);
    line_length: natural;
    comp_id_last : natural;
    temp_comp_profile: ComponentProfile;
    temp_haase_node: HaaseNode;
    temp_component: Component;
    the_generics_maps: GenericsMapSet;
    generics_mapping: GenericsMap;
    n_components: integer := 0;
    message: a_string;
    id: natural := 0;
begin

    -- parse header file and construct the_component_id_map
    --
    component_id_map_pkg.create(createComponent, the_component_id_map);

    open(header_file, in_file, header_filename);
    while (not end_of_file(header_file)) loop
        n_components := n_components + 1;
        get_line(header_file, input_line, line_length);
        ada.integer_text_io.get(input_line, comp_id, comp_id_last);
        -- trim spaces before and after directory name
        dir_name := reverse_order(trim(reverse_order(trim(a_strings.to_a(input_line(comp_id_last+l..line_length))))));
        -- create a component for each generic_mapping
        the_generics_maps := getGenericsMaps(convert(text(dir_name & "/PSDL_SPEC")));
        message := to_a(" Preparing ") & input_line(1..line_length);
        message := message & ";
        message := message & integer'image(generics_map_set_pkg.size(the_generics_maps));
        message := message & ";
        sb_utils.Display_Message(message.s);
        foreach((the_map: GenericsMap), generics_map_set_pkg.scan, (the_generics_maps),
            temp_component := createComponent;
            temp_component.pSDL_filename := text(dir_name & "/PSDL_SPEC");
            generics_map_pkg.assign(temp_component.generics_mapping, the_map);
            component_id_map_pkg.bind(new_id(comp_id), temp_component, the_component_id_map);
        )
        sb_utils.Display_Message_line("done");
    end loop;
    close(header_file);

    -- Create the ProfileLookupTable
    the_profile_lookup_table := profile_lookup_table_pkg.create(DEFAULT_PROFILE_ID);

    -- construct haase diagram
    --
    the_haase_diagram := createHaaseDiagram;
    -- for each item in the_component_id_map, get the component's
    -- profile and add it to the_haase_diagram
    foreach((the_comp_id: ComponentID; the_component: Component),
        component_id_map_pkg.scan, (the_component_id_map),
            message := to_a(" inserting " & integer'image(the_comp_id);
            sb_utils.Display_Message_line(message.s);
            temp_comp_profile := getComponentProfile(convert(the_component.pSDL_filename), the_component.generics_mapping);
            -- check if haase node with temp_comp_profile as its key
            -- already exists.  If it does then add the component id
            -- to that node rather than make a new node.
            if haase_node_map_pkg.member(temp_comp_profile, the_haase_diagram) then
                temp_haase_node := haase_node_map_pkg.fetch(the_haase_diagram, temp_comp_profile);
                temp_haase_node.component_id := the_comp_id;
            else
                temp_haase_node := haase_node_map_pkg.put(the_haase_diagram, temp_comp_profile, new_id(the_comp_id));
            end if;
        )
    sb_utils.Display_Message_line("done");
end;
temp_haase_node := createHaaseNode(temp_comp_profile);
end if;
addComponent(the_comp_id, temp_haase_node);
addHaaseNode(temp_haase_node, the_haase_diagram);

sb_utils.Display_Message(" adding base nodes..."),
addBaseNodes(the_haase_diagram);
sb_utils.Display_Message(" connecting nodes...");
connectNodes(the_haase_diagram);
sb_utils.Display_Message(" done");

sb_utils.Display_Message_line(" Saving the software base search data");
sb_utils.Display_Message(" the_component_id_map...");
sbfile_name := a_strings.to_a(sb_root) & "component_id_map.dat";
create(header_file, out_file, sbfile_name.s);
set_output (header_file);
componentIDMapPut(the_component_id_map);
set_output (standard_output);
close(header_file);
sb_utils.Display_Message_line(" done");

sb_utils.Display_Message(" the_profile_lookup_table...");
sbfile_name := a_strings.to_a(sb_root) & "profile_lookup_table.dat";
create(header_file, out_file, sbfile_name.s);
profileLookupTableFilePut(header_file, the_profile_lookup_table);
close(header_file);
sb_utils.Display_Message_line(" done");

sb_utils.Display_Message(" the_haase_diagram...");
sbfile_name := a_strings.to_a(sb_root) & "haase_diagram.dat";
create(header_file, out_file, sbfile_name.s);
set_output (header_file);
haaseDiagramPut(the_haase_diagram);
set_output (standard_output);
close(header_file);
sb_utils.Display_Message_line(" done");
end initialize;

procedure reinitialize(sb_root: in string) is

use a_strings;

header_file: file_type;
comp_id: ComponentID;
dir_name: a_string;
sbfile_name: a_string;
input_line: string(1..256);
line_length: natural;
comp_id_last: natural;
temp_comp_profile: ComponentProfile;
temp_haase_node: HaaseNode;
temp_component: Component;
the_generics_maps: GenericsMapSet;
generics_mapping: GenericsMap;
n_components: integer := 0;
key_id, res_id: psdl_id;

id: natural := 0;
old_start: natural := 0;

begin
  -- parse header file and construct the_component_id_map
  --
  component_id_map_pkg.create(createComponent, the_component_id_map);

  sb_utils.Display_Message(" Retrieving the_component_id_map...");
sbfile_name := a_strings.to_a(sb_root) & "component_id_map.dat";
open(header_file, in_file, sbfile_name.s);
set_input(header_file);
while (not end_of_file(header_file)) loop
    temp_component := createComponent;

sb_utils.get_char_line(input_line, line_length);
da.integer_text_io.get(input_line, comp_id, comp_id_last);
sb_utils$get_char_line(input_line, line_length);
temp_component.pdpl_filename := convert(input_line(1..line_length));
if lookahead_pkg.token /= '{' then
    lookahead_pkg.skip_char;
    generics_map_pkg.recycle(temp_component.generics_mapping);
else
    genericsMapGet(temp_component.generics_mapping);
end if;
component_id_map_pkg.bind(comp_id, temp_component, the_component_id_map);
end loop;
set_input(standard_input);
close(header_file);
sb_utils.Display_Message_line("done");
--
-- Create the ProfileLookupTable
-- the_profile_lookup_table :=
profile_lookup_table_pkg.create(DEFAULT_PROFILE_ID);
sb_utils.Display_Message(" Retrieving the_profile_lookup_table...");
sbfile_name := a_strings.to_a(sb_root) & "profile_lookup_table.dat";
open(header_file, in_file, sbfile_name.s);
set_input (header_file);
profileLookupTableGet(the_profile_lookup_table);
set_input (standard_input);
close(header_file);
sb_utils.Display_Message_line("done");
--
-- construct haase diagram
-- the_haase_diagram := createHaaseDiagram;

sb_utils.Display_Message(" Retrieving the_haase_diagram...");
sbfile_name := a_strings.to_a(sb_root) & "haase_diagram.dat";
open(header_file, in_file, sbfile_name.s);
set_input (header_file);
haaseDiagramGet(the_haase_diagram);
set_input (standard_input);
close(header_file);
sb_utils.Display_Message_line("done");
end reinitialize;
--
-- Function: numComponents
-- Description: return the number of components in the software base.
--
function numComponents return natural is
    return_val: natural;
begin
    return component_id_map_pkg.size(the_component_id_map);
end numComponents;
--
-- Function: numPartitions
-- Description: return the number of partitions in the software base.
--
function numPartitions return natural is
begin
    return haase_node_map_pkg.size(the_haase_diagram);
end numPartitions;
--
-- Function: numOccupiedPartitions
-- Description: return the number of occupied partitions in the
-- software base.
--
function numOccupiedPartitions return natural is
  return_val: natural := 0;
begin
  foreach((the_key: ComponentProfile; the_hn: HaaseNode),
    haase_node_map_pkg.scan, (the_haase_diagram),
    if component_id_set_pkg.size(the_hn.components) > 0 then
      return_val := return_val + 1;
    end if;
  )
  return return_val;
end numOccupiedPartitions;

-- Function: generateGML
--
procedure generateGML(gml_filename: string) is
begin
  generateGML(the_haase_diagram, gml_filename);
end generateGML;

-- Function: profileFilter
--
-- Description: performs profile filtering with the PSDL specified query
-- and returns an ordered set of candidates with the highest
-- profile ranking first.
-- Note the PSDL query must NOT contain generics.
function profileFilter(query_filename: in string) return CandidateSet is
  query_profile: ComponentProfile;
begin
  query_profile := getComponentProfile(query_filename,
    generics_map_pkg.create(empty));
  return profile_filter_pkg.findCandidates(query_profile, the_haase_diagram);
end profileFilter;

-- Function: signatureMatch
--
-- Description: performs signature matching between the PSDL specified
-- query and the_candidate and returns a copy of the_candidate
-- with the signature_matches field set.
function signatureMatch(query_filename: in string; the_candidate: in Candidate;
  srank: in float) return Candidate is
  q_ops, c_ops: OpWithProfileSeq;
  sn: SigMatchNode;
  temp_snp_set: SigMatchNodePtrSet;
  temp_component: Component;
  return_val: Candidate;
begin
  -- get the query's operators
  q_ops := getOpsWithProfiles(query_filename, generics_map_pkg.create(empty));
  -- get the candidate's operators
  temp_component := component_id_map_pkg.fetch(the_component_id_map,
    the_candidate.component_id);
  c_ops := getOpsWithProfiles(convert(temp_component.psdl_filename),
    temp_component.generics_mapping);
  -- perform signature matching
  sn := createSigMatchNode;
  sig_match.sigMatchStatsReset;
  sig_match.match_ops(q_ops, c_ops, sn);
  -- calculate the signature ranks
  sig_match_node_ptr_set_pkg.assign(temp_snp_set, getLeafNodePtrs(sn));
  foreach((smnp: SigMatchNodePtr), sig_match_node_ptr_set_pkg.scan,
    (temp_snp_set),
    smnp.signature_rank := float(op_map_pkg.size(smnp.V.OM)) /
float(owp_sequence_pkg.length(q_ops));

-- The following calculation for signature rank measures how well the
-- signature matching method works on its own. The calculation above
-- is really a mixture of profile filtering AND signature matching.

smnp.signature_rank := float(op_map_pkg.size(smnp.V.OM)) /
(return_val.profile_rank * float(owp_sequence_pkg.length(q_ops)));

-- add each SigMatchNodePtr to make sure return_val's signature_matches
-- field is sorted
candidateAssign(return_val, the_candidate);
foreach((smnp: SigMatchNodePtr), sig_match_node_ptr_set_pkg.scan,
(temp_snp_set),
  if smnp.signature_rank > srank then
    sig_match_node_ptr_set_pkg.add(smnp, return_val.signature_matches),
  end if;
)
return return_val;
end signatureMatch;

-- Function: getProfileID
-- Description: if the profile doesn't exist then add it first then
-- return its id. A new id is obtained from the global
-- variable unique_profile_id.
unique_profile_id: ProfileID := 0;
function getProfileID(p: Profile) return ProfileID is
  return_val: ProfileID;
begin
  return_val :=
    profile_lookup_table_pkg.fetch(the_profile_lookup_table, p);
  if return_val = DEFAULT_PROFILE_ID then
    return_val := unique_profile_id;
    unique_profile_id := unique_profile_id + 1;
    profile_lookup_table_pkg.bind(p, return_val, the_profile_lookup_table);
  end if;
  return return_val;
end getProfileID;

-- Function: getProfile
function getProfile(p_id: ProfileID) return Profile is
  return_val := 0;
begin
  foreach((p: Profile; id: ProfileID), profile_lookup_table_pkg.scan,
    (the_profile_lookup_table),
    if id = p_id then
      return_val := p;
      -- TODO: should return here but for each doesn't let me
      end if;
  )
  return return_val;
end getProfile;

-- Function: getProfileIDs
function getProfileIDs return profile_lookup_table_pkg.res_set is
begin
  return profile_lookup_table_pkg.map_range(the_profile_lookup_table);
end getProfileIDs;

-- Procedure to return the file name of the candidate psdl component
procedure getCandidateFilename(component_id: in integer;
    filename: out string;
    filename_length: out integer) is
begin
    temp_component := component_id_map_pkg.fetch(the_component_id_map, component_id);
    filename_length := temp_component.psdl_filename.len;
    for i in 1..filename_length loop
        filename(i) := temp_component.psdl_filename.s(i);
    end loop;
end getCandidateFilename;
end software_base;
<table>
<thead>
<tr>
<th></th>
<th>INITIAL DISTRIBUTION LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Defense Technical Information Center</td>
</tr>
<tr>
<td></td>
<td>8725 John J. Kingman Rd., Ste 0944</td>
</tr>
<tr>
<td></td>
<td>Ft. Belvoir, VA 22060-6218</td>
</tr>
<tr>
<td>2</td>
<td>Dudley Knox Library</td>
</tr>
<tr>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td>411 Dyer Rd.</td>
</tr>
<tr>
<td></td>
<td>Monterey, CA 93943-5101</td>
</tr>
<tr>
<td>3</td>
<td>Center for Naval Analysis</td>
</tr>
<tr>
<td></td>
<td>4401 Ford Ave.</td>
</tr>
<tr>
<td></td>
<td>Alexandria, VA 22302</td>
</tr>
<tr>
<td>4</td>
<td>Dr. Dan Boger, Chairman, Code CS</td>
</tr>
<tr>
<td></td>
<td>Computer Science Dept.</td>
</tr>
<tr>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td>Monterey, CA 93943</td>
</tr>
<tr>
<td>5</td>
<td>Chief of Naval Research</td>
</tr>
<tr>
<td></td>
<td>800 North Quincy St.</td>
</tr>
<tr>
<td></td>
<td>Arlington, VA 22217</td>
</tr>
<tr>
<td>6</td>
<td>Dr. Luqi, Code CS/Lq</td>
</tr>
<tr>
<td></td>
<td>Computer Science Dept.</td>
</tr>
<tr>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td>Monterey, CA 93943</td>
</tr>
<tr>
<td>7</td>
<td>CAPT Galik (OT1)</td>
</tr>
<tr>
<td></td>
<td>Space and Naval Warfare Systems Center</td>
</tr>
<tr>
<td></td>
<td>San Diego, CA 92152-5001</td>
</tr>
<tr>
<td>8</td>
<td>Gregory L. Meckstroth</td>
</tr>
<tr>
<td></td>
<td>Space and Naval Warfare Systems Center</td>
</tr>
<tr>
<td></td>
<td>53560 Hull Street (Code D711)</td>
</tr>
<tr>
<td></td>
<td>San Diego, CA 92152-5001</td>
</tr>
</tbody>
</table>