Component Design

For GMoDS Visualizer and Test Driver

Version 1.0

Submitted in partial fulfillment of the requirements of the degree of MSE

Mike Fraka
CIS 895 – MSE Project
Kansas State University

Table of Contents

1 Introduction

This is the component design for the GMoDS Visualizer and Test Driver Masters of Software Engineering final project.

1.1 References

1. "Vision Document 1.0 or 2.0" available at http://people.cis.ksu.edu/~mfraka/FrakaMSE.html.
2. "System Architecture 2.0" available at http://people.cis.ksu.edu/~mfraka/FrakaMSE.html.

2 System Context

The system context is shown in Figure 1 below.

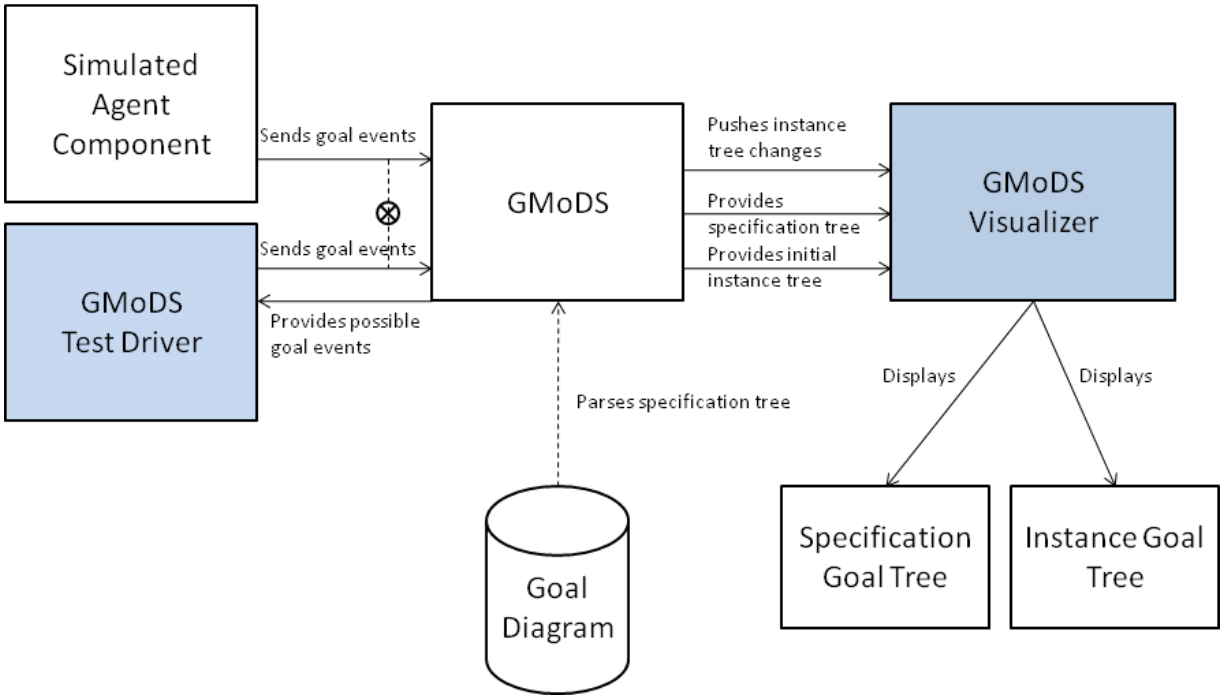

Figure 1 GMoDS Test Driver and Visualizer system context

More detail on the system context is available in [1] (see 1.1 above)

3 System Architecture

This section documents the system architecture in a component diagram and references [2] for more information.

3.1 System Components

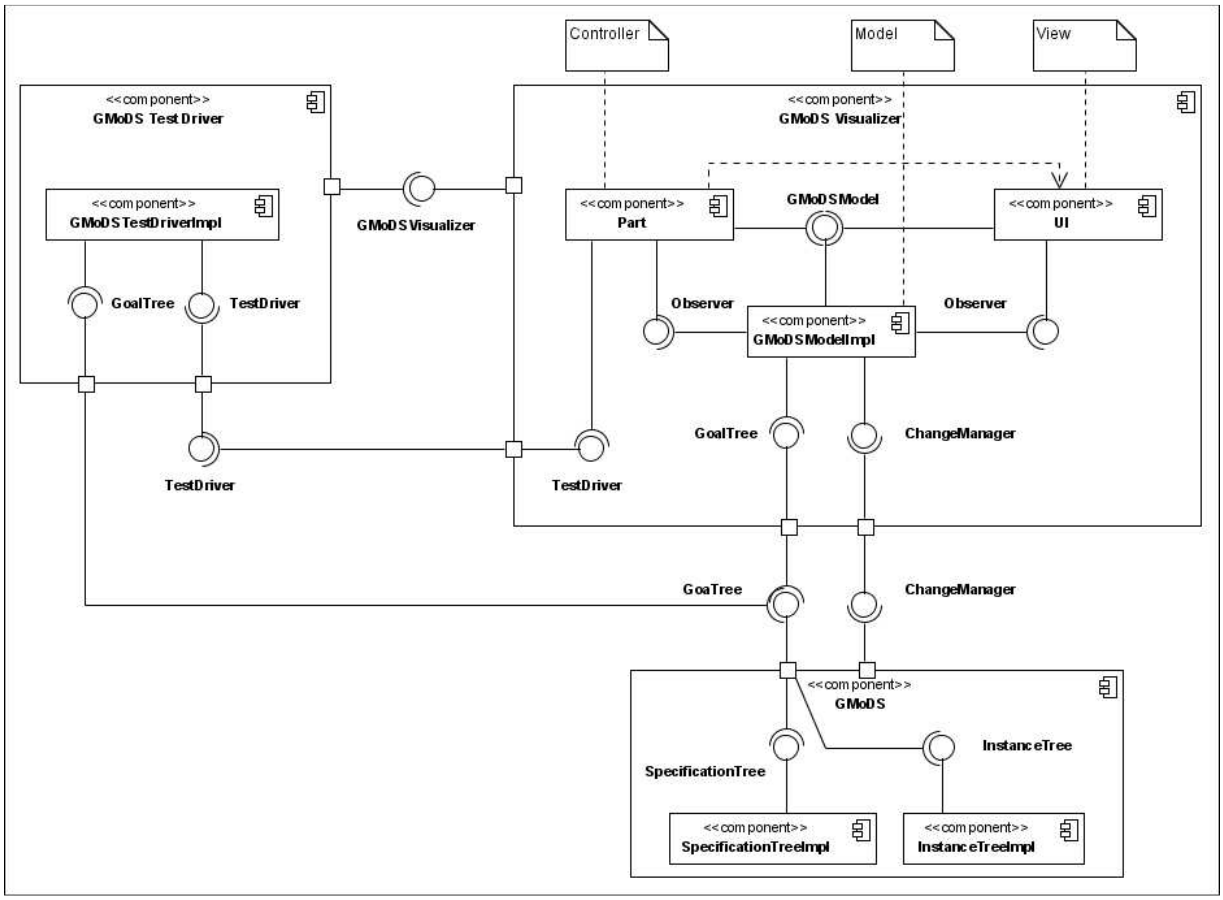

Figure 2 System components

Figure 2 System components shows the three components developed or reused in this project.

For details on this diagram and the component architecture, see [2] noted at 1.1 above.

4 Component Design

This section documents the detailed design of each system component.

4.1 GMoDS Test Driver Component Design

This section documents the detailed static and behavioral design of the GMoDS Test Driver component.

4.1.1 GMoDS Test Driver Static Structure

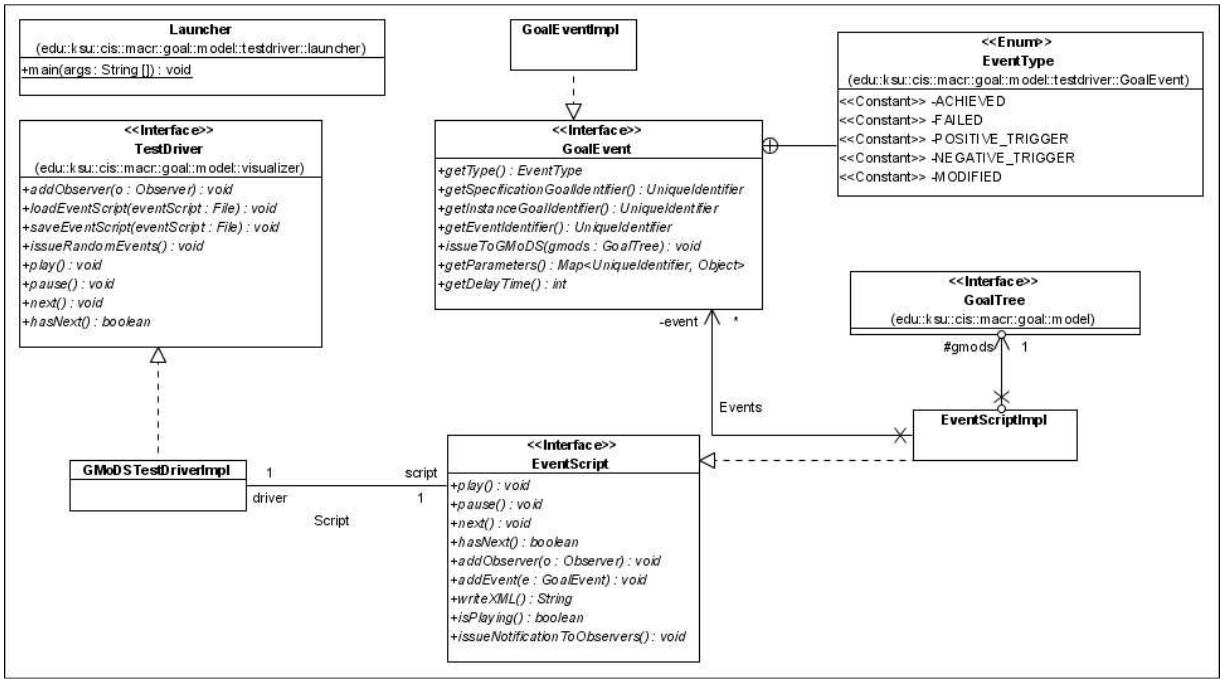

Figure 3 GMoDS Test Driver Architecture

Figure 3 above shows the GMoDS Test Driver architecture described in detail in [2]. Figure 4 below shows the component classes that implement random events for the GMoDS Test Driver.

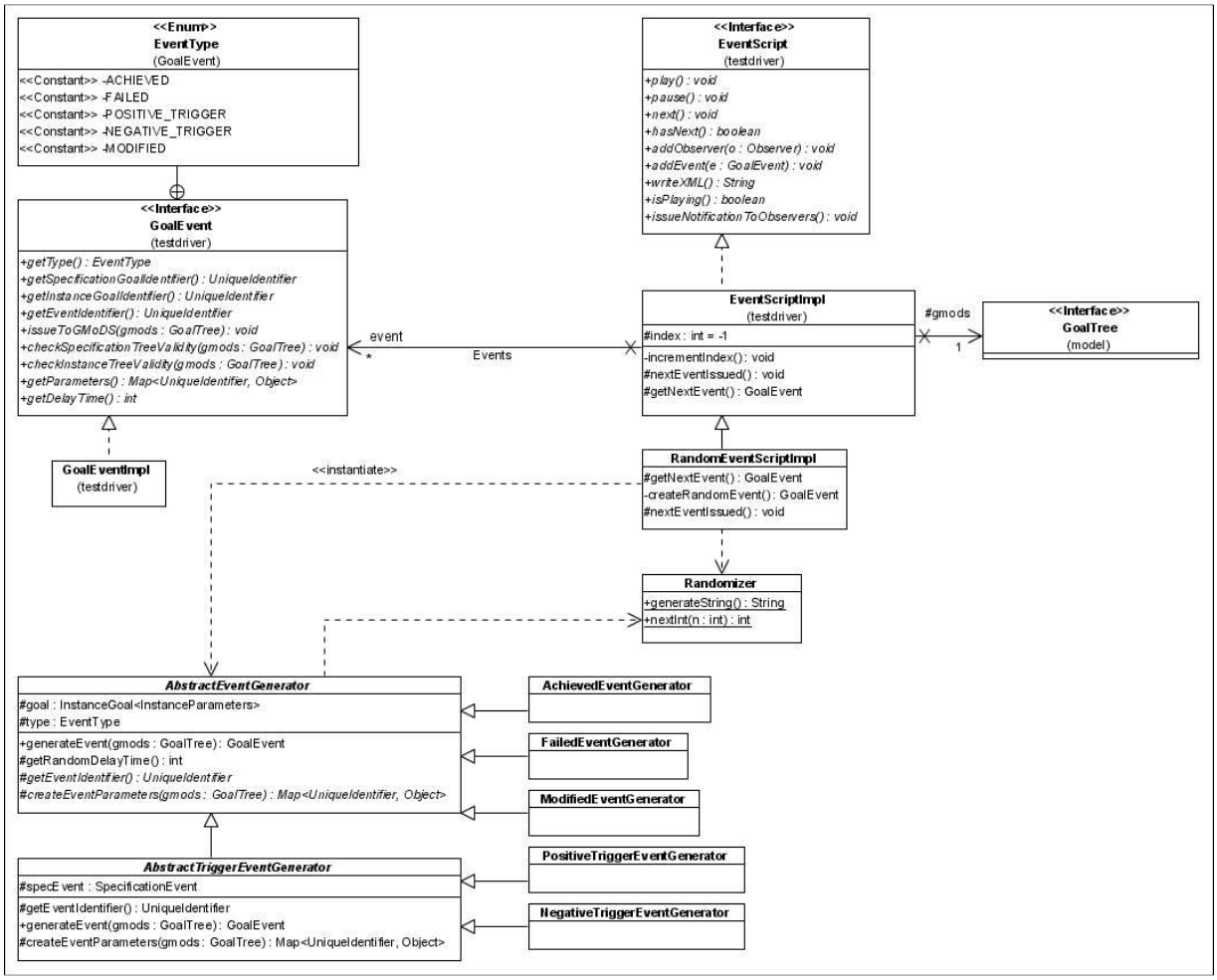

Figure 4 GMoDS Test Driver Random Events Component Classes

4.1.1.1 GMoDS Test Driver Local Module Responsibilities

This section describes the responsibilities of GMoDS Test Driver local modules (not described in [2]).

Table 1 GMoDS Test Driver Module Responsibilities

Component	Responsibilities
AbstractEventGenerator	Define the behaviors required to generate a random GoalEvent.
AbstractTriggerEventGenerator	Define the behaviors of a trigger-based GoalEvent.
AchievedEventGenerator	Generate a random ACHIEVED GoalEvent.
FailedEventGenerator	Generate a random FAILED GoalEvent.
ModifiedEventGenerator	Generate a random MODIFIED GoalEvent.

Component	Responsibilities
PositiveTriggerEventGenerator	Generate a random POSITIVE_TRIGGER GoalEvent.
NegativeTriggerEventGenerator	Generate a random NEGATIVE_TRIGGER GoalEvent.
Randomizer	Provide random number and string utilities.

4.1.1.2 GMoDS Test Driver Local Module Interface Specifications

Table 2 AbstractEventGenerator Interface Specifications

Generate a random GoalEvent.	Syntax:	generateEvent(gmods : GoalTree) : GoalEvent
	Pre:	gmods != null
	Pre:	GoalEvent that can be generated by this generator is applicable to the current state of GMoDS.
	Post:	Result = new random GoalEvent of the type represented by this generator.
Generate a random event delay time.	Syntax:	getRandomDelayTime() : int
	Pre:	none
	Post:	Result = new random integer in the range defined by the GMoDS Visualizer's RandomEventParameters.
Get the identifier of the random event known to GMoDS.	Syntax:	getEventIdentifier() : UniqueIdentifier
	Pre:	none
	Post:	Result = the UniqueIdentifier of the generated GoalEvent that identifies it to GMoDS.
Create random event parameters if applicable.	Syntax:	createEventParameters(gmods : GoalTree) : Map<UniqueIdentifier, Object>
	Pre:	none
	Post:	Result = a new Map<UniqueIdentifier, Object> containing the applicable parameter names and their random values.

4.1.1.3 GMoDS Test Driver Design Rationale

I chose event generators to compactly represent each potential GoalEvent available in the current state of GMoDS, delaying expansion until after the potential event is randomly selected. This increases the efficiency of incremental event generation.

4.1.2 GMoDS Test Driver Behavior

Figure 5 below shows the EventScriptImpl.addEvent method. Each GoalEvent added to the script must first pass all validity checks with respect to the specification tree. If an event fails, an IllegalGoalEventException is thrown, logged, and presented to the user; the event is not added. If the event passes the validity checks, it is added to the script and all observers of notified of the change.

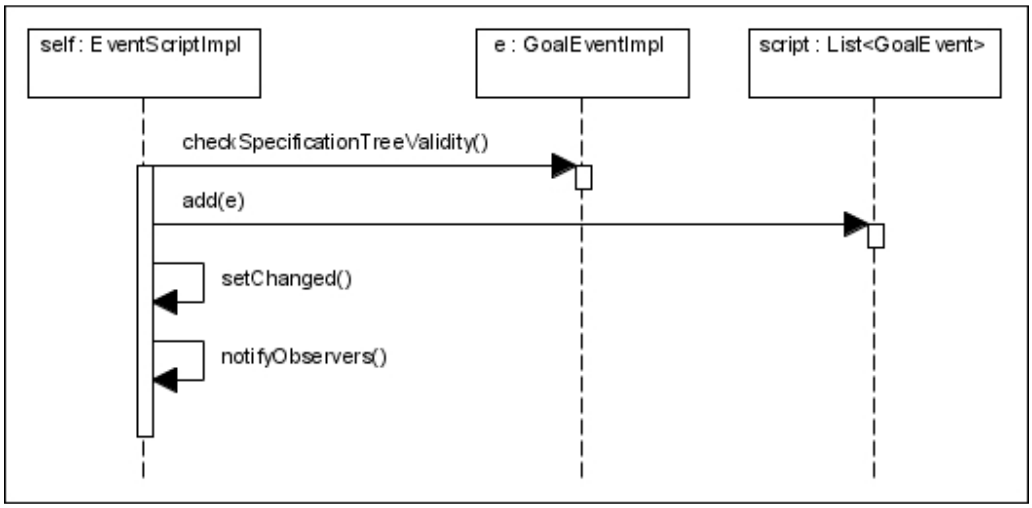

Figure 5 EventScriptImpl.addEvent(GoalEvent e)

Figure 6 below shows the EventScriptImpl.next method. The default implementation of getNextEvent() provides deterministic event script operation simply selecting the next event in the file. RandomEventScriptImpl overrides getNextEvent() to incrementally create the next random event. The incrementIndex() method moves the event pointer to the following event. The next event checks its validity with respect to GMoDS' instance tree. If an event fails, an IllegalGoalEventException is thrown, logged, and presented to the user; the event is not issued to GMoDS. If the event passes the validity checks, it is issued to GMoDS. The script then notifies itself that it has issued the next event. This is a hook for the RandomEventScriptImpl to override to prepare to create the next random event. Finally, script notifies observers of the change in its state.

Figure 7 below shows the RandomEventScriptImpl.getNextEvent method. This method refers to a data member called "nextEvent" used to hold onto the GoalEvent currently being issued to GMoDS, so that it can be added to the script in "nextEventIssued()" after is passes validity checks and is issued. This allows the script to grow incrementally and be saved to a file. The next event can be created randomly if the "nextEvent" data member has been set to null by "nextEventIssued()".

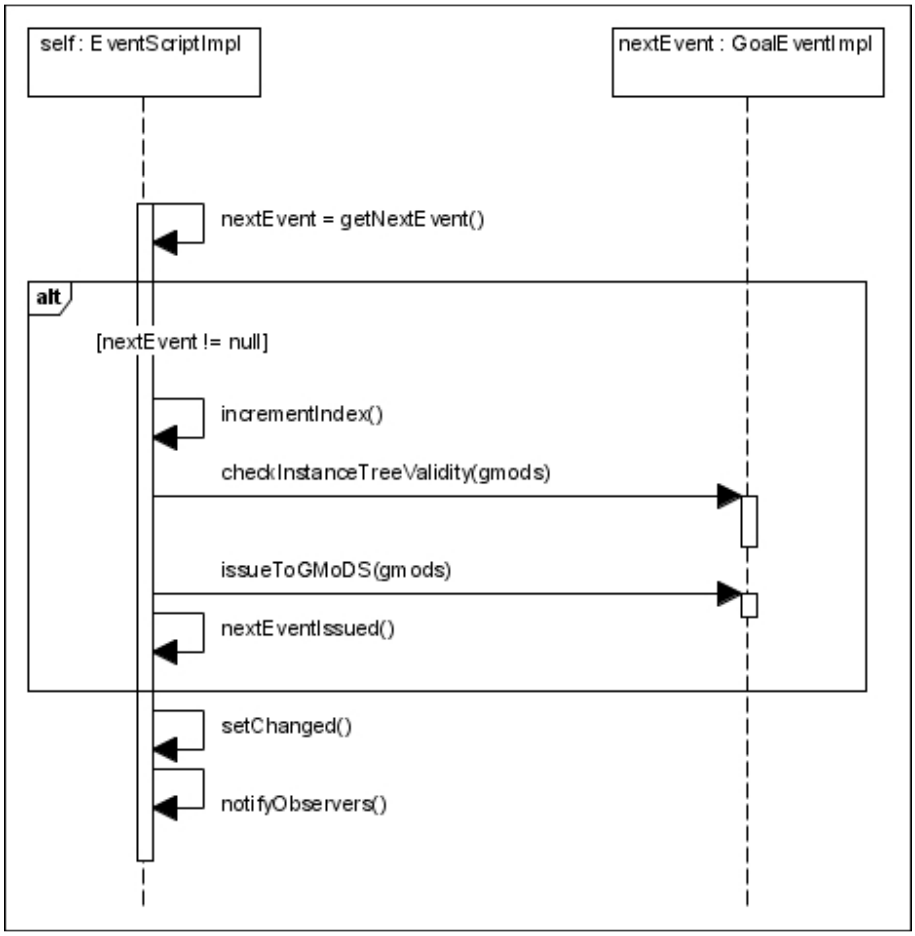

Figure 6 EventScriptImpl.next()

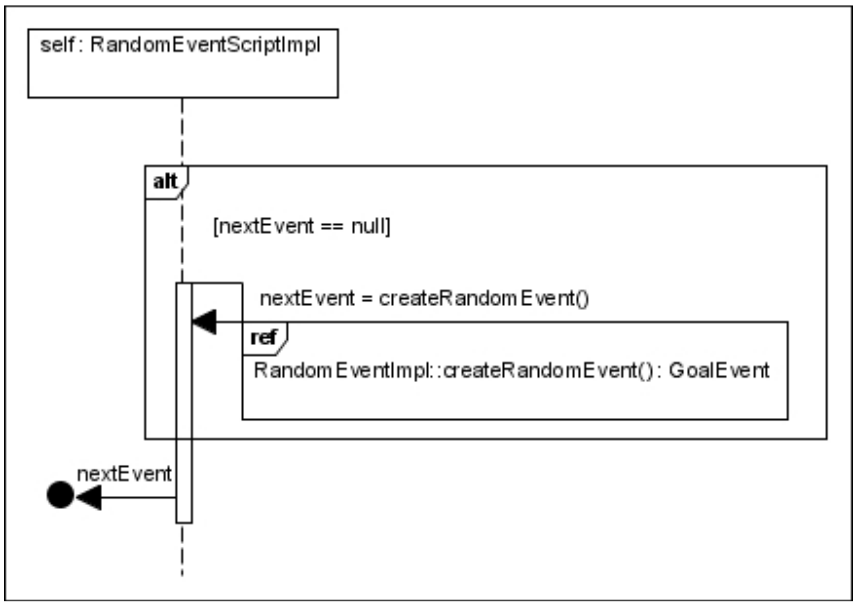

Figure 7 RandomEventScriptImpl.getNextEvent()

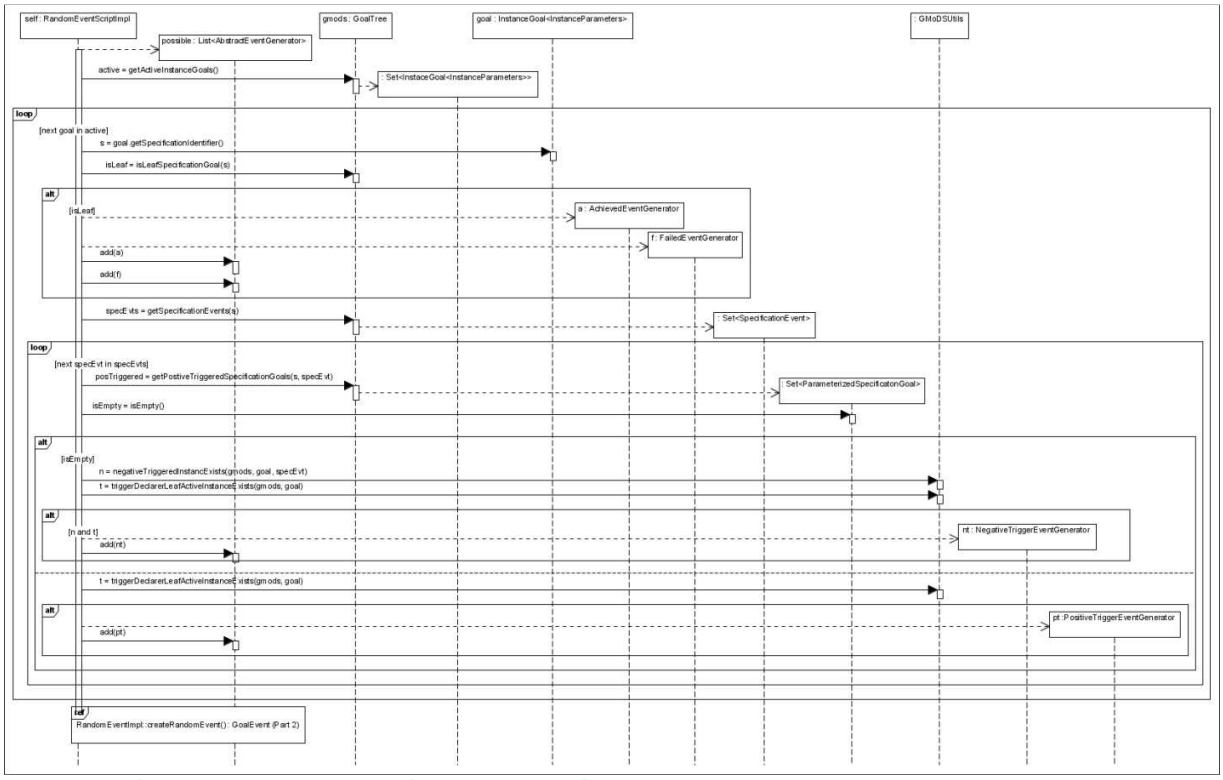

Figure 8 RandomEventScript.createRandomEvent() : GoalEvent (Part 1)

Figure 8 above shows the first half of the process of creating a random GoalEvent. Every active leaf instance goal may be ACHIEVED or FAILED so event generators of these types are added to the list "possible". Then every specification event of each active goal is obtained from GMoDS. The method loops on each specification event and determines whether it defines a positive trigger or a negative trigger.

If it is a negative trigger, an instance goal pointed to by the negative trigger must exist and an instance of a leaf specification goal descended from the specification goal that declared the trigger must exist. If these conditions are met, a NegativeTriggerGenerator is added to "possible" representing the specification event.

If it is a positive trigger, an instance of a leaf specification goal descended from the specification goal that declared the trigger must exist. If this condition is met, a PositiveTriggerGenerator is added to "possible" representing the specification event.

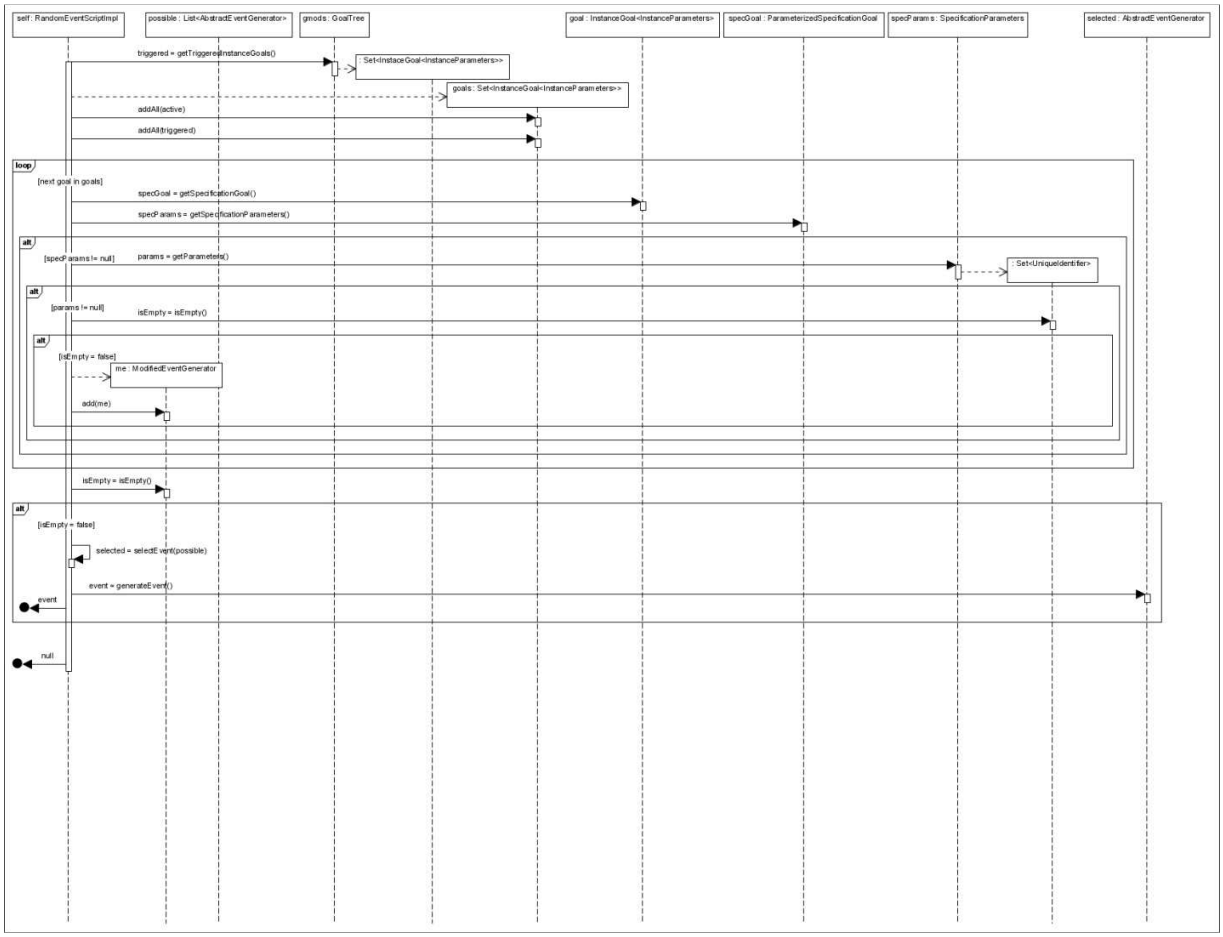

Figure 9 RandomEventScriptImpl.createRandomEvent() : GoalEvent (Part 2)

Figure 9 above shows the second half of the process of creating a random GoalEvent. All triggered and active instance goals may be modified if their specification goal defines parameters. If so, a ModifiedEventGenerator is added to "possible". If there is at least one possible GoalEvent, an AbstractsEventGenerator is randomly selected from "possible" and it generates a random GoalEvent which is returned.

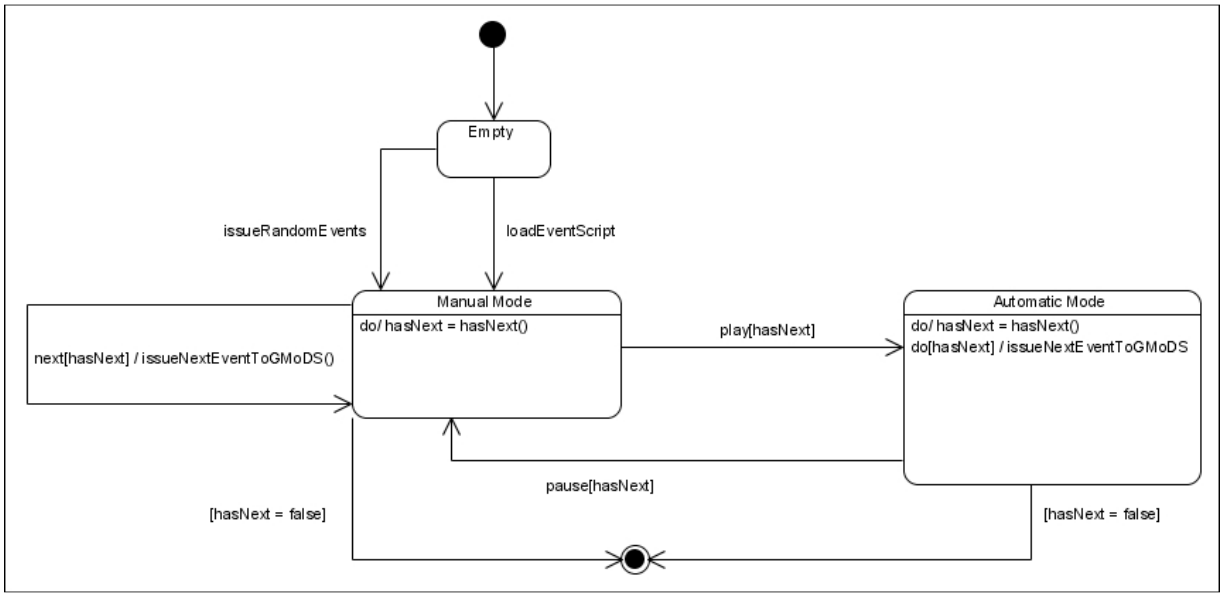

Figure 10 GMoDS Test Driver UI Controls State Diagram

Figure 10 above shows the states of the GMoDS Test Driver in response to the toolbar buttons and menu items that control it. This diagram suppresses differences between random and file-based events (incremental event generation versus a complete script load). The Test Driver starts with an empty script. If the user selects "load event script" or "issue random events" the Test Driver moves to Manual Mode. Clicking next issues the next event to GMoDS if valid and returns to Manual Mode if there is a next event possible. Clicking play while in Manual Mode moves the Test Driver to Automatic Mode if there is a next event possible. Clicking pause while in Automatic Mode moves the Test Driver to Manual Mode if there is a next event possible. In either Manual or Automatic Mode if there is not an event possible the Test Driver is finished.

4.2 GMoDS Visualizer Component Design

The GMoDS Visualizer uses the Model-View-Controller architecture. This section shows the detailed component design in separate sections for the model, view, and controller portions of the architecture.

4.2.1 GMoDS Visualizer Model Static Structure

Figure 11 below shows the "model" portion of the GMoDS Visualizer architecture to show how GMoDS is referenced.

Figure 12 below shows the detailed component classes of the "model".

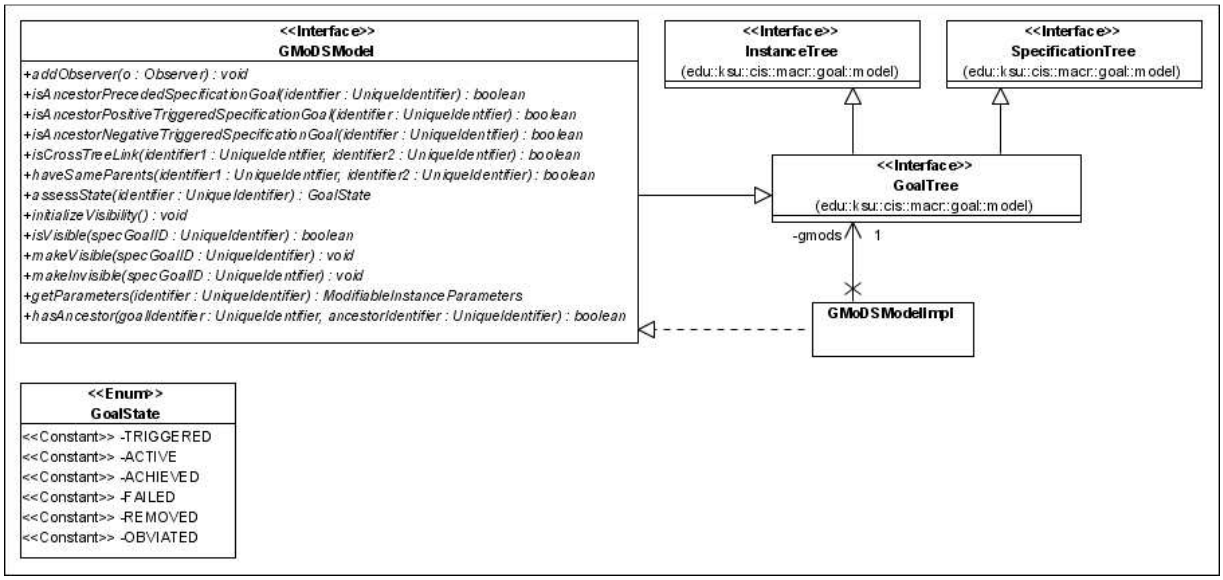

Figure 11 GMoDS Visualizer Model Architecture

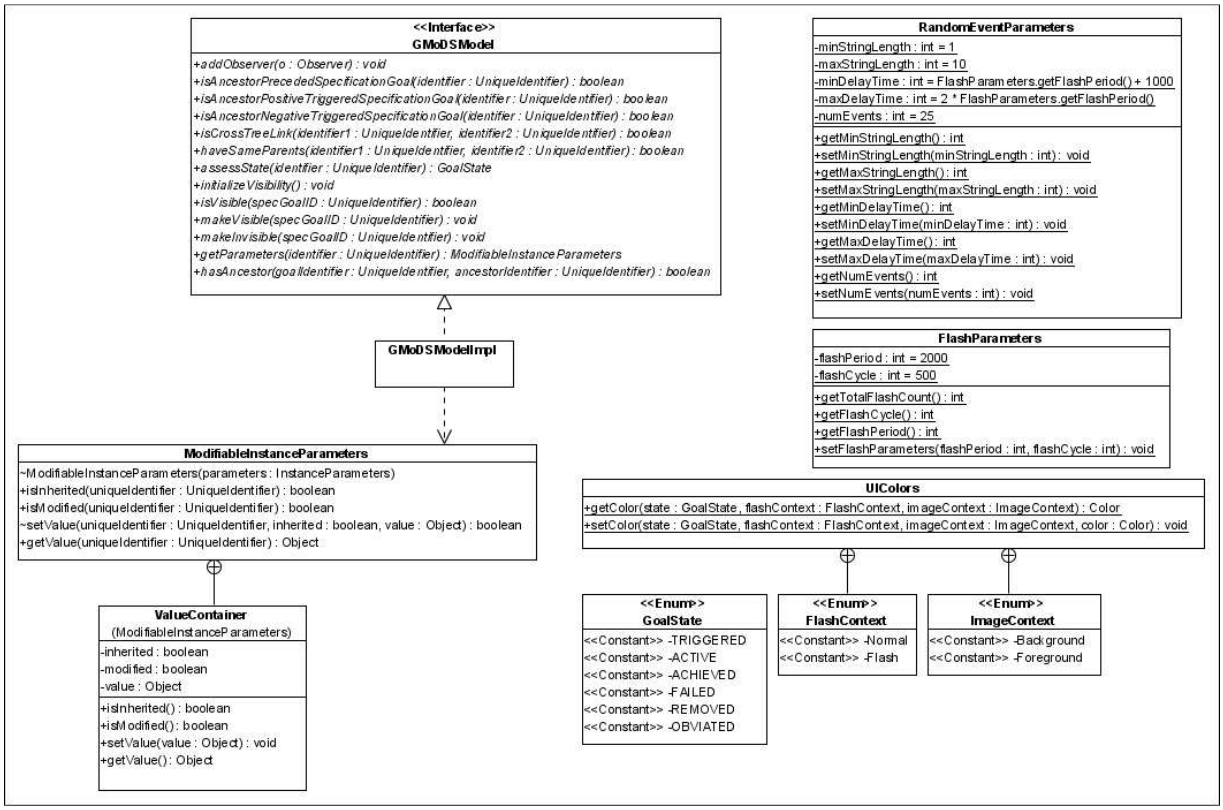

Figure 12 GMoDS Visualizer Model Component Classes

4.2.1.1 GMoDS Visualizer Model Local Module Responsibilities

Table 3 GMoDS Visualizer Model Module Responsibilities

Component	Responsibilities
ModifiableInstanceParameters	Record the current value of each InstanceGoal parameter so that if the value changes it can be ascribed the parameter value origin “MODIFICATION”.
ValueContainer	Record the current value of a particular InstanceGoal parameter.
RandomEventParameters	Define the parameters guiding random event generation.
FlashParameters	Define the parameters guiding InstanceGoalUI flashing.
UIColors	Define the colors used when drawing an InstanceGoalUI for the combination of GoalState, FlashContext, and ImageContext.
GoalState	Enumerate the possible goal states.
FlashContext	Enumerate the possible states of a flash.
ImageContext	Enumerate the portions of an image requiring colors.

4.2.1.2 GMoDS Visualizer Model Local Module Interface Specifications

Table 4 ModifiableInstanceParameters Interface Specifications

Query the inherited property of a specific parameter.	Syntax:	isInherited(uniqueIdentifier : UniqueIdentifier) : boolean
	Pre:	uniqueIdentifier != null
	Post:	Result = true if the specified parameter’s value is inherited.
Query the modified property of a specific parameter.	Syntax:	isModified(uniqueIdentifier : UniqueIdentifier) : boolean
	Pre:	uniqueIdentifier != null
	Post:	Result = true if the specified parameter’s value has changed.

Set the value of a specific parameter.	Syntax:	setValue(uniqueIdentifier : UniqueIdentifier, inherited : boolean, value : Object) : boolean
	Pre:	uniqueIdentifier != null
	Pre:	value != null
	Post:	Result = true if the specified parameter's value has changed.
Get the value of a specific parameter.	Syntax:	getValue(uniqueIdentifier : UniqueIdentifier) : Object
	Pre:	uniqueIdentifier != null
	Post:	Result = the value of the parameter.

Table 5 ValueContainer Interface Specifications

Query the inherited property of a specific parameter.	Syntax:	isInherited() : boolean
	Post:	Result = true if the specified parameter's value is inherited.
Query the modified property of a specific parameter.	Syntax:	isModified() : boolean
	Post:	Result = true if the specified parameter's value has changed.
Set the value of a specific parameter.	Syntax:	setValue(value : Object) : boolean
	Pre:	value != null
	Post:	Result = true if the specified parameter's value has changed.
Get the value of a specific parameter.	Syntax:	getValue() : Object
	Post:	Result = the value of the parameter.

Table 6 RandomEventParameters Interface Specification

Query the minimum string length for a random parameter value.	Syntax:	getMinStringLength() : int
	Post:	Result = the minimum string length for a parameter value.

Set the minimum string length for a random parameter value.	Syntax:	setMinStringLength(minStringLength : int) : void
	Post:	Record the minimum string length for a random parameter value.
Query the maximum string length for a random parameter value.	Syntax:	getMaxStringLength() : int
	Post:	Result = the maximum string length for a parameter value.
Set the maximum string length for a random parameter value.	Syntax:	setMaxStringLength(maxStringLength : int) : void
	Post:	Record the maximum string length for a random parameter value.
Query the minimum delay time for a random event.	Syntax:	getMinDelayTime() : int
	Post:	Result = the minimum delay time for a random event.
Set the minimum delay time for a random event.	Syntax:	setMinDelayTime (minDelayTime : int) : void
	Post:	Record the minimum delay time for a random event.
Query the maximum delay time for a random event.	Syntax:	getMaxDelayTime () : int
	Post:	Result = the maximum delay time for a random event.
Set the maximum delay time for a random event.	Syntax:	setMaxDelayTime (maxDelayTime : int) : void
	Post:	Record the maximum delay time for a random event.
Query the maximum number of random events.	Syntax:	getNumEvents () : int
	Post:	Result = the maximum number of random events.
Set the maximum number of random events.	Syntax:	setNumEvents (numEvents : int) : void
	Post:	Record the maximum number of random events.

Table 7 FlashParameters Interface Specification

Query the total number of times an InstanceGoalUI should flash.	Syntax:	getTotalFlashCount() : int
	Post:	Result = the total number of times an InstanceGoalUI should flash.

Query the number of milliseconds in a cycle of flash and normal display.	Syntax:	getFlashCycle() : int
	Post:	Result = the number of milliseconds in a cycle of flash and normal display.
Query the total number of milliseconds of flashing desired.	Syntax:	getFlashPeriod() : int
	Post:	Result = the total number of milliseconds of flashing desired.
Update the flash parameters with consistent values.	Syntax:	setFlashParameters(flashPeriod : int, flashCycle : int) : void
	Post:	Record values of the flash parameters consistent with each other.

Table 8 UIColors Interface Specification

Query the color of an InstanceGoalUI for the combination of GoalState, FlashContext, and ImageContext.	Syntax:	getColor(state : GoalState, flashContext : FlashContext, imageContext : ImageContext) : Color
	Post:	Result = the color of an InstanceGoalUI for the combination of GoalState, FlashContext, and ImageContext.
Set the color of an InstanceGoalUI for the combination of GoalState, FlashContext, and ImageContext.	Syntax:	setColor(state : GoalState, flashContext : FlashContext, imageContext : ImageContext, color : Color) : void
	Post:	Record the color of an InstanceGoalUI for the combination of GoalState, FlashContext, and ImageContext.

4.2.1.3 GMoDS Visualizer Model Design Rationale

I designed ModifiableInstanceParameters starting with GMoDS' InstanceParameters class to make it easy to incorporate support for the "MODIFICATION" parameter value origin directly into GMoDS, if desired.

4.2.2 GMoDS Visualizer Model Behavior

Figure 13 below shows the GMoDSModelImpl.notifyInstanceGoalModified method of the ChangeManager interface. This method records the new values of the instance parameters to add support for the "MODIFICATION" parameter value origin by calling the updateInstanceGoal method (see Figure 14). It then notifies observers that the model has changed. The observer initiates flashing of the affected InstanceGoalUI. The notifyInstanceGoalModified method is an example of how all the other instance tree-related ChangeManager methods notify the observers.

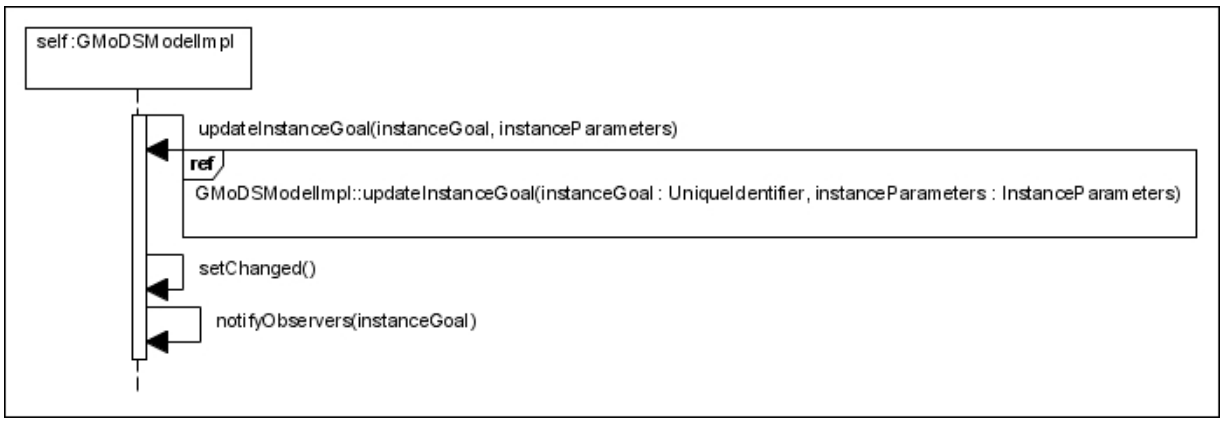

Figure 13 GMoDSModelImpl.notifyInstanceGoalModified(instanceGoal : UniqueIdentifier, instanceParameters : InstanceParameters)

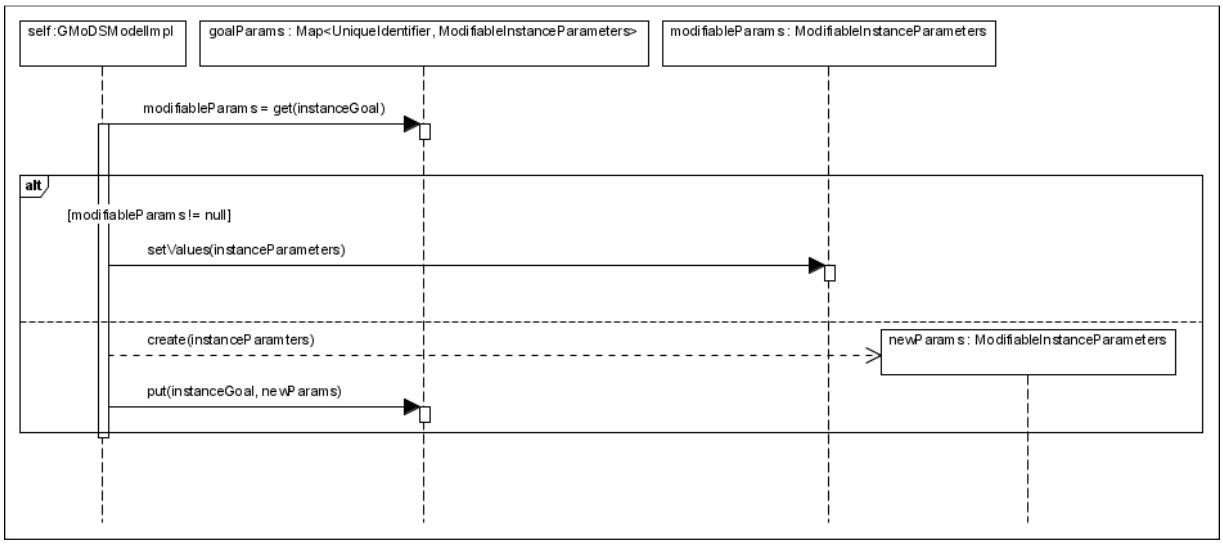

Figure 14 GMoDSModelImpl.updateInstanceGoal(instanceGoal : UniqueIdentifier, instanceParameters : InstanceParameters)

4.2.3 GMoDS Visualizer View Static Structure

Figure 15 below shows the architecture of the GMoDS Visualizer view package described in detail in [2]. The EditPreferenceUI has been added as a new view not shown in the component class diagrams focused on the specification and instance tree views below.

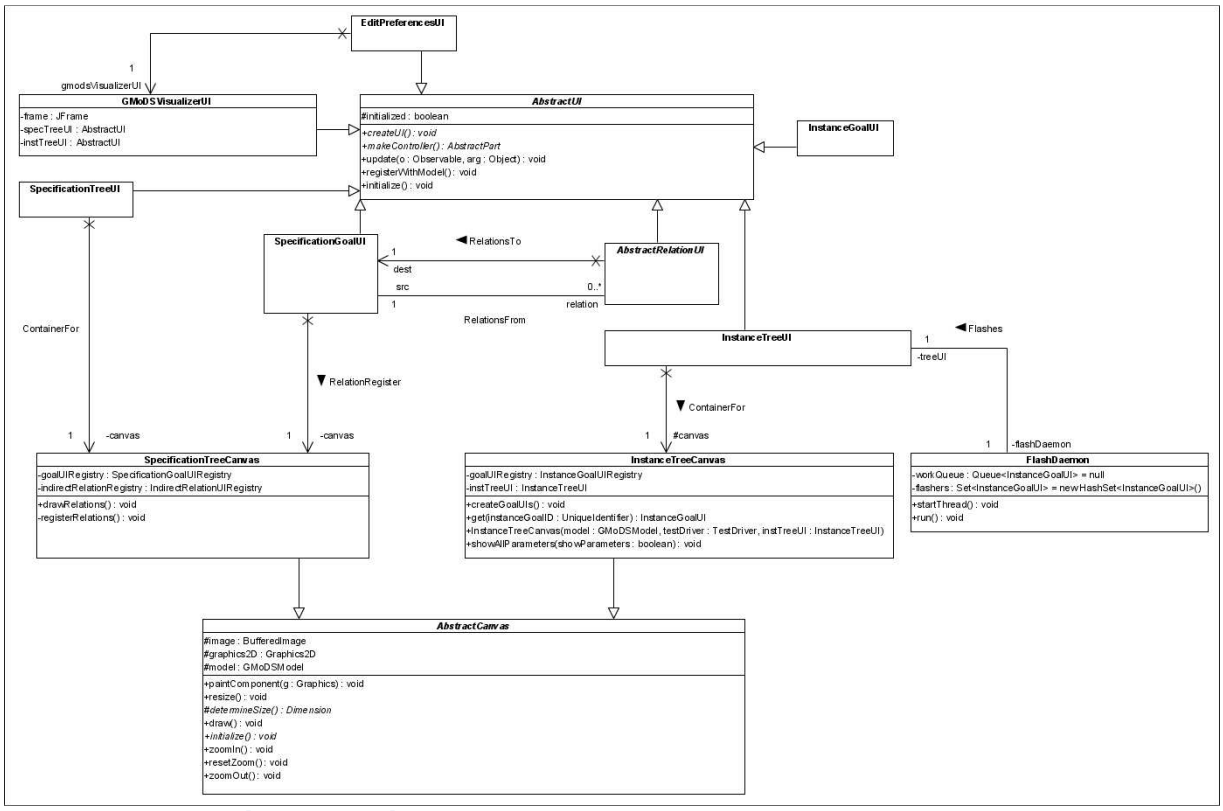

Figure 15 GMoDS Visualizer View Architecture

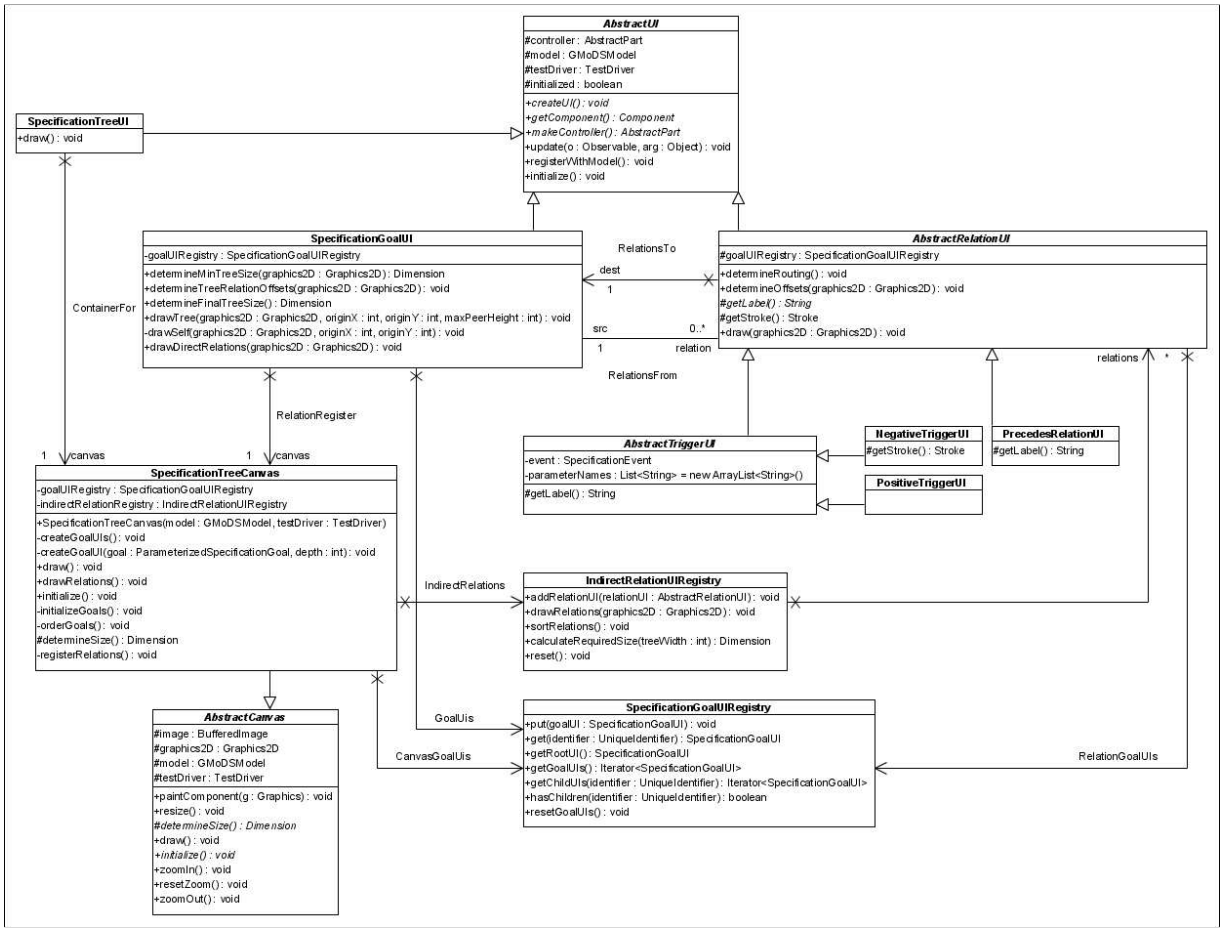

Figure 16 GMoDS Visualizer Specification Tree View Component Classes

Figure 16 above shows the component design of the specification tree view. Figure 17 below shows the component design of the instance tree view.

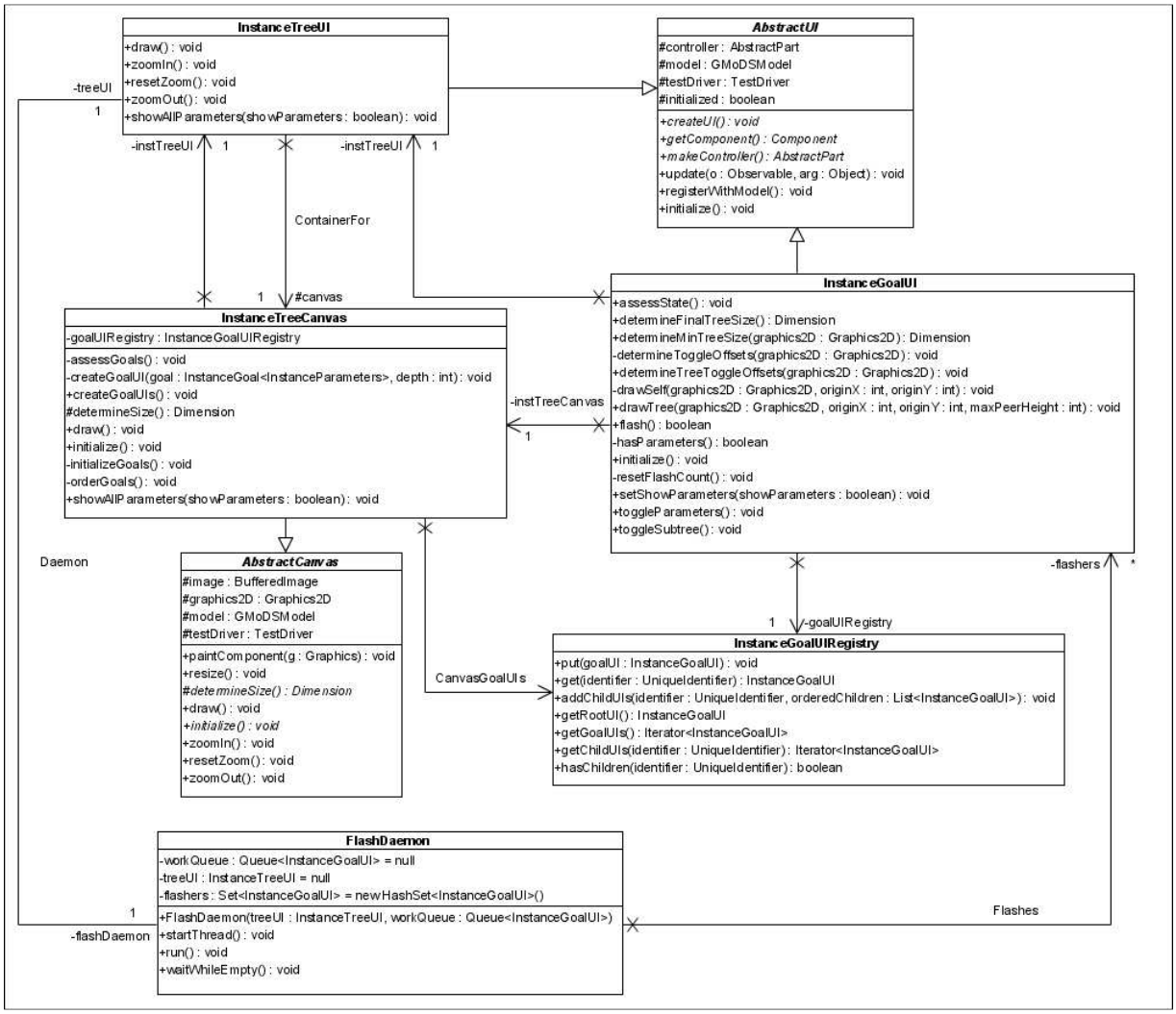

Figure 17 GMoDS Visualizer InstanceTreeUI Component Classes

4.2.3.1 GMoDS Visualizer View Local Module Responsibilities

Table 9 GMoDS Visualizer View Module Responsibilities

Component	Responsibilities
EditPreferencesUI	Provide the view for editing preferences.
SpecificationTreeUI	Provide the view for the specification tree.
SpecificationTreeCanvas	Draw the specification tree on an image.
SpecificationGoalUI	Provide the view for a specification goal.
AbstractRelationUI	Provide the view for relation UIs.
AbstractTriggerUI	Provide the view for trigger UIs.
PrecedesRelationUI	Provide the view for a “precedes” relation.

Component	Responsibilities
PositiveTriggerUI	Provide the view for a positive trigger.
NegativeTriggerUI	Provide the view for a negative trigger.
SpecificationGoalUIRegistry	Record and provide access to the view of each specification goal.
IndirectRelationUIRegistry	Record and manage the drawing of indirectly routed relation views.
InstanceTreeUI	Provide the view for the instance tree.
InstanceTreeCanvas	Draw the instance tree on an image.
InstanceGoalUI	Provide the view for an instance goal.
InstanceGoalUIRegistry	Record and provide access to the view of each instance goal.
FlashDaemon	Flash each changed InstanceGoalUI.

4.2.3.2 GMoDS Visualizer View Local Module Interface Specifications

Table 10 SpecificationTreeUI Interface Specifications

Draw the specification tree on the canvas.	Syntax:	draw() : void
	Pre:	none
	Post:	The specification tree is drawn on the canvas.

Table 11 SpecificationTreeCanvas Interface Specifications

Assure that all instance goals in the instance tree have views.	Syntax:	createGoalUIs() : void
	Pre:	none
	Post:	All instance goals in the instance tree have views.
Create the view for a particular specification goal.	Syntax:	createGoalUI(goal : ParameterizedSpecificationGoal, depth : int) : void
	Pre:	goal != null
	Post:	The specification goal has a view created and recorded.

Draw the specification tree on the canvas.	Syntax:	draw() : void
	Pre:	none
	Post:	The specification tree is drawn on the canvas.
Draw all relations.	Syntax:	drawRelations() : void
	Pre:	none
	Post:	All relations are drawn.
Initialize the canvas.	Syntax:	initialize() : void
	Pre:	none
	Post:	The canvas is initialized.
Initialize all specification goal views.	Syntax:	initializeGoals() : void
	Pre:	none
	Post:	All specification goal views are initialized.
Order the goals at each level of the specification tree to draw relations from left to right.	Syntax:	orderGoals() : void
	Pre:	none
	Post:	The goals at each level of the specification tree are ordered so relations can be drawn from left to right.
Determine the dimensions of the specification tree image.	Syntax:	determineSize() : Dimension
	Pre:	none
	Post:	Result – the total size of the specification tree image is returned.
Register and sort all relations.	Syntax:	registerRelations() : void
	Pre:	none
	Post:	All relations are registered with the appropriate views and registries and are sorted for drawing order.

Table 12 SpecificationGoalUI Interface Specifications

Determine the minimum size of the specification tree image.	Syntax:	determineMinTreeSize(graphics2D : Graphics2D) : Dimension
	Pre:	graphics2D != null
	Post:	Result = the minimum dimensions of the specification tree image.
Determine the horizontal offsets required to provide space for relations in the tree.	Syntax:	determineTreeRelationOffsets(graphics2D : Graphics2D) : void
	Pre:	graphics2D != null
	Post:	Each SpecificationGoalUI has recorded its required horizontal offset.
Determine the final overall size of the specification tree image.	Syntax:	determineFinalTreeSize() : Dimension
	Pre:	none
	Post:	Result = the final dimensions of the specification tree image.
Draw the specification tree rooted at this SpecificationGoalUI on the canvas.	Syntax:	drawTree(graphics2D : Graphics2D, originX : int, originY : int, maxPeerHeight : int) : void
	Pre:	graphics2D != null
	Post:	The specification tree rooted at this SpecificationGoalUI is drawn on the canvas.
Draw the SpecificationGoalUI on the canvas.	Syntax:	drawSelf(graphics2D : Graphics2D, originX : int, originY : int) : void
	Pre:	graphics2D != null
	Post:	The SpecificationGoalUI is drawn on the canvas.
Draw the directly-routed relations emanating from the SpecificationGoalUI on the canvas.	Syntax:	drawDirectRelations(graphics2D : Graphics2D) : void
	Pre:	graphics2D != null
	Post:	The directly-routed relations emanating from the SpecificationGoalUI are drawn on the canvas.

Table 13 AbstractRelationUI Interface Specifications

Determine whether the relation will be directly or indirectly routed.	Syntax:	determineRouting() : void
	Pre:	none
	Post:	The AbstractRelationUI has recorded whether the relation will be directly or indirectly routed.
Determine the horizontal offset required for the destination SpecificationGoalUI.	Syntax:	determineOffsets(graphics2D : Graphics2D) : void
	Pre:	graphics2D != null
	Post:	The destination SpecificationGoalUI has recorded the horizontal offset required for this AbstractRelationUI.
Query the required label for the relation.	Syntax:	getLabel() : String
	Pre:	none
	Post:	Result = the required label for the relation.
Draw the relation on the canvas.	Syntax:	draw(graphics2D : Graphics2D) : void
	Pre:	graphics2D != null
	Post:	The relation is drawn on the canvas.

Table 14 SpecificationGoalUIRegistry Interface Specifications

Record a SpecificationGoalUI.	Syntax:	put(goalUI : SpecificationGoalUI) : void
	Pre:	goalUI != null
	Post:	Recorded the SpecificationGoalUI.
Access a SpecificationGoalUI.	Syntax:	get(identifier : UniqueIdentifier) : SpecificationGoalUI
	Pre:	identifier != null
	Post:	Result = the SpecificationGoalUI.
Access the root SpecificationGoalUI.	Syntax:	getRootUI() : SpecificationGoalUI
	Pre:	none
	Post:	Result = the root SpecificationGoalUI.

Access all SpecificationGoalUIs.	Syntax:	getGoalUIs() : Iterator<SpecificationGoalUI>
	Pre:	none
	Post:	Result = all SpecificationGoalUIs.
Access all children UIs of the specified SpecificationGoalUI.	Syntax:	getChildUIs(identifier : UniqueIdentifier) : Iterator<SpecificationGoalUI>
	Pre:	identifier != null
	Post:	Result = all children UIs of the specified SpecificationGoalUI.
Query whether a SpecificationGoalUI has children.	Syntax:	hasChildren(identifier : UniqueIdentifier) : boolean
	Pre:	identifier != null
	Post:	Result = true if the SpecificationGoalUI has children; false, otherwise.
Reset all SpecificationGoalUIs data structures that support drawing.	Syntax:	resetGoalUIs() : void
	Pre:	none
	Post:	All SpecificationGoalUIs data structures that support drawing are reset to their default values.

Table 15 IndirectRelationUIRegistry Interface Specifications

Record an indirectly-routed AbstractRelationUI.	Syntax:	addRelationUI(relationUI : AbstractRelationUI) : void
	Pre:	relationUI != null
	Post:	The indirectly-routed AbstractRelationUI is recorded.
Draw the all indirectly-routed AbstractRelationUIs on the canvas.	Syntax:	drawRelations(graphics2D : Graphics2D) : void
	Pre:	graphics2D != null
	Post:	All indirectly-routed AbstractRelationUIs are drawn on the canvas.

Sort all indirectly-routed AbstractRelationUIs into drawing order.	Syntax:	sortRelations() : void
	Pre:	none
	Post:	All indirectly-routed AbstractRelationUIs are sorted into drawing order.
Calculate the size required for indirectly-routed relations below the specification tree.	Syntax:	calculateRequiredSize(width : int) : Dimension
	Pre:	none
	Post:	Result = the size required for indirectly-routed relations below the specification tree.
Reset indirectly-routed AbstractRelationUIs data structures that support drawing.	Syntax:	reset() : void
	Pre:	none
	Post:	All indirectly-routed AbstractRelationUIs data structures that support drawing are reset to their default values.

Table 16 InstanceTreeUI

Draw the instance tree on the canvas.	Syntax:	draw() : void
	Pre:	none
	Post:	The instance tree is drawn on the canvas.
Record whether all parameters should be shown in the instance tree.	Syntax:	showAllParameters(showParameters : boolean) : void
	Pre:	none
	Post:	Recorded whether all parameters should be shown in the instance tree.

Table 17 InstanceTreeCanvas Interface Specifications

Assess the GoalState of all InstanceGoalUIs.	Syntax:	assessGoals() : void
	Pre:	none
	Post:	The GoalState of each InstanceGoalUI is recorded.

Create an InstanceGoalUI.	Syntax:	createGoalUI(goal : InstanceGoal<InstanceParameters>, depth : int) : void
	Pre:	goal != null
	Post:	An InstanceGoalUI is created for goal and is recorded in the InstanceGoalUIRegistry.
Create all InstanceGoalUIs if they don't exist already.	Syntax:	createGoalUIs() : void
	Pre:	none
	Post:	An InstanceGoalUI is created for each goal in GMoDS if it does not already exist and is recorded in the InstanceGoalUIRegistry.
Determine the dimensions of the instance tree image.	Syntax:	determineSize() : Dimension
	Pre:	none
	Post:	Result – the total size of the instance tree image is returned.
Draw the instance tree on the canvas.	Syntax:	draw() : void
	Pre:	none
	Post:	The instance tree is drawn on the canvas.
Initialize the canvas.	Syntax:	initialize() : void
	Pre:	none
	Post:	The canvas is initialized.
Initialize all instance goal views.	Syntax:	initializeGoals() : void
	Pre:	none
	Post:	All specification goal views are initialized.
Order the goals at each level of the instance tree alphabetically.	Syntax:	orderGoals() : void
	Pre:	none
	Post:	The goals at each level of the instance tree are ordered alphabetically.

Record whether all parameters should be shown in the instance tree.	Syntax:	showAllParameters(showParameters : boolean) : void
	Pre:	none
	Post:	Recorded whether all parameters should be shown in the instance tree.

Table 18 InstanceGoalUI Interface Specifications

Assess the GoalState of the InstanceGoalUI.	Syntax:	assessState() : void
	Pre:	none
	Post:	The GoalState of the InstanceGoalUI is recorded.
Determine the final dimensions of the instance tree rooted at this InstanceGoalUI.	Syntax:	determineFinalTreeSize() : Dimension
	Pre:	none
	Post:	Result – the total size of the instance tree rooted at this InstanceGoalUI is returned.
Determine the minimum dimensions of the instance tree rooted at this InstanceGoalUI.	Syntax:	determineMinTreeSize() : Dimension
	Pre:	none
	Post:	Result – the minimum size of the instance tree rooted at this InstanceGoalUI is returned.
Determine the horizontal offset required to accommodate parameter toggles for the InstanceGoalUIs in the tree rooted at this InstanceGoalUI.	Syntax:	determineTreeToggleOffsets() : Dimension
	Pre:	none
	Post:	The horizontal offset required to accommodate parameter toggles for the InstanceGoalUIs in the tree rooted at this InstanceGoalUI are recorded with each InstanceGoalUI.
Draw this InstanceGoalUI on the canvas.	Syntax:	drawSelf(graphics2D : Graphics2D, originX : int, originY : int) : void
	Pre:	none
	Post:	This InstanceGoalUI is drawn on the canvas.

Draw the instance tree rooted at this InstanceGoalUI on the canvas.	Syntax:	drawTree(graphics2D : Graphics2D, originX : int, originY : int, maxPeerHeight : int) : void
	Pre:	none
	Post:	The instance tree rooted at this InstanceGoalUI is drawn on the canvas.
Invert the colors for this InstanceGoalUI to represent a flash and decrement the remaining flash count when the inversion has cycled back to normal.	Syntax:	flash() : boolean
	Pre:	none
	Post:	The color for this InstanceGoalUI is inverted and the remaining flash count is decremented when the inversion has cycled back to normal. Result – false if remaining flash count <= 0; true otherwise.
Query whether this InstanceGoalUI has parameters.	Syntax:	hasParameters() : boolean
	Pre:	none
	Post:	Result – true if this InstanceGoalUI has parameters; false, otherwise.
Initialize the InstanceGoalUI and recreate the labels.	Syntax:	initialize() : void
	Pre:	none
	Post:	The InstanceGoalUI is initialized and the labels are recreated.
Reset the flash count to the current total required by FlashParameters.	Syntax:	resetFlashCount() : void
	Pre:	none
	Post:	The remaining flash count is reset to the current total required by FlashParameters.
Record whether this InstanceGoalUI should show its parameters.	Syntax:	setShowParameters(showParameters : boolean) : void
	Pre:	none
	Post:	Recorded whether this InstanceGoalUI should show its parameters.

Toggle whether this InstanceGoalUI should show its parameters.	Syntax:	toggleParameters() : void
	Pre:	none
	Post:	Toggled whether this InstanceGoalUI should show its parameters.
Toggle whether this InstanceGoalUI should show its children.	Syntax:	toggleSubtree() : void
	Pre:	none
	Post:	Toggled whether this InstanceGoalUI should show its children.

Table 19 InstanceGoalUIRegistry

Record an InstanceGoalUI.	Syntax:	put(goalUI : InstanceGoalUI) : void
	Pre:	goalUI != null
	Post:	Recorded the InstanceGoalUI.
Access an InstanceGoalUI.	Syntax:	get(identifier : UniqueIdentifier) : InstanceGoalUI
	Pre:	identifier != null
	Post:	Result = the InstanceGoalUI.
Access the root InstanceGoalUI.	Syntax:	getRootUI() : InstanceGoalUI
	Pre:	none
	Post:	Result = the root InstanceGoalUI.
Access all InstanceGoalUIs.	Syntax:	getGoalUIs() : Iterator<InstanceGoalUI>
	Pre:	none
	Post:	Result = all InstanceGoalUIs.
Access all children UIs of the specified InstanceGoalUI.	Syntax:	getChildUIs(identifier : UniqueIdentifier) : Iterator<InstanceGoalUI>
	Pre:	identifier != null
	Post:	Result = all children UIs of the specified InstanceGoalUI.

Query whether an InstanceGoalUI has children.	Syntax:	hasChildren(identifier : UniqueIdentifier) : boolean
	Pre:	identifier != null
	Post:	Result = true if the InstanceGoalUI has children; false, otherwise.

Table 20 FlashDaemon Interface Specifications

Start the thread for the FlashDaemon.run method.	Syntax:	startThread() : void
	Pre:	none
	Post:	The thread for the FlashDaemon.run method is started.
Flash all changed InstanceGoalUIs.	Syntax:	run() : void
	Pre:	none
	Post:	Flashed all changed InstanceGoalUIs for the total times implied by FlashParameters at the time the goal changed.
Wait until a changed InstanceGoalUI is added to the daemon.	Syntax:	waitWhileEmpty() : void
	Pre:	none
	Post:	A changed InstanceGoalUI has been added to the daemon.

4.2.3.3 GMoDS Visualizer View Design Rationale

The Model-View-Controller architecture separates the business rules for interacting with the user away from the presentation of the interface. This will allow for maximum flexibility in designing new visual representations.

4.2.4 GMoDS Visualizer View Behavior

Figure 18 below shows the SpecificationTreeUI.initialize() method.

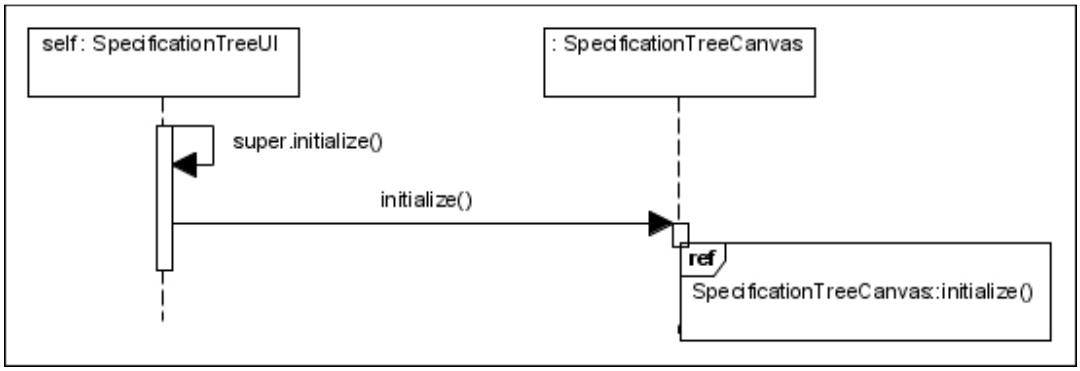

Figure 18 SpecificationTreeUI.initialize()

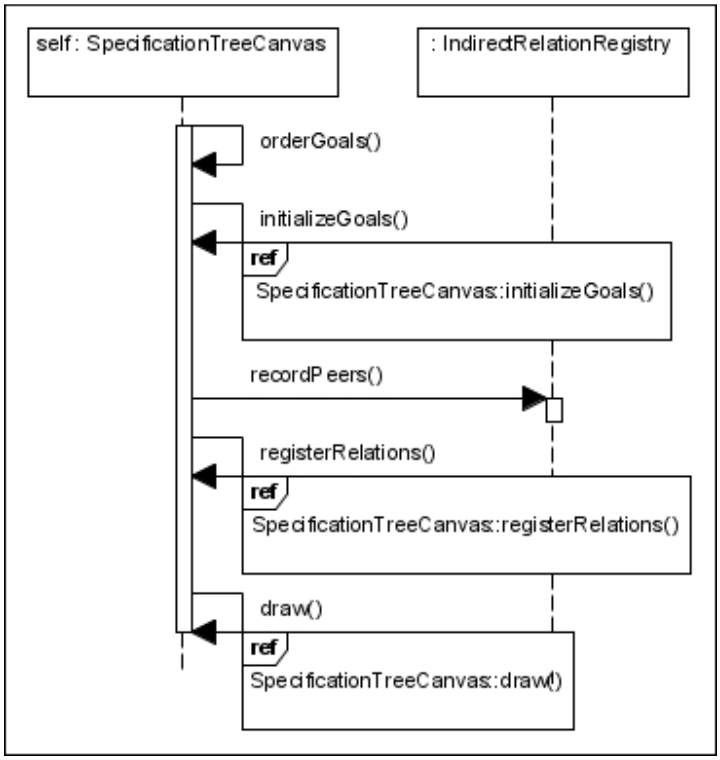

Figure 19 SpecificationTreeCanvas.initialize()

Figure 19 above shows the SpecificationTreeCanvas.initialize method. Figure 20 below shows the SpecificationTreeCanvas.initializeGoals method.

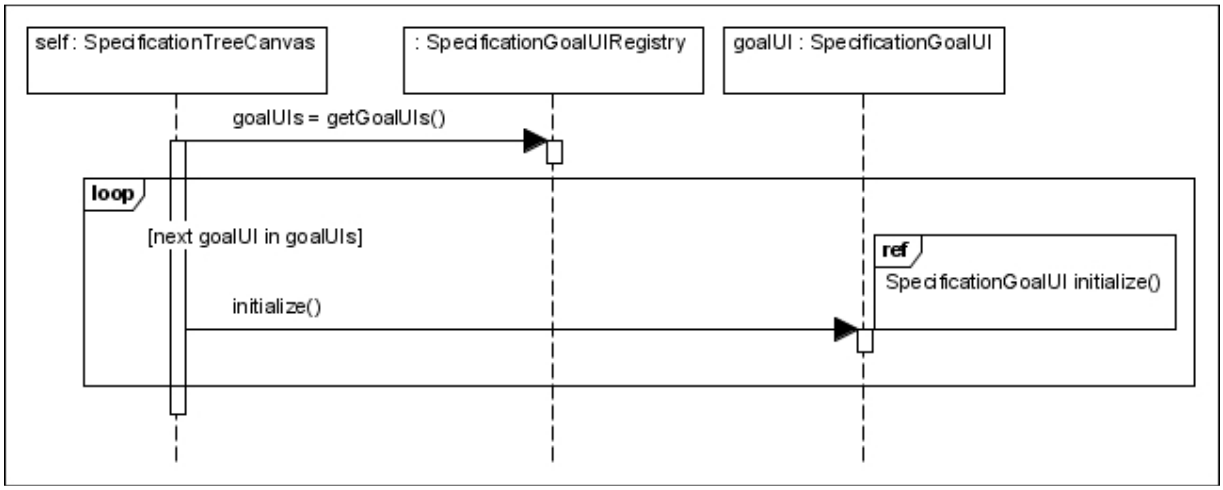

Figure 20 SpecificationTreeCanvas.initializeGoals()

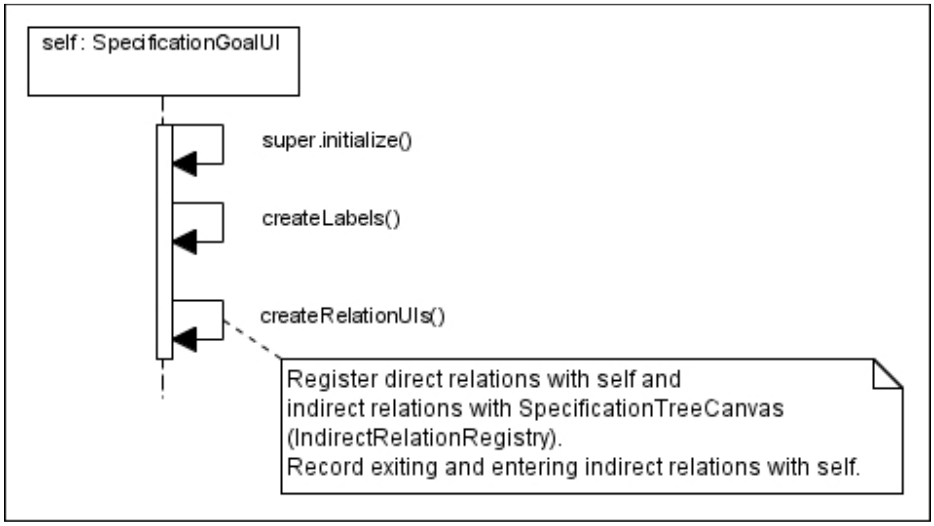

Figure 21 SpecificationGoalUI.initialize()

Figure 21 above shows the SpecificationGoalUI.initialize method. Figure 22 below shows the SpecificationTreeCanvas.registerRelations method. This method prepares for drawing relations by recording the relations with the object responsible for drawing them and sorting them in drawing order.

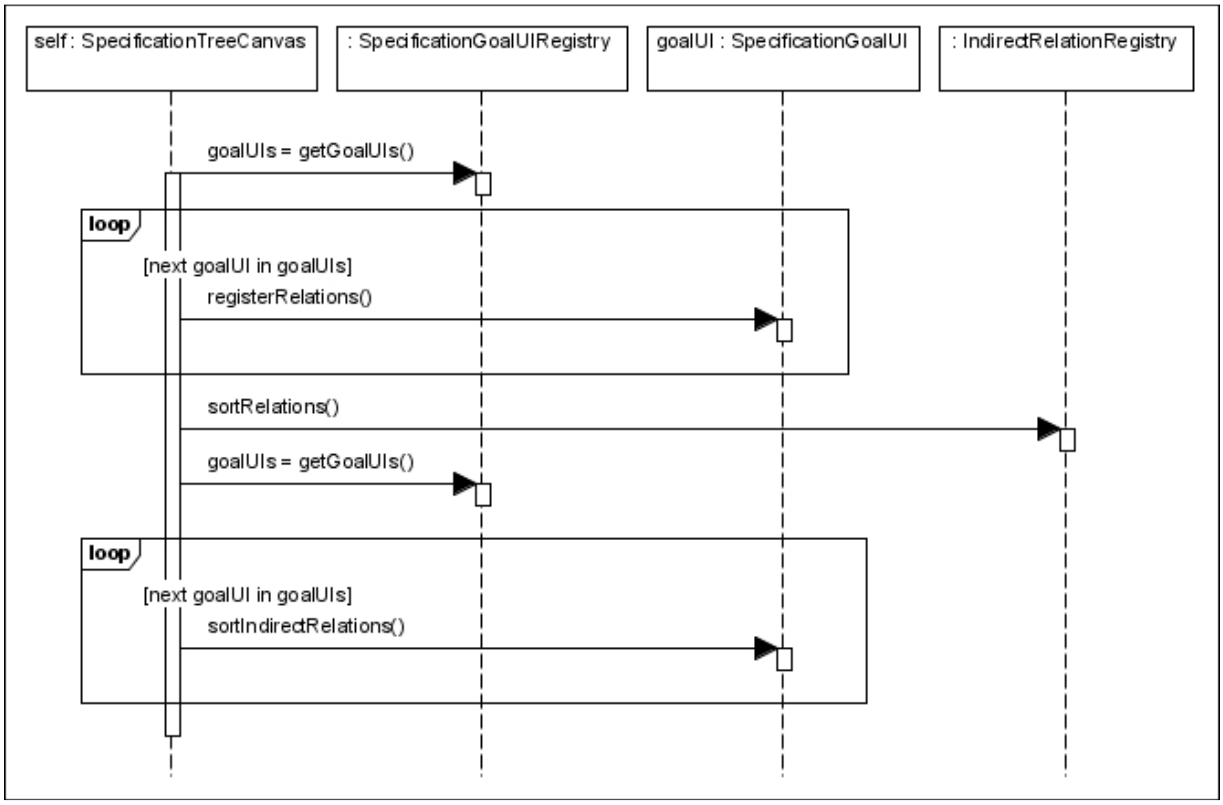

Figure 22 SpecificationTreeCanvas.registerRelations()

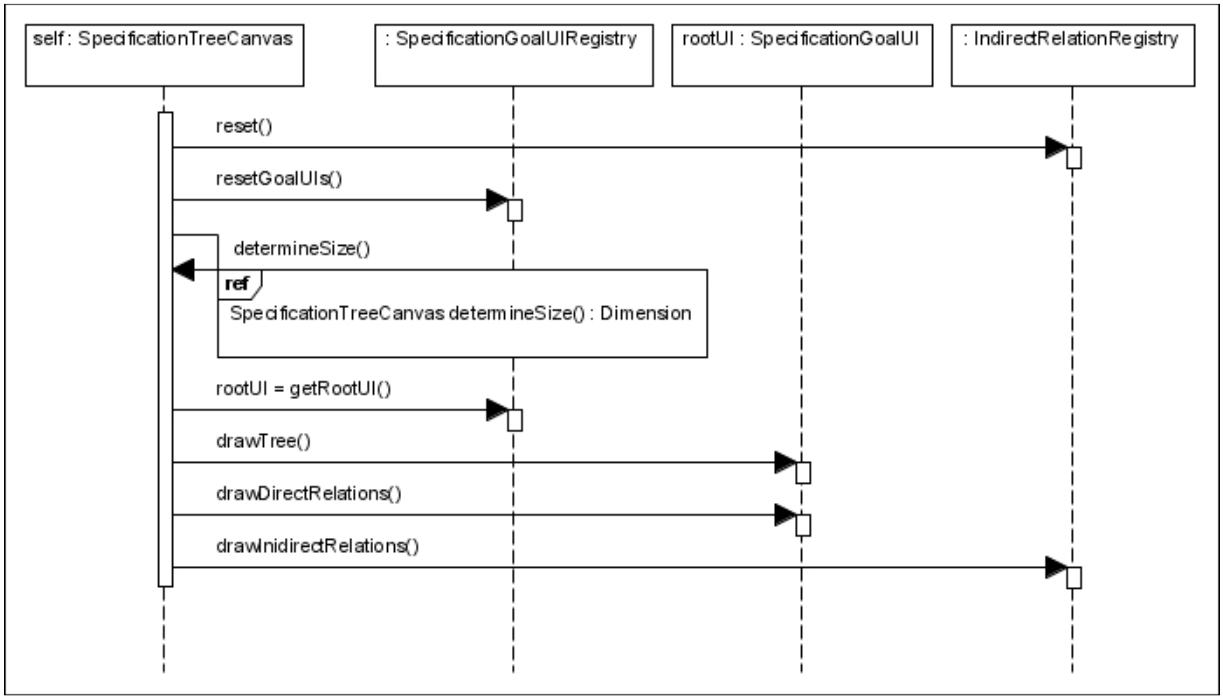

Figure 23 SpecificationTreeCanvas.draw()

Figure 23 above shows the SpecificationTreeCanvas.draw method. First, all SpecificationGoalUIs and AbstractRelationUIs data structures that are dynamically calculated

during drawing are reset to their default values. Next, the canvas is triggered to determine the total size of its image. Finally, the tree and the directly-routed and indirectly-routed relations are drawn. Figure 24 below shows the SpecificationTreeCanvas.determineSize method.

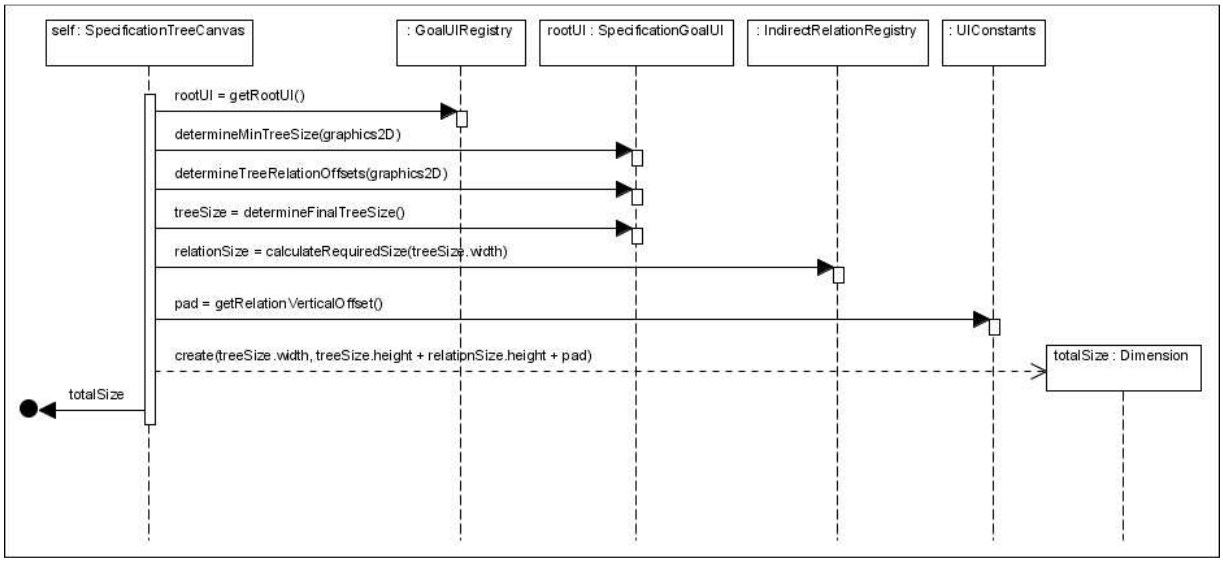

Figure 24 SpecificationTreeCanvas.determineSize()

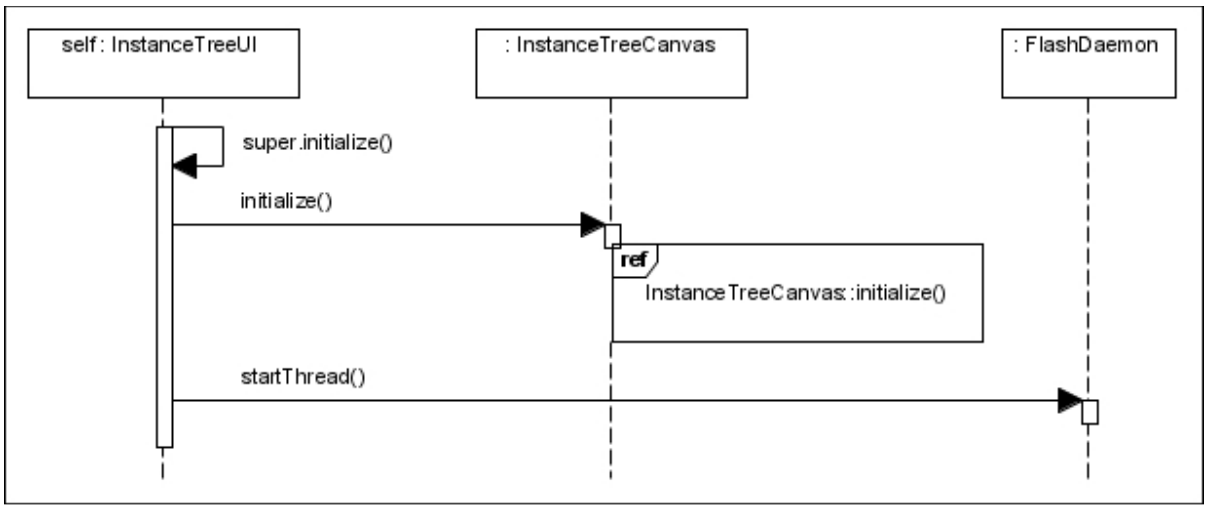

Figure 25 InstanceTreeUI.initialize()

Figure 25 above shows the InstanceTreeUI.initialize method. Figure 26 below shows the InstanceTreeCanvas.initialize method.

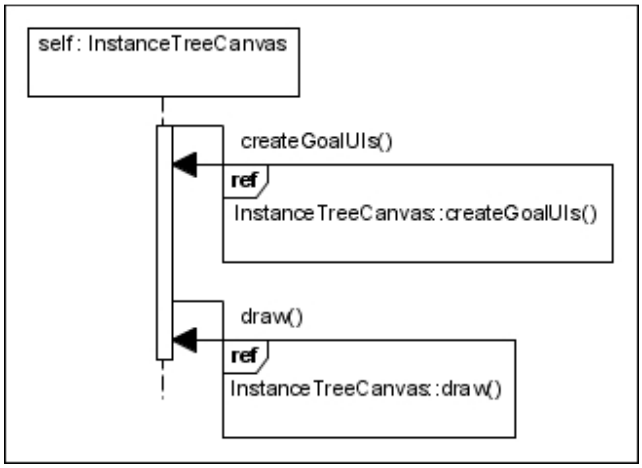

Figure 26 InstanceTreeCanvas.initialize()

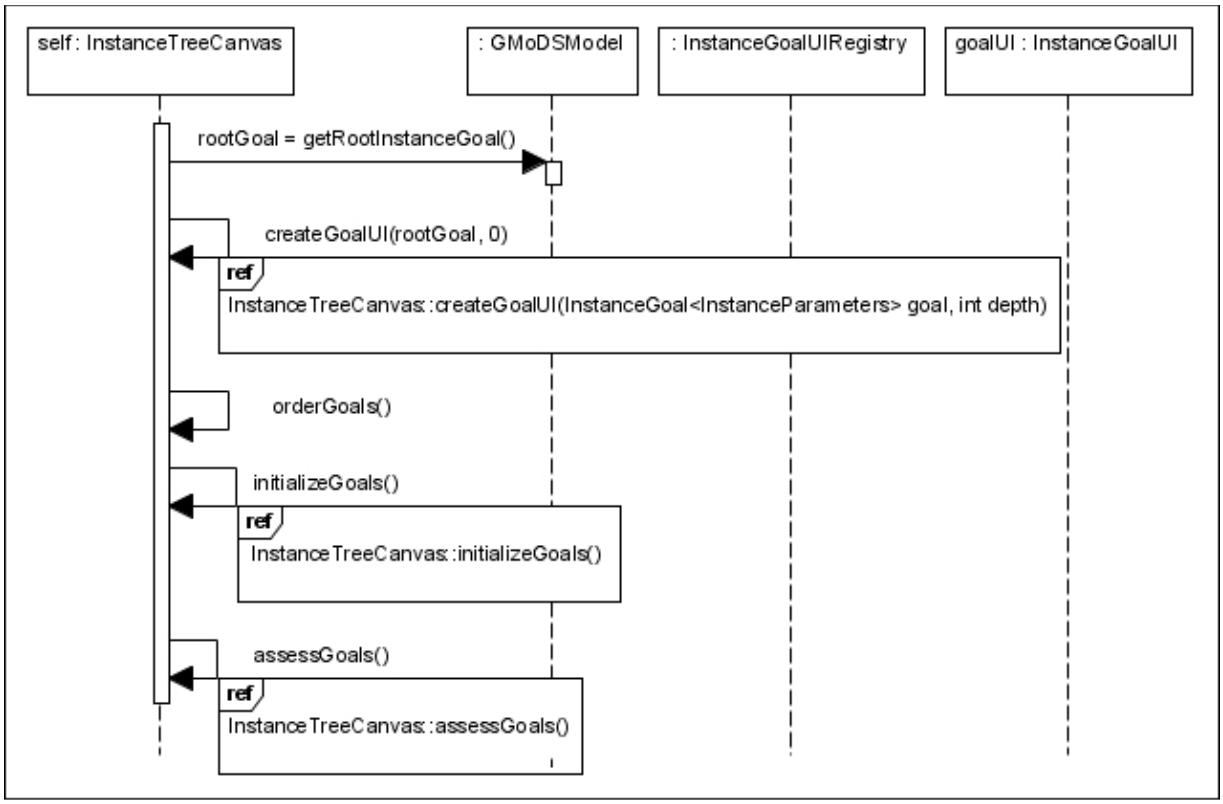

Figure 27 InstanceTreeCanvas.createGoalUIs()

Figure 27 above shows the InstanceTreeCanvas.createGoalUIs method. Using recursion, each an InstanceGoalUI is created for each InstanceGoal in GMoDS. The goal UIs are ordered to support drawing. Each goal UI is intitialized and its GoalState is assessed. Figure 28 below shows the InstanceTreeCanvas.createGoalUI method.

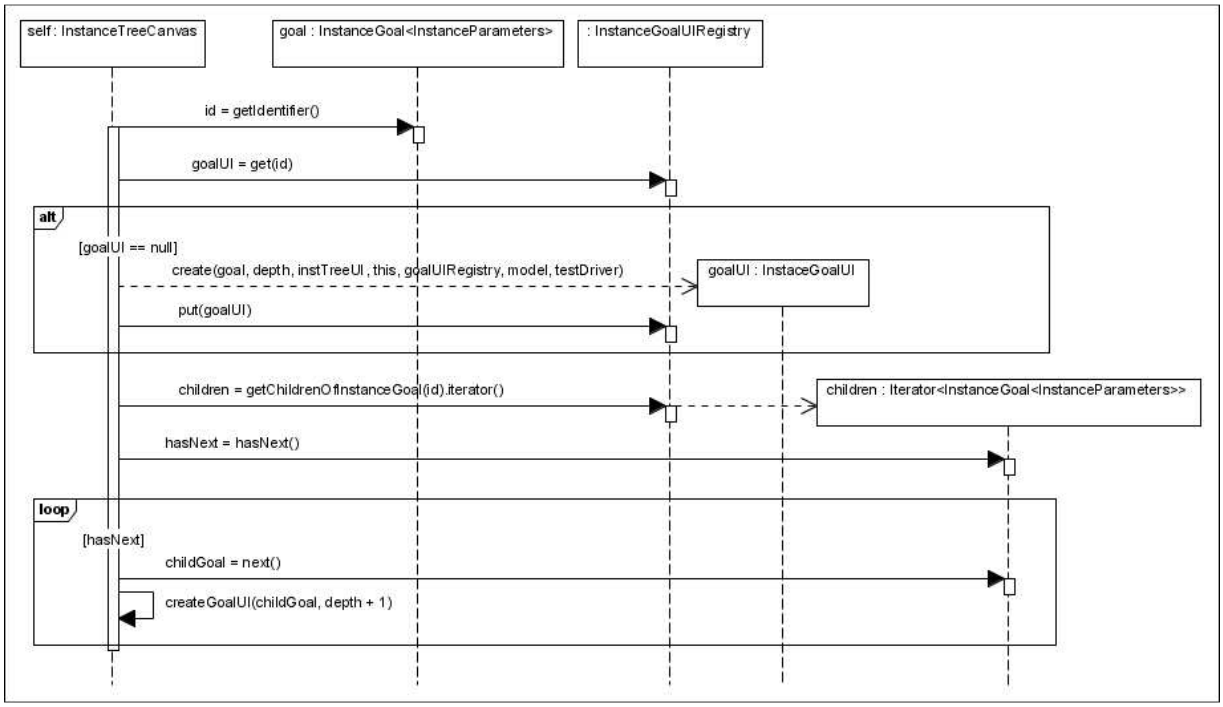

Figure 28 InstanceTreeCanvas.createGoalUI(goal : InstanceGoal<InstanceParameter> goal, depth :int)

Figure 29 below shows the InstanceTreeCanvas.initializeGoals method.

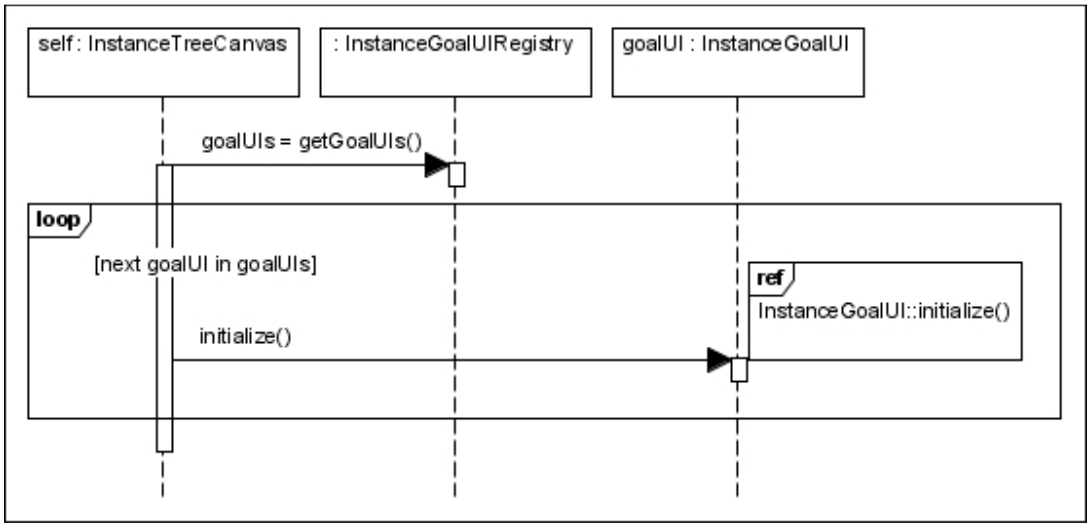

Figure 29 InstanceTreeCanvas.initializeGoals()

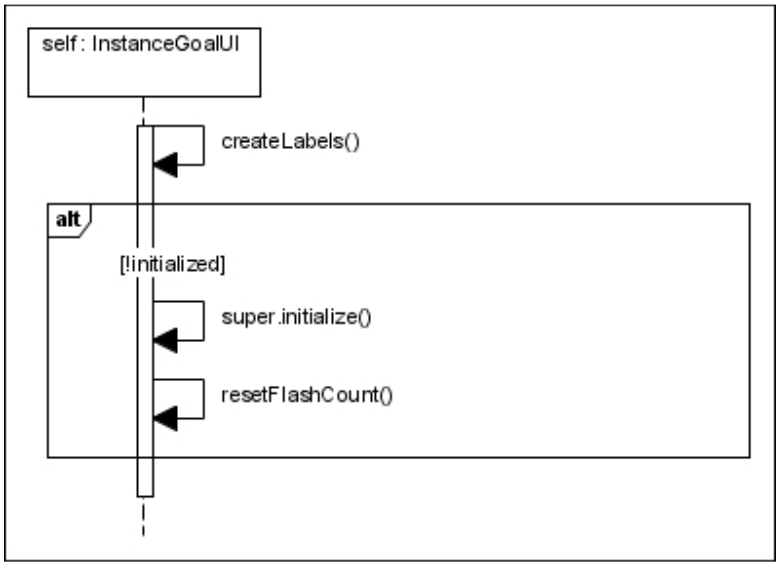

Figure 30 InstanceGoalUI.initialize()

Figure 30 above shows the InstanceGoalUI.initialize method. Figure 31 below shows the InstanceTreeCanvas.assessGoals method.

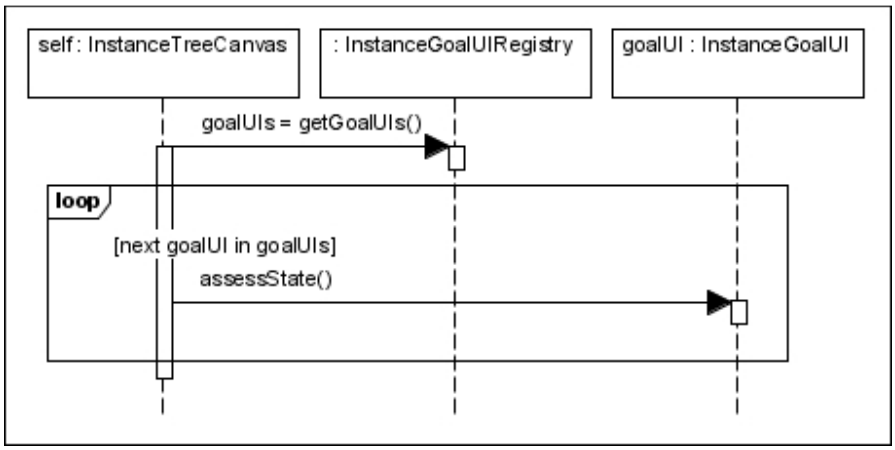

Figure 31 InstanceTreeCanvas.assessGoals()

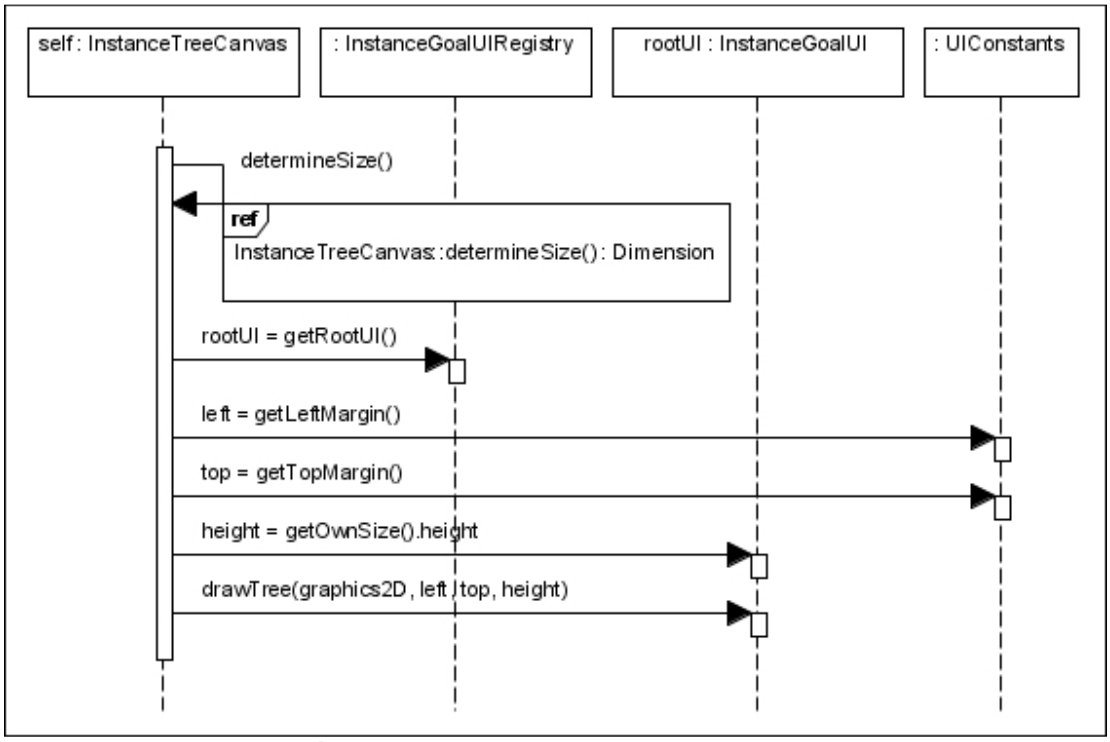

Figure 32 InstanceTreeCanvas.draw()

Figure 32 above shows the InstanceTreeCanvas.draw method. Figure 33 below shows the InstanceTreeCanvas.determineSize method.

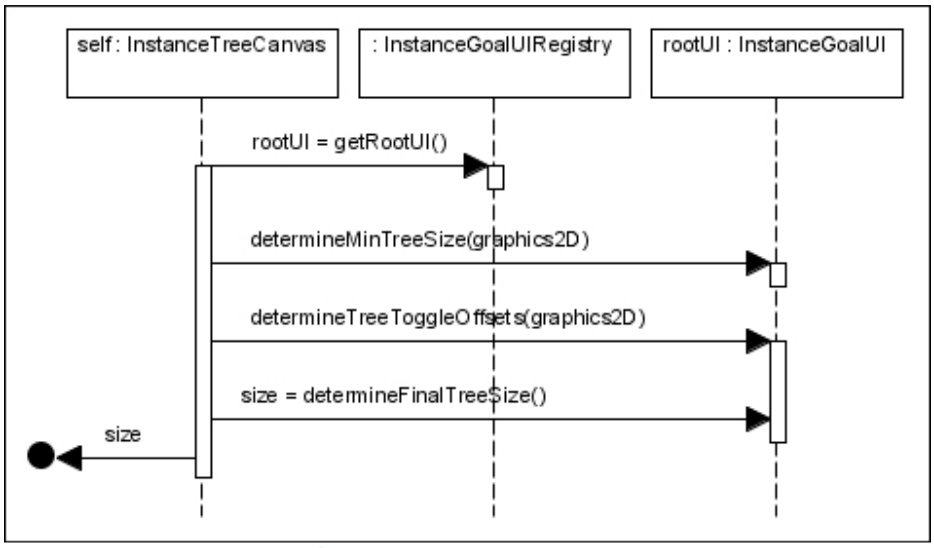

Figure 33 InstanceTreeCanvas.determineSize() : Dimension

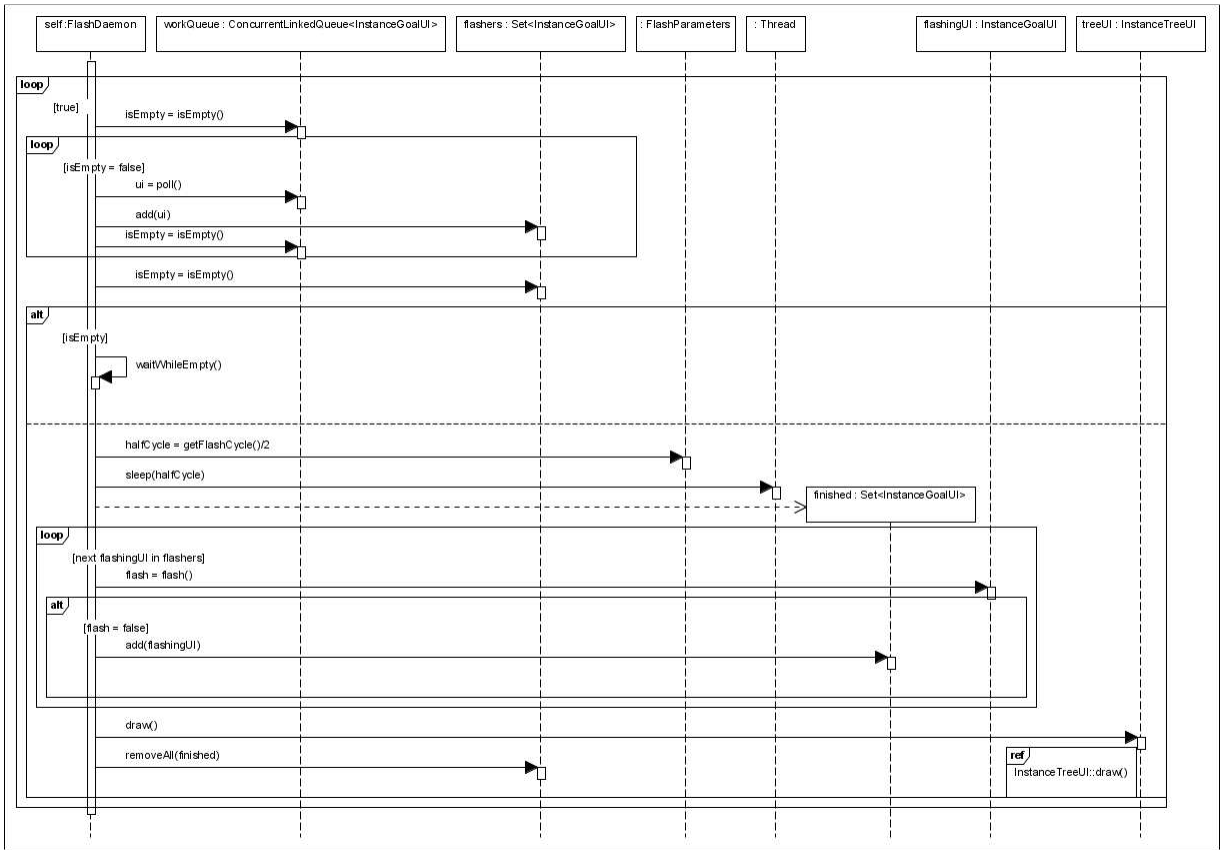

Figure 34 FlashDaemon.run()

Figure 34 above shows the FlashDaemon.run method. The daemon polls its queue for changed InstanceGoalUIs. If none are present, it waits. If at least one is present that has not finished flashing, it waits for a half flashing cycle and then toggles each changed InstanceGoalUI recording whether that UI is finished. Finally, the daemon asks the InstanceTreeUI to draw and then removes the finished UIs. This loop repeats indefinitely until the visualizer exits.

Figure 35 below shows the InstanceTreeUI.draw method.

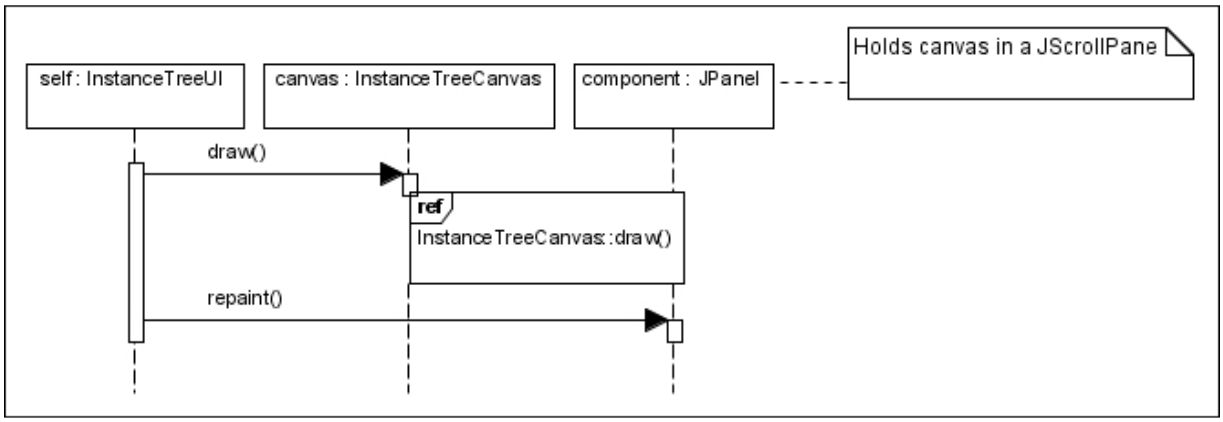

Figure 35 InstanceTreeUI.draw()

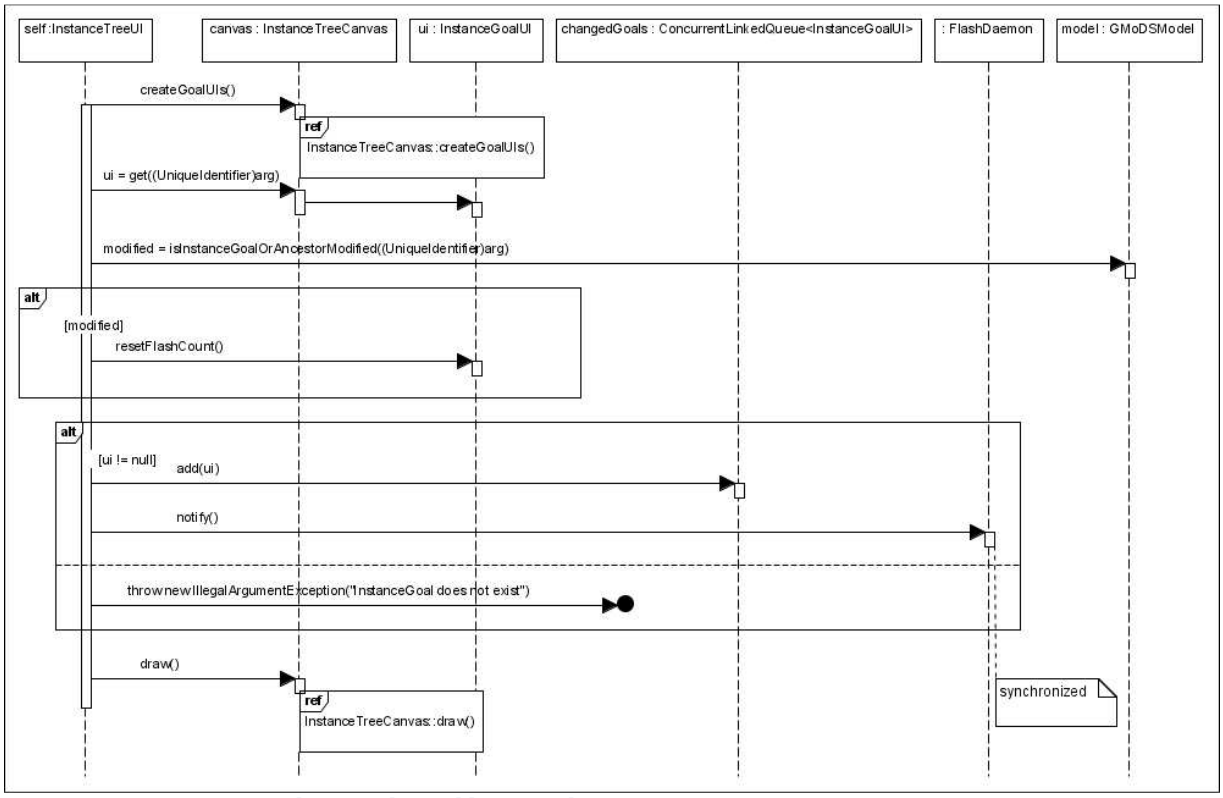

Figure 36 InstanceTreeUI.update(o : Observable, arg : Object)

Figure 36 above shows the InstanceTreeUI.update method. This method implements the observer design pattern on the GMoDSModel. The GMoDSModel implements the ChangeManager interface and notifies the InstanceTreeUI whenever an InstanceGoal has changed its GoalState or been modified. In response, the InstanceTreeUI assures that an InstanceGoalUI exists and is initialized for each InstanceGoal. Then the InstanceTreeUI notifies the FlashDaemon using a synchronized call to the notify method. Finally, the InstanceTreeCanvas is activated to draw the instance tree (and will be re-activated on each flash by the daemon).

4.2.5 GMoDS Visualizer Controller Static Structure

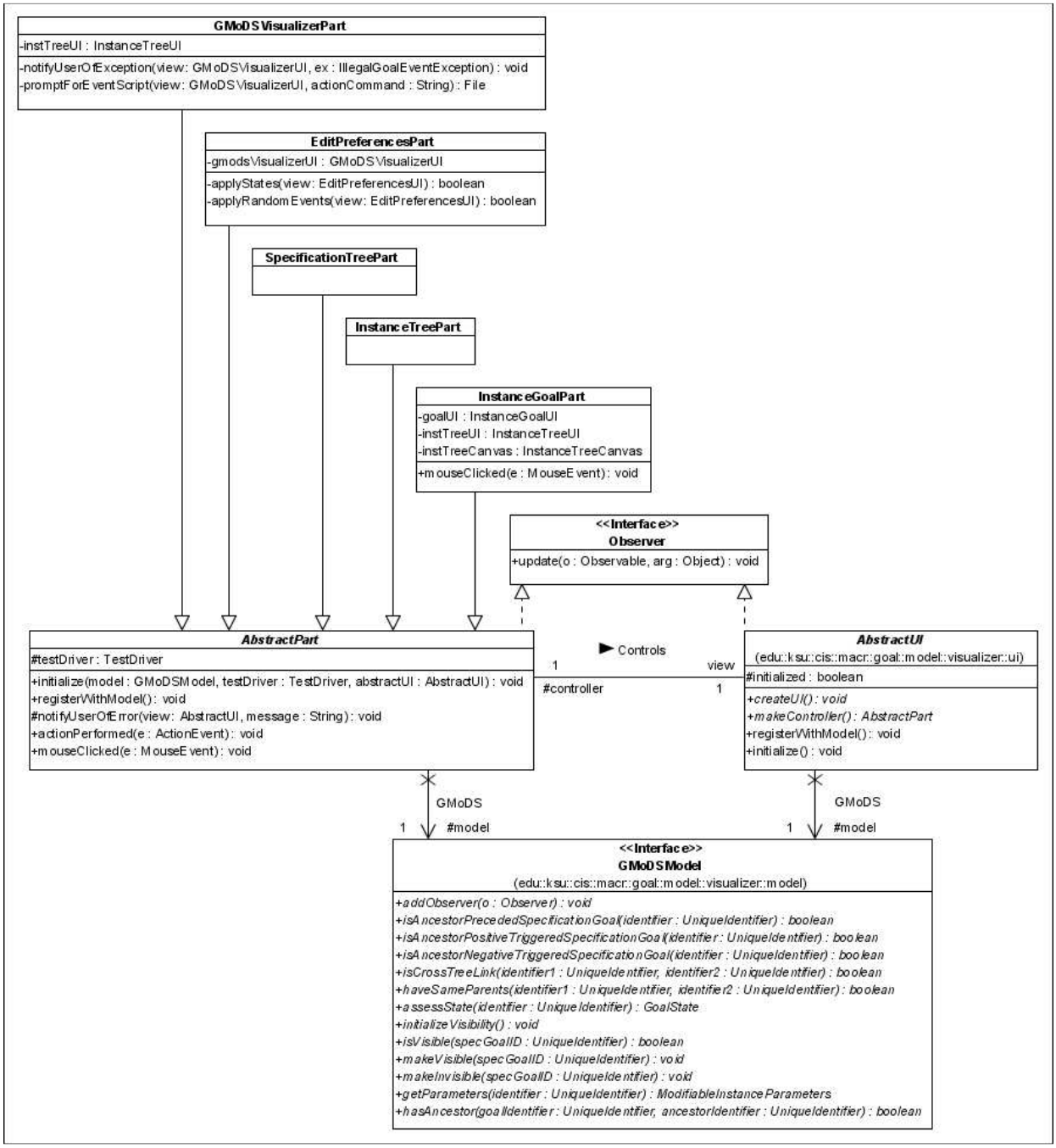

Figure 37 GMoDS Visualizer Controller Component Classes

4.2.5.1 GMoDS Visualizer Controller Local Module Responsibilities

Component	Responsibilities
GMoDSVisualizerPart	Control the main view and menu items.

Component	Responsibilities
EditPreferencesPart	Control the view for editing preferences.
SpecificationTreePart	Control the zooming of the view for the specification tree.
InstanceTreePart	Control the zooming of the view for the instance tree.
InstanceGoalPart	Control the view for a instance goal.

4.2.5.2 GMoDS Visualizer Controller Local Module Interface Specifications

Table 21 AbstractPart Interface Specifications

Respond to menu items and button clicks.	Syntax:	actionPerformed(e : ActionEvent) : void
	Pre:	none
	Post:	Necessary actions in response to menu items and button clicks have been performed.
Respond to mouse clicks.	Syntax:	mouseClicked(e : MouseEvent) : void
	Pre:	none
	Post:	Necessary actions in response to mouse clicks have been performed.

4.2.5.3 GMoDS Visualizer Controller Design Rationale

As described above, the Model-View-Controller architecture separates the business rules for interacting with the user away from the presentation of the interface, allowing for maximum flexibility. I used this flexibility to enforce constraints on the FlashParameters' flash cycle and period and RandomEventParameters' minimum and maximum delay time to assure that flashing will appear reasonable.

4.2.6 GMoDS Visualizer Controller Behavior

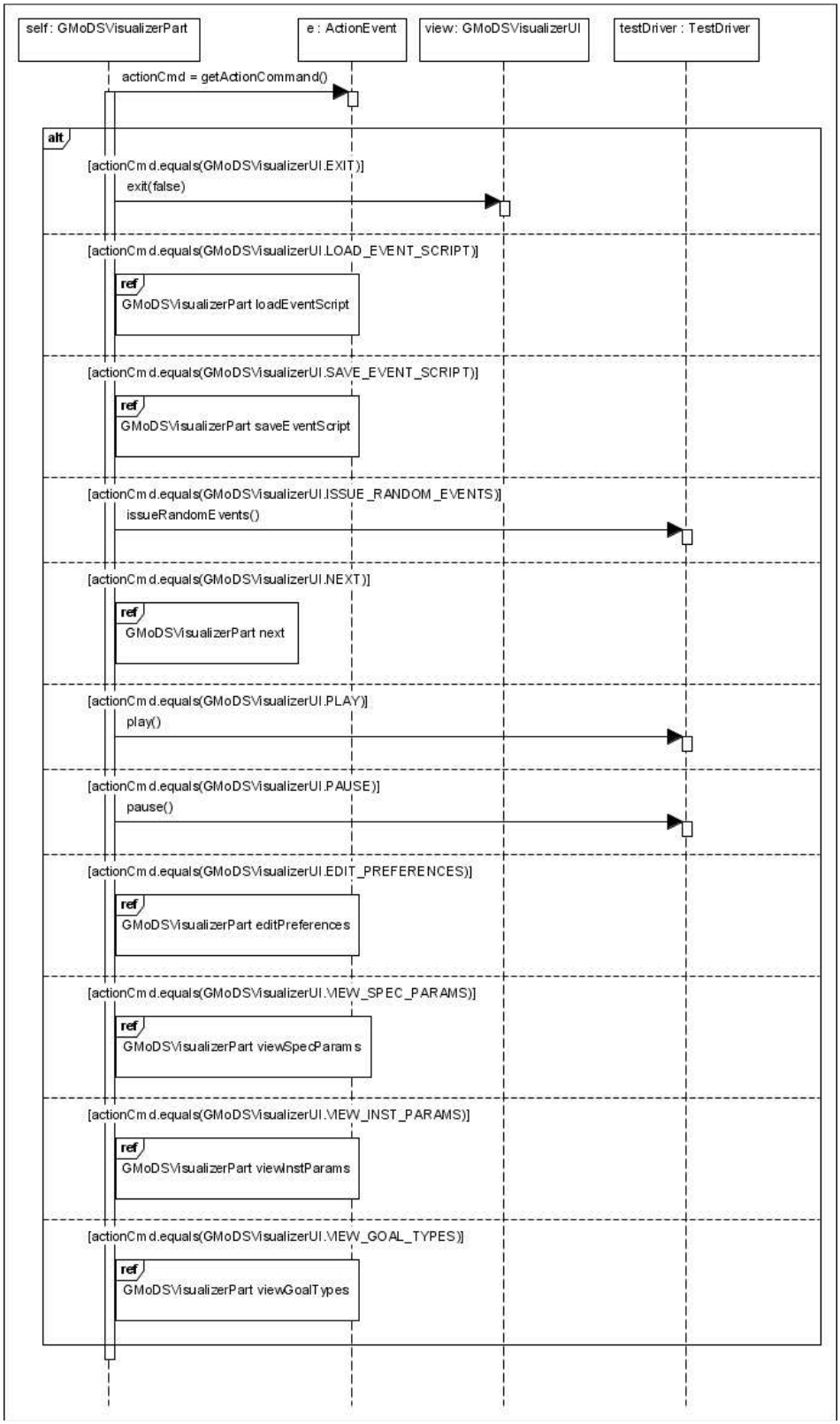

Figure 38 GMoDSVisualizerPart.actionPerformed(e : ActionEvent)

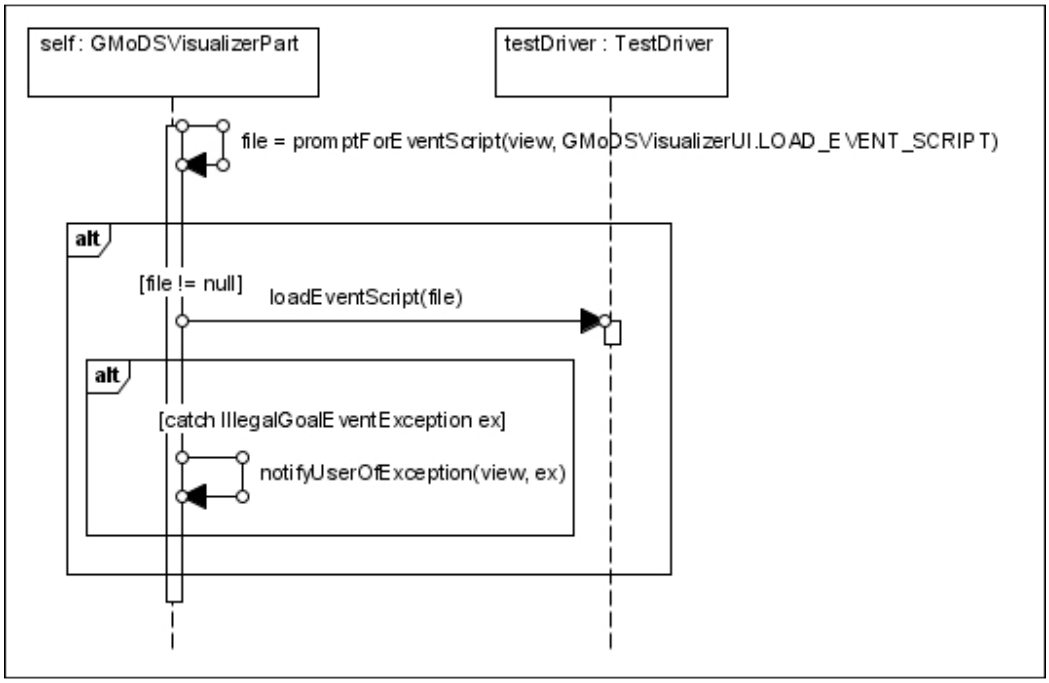

Figure 39 GMoDSVisualizerPart loadEventScript

Figure 38 above shows the GMoDSVisualizerPart.actionPerformed method. Figure 39 above shows the GMoDSVisualizerPart responding to the "load event script" command. Figure 40 below shows the GMoDSVisualizerPart responding to the "save event script" command.

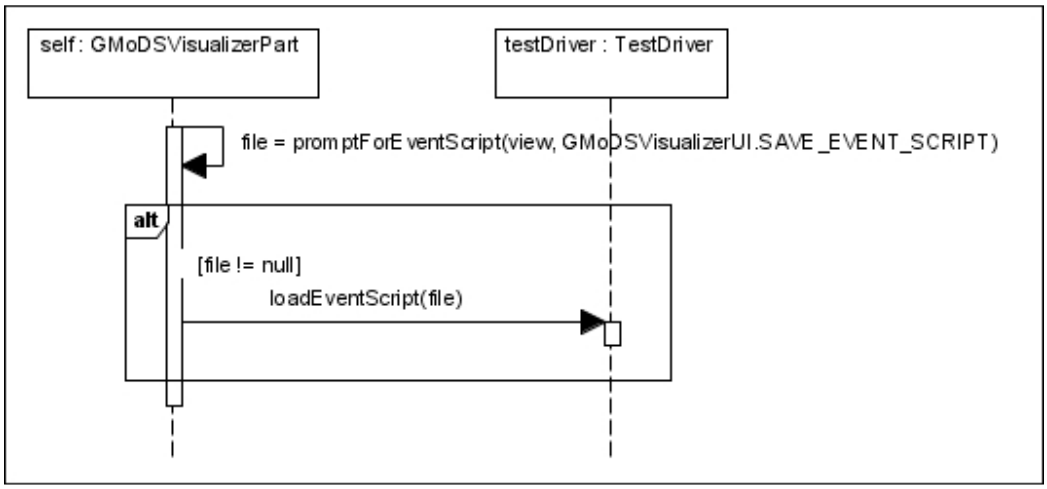

Figure 40 GMoDSVisualizerPart saveEventScript

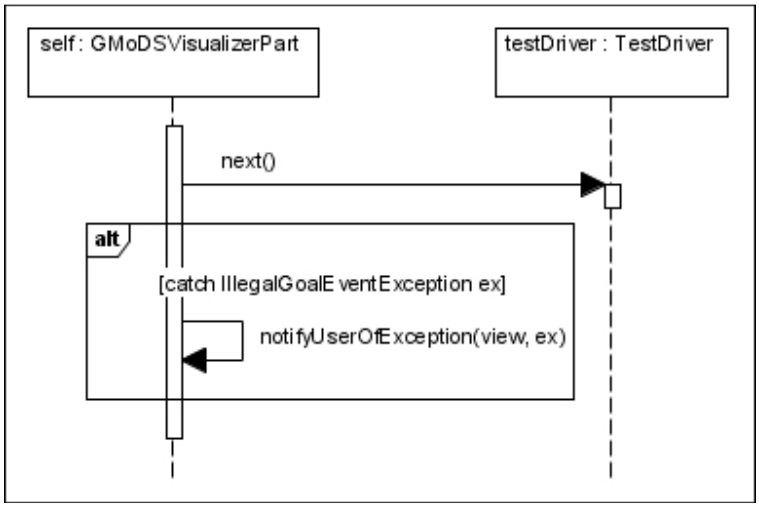

Figure 41 GMoDSVisualizer next

Figure 41 above shows the GMoDSVisualizerPart responding to the "next" command. It uses try/catch and catches IllegalGoalEventExceptions when a GoalEvent is illegal with respect to the instance tree. Figure 42 below shows the GMoDSVisualizerPart responding to the "Edit Preferences" command. Figure 43 below shows the GMoDSVisualizerPart responding to the "View | Specification Goals | Parameters" command.

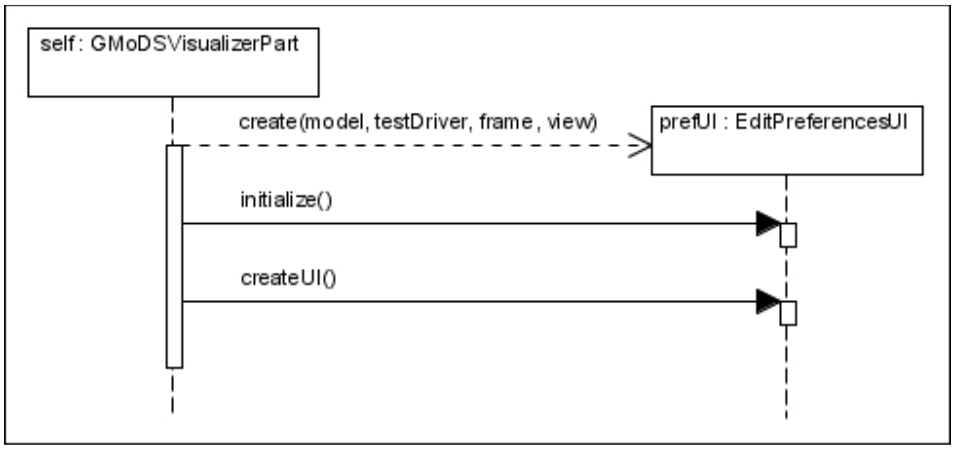

Figure 42 GMoDSVisualizerPart editPreferences

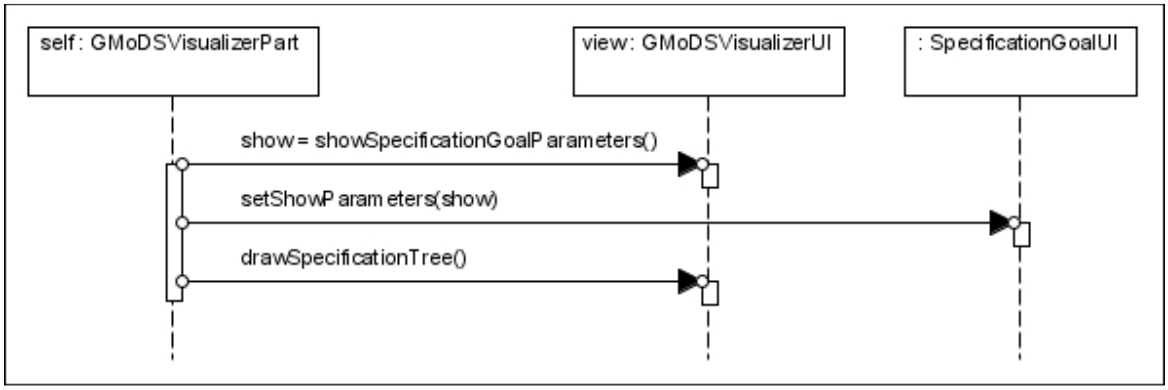

Figure 43 GMoDSVisualizerPart viewSpecParams

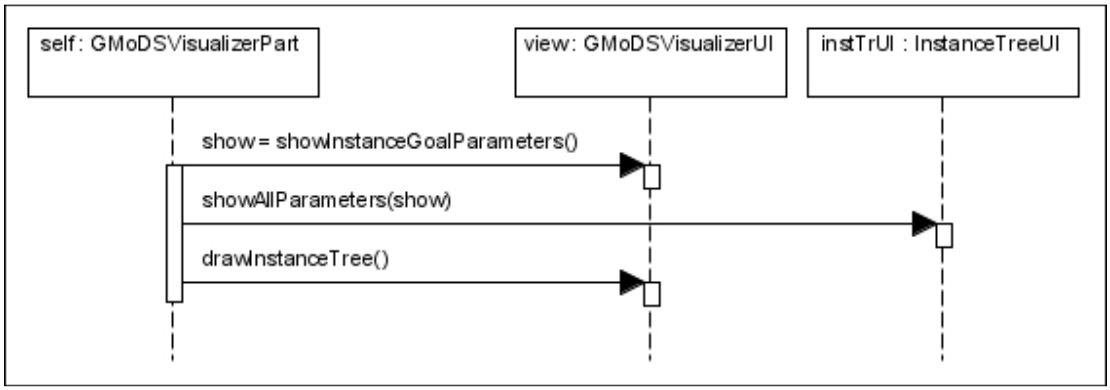

Figure 44 GMoDSVisualizerPart viewInstParams

Figure 44 above shows the GMoDSVisualizerPart responding to the "View | Instance Goals | Parameters" command. Figure 45 below shows the GMoDSVisualizerPart responding to the "View | Instance Goals | Goal Types" command.

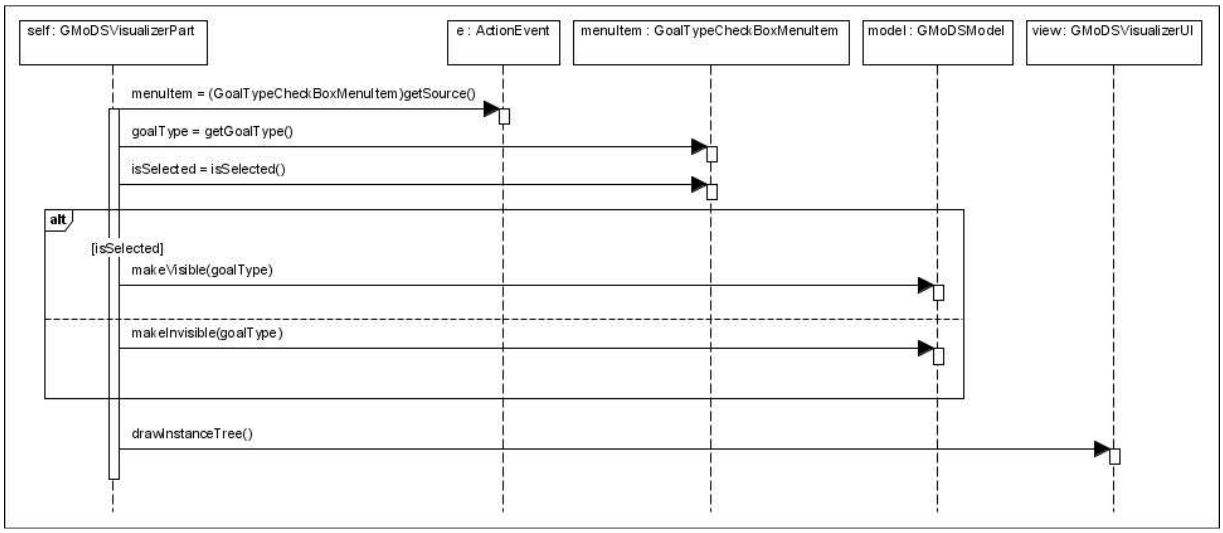

Figure 45 GMoDSVisualizerPart viewGoalTypes

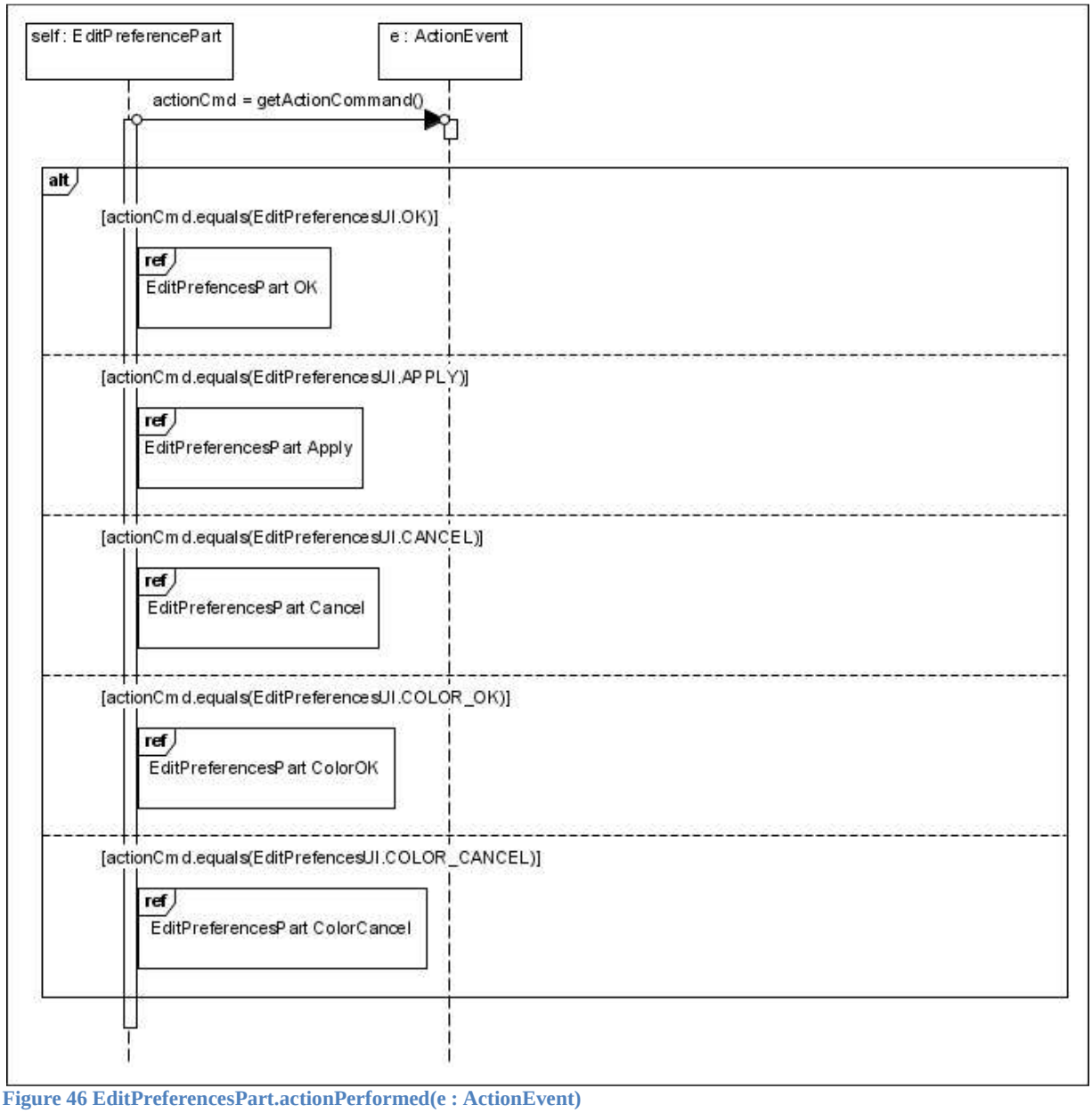

Figure 46 EditPreferencesPart.actionPerformed(e : ActionEvent)

Figure 46 above shows the EditPrefencesPart.actionPerformed method. Figure 47 below shows the EditPreferencesPart responding to the "OK" button on the main dialog presented by its UI. The methods "applyRandomEvents" and "applyStates" enforce the business rules for user input regarding the values of the flash and random event parameters. They will return true only if the business rules are satisfied and inform the user of the violation otherwise. Figure 48 below shows the EditPreferencesPart responding to the "Apply" button on the main dialog presented by its UI. Figure 49 below shows the EditPreferencesPart responding to the "Cancel" button on the main dialog presented by its UI.

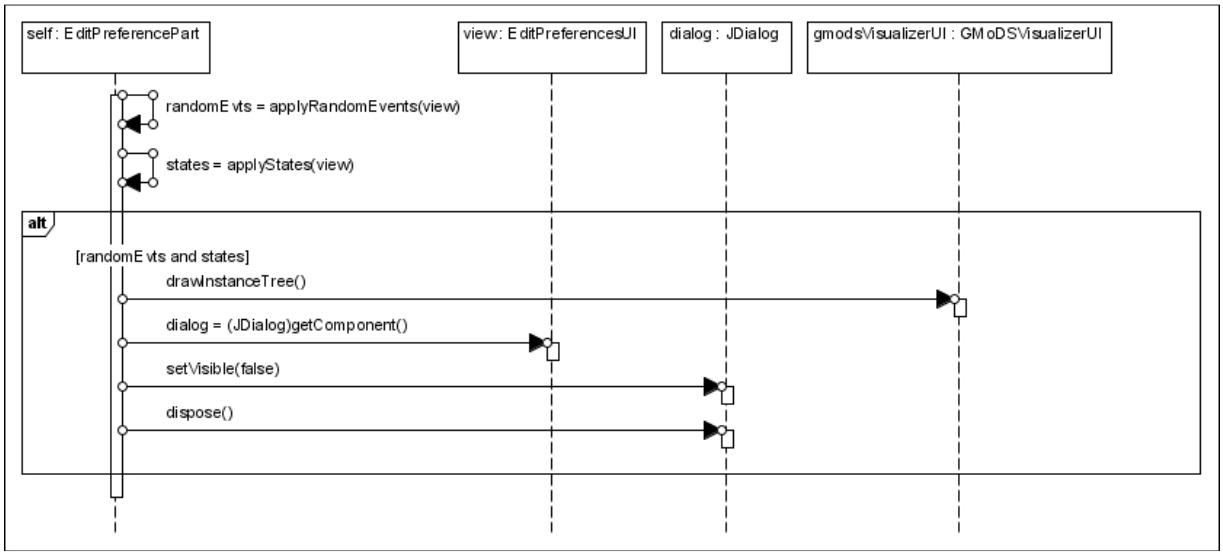

Figure 47 EditPreferencesPart OK

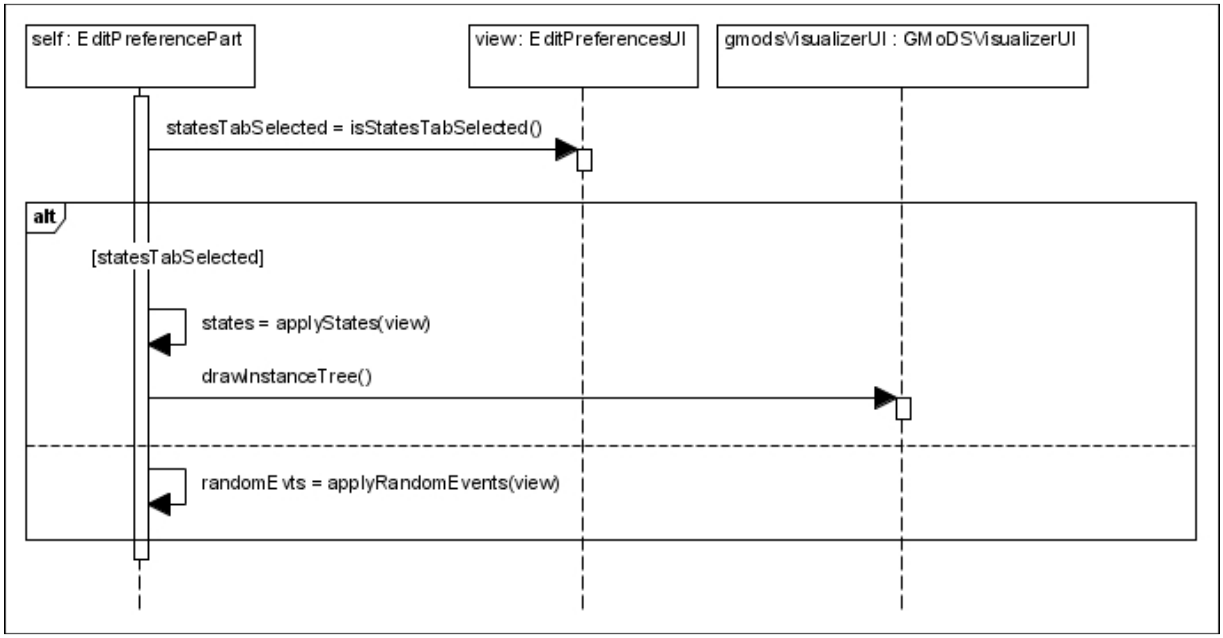

Figure 48 EditPreferencesPart Apply

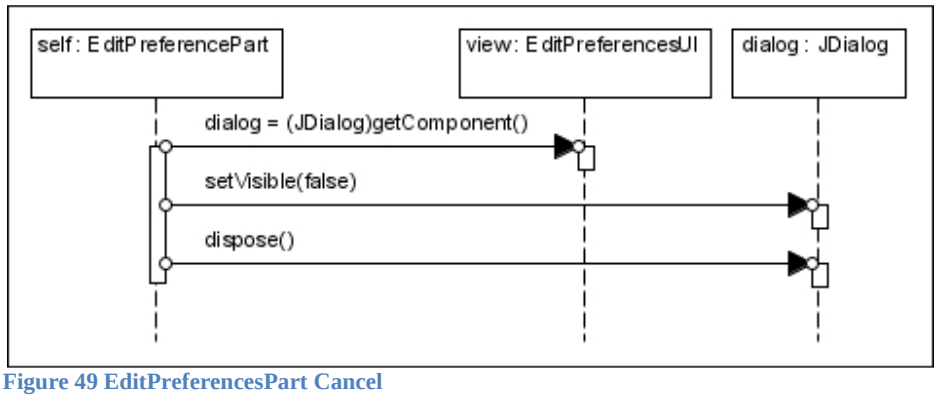

Figure 49 EditPreferencesPart Cancel

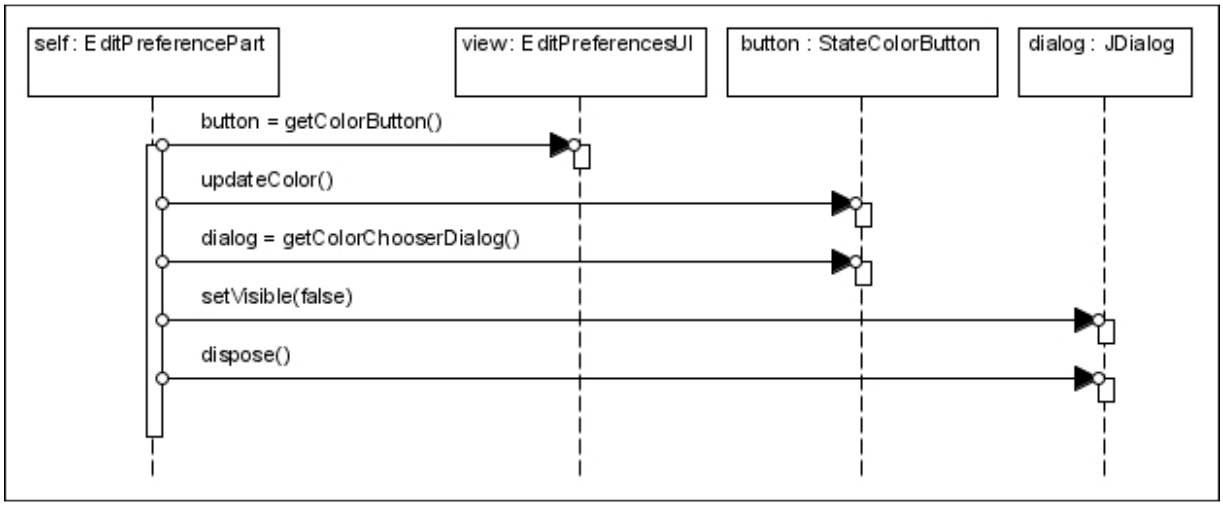

Figure 50 EditPreferencesPart ColorOK

Figure 50 above shows the EditPreferencesPart responding to the "OK" button on the color chooser dialog presented by its UI. Figure 51 below shows the EditPreferencesPart responding to the "Cancel" button on the color chooser dialog presented by its UI.

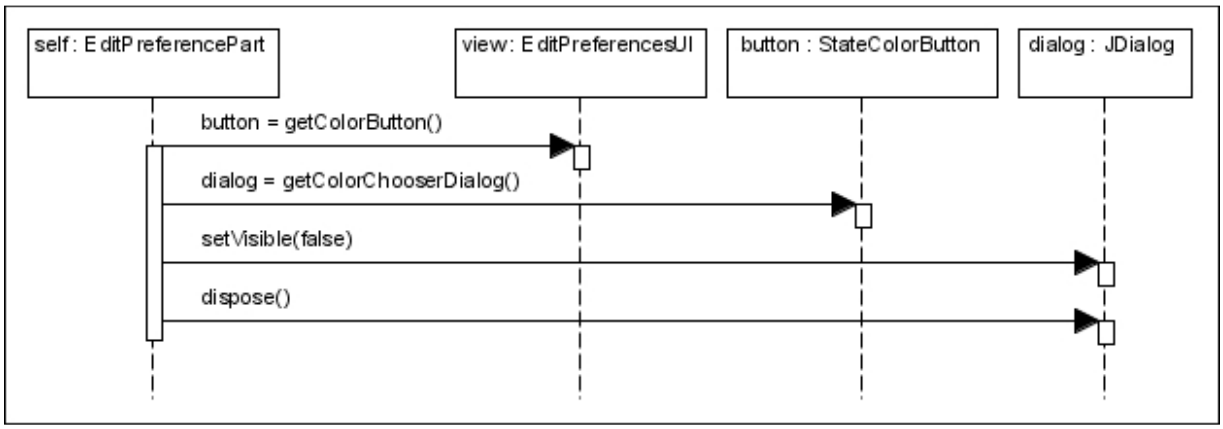

Figure 51 EditPreferencesPart ColorCancel

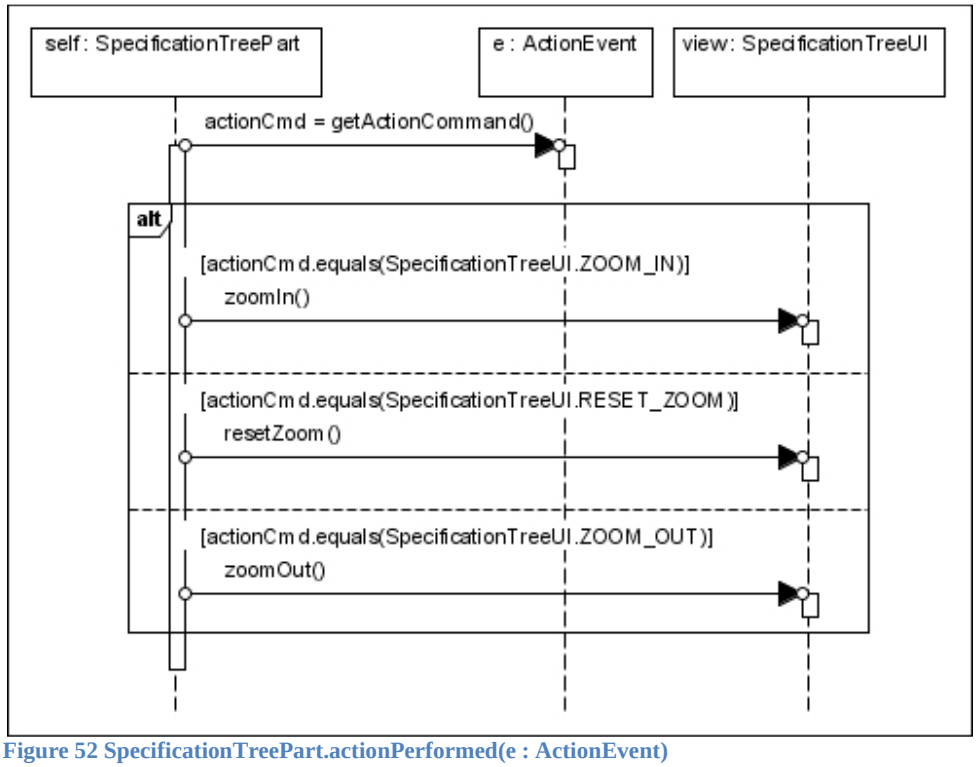

Figure 52 SpecificationTreePart.actionPerformed(e : ActionEvent)

Figure 52 above shows the SpecificationTreePart.actionPerformed method. Figure 53 below shows the InstanceTreePart.actionPerformed method.

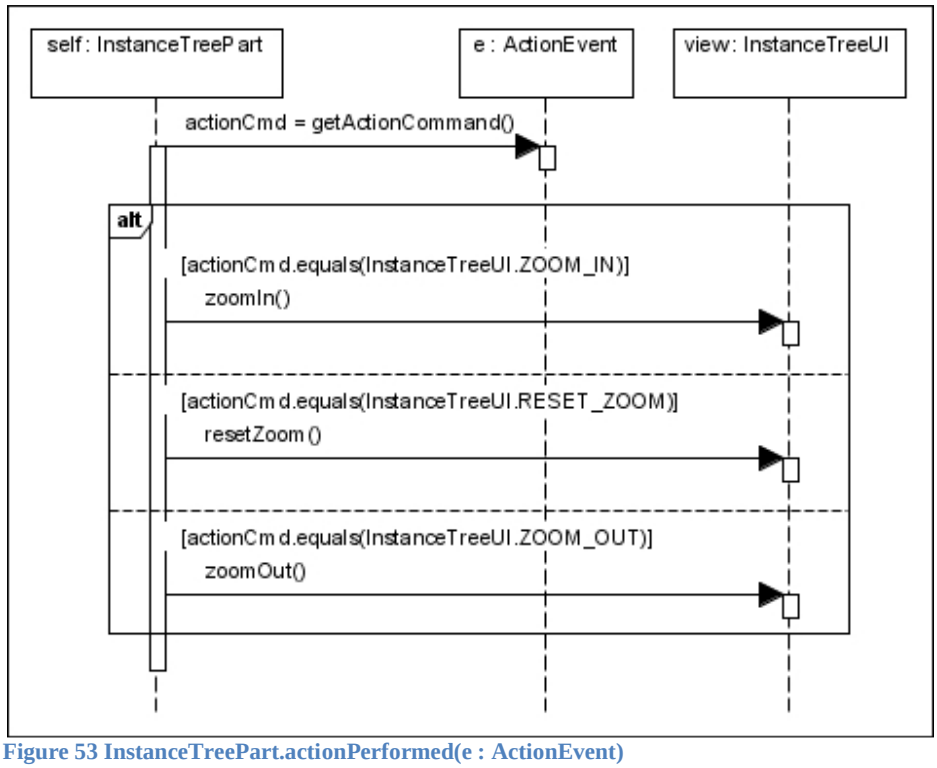

Figure 53 InstanceTreePart.actionPerformed(e : ActionEvent)

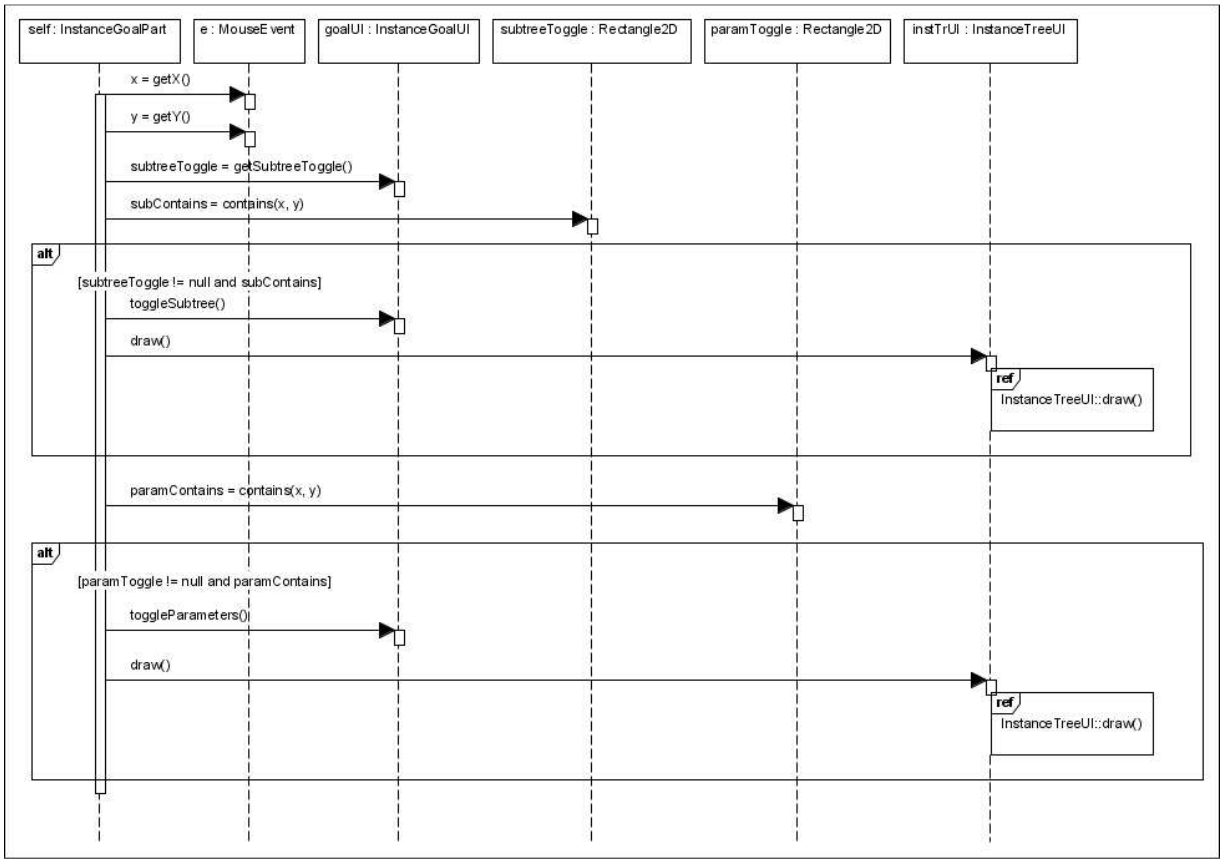

Figure 54 InstanceGoalPart.mouseClicked(e : MouseEvent)

Figure 54 above shows the InstanceGoalPart.mouseClicked method.