System Architecture Design

For a GMoDS-based Runtime Agent Role Interpreter

Version 1.0

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

Kyle Hill
CIS 895 – MSE Project
Kansas State University

Table of Contents

1 Introduction 3
2 Architecture 3
2.1 Component Design 4
2.2 Component Interface Specification 4
2.3 System Analysis 7
2.4 High-Level Design 8
3 Mid-Level Design 9
3.1 Agents 9
3.2 Capabilities 10
3.3 Role Interpreter 11
3.4 Roles 12
4 Component Interaction 12
4.1 Role Execution Sequence 13
5 Role Models 14
5.1 Area Searcher 14
5.2 Gold Fetcher 14
5.3 Gold Returner 15
5.4 Hunter-Killer 15
6 USE/OCL Model 15

1 Introduction

This document provides system design information for the GMoDS-based Runtime Agent Role Interpreter. This interpreter serves as the basis for an agent architecture in an OMACS multiagent system. This document details the component design and interface specification. In addition, it provides a high-level overview of the entire system's static design. It also provides mid-level design details for each component. However, a full interface specification for each component is not provided. Finally, component interaction during role execution is specified via a sequence diagram.

2 Architecture

The overall system architecture is constrained by the existing OMACS and GMoDS frameworks into which the role interpreter and its example agent architecture must fit. The system can be decomposed into four major components: The Agent Architecture, the Capability definitions, the Role Interpreter itself, and finally the OMACS Role Adapters. The core of the system consists of the Role Interpreter and its three constituent parts: The RoleLevelGoalModel, the GoalCapabilityMap, and the RoleInterpreter itself. In addition, an example agent architecture is provided to demonstrate the viability of the Role Interpreter in the WumpiWorld test environment. The remainder of the system provides the components necessary for interaction with the rest of the GMoDS and OMACS frameworks.

2.1 Component Design

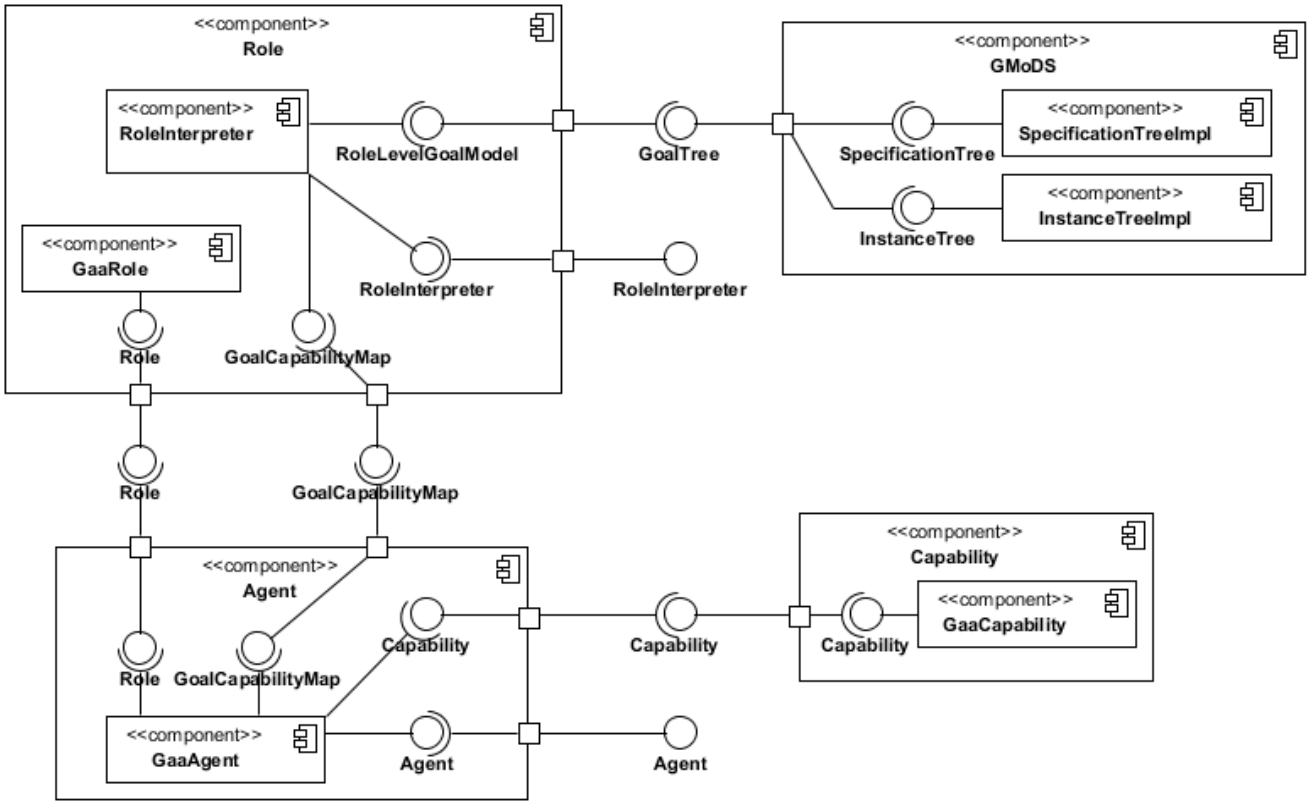


Figure 1 – Component Diagram

2.2 Component Interface Specification

Only the public interface exported and used by the Role Interpreter is defined in detail. While other components, such as the Agent Architecture and Capability definitions provide other public interfaces, these are less interesting as they are largely defined by OMACS and GMoDS frameworks with which they interact. In addition, those modules only exist to demonstrate the viability of the core of the system: The Role Interpreter.

GoalCapabilityMap

Signature	`addMapping( s : String, m : Method, c : Capability )`
Purpose	Adds the given name, method, capability entry to the map
Pre-conditions	The given string, method, and capability are not null.
Post-conditions	A new mapping of the given string to method, capability pair has been added to the database
Signature	`invoke( g : String, i : InstanceParameters ) : Object`

Purpose	Invokes the given goal name (capability method name) on the object that maps to the given goal name in the map.
Pre-conditions	The given string is not null. A mapping whose key matches the given name that takes the given InstanceParameters exists within the map. If no method is found a NoSuchMethodException is thrown.
Post-conditions	The method that maps to the given name and formal parameters has been called with the given actual parameters.

RoleLevelGoalModel

Signature	`event( g : InstanceGoal<InstanceParameters>, s : SpecificationEvent ) : InstanceChanges`
Purpose	Fires the given SpecificationEvent from the given InstanceGoal
Pre-conditions	The InstanceGoal and SpecificationEvent are not null
Post-conditions	The given event has been fired and the InstanceTree has been updated to reflect the event
Signature	`getEventsToFire( g : SpecificationGoal, r : Object) : Set<SpecificationEvent>`
Purpose	Returns a set of SpecificationEvents from the given SpecificationGoal and method invocation return object. This set is the set of events that should be fired based on the invocation return value.
Pre-conditions	The given SpecificationGoal is not null
Post-conditions	The returned set contains all events that should be fired
Signature	`getNextInstanceGoal() : InstanceGoal<InstanceParameters>`
Purpose	Returns a leaf-level InstanceGoal from the set of active InstanceGoals whose preconditions have been met.
Pre-conditions	None
Post-conditions	A leaf-level InstanceGoal from the set of active InstanceGoals whose preconditions have been met has been returned.
Signature	`hasActiveInstanceGoals() : Boolean`
Purpose	Returns true if a call to getNextInstanceGoal would return a non-null value.
Pre-conditions	None
Post-conditions	True has been returned if there is an active, leaf instance goal to pursue, false otherwise.
Signature	`reset(i : InstanceParameters)`
Purpose	Resets the InstanceTree back to its default state.
Pre-conditions	None
Post-conditions	The InstanceTree has been reset back to its default state.

RoleInterpreter

Signature	`execute( a : GaaAgent, g : InstanceGoal<InstanceParameters> )`
Purpose	Executes this interpreter's RLGM using the given agent in pursuit of the given top-level goal.
Pre-conditions	The agent and goal are not null. The given goal is a top-level organizational goal.
Post-conditions	Exactly one leaf-level active goal from the set of current active goals in the RLGM's instance tree has been executed and the resulting events from that execution have been fired (updating the RLGM's instance tree for the next call to execute).
Signature	`isDone() : boolean`
Purpose	Returns true if there are no active instance goals in the RLGM to execute.
Pre-conditions	None
Post-conditions	True is returned if there are no active instance goals in the RLGM to execute, false otherwise.

GaaRole

Signature	`getRoleLevelGoalModel() : RoleLevelGoalModel`
Purpose	Returns the RoleLevelGoalModel object that defines this Role
Pre-conditions	None
Post-conditions	A new RoleLevelGoalModel object that defines this Role has been returned. The RoleLevelGoalModel InstanceTree has been reset back to its initial state.

GaaAgent

Signature	`getGoalCapabilityMap() : GoalCapabilityMap`
Purpose	Returns the GoalCapabilityMap that contains goal name to capability method mappings for this Agent's capabilities.
Pre-conditions	None
Post-conditions	The GoalCapabilityMap that contains goal name to capability method mappings for this Agent's capabilities has been returned.
Signature	`registerCapability( c : GaaCapability )`
Purpose	Registers the given capability with this agent's GoalCapabilityMap
Pre-conditions	The given capability is not null.
Post-conditions	The given capability has been registered with this agent's GoalCapabilityMap.

GaaCapability

Signature	`registerMethods( gcm : GoalCapabilityMap )`
Purpose	Registers all methods that this GaaCapability wishes to expose to Roles with

	the given GoalCapabilityMap.
Pre-conditions	The given GoalCapabilityMap is not null.
Post-conditions	All methods that this GaaCapability wishes to expose to Roles with the given GoalCapabilityMap have been registered.

2.3 System Analysis

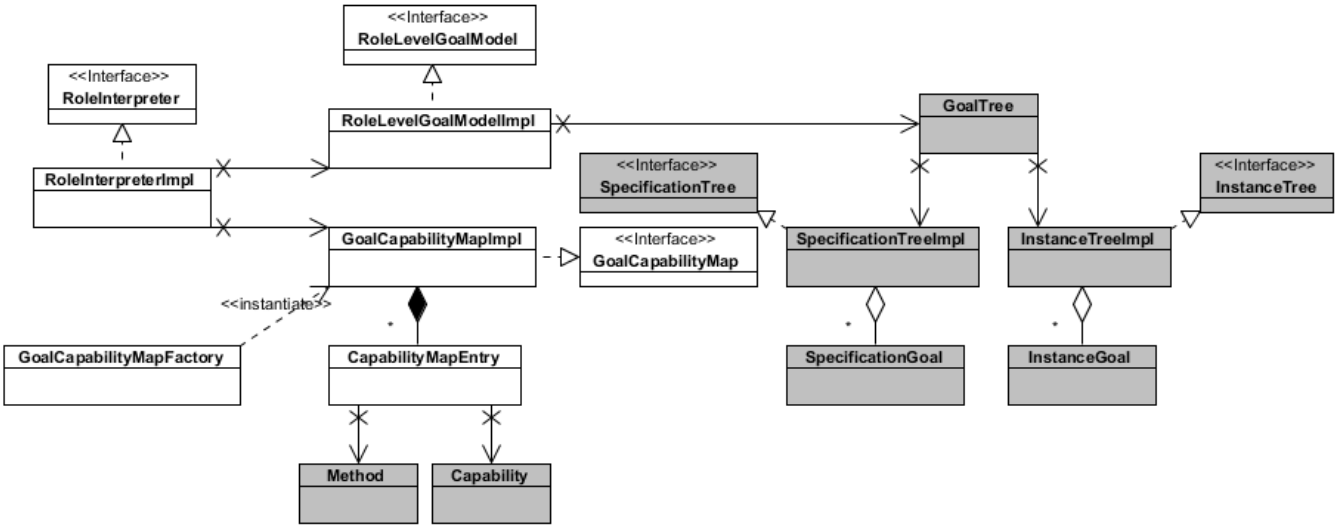


Figure 2 – Analysis Class Diagram

This analysis class diagram captures the basic relations between the core Role Interpreter classes and the rest of the system. The RoleInterpreterImpl contains internal references to the RoleLevelGoalModelImpl and the GoalCapabilityMapImpl. The RoleLevelGoalModelImpl is a delegate to the GMoDS GoalTree, which represents both an InstanceTree and a SpecificationTree. The GoalCapabilityMapImpl is a wrapper around a Java map of String goal names to CapabilityMapEntry objects, which are a simple tuple of a Method and its owning Capability object.

2.4 High-Level Design

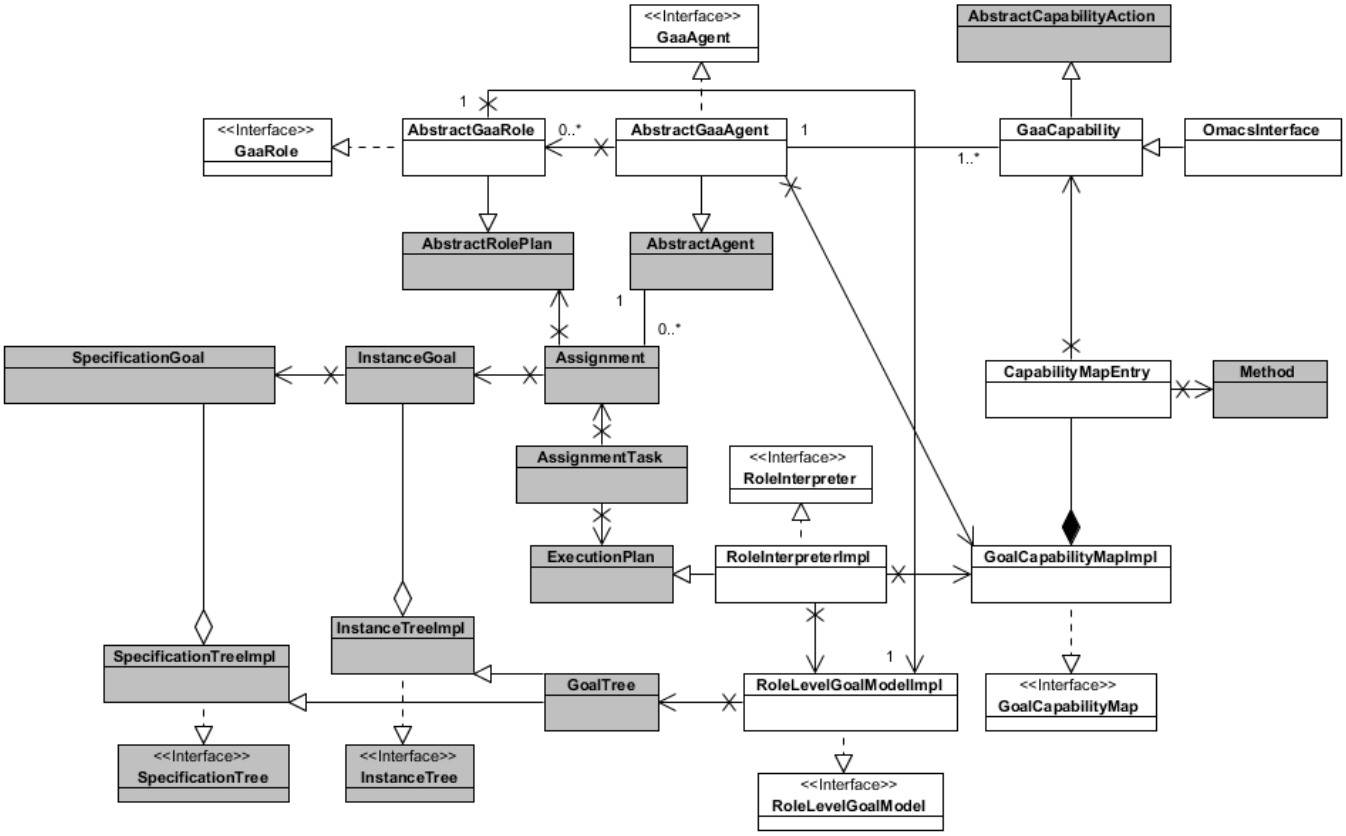


Figure 3 – High-Level Class Diagram

This high-level class diagram captures the basic relations and concepts of the entire system: The Role Interpreter, the agent architecture, and the rest of the code required to interface with the OMACS and GMoDS frameworks. The gray classes are existing classes that are used by the system, but not actually provided by it. The GaaRoleImpl is a simple adapter between the OMACS AbstractRolePlan and the RoleInterpreter by associating RoleLevelGoalModels with Roles.

The AbstractGaaAgent serves as the basis for an agent architecture that demonstrates the viability of the Role Interpreter. The AbstractGaaAgent contains a lot of code that is specific to the WumpiWorld demonstration. However, it also contains code to bootstrap the GoalCapabilityMap by registering the agent's GaaCapability objects with the GoalCapabilityMap. This map is provided to the RoleInterpreter during goal execution. The GaaCapability provides a thin wrapper around standard capabilities that allows for method registration with the GoalCapabilityMap. The rest of the system has already been described with the analysis class diagram.

3 Mid-Level Design

3.1 Agents

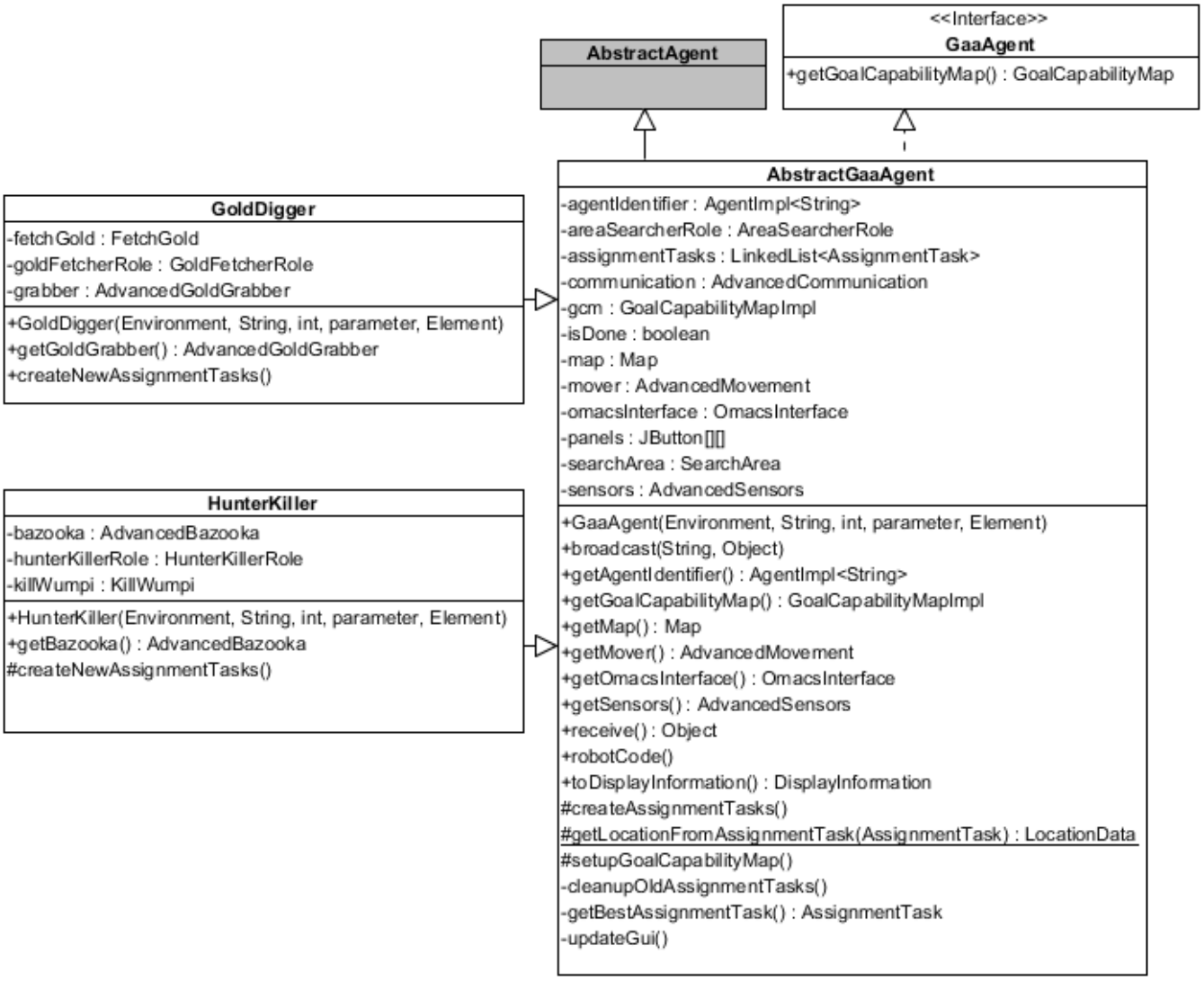


Figure 4 – Agents Package Class Diagram

The class diagram for the agent component is shown in Figure 4. The class consists of three classes: The GaaAgent, the GoldDigger, and the HunterKiller. Most of this code is specific to the WumpiWorld demonstration. Agents serve as platforms for the deployment of capabilities into the system. They are responsible for actually playing the roles they are assigned. In this system, all role and capability assignments are done statically at compile time. There is no real reason for this other than to make the demonstration system as simple as possible. The GoldDigger class adds capabilities that are required to perform the FetchGold and ReturnGold roles. Similarly, the HunterKiller class adds capabilities for the KillWumpi role.

3.2 Capabilities

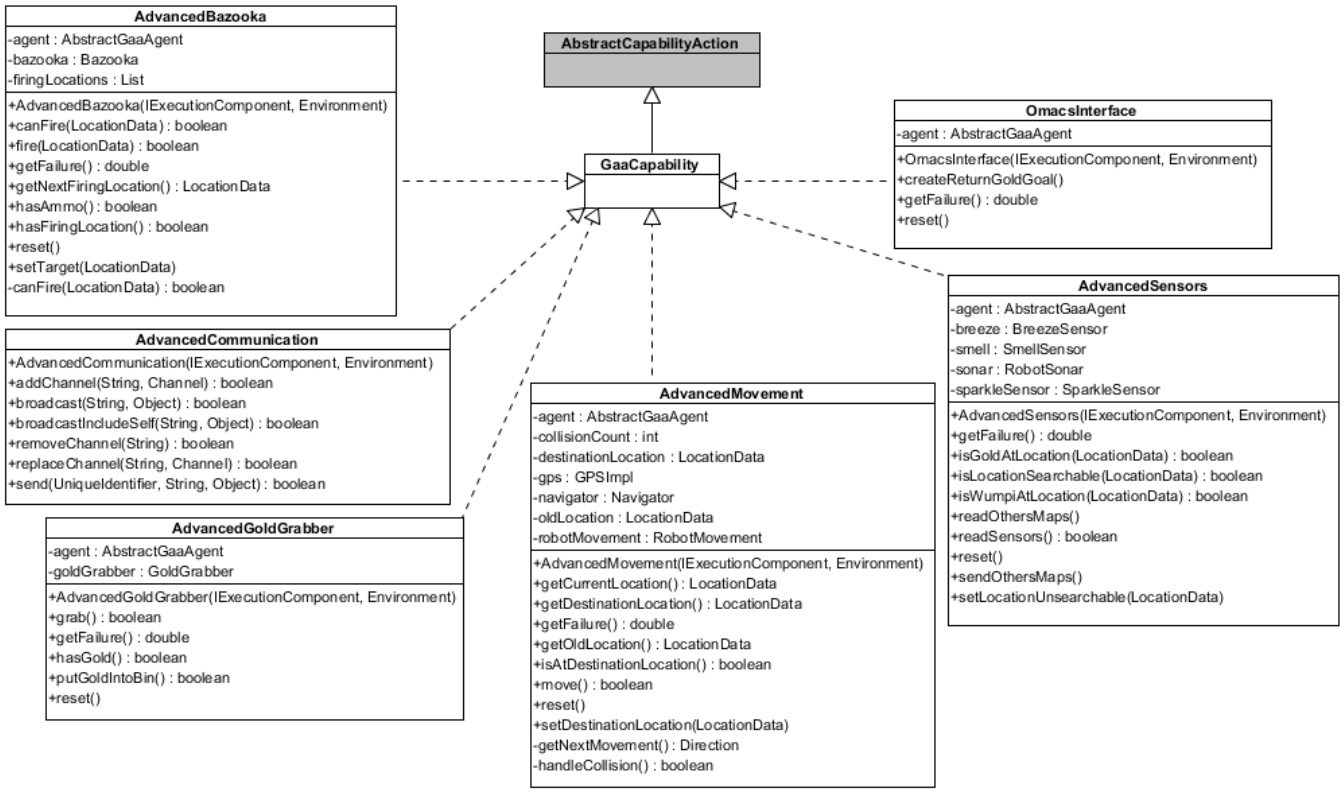


Figure 5 – Capability Package Class Diagram

The class diagram from the Capability component is shown in Figure 5. This component is almost completely specific to the agent architecture demonstration in Wumpi World. This component contains a number of capabilities that the agents possess and make use of to achieve their assigned roles. Each public capability method is mapped to a leaf-level goal in the Role Level Goal Model through an agent's GoalCapabilityMap.

The abstract parent class GaaCapability requires child classes to implement the registerMethods() method to provide a mapping from goal name to capability method. The OmacsInterface capability is somewhat special. This capability contains methods that are needed to interact with the rest of the OMACS system. Currently, it only contains the createGoal() method that will instantiate a new instance goal when called.

3.3 Role Interpreter

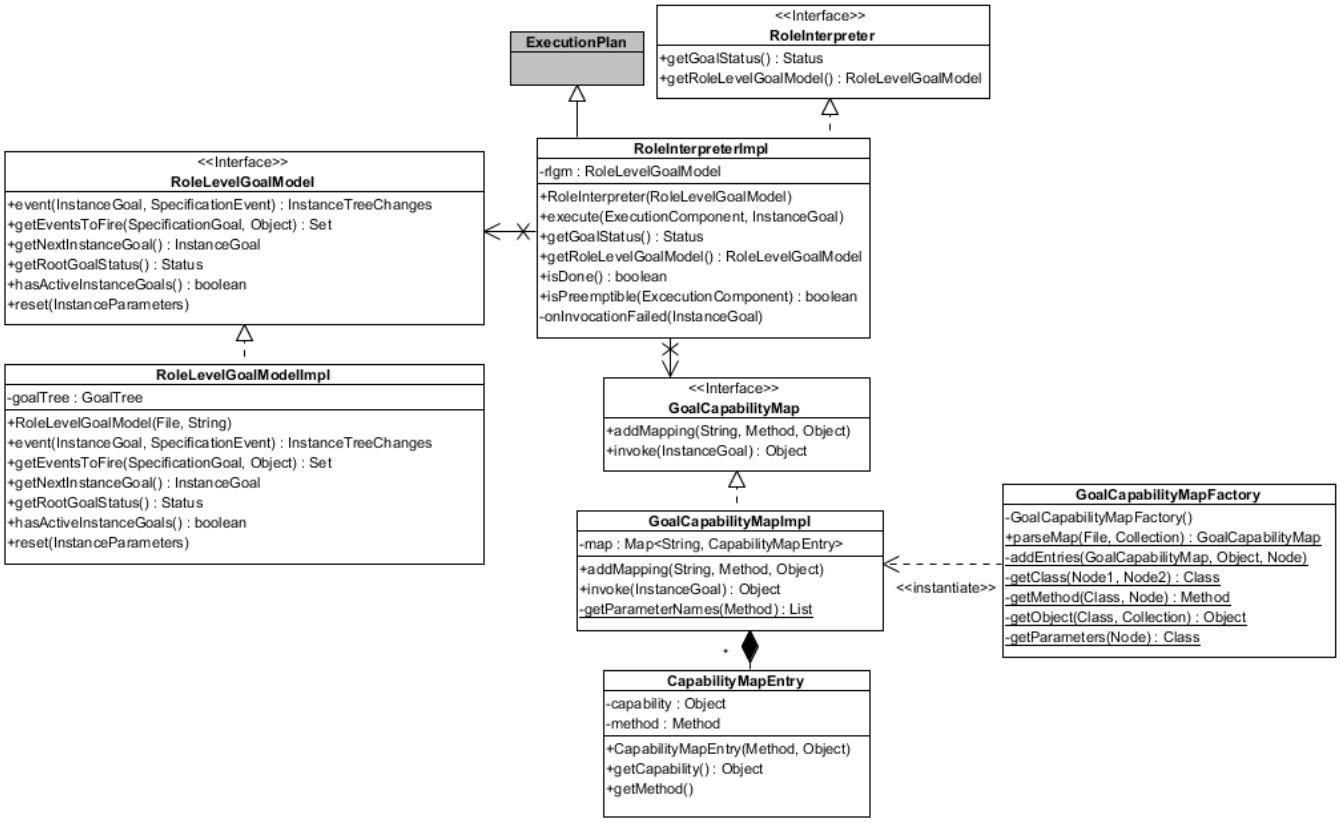


Figure 6 – Role Interpreter Package Class Diagram

The Role Interpreter component is shown in Figure 6. The details of this component have already been given in previous sections. This diagram shows the same relationships between classes and provides information about methods and attributes of those classes.

3.4 Roles

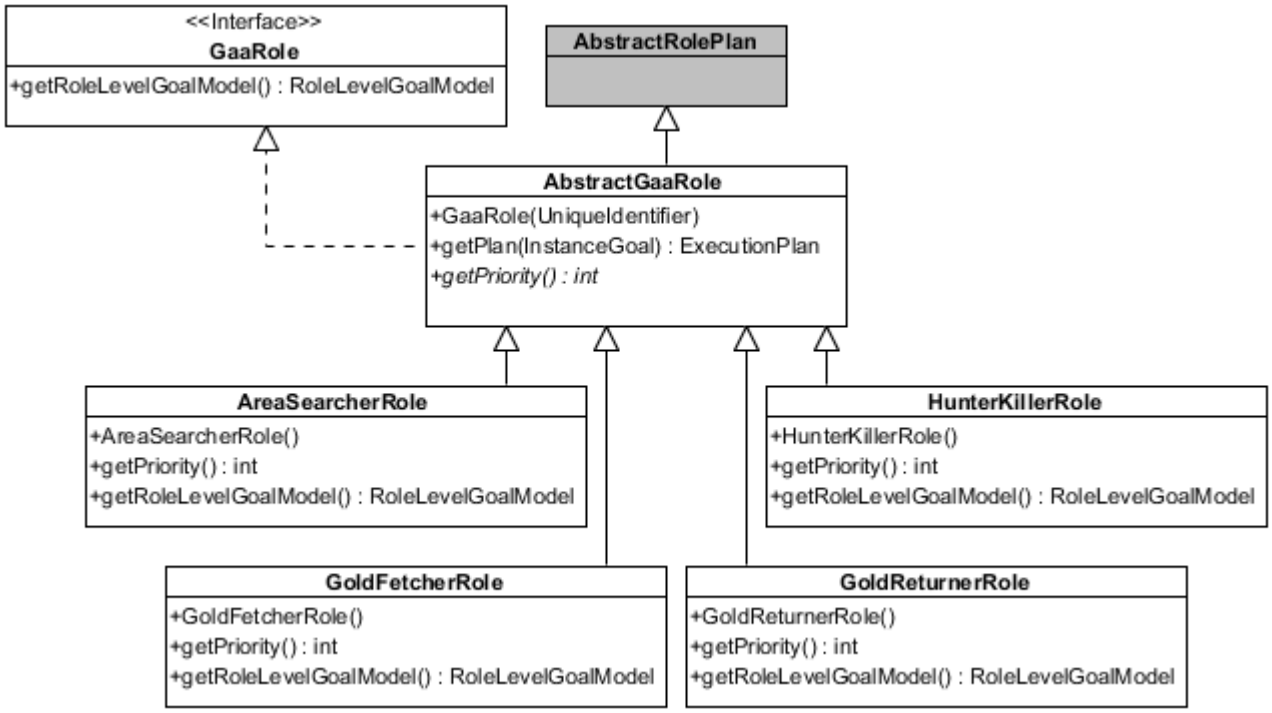


Figure 7 – Role Package Class Diagram

The Role component class diagram is shown in Figure 7. This component's responsibility is to provide an adapter between the Role representation used by OMACS in WumpiWorld, and the Role Interpreter. The parent class, GaaRole provides methods to return a statically defined role priority (to make the agent architecture as simple as possible) and to return the RoleLevelGoalModel that defines the Role's behavior for the RoleInterpreter.

The RoleLevelGoalModel is defined by a GMoDS Goal Model XML file that is created by the AgentTool3 graphical editor. This XML file is read in at runtime by the RoleLevelGoalModel class. This allows for users of the system to customize its behavior by simply editing the XML file in the graphical editor.

4 Component Interaction

This system contains a large number of interactions between various components. Startup behavior, top-level goal creation, capability-specific interactions, and role execution are all complex and vital for the agent architecture demonstration. However, the most interesting sequence is the role execution sequence. This sequence is the heart of the system and defines how agent assignments are transformed into actions that are executed through capabilities to accomplish system goals. This sequence's behavior is independent of the agent architecture and scenario in which it is deployed.

4.1 Role Execution Sequence

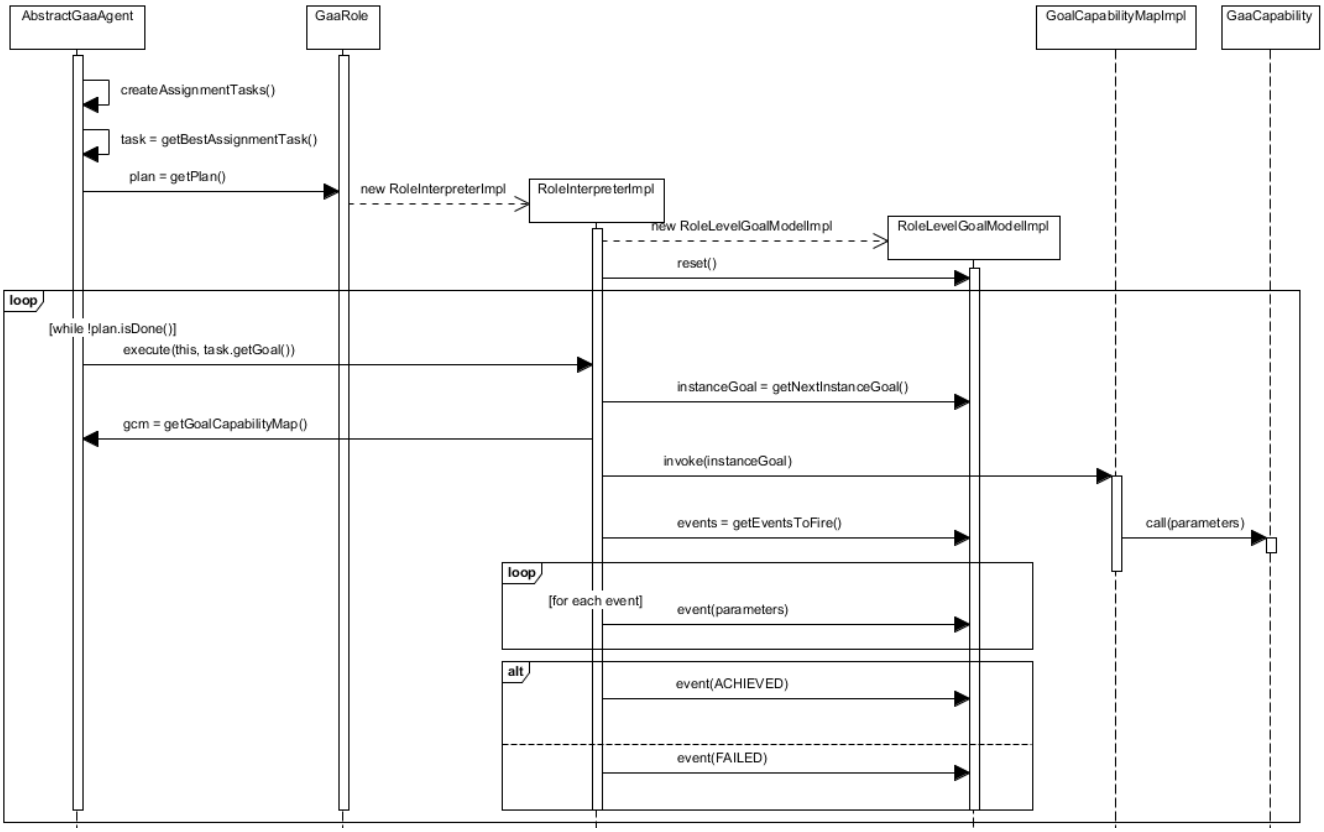


Figure 8 – Role Execution Sequence Diagram

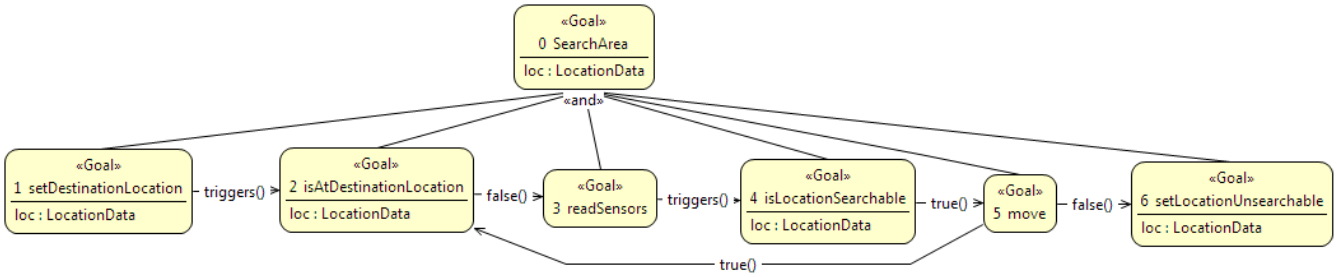
Figure 8 shows the interactions required between various objects in order to execute a role using the Role Interpreter.

1. The GaaAgent starts its main execution loop by creating new assignment tasks for itself based on the initial system conditions and any sensor data that is immediately available. This self-task assignment is not strictly required, any other method for assigning tasks to an agent would be sufficient. However, this method keeps the agent architecture as simple as possible.
2. Next the agent selects the "best" task to work on based on some built heuristics. Again, the details of this are specific to the agent architecture. In this system, the highest priority task is selected. If two tasks share the same priority, then the one whose goal is the "closest" is chosen first.
3. The agent then gets the plan associated with the assignment's role. This causes a lazy instantiation of a RoleInterpreter and its associated RoleLevelGoalModel.
4. Then, while the plan is not done, execute is called repeatedly by the agent
5. The RoleInterpreter gets an active leaf-level instance goal from the instance tree to execute.
6. The RoleInterpreter also gets the GoalCapabilityMap from the calling agent.
7. Next, the RoleInterpreter actually invokes the capability that is associated with the active goal using the GoalCapabilityMap.

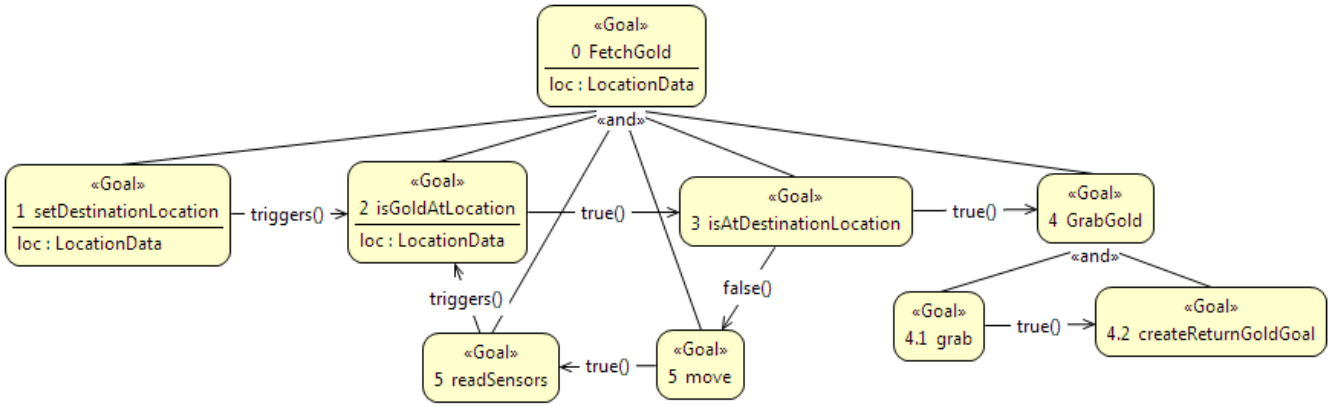
8. The RoleInterpreter calls into the RoleLevelGoalModel with the return value of the capability call to determine what events to fire to cause the correct changes in the instance tree.
9. Then, for each event that is to be fired, the RoleInterpreter actually triggers the event in the instance tree.
10. Finally, based on the return value from the capability method, the current goal is either marked as achieved or failed.

5 Role Models

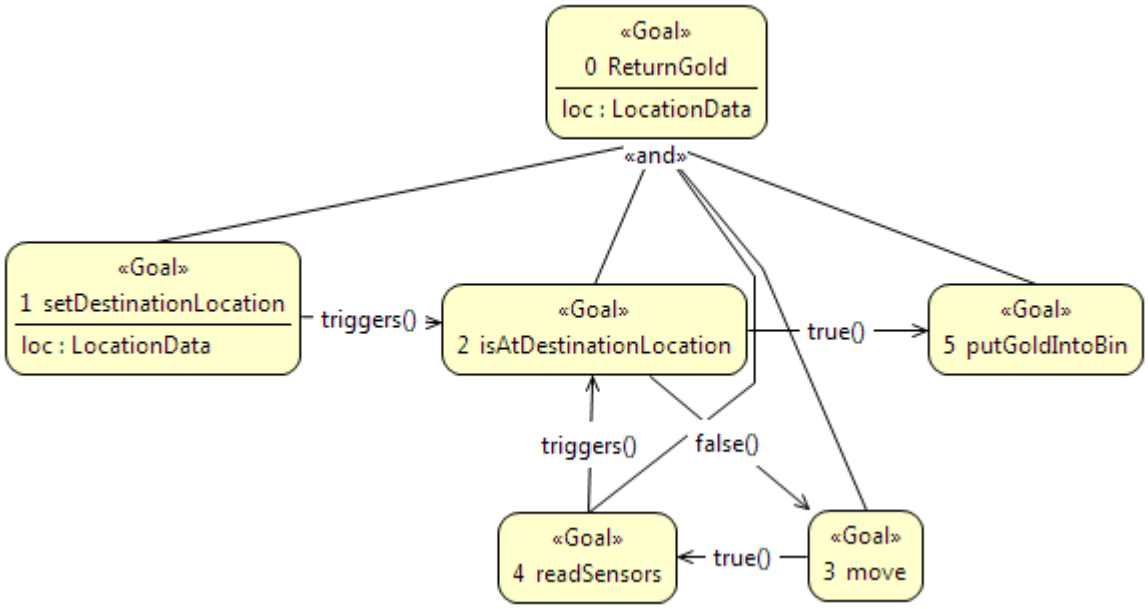
5.1 Area Searcher



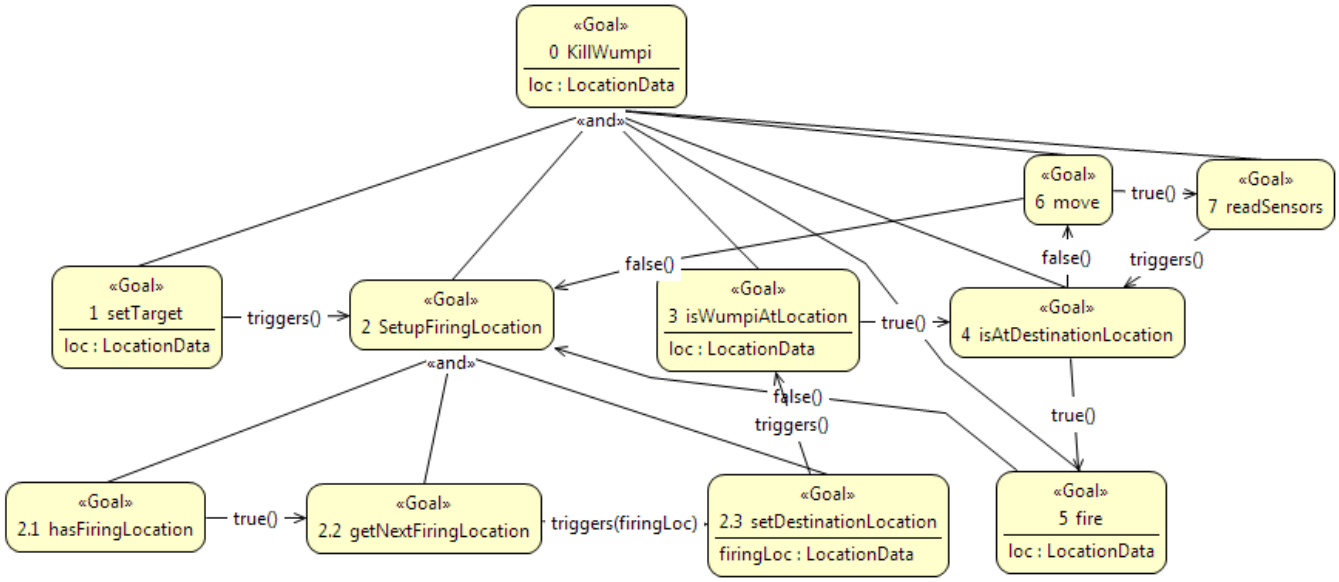
5.2 Gold Fetcher



5.3 Gold Returner



5.4 Hunter-Killer



6 USE/OCL Model

```
--------------------------------------------------------------------------------
-- GMoDS Based Agent Architecture
--
-- This file contains a formal specification of the invariants maintained by
```

```
-- the getEventsToFire method from the RoleLevelGoalModel
--
-- File:    GMoDSAgentArchitecture.use
-- Author:  Kyle Hill
-- Date:    June 20, 2011
------------------------------------------------------------------------------

model GMoDSAgentArchitecture

------------------------------------------------------------------------------
-- Classes
------------------------------------------------------------------------------
class Object
end

------------------------------------------------------------------------------
-- GAA Classes
------------------------------------------------------------------------------
class RoleInterpreter
end

class RoleLevelGoalModel
operations
    getEventsToFire(g : ParameterizedSpecificationGoal, r : Boolean) :
Set(SpecificationEvent)
    getNextInstanceGoal() : InstanceGoal
end

class GoalCapabilityMap
operations
    addMapping(s : String, e : CapabilityMapEntry)
    invoke(g : ParameterizedSpecificationGoal, p : InstanceParameters)
end

class CapabilityMapEntry
attributes
    id : String
end

class Method
attributes
    name : String
    return : Object
end

class Capability
end

------------------------------------------------------------------------------
-- GMoDS Classes
------------------------------------------------------------------------------
class GoalTree
end

class SpecificationTree
end
```

```
class SpecificationEvent
attributes
    id : String
end

class ParameterizedSpecificationGoal
attributes
    id : String
    isLeaf : Boolean
end

class SpecificationParameters
end

class SpecificationParameter
attributes
    key : String
end

class InstanceTree
end

class InstanceGoal
attributes
    id : String
end

class InstanceParameters
end

class InstanceParameter
attributes
   key : String
   value : Object
end

--------------------------------------------------------------------------------
-- Associations
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- GAA Associations
--------------------------------------------------------------------------------
association RLGM between
    RoleInterpreter[1]
    RoleLevelGoalModel[1] role rlgm
end

association RLGMGoalTree between
    RoleLevelGoalModel[1]
    GoalTree[1] role goalTree
end

association GCM between
    RoleInterpreter[1]
```

```
    GoalCapabilityMap[1] role gcm
end

association MapEntries between
    GoalCapabilityMap[1]
    CapabilityMapEntry[0..*] role entries
end

association EntryMethod between
    CapabilityMapEntry[0..*]
    Method[1] role method
end

association EntryCapability between
    CapabilityMapEntry[0..*]
    Capability[1] role capability
end

association MethodParams between
    Method[1] role method1
    Object[0..*] role params
end

-------------------------------------------------------------------------------
-- GMoDS Associations
-------------------------------------------------------------------------------
association SpecTree between
   GoalTree[1]
   SpecificationTree[1] role specTree
end

association SpecGoals between
   SpecificationTree[1]
   ParameterizedSpecificationGoal[1..*] role goals
end

association SpecGoalParam between
   ParameterizedSpecificationGoal[1]
   SpecificationParameters[0..1] role param
end

association SpecEvents between
   SpecificationTree[1]
   SpecificationEvent[0..*] role events
end

association SpecEventParams between
   SpecificationEvent[1]
   SpecificationParameters[0..1] role param
end

association SpecParams between
   SpecificationParameters[1]
   SpecificationParameter[0..*] role params
end
```

```
association InstTree between
   GoalTree[1]
   InstanceTree[1] role instTree
end

association ActiveInstGoals between
   InstanceTree[1]
   InstanceGoal[0..*] role activeGoals
end

association InstGoalParam between
   InstanceGoal[1]
   InstanceParameters[0..1] role param
end

association InstParams between
   InstanceParameters[1]
   InstanceParameter[0..*] role params
end

--------------------------------------------------------------------------------
-- Constratints
--------------------------------------------------------------------------------
constraints

--------------------------------------------------------------------------------
-- RoleLevelGoalModel Constraints
--------------------------------------------------------------------------------
context RoleLevelGoalModel::getEventsToFire(g : ParameterizedSpecificationGoal, r :
Boolean) : Set(SpecificationEvent)

-- The given specification goal must exist within the specification tree and be
-- unique
pre GoalInTree:
    goalTree.specTree.goals->select(id = g.id)->size() = 1

-- If the return value is not boolean, then all goal model specified events are
-- returned
post NoBoolReturnsAll:
    r.isUndefined() implies goalTree.specTree.events = result

-- If the return value is true, then all the "true" prefixed events are returned.
--   If no events are prefixed with "true", then all unconditional events are
returned.
-- Otherwise, if the return value is false, then all the "false" prefixed events
are returned.
--   If no events are prefixed with "false", then all unconditional events are
returned.
post CorrectEventsReturned:
    let allEvents   : Set(SpecificationEvent) = goalTree.specTree.events in
    let trueEvents  : Set(SpecificationEvent) = allEvents-
>select(id.toLower().substring(1, 4) = 'true') in
    let falseEvents : Set(SpecificationEvent) = allEvents-
>select(id.toLower().substring(1, 5) = 'false') in
    let unconEvents : Set(SpecificationEvent) = allEvents - trueEvents -
falseEvents in
```

```
        if r then
            if trueEvents->isEmpty() then
                result = unconEvents
            else
                result = trueEvents
            endif
        else
            if falseEvents->isEmpty() then
                result = unconEvents
            else
                result = falseEvents
            endif
        endif

context RoleLevelGoalModel::getNextInstanceGoal() : InstanceGoal

-- The returned instance goal is either null, or it is in the set of active
-- goals and it is a leaf goal
post NullOrActiveLeaf:
    result.isUndefined() or
        (goalTree.instTree.activeGoals->includes(result) and
            goalTree.specTree.goals->select(id = result.id and isLeaf)->notEmpty())

--------------------------------------------------------------------------------
-- GoalCapabilityMap Constratints
--------------------------------------------------------------------------------
context GoalCapabilityMap::addMapping(s : String, e : CapabilityMapEntry)
-- The given mapping contains no null values
pre NoNulls:
    not s.isUndefined() and
        not e.isUndefined() and
            not e.method.isUndefined() and
                not e.capability.isUndefined()

-- The entry has been added to the mapping
post EntryAdded:
    entries = entries@pre->including(e) and
        e.id = s

context GoalCapabilityMap::invoke(g : ParameterizedSpecificationGoal, p :
InstanceParameters)
-- The given goal is not undefined
pre NoNulls:
    not g.isUndefined()

-- A mapping is already present in the GCM for this goal
pre MappingExists:
    entries->exists(id = g.id)

-- The specification parameters and instance parameters are null, or
-- There exists an instance parameter for each specification parameter of the goal
pre FormalParamsMatch:
    let specificationParams = g.param.params in
    let instanceParams      = p.params in
        ((g.param.isUndefined() or specificationParams->isEmpty()) and
(p.isUndefined() or instanceParams->isEmpty())) or
```

```
            specificationParams->forAll(fp : SpecificationParameter |
instanceParams->exists(ap : InstanceParameter | fp.key = ap.key))
```