System Architectural Design

For GMoDS Visualizer and Test Driver

Version 2.0

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 ........................................................................................................................................... 4
2 References ............................................................................................................................................... 4
3 System Context ....................................................................................................................................... 4
4 System Architecture ............................................................................................................................. 5
  4.1 System Components .............................................................................................................................. 5
  4.2 System Component Responsibilities ..................................................................................................... 6
  4.3 System Interface Specifications .......................................................................................................... 6
  4.4 System Architecture Design Rationale ............................................................................................... 9
5 GMoDS Test Driver Architecture ........................................................................................................ 10
  5.1 GMoDS Test Driver Decomposition .................................................................................................. 10
    5.1.1 GMoDS Test Driver Module Responsibilities .............................................................................. 10
    5.1.2 GMoDS Test Driver Interface Specifications ............................................................................. 11
    5.1.3 GMoDS Test Driver Design Rationale ....................................................................................... 14
6 GMoDS Visualizer Architecture .......................................................................................................... 14
  6.1 GMoDS Visualizer Model Decomposition .......................................................................................... 14
    6.1.1 GMoDS Visualizer Model Module Responsibilities .................................................................... 14
    6.1.2 GMoDS Visualizer Model Interface Specifications .................................................................... 15
    6.1.3 GMoDS Visualizer Model Design Rationale .............................................................................. 16
  6.2 GMoDS Visualizer View Decomposition ............................................................................................ 17
    6.2.1 GMoDS Visualizer View Module Responsibilities ....................................................................... 17
    6.2.2 GMoDS Visualizer View Interface Specifications ....................................................................... 18
    6.2.3 GMoDS Visualizer View Design Rationale .................................................................................. 22
  6.3 GMoDS Visualizer Controller Decomposition ................................................................................... 23
    6.3.1 GMoDS Visualizer Controller Module Responsibilities ............................................................... 23
    6.3.2 GMoDS Visualizer Controller Interface Specifications ............................................................... 23
    6.3.3 GMoDS Visualizer Controller Design Rationale ......................................................................... 24
7 System Startup Behavior ...................................................................................................................... 24
8 GMoDS Architecture ............................................................................................................................ 28
9 USE/OCL Model ..................................................................................................................................... 28
<table>
<thead>
<tr>
<th>Change</th>
<th>Section, Figure, or Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replaced references to DeterministicEventScript with EventScriptImpl</td>
<td>Throughout</td>
<td>Throughout</td>
</tr>
<tr>
<td>Added Offspring association between ParameterizedSpecificationGoal and itself.</td>
<td>Figure 14</td>
<td>28</td>
</tr>
<tr>
<td>Added “{ordered}” to Events association.</td>
<td>Figure 14</td>
<td>28</td>
</tr>
<tr>
<td>Updated USE/OCL model based on feedback from USE.</td>
<td>Section 9</td>
<td>28</td>
</tr>
</tbody>
</table>
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

3 System Context
The system context is shown in Figure 1 below.

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 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 2 System component responsibilities**

<table>
<thead>
<tr>
<th>Component</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMoDS</td>
<td>Provide the core objects and behaviors to be visualized. Provide pull and push access to these core objects.</td>
</tr>
<tr>
<td>GMoDS Visualizer</td>
<td>Provide the user interface for visualizing GMoDS object behaviors. Provide the user interface controls for the GMoDS Test Driver if configured.</td>
</tr>
<tr>
<td>GMoDS Test Driver</td>
<td>Provide the capability to test the GMoDS Visualizer in manual and automatic mode.</td>
</tr>
</tbody>
</table>

### 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 3 GMoDS Test Driver Launcher interface specifications**

<table>
<thead>
<tr>
<th>Launch the GMoDS Test Driver for a specific goal diagram.</th>
<th>Syntax: <code>main(args : string[]) : void</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: <code>args.length = 1</code></td>
<td></td>
</tr>
<tr>
<td>Pre: <code>args[0] is the goal diagram file name.</code></td>
<td></td>
</tr>
<tr>
<td>Pre: <code>args[0] is a file that exists and is readable.</code></td>
<td></td>
</tr>
<tr>
<td>Post: <code>The GMoDS component is created, initialized, and passed to the GMoDSTestDriverImpl and GMoDSVisualizerImpl.</code></td>
<td></td>
</tr>
<tr>
<td>Post: <code>The GMoDSTestDriverImpl is created and passed to the GMoD Visualizer component.</code></td>
<td></td>
</tr>
<tr>
<td>Post: <code>The GMoDSVisualizerImpl is created and initialized. The user interface is created, initialized, and made visible.</code></td>
<td></td>
</tr>
</tbody>
</table>
Table 4 GMoDSVisualizer interface specifications

<table>
<thead>
<tr>
<th>Initialize the GMoDS Visualizer resulting in a visible, ready user interface.</th>
<th>Syntax:</th>
<th>initialize() : void</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre:</td>
<td>GMoDS GoalTree implementation != null.</td>
<td></td>
</tr>
<tr>
<td>Pre:</td>
<td>GMoDS GoalTree implementation is initialized.</td>
<td></td>
</tr>
<tr>
<td>Post:</td>
<td>The GMoDSVisualizerImpl is initialized. The user interface is created, initialized, and made visible.</td>
<td></td>
</tr>
</tbody>
</table>

Table 5 Test Driver interface specifications

<table>
<thead>
<tr>
<th>Add an Observer of the event script (as in the Observer design pattern).</th>
<th>Syntax:</th>
<th>addObserver(o : Observer) : void</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre:</td>
<td>o != null.</td>
<td></td>
</tr>
<tr>
<td>Post:</td>
<td>An Observer o is recorded and will be notified whenever the state of the EventScript changes.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Load an event script XML file into the TestDriver.</th>
<th>Syntax:</th>
<th>loadEventScript(eventScript : File) : void</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre:</td>
<td>eventScript != null.</td>
<td></td>
</tr>
<tr>
<td>Pre:</td>
<td>eventScript File exists, is a File, and can be read.</td>
<td></td>
</tr>
<tr>
<td>Post:</td>
<td>An EventScriptImpl is created from the eventScript File.</td>
<td></td>
</tr>
<tr>
<td>Post:</td>
<td>All valid GoalEvents specified in eventScript are included in the EventScriptImpl</td>
<td></td>
</tr>
<tr>
<td>Post:</td>
<td>The TestDriver enters manual mode.</td>
<td></td>
</tr>
<tr>
<td>Post:</td>
<td>All invalid GoalEvents are discarded and the user is notified visually and in a log file of discarded GoalEvents.</td>
<td></td>
</tr>
<tr>
<td>Save the current event script as an XML file.</td>
<td>Syntax: <code>saveEventScript(eventScript : File) : void</code></td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Pre: TestDriver is in manual mode.</td>
<td>Pre: <code>eventScript != null</code>.</td>
<td></td>
</tr>
<tr>
<td>Pre: User must have permission to write the eventScript File.</td>
<td>Pre: If eventScript File exists then user must confirm that it will be overwritten.</td>
<td></td>
</tr>
<tr>
<td>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).</td>
<td>Post: The TestDriver remains in manual mode.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Begin issuing random events using the current random event configuration parameters.</th>
<th>Syntax: <code>issueRandomEvents() : void</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: None.</td>
<td>Post: A RandomEventScriptImpl is created using the RandomEventParameters in effect during the method call.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Place the TestDriver in automatic mode.</th>
<th>Syntax: <code>play() : void</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: TestDriver is in manual mode.</td>
<td>Pre: TestDriver has a next GoalEvent it can issue.</td>
</tr>
<tr>
<td>Pre: TestDriver has a next GoalEvent it can issue.</td>
<td>Post: The TestDriver enters automatic mode.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Place the TestDriver in manual mode.</th>
<th>Syntax: <code>pause() : void</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: TestDriver is in automatic mode.</td>
<td>Pre: TestDriver has a next GoalEvent it can issue.</td>
</tr>
<tr>
<td>Pre: TestDriver has a next GoalEvent it can issue.</td>
<td>Post: The TestDriver enters manual mode.</td>
</tr>
</tbody>
</table>
### 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

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 6 GMoDS Test Driver module responsibilities

<table>
<thead>
<tr>
<th>Component</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launcher</td>
<td>Configure GMoDS, GMoDSTestDriverImpl, and the GMoDSVisualizerImpl. Initialize GMoDS and the GMoDSVisualizerImpl.</td>
</tr>
<tr>
<td>GMoDSTestDriverImpl</td>
<td>Hold an EventScript. Implement loadEventScript and issueRandomEvents to create and install EventScriptImpl and RandomEventScriptImpl, respectively.</td>
</tr>
<tr>
<td>EventScript</td>
<td>Define the behaviors of any EventScript.</td>
</tr>
<tr>
<td>EventScriptImpl</td>
<td>Hold the list of GoalEvents defining the script and provide default implementations of the EventScript interface.</td>
</tr>
</tbody>
</table>
Component Responsibilities

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 7 GMoDS Test Driver GoalEvent interface specifications

<table>
<thead>
<tr>
<th>Access the EventType of a GoalEvent.</th>
<th>Syntax: getType() : EventType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: None.</td>
<td>Post: Result = this.eventType</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access the UniqueIdentifier of the specification goal referenced by a GoalEvent.</th>
<th>Syntax: getSpecificationGoalIdentifier() : UniqueIdentifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: None.</td>
<td>Post: Result = this.specificationGoalID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access the UniqueIdentifier of the instance goal referenced by a GoalEvent.</th>
<th>Syntax: getInstanceGoalIdentifier() : UniqueIdentifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: None.</td>
<td>Post: Result = this.instanceGoalID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access the UniqueIdentifier of the SpecificationEvent referenced by a GoalEvent.</th>
<th>Syntax: getEventGoalIdentifier() : UniqueIdentifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: this.eventType = EventType.POSITIVE_TRIGGER or this.eventType = EventType.NEGATIVE_TRIGGER</td>
<td>Post: Result = this.eventID</td>
</tr>
</tbody>
</table>
Table 8 GMoDS Test Driver EventScript interface specifications

<table>
<thead>
<tr>
<th>Action Description</th>
<th>Syntax:</th>
<th>Pre:</th>
<th>Post:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add an event valid with respect to the GMoDS specification tree to the end of the script.</td>
<td>addEvent(e : GoalEvent) : void</td>
<td>e != null</td>
<td>(events – events@pre)-&gt;size() = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e is not already included in the script.</td>
<td>events.includes(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e.type is valid.</td>
<td>events.last() = e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if e.type = #MODIFIED then at least one parameter must be provided for the event.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e.getSpecificationGoalIdentifier() refers to a specification goal that exists in the specification tree.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>if e.type = #ACHIEVED then e.getSpecificationGoalIdentifier() = ‘ACHIEVED’ and the specification goal is a leaf.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>if e.type = #FAILED then e.getSpecificationGoalIdentifier() = ‘FAILED’ and the specification goal is a leaf.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>if e.type != #MODIFIED then e.getSpecificationEventIdentifier() refers to a specification event defined in the specification tree.</td>
<td></td>
</tr>
<tr>
<td>Place the EventScript in automatic mode.</td>
<td>play() : void</td>
<td>EventScript is in manual mode.</td>
<td>The EventScript enters automatic mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EventScript has a next GoalEvent it can issue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Syntax</td>
<td>Pre Conditions</td>
<td>Post Conditions</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Place the EventScript in manual mode.</td>
<td>pause() : void</td>
<td>EventScript is in automatic mode.</td>
<td>The EventScript enters manual mode.</td>
</tr>
<tr>
<td>Issue the next event to GMoDS.</td>
<td>next() : void</td>
<td>EventScript is in manual mode.</td>
<td>If the next GoalEvent type != #MODIFIED then the next event refers to a valid active instance goal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EventScript has at least 1 event.</td>
<td>If the next GoalEvent type != #MODIFIED the EventScript issues the next GoalEvent to the GMoDS event method.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EventScript has a next GoalEvent it can issue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The next GoalEvent refers to a valid instance goal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If next GoalEvent type != #MODIFIED then the next event refers to a valid active instance goal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pre: The next GoalEvent refers to a valid instance goal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post: If the next GoalEvent type != #MODIFIED the EventScript issues the next GoalEvent to the GMoDS event method.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post: If the next GoalEvent type = #MODIFIED the EventScript issues the next GoalEvent to the GMoDS modifyInstanceGoal method.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post: The EventScript index refers to the next event if one exists.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post: The EventScript remains manual mode.</td>
<td></td>
</tr>
<tr>
<td>Determine if the EventScript has a next event to issue to GMoDS.</td>
<td>hasNext() : boolean</td>
<td>None.</td>
<td>Result = EventScript has a next valid GoalEvent that can be issued to GMoDS.</td>
</tr>
<tr>
<td>Add an Observer of the event script (as in the Observer design pattern).</td>
<td>addObserver(o : Observer) : void</td>
<td>o != null.</td>
<td>An Observer o is recorded and will be notified whenever the state of the EventScript changes.</td>
</tr>
</tbody>
</table>
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

![GMoDS Visualizer model modules](image)

Figure 4 GMoDS Visualizer model modules

6.1.1 GMoDS Visualizer Model Module Responsibilities

Table 9 GMoDS Visualizer model module responsibilities

<table>
<thead>
<tr>
<th>Component</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoalState</td>
<td>Enumeration of possible goal states.</td>
</tr>
<tr>
<td>GMoDSModel</td>
<td>Define methods for access and evaluation of the core GMoDS objects.</td>
</tr>
<tr>
<td>GMoDSModelImpl</td>
<td>Implement methods for access and evaluation of the core GMoDS objects.</td>
</tr>
</tbody>
</table>
### 6.1.2 GMoDS Visualizer Model Interface Specifications

Table 10 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 10 GMoDS Visualizer GMoDSModel interface specifications**

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

Figure 5 GMoDS Visualizer view modules

6.2.1 GMoDS Visualizer View Module Responsibilities

Table 11 GMoDS Visualizer view module responsibilities

<table>
<thead>
<tr>
<th>Component</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractUI</td>
<td>Define the basic behaviors and responsibilities of the view role. Hold references to the core model and TestDriver if present.</td>
</tr>
<tr>
<td>AbstractCanvas</td>
<td>Hold the Java 2D image upon which a diagram is drawn. Define the methods concrete canvases must support.</td>
</tr>
<tr>
<td>GMoDSVisualizerUI</td>
<td>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.</td>
</tr>
<tr>
<td>SpecificationTreeUI</td>
<td>Define the UI for the specification tree. Provide zoom and scroll controls for the specification tree.</td>
</tr>
<tr>
<td>SpecificationTreeCanvas</td>
<td>Draw the specification tree.</td>
</tr>
<tr>
<td>SpecificationGoalUI</td>
<td>Define the UI for a specification goal.</td>
</tr>
</tbody>
</table>
## Component Responsibilities

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

### 6.2.2 GMoDS Visualizer View Interface Specifications

**Table 12 GMoDS Visualizer AbstractUI interface specifications**

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

---

**Table 13 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 14 GMoDS Visualizer SpecificationTreeCanvas interface specifications

<table>
<thead>
<tr>
<th>Add an AbstractRelationUI to the list of relations to draw.</th>
<th>Syntax: <code>addRelationUI(relationUI : AbstractRelationUI) : void</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: None.</td>
<td>Post: The relationUI is added to the list of relationUIs drawn on the canvas.</td>
</tr>
<tr>
<td>Draw the AbstractRelationUIs on the image.</td>
<td>Syntax: <code>drawRelations() : void</code></td>
</tr>
<tr>
<td>Pre: None.</td>
<td>Post: All AbstractRelationUIs are drawn on the canvas.</td>
</tr>
</tbody>
</table>

Table 15 GMoDS Visualizer SpecificationGoalUI interface specifications

<table>
<thead>
<tr>
<th>Draw the SpecificationGoalUI and its descendants on the image.</th>
<th>Syntax: <code>drawTree(graphics2D : Graphics2D) : void</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: None.</td>
<td>Post: This SpecificationGoalUI and its descendants are drawn on the Java 2D image.</td>
</tr>
</tbody>
</table>

Table 16 GMoDS Visualizer InstanceTreeUI interface specifications

<table>
<thead>
<tr>
<th>Begin flashing added or changed InstanceGoalUIs.</th>
<th>Syntax: <code>update (o : Observable, arg : Object) : void</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre: The Observable o (the model) has changed state.</td>
<td>Post: Added or changed InstanceGoalUIs begin to flash.</td>
</tr>
<tr>
<td>Draw and repaint the canvas.</td>
<td>Syntax: <code>draw() : void</code></td>
</tr>
<tr>
<td>Pre: None.</td>
<td>Post: The canvas held by this view is redrawn and repainted to allow dynamic changes to appear.</td>
</tr>
</tbody>
</table>
Table 17 GMoDS Visualizer InstanceTreeCanvas interface specifications

<table>
<thead>
<tr>
<th>Method Description</th>
<th>Syntax:</th>
<th>Pre:</th>
<th>Post:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create all added InstanceGoalUIs.</td>
<td>createGoalUIs() : void</td>
<td>None</td>
<td>This canvas creates all added InstanceGoalUIs and assures they are ordered, assessed, and registered for later display.</td>
</tr>
<tr>
<td>Get the specified InstanceGoalUI.</td>
<td>get(instanceGoalID : UniqueIdentifier) : InstanceGoalUI</td>
<td>None</td>
<td>Result = the InstanceGoalUI specified by the instanceGoalID.</td>
</tr>
</tbody>
</table>

Table 18 GMoDS Visualizer InstanceGoalUI interface specifications

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

Figure 6 GMoDS Visualizer controller modules

6.3.1 GMoDS Visualizer Controller Module Responsibilities

Table 20 GMoDS Visualizer controller module responsibilities

<table>
<thead>
<tr>
<th>Component</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractPart</td>
<td>Define basic methods for setting up a controller associated with its view, model, and TestDriver.</td>
</tr>
</tbody>
</table>

6.3.2 GMoDS Visualizer Controller Interface Specifications

Table 21 GMoDS Visualizer AbstractPart interface specifications

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre:</td>
<td>None.</td>
</tr>
<tr>
<td>Post:</td>
<td>This controller is initialized with references to the model, view, and TestDriver.</td>
</tr>
</tbody>
</table>
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.

![Figure 7 GMoDS Test Driver Launcher main program behavior](image)

![Figure 8 GMoDSVisualizerImpl create(goalTree : GoalTree, testDriver : TestDriver)](image)
Figure 9 GMoDSVisualizerUI create(model : GMoDSModel, testDriver : TestDriver)

Figure 10 AbstractUI create(model : GMoDSModel, testDriver : TestDriver)
Figure 11 GMoDSVisualizerImpl initialize()

Figure 12 GMoDSVisualizerUI initialize()
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

--
-- ENUMERATIONS
--
enum EventType {ACHIEVED, FAILED, POSITIVE_TRIGGER, NEGATIVE_TRIGGER, MODIFIED}
enum GoalState {TRIGGERED, ACTIVE, ACHIEVED, FAILED, REMOVED, OBVIATED}

--
-- CLASSES
--
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 : InstanceGoalImpl, event : SpecificationEvent, param : InstanceParameters)
    modifyInstanceGoal(specID : String, instID : Integer, param : InstanceParameters)
end
class SpecificationTreeImpl end

class SpecificationEvent
attributes
  id : String
  declaredGoalID : String
end

class ParameterizedSpecificationGoal
attributes
  id : String
  isLeaf : Boolean
operations
  closureChildren(s : Set(ParameterizedSpecificationGoal)) :
  Set(ParameterizedSpecificationGoal) =
  if s->includesAll(s.child->asSet()) then s
  else closureChildren(s->union(s.child->asSet()))
  endif
  descendantsAndSelf() : Set(ParameterizedSpecificationGoal) =
  closureChildren(Set{self})
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
-- ASSOCIATIONS

-- 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
-- and a GoalEventImpl is associated with zero or one script.
association Events between
  EventScriptImpl [0..1] role script
  GoalEventImpl [0..*] role event ordered
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

-- Offspring: a ParameterizedSpecificationGoal has 0 or 1 parents
and
-- 0 or more children
association Offspring between
  ParameterizedSpecificationGoal [0..1] role parent
  ParameterizedSpecificationGoal [0..*] role child
end
-- SpecEventParams: a SpecificationEvent has 0 or more SpecificationParameters association SpecEventParams between SpecificationEvent [0..1] role event SpecificationParameter [0..*] role param end

-- SpecGoalParams: a ParameterizedSpecificationGoal has 0 or more SpecificationParameters association SpecGoalParams between ParameterizedSpecificationGoal [0..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 paramColctn end

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

--

-- CONSTRAINTS
--

constraints

-- The index of the event script initially points to just before the first event.
-- In Java, this is -1.
-- USE 2.6.2 does not support this legal OCL syntax
-- context EventScriptImpl::index
--    init: 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: e.type = #MODIFIED implies e.param->size > 0
  -- A #MODIFIED event's parameter names must match specification goal's
  parameter names
  pre ValidModifiedParamNames:
    e.type = #MODIFIED and e.param->size > 0 implies
    e.param->forAll(ep | gmods.specTree.goal->exists(sg | sg.id = e.specGoalID and
    sg.param->exists(sgp | sgp.id = ep.id)))
  -- 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:
    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:
    e.type = #FAILED implies e.specEventID = 'FAILED' and
    gmods.specTree.goal->exists(sg | sg.id = e.specGoalID and
    sg.isLeaf = true)
  -- If the type is #POSITIVE_TRIGGER or #NEGATIVE_TRIGGER
  -- the added event refers to a SpecificationEvent that exists in GMoDS
  specification tree,
  -- the event's specification goal is a leaf goal, the event's
  specification event's
  -- declared goal exists, and the event's specification goal is either
  the goal on which the
  -- event was declared or a descendant of the declared goal.
  pre ValidSpecEvent:
    e.type = #POSITIVE_TRIGGER or e.type = #NEGATIVE_TRIGGER implies
    (gmods.specTree.event->exists(se | se.id = e.specEventID and
    gmods.specTree.goal->exists(sg,dg |
    sg.isLeaf = true and sg.id = e.specGoalID and dg.id =
    se.declaredGoalID and dg.descendantsAndSelf()->includes(sg))))
-- if the type is #POSITIVE_TRIGGER or #NEGATIVE_TRIGGER
-- then it must provide the parameters required by the specification
-- event
    pre ValidTriggerParamNames:
        e.type = #POSITIVE_TRIGGER or e.type = #NEGATIVE_TRIGGER implies
        (gmods.specTree.event->exists(se | se.id = e.specEventID and
            se.param->forAll(sep | e.param->exists(ep | ep.id = sep.id))))
-- 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->asSet - event@pre->asSet)->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 < event->size
-- The next event refers to an InstanceGoal that exists in GMoDS
    pre ValidInstGoal: let nextEvt : GoalEventImpl = event->at(index + 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 + 1) in
            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 + 1) in
            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 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
            (nextEvt.type <> #MODIFIED implies
(let
    instParams : InstanceParameters =
    gmods.instTree.goal->any(ig | ig.instID = nextEvt.instGoalID and
                              ig.specID = nextEvt.specGoalID).paramCollection
-- USE 2.6.2 would not accept these additional local variable
declarations needed to specify the message sent in the next method.
--
-- instGoal : InstanceGoalImpl =
-- gmods.instTree.goal->any(ig | ig.instID = nextEvt.instGoalID and
--                         ig.specID = nextEvt.specGoalID),
-- specEvt : SpecificationEvent =
-- gmods.specTree.event->any(se | se.id = nextEvt.specEventID)
in
    instParams.oclIsNew() and
nextEvt.param->forAll(np | instParams.param->exists(ip | ip.oclIsNew() and
                                   ip.id = np.id and ip.value = np.value))
-- USE 2.6.2 does not appear to support the "isSent" operator denoted
by "^" in Warmer and Kleppe "The Object Constraint Language", 2nd
-- 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
    (nextEvt.type = #MODIFIED implies
      (let
        instParams : InstanceParameters =
        gmods.instTree.goal->any(ig | ig.instID =
                                   nextEvt.instGoalID and
                                   ig.specID =
        nextEvt.specGoalID).paramCollection
        in
        instParams.oclIsNew() and
        nextEvt.param->forAll(np | instParams.param->exists(ip | ip.oclIsNew() and
                                                        ip.id = np.id and
                                                        ip.value = np.value))
-- USE 2.6.2 does not appear to support the "isSent" operator denoted
by "^" in
-- Warmer and Kleppe "The Object Constraint Language", 2nd Edition,
-- and gmods^modifyInstanceGoal(nextEvt.specGoalID,
nextEvt.instGoalID, instParams)
)