Design of Ptolemy Development Eclipse Plugin

Iteration 1: Version 1.3

3/28/2012

Team 13
Kimberly Eue, Adam Rotondi, Justin Landsgard, Kyle Bradwell, Tim Flanigan

Contents

1: Introduction

This document is the design document for the creation of the Ptolemy eclipse plugin. The plugin is intended to help developers use the Ptolemy language in a familiar and easily distributable environment. The purpose of this document is to describe in technical levels of detail the design and possibly implementation of the project or its parts. Further sections of this document relate to individual aspects of the design and development of this project in detail.

1.1: Table of revisions

Revision #	Date	Summary of changes
1.0	03/01/2012	Initial document creation
1.1	03/14/2012	Addition of Project breakdown and package structure.
1.2	03/23/2012	Addition of Use cases and diagrams for project breakdown.
1.2.1	03/25/2012	Added Mapping table and distinguished use cases.
1.2.3	04/03/2012	Reformatted section 4 to be clearer. Clarified module diagram in section 1.4
1.3	04/08/2012	Corrected Ptolemy capitalizations and removed language referring to iteration 2 before it was needed.
1.4	10/06/2012	Added Language and Diagrams for Iteration 2, as well as corrected Iteration 1 sections that differed from implementation requirements.
1.4.1	11/20/2012	Corrected Language in some cases for Iteration 1 requirements and reformatted Iteration 2 structure to be more included in existing sections of document.

1.2: Table of Terminology.

This table includes those terms used throughout the Document that are commonplace and yet are not immediately clear.

Term	Description
Abstract Syntax Tree	The breaking down of the syntax of a file into a structured tree of **Tokens**, so that syntax can be appropriately interpreted by not only the compiler, but the front end of the program. Uses Include dynamic error checking and syntax highlighting.
API	An application programming interface, which is a Source code based specification intended to be used as an interface by software components to communicate with each other.
Eclipse	An IDE for programming in a variety of languages, which relies on the open source community to develop and distribute its different functions as plugins.
Eclipse Package	A collection of files, classes, interfaces, etc. that are related in a functional or organizationally important manner within an eclipse project. Higher levels of abstraction are considered better for these collections.
Eclipse plugin	A feature or set of features for eclipse that perform tasks or otherwise allow the eclipse program to perform such tasks.
Eclipse project	A structured and hierarchical organization of files to be used by eclipse.
Extension point	In eclipse plugin development, an extension point is a feature that is extendable from another eclipse plugin. It is common practice to use extension points of other plugins, and then broadcast the new plugin's extension point after modification.
IDE	Integrated development environment. A program that boasts features beyond simple text editing for programmers, often including compilers and other advanced features.
JVM	Java Virtual Machine, the underlying framework that allows all java programs to run on a variety of clients, regardless of operating system or hardware.
Library	A collection of resources used to develop software. This often includes pre

	written code, subroutines, classes, etc. Most provide an API for the developers using them.
Post condition	Those conditions that are assumed to be true after the use case in which it suffixes.
Precondition	Those conditions assumed to be true before the use case in which this prefixes.
SWT	Standard Widget Toolkit. The library of objects and methods used to create new eclipse views, perspectives, and other related features.
Workspace	A file system created by the user and used by eclipse to manage projects and other files for use in the eclipse program.
Wizard	A programmatic way to collect common information from users and to achieve an expected result based on the input information.

1.3: Coverage of this document

This document is meant to serve as a functional requirement and development guide. That is, the terminology and standards observed here should remain throughout the development of the deliverables of this project. Deviating from this document and the stated intentions within would be considered an error, bug, etc.

Additionally, this document is only currently intended to describe those facets of the project that fall under Iterations 1 and 2 in the Project plan. An integrated Design Document will be made at a later date, and include all of the information present in this document as well as information from Iteration 3. Until that document is created, all information within this document is to be considered “sacred” for the development of the project.

1.4: Package Structure for the project in iteration 1.

The structure for this project includes several Eclipse Packages, Which can serve as functional workloads for team members, or be divided between team members. Those packages are described in detail in further sections.

This section and the subsections cover their structure in relation to each other.

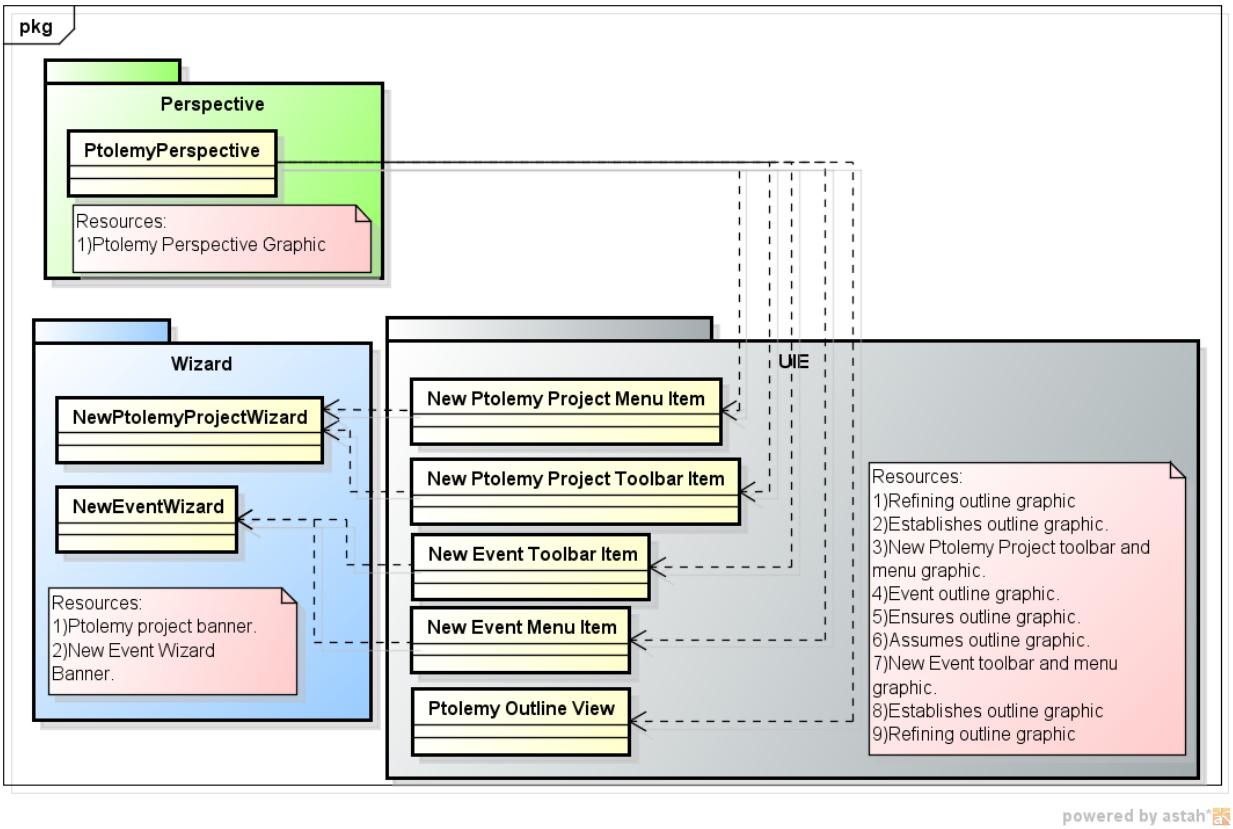

Figure 1: Package Overview

1.4.1: The Perspective package.

This package should be considered the "top level" package, and will rely heavily on the other packages and will be the "container" for these other objects. Graphics and other objects that relate directly to the perspective and its creation, but not individual components (see **1.4.3: the UIE package**) should be located within this package.

1.4.1: The Wizard package.

This package is composed of those units known as "wizards" in eclipse, and follows the schemes and styles associated with wizards already in eclipse.

This package will rely on information gathered from **1.4.4: The AST package** to complete its functions. Any objects that are required for generation of the wizards should be included in this package for consistency.

1.4.3: The UIE package.

This package is the location of the individual elements that are to be added to eclipse's Graphical User Interface to allow Ptolemy event and project creation, such as the menu items and toolbar buttons. This package will rely on **1.3.1: The Wizard Package** for actual creation of new Ptolemy projects and items. This package may call upon features in **1.3.1** in several ways to emphasize usability.

1.5: Project structure for Iteration 2.

In addition to the structure defined in section **1.4: package structure for the project in iteration 1**, additional changes to the eclipse environment must be made to achieve the technical details of iteration 2. Upon investigation, these changes cannot be made using only the eclipse API's for a plugin, and require more substantial changes to existing code found in Eclipse's internals. The high level interpretation of these changes will be noted in this document, while the technical details of the changes will be recorded in later sections of this document.

1.5.1: Recognizing Ptolemy language features.

Currently, the Eclipse platform only allows creation of editors for new languages, without inclusion of features for the Java language. Therefore, to maintain Java language features add Ptolemy language features described in **2.2.: Product Use Requirements,** as well as **2.1.2: User interface requirements,** the internals of Eclipse must be modified. All programming architecture and styles will be maintained as much as possible, as the changes required are neither extremely substantive nor extremely complex.

1.5.2: Syntax Highlighting for Ptolemy language features.

Currently, the Eclipse platform only allows addition of highlighting rules for new languages, without inclusion of that feature for the existing java language. Therefore, in order to maintain Java syntax highlighting and add those requirements for highlighting Ptolemy language features, the internals of the Eclipse platform must be modified. All programming architecture and styles will be maintained and adhered to as much as possible, as the changes required are not extremely substantive.

2: Overall Requirements Description

This section covers those aspects of the project that deal directly with requirements specified by the client, the academic advisors, and the senior design class at ISU. This section also covers related assumptions, risks, and proposals for solutions.

2.1: Product Perspective

This section and its subsections deal with describing the product being produced by team 13 in several ways.

2.1.1: Product description and concept

The product will be an Eclipse plugin. The reasoning for this is that Eclipse is currently implemented as a core product with all features and functionality built as plugins. Therefore, making an additional plugin for use with the Ptolemy language is within the industry standard for developing with and for eclipse. Additionally, Eclipse is widely used by many developers and programmers for advanced industry applications, so making a plugin for use of the Ptolemy language would greatly increase the awareness and usability of this language in industry.

The plugin will be itself a developer tool, and be able to perform several different and varied functions in relation to being a tool for programmers using the Ptolemy language.

2.1.2: User interface Requirements

This section deals with those requirements that involve the front facing user interface of the product.

2.1.2.1: Perspective Requirements

The eclipse plugin will offer a perspective to cleanly separate it from other functionality provided by eclipse, and to contain all of the features of the project and those created by this team.

2.1.2.2: Text Editor Requirements

The product will provide a modified java code editor capable of displaying and modifying the file contents of a Ptolemy compatible file. This editor will be able to display the contents in a way to show that it is correct use of Ptolemy syntax (or display that it is in fact not, and what errors are causing this). This editor will be able to perform functions already provided by eclipse, but modified for use with Ptolemy.

2.1.2.3: Menu and toolbar Requirements

The Ptolemy language includes several new syntax items, some of which can be specified on the same level as classes and interfaces, and use a similar hierarchical structure. These are known as Events. To programmatically create these events in a standard manner, several menu and toolbar items will be created that allow this creation to be accessible to the user.
These items will not actually perform the creation, but instead direct the user towards the programmatic creation.

2.1.2.3: Outline Requirements

Eclipse provides a graphical view of a file as an outline, which cleanly summarizes the important contents of a file into a hierarchical list. The product will include a modified Outline View that allows for Ptolemy objects to be included in the hierarchy.
The modified Outline View will not change the hierarchy standing of java objects that are used in the Ptolemy language, but merely add those objects that are not currently supported, as well as the graphical objects used to correctly display them.

2.1.3: Hardware Requirements

The only hardware requirements are any computer or other similar device capable of running the JVM on which eclipse can run.

2.1.4: Software requirements

The product will conform to standards introduced by the eclipse plugin development community. This includes the proper implementation of all extension points required and the broadcasting of those extension points with our modifications to future plugin developers.

2.2: Product Use Requirements

This section describes the uses that the client, advisors, or other persons of authority on the project have dictated the product should allow.
The product will:

A. Allow for the creation of eclipse projects in the Ptolemy language.

B. Allow for the compiling and running of Ptolemy Programs.
C. Allow for the creation of files using Ptolemy syntax.
D. Allow viewing and editing of Ptolemy Syntax in a manner consistent with viewing and editing java syntax, but with the addition of Ptolemy objects, as described in **2.1.2: User interface requirements.**
E. Allow for the tracking of Ptolemy syntax for use in syntax highlighting
F. Allow for the tracking of Ptolemy syntax for use in refactoring
G. Allow for the tracking of Ptolemy syntax for use in code completion

Please see **4: Project Decomposition,** for the specific mapping of these requirements.

2.3: Dependencies

This System has several dependencies, of which the most relevant are:

- The Eclipse IDE on which the system will run.
- The JVM on which the system and the eclipse IDE will run.
- The Industry Strength compiler provided by our client and advisor.
- The SWT which our team uses to create the system.
- The Eclipse Plugin Development Libraries used by our team.
- The Extension points provided by the Eclipse Plugin Development community which our team uses to build the system.

3: Design Goals

Several aspects go into the goals of the product, and include:

- Correctness
- Usability
- Robustness
- Efficiency
- Maintainability
- Extensibility

Each of these goals and our team's intended ways of meeting these goals are explained below.

3.1: Correctness

The product shall meet the functional requirements in this document, and the non functional requirements both in this document and the project plan.
Any requirements given by the client are to be represented in this document or referenced directly from the project plan.
Verification for these requirements shall be done through code review, static code analysis, and testing.
Our team shall use the practice of "Test-Driven Design" to ensure correctness in our specifications.

3.2: Usability

The product shall be easy to use by providing a simple, standard interface. This interface shall not deviate from the standard Java development interface in eclipse, save the addition of new UI Elements described in **1.4: Package Structure.**
Those items that are added shall be done in an intuitive and non-intrusive manner.
Usability testing using government standards outlined at Usability.gov shall be used to verify the usability of the system by those in the field.

3.3: Robustness

The system shall be tolerant of misuse (e.g. non-valid code, bad file imports, and invalid projects) without catastrophic failure. This shall be achieved using standard techniques of data encapsulation, exception handling, using simple interfaces, formal method parameters, and invariants.
Robustness shall also be verified in design with exploration of exception scenarios for the use cases of the system. These exception scenarios will be tested to ensure robustness of the system.

3.4: Efficiency

The system shall avoid misuse of system resources such as processor and memory capacity to avoid performance issues. This is to be done via efficient use of objects and data structures.

3.5: Maintainability

The system shall achieve maintainability by using standard architectural patterns, reuse of source code, and reuse of object code.
The system will hold to standard naming conventions of eclipse plugin development and SWT, so that it will be easily maintainable by future teams of developers.
The system will be modular by reducing dependencies between classes' as much as necessary and by tying abstract classes and modules directly to their functions.
This shall be verified by code reviews, as well as creation of an API for future developer use.

3.6: Extensibility

The system shall achieve extensibility by use of abstract classes and/or interfaces, so that functionality may be extended in different directions from the same base.
This shall be verified by the creation of API's to be extended and used by future teams and for system development.

4: Project Decomposition

The system shall be decomposed into packages previously described in **1.4: Package Structure**, but with each package having its purpose specifically stated with ways of achieving said purpose. Each section includes use cases and diagrams. The table below shows correlation between use cases and the features described earlier in section **2.2: Product Use Requirements**.

Requirement / Use Case	A: Create Ptolemy Projects	B: Compiling and running Ptolemy programs	C: Creating Ptolemy Files	D:Open Ptolemy Files	E: Open Ptolemy Projects	F: Recognition of Ptolemy Language Features	G: Syntax highlighting of Ptolemy Language Features	E: Content Assist for Ptolemy
1: Creating the Ptolemy Perspective	X	X	X	X	X			
2: New Event Creation			X				X	
3: Ptolemy project creation	X							
4: User Selects menu item to create new project	X							
5: User Selects toolbar item to create new project	X							
6: User Selects menu item to create new Event			X				X	
7: User Selects toolbar item to create a new Event			X				X	
8: User runs the Ptolemy program from menu or toolbar		X						
9:ntroducing Ptolemy language features to the text editor						X	X	
10: User invokes content assist on a Ptolemy language feature								X

4.1: Use case structure

This section contains several Use cases, and is of the format:

- Title:
- Primary Actor
- Secondary Actor(s)*
- Level
- Stakeholders and Interests*
- Precondition*
- Success Guarantees
- Trigger
- Main Success Scenario and steps
- Alternate Success Scenario(s)*
- Exceptions and handling scenarios
- Related Information.*

**These listings are optional, and may not be included in every use case.*

4.1.1: General Use Case outline

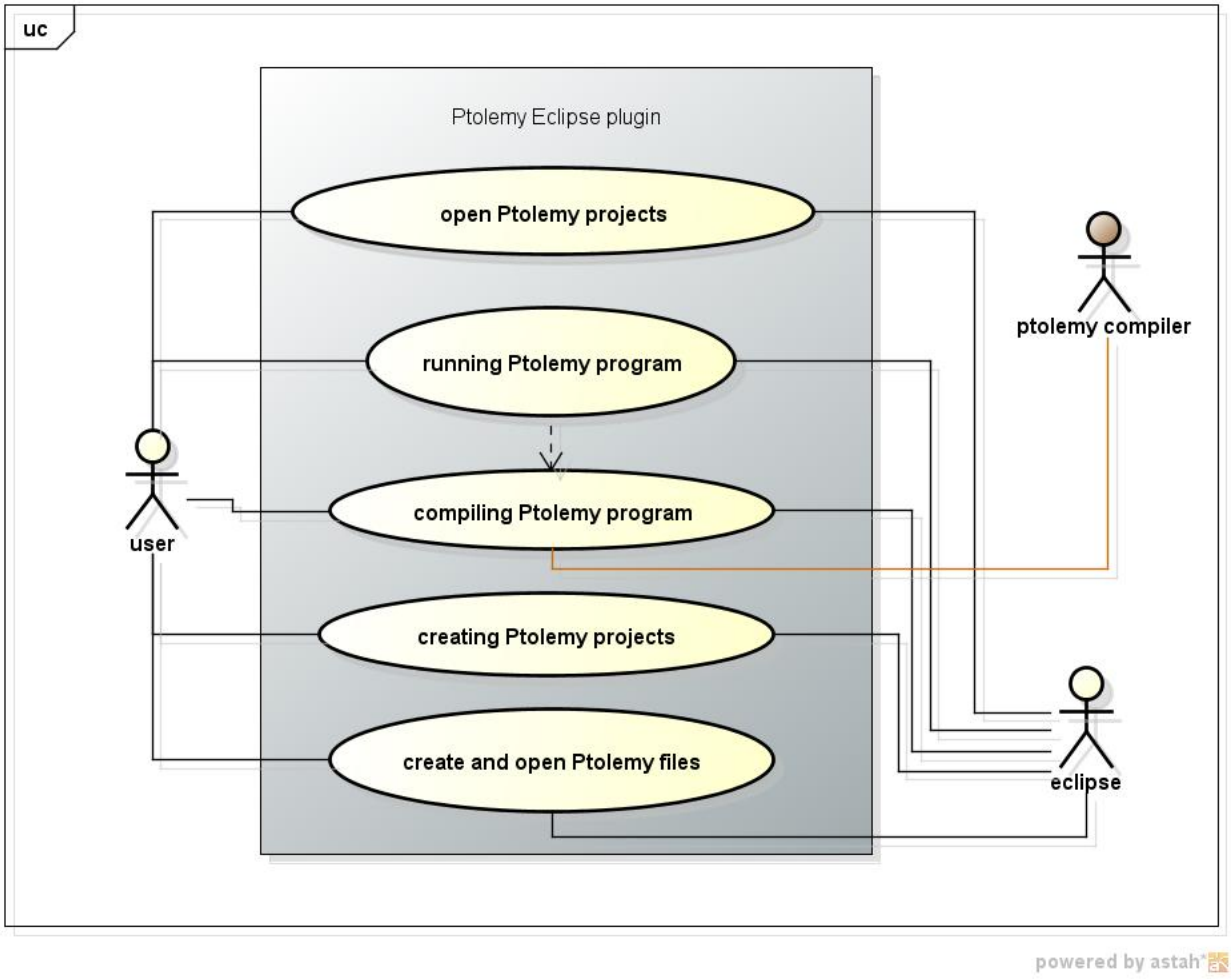

Figure 2: Iteration 1 Use Cases

4.2: The Perspective package

This package serves as a top level package, in which the actual eclipse perspective is created, the UI elements (from **4.4**) assembled into the proper place, and the graphics objects for the perspective are located. The only element as a class should be the Perspective creation class.

4.2.1: The Perspective Creation Class

This class is the class that is called for the perspective creation for the system. It uses many parts of other packages in the system.

4.2.1.1: Technical overview

This class is responsible for the initial setup and runtime action handling of the Ptolemy Perspective. That is to say that it's responsibilities are to create the different perspective, manage the creation and placing of UI Elements, and handle setup and runtime of the standard eclipse elements that are used by the perspective. This class uses the extension point for a new Perspective, as well as those for any views it is using when creating the perspective.

4.2.1.2: Use Cases and Diagrams

This section contains use cases for this class, in the format described in **4.1: Use Case Structure**.

Use Case 1: Creation of the Ptolemy Perspective.
Primary Actor: User
Secondary Actor: Eclipse
Level: User
Precondition: The user is running eclipse.
Success Guarantees: The Ptolemy Perspective is opened, with the ability to view and edit code using Ptolemy Syntax.
Trigger: The Perspective is selected via the View->Perspective menu or by the UI Icon in the top right of eclipse (by default).
Main Success Scenario:

1. The User Selects the Perspective to be opened.
2. The perspective creates all UI elements needed to be displayed.
3. The perspective creates all UI menu items needed to be displayed at some time.
4. The Perspective creates all Views that are displayed by opening this perspective. These views are similar to those displayed by the default Java Perspective.
5. The Perspective configures the layout of these views and UI elements to be similar to the look of the Java perspective.
6. The Perspective Displays those views as well as those UI elements that are visible on the toolbar.

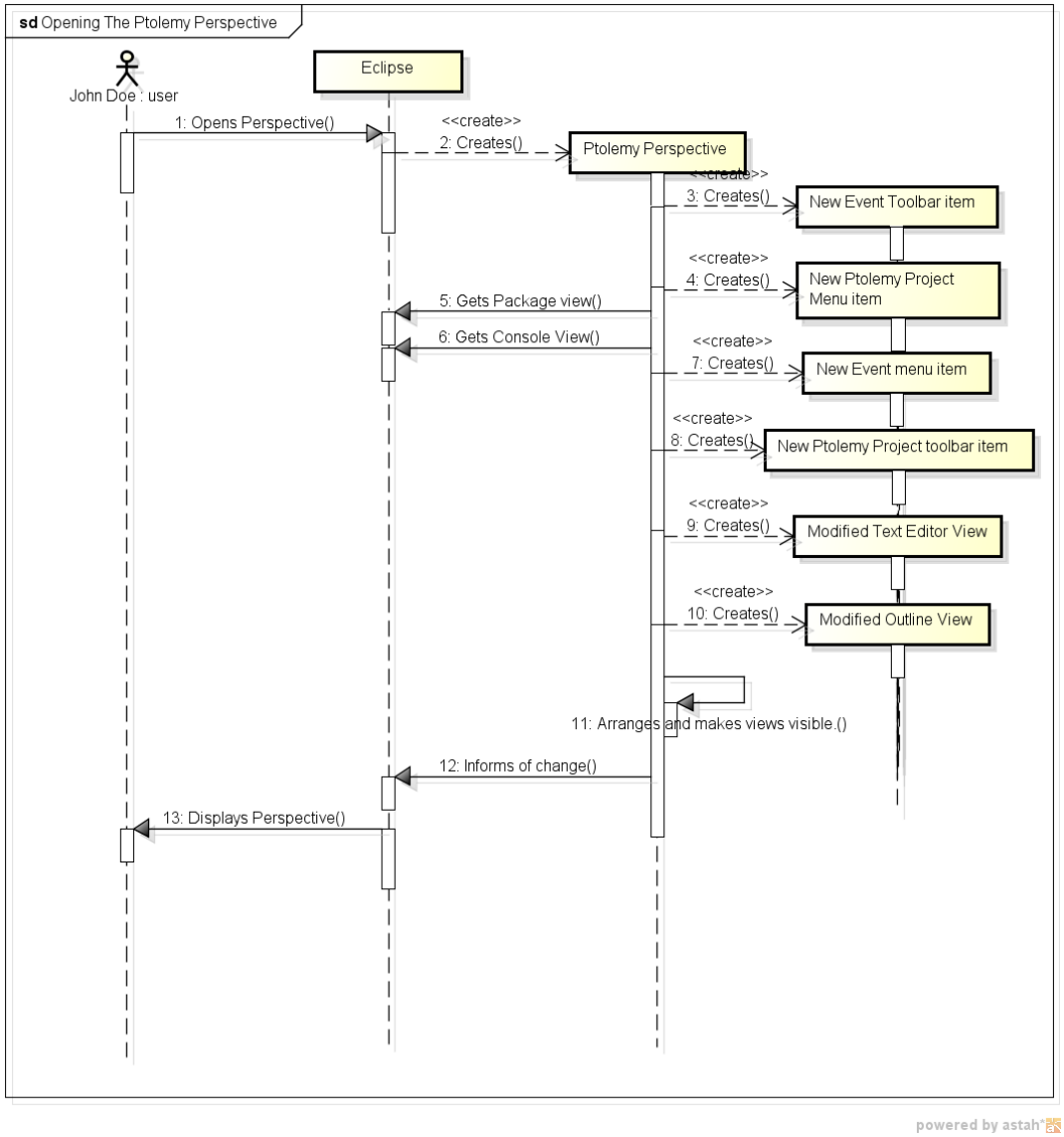

Figure 3: Creation of the Ptolemy Perspective

4.3: The wizard package

The wizard package will contain all objects, files, classes, and architectural designs relating to creation of wizards for use in eclipse.

All wizards use a combination of the SWT framework and java Swing library to achieve a programmatic, reusable method of gathering user input and creating valid Ptolemy compliant output.

Any miscellaneous objects such as graphics required to comply with the requirements in this document or the project plan that directly affect the generation of the wizards will be included in this package.

4.3.1: The New Event Wizard

This Section covers those aspects of The New Event Wizard, including an overview, use cases, and diagrams representing those use cases.

4.3.1.1: Technical Overview

The New Event Wizard will borrow heavily from the new Class wizard already provided by eclipse's SWT framework. It allows specification of a new file to be created in the eclipse project, or the importing of an outside file to the eclipse project (as long as that file has an event type within it). This aspect of the wizard will rely on the AST, to be covered in Iteration 2.

The wizard can also allow the user to specify a package for the new event to be included in (to easily maintain the hierarchy of eclipse projects).

The wizard can allow the user to input valid Events for this event to implement, much like an interface in java can implement another interface. This is only valid if the specified file is indeed containing an event and it is in the project already.

After collecting the specified information from the user, the wizard will either import the specified file from the file path (assuming again it is a valid event file) or create it with the parameters specified, inside of the Ptolemy project.

4.3.1.2: Use Cases and diagrams

The following are Use Cases for the New Event Wizard, following the format outlined in **4.1: Use case structure.**

Use Case 2: New event Creation.
Primary actor: User
Secondary actor: Eclipse environment
Level: System
Precondition: The eclipse environment is in the Ptolemy perspective, and has at least one open Ptolemy Project.
Success Guarantees: A new file with a stub of Ptolemy Syntax representing an event is created in the file structure of the eclipse program, and displayed accordingly.
Trigger: The System calls for the Wizard to be opened.
Main Success Scenario:

1. The System calls for the wizard to open.
2. The wizard displays for the user.
3. The user enters valid parameters for the creation of the new event. (Such as package name that it is under in the file system, the name of the file, etc.)
4. The wizard creates the file in the appropriate location as specified.
5. The wizard creates inside that file a collection of Ptolemy syntax based on input from the user.
6. The Wizard informs the system of the change, which in turn informs Eclipse of the change, so that they may display and track these according to their specifications.

Alternate Success Scenario 1: Creation of a new Event that extends another Event.

1. The system calls for the wizard to open.

2. The wizard displays for the user.
3. The user enters valid parameters for the creation of the new event. One of these parameters is the extension of another event in the Eclipse Environment or valid outside library.
4. The wizard creates the file in the location specified by the user.
5. Inside of the file, the system creates a collection of Ptolemy syntax based on input from the user.
6. One of these statements is the extension of another event from eclipse or an outside library.
7. Inside of the event syntax block created in the file, the wizard adds other collections of Ptolemy syntax from the extended event.
8. The wizard notifies the system of the change, which in turn notifies eclipse of the change, so that they may display and track it according to their specifications.

Alternate Success Scenario 2: Importing an event from an outside source.

1. The system calls for the wizard to be opened.
2. The wizard is displayed for the user.
3. The user specifies the location of a file containing an event.
4. The wizard creates a copy of that file inside of eclipse's workspace as specified by the user.
5. For the completion of (4), additional syntax may need to be added to comply with Eclipse's workspace requirements or requirements of the JVM. These are added unobtrusively.
6. The wizard notifies the system of the change, which in turn notifies eclipse of the change. The file is then tracked and displayed according to their specifications.

Exception Scenario 1: Invalid User Parameters.

1. The system calls for the wizard to open.
2. The wizard displays for the user.
3. The user enters at least one piece of invalid information. (Examples include an invalid name, location of the file, etc.)
4. The wizard responds by informing the user within the wizard of the error, in a nonintrusive but clear manner.
5. The wizard does not allow the user to proceed with file creation until the invalid information is corrected to an acceptable parameter.
6. Once corrected, the main success scenario or an alternate success scenario is followed from (4).

Exception Scenario 2: Extended event is invalid.

1. The system calls for the wizard to open.
2. The wizard displays for the user.
3. The user enters at least one event to be extended that is not a valid event file.
4. The wizard responds by informing the user within the wizard of the error, in a nonintrusive but clear manner.
5. The wizard does not allow for file importing until the user specifies a valid event file from outside eclipse or from a library.
6. Once corrected, the main success scenario or an alternate success scenario is followed from (4).

Exception Scenario 3: An invalid file is specified.

1. The system calls for the wizard to be opened.

2. The wizard is displayed for the user.
3. The user specifies the location of a file containing an event.
4. The wizard determines that this file does not exist or that it does not contain an event.
5. The wizard responds by informing the user of this error, in a clear but nonintrusive manner.
6. The wizard does not allow the user to continue with importing the file until the issue is resolved.
7. After the user corrects the issue, the main success scenario is followed from (4).

Exception Scenario 4: The import location is invalid.

1. The system calls for the wizard to be opened.
2. The wizard is displayed for the user.
3. The user specifies a location that the file should be imported to that does not comply with eclipse's requirements for importing files into the workspace.
4. The wizard responds by informing the user of this error in a clear but nonintrusive manner.
5. The wizard does not allow the user to continue with file importing until the issue is resolved.
6. After the user corrects the issue, the main success scenario or an alternate success scenario is followed from (4).

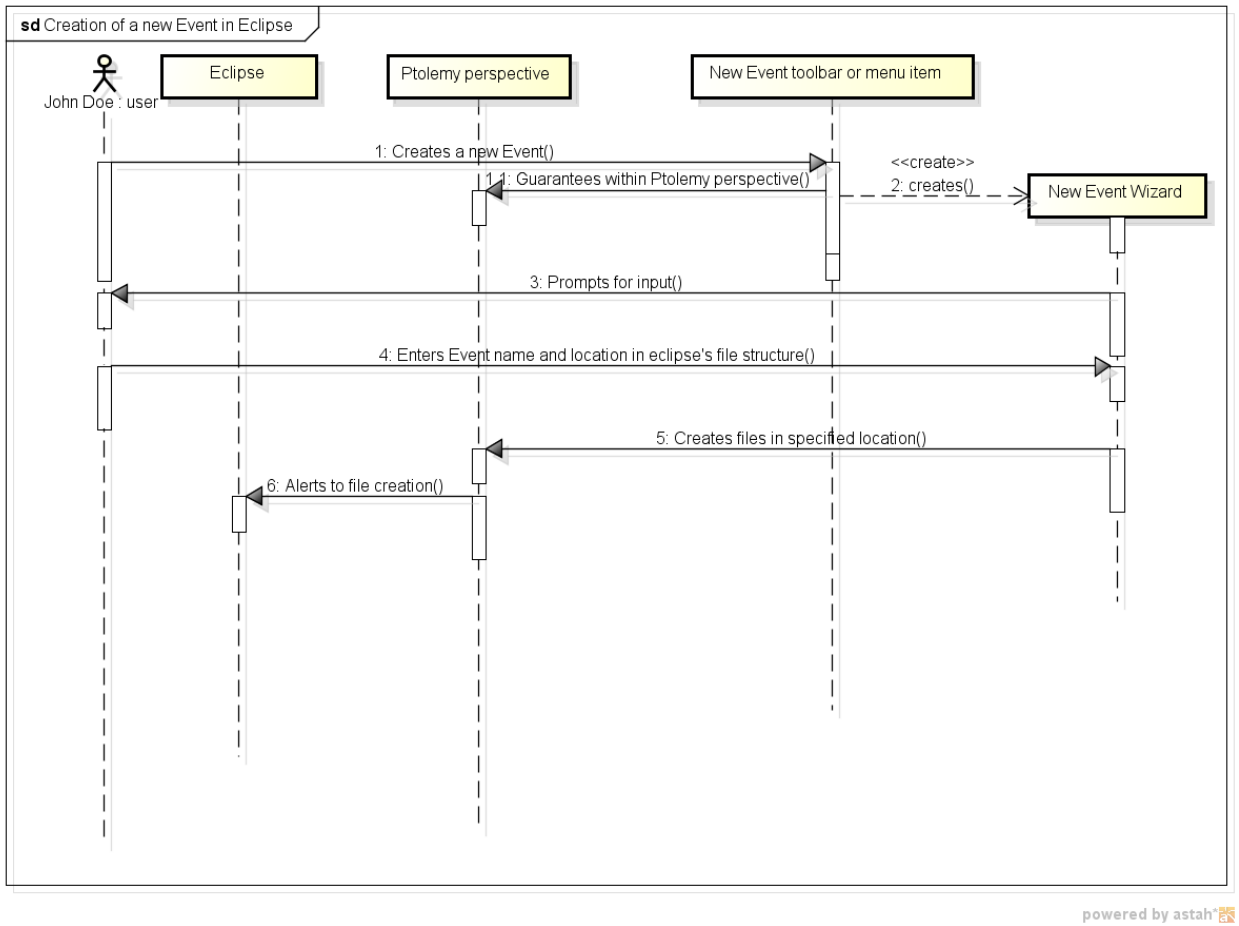

Figure 4: Event Creation

4.3.2: The new Ptolemy Project Wizard.

This section covers the requirements of the new Ptolemy Project Wizard. This covers a technical overview, use cases, and diagrams representing the flow of those use cases.

4.3.2.1: Technical Overview

This wizard allows for the creation of a new Ptolemy project, or the importation of a Ptolemy compliant project. This will be mostly borrowing from the Java project wizard, as all java projects are by definition compatible with Ptolemy (an extension of java).

The new project wizard will, after collecting all users input, create the hierarchy for a Ptolemy project inside of the workspace provided by eclipse.

4.3.2.2: Use cases and Diagrams

The following are Use Cases for the New Event Wizard, following the format outlined in **4.1: Use case structure.**

Use Case 3: Creation of a new Ptolemy project.

Primary Actor: User

Secondary Actor: Eclipse

Level: System

Preconditions: The user is currently in the Ptolemy Perspective.

Success Guarantees: A new Ptolemy project is created with the specified parameters in the location specified by the user.

Trigger: The system calls for the New Ptolemy Project wizard to be opened. This may occur from several sources within the system.

Main Success Scenario:

1. The System calls for the new Ptolemy project wizard.
2. The wizard is displayed for the user.
3. The user specifies a name for the new project, as well as a location for its creation.
4. The location is inside of the current eclipse workspace.
5. The user enters any other valid additional parameters.
6. The wizard creates the file structure required by eclipse, and includes those parameters specified by the user.
7. The wizard informs the system of the change, which in turn informs eclipse. The file structure is then displayed and tracked according to their specifications.

Alternate Success Scenario:

1. The system calls for the new Ptolemy project wizard to be opened.
2. The wizard is displayed for the user.
3. The user specifies that the project will be created in a custom location outside of the eclipse workspace.
4. The user provides the name of the project and the file path for the creation of the project.
5. The wizard creates the file structure with the user parameters so that it may be interpreted by eclipse as a valid project, but in the specified location.

6. The Wizard informs the system of the change, which does not inform eclipse, as it is not in the workspace and does not need to be tracked or displayed according to their standards.

Exception Scenario 1: The user specifies an invalid location.

1. The system calls for the new Ptolemy project wizard.
2. The wizard is displayed for the user.
3. The user specifies a name for the new project, as well as an invalid location for its creation (such as an already occupied location.
4. The wizard responds by informing the user of the error, in a clear but nonintrusive manner.
5. The wizard does not allow creation of the project until the issue is resolved.
6. Once resolved, the main success scenario is followed from (5).

Exception scenario 2: The user specifies an invalid name for the project.

1. The system calls for the new Ptolemy project wizard.
2. The wizard is displayed for the user.
3. The user specifies an invalid name for the new project,
4. The wizard responds by informing the user of the error, in a clear but nonintrusive manner.
5. The wizard does not allow creation of the project until the issue is resolved.
6. Once resolved, the main success scenario is followed from (5).

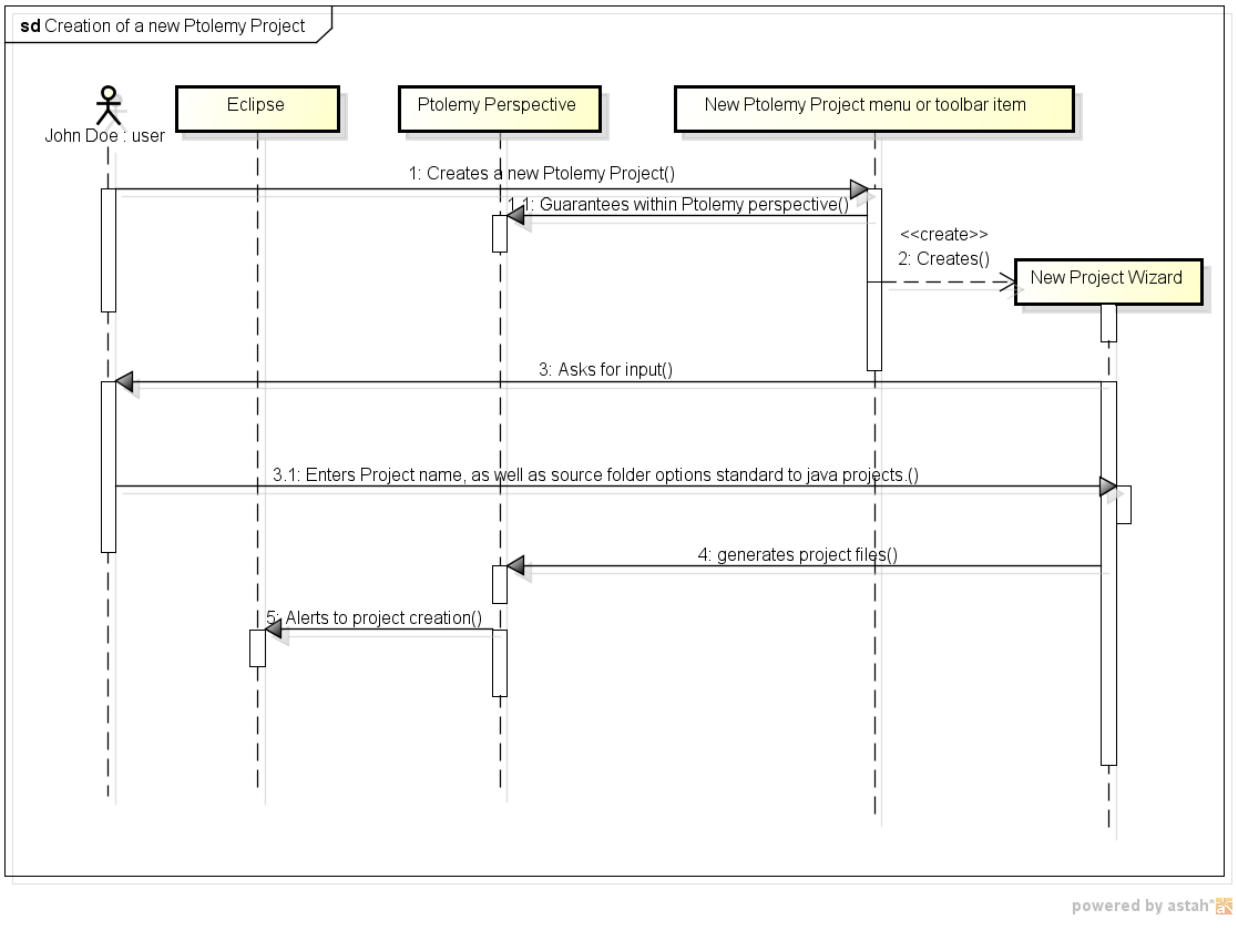

Figure 5: Ptolemy Project Creation

4.4: The UIE package

This package contains all UI elements that require being added to the Perspective upon its opening. This package also includes any miscellaneous objects such as graphics that the UI elements require.

4.4.1: The new Ptolemy Project menu item

This UI item is a simple class that merely calls the wizard for creating a new Ptolemy project. In the setup of the perspective, it should be placed in the file-> menu, next to the new java project item.

4.4.1.1: Technical Overview

This class is responsible for calling the wizard to facilitate the creation of a new Ptolemy Project. This uses the extension point for an eclipse menu item.

4.4.1.2: Use cases and diagrams

This section contains use cases of the format described in **4.1: Use Case Structure.**

Use Case 4: The User Selects the menu item to create a new project
Primary Actor: User
Secondary Actor: Eclipse
Level: User
Preconditions: The user is in the Ptolemy Perspective.
Success Guarantees: The wizard for creating a new Ptolemy project is called.
Trigger: The user navigates to the menu item and selects it.
Main Success Scenario:

1. The user selects the Menu item displayed by the Ptolemy Perspective.
2. The Menu Item creates a new Ptolemy Project Wizard.
3. The use case in **4.2.2: The new Ptolemy Project Wizard** is followed.

For a use case diagram, please refer to figure 5: Ptolemy Project Creation.

4.4.2: The new Ptolemy Project Toolbar button

This UI item is a simple class that merely calls the wizard for creating a new Ptolemy project. In the setup of the perspective, this item should be placed on the toolbar as the default “new project” button, in the same place as the new java project.

4.4.2.1: Technical Overview

This class is responsible for calling the wizard to facilitate the creation of a new Ptolemy Project. This uses the extension point for an eclipse toolbar item.

4.4.2.2: Use cases and diagrams

This section contains use cases of the format described in **4.1: Use Case Structure.**

Use Case 5: The item is used to create a new Ptolemy project.
Primary Actor: User
Secondary Actor: Eclipse

Level: User
Preconditions: The user is in the Ptolemy Perspective.
Success Guarantees: The wizard for creating a new Ptolemy project is called.
Trigger: The user navigates to the toolbar item and selects it.
Main Success Scenario:

1. The user selects the toolbar item displayed by the Ptolemy Perspective.
2. The Menu Item creates a new Ptolemy Project Wizard.
3. The use case in **4.2.2: The new Ptolemy Project Wizard** is followed.

For a use case diagram, please refer to figure 5: Ptolemy Project Creation.

4.4.3: The new event menu item

This UI item is a simple class that merely calls the wizard for creating a new event file. In the setup of the perspective, this UI element should be placed under the file-> new menu, in logical order along with the new class and new interface menu items.

4.4.3.1: Technical Overview

This class is responsible for calling the wizard to facilitate the creation of a new Event file using Ptolemy syntax. This uses the extension point for an eclipse menu item.

4.4.3.2: Use cases and diagrams

This section contains use cases of the format described in **4.1: Use Case Structure.**

Use Case 6: The menu item is used to create a new Event.
Primary Actor: User
Secondary Actor: Eclipse
Level: User
Preconditions: The user is in the Ptolemy Perspective.
Success Guarantees: The wizard for creating a new Event is called.
Trigger: The user navigates to the menu item and selects it.
Main Success Scenario:

1. The user selects the Menu item displayed by the Ptolemy Perspective.
2. The Menu Item creates a new Ptolemy Project Wizard.
3. The use case in **4.3.2: The new Event Wizard** is followed.

For a use case diagram, please refer to figure 4: Event Creation.

4.4.4: The new event toolbar item

This UI item is a simple class that merely calls the wizard for creating a new event file. In the setup of the perspective, this UI element should be placed in the combo box item on the toolbar of eclipse, alongside the new interface and new class toolbar items.

4.4.4.1: Technical Overview

This class is responsible for calling the wizard to facilitate the creation of a new Ptolemy Project. This uses the extension point for an eclipse toolbar item.

4.4.4.2: Use cases and diagrams

This section contains use cases of the format described in **4.1: Use Case Structure.**

Use Case 7: The toolbar item is selected.
Primary Actor: User
Secondary Actor: Eclipse
Level: User
Preconditions: The user is in the Ptolemy Perspective.
Success Guarantees: The wizard for creating a new event is called.
Trigger: The user navigates to the menu item and selects it.
Main Success Scenario:

1. The user selects the Menu item displayed by the Ptolemy Perspective.
2. The Menu Item creates a new Event Wizard.
3. The use case in **4.3.2: The new Event Wizard** is followed.

For a use case diagram, please refer to figure 4: Event Creation.

4.4.5: The run as Ptolemy Program Menu and toolbar item

This UI item is a class that is responsible for compiling the Ptolemy project if it is not already, and running the program.

4.4.5.1: Technical Overview

This class is responsible for calling the compiler and then determining what main types are in the program. After such, it runs the program using the JVM and displays the console for the user. The searching of main types in the program is dependent on the AST, and is left to iteration 2.

4.4.5.2: Use cases and diagrams

This section contains use cases of the format described in **4.1: Use Case Structure.**

Use Case 8: The Menu item is selected to run a Ptolemy program
Primary Actor: User
Secondary Actor: Eclipse, Ptolemy Compiler.
Level: User
Preconditions: The user is in the Ptolemy Perspective and has at least one open Ptolemy project.
Success Guarantees: The Program is compiled and run.
Trigger: The user navigates to the toolbar item and selects it.
Main Success Scenario:

1. The user selects the menu item.
2. The item calls those methods in eclipse to compile the program in the Ptolemy compiler, if it determines that changes have been made since the last compile.
3. The item then calls those methods provided by eclipse to run the program using the class files provided by the compiler and project after compilation.

Exception scenario 1: The Project does not contain a main method to run.

1. After searching, the item or those methods provided by eclipse do not find a main method for program execution.
2. The program halts its compilation function, and informs the user of this error.
3. No further action is taken until the inclusion of a main type, which will cause recompiling of the program. See Main Success scenario for a valid program's compilation.

Exception scenario 2: The project encounters a compilation error.

1. After starting the compilation process, the compiler returns an error.
2. The item stops its compilation process.
3. The stack trace is printed to the console. This effectively displays the error.
4. The program will need modification and eventual recompiling before the error can be resolved. See Main Success Scenario for compilation of a valid program.

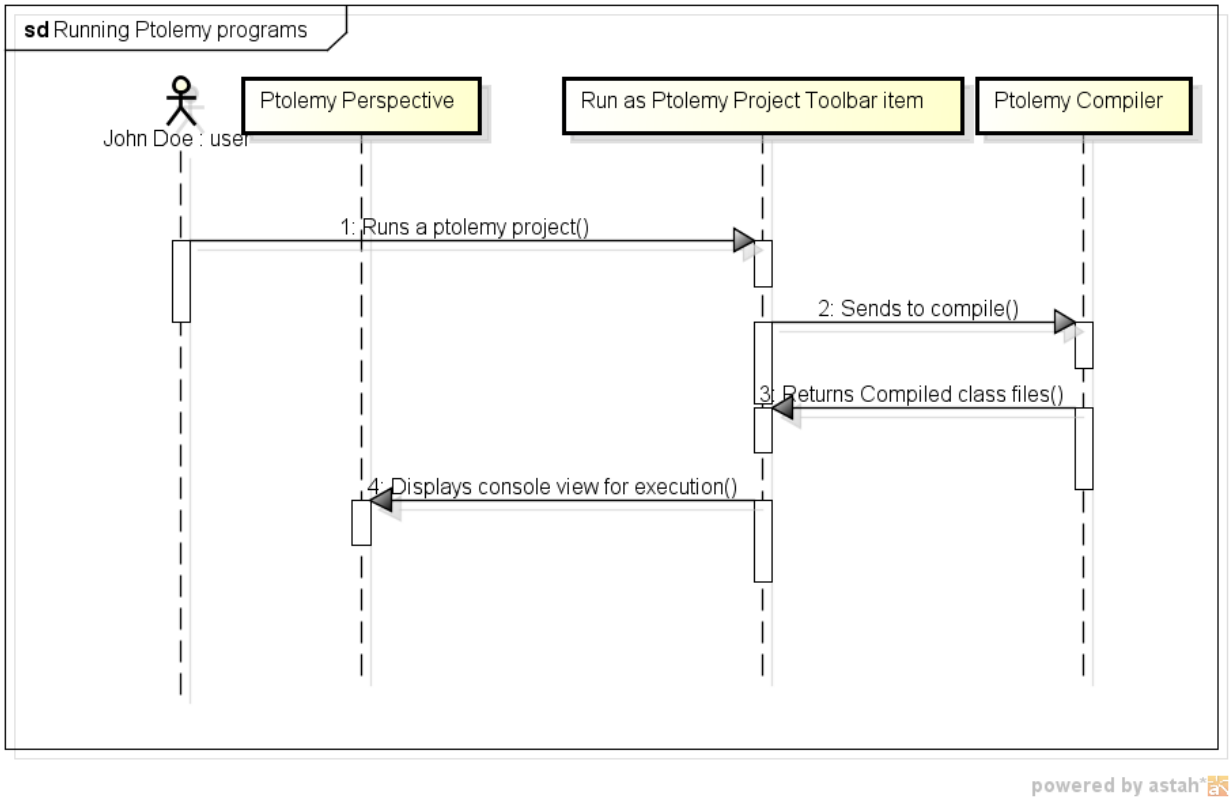

Figure 5: Running Ptolemy programs from the toolbar item

4.5: Iteration 2 Requirements

As stated in section **1.5: Project structure for Iteration 2**, a high level explanation of the structure and use cases of the iteration 2 changes will be detailed here. For more detailed aspects of these changes, view the Design Document.

4.5.1: Recognizing Ptolemy language features

The changes that involve recognizing language features specific to Ptolemy should be implemented in a way consistent with the structure and style of Eclipse's implementation of recognizing Java language features, as the Ptolemy language is an extension of Java.

4.5.1.1: Technical Description and Process

Within Eclipse's internal structure is a collection of files relating to using the java language, which is known as the Java Development Toolkit (hereafter referred to as the JDT). Within this collection of files is the structure and implementation of recognition of Java language items. Our project has retrieved the source files of the JDT and modified several files to be compatible with and recognize Ptolemy language features.

Currently in eclipse we must use this modified JDT to run both eclipse and Ptolemy programs, and ensuring that the user is within the Ptolemy perspective is not possible. We have discussed this with our client and he assures us that it is within acceptable limits to proceed in this manner.

4.5.1.2: Use cases and diagrams

This section contains use cases of the format described in **4.1: Use Case Structure.**

Use Case 9: Introducing Ptolemy language features to the Eclipse Text Editor.
Primary Actor: User
Secondary Actor: Eclipse, Ptolemy Compiler.
Level: User
Preconditions: The user is in the Ptolemy Perspective and has the text editor open.
Success Guarantees: The file is recognized to have Ptolemy language features and displays them accordingly, or is syntactically incorrect and displays a helpful error message.
Trigger: The user navigates to the text editor and enters a Ptolemy language feature.
Main Success Scenario:

1. The user selects the text editor
2. The user enters a Ptolemy language feature in the file opened in the text editor
3. Eclipse and the Ptolemy plugin recognizes the Ptolemy language feature
4. The feature is syntactically correct and displayed accordingly

Alternate Success Scenario:

1. The user selects the text editor
2. The user enters a Ptolemy language feature in the file opened in the text editor
3. Eclipse and the Ptolemy plugin recognizes the Ptolemy language feature

4. The feature is recognized to be syntactically invalid and displays an error accordingly

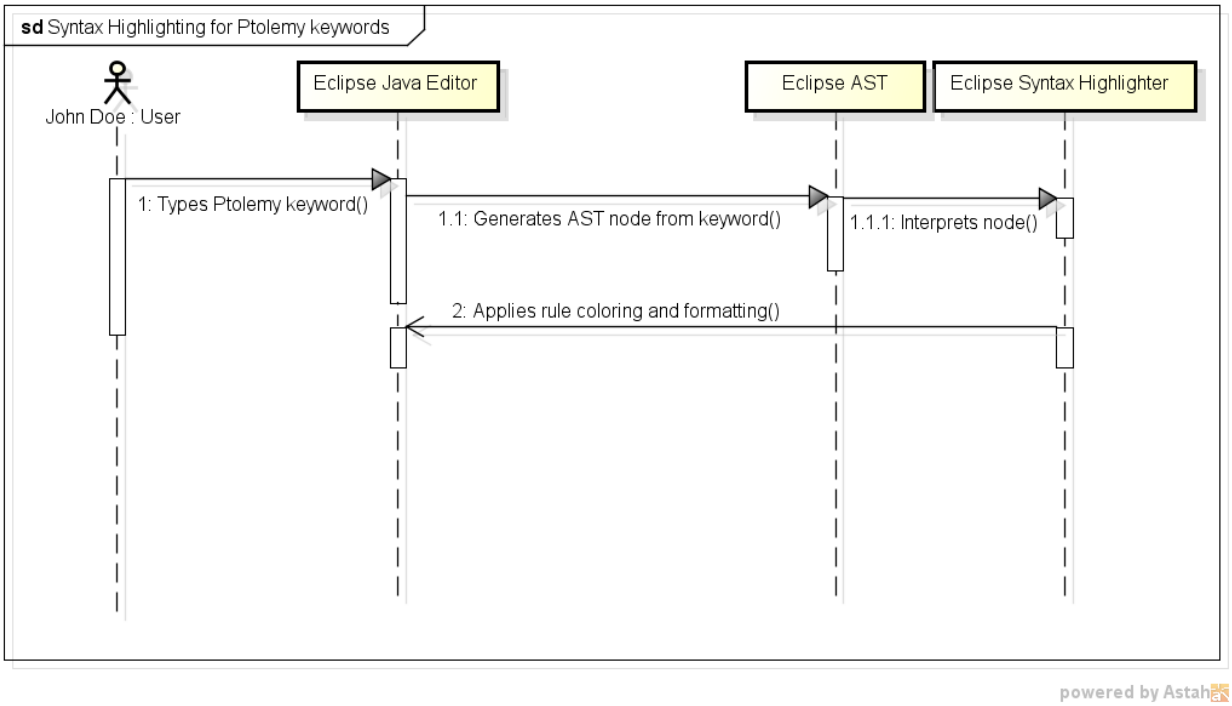

Figure 6: Recognition of Ptolemy Language Features

4.5.2: Syntax Highlighting Ptolemy language features

The changes that involve displaying Ptolemy language features in a way consistent with the display of Java language features in the Eclipse editor should maintain the structure and style of Java syntax highlighting. This is due to the fact that Ptolemy is an extension of Java, and the requirement that Java features should be maintained while adding Ptolemy Features.

4.5.2.1: Use cases and diagrams

This section contains use cases of the format described in **4.1: Use Case Structure.**

This feature also falls under the purview of **Use Case 9: Introducing Ptolemy language features to the Eclipse Text Editor,** as well as **Use case 2: New Event Creation,** and any other instance where Ptolemy language features would be added to the text editor.

For a diagram, please refer to **Figure 6: Recognition of Ptolemy Language Features**

4.5.3: Content Assist in the Ptolemy language

Changes that involve the Ptolemy language features and content assist (aka code completion) should be consistent with both the style and architecture of java Content assist. This is again because Ptolemy is an extension of java, and therefore the feature should extend in the same style.

4.5.3.1: Technical Description and Process

Within the Internal source code for Eclipse there is a collection of files known as the JDT, which handles all features relating to java development. Within this file system is the source files for content assist,

which may need modification to support content assist with Ptolemy language features described earlier in this document.

4.5.3.2: Use cases and Diagrams

This section contains use cases of the format described in **4.1: Use Case Structure.**

Use Case 10: Invoking content assist on a Ptolemy language feature
Primary Actor: User
Secondary Actor: Eclipse
Level: User
Preconditions: The user is in the Ptolemy Perspective and chooses content assist on a Ptolemy Language feature.
Success Guarantees: The Eclipse content assist window appears with a list of valid options for content assist with regards to the syntax and style of the Ptolemy language. This may include other Ptolemy language features.
Trigger: The user selects content assist on a Ptolemy language feature.
Main Success Scenario:

5. The user selects the text editor
6. The user enters a Ptolemy language feature in the file opened in the text editor
7. The user selects content assist on the created feature
8. The feature is analyzed and a content assist window appears with valid options
9. The user chooses an option
10. The option is added to the text editor in a syntactically correct fashion

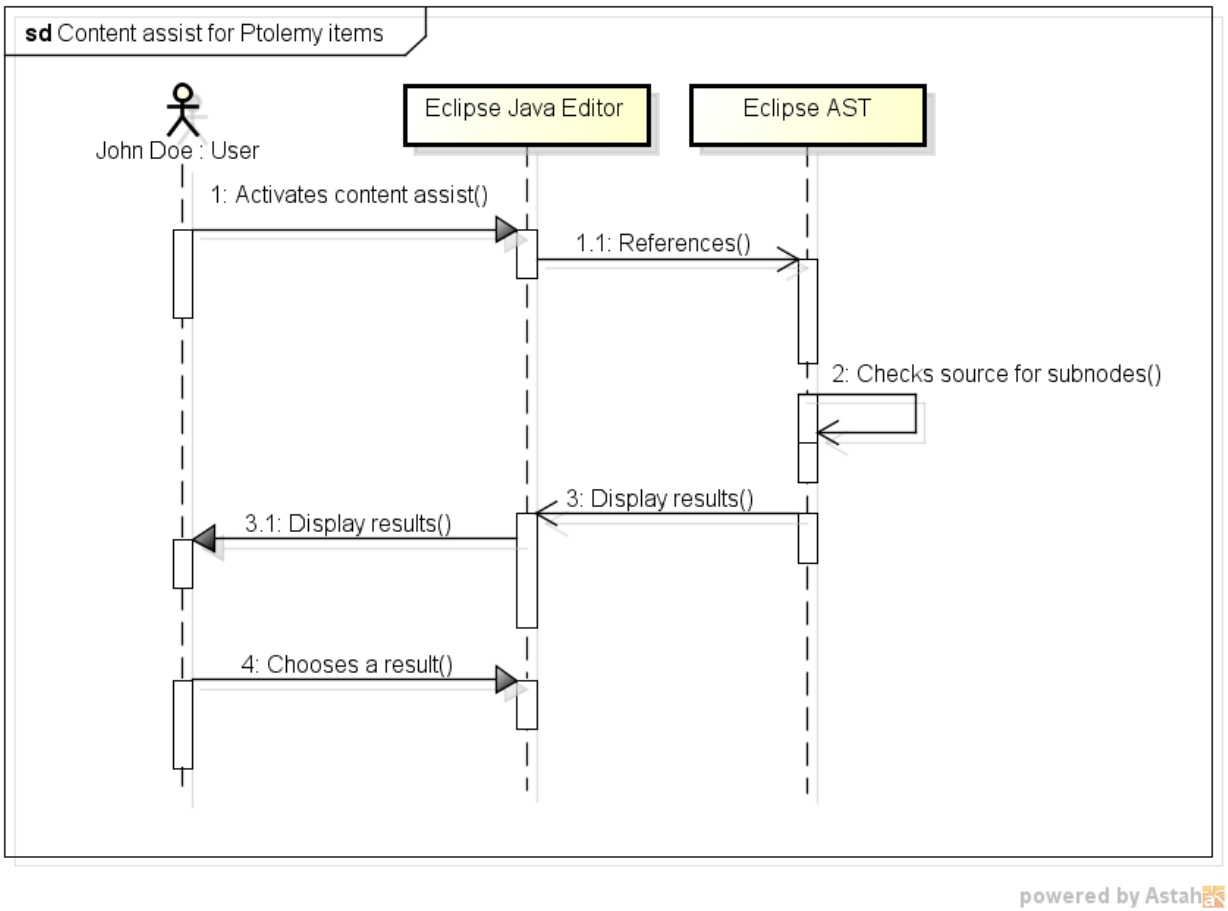

Figure 7: Content assist for Ptolemy items

5: Test and Evaluation Plan

This section covers test plans and thoughts on testing the code produced by this team.

5.1: Test Driven Design

A standard industry level technique is to meet with the client and elicit requirements, and to turn those requirements into test cases. This technique is known as test driven design. With this project, we have taken requirements and other client needs and turned them into use cases and other verifiable basis for testing. This helps us ensure that requirements are met and that development focuses on client needs, instead of using resources on extraneous or non-crucial features.

5.2: Code Reviews

Code reviews will be performed weekly or bi-weekly depending on client needs (less client needs means more time for review, more client needs means less time for review). During this time, we will go over the current set of revisions and or work by the team, and determine if their contribution is in line with these specifications.

5.3: Static Testing

After successfully passing code review and commits to the repository provided by the client, the code will go into a test phase, where either a Junit test suite or manual testing may be applied to validate the correctness of the code. These test results will be recorded and made available to the team as well as the client and advisors.

5.4: Regression Testing

Upon completing iterations (or mini iterations), previous test cases will be evaluated and modified if needed, and testing previous commits or code will ensure that we continue moving forward with working product. Testing these older portions of code will be done in the manner they were originally done if possible. These test results will be held separately and made available to the team, client, and advisors.

5.5: Thoughts on Testing

Since Iteration 1 depends almost solely on UI testing, programmatic UI testing materials may be considered, such as abbot/Costello. These tools may not work for more complex tasks involving the Ptolemy Syntax, and so their usefulness will be considered before making concrete decisions.

6: Conclusion

This section covers miscellaneous information relating to those involved with this project.

6.1: Client information

Hirdesh Rajan
101 Atanasoff hall
hridesh@iastate.edu

Robert Dyer
226 Atanassoff hall
rdyer@iastate.edu

6.2: Advisor Information

Hirdesh Rajan
101 Atanasoff hall
hridesh@iastate.edu

Robert Dyer
226 Atanassoff hall
rdyer@iastate.edu

6.3: Team Information

Adam Rotondi **-Team Leader** **-Developer** **arotondi@iastate.edu**	**Kimberly Eue** **-webmaster** **-developer** **keue@iastate.edu**	**Justin Landsgard** **-webmaster** **-developer** **jstnlnds@iastate.edu**	**Tim Flanigan** **-communicator** **-developer** **flanigan@iastate.edu**
Kyle Bradwell **-Communicator** **-developer** **bradwell@iastate.edu**	Team 13 Dec1213@iastate.edu		