

NASA / GODDARD SPACE FLIGHT CENTER
Advanced Architectures and Automation Branch

**NGST Scientist's Expert Assistant
(SEA) Design Document**

Release 1

February, 1998



National Aeronautics and
Space Administration

————— Goddard Space Flight Center —————
Greenbelt, Maryland

TABLE OF CONTENTS

1	INTRODUCTION	1
1.1	Purpose	1
1.2	Audience	1
1.3	Applicable Documents	1
1.4	Document Organization	1
2	OVERVIEW	1
3	REQUIREMENTS SUMMARY	2
3.1	Functional Requirements	2
3.1.1	Overview	2
3.1.2	Proposal	2
3.1.3	User Interface	2
3.2	Hardware requirements	3
3.3	Software requirements	3
4	CONCEPTUAL MODEL	4
4.1	High Level	4
4.2	Proposal Data Model	4
4.2.1	Proposal	4
4.2.2	Visit	5
4.2.3	Exposure	5
4.2.4	Instrument	5
4.2.5	Detector	6
4.2.6	Filter	6
4.2.7	Astronomical Object	6
4.2.8	Morphology	6
4.2.9	Spectrum	6
4.2.10	Normalization	6
4.2.11	Observatory Parameters	6
4.2.12	Aperture	7
4.2.13	Wavelength	7
4.3	Use Cases	7
5	DESIGN MODEL	8

5.1	Diagram Notation	8
5.2	User Interface	9
5.2.1	Interview Mode	9
5.2.2	Browser Mode	10
5.2.3	Concepts	11
5.2.4	Exposure Time Calculator	12
5.2.5	Visual Target Tuner	13
5.3	Architecture Model	15
5.3.1	High Level Overview	15
5.3.2	Science Objects	16
5.3.3	Expert System Objects	16
5.3.4	User Interface Framework	17
5.3.5	User Interface Modules	18
5.4	Design Class Model	21
5.4.1	Science Package	21
5.4.2	User Interface Framework	34
5.4.3	Visual Target Tuner	39
5.5	Class Interaction Diagrams	43
5.5.1	Diagram 1: Starting a SEA browser window	44
5.5.2	Diagram 2: Opening an existing proposal	44
5.5.3	Diagram 3: Selecting an exposure time in the proposal tree	45
5.5.4	Diagram 4: Setting a new value for the exposure time	45
5.5.5	Diagram 5: Saving the proposal to disk	46
6	JAVA IMPLEMENTATION NOTES	47
6.1	Development Tools	47
6.1.1	Integrated Development Tool: Visual Café	47
6.1.2	Configuration Management: Visual SourceSafe	47
6.1.3	Expert System Engine: Advisor/J	47
6.2	Additions/Exemptions to the Java Style Guide	47
6.2.1	Property names	47
6.3	Security Management	47
6.3.1	User security	47
6.3.2	Applet security	48
6.4	Deployment notes/comments	48
6.5	Location of implementation files	48

1 Introduction

1.1 Purpose

This document describes the design for Release 1 of the NGST Scientist's Expert Assistant (SEA). It contains a non-technical analysis of the underlying astronomical concepts, as well as a technical description of the software design model. This document was written for the Advanced Architectures and Automation Branch of the NASA Goddard Space Flight Center.

1.2 Audience

The design portions of this document were written with the assumption that the reader is familiar with object-oriented software engineering concepts. Familiarity with the Java programming language was also assumed. The sections related to the conceptual model, however, were written for a much broader audience and do not require software engineering knowledge.

1.3 Applicable Documents

1. *NGST Scientist's Expert Assistant (SEA) Phase I Summary*, October 1997.
2. *NGST Scientist's Expert Assistant (SEA) Requirements Document*, February 1998.

1.4 Document Organization

This document is organized into four main sections. Section 3 summarizes the high-level requirements that drove the design. Section 4 describes our analysis of the underlying astronomical concepts. Section 5 contains the detailed design model. Finally, section 6 contains Java-specific implementation details.

2 Overview

The SEA is a research effort to explore the feasibility of developing Java-based Visual Tools, and a Rule-Based Expert System to assist NGST scientists. It targets the Phase II proposal process, for which proposing scientists are currently required to provide extensive spacecraft and instrument parameters to specify their requested observation. The goal of the SEA is to communicate with the scientist in scientific language and guide them through the proposal specification process.

Object-oriented techniques have been applied throughout the SEA design. Exclusive use of the Java language helps to promote these techniques. In addition, the Unified Modeling Language (UML) has been used whenever possible to enforce these techniques and to clarify the design. An iterative process of requirements gathering, analysis, and design has been applied.

3 Requirements Summary

This section summarizes the high-level requirements for the Scientist's Expert Assistant. See the SEA Requirements Document for a more detailed description of the requirements.

3.1 Functional Requirements

3.1.1 Overview

The SEA will provide the ability to construct Phase II proposals for HST's Advanced Camera for Surveys (ACS), and ultimately the Next Generation Space Telescope (NGST). It will allow less-experienced users access to the proposal through an easy-to-use interview process while allowing experienced users full control in accessing the entire proposal. The SEA will use expert system technology to insulate the user from instrument knowledge as much as possible. The user will be encouraged to express their proposal in terms of the science that they wish to achieve, rather than the instrument parameters necessary to achieve that science. Finally, the SEA must be easily available over the World Wide Web, and must not require any client software other than an up-to-date Web browser.

3.1.2 Proposal

The user will be able to create a new proposal, load an existing proposal from a file, save a proposal to a file, and submit a completed proposal. The user will also be able to output their proposal to a printer or to a text file. A proposal may include multiple exposures which may be organized into multiple visits. Multiple proposals may be open at the same time, and the user will be able to copy items from one proposal to another.

3.1.3 User Interface

The SEA will contain two main user interfaces: an interview interface for less-experienced users, and a browser interface for experienced users. These interfaces will be completely separate from the proposal that they are modifying. Indeed, the user will be able to switch between them at any time while working on the same proposal. The browser user interface will allow the user maximum control over the proposal. The user may jump to any area of the proposal at any time. Multiple editors for different proposal areas may be open at once. When the user makes a change to any area of the proposal, it will automatically be propagated to the rest of the proposal.

The user interface will employ visual techniques to communicate information wherever possible. The Exposure Time Calculator, for example, will contain graphs that the user may manipulate in real time. Also, the Visual Target Tuner will present an image of the area surrounding the target, allowing the user to specify position and orientation visually by selecting and dragging icons in the image.

Help will be built into the SEA. The user should have easy access to online documents related to the proposal process and the operation of the SEA. In addition, the tool should allow the user to receive help on any individual component in the tool by selecting that component. Help documents will be HTML-based.

3.2 Hardware requirements

Because the SEA uses advanced software technology, it requires a relatively recent machine to execute smoothly. For the Windows 95/NT environment, at least a 166 MHz Pentium PC with 32 MB RAM is required. For the Solaris environment, at least a 166 MHz SparcStation with 32 MB RAM is required. For the Macintosh environment, at least a 150 MHz PowerPC machine with 32 MB RAM is required.

3.3 Software requirements

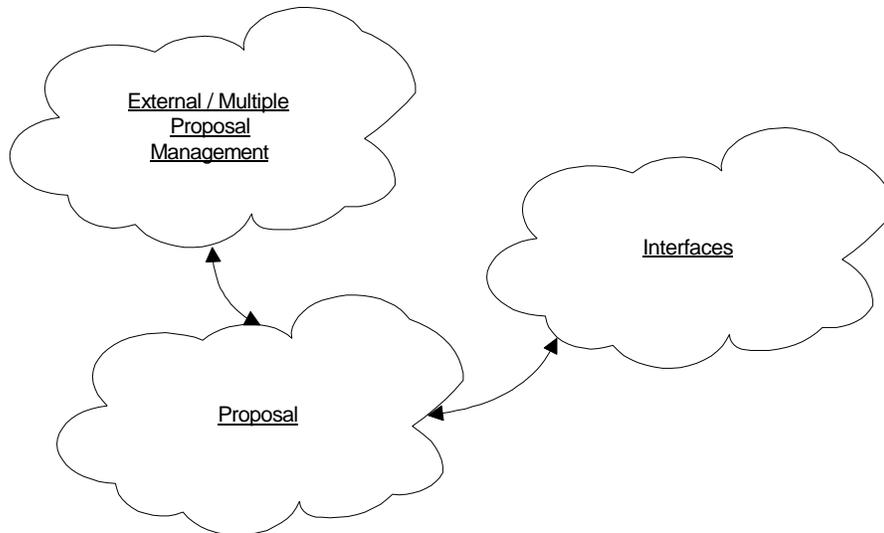
Because the SEA is written in Java, it should run on any platform that provides a compatible implementation of the Java virtual machine. If the user opts to run the SEA as an applet, the user must have a Web browser that supports version 1.1.4 or later of the Java Development Kit (JDK). The SEA will initially support Netscape Navigator from Netscape Communications and HotJava from Sun Microsystems. If the user runs the SEA as a local application, the user must have installed a Java virtual machine that supports version 1.1.4 or later of the JDK.

4 Conceptual Model

This chapter will focus on describing the various "concepts" that we believed are involved in the SEA in non-technical analytical terms. The conceptual model is presented in a top-down fashion with a high-level look at the model and the focusing in on the more detailed pieces.

4.1 High Level

At the highest level, the SEA will have three main areas:



The Proposal contains all the information for a single proposal, this includes such items as specifications for one or more exposures, information on grouping and ordering of the exposures, knowledge for validating the proposal.

The Interfaces domain contains all the various means for editing a proposal. This will include the various Graphical User Interfaces (GUI) whether they be interview-style or browser-style.

The External domain will include concepts such as managing multiple proposals and transmitting proposals to other sites.

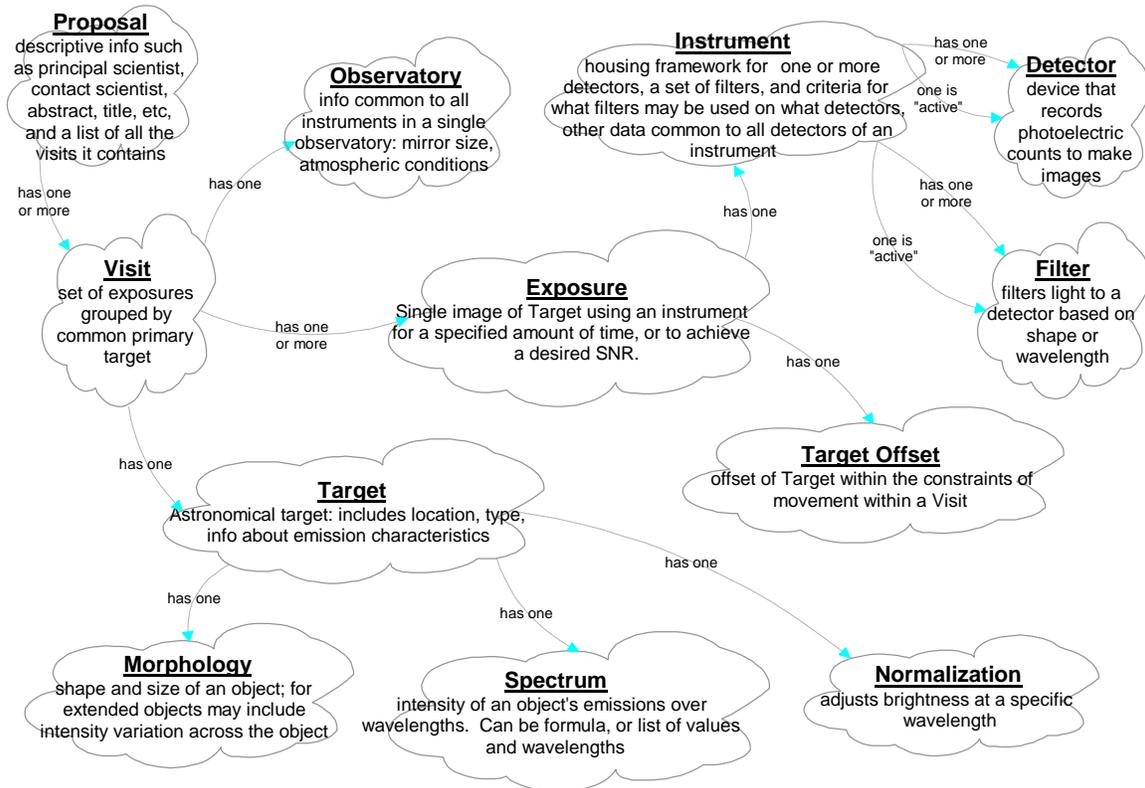
4.2 Proposal Data Model

This group of concepts focus on the definition of a proposal and the various astronomical concepts contained in a proposal.

4.2.1 Proposal

The Proposal contains all the information for executing a Phase II observing program. This includes descriptive information about the proposal such as title, authors, and abstract and a series of exposures that are grouped into Visits. It also includes the knowledge for validating the current proposal data and for evaluating (and possibly rejecting) changes.

The "core" of the proposal is a list of one or more Visits.



4.2.2 Visit

In the simple form, a Visit is collection of related exposures that share a common primary target. While the individual exposures may small offsets, the telescope should be able to execute all of the exposures in a single visit using the same target "locks" (for HST, this means using the same set of guide stars). In the initial releases of SEA tools, the Visit will remain very simple. As the "Visit Planner" gets designed and developed further, the Visit will gain additional knowledge about ordering and validating its group of exposures.

4.2.3 Exposure

An Exposure combines the information necessary to specify a single image. This includes configuration specifications for the instrument, information about the target to be observed, other specifications about the observatory and observing conditions, and the length of time of the exposure. In addition the exposure should be able to calculate various predicted photon counts and their sources that are accumulated in the image. The target for an exposure should be the same as the parent target for the Visit, or an allowed offset from that target that means the constraints of remaining within the visit

4.2.4 Instrument

An Instrument contains the specifications for a single instrument involved in a single exposure. It includes identifying the selected detector, filter, and additional instrument configuration parameters. It will also include characteristics of an instrument including such things as a list of the detectors it contains, the filters, the valid assignments of filters to detectors.

4.2.5 Detector

A Detector contains information specific to a single detector. This includes things such as width and height in pixels, the size of a pixel, and information about the detector's sensitivity and its bright limits.

4.2.6 Filter

The Filter contains information and knowledge specific to a particular filter.

4.2.7 Astronomical Object

An Astronomical Object is an entity in space that is the target of an observer. An astronomical object has information such as a location, and distance and velocity relative to the earth, a shape and size, a spectral model, and normalization factors to adjust the spectral model's brightness and redshift to the specific object. For the SEA one the key issues is how intensely an object emits light in different places at different wavelengths. For defining this, an object has:

- A Morphology which defines its shape and size and its intensity a different points within the object;
- a Spectrum which defines how the object radiates at different wavelengths and possible different places,
- and a Normalization which adjusts the spectral model for brightness and at user specified wavelength.
- a distance or redshift value.

All of these combine to determine what actual observed emissions from the astronomical object are likely to be.

Caveats: the description does not currently provide for an object whose emissions vary over time. Nor does it accommodate a larger astronomical object that might have an entirely different spectral emission pattern at different places.

4.2.8 Morphology

An astronomical object has a defined shape or Morphology. This can be as simple as "point source", or a "flat" circle of a specified diameter. Or it could be more complex as in the case of a galaxy.

4.2.9 Spectrum

Describes an set of emissions across a series of wavelengths. Several standard spectral models exists such as Power-Law, and Black Body. In addition a Spectrum can be based on a series of measurements made on a specific object. An object's actual spectrum is modified by its normalization, redshift, and sometimes in morphology to determine the "observed" spectrum.

4.2.10 Normalization

The Normalization of an spectrum adjusts the spectral model's intensity based on a specified intensity at a specific wavelength.

4.2.11 Observatory Parameters

The Observatory contains information and knowledge that pertains to an observatory and is common to all of its instruments. This include things such as primary mirror size and other attributes, atmospheric data. The Observatory should know how to return information about the primary mirror, and how background light and noise is affected by the atmosphere or other attributes of the Observatory.

In the case of NGST and HST, zodiacal light (background emissions deflected from the sun) and earth shine (background emissions related to the earth) are two important parameters.

While this concept needs to be modelled, we do expect that the information contained in the observatory parameters to be modified very much by a user.

4.2.12 Aperture

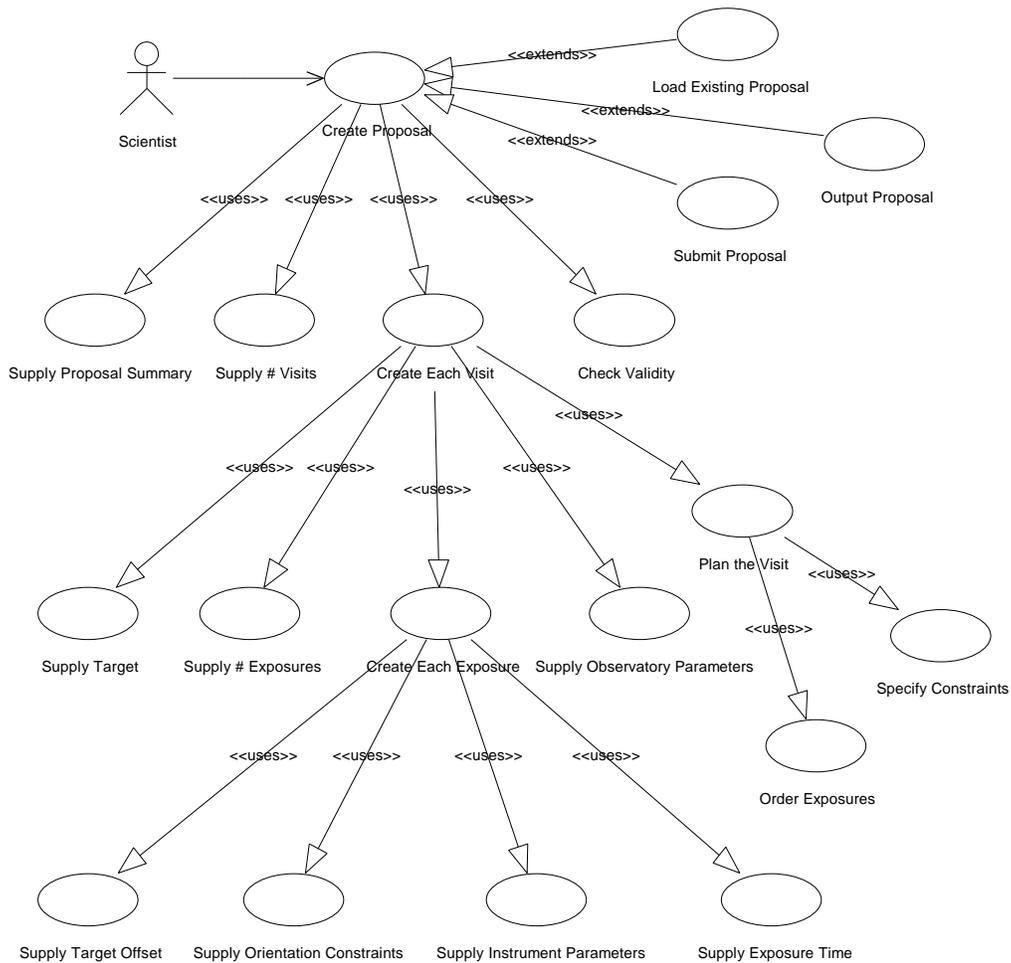
The Aperture defines information about the shape and size of the viewable area of an instrument configuration (including the detector and filter applied).

4.2.13 Wavelength

Contains a wavelength value and its defined unit of measurement. A wavelength will also know how to convert its value to other units of measurement.

4.3 Use Cases

The following Extended Use Case Diagram illustrates the cases that comprise the creation of a proposal. The user is represented by a single actor, the scientist or General Observer (GO). The actor initiates the “Create Proposal” use case. Each use case represents a specific sequence of events that occur when the user interacts with the system. If a “<<uses>>” arrow exists from use case A to use case B, it indicates that A is dependent on and requires B. If an “<<extends>>” arrow exists from use case A to use case B, it indicates that B may also contain A.



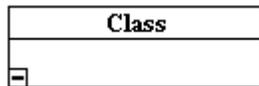
5 Design Model

The description of the design model is organized into five sections. The first describes the notation used in the class model diagrams. The second illustrates the user interface (UI) design with examples of the main dialogs and descriptions of the underlying UI concepts. The third section describes the high-level design, emphasizing the breakdown of the SEA into subsystems and how those subsystems interact. The fourth section explains the inner workings of each subsystem by describing the classes contained within them. The last section illustrates how these classes interact with each other.

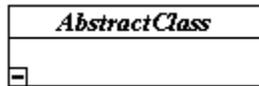
5.1 Diagram Notation

The following notation is used in the class diagrams throughout this section.

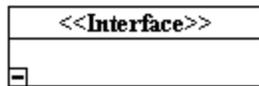
Classes and Interfaces:



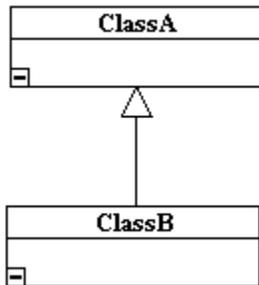
Non-abstract class



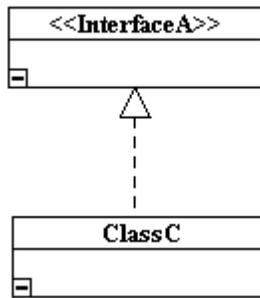
Abstract class



Interface



Inheritance. Class B inherits from, or extends, Class A.



Implementation. Class C implements Interface A.

Variables and Methods:

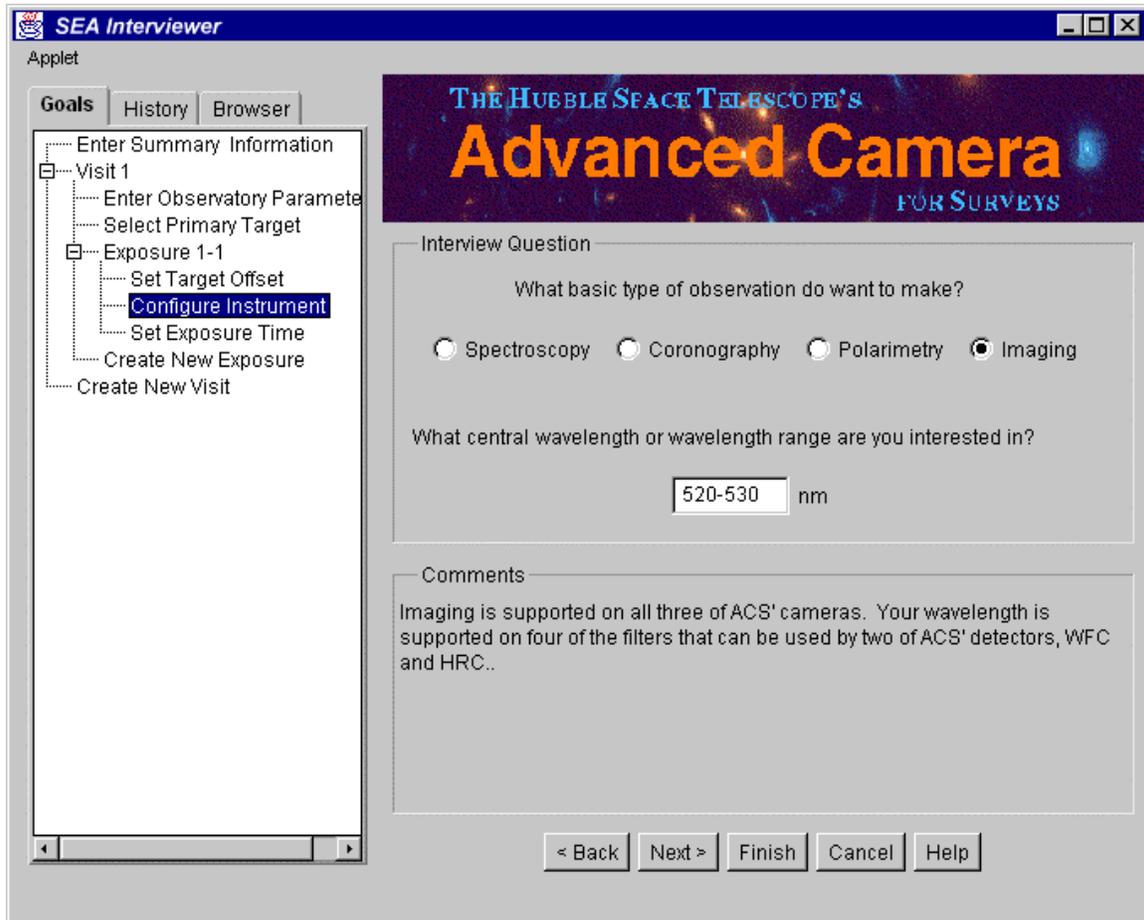
-  Public or Package Variable
-  Private Variable
-  Protected Variable
-  Public or Package Method
-  Private Method
-  Protected Method

5.2 User Interface

One of the main goals in designing the SEA is that it be flexible enough to appeal to users with varying degrees of experience in writing proposals. The SEA should be the ideal tool for scientists who have never submitted a proposal before and want to be guided through the process. Yet the SEA should also be the ideal tool for experienced scientists who know how the proposal process works and want more flexibility in working with the tool. To that end, the SEA includes two different modes for working with the proposal.

5.2.1 Interview Mode

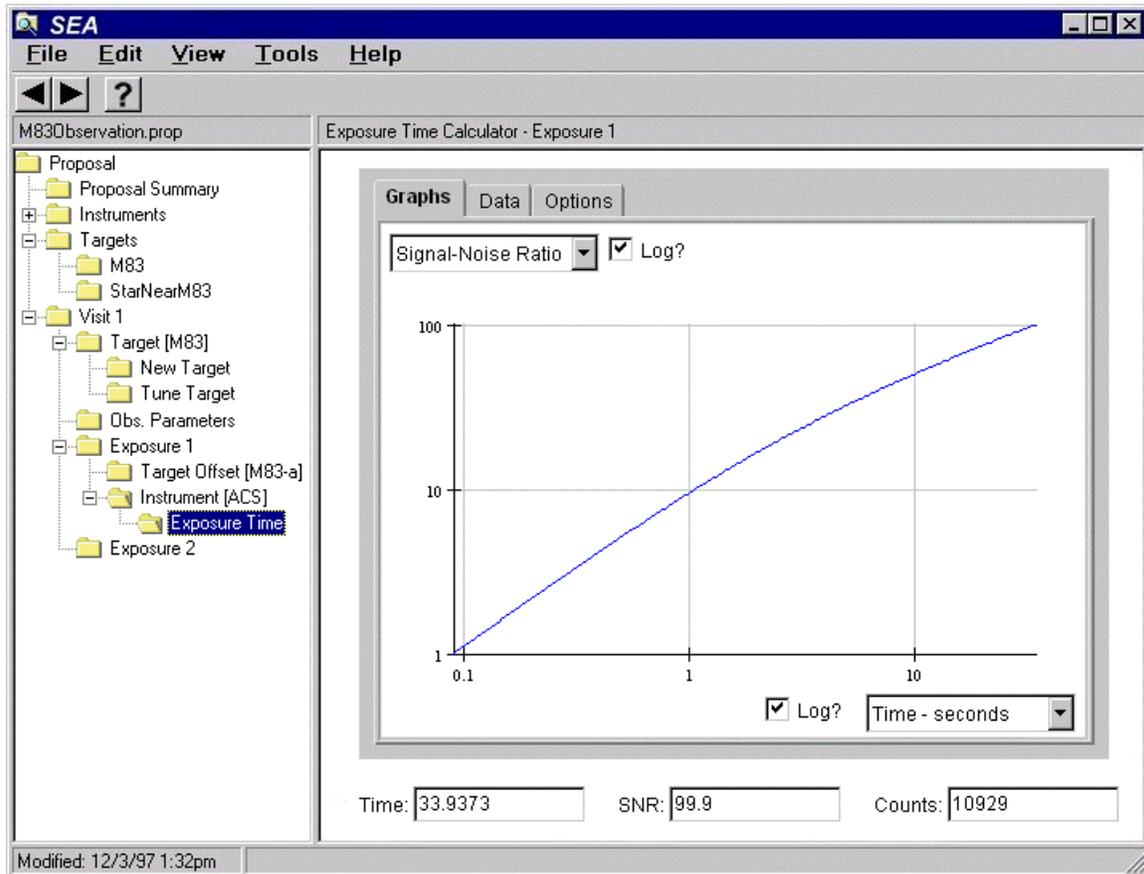
The interview mode allows the user to write their proposal through a series of steps. Each step will ask the user information about the science that they want to achieve. This information will be fed into the expert system which will determine how the information translates into a proposal. The user interface for this mode will be a Wizard, a commonly used technique on the Microsoft Windows platform that asks the user a series of questions and allows the user to step forward or backward in the list of questions. The user may switch between the interview mode and the browser mode at any time.



This is an example of what the interview window might look like. The area on the left would contain a list of goals with the current goal selected. The area on the right would contain instructions and questions for the user to answer. In some cases, a simplified version of a Proposal Browser editor might be used, for example when determining the exposure time. In most cases, however, the expert system would generate a list of questions represented as entry fields or choice boxes. The interview window also includes a comments section that contains notes on the user's choices and how they affect other goals in the interview.

5.2.2 Browser Mode

The browser mode is intended for users who understand the proposal process and want maximum capability in editing their proposal. In this mode, the user views the proposal with the Proposal Browser window. This window resembles the Microsoft Windows Explorer interface, allowing the user to quickly navigate through the proposal by selecting elements in a tree view of the proposal.



On the left, the contents of the proposal are represented as a tree. Please note that in the final release, the generic folder icons will be replaced with icons that represent specific areas of the proposal. When the user selects an item in the proposal tree, the editor for that item will appear on the panel to the right. The user may jump to any area of the proposal at any time. This technique allows only a single editor to be open at once, similar to a page in a Web browser. To overcome this limitation, the Proposal Browser also allows the user to open an editor in its own separate window by double-clicking on the item in the proposal tree, or selecting an “Open in New Window” option from the menubar. This allows the user to have many different editors open at once. In this case, all editors are linked to the proposal such that if the user makes a change in one editor, it is automatically propagated to all other editors.

5.2.3 Concepts

The following are key concepts that have guided the design of the user interface:

5.2.3.1 Automatic change propagation

The SEA design enforces the idea that if the user makes a change in one area of the proposal, that change is automatically propagated to all other areas of the proposal. If, for example, the user has both the Exposure Time Calculator and the Target Selector open, and they change the target type from a Point Source to an Extended Source, the Exposure Time graph would automatically update to reflect the properties of the new target.

5.2.3.2 Drag and drop

Drag and drop will be supported wherever possible. The user should be able to, for example, copy a target object from one exposure to another exposure by dragging it. We are investigating using the Drag and Drop API in the Java Foundation Classes.

5.2.3.3 Navigation history

The SEA should maintain a history of the user's navigation decisions, and allow the user to go backwards and forwards in that history. This would be similar to the history feature found on most Web browsers. The arrow buttons shown at the top left of the previous images would serve this function.

5.2.3.4 Help

The SEA will have three different kinds of help:

5.2.3.4.1 Global Help

Global help will consist of HTML documentation accessible from the Help menu. This might include a user's guide, or reference documentation related to the proposal or instrument.

5.2.3.4.2 ToolTips

The SEA will support ToolTips-style help. This has become a standard feature in most applications, where a small one-line description of an item is displayed at the cursor if the cursor pauses for a few seconds.

5.2.3.4.3 Context-Sensitive Help

Context-sensitive help allows the user to get help on any user interface item by clicking on a help icon and then clicking on the item in question. Clicking on the item displays a small window of HTML documentation related to the item or the contents of the item. This should be very similar to the context-sensitive help feature found on Windows 95/NT. The question mark icon shown at the top left of the previous images would initiate the help operation.

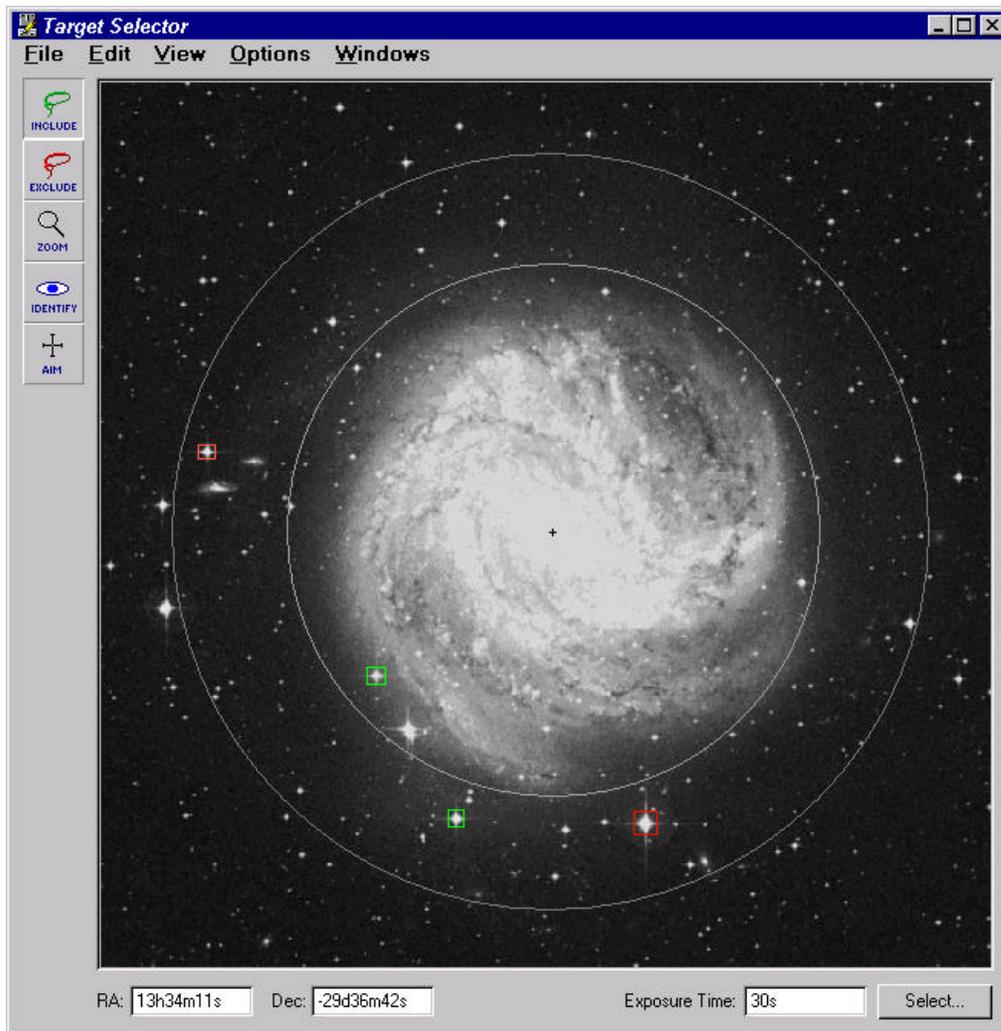
5.2.4 Exposure Time Calculator

The Exposure Time Calculator (ETC) generates real-time interactive graphs showing Signal-Noise Ratio and Source counts across a range of exposure times and wavelengths. The user can manipulate the graphs by zooming in or out of specific areas. The user can also specify what type of data is displayed in the graph by selecting a data type from the choice boxes at the graph's axes. Selecting a point on the plot causes its data value to be inserted into the corresponding entry field on the bottom of the ETC. Because these fields display the most important data: exposure time, signal-to-noise ratio, and source counts, they are always shown. The graph, however, can be replaced with a tabular view of the graph data by selecting the "Data" tab. Likewise, selecting the "Options" tab replaces the graph with a series of options that allow the user to tweak the display of the graph. The "Graphs" tab returns to the graph display. The following image shows the ETC with the "Graphs" tab selected.



5.2.5 Visual Target Tuner

The Visual Target Tuner (VTT) allows the user to visualize the area surrounding the target and to tune the target position. The VTT also allows the user to specify constraints for the orientation of the instrument. These constraints are specified as points or regions to include or exclude from the exposure. The SEA can then calculate the set of valid orientations given those constraints. The overriding design goal for the VTT is to create a highly visual and interactive environment for dealing with the position of the target. The user should be able to easily move around the target area, zoom in or out of the visualization, identify objects on the visualization, and be able to drag the target to change its position. A future release of the VTT will allow the user to perform a centroid fit of the target position. This later release will also simulate image artifacts such as diffraction spikes and CCD bleeding, and will provide the ability to show a model of the target area as well as a FITS image.



This sample of an early VTT prototype illustrates the more important user interface concepts. The visualization is clearly the most important item and thus contains most of the window area. One or more FITS images of the target area are contained within the visualization, along with a simple crosshair to indicate the current position of the target. The target position is also shown as a set of coordinates in the bottom of the window. The user may change the position by dragging the target or changing the values in the entry fields.

The visualization shows two concentric circles: the inner circle contains the area that will be included regardless of the orientation, and the outer circle contains the area that may potentially be included. Orientation constraints are specified by selecting the inclusion or exclusion tool, and then selecting either an object or region in the visualization. Areas that must be included in the exposure are shown in green. Areas that must be excluded are shown in red. In the future, the tool will visually indicate the set of possible orientations, perhaps by shading the areas that will be excluded from the exposure. The user can also open the Orientation Constraints window to see how their constraints affect the schedulability of their proposal. They may monitor how the schedulability changes as they alter their inclusion and exclusion points in real-time.

The upper left corner of the VTT window contains the set of tools that may be used on the visualization. In addition, certain visualization options may be set using items under the “Options” menu. The “View” menu provides access to secondary VTT windows. These include:

- an Orientation Constraints window that shows the inclusion and exclusion points as a table. This window also displays summary information about the schedulability of the proposal given the current set of constraints.
- an Image Chooser window that allows the user to specify the FITS image files to be shown in the visualization. The user may choose from the images retrieved from an astronomical database, or may specify a local image to use.
- an image tools window that allows the user to manipulate the images contained in the visualization. Images may be adjusted by changing their brightness, contrast, color table, or toggling the negative of the image.

5.3 Architecture Model

This section describes the overall architecture and structure of the various classes in the SEA system. Following a high-level overview, we'll work down through the class hierarchy in greater detail.

5.3.1 High Level Overview

At a high-level, the classes in the SEA are broken down very similarly to the conceptual layout. This trend is reflected in the package structure of the objects as they are implemented in the Java language. The classes are grouped as follows:

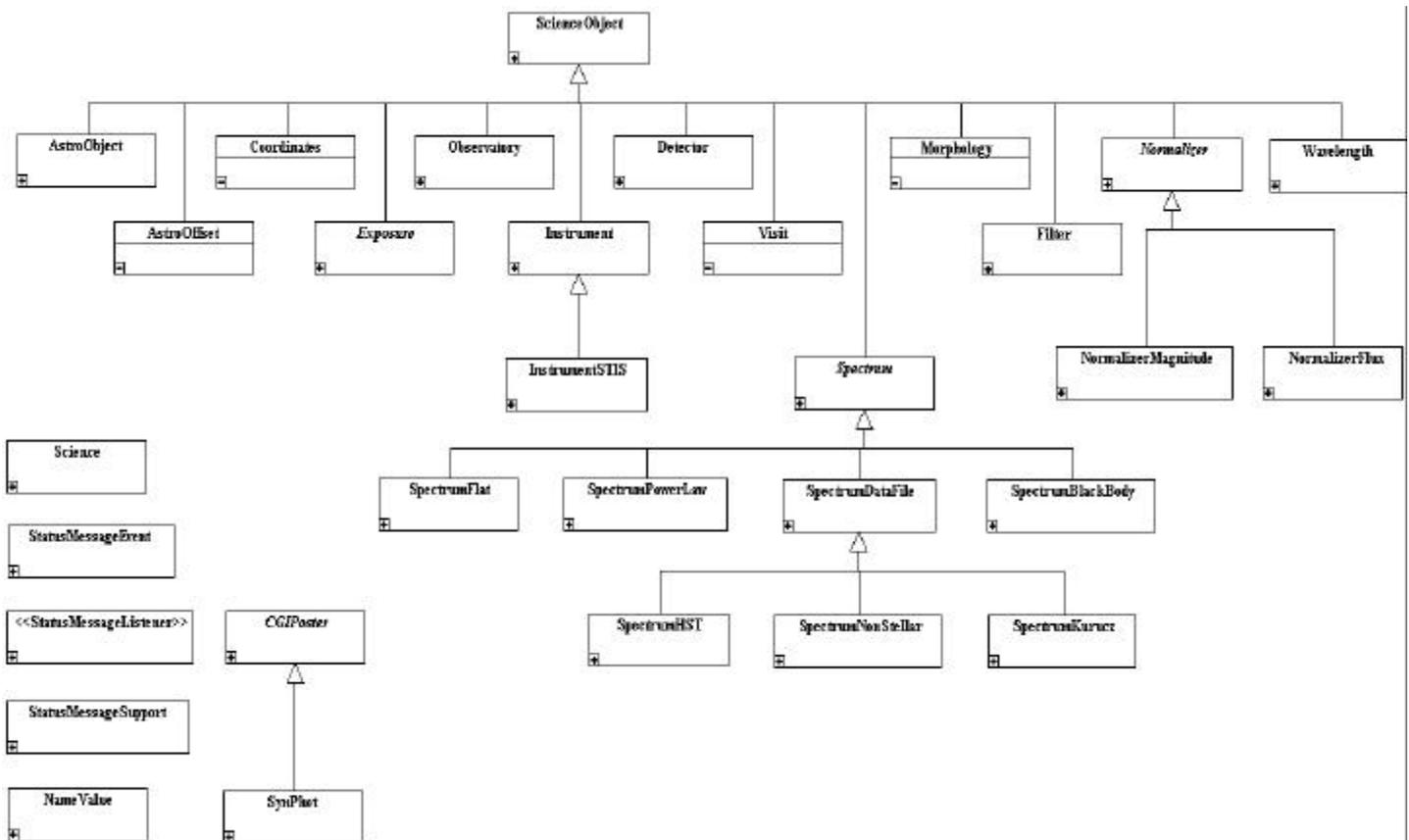
- Science Objects: these classes represent the objects in the Proposal data model. The focus of classes here is on the science and data contain in the data model. Emphatically, user interface related information is NOT a part of these classes.
- Expert System Objects: this grouping contains classes that provide an interface between the Science Objects and the Expert System engine. It includes classes that support the management of the questions and answers for the Interview user interface.
- User Interface Framework: this grouping contains the classes that implement the user interface components for the browser and interview windows. It includes classes that provide a user interface presentation layer onto objects in the proposal data model.
- User Interface Modules: user interface components that allow the user to view or edit a particular area of the proposal are called Modules. Each module's classes are contained in a separate package. The initial release of the SEA will include:
 - Exposure Time Calculator: this module currently provides an interface to the Exposure object and preliminary viewers for Instruments and AstroObjects. The exposure time calculator is the first of the browser modules to be implemented. Its prototype implementation has served as a design test-bed. This design description is for the original prototype release.
 - Visual Target Tuner: this module provides an interface for visualizing the target area and allowing the user to tune the target position and specify orientation constraints.
 - Target Selector: this module provides an interface for searching astronomical databases and retrieving target information. The user can then use that information to choose a target.
 - Instrument Editor: this module provides the browser-style interface for configuring the instrument. Because it allows the user to edit individual instrument parameters, it is intended for users who are familiar with the instrument.
 - Proposal Summary Editor: this module provides the browser-style interface for editing proposal summary information.

- **Component Viewer:** this module shows an icon view of the contents of an arbitrary component of the proposal. It is used when no editor exists for a given area of the proposal.

The design of the modules is intended to allow a data model to be updated either in browser mode or in interview mode. This allows the user to use the style of update that suits them best and different parts of the development process. It also provides a framework for allowing the various components to be developed in parallel and allows a system framework to be prototyped before all the various viewers are completed.

5.3.2 Science Objects

The science package contains all classes that represent astronomical concepts or data. Every class in the Proposal data model is such a class. Many of these classes correspond to one of the specific astronomical concepts described in the Conceptual Model. All of these objects classes inherit from the root ScienceObject class. The following diagram shows the inheritance tree for the science package.



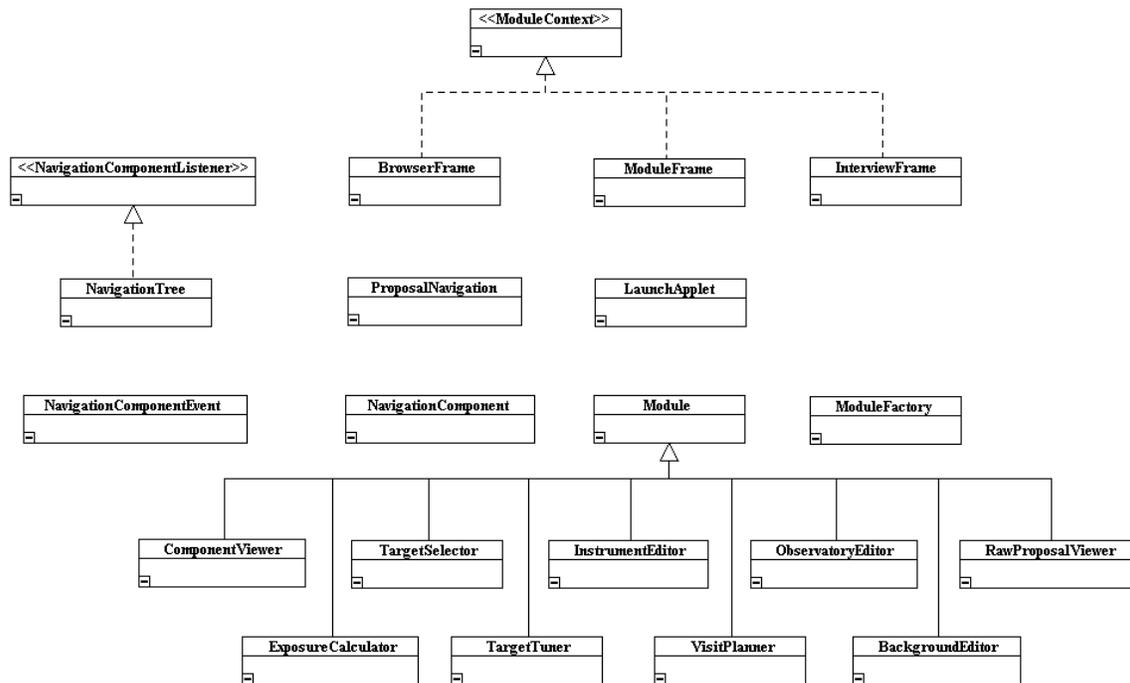
5.3.3 Expert System Objects

The design of the Expert System was not complete at the time of this writing. We have decided to delay delivering an Expert System design until we become more familiar with the Advisor/J

product. An addendum to this document will be delivered on our Web site (<http://aaadev.gsfc.nasa.gov/NGSTProtos/>) within the month. You may also request a hardcopy by sending e-mail to Jeremy.E.Jones@gsfc.nasa.gov.

5.3.4 User Interface Framework

The main SEA user interface is represented as a proposal browser or interviewer, depending on the mode selected by the user. Both the proposal browser and the interviewer are windows that allow the user to modify a proposal. Both windows contain Modules which allow the user to view and edit a specific area of the proposal. The following diagram shows the classes that comprise this package and their inheritance relationships.

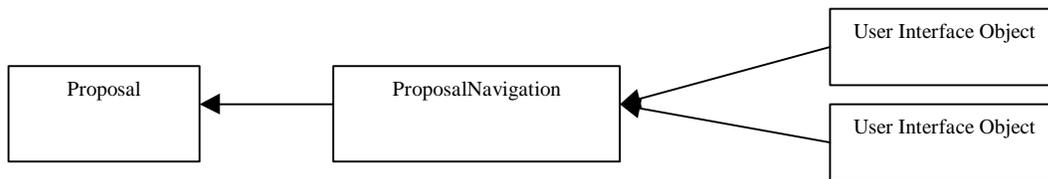


This diagram shows that the various parts of the proposal each have an editor class that inherits from Module. Module is a subclass of java.awt.swing.JPanel and can be inserted into any of the Frame classes shown. An additional design goal was to allow an individual module to be reused as an applet on a Web page, rather than as part of the SEA. To accomplish this, the ModuleContext interface was created, and the SEA frames implement this interface. A Module may be contained in any object that implements the ModuleContext interface, and an applet can easily do this.

5.3.4.1.1 Proposal Navigation

The proposal browser presents the overall structure of the proposal to the user. This structure loosely corresponds to the internal structure of the proposal data model, but differences exist between the internal structure and the structure that should be presented to the user. To separate these two structures, another data model was created to insulate the user interface from the underlying proposal data model. ProposalNavigation accomplishes this. Instead of user interface

objects connecting directly to the Proposal object, they connect to the ProposalNavigation object. The ProposalNavigation object connects to the Proposal object.

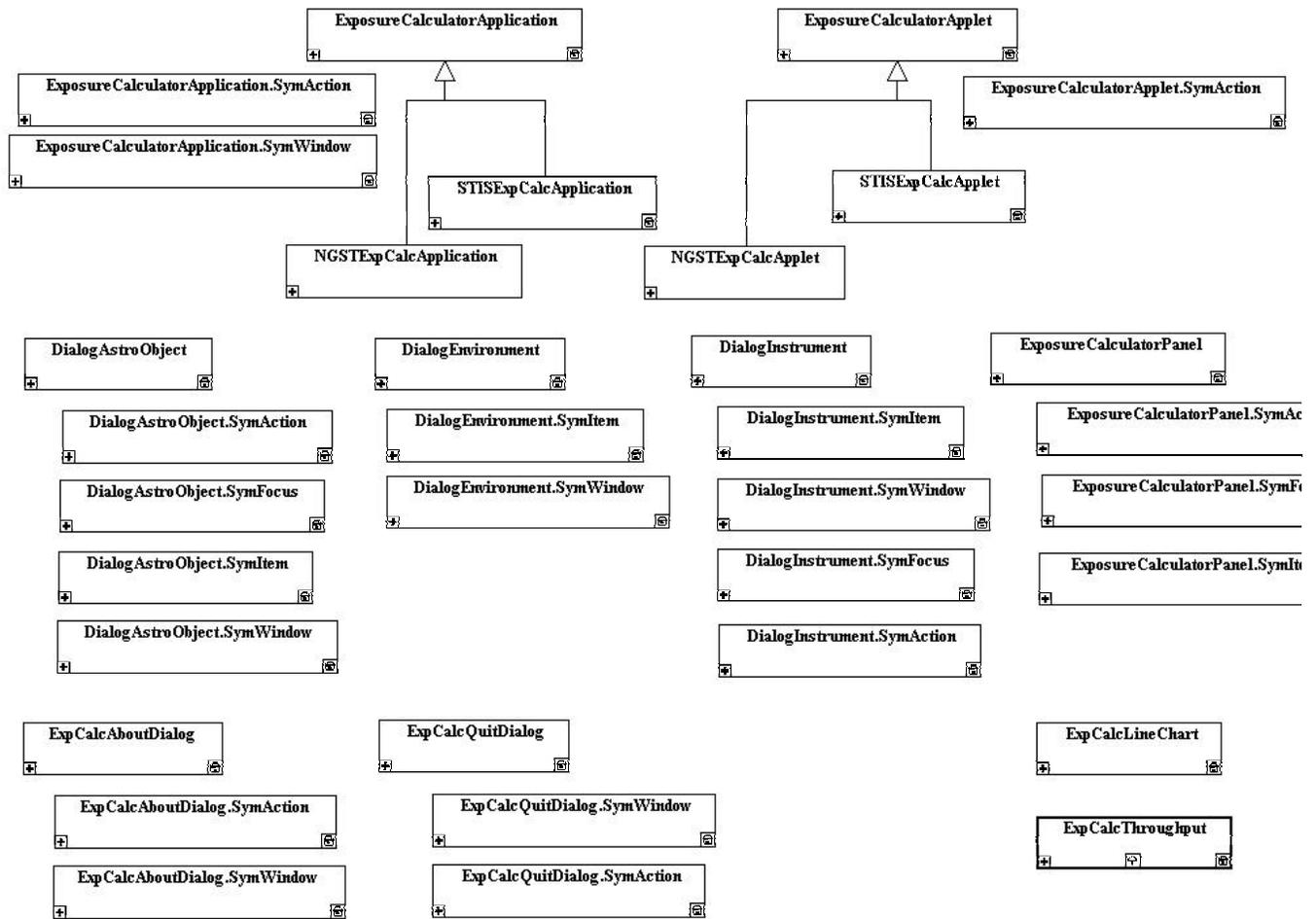


ProposalNavigation contains NavigationComponents that loosely correspond to the separate ScienceObjects contained within the proposal. Each NavigationComponent contains a descriptive name and icons. It also knows how to launch the appropriate editor for its underlying ScienceObjects.

5.3.5 User Interface Modules

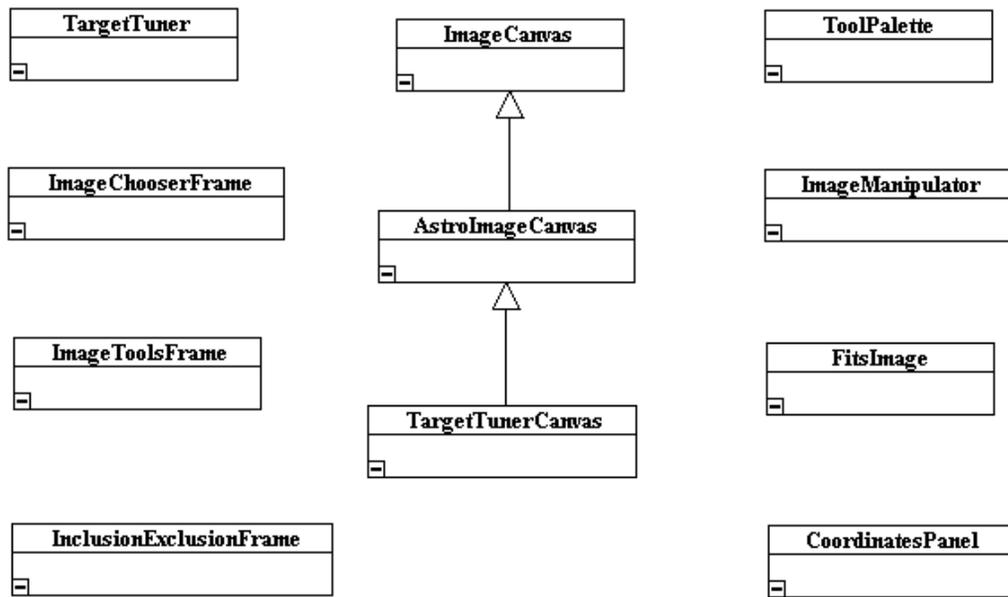
5.3.5.1 Exposure Time Calculator

The Exposure Time Calculator (ETC) has served as a testbed for the SEA design. A standalone prototype of the ETC has been built and is the basis for this design model. It includes prototypes for items that will later become separate modules, such as the instrument editor and target selector. The following diagram shows the classes that comprise the ETC.



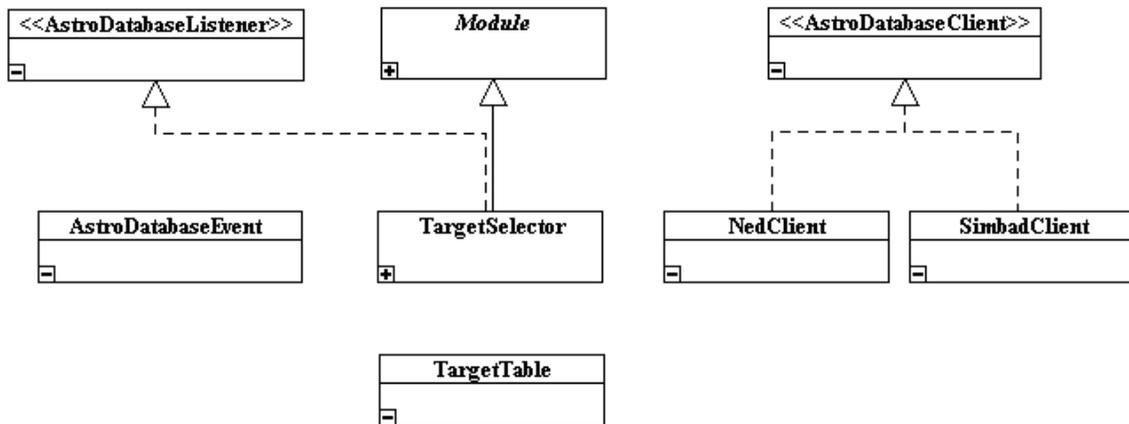
5.3.5.2 Visual Target Tuner

The Visual Target Tuner (VTT) allows the user to visualize the target area and tune the target position. It also allows the user to specify constraints on the possible orientation of the instrument. The VTT is implemented as a subclass of Module. It mainly consists of a large canvas area (TargetTunerCanvas) that contains the visualization of the target area. Most of the VTT implementation is contained within the visualization canvas. The VTT also includes dialogs that allow access to additional features for manipulating the visualization (ImageChooserFrame, ImageToolsFrame, and InclusionExclusionFrame). The following diagram shows the high-level classes in the VTT package.



5.3.5.3 Target Selector

TargetSelector is a subclass of Module that provides a user interface to various astronomical databases. The TargetSelector panel contains a set of input fields where the user can specify search parameters. The user can also select a particular database (NED or SIMBAD) to search, in which case a new AstroDatabaseClient of the proper type is created. When the user selects the “Search” button, TargetSelector asks the AstroDatabaseClient to initiate the search. It receives results through the AstroDatabaseListener interface, which TargetSelector implements. This allows the search to continue in a separate thread and the results to be delivered asynchronously. Once the results are received, they are displayed in the TargetTable where the user may select targets and add them to the list of proposal targets or assign a target to a particular visit.



5.3.5.4 Instrument Editor

The Instrument Editor allows the user to edit individual instrument parameters. It includes a single class, InstrumentEditor, that is a subclass of Module. InstrumentEditor is an abstract class that would be implemented for each type of instrument, since each instrument has its own

parameters and rules for validation of those parameters. Subclasses will display selectable widgets for each of the parameters contained within the corresponding Instrument object. These parameters include CCD, filter, binning, gain, and CR split.

5.3.5.5 Proposal Summary Editor

ProposalSummaryEditor is a subclass of Module that allows the user to input summary text information about the proposal, such as contact information for the General Observer (GO) and a textual explanation of the proposal goals. These items are shown as simple text fields that the user can edit.

5.3.5.6 Component Viewer

The design of the Component Viewer is quite simple. ComponentViewer is a subclass of Module, and since Module is a subclass of java.awt.JPanel, the component viewer already contains the necessary drawing area for rendering icons. ComponentViewer includes a reference to the NavigationComponent that it represents. It draws the contents of the NavigationComponent using the icons stored within the NavigationComponent's children. It also provides a text entry field where the user may rename the Proposal data object specified by the component. ComponentViewer traps mouse events on the drawing area to determine if the user double-clicks on an icon. If so, ComponentViewer asks its ModuleContext to launch the appropriate Module for the selected icon.

ScienceObject	
	fChildren
	fListeners
	pAnnotation
	pCreateDate
	pLastModified
	pName
	pNoNameIndex
	ScienceObject
	ScienceObject
	addChild
	addPropertyChangeListener
	children
	clone
	equals
	firePropertyChange
	getCreateDate
	getLastModified
	getName
	getAnnotation
	hashtableEquals
	propertyChange
	readObject
	removeChild
	removePropertyChangeListener
	resetModifiedDate
	saveAsText
	saveAsText
	setCreateDate
	setLastModified
	setName
	setAnnotation
	toString

5.4 Design Class Model

This section describes the most important SEA packages in detail. The important classes in a package are described, along with diagrams illustrating the principle operations, data members, and relationships to other classes.

5.4.1 Science Package

5.4.1.1 ScienceObject

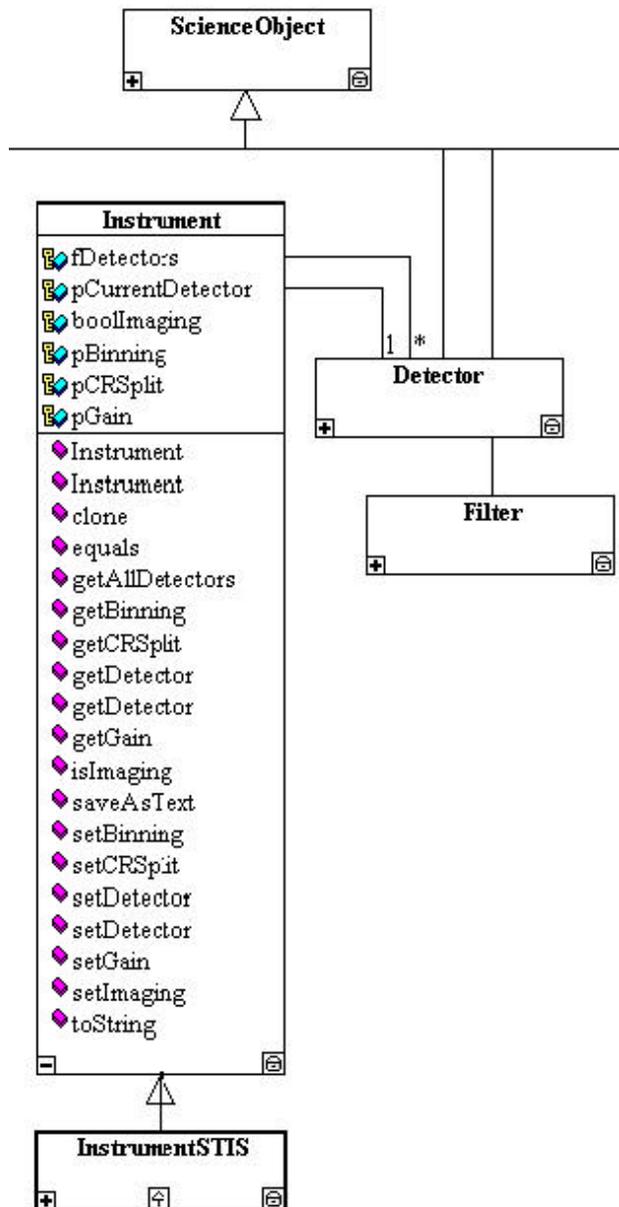
ScienceObject is the parent class for all of the scientific object classes. It provides common functionality for all the classes in the science data model. This includes:

- maintaining common properties such as name, creation date, last modification date, and annotation text
- providing support for property change handling,
- support for managing member (child) objects

Support for property change handling is integrated with support for managing child objects. If a subclass of ScienceObject (say, Instrument) wants to maintain a child object (say Detector), it should avoid managing the child object itself, and instead register the child by use of ScienceObject's addChild(childobject) method. Once added, the ScienceObject will handle such tasks as

persistence of the instance's children, propagating change information to and from children, and ensuring that children are properly handled when replicating the object.

5.4.1.2 Instrument

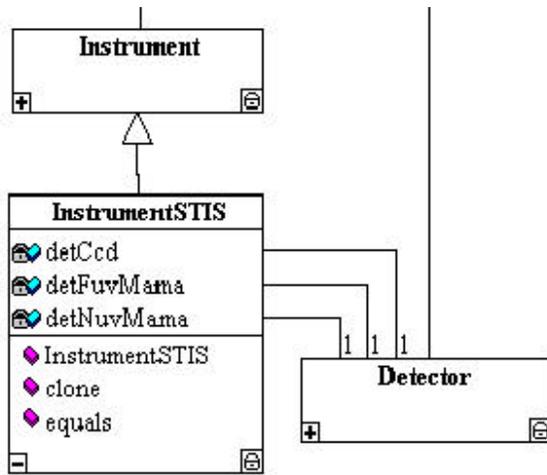


An instrument is responsible for information and actions related to a specific instrument within an observatory. Each instrument has a list of detectors, knows its currently selected detector, and has additional properties such as:

- CRSplit, the number of independent images that would be taken with the current configuration.
- Binning, the aggregation of pixels within the instrument's detectors.

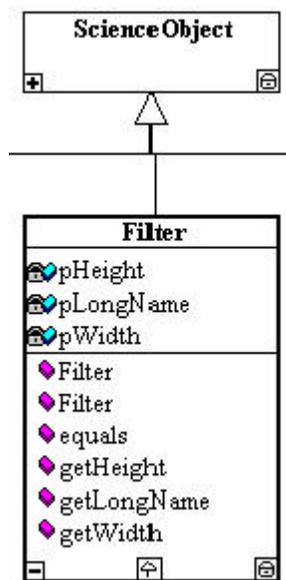
Instrument includes functionality that should be common to all instruments. However, it is expected that a subclass of Instrument will be needed for each specific instrument supported (i.e. InstrumentACS would extend Instrument). These subclasses will provide support for parameters unique to their specific instrument, and might override default behavior.

5.4.1.2.1 InstrumentSTIS



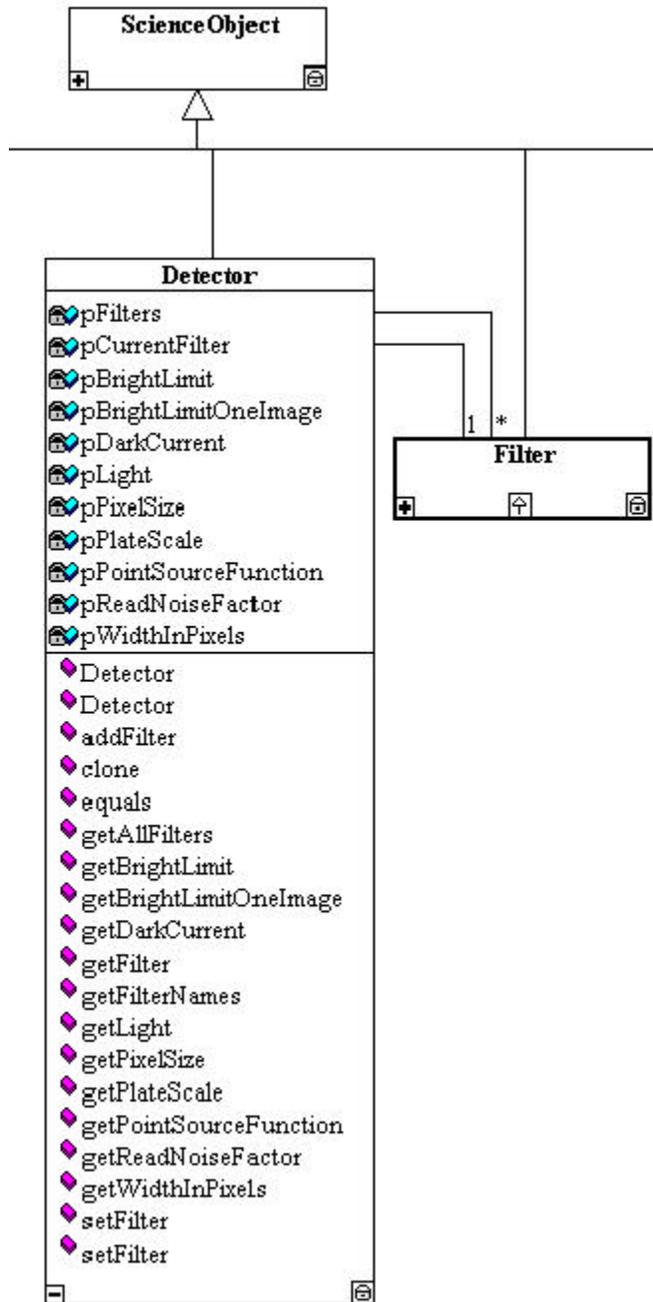
The InstrumentSTIS is a subclass of Instrument to handle the STIS instrument within the Hubble Space Telescope. It has no additional functionality beyond Instrument, but does contain three detectors each of which has a list of filters.

5.4.1.3 Filter



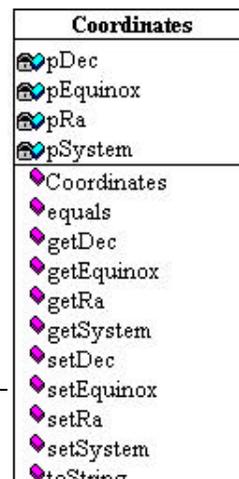
Filter contains information related to a filter within an instrument. Currently this information is limited to the properties of width, height, and a descriptive name. Each of these properties is read-only except when the filter is defined.

5.4.1.4 Detector



Detector contains information and knowledge pertaining to an instrument's detector. Currently, a detector contains a list of valid filters and the currently selected filter. It also contains the following read-only properties (defined when the detector is created):

- BrightLimit: the maximum number of photons the detector can accumulate in a single pixel across multiple "splits".
- BrightLimitOneImage: the maximum number of photons the detector can accumulate in a single pixel in a single image or split.
- DarkCurrent: the amount of dark current inherent in the detector.
- PixelSize: the size of a single pixel in the detector (assumed to be square).
- PlateScale: the platescale of the detector.
- PointSourceFunction: the PSF of the detector. NOTE: while this is currently a single value, it may grow in the future to be a true function based on one or more other parameters.
- ReadNoiseFactor.
- WidthInPixels: the width of the detector in pixels (assumes a square detector). Note that Height is likely to be added shortly.

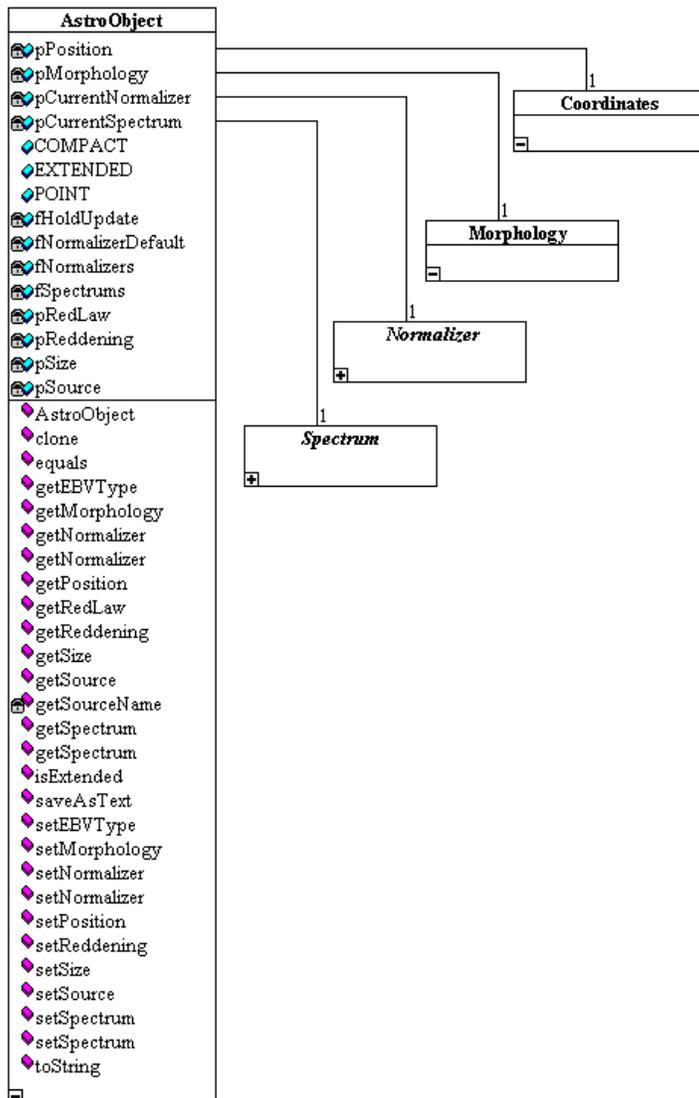


5.4.1.5 Coordinates

Coordinates is a class that represents a position in space. It provides

support for representing position as a right ascension and declination pair. It includes the ability to set the equinox and coordinate system, and to convert between two equinoxes or coordinate systems.

5.4.1.6 AstroObject



AstroObject represents a single astronomical object in space. This class is designed to contain information relevant to an astronomical object itself and is specifically not designed to incorporate any assumptions about an object being used to observe the AstroObject.

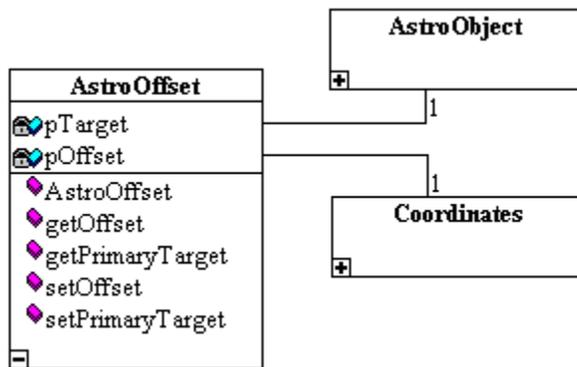
An AstroObject has a position, represented by a Coordinates object, and a morphology that defines the shape and size of the object.

In addition, AstroObject contains a spectrum and normalization information for relating the spectral model to the specific brightness of the object.

The object also has the following properties:

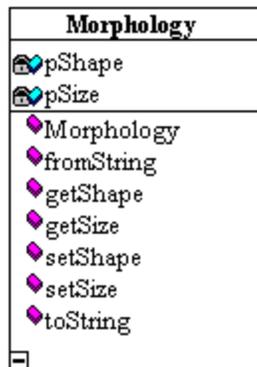
- Object Type
- Redshift
- Redlaw
- Reddening
- Source
- Bibliographic references

5.4.1.7 AstroOffset



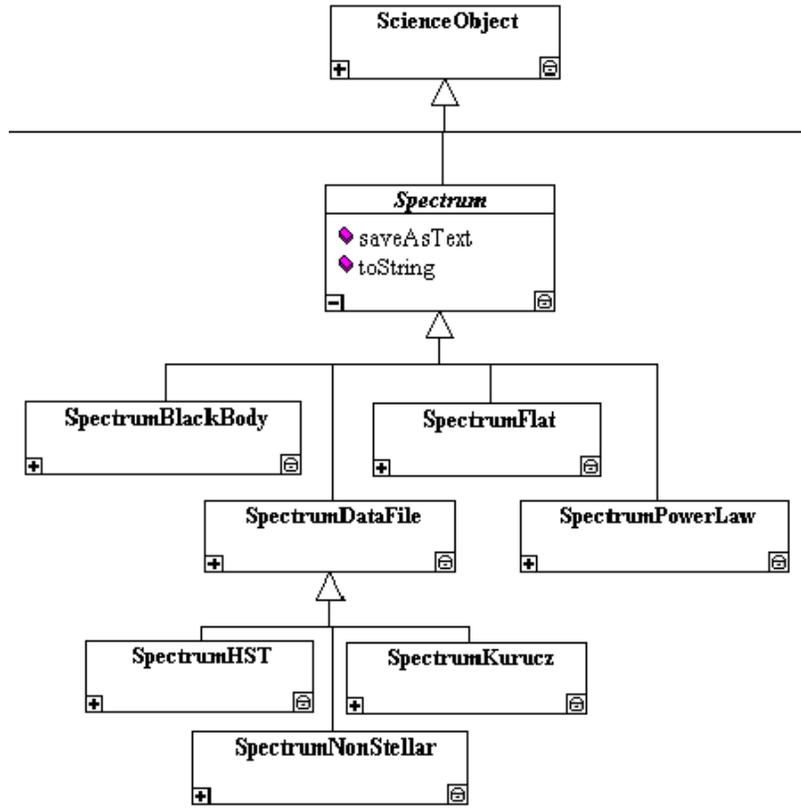
AstroOffset is used to indicate a target that is an offset relative to some primary target. It contains the offset coordinates and a reference to the primary target.

5.4.1.8 Morphology



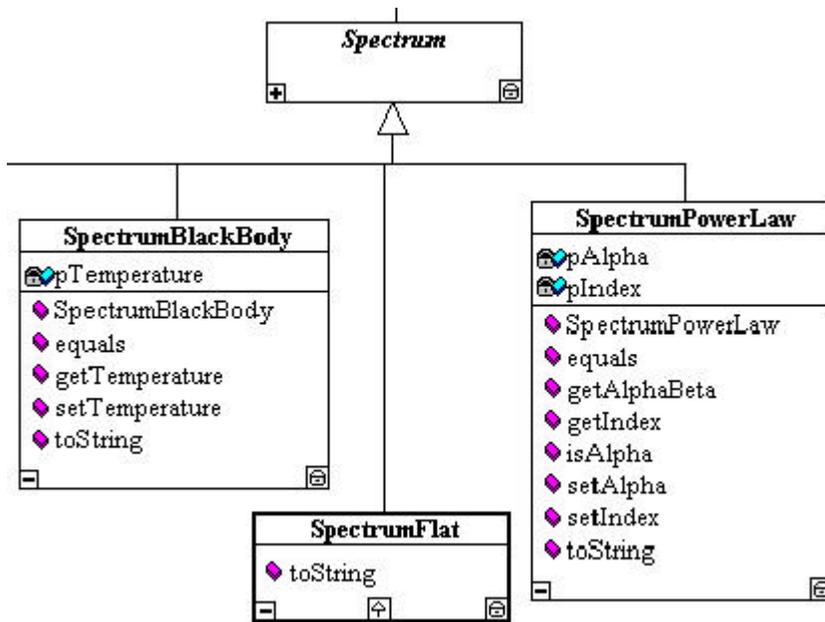
Morphology contains information about the shape and size of an AstroObject. For example, a galaxy is classified into many different types of shapes, depending on whether it is spiral or elliptical, the type of spiral, etc. Morphology includes methods for retrieving and setting the shape and as a morphology classification string. It also provides support for retrieving the size of the object.

5.4.1.9 Spectrum



Spectrum contains information and knowledge about the spectral emission characteristics of an astronomical object. The covering Spectrum abstract class details the methods that all subclasses must support.

Currently, the Spectrum class is only detailed as necessary to support SynPhot based calculations.



5.4.1.9.1 SpectrumFlat

This subclass has no additional parameters. It returns a constant flux regardless of wavelength.

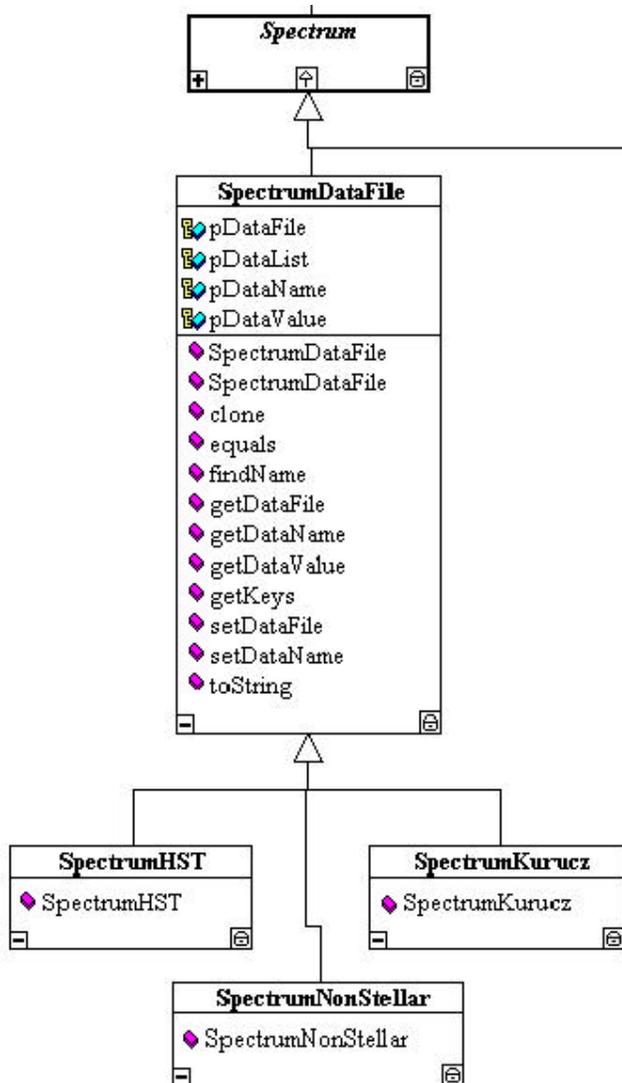
5.4.1.9.2 SpectrumPowerLaw

The Power-Law spectrum models a spectrum that meets the Power-Law model. It has two new properties: Alpha value and an Index.

5.4.1.9.3 SpectrumBlackBody

SpectrumBlackBody assumes a spectrum that emits as a black body of a specified temperature. It contains an additional property of Temperature.

5.4.1.9.4 SpectrumDataFile

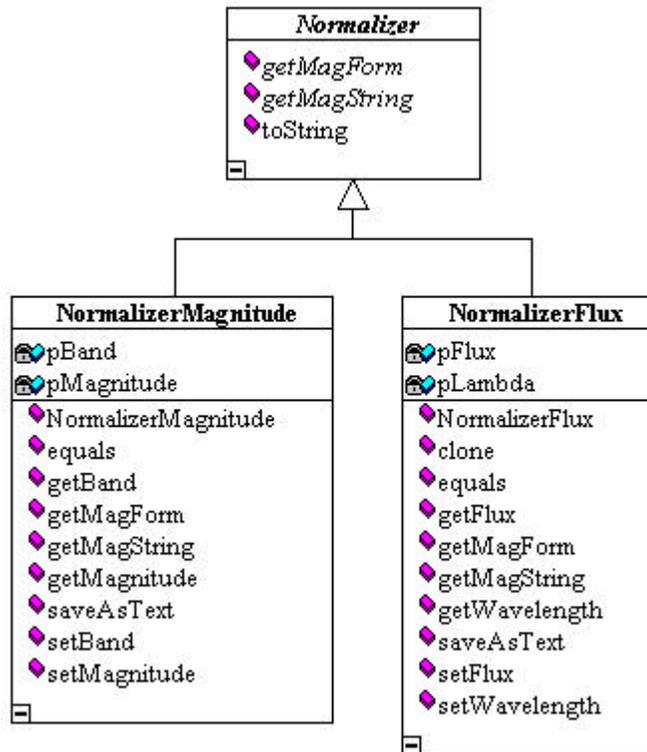


Three different subclasses of **SpectrumDataFile** allow a spectrum to be defined by a list of keyword values. This style is only supported when the SynPhot interface is used.

The **SpectrumDataFile** subclass contains a list of **Spectrum** names and matching references to "table" files accessible by a host SynPhot process.

Currently the list of table files is hard-coded into each sub-class' constructor. The intent of this class is that this reference list will be maintained in a separate data file. When this step is implemented, the three subclasses, **SpectrumHST**, **SpectrumKurucz**, and **SpectrumNonStellar** will likely become instances rather than subclasses of **SpectrumDataFile** with a reference to the data file name passed into the constructor.

5.4.1.10 Normalizer



This class is used to maintain information about adjusting a Spectrum's brightness to a specified base value. Currently, the Normalizer class is sub-classed into NormalizerFlux and NormalizerMagnitude. The difference between the two classes is not in the role, but in their units.

Each subclass contains a base wavelength and a brightness factor. A spectrum will be normalized by adjusting the intensity of its spectral model so that at the specified Normalizer wavelength, the emission has the specified intensity.

The distinction between the two subclasses is in their units. NormalizerFlux contains a specific wavelength specified in Angstroms and a base Flux. While NormalizerMagnitude contains a string reference to a standard wavelength

"band" (for example "V" for Visible) and a Magnitude factor for the spectrum in that band.

It is likely that once formulae are obtain to convert between the two systems, that the subclasses will be merged into a single class.

5.4.1.10.1 NormalizerWavelength

This class is related to the primary Normalizer class. While NormalizerFlux and NormalizerMagnitude adjust the spectrum's intensity to a base point, the NormalizerWavelength would adjust the Spectrum's wavelengths. This would be used to account for redshifts and velocity considerations.

5.4.1.11 Observatory



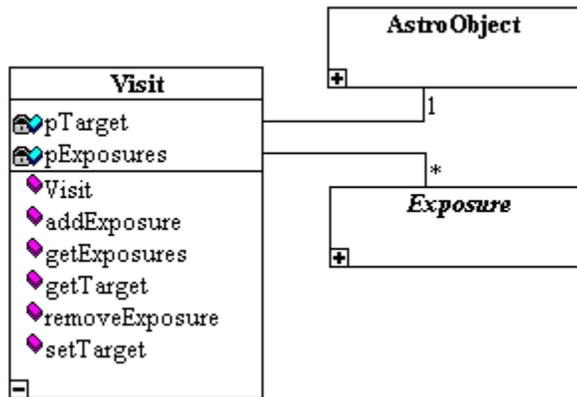
Observatory contains attributes that affect a telescope’s detection characteristics but are not properties of the instrument or the astronomical object being observed.

The contents of this class are likely to be dependent upon the overall telescope and observatory, thus subclasses of Observatory will be necessary.

Currently, two attributes are maintained in this class. These are currently tailored to provide parameters to SynPhot-based calculations.

These two parameters are EarthShine and ZodiacalLight. EarthShine predicts the amount of Earth-based emissions that will interact with an exposure. ZodiacalLight predicts the amount of background emissions deflected from the sun.

5.4.1.12 Visit



A Visit is a group of exposures. It contains the primary target and a set of exposures. Observatory parameters are also contained within Visit. Once the Visit Planner is incorporated into the SEA design, information about how the exposures are ordered and validated will be added to the Visit class.

5.4.1.13 Exposure

<i>Exposure</i>
COUNTS
SNR
TIME
fGCCheck
fHeldNames
fMessenger
fWhatsFixed
pAstroObject
pBackgroundCounts
pBackgroundFlux
pBrightestPixelRate
pCounts
pCountsTotal
pDetectorNoiseRate
pEnvironment
pErrorMessage
pGlobalCountRate
pHasError
pHoldNotify
pInstrument
pPixels
pReadNoise
pSNR
pSkyNoiseRate
pSourceCounts
pSourceFlux
pSourceRate
pTime
pTimeBrightLimit
pTimeBrightLimitOneImage
Exposure
Exposure
Exposure
addStatusMessageListener
calcBackground

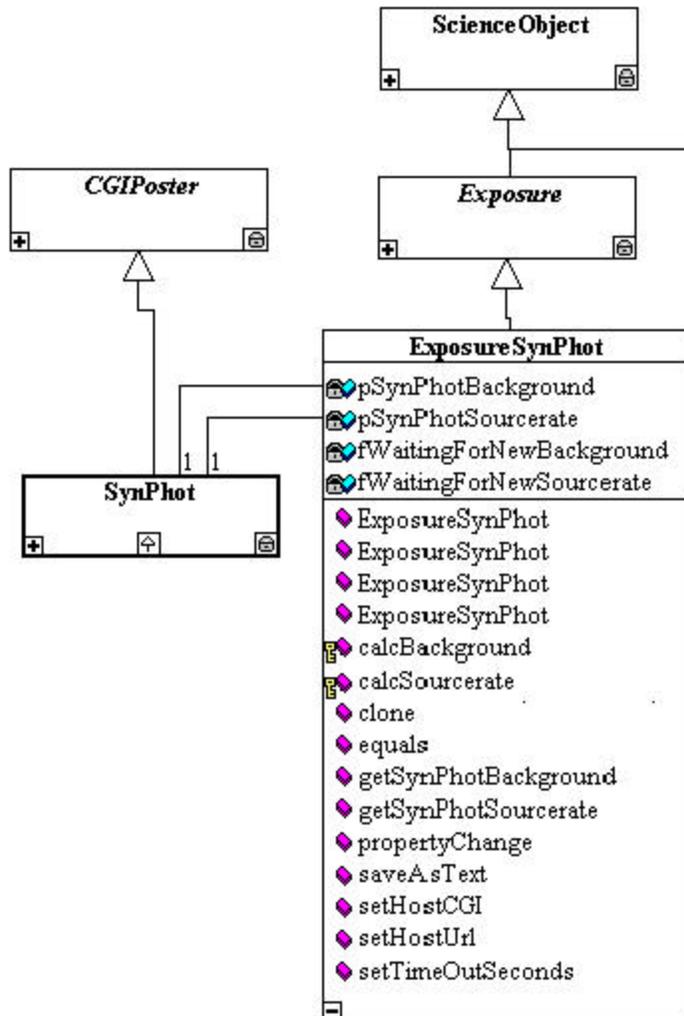
calcCounts
calcGivenCounts
calcGivenSnr
calcGivenTime
calcPixels
calcRates
calcSourcerate
clone
equals
firePropertyChange
getAstroObject
getBackgroundCounts
getBackgroundFlux
getBrightestPixelOneImageRate
getBrightestPixelRate
getCounts
getDetectorNoiseRate
getEnvironment
getErrorMessage
getGlobalCountRate
getInstrument
getPixels
getReadNoise
getSkyNoiseRate
getSnr
getSourceCounts
getSourceFlux
getSourceRate
getTime
getTimeBrightLimit
isError
isHoldNotify
propertyChange
readObject
removeStatusMessageListener
saveAsText
sendStatusMessage
setAstroObject
setCounts
setEnvironment
setHoldNotify
setInstrument
setSnr
setTime
statusMessage
updateRateStart
updateSelects

The exposure maintains information specific to making single exposure. It contains an Instrument and an AstroOffset. It is responsible for having the knowledge necessary to calculate and report on the various "counts" and "fluxes" associated with an exposure.

It is an abstract class, requiring its subclasses to implement two methods: calcBackground to calculate Background flux, and calcSourcerate to calculate the flux received from the source target.

With that information, the exposure currently calculates several properties containing rates at which photons are received from various components of exposure. Given a Signal-to-Noise Ratio (SNR) it will also calculate the exposure time (in seconds) necessary to reach that SNR. Similarly, the SNR would be achieved for an exposure of a specified amount of time.

5.4.1.14 ExposureSynPhot



ExposureSynPhot is a subclass of Exposure that adds methods for calculating Background and Source fluxes by making calls to a host machine containing the STSDAS routine, SynPhot.

It is currently the only executable implementation of an Exposure subclass.

While the default host server is 'garnet.stsci.edu', the class has the ability to change the host server, provided the specified server has both SynPhot installed and a required "CGI-script" to provide an interface to the client machine.

5.4.2 User Interface Framework

5.4.2.1 ModuleContext

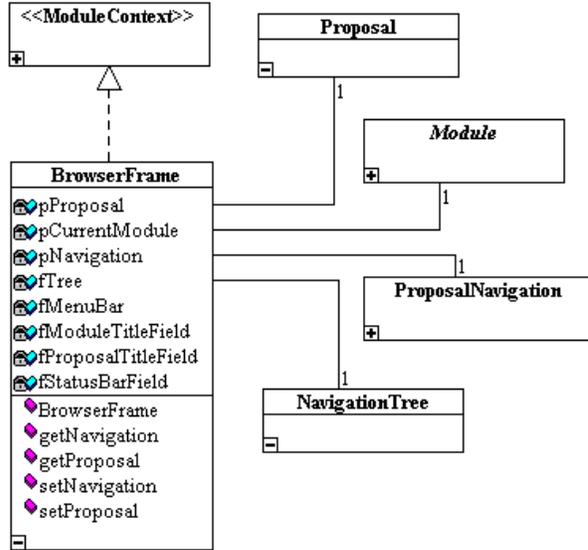


ModuleContext is an interface that must be implemented by any object

that wishes to contain a Module. It defines methods for accessing a standard set of services that are provided to Modules. These include setting the title of the module and setting message text that appears in a status bar. In addition, ModuleContext defines a method for returning a reference to a menu bar. Since a Module is a Panel, it may not have a menu bar. Instead, a Module may call this method to get a reference to its container's menu bar and may add its own

menus to that menu bar. Objects that do not have menu bars may return null from this method, in which case the Module must setup its menus elsewhere (as a popup menu, perhaps).

5.4.2.2 BrowserFrame



BrowserFrame is the main window in the SEA. It represents a single proposal when in Proposal Browser mode. BrowserFrame contains several user interface objects that allow the user to interact with the proposal:

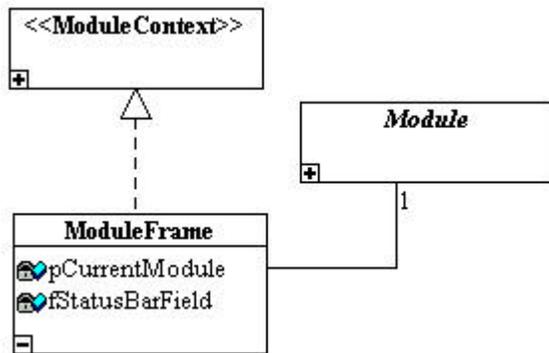
- A ProposalTree displays a view of the overall proposal. The user can select items from this tree.
- A module allows the user to view and edit the currently selected item in the proposal tree.
- A menu bar provides access to most functionality. It may contain menus added by the currently selected module.
- Navigation buttons maintain a history of the user’s selections in the proposal tree and allow the user to move

backward or forward in that history.

- A help button provides access to the context-sensitive help feature.
- Title labels display the name of the proposal and the name of the currently selected module.
- A status bar displays status messages.

BrowserFrame is responsible for ownership of the Proposal data object. If the user switches to Interview mode, however, the BrowserFrame is closed and the Proposal data object is passed to a new InterviewFrame. BrowserFrame should only receive updates from the Proposal object through PropertyChange events. BrowserFrame should never modify the Proposal. It is the responsibility of Modules to do this.

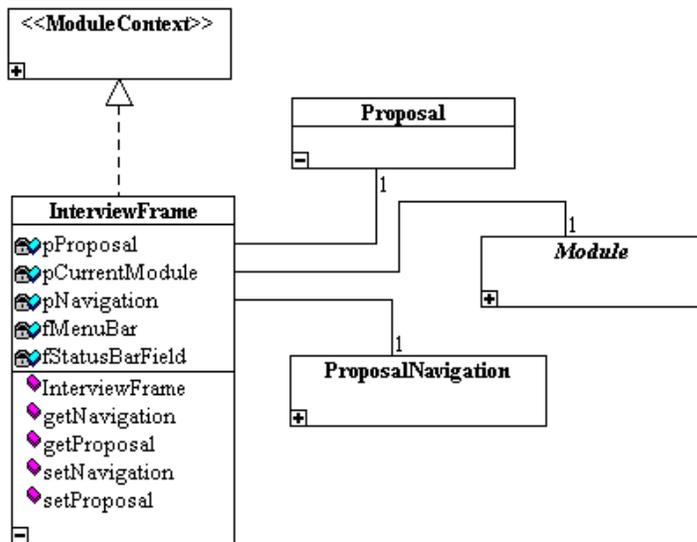
5.4.2.3 ModuleFrame



When the user chooses to open a module in its own separate window, a ModuleFrame is created and the new Module is passed to it. Like BrowserFrame, ModuleFrame is a window that implements ModuleContext. Unlike BrowserFrame, however, ModuleFrame does not contain a proposal tree, nor does it own the Proposal object. ModuleFrame is essentially a container for a Module and the objects necessary to support the ModuleContext features. It should never need

to access the Proposal or ProposalNavigation objects.

5.4.2.4 InterviewFrame



InterviewFrame is the main window used when in Interview mode. It may contain modules, and thus implements the ModuleContext interface. It communicates with the expert system, which formulates questions and adds them to the InterviewFrame. The InterviewFrame controls the navigation between sets of questions that correspond to areas of the proposal. The questions themselves, however, are managed by the expert system. The InterviewFrame contains several user interface objects including:

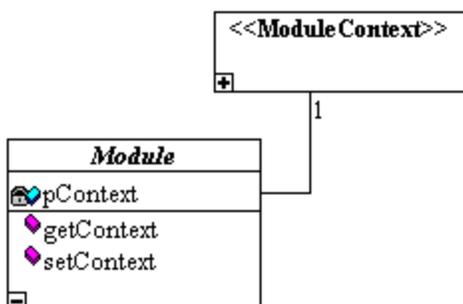
- A summary list of the steps required to complete the proposal and the current status of those steps.
- A panel that contains the set of questions for the current step in the proposal.
- Navigation buttons similar to those found on a Microsoft Windows Wizard interface. These allow the user to go backward or forward in the list of steps.
- A menu bar provides access to most functionality. It may contain menus added by a module if one exists in the question panel.
- A help button provides access to the context-sensitive help feature.
- A status bar displays status messages.

5.4.2.5 LaunchApplet



LaunchApplet is a simple applet that starts the SEA. It contains a single button that, when pressed, opens a BrowserFrame window. This may be done multiple times if the user wishes to open multiple proposals at once. LaunchApplet will be the initial access point for the SEA when run from a Web page. LaunchApplet keeps a reference to its BrowserFrames and dereferences them when it is dereferenced. In the future, LaunchApplet may display a list of existing proposals and allow the user to open specific proposals, delete proposals, etc.

5.4.2.6 Module

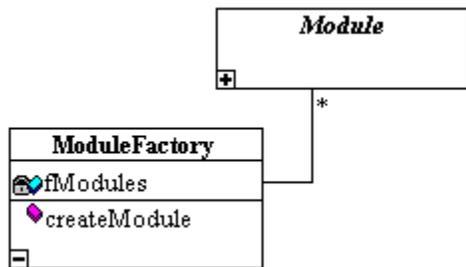


Module is the abstract root class for all proposal editors and viewers. It is a subclass of java.awt.swing.JPanel and can be placed in any container that implements the ModuleContext interface. Module contains a reference to its

ModuleContext. It does not contain any references to ScienceObjects. It is the responsibility of Module subclasses to define those references, and to subscribe to ScienceObject PropertyChange events.

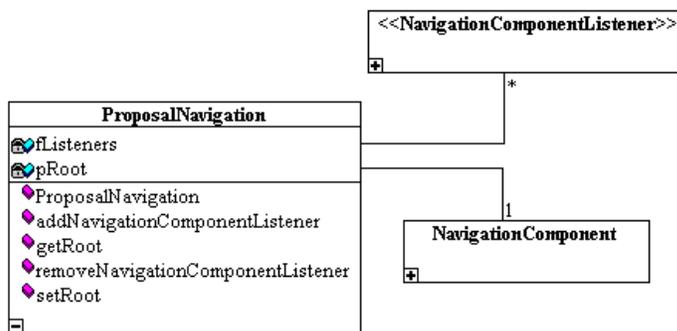
The Module subclasses will be described in a separate section.

5.4.2.7 ModuleFactory



ModuleFactory is a standard means by which containers get instances of Modules when new Modules are needed. It contains a single static method, createModule(), that takes a Class object as its argument. It returns an instance of the Module subclass represented by the Class argument. ModuleFactory retains Module instances and reuses them whenever possible instead of creating a new one each time. This is important for performance since the user will typically be switching between Modules often, and Modules are expensive to create.

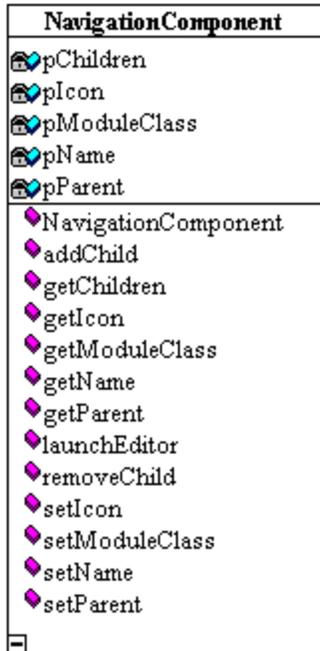
5.4.2.8 ProposalNavigation



ProposalNavigation is a container for NavigationComponent objects. It provides access to the NavigationComponent tree. It is also the source of NavigationComponentEvents. It includes methods for adding and removing listeners for these events, which can be triggered when any of the navigation components is modified, added, or removed.

Typically a single ProposalNavigation instance will be created for each Proposal instance. When created, ProposalNavigation is given a Proposal instance as an argument to its constructor. ProposalNavigation is responsible for creating its NavigationComponents in its constructor.

5.4.2.9 NavigationComponent



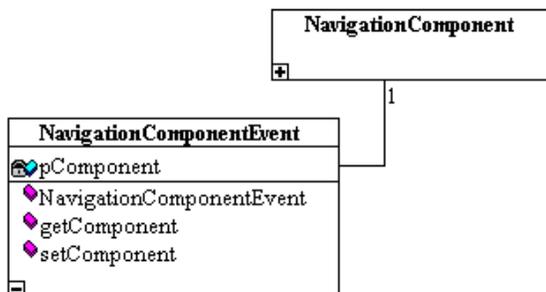
A NavigationComponent represents a specific area of the proposal. For example, a NavigationComponent could represent an Exposure, while many items within the Exposure such as a Target, Exposure Time, and Instrument, would also have a corresponding NavigationComponent. NavigationComponent is responsible for representing a proposal item to the user. It contains the following properties:

- A descriptive name of the proposal area represented.
- Icons that represent the proposal area.
- A reference to its parent NavigationComponent.
- References to its child NavigationComponents.

In addition, a NavigationComponent knows how to launch the Module subclass that is used for viewing or editing its particular proposal area. This may be as simple as knowing the Module Class type, in which case it requests a new instance from ModuleFactory and adds it to the ModuleContext. Subclasses could, however, provide other ways of launching editors by overriding the launchEditor() method.

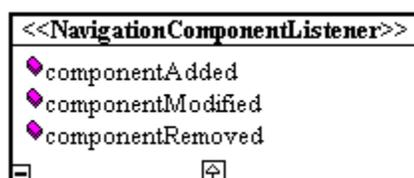
A NavigationComponent contains icons that represent the proposal area. getIcon() takes an argument which indicates the type of icon desired. This argument is the same as that used in java.beans.BeanInfo and allows icons to be specified in color or black and white, and 16x16 or 32x32 pixels in size. The default implementation of getIcon returns an icon specific to the type of Module represented. It is expected that subclasses of NavigationComponent will further define the icon image by the type of ScienceObject used in the Module (for example, with a Target, different icons could be provided for Galaxy and Star).

5.4.2.10 NavigationComponentEvent



A NavigationComponentEvent is triggered whenever a NavigationComponent is modified, added, or removed. NavigationComponentEvent contains a reference to the NavigationComponent that has changed. NavigationComponentEvents are initiated by the ProposalNavigation class.

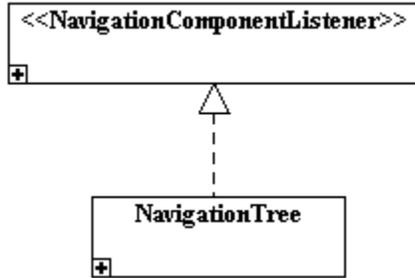
5.4.2.11 NavigationComponentListener



Any class that is interested in when the components of a ProposalNavigation change should implement the NavigationComponentListener interface. It contains separate methods that are called when a component is added, modified, or removed. These actions are triggered

when the user modifies the proposal. For example, adding a new exposure causes new NavigationComponents to be added to the ProposalNavigation object. This listener allows outside objects to be notified of these changes.

5.4.2.12 NavigationTree



A NavigationTree is a java.awt.swing.JTree user interface object that displays the contents of a proposal. Each node in the NavigationTree is a NavigationComponent. NavigationTree implements the NavigationComponentListener interface so that it is notified when it needs to redraw its contents. NavigationTree informs its parent when the user clicks on an item in the tree.

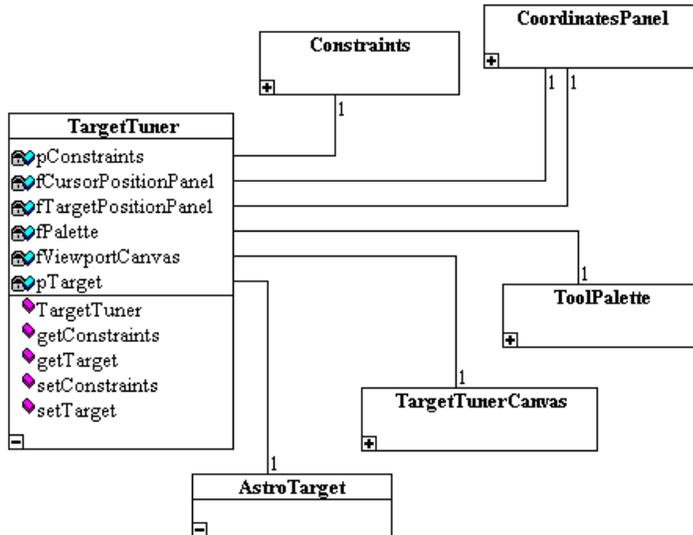
BrowserFrame contains a NavigationTree object. It responds to selections in the tree by asking the selected NavigationComponent to launch its corresponding editor, which causes a new Module to be placed in the BrowserFrame's Module area.

NavigationTree provides only a view of the ProposalNavigation and its contents. It should not modify those contents in any way.

5.4.3 Visual Target Tuner

The following class model is for the initial release of the Visual Target Tuner (VTT). It will allow the user to view a FITS image of the target area, tune the target position, and specify orientation constraints by selecting inclusion and exclusion points. This design does not support the full VTT release, which will include features such as the ability to simulate diffraction spikes and CCD bleeding. These features will be added in the next major SEA release.

5.4.3.1 TargetTuner



TargetTuner is a subclass of Module that contains the VTT user interface. It contains the following user interface components:

- The TargetTunerCanvas, which displays the visualization of the target area.
- A CoordinatesPanel which displays the target position.
- A CoordinatesPanel which displays the currently selected position.
- A ToolPalette that contains the available tools for use on the visualization area.

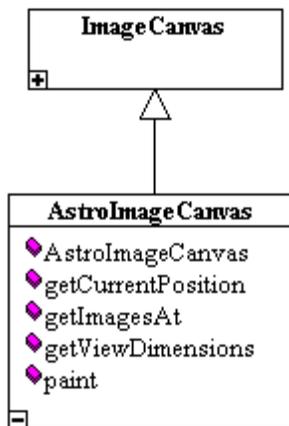
In addition to providing the user interface, TargetTuner is responsible for maintaining references and listening for changes to the AstroTarget and Constraints objects being modified. TargetTuner also communicates with its ModuleContext by adding menus to the context’s menu bar and providing its module name and status messages.

5.4.3.2 ImageCanvas



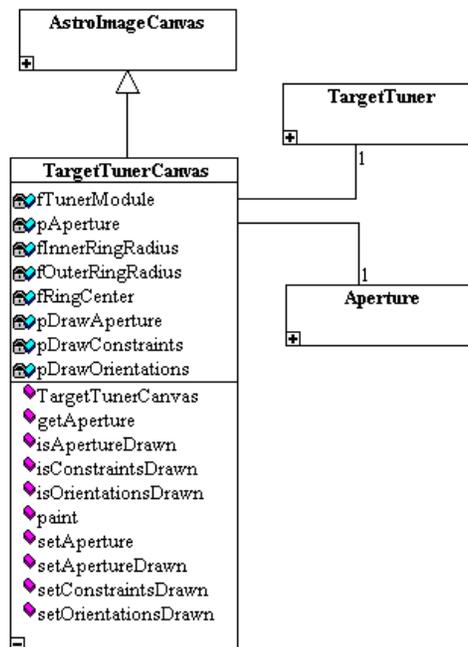
ImageCanvas is a subclass of java.awt.Canvas. It provides the ability to draw multiple images in a scrollable canvas area. It also provides the ability to move the viewport to an arbitrary position and to zoom the viewport to less than or greater than normal magnification. Images are assumed to be java.awt.Images and coordinates are assumed to be in pixels. ImageCanvas does not provide any astronomical support. It is intended as a utility class that does not require any other classes other than the standard Java classes.

5.4.3.3 AstroImageCanvas



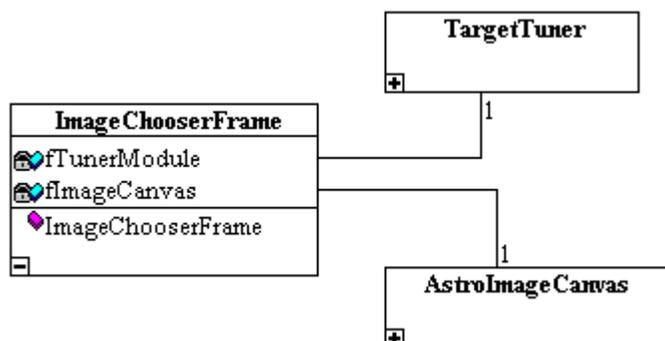
AstroImageCanvas is a subclass of ImageCanvas that adds astronomical support. It provides support for setting and getting position as an astronomical Coordinates object and automatically translates Coordinates to pixels. It does not provide support for any new drawing functionality. It is intended as a generic class that might be used for many different astronomical applications.

5.4.3.4 TargetTunerCanvas



TargetTunerCanvas is a subclass of AstroImageCanvas that adds support for drawing the other visualization components of the Target Tuner. Callers may toggle drawing of the instrument aperture, orientation rings, and inclusion/exclusion constraints. It is expected that TargetTunerCanvas will also supply support for the selection of objects within the canvas and for interactive dragging of the target selection.

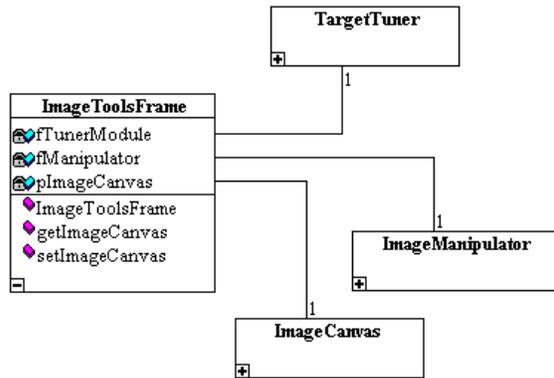
5.4.3.5 ImageChooserFrame



ImageChooserFrame allows the user to select one or more FITS images for inclusion in the visualization. It provides the ability to search astronomical databases for images with positions relative to the current

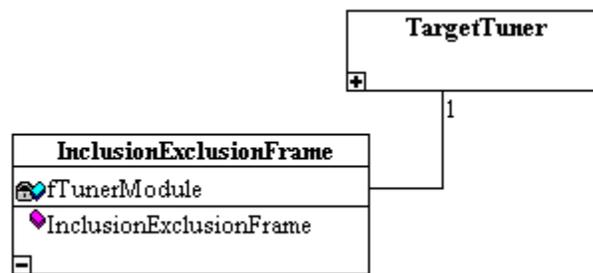
target. ImageChooserFrame is a separate window that is accessible from the TargetTuner menus.

5.4.3.6 ImageToolsFrame



ImageToolsFrame allows the user to interactively manipulate the FITS images contained within the visualization. It provides a user interface to all the features of the ImageManipulator class. The modified images may be saved for later recall.

5.4.3.7 InclusionExclusionFrame



InclusionExclusionFrame is a separate window that displays the current set of inclusion and exclusion points as a table. The user may remove items from the table. The user may continue to work with the visualization while this window remains open, in which case the table is automatically updated whenever the user changes the inclusion or exclusion points in

the visualization. It is also expected that this window will contain summary information about the effects of the current constraints on the proposal, i.e. the percent of orientations that are valid with the current constraints, and some representation of the schedulability of the proposal given these constraints.

5.4.3.8 ToolPalette



ToolPalette is a simple user interface item that contains a set of toolbar buttons. These buttons represent the functions that the user may perform on the visualization. These functions include specifying inclusion and exclusion points or regions, panning and zooming the visualization, and retrieving information about objects within the visualization. By selecting a function in the ToolPalette, the initiates that operation. The ToolPalette only provides initial access to these functions and does not actually perform them.

5.4.3.9 ImageManipulator

ImageManipulator
fSourceImages
ImageManipulator
setBrightness
setColorTable
setContrast
setNegativeImage

ImageManipulator is a utility class that takes a set of images as an argument and allows various operations to be performed on the images. These operations include setting an image to its negative, adjusting the image brightness and contrast, and modifying the image's color table.

5.4.3.10 FitsImage

FitsImage
fFitsData
FitsImage
getGraphics
getHeight
getWidth

FitsImage is a subclass of java.awt.Image that implements the features of Image as a FITS image file. It can be used wherever a java.awt.Image is required, including in the ImageCanvas class. FitsImage uses existing code from Thomas McGlynn of the NASA Goddard SkyView project. This existing code provides support for reading and writing of FITS images. The FitsImage class translates that data into an Image object.

5.4.3.11 CoordinatesPanel

CoordinatesPanel
fDecEntry
fRaEntry
pEquinox
pSystem
CoordinatesPanel
getCoordinates
getEquinox
getSystem
setEquinox
setSystem

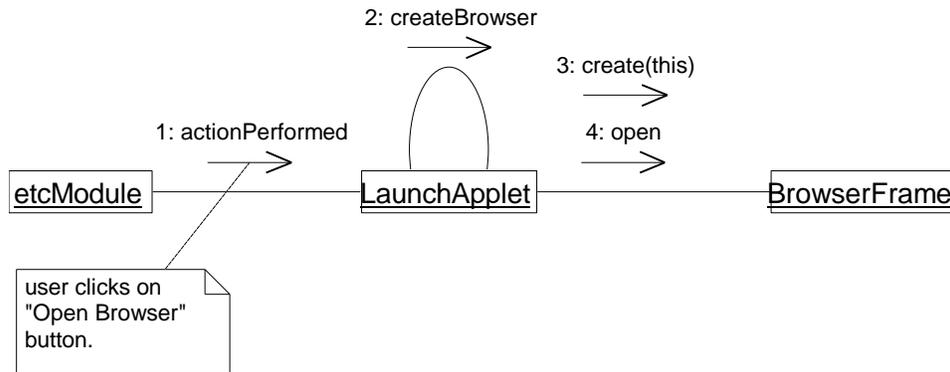
CoordinatesPanel is a java.awt.swing.JPanel that includes text entry fields for the RA and Declination of an astronomical position. It provides standard support for coordinate input, including validation, setting the equinox, and setting the coordinate system. It is expected that this class will live in a utility package rather than the TargetTuner package since it is likely to be used in other areas of the system.

5.5 Class Interaction Diagrams

This section contains a series of collaboration diagrams that illustrate the following sequence of events:

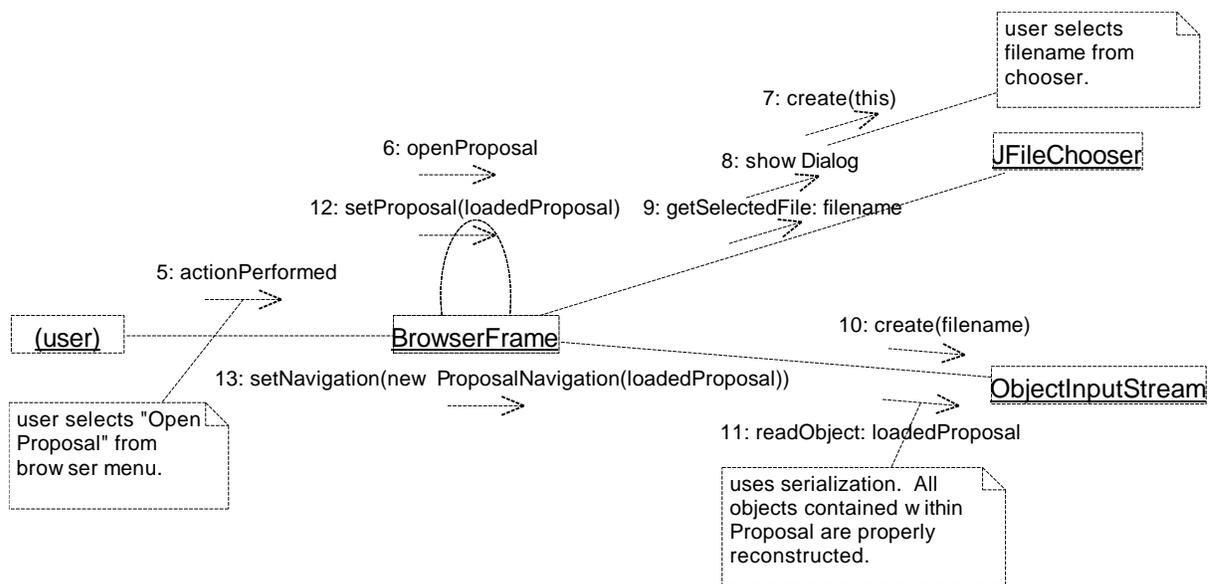
- User starts a SEA browser window.
- User opens an existing proposal from a file.
- User selects an Exposure Time in the proposal tree, causing the Exposure Time Calculator to open.
- Using the Exposure Time Calculator, the user sets a new value for the exposure time.
- The user saves the modified proposal to disk.

5.5.1 Diagram 1: Starting a SEA browser window



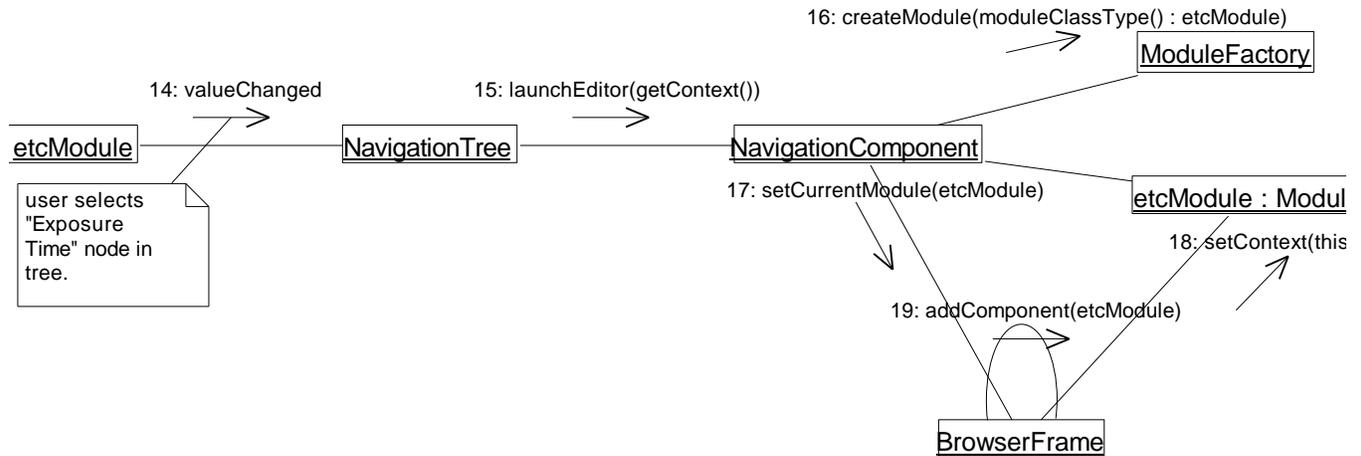
This diagram demonstrates how a browser window is first created. The user selects a button on the LaunchApplet, which causes the LaunchApplet to create an instance of BrowserFrame and open it. At that point, the BrowserFrame opens with a new, untitled proposal.

5.5.2 Diagram 2: Opening an existing proposal



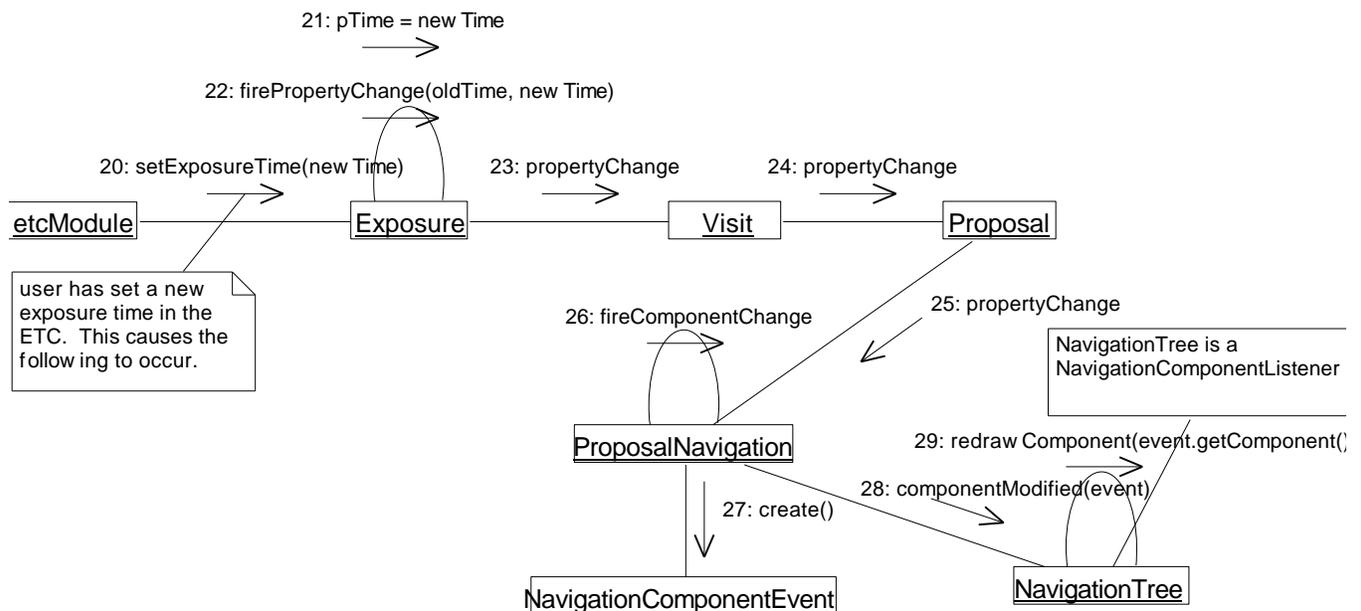
This diagram shows how a Proposal object is loaded from disk. Once the user selects the “Open Proposal” option from the browser menu, a file chooser is created which prompts the user for the filename. Once the filename has been chosen, an object stream is created, from which the proposal object is extracted. The Java seralization mechanism is used to read and write proposal objects to disk. All of the objects contained within the Proposal object are automatically reconstructed. Finally, the browser sets its current proposal to the newly loaded Proposal and creates a new ProposalNavigation instance from the new Proposal. Setting the current proposal causes user interface items such as the NavigationTree to be refreshed with the contents of the new Proposal.

5.5.3 Diagram 3: Selecting an exposure time in the proposal tree



This diagram shows how the Exposure Time Calculator module becomes activated when the user selects an exposure time in the proposal tree. These events are true of all modules and all items in the proposal tree. The NavigationTree contains NavigationComponents, one of which represents an exposure time. When the user selects an item in the tree, they are selecting a NavigationComponent. The selection causes a new module to be created and added to the BrowserFrame. The Module is added to the BrowserFrame via the ModuleContext interface, which BrowserFrame implements.

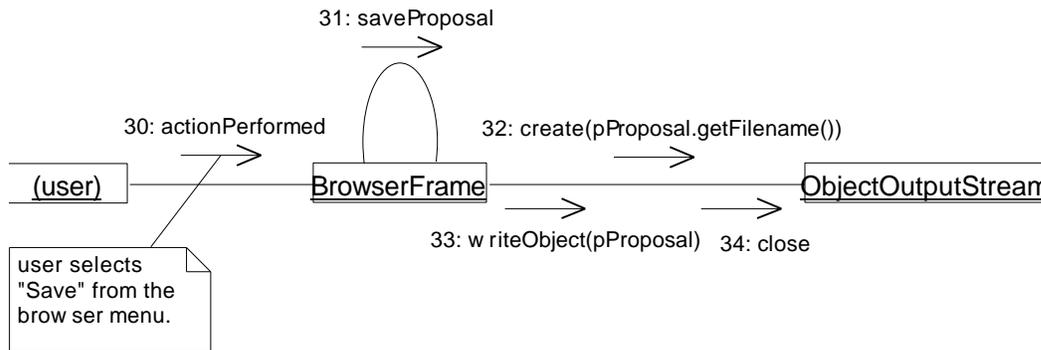
5.5.4 Diagram 4: Setting a new value for the exposure time



Once the user modifies any value in the proposal, that change is automatically propagated to the rest of the proposal and to objects outside the proposal. This diagram illustrates that concept

using the exposure time as an example. Setting the exposure time in the ETC causes the new value to be passed to the Exposure object, which modifies the value and fires a `PropertyChangeEvent`. That event is propagated to all appropriate listeners, which includes the parent object of Exposure: `Visit`. `Visit` passes the event to its parent, `Proposal`, which passes it to its listeners. `ProposalNavigation` always listens to all `Proposal` changes so that it can notify user interface components. It fires a `NavigationComponentEvent`, indicating that the contents of the Exposure Time component have changed. Since the `NavigationTree` is listening for those events, it receives the event and redraws its contents.

5.5.5 Diagram 5: Saving the proposal to disk



Saving a proposal to disk is a simple operation. Since serialization is used to store the `Proposal` and its subobjects, we simply open an output stream with the specified filename, write the `Proposal` object to the stream, then close the stream.

6 Java Implementation Notes

6.1 *Development Tools*

6.1.1 Integrated Development Tool: Visual Café

The SEA team has chosen Symantec's Visual Café for Java 2.0 as the integrated development environment. Other IDEs were considered such as Borland's JBuilder and Cosmo Software's Cosmo Code. Visual Café is currently the only second-generation Java IDE. This gave Visual Café an advantage in robustness and completeness of features, while remaining intuitive and easy to use.

6.1.2 Configuration Management: Visual SourceSafe

Microsoft's Visual SourceSafe was chosen as our configuration management tool. It provides an easy-to-use graphical interface, works transparently over the Microsoft network protocol, and includes all the features that we expect to need from a configuration management tool.

6.1.3 Expert System Engine: Advisor/J

Neuron Data's Advisor/J was chosen as our expert system engine. Advisor/J fits well into our development environment because it is entirely written in Java. This will make it much easier to incorporate expert system technology into the SEA. In addition, Advisor/J includes an integrated development environment for building rules and managing the rule base.

6.2 *Additions/Exemptions to the Java Style Guide*

The (formerly) Code 522 Java Style Guide will be used for development of the SEA. While most of the guide meets our needs, the following changes will be adopted:

6.2.1 Property names

If a field represents a JavaBean property, begin the field name with the letter "p", then conform to the variable format. All other fields will continue to use the existing name formats.

6.3 *Security Management*

6.3.1 User security

The initial release of the SEA will not have user authentication or any other mode of user security. This is partly due to the fact that users want to be able to save their proposal as a file that can be transmitted to colleagues. If proposals were stored in a secure database, this would not be possible. However, we are aware of the sensitivity issues involved with proposals and are considering instituting some sort of authentication. Digitally signing the proposal file is one option. Another would be to encrypt the file. We will revisit this issue after the prototype release.

6.3.2 Applet security

Many requested features, such as the ability to save the proposal to a local disk, are not possible in an applet due to the Web browser's security model. We must find a way to overcome this limitation. The most obvious possibility is to digitally sign the SEA distribution file. This will enable the SEA applet to access secure features in the Web browser, such as saving to a local disk. We are investigating incorporating digital signing into the prototype release.

6.4 Deployment notes/comments

The SEA will be deployed both as an applet and as an application. In the case of an applet, a web page will be setup where the user can easily access the tool. Application deployment, however, is a little more problematic. Certainly we will deploy the tool as a JAR file, but other questions remain: Should we deploy archives with the JRE embedded, or should we assume that they already have a proper virtual machine? Should we use an automatic download mechanism such as Netscape's SmartUpdate, or should we just allow the user to download the JAR file and run it? The initial release of the SEA application will most likely be deployed as a JAR file that can be downloaded, along with instructions on how to run it.

6.5 Location of implementation files

We plan to maintain up-to-date JavaDoc files for our implementation, as well as source files and demonstrations. These files may be obtained at <http://aaadev.gsfc.nasa.gov/NGSTProtos/>.

