GMoDS Test Driver and Visualizer
MSE Project Presentation II

Mike Fraka
Agenda

- Action Items
- Project Plan
- Architectural Design
- Formal Specification
- Technical Inspection Checklist
- Test Plan
- Top Technical Risks
- Architectural Prototype Demonstration
- Questions/Comments
Action Items (from Phase 1)

- No action items
# Project Plan – Productivity

<table>
<thead>
<tr>
<th>Activity Type</th>
<th>Project Time (hours)</th>
<th>Quantity</th>
<th>Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Code</td>
<td>85.7</td>
<td>5000 SLOC</td>
<td>58.4 SLOC/Hr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(51 of 72 req. ~ 71% complete)</td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>68.5</td>
<td>11 Documents</td>
<td>0.16 Doc./Hr</td>
</tr>
<tr>
<td>Reading/Research/Misc.</td>
<td>24.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>178.7</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
Project Plan – Phase 3 Deliverables

- Action items.
- User manual.
- Component design.
- Source code.
- Assessment evaluation.
- Project evaluation.
- References.
- Technical inspection letters.
## Project Plan – Cost Estimate

<table>
<thead>
<tr>
<th>Item</th>
<th>Estimate</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Code To Be Developed</td>
<td>2000 SLOC remain</td>
<td>5000/0.71 ~ 7000 total SLOC</td>
</tr>
<tr>
<td>Source Code Time</td>
<td>36 hours (18 days @ 2 hrs. per day)</td>
<td>2000/58.4 ~ 35.3 hours</td>
</tr>
<tr>
<td>Testing Time</td>
<td>28 hours (14 days)</td>
<td></td>
</tr>
<tr>
<td>Documents Remaining</td>
<td>5 major documents</td>
<td>Including testing docs.</td>
</tr>
<tr>
<td>Documentation Time</td>
<td>32 hours (16 days)</td>
<td>5/0.16 ~ 31.25 hours</td>
</tr>
<tr>
<td>Totals</td>
<td>48 days</td>
<td></td>
</tr>
</tbody>
</table>
Architectural Design – Components
Architectural Design – Visualizer Model

<<Interface>>
GMoUSModel

+ addObserver(o : Observer) : void
+ isAncestorPrecededSpecificationGoal(identifier : UniqueIdentifier) : boolean
+ isAncestorPositiveTriggeredSpecificationGoal(identifier : UniqueIdentifier) : boolean
+ isAncestorNegativeTriggeredSpecificationGoal(identifier : UniqueIdentifier) : boolean
+ hasSameParents(identifier1 : UniqueIdentifier, identifier2 : UniqueIdentifier) : boolean
+ assessState(identifier : UniqueIdentifier) : GoalState

GMoUSModelImpl

<<Interface>>
InstanceTree

(education : cis : macr : goal : model)

<<Interface>>
SpecificationTree

(education : cis : macr : goal : model)

<<Interface>>
GoalTree

(education : cis : macr : goal : model)

<<Enum>>
GoalState

<<Constant>> TRIGGERED
<<Constant>> ACTIVE
<<Constant>> ACHIEVED
<<Constant>> FAILED
<<Constant>> REMOVED
<<Constant>> OBVIATED
Architectural Design – Visualizer Part

```
<<Interface>>
Observer
update(Observable, arg: Object): void

<<Interface>>
AbstractPart
#testDriver: TestDriver
#initialize(model: GMoDSModel, testDriver: TestDriver, abstractUI: AbstractUI): void
#registerWithModel(): void

<<Interface>>
AbstractUI
#controller 1
#initialized: boolean
createUI(): void
registerController(): AbstractPart
registerWithModel(): void
initialize(): void

<<Interface>>
GMoDSModel
#addObserver(Observable): void
#acceptsPrecedeSpecificationGoalIdentifier: UniqueIdentifier: boolean
#acceptsPositiveTriggeredSpecificationGoalIdentifier: UniqueIdentifier: boolean
#acceptsNegativeTriggeredSpecificationGoalIdentifier: UniqueIdentifier: boolean
#acceptsGotoStateLinkIdentifier1: UniqueIdentifier, Identifier2: UniqueIdentifier: boolean
#haveSameParentIdentifier1: UniqueIdentifier, Identifier2: UniqueIdentifier: boolean
#assocState(Identifier: UniqueIdentifier): GoalState
```
Architectural Design – Visualizer UI
Architectural Design – Startup
Formal Specification

-- CONSTRUCTOR

-- The index of the event script initially points to just before the first event.
-- In Java, this is -1.
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 = ANEGATIVE_TRIGGER or e.type = UNMODIFIED
-- At least one parameter must be provided if type is UNMODIFIED
  pre UnmodifiedReqParam: if e.type = UNMODIFIED then e.params->size > 0
-- The added event refers to a ParameterizedSpecificationGoal that
-- exists in GNOME's specification tree
  pre ValidSpecGoal: gnome.specTree.goal->exists(sg | sg.id = e.specGoalID)
-- An ACHIEVED event will access the special 'ACHIEVED' event of GNOME and
-- must apply to a leaf specification goal.
  pre ValidAchievedEvent: if e.type = ACHIEVED then e.specEventID = 'ACHIEVED' and
gnome.specTree.goal->exists(sg | sg.id = e.specGoalID and sg.isLeaf = true)
-- A FAILED event will access the special 'FAILED' event of GNOME and
-- must apply to a leaf specification goal.
  pre ValidFailedEvent: if e.type = FAILED then e.specEventID = 'FAILED' and
gnome.specTree.goal->exists(sg | sg.id = e.specGoalID and sg.isLeaf = true)
-- The added event refers to a SpecificationEvent that exists in GNOME
-- Specification tree if the type is not UNMODIFIED
  pre ValidSpecification: if e.type != UNMODIFIED then gnome.specTree.event->exists(se | se.id = e.specEventID)
-- The event is added to the script if all preconditions are met
  post NowInScript: event->includes(c)
-- 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
Formal Specification

context EventScriptImpl::next(
  -- The script must have at least 1 event
  -- The script has a next event to issue to GMod)
pre Not improbable
  -- The next event is a member of the script

post ValidInstGoal:
  let nextEvt : GoalEventImpl = event->at(index$pre + 1) in
  gmods.instanceTree .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 NotModifiedNextActiveGoal:
  let nextEvt : GoalEventImpl = event->at(index$pre + 1) in
  if nextEvt .type != #MODIFIED implies
  gmods.instanceTree .goal->exists([ig | ig .instID = nextEvt .instGoalID and ig .specID = nextEvt .specGoalID])
  -- If the next event type is #NEGATIVE_TRIGGER then all of its parameters
  -- values must match an existing instance goal's parameter values
  pre ValidateNegativeTrigger:
    let nextEvt : GoalEventImpl = event->at(index$pre + 1) in
    if nextEvt .type = #NEGATIVE_TRIGGER and nextEvt .param->size > 0 implies
    gmods.instanceTree .goal->exists([ig | ig .instID = nextEvt .instGoalID and ig .specID = nextEvt .specGoalID])
    and nextEvt .param->forall([ap | ap .paramColl .param->exists([ip | ip .id = ap .id and ap .value = ip .value])])

  -- Advance the script index
  post ScriptIndexAdvanced:
    index = index$pre + 1
  -- If preconditions not and the next event is not #MODIFIED then
  -- the 'event' message is sent to GModS with appropriate parameter values.
  post NotModifiedSendEvent:
    let nextEvt : GoalEventImpl = event->at(index$pre + 1) in
    if nextEvt .type != #MODIFIED implies
    let instGoal : InstanceGoal =
      gmods.instanceTree .goal->select([ig | ig .instID = nextEvt .instGoalID and ig .specID = nextEvt .specGoalID])
    let specEvt : SpecificationEvent =
      gmods.specTree .event->select([se | se .id = nextEvt .specGoalID])
    let instParams : InstanceParameters =
      newInstParams (newInstParams .params->forall([ip | instParams .param->exists([ip | ip .coll .param .coll->exists([ip | ip .id = ip .id and ip .value = ip .value])])
    newInstParams and gmods.event .instGoal .specEvt .instParams])
    -- If preconditions not and the next event is #MODIFIED then the
    -- 'modifyInstanceGoal' message is sent to GModS with appropriate parameter values.
    post ModifiedSendModifyInstanceGoal:
    let nextEvt : GoalEventImpl = event->at(index$pre + 1) in
    if nextEvt .type = #MODIFIED implies
    let instParams : InstanceParameters =
      newInstParams (newInstParams .params->forall([ip | instParams .param->exists([ip | ip .coll .param .coll->exists([ip | ip .id = ip .id and ip .value = ip .value])])
    newInstParams and gmods.modifyInstanceGoal .nextEvt .specGoalID .nextEvt .instGoalID .instParams]
Technical Inspection Checklist

- The system component diagram (Figure 2) uses legal UML elements.
- Section 4.1 clearly explains the elements of the system component diagram.
- Table 1 clearly explains the responsibilities of each system component.
- Table 2 clearly specifies the GMoDS Test Driver main program interface.
- Table 3 clearly specifies the GMoDSVisualizer interface.
- Table 4 clearly specifies the TestDriver interface.
- Section 4.4 clearly explains the rationale for the system architecture.
- The GMoDS Test Driver architectural module class diagram (Figure 3) uses legal UML elements.
- Table 5 clearly explains the responsibility of each GMoDS Test Driver architectural class or interface (Note: GoalTree is a GMoDS interface not a GMoDS Test Driver interface).
- Table 6 clearly specifies the GoalEvent interface.
- Table 7 clearly specifies the EventScript interface.
- Section 5.1.3 clearly explains the rationale for the GMoDS Test Driver architecture.
- The GMoDS architectural class diagram (Figure 14) uses legal UML elements.
- Section 8 clearly explains the rationale for Figure 14 elements.
- Classes in the USE/OCL model (section 9) are consistent with the classes in Figure 14.
- Attributes in the USE/OCL model (section 9) are consistent with the corresponding classes in Figure 14.
- Associations in the USE/OCL model (section 9) are consistent with associations in Figure 14.
- Multiplicities in the USE/OCL model (section 9) are consistent with multiplicities on the corresponding associations in Figure 14.
Test Plan

- Mostly manual test cases (black box testing).
- Test cases closely align with the use cases.
  - Test Case TC.GTD–1 – Load Event Script
  - Test Case TC.GTD–2 – Event Script Operation
  - Test Case TC.GTD–3 – Random Event Script Operation
  - Test Case TC.GTD–4 – Save Event Script
  - Test Case TC.GV–1 – Display Specification Tree
  - Test Case TC.GV–2 – Display Instance Tree
  - Test Case TC.GV–3 – Zooming
  - Test Case TC.GV–4 – Show/Hide Instance Goals of Specific Types
  - Test Case TC.GV–5 – Show/Hide All Specification Goal Parameters
  - Test Case TC.GV–6 – Show/Hide All Instance Goal Parameters
  - Test Case TC.GV–7 – Show/Hide Specific Instance Goal Parameters
  - Test Case TC.GV–8 – Collapse/Expand Instance Goal Sub–tree
  - Test Case TC.GV–9 – Change Instance Goal State Colors
Test Plan

- Debug logs used to determine expected results in many manual test cases.
- Unit tests for event/event script methods.
  - Controller unit tests are unneeded due to method simplicity.
## Test Plan – Sample Manual Test Case

<table>
<thead>
<tr>
<th>Use Cases Tested</th>
<th>GTD–2 Issue Random Events</th>
<th>GTD–3 Save Event Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features Tested</td>
<td>TF.GTD–2.3</td>
<td>TF.GTD–3.1</td>
</tr>
<tr>
<td></td>
<td>TF.GTD–3.2</td>
<td>TF.GTD–3.2.1</td>
</tr>
<tr>
<td></td>
<td>TF.GTD–3.2.2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal Diagrams</th>
<th>A goal diagram compatible with the event scripts.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Required Event Scripts (repeat procedure for each script listed here)</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. For the no input case, click &quot;Issue Random Events&quot; and “Play” and allow the event script to end. Then, select “File</td>
</tr>
<tr>
<td></td>
<td>2. For the other input cases, load the event script and then select “File</td>
</tr>
</tbody>
</table>

| Expected Results For Each Required Event Script | 1. For the no input case, the two runs should have exactly the same debug logs except for actual time stamps. |
|                                                  | 2. For the event script with all valid events the saved script should match the input script. |
|                                                  | 3. For the event script with invalid events, the saved script should be empty. |
Top Technical Risks

- Specification tree
  - Drawing relations between non-adjacent goals
- Instance tree
  - Implementing “modification” as a parameter value origin
Architectural Prototype Demonstration
Questions/Comments