System Architectural 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 system and component architectural design document for the Goal Model for Dynamic Systems (GMoDS) Test Driver and Visualizer system. This document first provides a reference to the system context. Next, I describe the system architecture in terms of components, their responsibilities, and the rationale for these choices. Third, I decompose the GMoDS Test Driver component into architectural level modules, the module responsibilities, interface specifications, and design rationale. Fourth, I briefly describe the GMoDS Visualizer Model-View-Controller (MVC) architecture decomposing the model, view, and controller roles into their modules, the module responsibilities, interface specifications, and design rationale. Fifth, I describe the start up of the GMoDS Visualizer from the perspective of the GMoDS Test Driver main program as an example for simulation components to follow. Finally, I conclude the paper with a USE/OCL formal specification of event script methods.

2 References

1. “Vision Document 1.0 or 2.0” available at http://people.cis.ksu.edu/~mfraka/FrakaMSE.html.

3 System Context

The system context is shown in Figure 1 below.

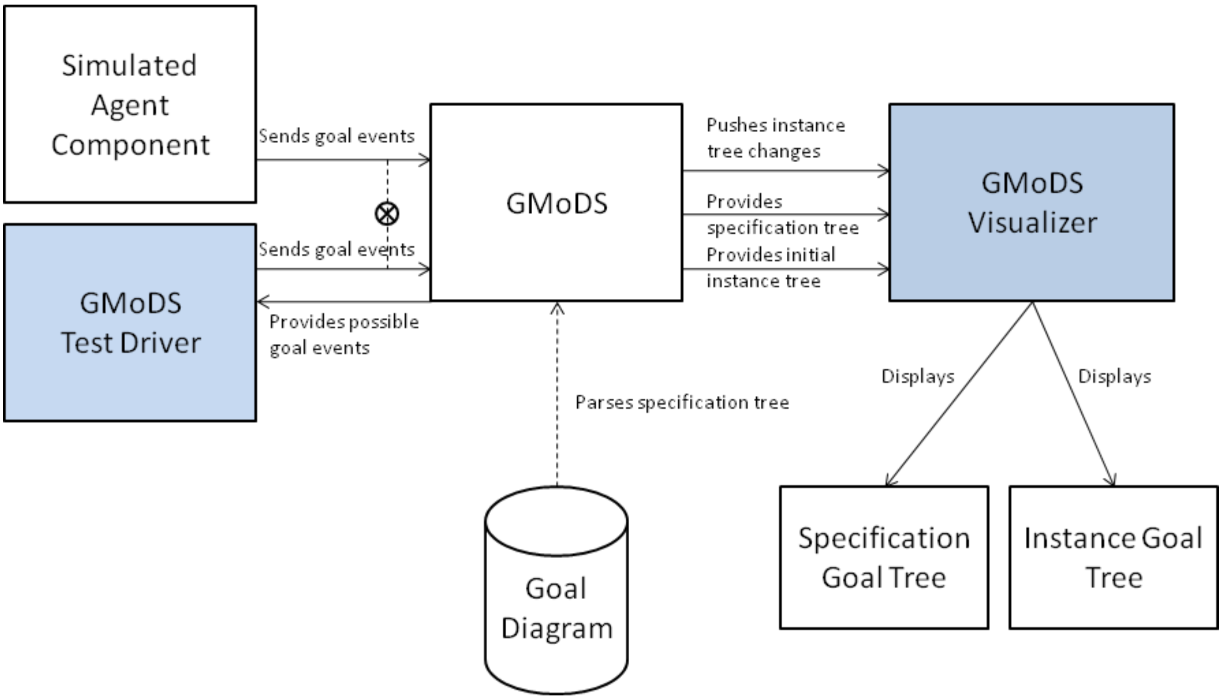
Simulated Agent Component
GMoDS Test Driver
Sends goal events
Sends goal events
Provides possible goal events
GMoDS
Pushes instance tree changes
Provides specification tree
Provides initial instance tree
GMoDS Visualizer
Parses specification tree
Goal Diagram
Displays
Displays
Specification Goal Tree
Instance Goal Tree

Figure 1 GMoDS Test Driver and Visualizer system context

More detail on the system context is available in [1] (see 2 above).

4 System Architecture

This section documents the system architecture in a component diagram, lists module responsibilities and interface specifications, and describes the design rationale.

4.1 System Components

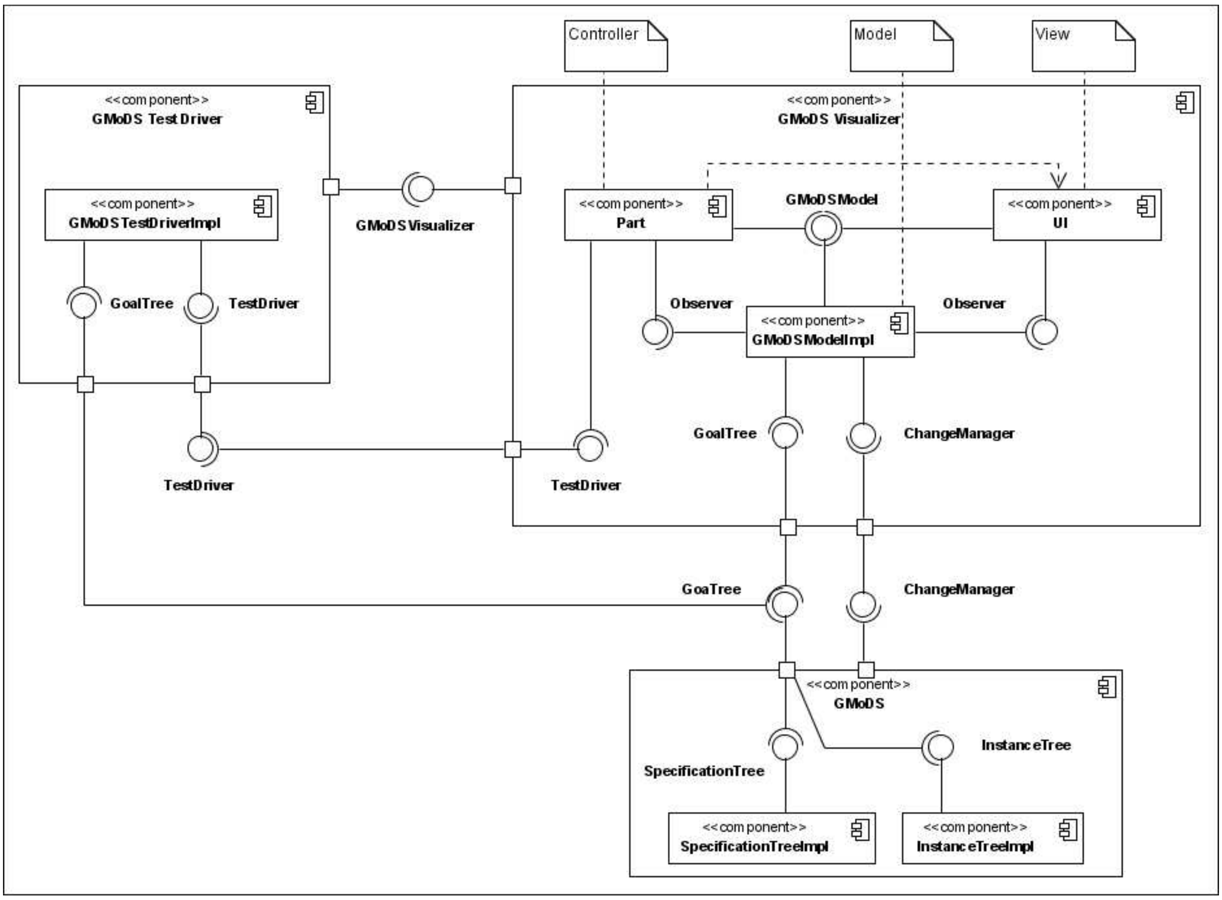

Figure 2 System components

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

The system reuses the Goal Model for Dynamic Systems (GMoDS) component to visualize its behavior. The exact version of GMoDS reused is specified in [1] (see 2 above). The GMoDS component provides the GoalTree interface and requires the ChangeManager interface. The client uses the GoalTree interface to pull information from GMoDS. GMoDS uses the ChangeManager interface to push information to the client.

The GMoDS Visualizer component provides the user interface for visualizing the behavior of GMoDS. Figure 2 notes show that the GMoDS Visualizer uses the Model-View-Controller (MVC) architecture. The GMoDS Visualizer defines the TestDriver interface that must be provided by the GMoDS Test Driver when the visualizer is tested using this component. The GMoDS Visualizer provides the GMoDSVisualizer interface to support its initialization.

The GMoDS Test Driver component provides the TestDriver interface implementation to support testing of the GMoDS Visualizer and uses the GMoDSVisualizer interface to initialize it.

4.2 System Component Responsibilities

Table 1 System component responsibilities

Component	**Responsibilities**
GMoDS	Provide the core objects and behaviors to be visualized. Provide pull and push access to these core objects.
GMoDS Visualizer	Provide the user interface for visualizing GMoDS object behaviors. Provide the user interface controls for the GMoDS Test Driver if configured.
GMoDS Test Driver	Provide the capability to test the GMoDS Visualizer in manual and automatic mode.

4.3 System Interface Specifications

All interfaces throw an IllegalArgumentException if their preconditions are violated except for the GMoDS Test Driver Launcher main program which prints an error message to the console and exits if its preconditions are not met.

Table 2 GMoDS Test Driver Launcher interface specifications

Launch the GMoDS Test Driver for a specific goal diagram.	Syntax:	main(args : string[]) : void
	Pre:	args.length = 1
	Pre:	args[0] is the goal diagram file name.
	Pre:	args[0] is a file that exists and is readable.
	Post:	The GMoDS component is created, initialized, and passed to the GMoDSTestDriverImpl and GMoDSVisualizerImpl.
	Post:	The GMoDSTestDriverImpl is created and passed to the GMoD Visualizer component.
	Post:	The GMoDSVisualizerImpl is created and initialized. The user interface is created, initialized, and made visible.

Table 3 GMoDSVisualizer interface specifications

Initialize the GMoDS Visualizer resulting in a visible, ready user interface.	Syntax:	initialize() : void
	Pre:	GMoDS GoalTree implementation != null.
	Pre:	GMoDS GoalTree implementation is initialized.
	Post:	The GMoDSVisualizerImpl is initialized. The user interface is created, initialized, and made visible.

Table 4 Test Driver interface specifications

Add an Observer of the event script (as in the Observer design pattern).	Syntax:	addObserver(o : Observer) : void
	Pre:	o != null.
	Post:	An Observer o is recorded and will be notified whenever the state of the EventScript changes.
Load an event script XML file into the TestDriver.	Syntax:	loadEventScript(eventScript : File) : void
	Pre:	eventScript != null.
	Pre:	eventScript File exists, is a File, and can be read.
	Post:	A DeterministicEventScript is created from the eventScript File.
	Post:	All valid GoalEvents specified in eventScript are included in the DeterministicEventScript
	Post:	The TestDriver enters manual mode.
	Post:	All invalid GoalEvents are discarded and the user is notified visually and in a log file of discarded GoalEvents.

Save the current event script as an XML file.	Syntax:	saveEventScript(eventScript : File) : void
	Pre:	TestDriver is in manual mode.
	Pre:	eventScript != null.
	Pre:	User must have permission to write the eventScript File.
	Pre:	If eventScript File exists then user must confirm that it will be overwritten.
	Post:	The current EventScript of validated Goal Events (events that have already been confirmed to refer to instance goals that exist in GMoDS) will be written to eventScript File using the XML schema defined in [1] (see 2 above).
	Post:	The TestDriver remains in manual mode.
Begin issuing random events using the current random event configuration parameters.	Syntax:	issueRandomEvents() : void
	Pre:	None.
	Post:	A RandomEventScriptImpl is created using the RandomEventParameters in effect during the method call.
	Post:	The TestDriver enters manual mode.
Place the TestDriver in automatic mode.	Syntax:	play() : void
	Pre:	TestDriver is in manual mode.
	Pre:	TestDriver has a next GoalEvent it can issue.
	Post:	The TestDriver enters automatic mode.
Place the TestDriver in manual mode.	Syntax:	pause() : void
	Pre:	TestDriver is in automatic mode.
	Pre:	TestDriver has a next GoalEvent it can issue.
	Post:	The TestDriver enters manual mode.

Issue the next event to GMoDS.	Syntax:	next() : void
	Pre:	TestDriver is in manual mode.
	Pre:	TestDriver has a next GoalEvent it can issue.
	Pre:	The next GoalEvent refers to a valid instance goal.
	Post:	The TestDriver issues the next GoalEvent to GMoDS.
	Post:	The TestDriver remains manual mode.
Determine if the TestDriver has a next event to issue to GMoDS.	Syntax:	hasNext() : boolean
	Pre:	None.
	Post:	Result = TestDriver has a next valid GoalEvent that can be issued to GMoDS.

4.4 System Architecture Design Rationale

The system architecture uses the Model-View-Controller (MVC) design pattern. The GMoDS Visualizer component has both the view and controller roles. The GMoDS Test Driver (if applicable) and the GMoDS components are both assigned the model role. The GMoDS Test Driver encapsulates the core GoalEvent objects that it can issue to GMoDS behind a well-defined TestDriver interface. This interface also implements the Observer design pattern to support the notification of the GMoDS Visualizer that it should check whether valid GoalEvents remain to be issued. The GMoDS component is encapsulated behind a GMoDSModel interface within the GMoDS Visualizer component allowing custom methods to support GMoDS Visualizer requirements.

5 GMoDS Test Driver Architecture

5.1 GMoDS Test Driver Decomposition

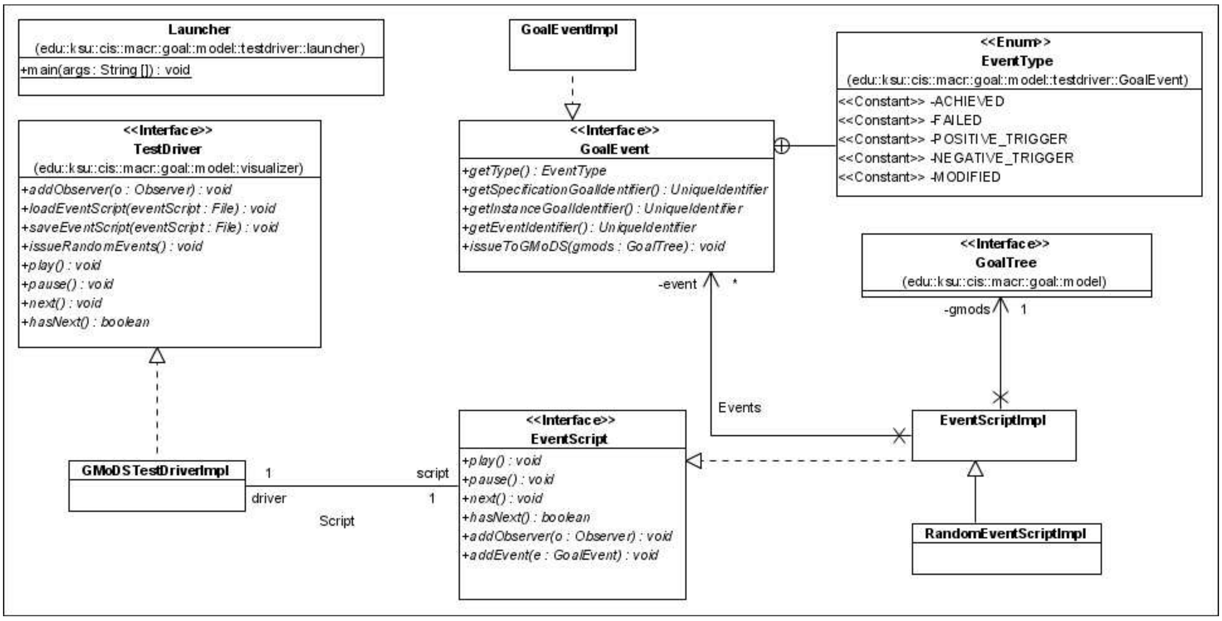
Launcher
(edu::ksu::cis::macr::goal::model::testdriver::launcher)
+main(args : String []) : void
GoalEventImpl
<<Enum>>
EventType
(edu::ksu::cis::macr::goal::model::testdriver::GoalEvent)
<<Constant>> -ACHIEVED
<<Constant>> -FAILED
<<Constant>> -POSITIVE_TRIGGER
<<Constant>> -NEGATIVE_TRIGGER
<<Constant>> -MODIFIED
<<Interface>>
TestDriver
(edu::ksu::cis::macr::goal::model::visualizer)
+addObserver(o : Observer) : void
+loadEventScript(eventScript : File) : void
+saveEventScript(eventScript : File) : void
+issueRandomEvents() : void
+play() : void
+pause() : void
+next() : void
+hasNext() : boolean
<<Interface>>
GoalEvent
+getType() : EventType
+getSpecificationGoalIdentifier() : UniqueIdentifier
+getInstanceGoalIdentifier() : UniqueIdentifier
+getEventIdentifier() : UniqueIdentifier
+issueToGMoDS(gmods : GoalTree) : void
<<Interface>>
GoalTree
(edu::ksu::cis::macr::goal::model)
-event
*
-gmods
1
Events
EventScriptImpl
<<Interface>>
EventScript
+play() : void
+pause() : void
+next() : void
+hasNext() : boolean
+addObserver(o : Observer) : void
+addEvent(e : GoalEvent) : void
GMoDSTestDriverImpl
1
script
driver
1
Script
RandomEventScriptImpl

Figure 3 GMoDS Test Driver architectural modules

Figure 3 above shows the GMoDS Test Driver component architecture. Since this is a small component and since it is used in the formal specification all GMoDS Test Driver modules are shown in the diagram.

5.1.1 GMoDS Test Driver Module Responsibilities

Table 5 GMoDS Test Driver module responsibilities

Component	Responsibilities
Launcher	Configure GMoDS, GMoDSTestDriverImpl, and the GMoDSVisualizerImpl. Initialize GMoDS and the GMoDSVisualizerImpl.
GMoDTestDriverImpl	Hold an EventScript. Implement loadEventScript and issueRandomEvents to create and install DeterministicEventScript and RandomEventScriptImpl, respectively.
EventScript	Define the behaviors of any EventScript.

Component	Responsibilities
EventScriptImpl	Hold the list of GoalEvents defining the script and provide default implementations of the EventScript interface.
RandomEventScriptImpl	Override the EventScriptImpl to create and issue random GoalEvents based on the RandomEventParameters configured by the user and the events defined by the goal diagram.
GoalEvent	Define the behaviors of a GoalEvent.
EventType	Define the possible types of any event in a goal diagram.
GoalEventImpl	Implement the GoalEvent interface.

5.1.2 GMoDS Test Driver Interface Specifications

Table 6 GMoDS Test Driver GoalEvent interface specifications

Access the EventType of a GoalEvent.	Syntax:	getType() : EventType
	Pre:	None.
	Post:	Result = this.eventType
Access the UniqueIdentifier of the specification goal referenced by a GoalEvent.	Syntax:	getSpecificationGoalIdentifier() : UniqueIdentifier
	Pre:	None.
	Post:	Result = this.specificationGoalID
Access the UniquieIdentifier of the instance goal referenced by a GoalEvent.	Syntax:	getInstanceGoalIdentifier() : UniqueIdentifier
	Pre:	None.
	Post:	Result = this.instanceGoalID
Access the UniqueIdentifier of the SpecificationEvent referenced by a GoalEvent.	Syntax:	getEventGoalIdentifier() : UniqueIdentifier
	Pre:	this.eventType = EventType.POSITIVE_TRIGGER or this.eventType = EventType.NEGATIVE_TRIGGER
	Post:	Result = this.eventID

Table 7 GMoDS Test Driver EventScript interface specifications

Add an event valid with respect to the GMoDS specification tree to the end of the script.	Syntax:	addEvent(e : GoalEvent) : void
	Pre:	e != null
	Pre:	e is not already included in the script.
	Pre:	e.type is valid.
	Pre:	if e.type = #MODIFIED then at least one parameter must be provided for the event.
	Pre:	e.getSpecificationGoalIdentifier() refers to a specification goal that exists in the specification tree.
	Pre:	if e.type = #ACHIEVED then e.getSpecificationGoalIdentifier() = ‘ACHIEVED’ and the specification goal is a leaf.
	Pre:	if e.type = #FAILED then e.getSpecificationGoalIdentifier() = ‘FAILED’ and the specification goal is a leaf.
	Pre:	if e.type != #MODIFIED then e.getSpecificationEventIdentiifer() refers to an specification event defined in the specification tree.
	Post:	(events – events@pre)->size() = 1
	Post:	events.includes(e)
	Post:	events.last() = e
Place the EventScript in automatic mode.	Syntax:	play() : void
	Pre:	EventScript is in manual mode.
	Pre:	EventScript has a next GoalEvent it can issue.
	Post:	The EventScript enters automatic mode.

Place the EventScript in manual mode.	Syntax:	pause() : void
	Pre:	EventScript is in automatic mode.
	Pre:	EventScript has a next GoalEvent it can issue.
	Post:	The EventScript enters manual mode.
Issue the next event to GMoDS.	Syntax:	next() : void
	Pre:	EventScript is in manual mode.
	Pre:	EventScript has at least 1 event.
	Pre:	EventScript has a next GoalEvent it can issue.
	Pre:	The next GoalEvent refers to a valid instance goal.
	Pre:	If next GoalEvent type != #MODIFIED then the next event refers to a valid active instance goal.
	Post:	If the next GoalEvent type != #MODIFIED the EventScript issues the next GoalEvent to the GMoDS event method.
	Post:	If the next GoalEvent type = #MODIFIED the EventScript issues the next GoalEvent to the GMoDS modifyInstanceGoal method.
	Post:	The EventScript index refers to the next event if one exists.
	Post:	The EventScript remains manual mode.
Determine if the EventScript has a next event to issue to GMoDS.	Syntax:	hasNext() : boolean
	Pre:	None.
	Post:	Result = EventScript has a next valid GoalEvent that can be issued to GMoDS.
Add an Observer of the event script (as in the Observer design pattern).	Syntax:	addObserver(o : Observer) : void
	Pre:	o != null.
	Post:	An Observer o is recorded and will be notified whenever the state of the EventScript changes.

5.1.3 GMoDS Test Driver Design Rationale

The heart of the GMoDS Test Driver is the EventScriptImpl and RandomEventScriptImpl that extends it and the GoalEventImpl. The EventScriptImpl provides the deterministic (usually file-based) event script functionality. The RandomEventScriptImpl provides random GoalEvent generation. The GoalEventImpl enforces the invariants that assure valid InstanceGoals and SpecificationEvents are sent to GMoDS. The GMoDS Test Driver architecture was derived from analysis of the objects referenced in Vision Document 1.0 [1] (see 2 above).

6 GMoDS Visualizer Architecture

The GMoDS Visualizer uses the MVC architectural design pattern. Each section that follows decomposes the modules that take on each role in the MVC design pattern. I did not make use of the Command design pattern because the visualizer has no requirement to support undo operations.

6.1 GMoDS Visualizer Model Decomposition

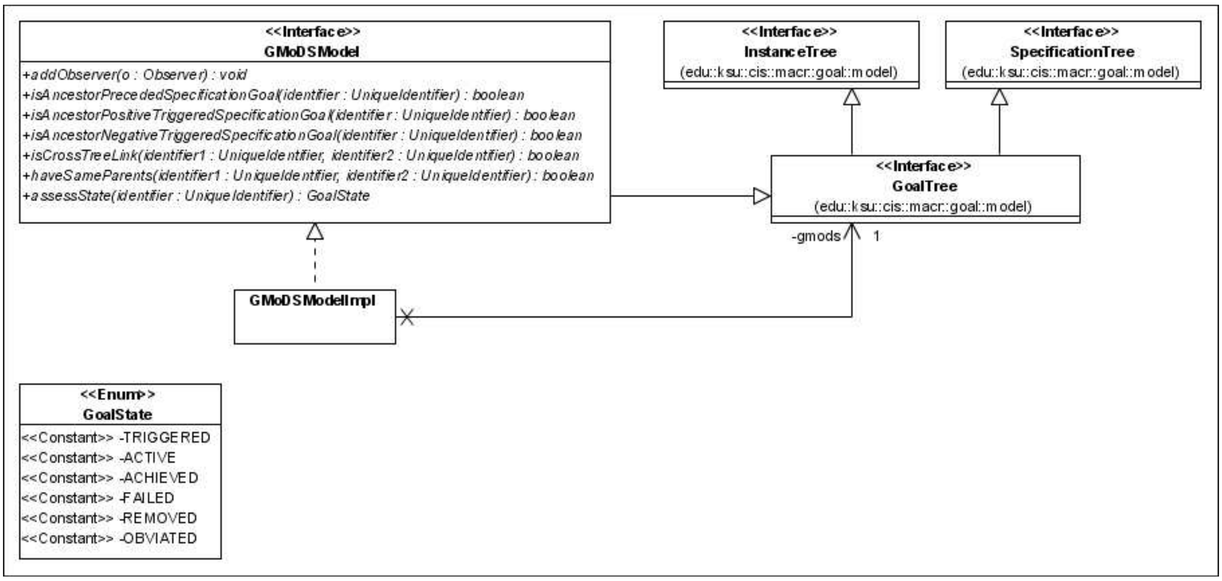
<<Interface>>
GMoDSModel
+addObserver(o : Observer) : void
+isAncestorPrecededSpecificationGoal(identifier : UniqueIdentifier) : boolean
+isAncestorPositiveTriggeredSpecificationGoal(identifier : UniqueIdentifier) : boolean
+isAncestorNegativeTriggeredSpecificationGoal(identifier : UniqueIdentifier) : boolean
+isCrossTreeLink(identifier1 : UniqueIdentifier, identifier2 : UniqueIdentifier) : boolean
+haveSameParents(identifier1 : UniqueIdentifier, identifier2 : UniqueIdentifier) : boolean
+assessState(identifier : UniqueIdentifier) : GoalState
<<Interface>>
InstanceTree
(edu::ksu::cis::macr::goal::model)
<<Interface>>
SpecificationTree
(edu::ksu::cis::macr::goal::model)
<<Interface>>
GoalTree
(edu::ksu::cis::macr::goal::model)
-gmods
1
GMoDSModelImpl
<<Enum>>
GoalState
<<Constant>> -TRIGGERED
<<Constant>> -ACTIVE
<<Constant>> -ACHIEVED
<<Constant>> -FAILED
<<Constant>> -REMOVED
<<Constant>> -OBVIATED

Figure 4 GMoDS Visualizer model modules

6.1.1 GMoDS Visualizer Model Module Responsibilities

Table 8 GMoDS Visualizer model module responsibilities

Component	Responsibilities
GoalState	Enumeration of possible goal states.
GMoDSModel	Define methods for access and evaluation of the core GMoDS objects.
GMoDSModelImpl	Implement methods for access and evaluation of the core GMoDS objects.

6.1.2 GMoDS Visualizer Model Interface Specifications

Table 9 below shows custom methods for accessing and evaluating core GMoDS objects. The methods defined for GMoDS native interfaces (GoalTree, SpecificationTree, and InstanceTree) are not documented in this paper.

Table 9 GMoDS Visualizer GMoDSModel interface specifications

Add an Observer of the GMoDSModel (as in the Observer design pattern).	Syntax:	addObserver(o : Observer) : void
	Pre:	o != null.
	Post:	An Observer o is recorded and will be notified whenever the state of GMoDS changes.
Determine if any ancestor of the specified specification goal is the target of a precedes relation.	Syntax:	isAncestorPrecededSpecificationGoal(identifier : UniqueIdentifier) : boolean
	Pre:	identifier != null.
	Post:	Result = true if any ancestor of the specified specification goal is the target of a precedes relation.
Determine if any ancestor of the specified specification goal is the target of a positive trigger.	Syntax:	isAncestorPositiveTriggeredSpecificationGoal (identifier : UniqueIdentifier) : boolean
	Pre:	identifier != null.
	Post:	Result = true if any ancestor of the specified specification goal is the target of a positive trigger.
Determine if any ancestor of the specified specification goal is the target of a negative trigger.	Syntax:	isAncestorNegativeTriggeredSpecificationGoal (identifier : UniqueIdentifier) : boolean
	Pre:	identifier != null.
	Post:	Result = true if any ancestor of the specified specification goal is the target of a negative trigger.

Determine if the two specified specification goals do not have the same parents.	Syntax:	isCrossTreeLink (identifier1 : UniqueIdentifier, identifier2 : UniqueIdentifier) : boolean
	Pre:	identifier1 != null.
	Pre:	identifier2 != null.
	Post:	Result = true if the two specified specification goals do not have the same parents.
Determine if the two specified specification goals have the same parents.	Syntax:	haveSameParents (identifier1 : UniqueIdentifier, identifier2 : UniqueIdentifier) : boolean
	Pre:	identifier1 != null.
	Pre:	identifier2 != null.
	Post:	Result = true if the two specified specification goals have the same parents.
Evaluate the GoalState of the specified instance goal.	Syntax:	assessState(identifier : UniqueIdentifier) : GoalState
	Pre:	identifier != null.
	Post:	Result = the GoalState of the specified instance goal.

6.1.3 GMoDS Visualizer Model Design Rationale

The GMoDS component is encapsulated behind a GMoDSModel interface within the GMoDS Visualizer component to allow custom methods to support GMoDS Visualizer requirements.

6.2 GMoDS Visualizer View Decomposition

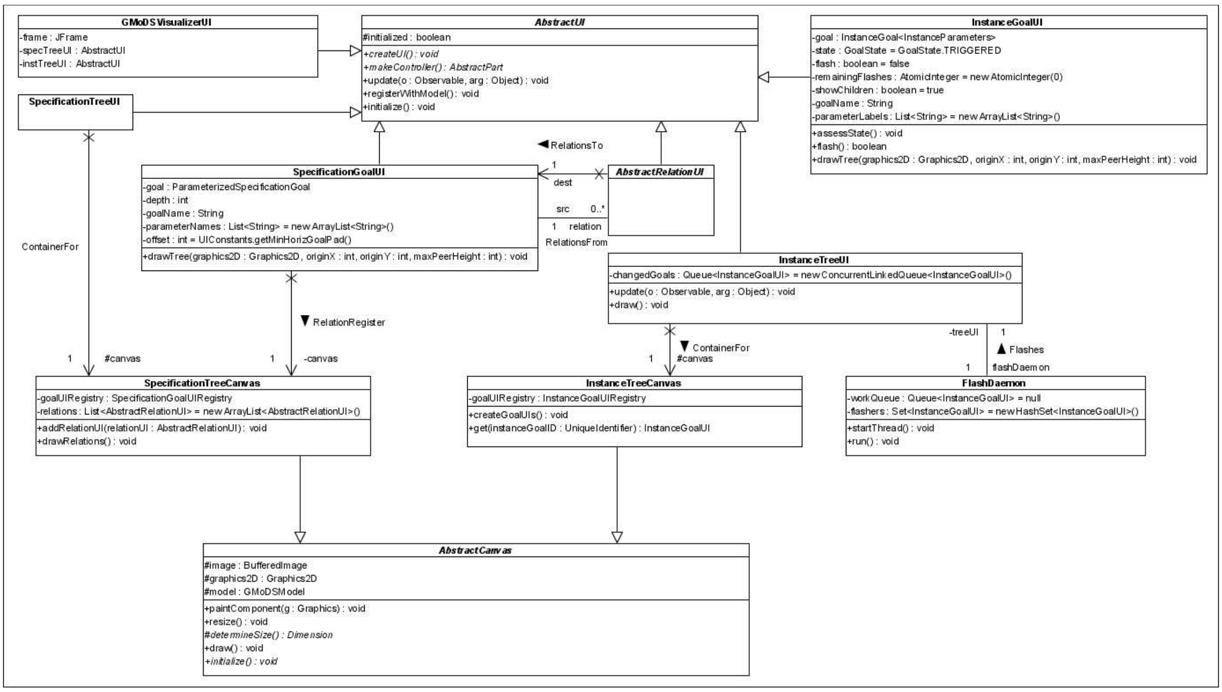
GMoDSVisualizerUI
-frame : JFrame
-specTreeUI : AbstractUI
-instTreeUI : AbstractUI
AbstractUI
#initialized : boolean
+createUI() : void
+makeController() : AbstractPart
+update(o : Observable, arg : Object) : void
+registerWithModel() : void
+initialize() : void
InstanceGoalUI
-goal : InstanceGoal<InstanceParameters>
-state : GoalState = GoalState.TRIGGERED
-flash : boolean = false
-remainingFlashes : AtomicInteger = new AtomicInteger(0)
-showChildren : boolean = true
-goalName : String
-parameterLabels : List<String> = new ArrayList<String>()
+assessState() : void
+flash() : boolean
+drawTree(graphics2D : Graphics2D, originX : int, originY : int, maxPeerHeight : int) : void
SpecificationTreeUI
RelationsTo
1
dest
src
0..*
1
relation
RelationsFrom
SpecificationGoalUI
-goal : ParameterizedSpecificationGoal
-depth : int
-goalName : String
-parameterNames : List<String> = new ArrayList<String>()
-offset : int = UIConstants.getMinHorizGoalPad()
+drawTree(graphics2D : Graphics2D, originX : int, originY : int, maxPeerHeight : int) : void
AbstractRelationUI
ContainerFor
InstanceTreeUI
-changedGoals : Queue<InstanceGoalUI> = new ConcurrentLinkedQueue<InstanceGoalUI>()
+update(o : Observable, arg : Object) : void
+draw() : void
RelationRegister
-treeUI
1
Flashes
ContainerFor
#canvas
1
1
flashDaemon
1
#canvas
1
-canvas
SpecificationTreeCanvas
-goalUIRegistry : SpecificationGoalUIRegistry
-relations : List<AbstractRelationUI> = new ArrayList<AbstractRelationUI>()
+addRelationUI(relationUI : AbstractRelationUI) : void
+drawRelations() : void
InstanceTreeCanvas
-goalUIRegistry : InstanceGoalUIRegistry
+createGoalUIs() : void
+get(instanceGoalID : UniqueIdentifier) : InstanceGoalUI
FlashDaemon
-workQueue : Queue<InstanceGoalUI> = null
-flashers : Set<InstanceGoalUI> = new HashSet<InstanceGoalUI>()
+startThread() : void
+run() : void
AbstractCanvas
#image : BufferedImage
#graphics2D : Graphics2D
#model : GMoDSModel
+paintComponent(g : Graphics) : void
+resize() : void
#determineSize() : Dimension
+draw() : void
+initialize() : void

Figure 5 GMoDS Visualizer view modules

6.2.1 GMoDS Visualizer View Module Responsibilities

Table 10 GMoDS Visualizer view module responsibilities

Component	Responsibilities
AbstractUI	Define the basic behaviors and responsibilities of the view role. Hold references to the core model and TestDriver if present.
AbstractCanvas	Hold the Java 2D image upon which a diagram is drawn. Define the methods concrete canvases must support.
GMoDSVisualizerUI	The top level concrete user interface. Hold the JFrame containing all visual components. Provide the JMenuBar and host the TestDriver JToolBar. Hold the SpecificationTreeUI and InstanceTreeUI in a JSplitPane.
SpecificationTreeUI	Define the UI for the specification tree. Provide zoom and scroll controls for the specification tree.
SpecificationTreeCanvas	Draw the specification tree.
SpecificationGoalUI	Define the UI for a specification goal.

Component	Responsibilities
AbstractRelationUI	Define the basic behaviors of a relation UI between 2 specification goal UIs. Used for positive and negative triggers and precedes relations.
InstanceTreeUI	Define the UI for the instance tree. Provide zoom and scroll controls for the instance tree.
InstanceTreeCanvas	Draw the instance tree.
InstanceGoalUI	Define the UI for an instance goal.
FlashDaemon	Flash added and changed instance goal UIs for the desired rate and duration.

6.2.2 GMoDS Visualizer View Interface Specifications

Table 11 GMoDS Visualizer AbstractUI interface specifications

Create this view and all subordinate views.	Syntax:	createUI() : void
	Pre:	None.
	Post:	This view and all subordinate views are created.
Create the appropriate controller for this view.	Syntax:	makeController() : AbstractPart
	Pre:	None.
	Post:	Result = The appropriate controller for this view is created.
Respond to notification of a change in the model.	Syntax:	update (o : Observable, arg : Object) : void
	Pre:	The Observable o (the model) has changed state.
	Post:	This view makes appropriate changes to the view based on changes in the Observable.
Register with the model if this view needs to do so.	Syntax:	registerWithModel() : void
	Pre:	None.
	Post:	If this view needs to receive updates from the model it registers as an Observer with it.

Initialize this view.	Syntax:	initialize() : void
	Pre:	None.
	Post:	This view and all subordinate views are initialized.

Table 12 GMoDS Visualizer AbstractCanvas interface specifications

Paint the component holding the Java 2D image.	Syntax:	paintComponent(g : Graphics) : void
	Pre:	None.
	Post:	This canvas paints the component it holds that displays the image with the Java 2D drawing.
Create an image with a white background using the dimensions that will contain all drawing elements.	Syntax:	resize() : void
	Pre:	None.
	Post:	This canvas calculates the minimum dimensions for its displayed image, resizes it, and fills it with a white background.
Determine the minimum dimensions that will contain all drawing elements.	Syntax:	determineSize() : void
	Pre:	None.
	Post:	The concrete canvas should calculate the minimum dimensions for its displayed elements.
Draw viewed elements on the Java 2D image.	Syntax:	draw() : void
	Pre:	None.
	Post:	The concrete canvas draws its elements on the Java 2D image.
Initialize the canvas.	Syntax:	initialize() : void
	Pre:	None.
	Post:	The canvas is initialized.

Table 13 GMoDS Visualizer SpecificationTreeCanvas interface specifications

Add an AbstractRelationUI to the list of relations to draw.	Syntax:	addRelationUI(relationUI : AbstractRelationUI) : void
	Pre:	None.
	Post:	The relationUI is added to the list of relationUIs drawn on the canvas.
Draw the AbstractRelationUIs on the image.	Syntax:	drawRelations() : void
	Pre:	None.
	Post:	All AbstractRelationUIs are drawn on the canvas.

Table 14 GMoDS Visualizer SpecificationGoalUI interface specifications

Draw the SpecificationGoalUI and its descendants on the image.	Syntax:	drawTree(graphics2D : Graphics2D) : void
	Pre:	None.
	Post:	This SpecificationGoalUI and its descendants are drawn on the Java 2D image.

Table 15 GMoDS Visualizer InstanceTreeUI interface specifications

Begin flashing added or changed InstanceGoalUIs.	Syntax:	update (o : Observable, arg : Object) : void
	Pre:	The Observable o (the model) has changed state.
	Post:	Added or changed InstanceGoalUIs begin to flash.
Draw and repaint the canvas.	Syntax:	draw() : void
	Pre:	None.
	Post:	The canvas held by this view is redrawn and repainted to allow dynamic changes to appear.

Table 16 GMoDS Visualizer InstanceTreeCanvas interface specifications

Create all added InstanceGoalUIs.	Syntax:	createGoalUIs() : void
	Pre:	None.
	Post:	This canvas creates all added InstanceGoalUIs and assures they are ordered, assessed, and registered for later display.
Get the specified InstanceGoalUI.	Syntax:	get(instanceGoalID : UniqueIdentifier) : InstanceGoalUI
	Pre:	None.
	Post:	Result = the InstanceGoalUI specified by the instanceGoalID.

Table 17 GMoDS Visualizer InstanceGoalUI interface specifications

Assess and record the GoalState of this InstanceGoalUI.	Syntax:	assessState() : void
	Pre:	None.
	Post:	this.state = model.assessState(goal.getIdentifier())
Invert the flash property, calculate the remaining number of flashes, and return false when there are no remaining flashes.	Syntax:	flash() : boolean
	Pre:	None.
	Post:	flash = !flash
	Post:	if (!flash) remainingFlashes = remainingFlashes@pre – 1
	Post:	Result = remainingFlashes > 0
Draw this InstanceGoalUI and its descendants on the image.	Syntax:	drawTree(graphics2D : Graphics2D) : void
	Pre:	None.
	Post:	This InstanceGoalUI and its descendants are drawn on the Java 2D image.

Table 18 GMoDS Visualizer FlashDaemon interface specifications

Start the Thread executing the run() method of the FlashDaemon.	Syntax:	startThread() : void
	Pre:	The thread is not running.
	Post:	The thread calling FlashDaemon.run() is started.
The asynchronous process that signals added/changed InstanceGoalUIs to invert their flash property and redraws the instance tree.	Syntax:	run() : void
	Pre:	The thread is running.
	Body:	The FlashDaemon polls for and adds all InstanceGoalUIs in its workQueue to the set of flashing goals (flashers).
	Body:	If there are no flashers, wait until notified that a flasher has been added.
	Body:	If there are flashers, flash each flasher and redraw the InstanceTreeUI.
	Body:	Remove all flashers whose flash() method returns false.
	Post:	None. The thread never exits until the system exits.

6.2.3 GMoDS Visualizer View Design Rationale

I selected the MVC design pattern to allow for maximum flexibility in designing views of the core GMoDS objects.

6.3 GMoDS Visualizer Controller Decomposition

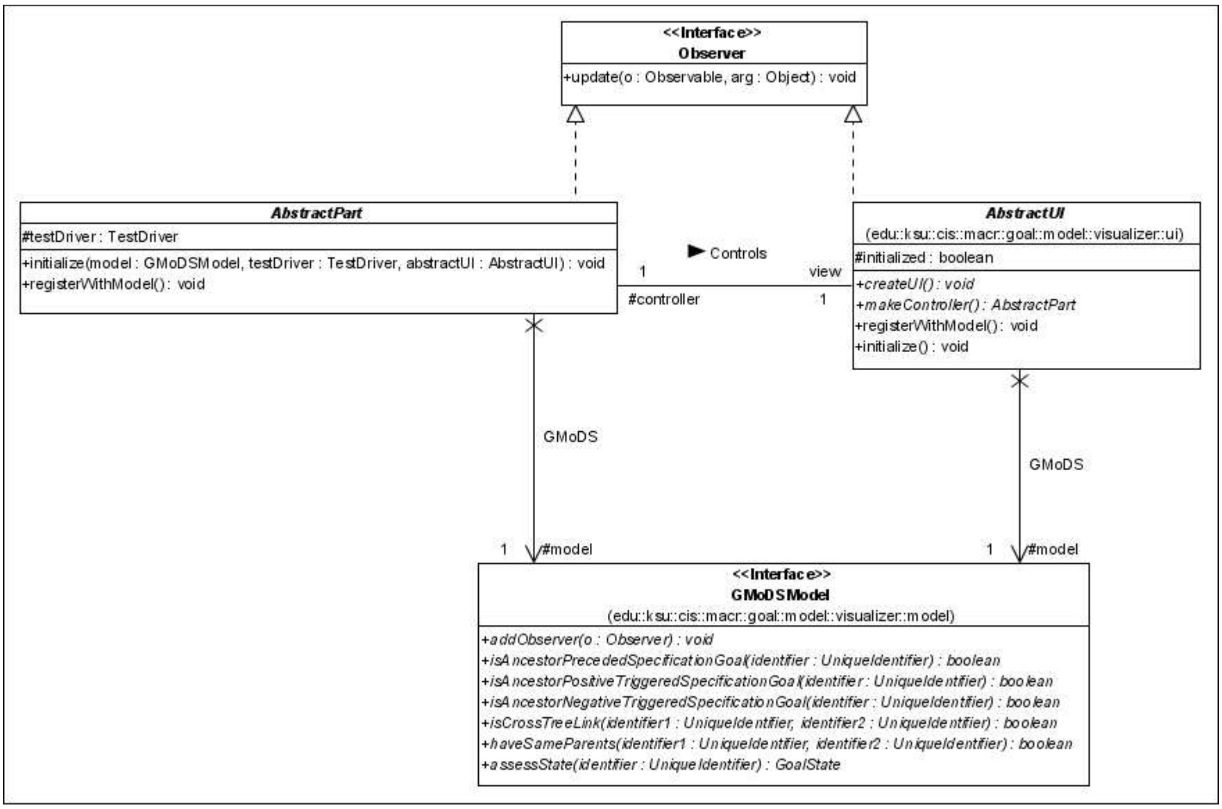
<<Interface>>
Observer
+update(o : Observable, arg : Object) : void
AbstractPart
#testDriver : TestDriver
+initialize(model : GMoDSModel, testDriver : TestDriver, abstractUI : AbstractUI) : void
+registerWithModel() : void
Controls
1
view
#controller
1
AbstractUI
(edu::ksu::cis::macr::goal::model::visualizer::ui)
#initialized : boolean
+createUI() : void
+makeController() : AbstractPart
+registerWithModel() : void
+initialize() : void
GMoDS
GMoDS
1
#model
1
#model
<<Interface>>
GMoDSModel
(edu::ksu::cis::macr::goal::model::visualizer::model)
+addObserver(o : Observer) : void
+isAncestorPrecededSpecificationGoal(identifier : UniqueIdentifier) : boolean
+isAncestorPositiveTriggeredSpecificationGoal(identifier : UniqueIdentifier) : boolean
+isAncestorNegativeTriggeredSpecificationGoal(identifier : UniqueIdentifier) : boolean
+isCrossTreeLink(identifier1 : UniqueIdentifier, identifier2 : UniqueIdentifier) : boolean
+haveSameParents(identifier1 : UniqueIdentifier, identifier2 : UniqueIdentifier) : boolean
+assessState(identifier : UniqueIdentifier) : GoalState

Figure 6 GMoDS Visualizer controller modules

6.3.1 GMoDS Visualizer Controller Module Responsibilities

Table 19 GMoDS Visualizer controller module responsibilities

Component	Responsibilities
AbstractPart	Define basic methods for setting up a controller associated with its view, model, and TestDriver.

6.3.2 GMoDS Visualizer Controller Interface Specifications

Table 20 GMoDS Visualizer AbstractPart interface specifications

AbstractPart intialize	Syntax:	initialize(model : GMoDSModel, testDriver : TestDriver, abstractUI : AbstractUI) : void
	Pre:	None.
	Post:	This controller is initialized with references to the model, view, and TestDriver.

AbstractPart registerWithModel	Syntax:	registerWithModel() : void
	Pre:	None.
	Post:	If this controller needs to receive updates from the model it registers as an Observer with it or the TestDriver.

6.3.3 GMoDS Visualizer Controller Design Rationale

I selected the MVC design pattern to support unit testing of controller behaviors.

7 System Startup Behavior

Figure 7 through Figure 13 illustrate the system startup behavior. Figure 7 shows the steps taken by the GMoDS Test Driver Launcher main program to make use of the GMoDS Visualizer. Simulation components should follow these same steps except that they will skip creating a TestDriver and pass null into the constructor of GMoDSVisualizerImpl for the TestDriver parameter. The figures also illustrate the initialization of the MVC architecture.

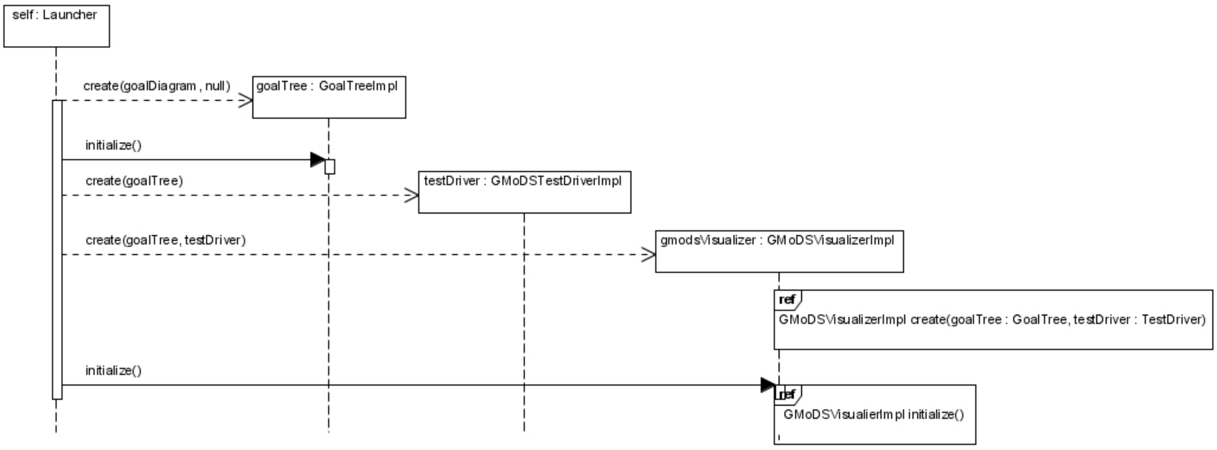
self : Launcher
create(goalDiagram , null)
goalTree : GoalTreeImpl
initialize()
create(goalTree)
testDriver : GMoDSTestDriverImpl
create(goalTree, testDriver)
gmodsVisualizer : GMoDSVisualizerImpl
ref
GMoDSVisualizerImpl create(goalTree : GoalTree, testDriver : TestDriver)
initialize()
ref
GMoDSVisualierImpl initialize()

Figure 7 GMoDS Test Driver Launcher main program behavior

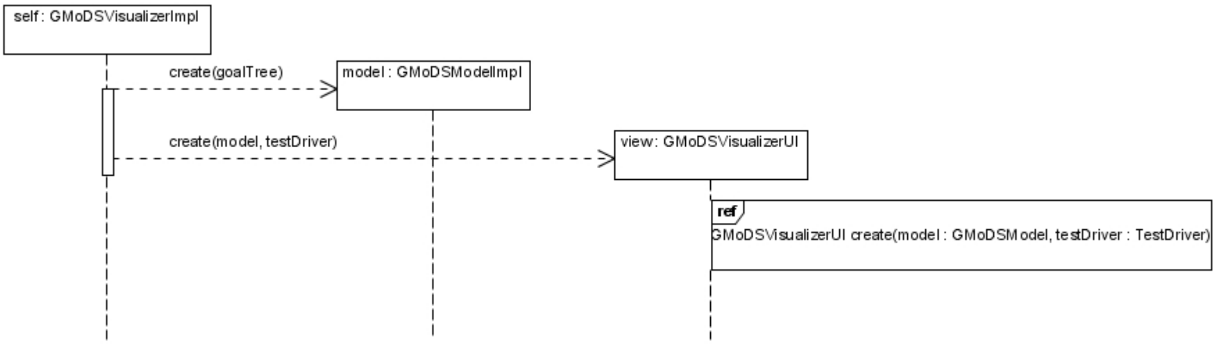
self : GMoDSVisualizerImpl
create(goalTree)
model : GMoDSModelImpl
create(model, testDriver)
view : GMoDSVisualizerUI
ref
GMoDSVisualizerUI create(model : GMoDSModel, testDriver : TestDriver)

Figure 8 GMoDSVisualizerImpl create(goalTree : GoalTree, testDriver : TestDriver)

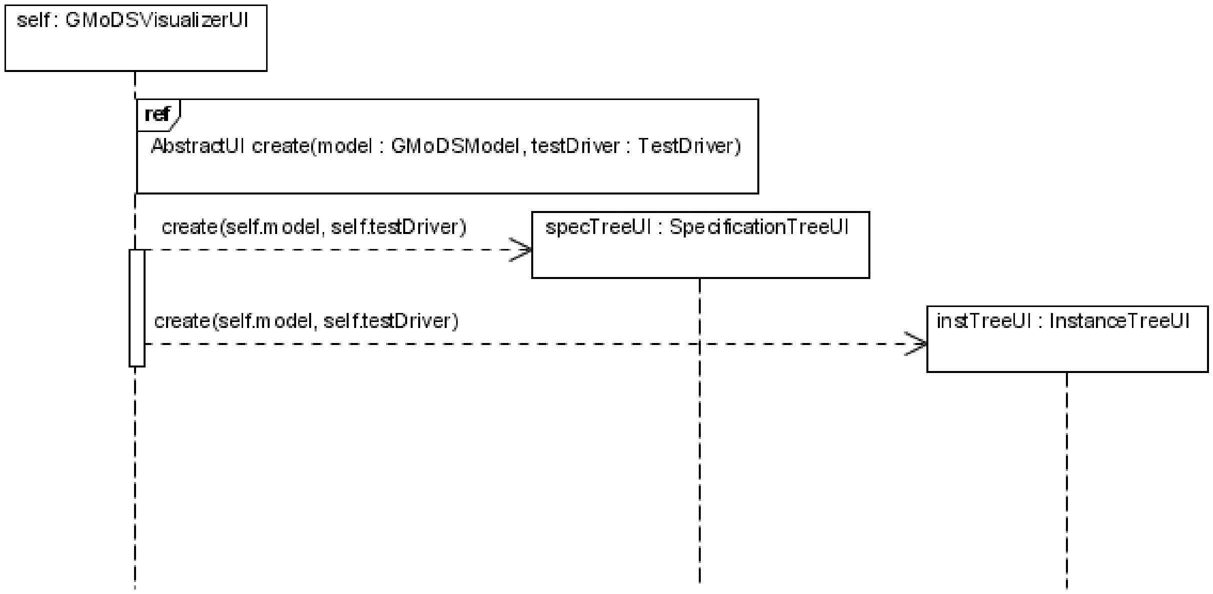
self : GMoDSVisualizerUI
ref
AbstractUI create(model : GMoDSModel, testDriver : TestDriver)
create(self.model, self.testDriver)
specTreeUI : SpecificationTreeUI
create(self.model, self.testDriver)
instTreeUI : InstanceTreeUI

Figure 9 GMoDSVisualizerUI create(model : GMoDSModel, testDriver : TestDriver)

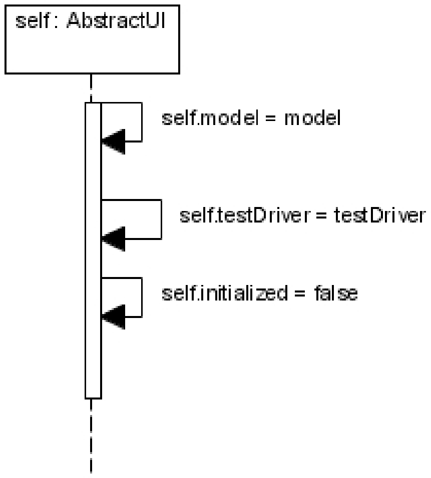

Figure 10 AbstractUI create(model : GMoDSModel, testDriver : TestDriver)

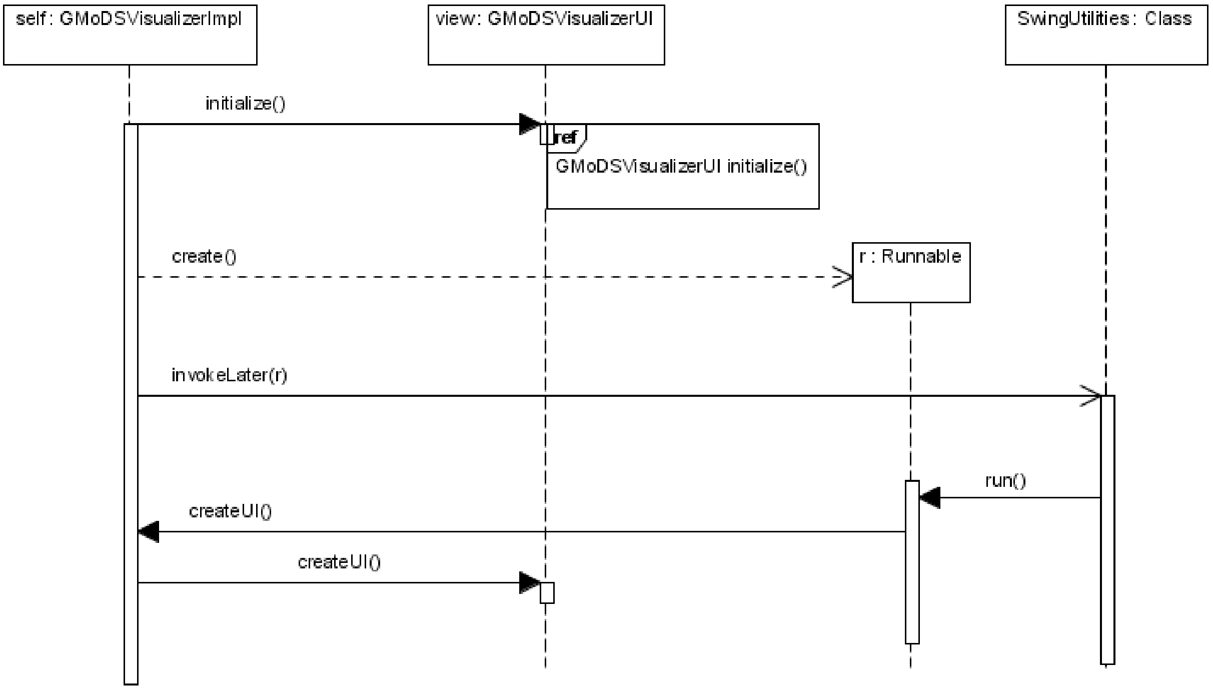

Figure 11 GMoDSVisualizerImpl initialize()

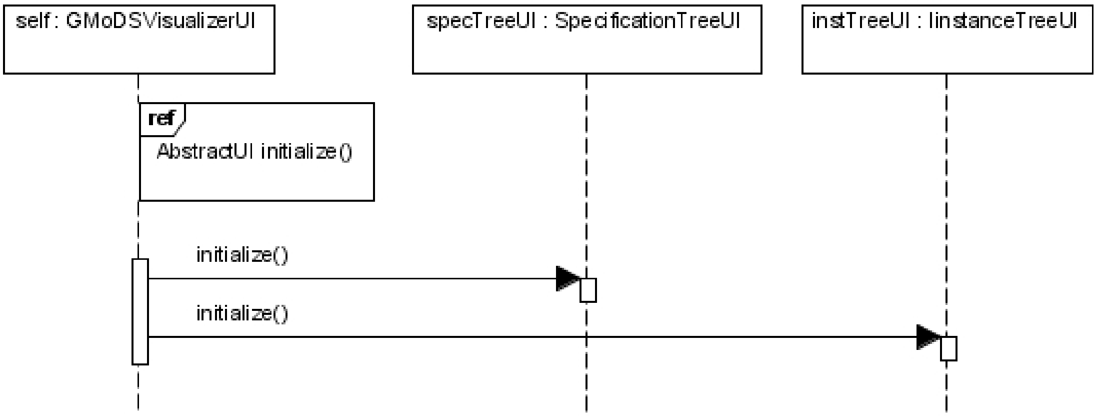

Figure 12 GMoDSVisualizerUI initialize()

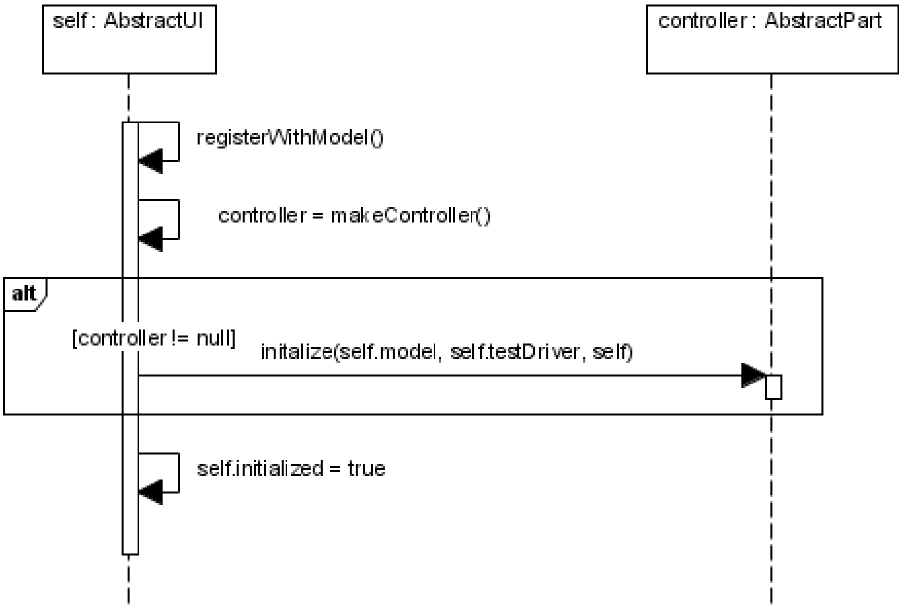
self : AbstractUI
controller : AbstractPart
registerWithModel()
controller = makeController()
alt
[controller != null]
initalize(self.model, self.testDriver, self)
self.initialized = true

Figure 13 AbstractUI initialize()

8 GMoDS Architecture

Figure 14 below documents selected GMoDS and GMoDS Test Driver classes for the sole purpose of supporting USE/OCL modeling of invariants on EventScriptImpl (a GMoDS Test Driver class). This diagram should not be taken for official GMoDS documentation. The diagram is an abstraction of the real architecture designed to make it easier to perform USE/OCL modeling. In particular, I replaced use of UniqueIdentifier with the equivalent primitive data types used for specification and instance goal identifiers. Also, GoalEventParameter, SpecificationParameter, and InstanceParameter were created to replace the use of Map data structures mapping from a parameter UniqueIdentifier to an arbitrary value Object. I omitted the SpecificationParameters class since it was not needed in any OCL invariants. Finally, the signature of "modifyInstanceGoal" was altered to include separate specification and instance goal IDs where the real signature uses a UniqueIdentifier that encapsulates both of these IDs.

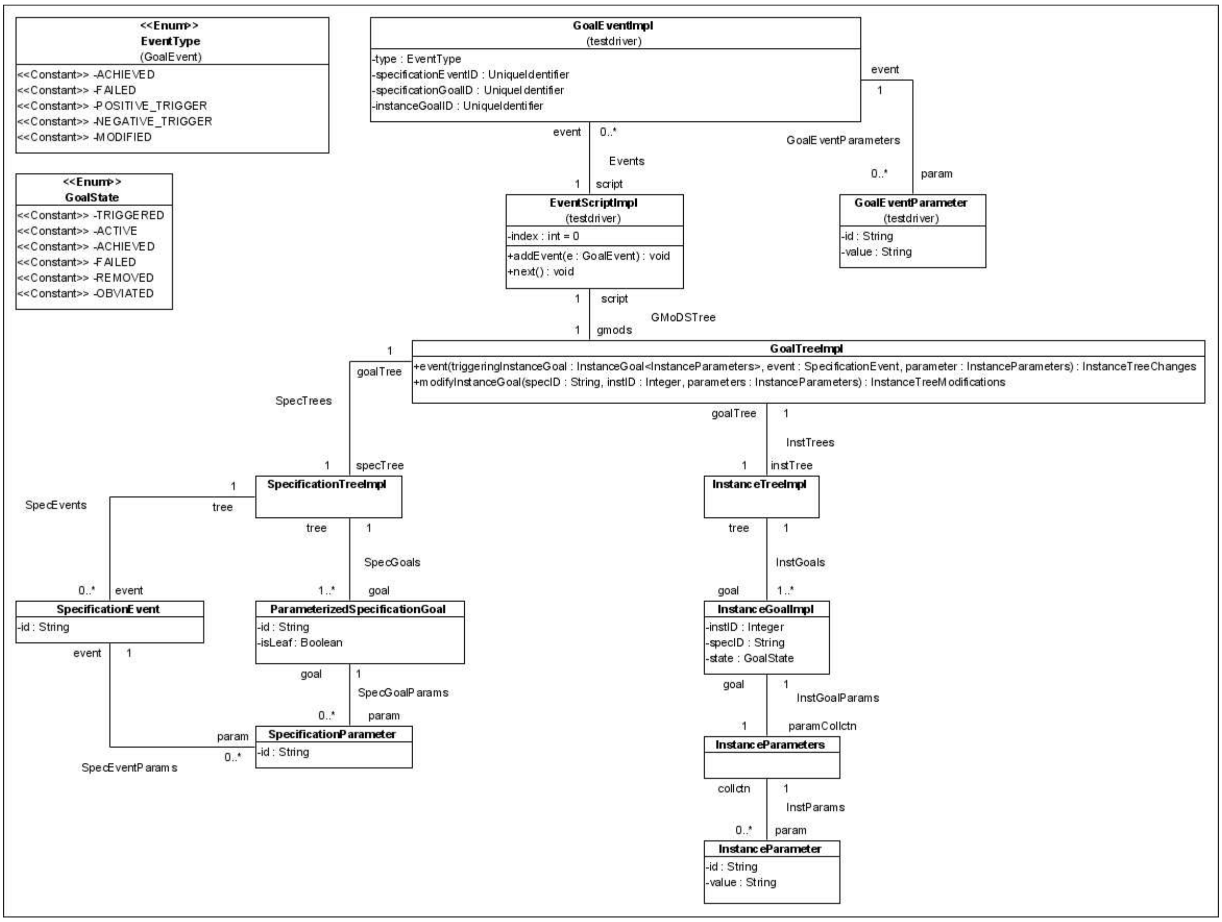

Figure 14 GMoDS and GMoDS Test Driver classes supporting formal specification

9 USE/OCL Model

```
-- GMoDS Test Driver Formal Specifications
--
-- GMoDSTestDriver.use
--
-- A formal specification of invariants maintained by EventScriptImpl addEvent
-- and next methods.
```

```
--
-- Author : Mike Fraka
-- Date: November 30, 2010
--

model GMoDSTestDriver

--
-- E N U M E R A T I O N S
--
enum EventType {ACHIEVED, FAILED, POSITIVE_TRIGGER, NEGATIVE_TRIGGER, MODIFIED}

enum GoalState {TRIGGERED, ACTIVE, ACHIEVED, FAILED, REMOVED, OBVIATED}

--
-- C L A S S E S
--
class GoalEventImpl
attributes
     type : EventType
     specEventID : String
     specGoalID : String
     instGoalID : Integer
end

class GoalEventParameter
attributes
    id : String
    value : String
end

class EventScriptImpl
attributes
    index : Integer
operations
    addEvent(e : GoalEventImpl)
    next()
end

class GoalTreeImpl
operations
    event(ig : InstanceGoal, event : SpecificationEvent, param : InstanceParameters)
    modifyInstanceGoal(specID : String, instID : String, param : InstanceParameters)
end

class SpecificationTreeImpl end

class SpecificationEvent
attributes
    id : String
end

class ParameterizedSpecificationGoal
attributes
    id : String
    isLeaf : Boolean
end

class SpecificationParameter
attributes
    id : String
end
```

```
class InstanceTreeImpl  end
class InstanceGoalImpl
attributes
    instID : Integer
    specID : String
    state : GoalState
end

class InstanceParameters end

class InstanceParameter
attributes
   id : String
   value : String
end

--
-- A S S O C I A T I O N S
--

-- GoalEventParameters: a GoalEventImpl has zero or more parameters
association GoalEventParameters between
  GoalEventImpl [1] role event
  GoalEventParameter [0..*] role param
end

-- Events: a EventScriptImpl has zero or more events
association Events between
   EventScriptImpl [1] role script
   GoalEventImpl [0..*] role event
end

-- GMoDSTree: a EventScriptImpl has 1 GoalTreeImpl
association GMoDSTree between
   EventScriptImpl [1] role script
   GoalTreeImpl [1] role gmods
end

-- SpecTrees: a GoalTreeImpl has 1 SpecificationTreeImpl
association SpecTrees between
   GoalTreeImpl [1] role goalTree
   SpecificationTreeImpl [1] role specTree
end

-- SpecEvents: a SpecificationTreeImpl has 0 or more SpecificationEvents
association SpecEvents between
   SpecificationTreeImpl [1] role tree
   SpecificationEvent [0..*] role event
end

-- SpecGoals: a SpecificationTreeImpl has 1 or more ParameterizedSpecificationGoals
association SpecGoals between
   SpecificationTreeImpl [1] role tree
   ParameterizedSpecificationGoal [1..*] role goal
end

-- SpecEventParams: a SpecificationEvent has 0 or more SpecificationParameters
association SpecEventParams between
   SpecificationEvent [1] role event
   SpecificationParameter [0..*] role param
end
```

```
-- SpecGoalParams: a ParameterizedSpecificationGoal has 0 or more
SpecificationParameters
association SpecGoalParams between
   ParameterizedSpecificationGoal [1] role goal
   SpecificationParameter [0..*] role param
end

-- InstTrees: a GoalTreeImpl has 1 InstanceTreeImpl
association InstTrees between
   GoalTreeImpl [1] role goalTree
   InstanceTreeImpl [1] role instTree
end

-- InstGoals: an InstanceTreeImpl has 1 or more InstanceGoalImpl
association InstGoals between
   InstanceTreeImpl [1] role tree
   InstanceGoalImpl [1..*] role goal
end

-- InstGoalParams: an InstanceGoalImpl has 1 InstanceParameters
association InstGoalParams between
   InstanceGoalImpl [1] role goal
   InstanceParameters [1] role paramCollctn
end

-- InstParams: an InstanceParameters has 0 or more InstanceParamter objects
association InstParams between
   InstanceParameters [1] role collec
   InstanceParameter [0..*] role param
end

--
-- C O N S T R A I N T S
--

-- The index of the event script initially points to just before the
-- first event. In Java, this is -1. In OCL, this is 0.
context EventScriptImpl
  init: index = 0

context EventScriptImpl::addEvent(e : GoalEventImpl)
  -- The event does not already exist in the script
  pre NotInScript: event->excludes(e)
  -- The added event’s type is valid
  pre ValidType:
      e.type = #ACHIEVED or e.type = #FAILED or e.type = #POSITIVE_TRIGGER or
      e.type = #NEGATIVE_TRIGGER or e.type = #MODIFIED
  -- At least one parameter must be provided if type is #MODIFIED
  pre ModifiedReqParam: if e.type = #MODIFIED implies e.param->size > 0
  -- The added event refers to a ParameterizedSpecificationGoal that
  -- exists in GMoDS’ specification tree
  pre ValidSpecGoal: gmods.specTree.goal->exists(sg | sg.id = e.specGoalID)
  -- An #ACHIEVED event will access the special ‘ACHIEVED’ event of GMoDS and
  -- must apply to a leaf specification goal.
  pre ValidAchievedEvent: if e.type = #ACHIEVED implies e.specEventID = 'ACHIEVED' and
gmods.specTree.goal->exists(sg | sg.id = e.specGoalID and sg.isLeaf = true)
  -- A #FAILED event will access the special ‘FAILED’ event of GMoDS and
  -- must apply to a leaf specification goal.
  pre ValidFailedEvent: if e.type = #FAILED implies e.specEventID = 'FAILED' and
gmods.specTree.goal->exists(sg | sg.id = e.specGoalID and sg.isLeaf = true)
  -- The added event refers to a SpecificationEvent that exists in GMoDS
  -- specification tree if the type is not #MODIFIED
  pre ValidSpecEvent: if e.type != #MODIFIED implies
        gmods.specTree.event->exists(se | se.id = e.specEventID)
  -- The event is added to the script if all preconditions are met
  post NowInScript: event->includes(e)
  -- The number of events is increased by 1
  post OneMoreEvent: (event - event@pre)->size = 1
  -- The new event is appended to the end of the script
  post Appended: event->last = e

context EventScriptImpl::next()
  -- The script must have at least 1 event
  pre HasAtLeastOneEvent: event->size > 0
  -- The script has a next event to issue to GMoDS
  pre HasNextEvent: index@pre < event->size
  -- The next event refers to an InstanceGoal that exists in GMoDS
  pre ExistingInstGoal: let nextEvt : GoalEventImpl = event->at(index@pre + 1) in
      gmods.instTree.goal->exists(ig | ig.instID = nextEvt.instGoalID and
                                       ig.specID = nextEvt.specGoalID)
  -- An event whose type is not #MODIFIED must reference
  -- an #ACTIVE InstanceGoal
  pre NotModifiedRefActiveGoal:
    let nextEvt : GoalEventImpl = event->at(index@pre + 1) in
      if nextEvt.type != #MODIFIED implies
       gmods.instTree.goal->exists(ig | ig.instID = nextEvt.instGoalID and
ig.specID = nextEvt.specGoalID and ig.state = #ACTIVE)
  -- If the next event type is #NEGATIVE_TRIGGER then all of its parameter
  -- values must match an existing instance goal's parameter values
  pre ValidNegativeTrigger:
    let nextEvt : GoalEventImpl = event->at(index@pre + 1) in
      if nextEvt.type = #NEGATIVE_TRIGGER and nextEvt.param->size > 0 implies
       gmods.instTree.goal->exists(ig | ig.instID = nextEvt.instGoalID and
ig.specID = nextEvt.specGoalID
       and nextEvt.param->forAll( nep | ig.paramCollctn.param->exists(igp |
igp.id = nep.id and igp.value = nep.value)))
  -- Advance the script index
  post ScriptIndexAdvanced: index = index@pre + 1
  -- If preconditions met and the next event is not #MODIFIED then
  -- the ‘event’ message is sent to GMoDS with appropriate parameter values.
  post NotModifiedSendsEvent:
     let nextEvt : GoalEventImpl = event->at(index@pre + 1)
         in
        if nextEvt.type != #MODIFIED implies
           let instGoal : InstanceGoal =
                gmods.instTree.goal->select(ig | ig.instID = nextEvt.instGoalID and
ig.specID = nextEvt.specGoalID)->first,
               specEvt : SpecificationEvent =
                gmods.specTree.event->select(se | se.id = nextEvt.specEventID)->first,
               instParams : InstanceParameters,
               newInstParams : Boolean = instParams.oclIsNew() and
                nextEvt.param->forAll(np | instParams.param->exists(ip | ip.oclIsNew()
and ip.id = np.id and ip.value = np.value))
           in
             newInstParams and gmods^event(instGoal, specEvt, instParams)
  -- If preconditions are met and the next event is #MODIFIED then the
  -- ‘modifyInstanceGoal’ message is sent to GMoDS with appropriate parameter values.
  post ModifiedSendsModifyInstanceGoal:
     let nextEvt : GoalEventImpl = event->at(index@pre + 1)
         in
        if nextEvt.type = #MODIFIED implies
           let instParams : InstanceParameters,
               newInstParams : Boolean = instParams.oclIsNew() and
```

```
                nextEvt.param->forAll(np | instParams.param->exists(ip | ip.oclIsNew()
and ip.id = np.id and ip.value = np.value))
           in
              newInstParams and
    gmods^modifyInstanceGoal(nextEvt.specGoalID, nextEvt.instGoalID, instParams)
```