Component 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

This document provides component 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 component-level detailed design information and interface specifications. In addition, this document provides a full description of the method, attributes, and other properties of all system classes.

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.

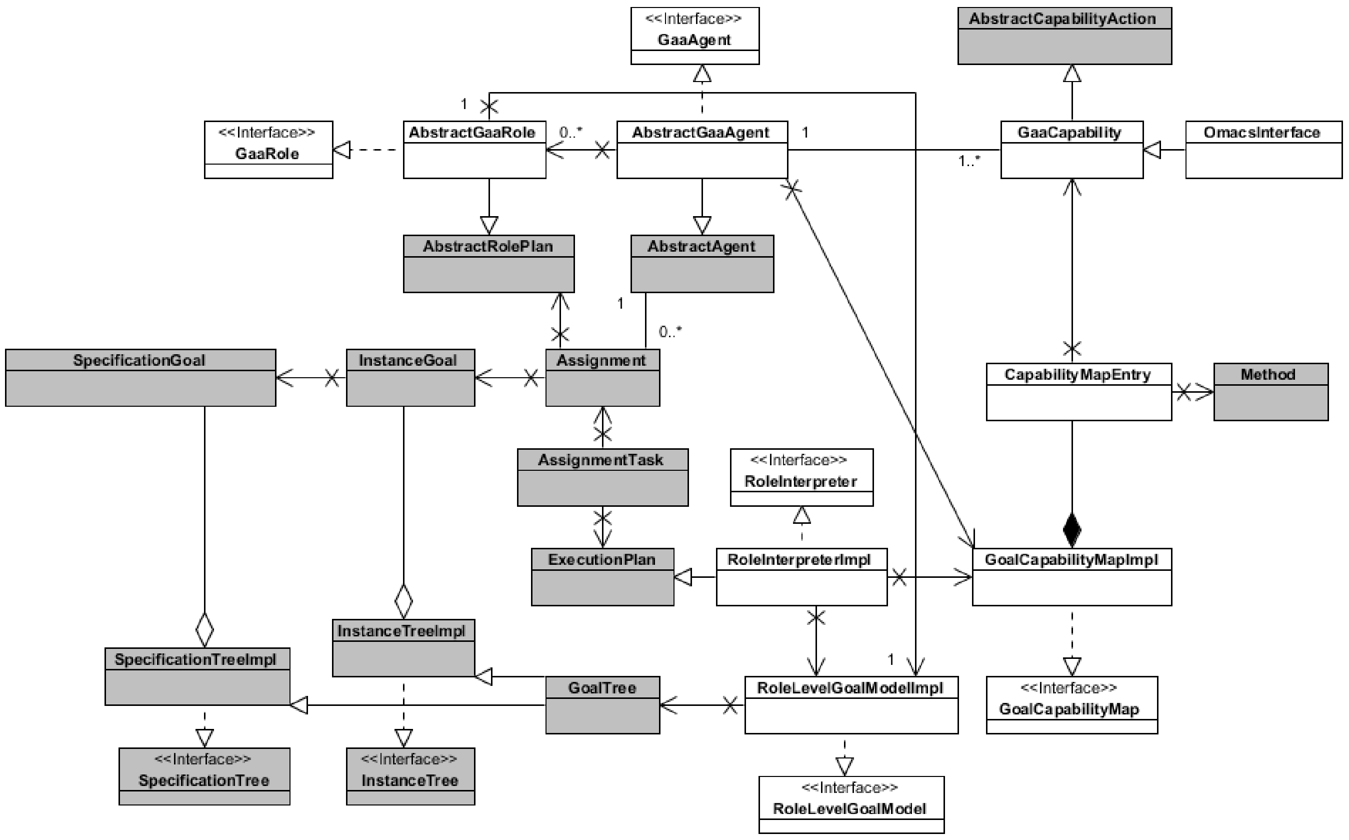

Figure 1 – High-Level Class Diagram

Figure 1 shows a high-level class diagram of the most important components of the system. Each component will be described in detail later in this document. This diagram serves to illustrate how the various pieces of the system fit together. This system is designed to integrate seamlessly with the existing OMACS and GMoDS frameworks. The RoleInterpreter itself extends directly from the ExecutionPlan class provided by OMACS. Similarly, the RoleLevelGoalModel internally delegates its public API to a GoalTree provided by GMoDS. The capabilities that are used in the GoalCapabilityMap are special versions of AbstractCapabilityActions provided by OMACS. The only difference is that these classes have annotated parameters so that name information is available at runtime for GMoDS parameter to Java parameter mapping.

3 Component Design

The following sections provide a detailed description of the component-level design for each major component of the system. This description includes a detailed class diagram of all class methods and attributes, a description of each class, and an interface definition for the major operations of each package.

3.1 Agents

3.1.1 Description

The agent package consists of three primary classes: AbstractGaaAgent, GoldDigger, and HunterKiller. The public interface exported to the rest of the system is declared in the GaaAgent interface. 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. Future implementations of this system could include a more robust organization that would allow assignments to be made dynamically based on agent capabilities and other environmental factors.

3.1.2 Class Diagram

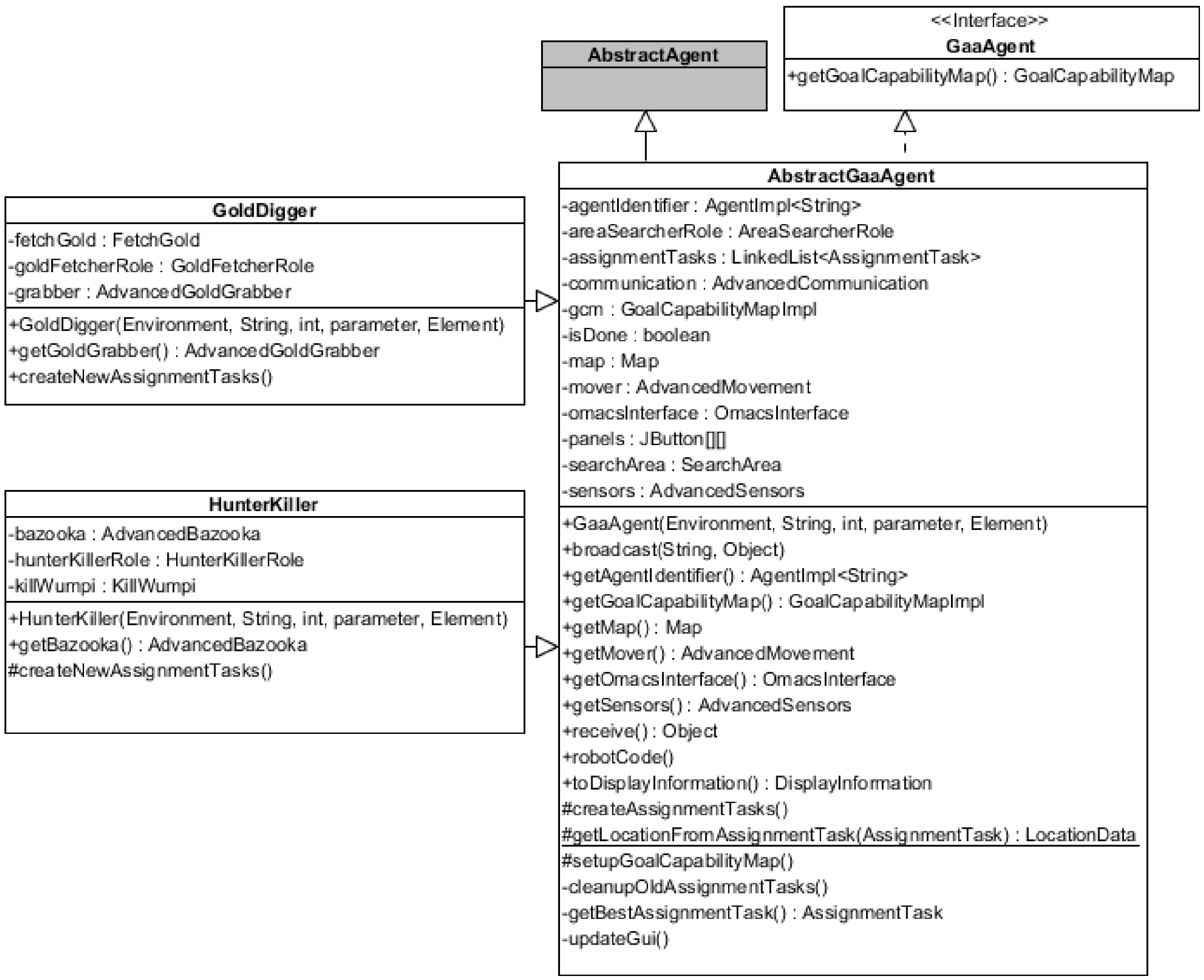
AbstractAgent
<<Interface>>
GaaAgent
+getGoalCapabilityMap() : GoalCapabilityMap
AbstractGaaAgent
-agentIdentifier : AgentImpl<String>
-areaSearcherRole : AreaSearcherRole
-assignmentTasks : LinkedList<AssignmentTask>
-communication : AdvancedCommunication
-gcm : GoalCapabilityMapImpl
-isDone : boolean
-map : Map
-mover : AdvancedMovement
-omacsInterface : OmacsInterface
-panels : JButton[][]
-searchArea : SearchArea
-sensors : AdvancedSensors
+GaaAgent(Environment, String, int, parameter, Element)
+broadcast(String, Object)
+getAgentIdentifier() : AgentImpl<String>
+getGoalCapabilityMap() : GoalCapabilityMapImpl
+getMap() : Map
+getMover() : AdvancedMovement
+getOmacsInterface() : OmacsInterface
+getSensors() : AdvancedSensors
+receive() : Object
+robotCode()
+toDisplayInformation() : DisplayInformation
#createAssignmentTasks()
#getLocationFromAssignmentTask(AssignmentTask) : LocationData
#setupGoalCapabilityMap()
-cleanupOldAssignmentTasks()
-getBestAssignmentTask() : AssignmentTask
-updateGui()
GoldDigger
-fetchGold : FetchGold
-goldFetcherRole : GoldFetcherRole
-grabber : AdvancedGoldGrabber
+GoldDigger(Environment, String, int, parameter, Element)
+getGoldGrabber() : AdvancedGoldGrabber
+createNewAssignmentTasks()
HunterKiller
-bazooka : AdvancedBazooka
-hunterKillerRole : HunterKillerRole
-killWumpi : KillWumpi
+HunterKiller(Environment, String, int, parameter, Element)
+getBazooka() : AdvancedBazooka
#createNewAssignmentTasks()

Figure 2 – Agents Package Class Diagram

3.1.3 Class Descriptions

3.1.3.1 AbstractGaaAgent

The AbstractGaaAgent serves as the common base class for all agent implementations. It also serves as the primary class of the demonstration agent architecture. It is responsible for constructing and housing the GoalCapabilityMap for the agent, and for actually invoking the RoleInterpreter with the agent's assigned Role.

In this system, tasks are self-assigned by the agents. The AbstractGaaAgent creates tasks for the agent by interpreting shared map data. These tasks are added to the set of active tasks for the agent to consider performing. Tasks that have been achieved or failed by the agent are removed from consideration. When the agent goes to perform its next task, it chooses the "best" task to work on selecting the one with the highest priority whose goal destination is closest to the agent's current location.

3.1.3.2 HunterKiller

The HunterKiller agent is responsible for searching the map for Wumpi to kill. It contains special code to instantiate new KillWumpi goals whenever a new, unclaimed Wumpi is discovered on the map. This is the only agent type who can play the KillWumpiRole.

3.1.3.3 GoldDigger

The GoldDigger agent is responsible for searching the map for gold, retrieving it, and then returning it to the bin at the start location on the map. It contains special code to instantiate new FetchGold and ReturnGold goals whenever new, unclaimed gold is discovered on the map, or when gold has been picked up. This is the only agent type who can play the FetchGoldRole and the ReturnGoldRole.

3.1.4 Interface Specification

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.

3.2 Capabilities

3.2.1 Description

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 RoleLevelGoalModel through an agent's GoalCapabilityMap. Each public capability method in this component is annotated with a custom @Name annotation on each parameter. This annotation contains the name of the parameter and allows the GoalCapabilityMap to intelligently map GMoDS instance parameters to Java method parameters by matching arguments with the same name.

The interface GaaCapability does not export any public methods, but simply serves as an indication that implementing classes have had their public methods appropriately annotated. Future versions of this system could use these annotations to hook into the Java compiler and provide compile-time verification that all parameters have been correctly annotated and that the annotations match the expected name.

3.2.2 Class Diagram

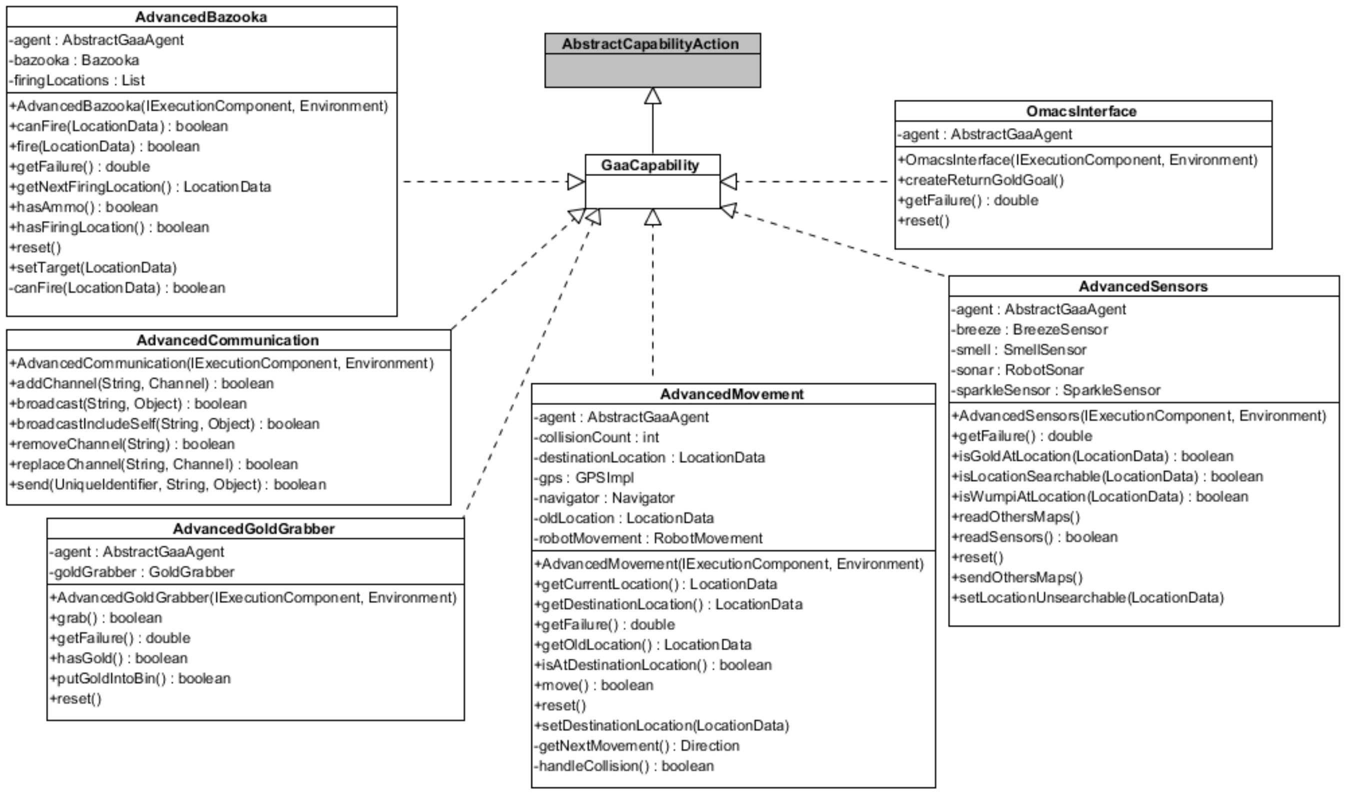
AdvancedBazooka
-agent : AbstractGaaAgent
-bazooka : Bazooka
-firingLocations : List
+AdvancedBazooka(IExecutionComponent, Environment)
+canFire(LocationData) : boolean
+fire(LocationData) : boolean
+getFailure() : double
+getNextFiringLocation() : LocationData
+hasAmmo() : boolean
+hasFiringLocation() : boolean
+reset()
+setTarget(LocationData)
-canFire(LocationData) : boolean
AbstractCapabilityAction
GaaCapability
OmacsInterface
-agent : AbstractGaaAgent
+OmacsInterface(IExecutionComponent, Environment)
+createReturnGoldGoal()
+getFailure() : double
+reset()
AdvancedCommunication
+AdvancedCommunication(IExecutionComponent, Environment)
+addChannel(String, Channel) : boolean
+broadcast(String, Object) : boolean
+broadcastIncludeSelf(String, Object) : boolean
+removeChannel(String) : boolean
+replaceChannel(String, Channel) : boolean
+send(UniqueIdentifier, String, Object) : boolean
AdvancedSensors
-agent : AbstractGaaAgent
-breeze : BreezeSensor
-smell : SmellSensor
-sonar : RobotSonar
-sparkleSensor : SparkleSensor
+AdvancedSensors(IExecutionComponent, Environment)
+getFailure() : double
+isGoldAtLocation(LocationData) : boolean
+isLocationSearchable(LocationData) : boolean
+isWumpiAtLocation(LocationData) : boolean
+readOthersMaps()
+readSensors() : boolean
+reset()
+sendOthersMaps()
+setLocationUnsearchable(LocationData)
AdvancedMovement
-agent : AbstractGaaAgent
-collisionCount : int
-destinationLocation : LocationData
-gps : GPSImpl
-navigator : Navigator
-oldLocation : LocationData
-robotMovement : RobotMovement
+AdvancedMovement(IExecutionComponent, Environment)
+getCurrentLocation() : LocationData
+getDestinationLocation() : LocationData
+getFailure() : double
+getOldLocation() : LocationData
+isAtDestinationLocation() : boolean
+move() : boolean
+reset()
+setDestinationLocation(LocationData)
-getNextMovement() : Direction
-handleCollision() : boolean
AdvancedGoldGrabber
-agent : AbstractGaaAgent
-goldGrabber : GoldGrabber
+AdvancedGoldGrabber(IExecutionComponent, Environment)
+grab() : boolean
+getFailure() : double
+hasGold() : boolean
+putGoldIntoBin() : boolean
+reset()

Figure 3 – Capabilities Package Class Diagram

3.2.3 Class Descriptions

3.2.3.1 AdvancedBazooka

The AdvancedBazooka capability extends the existing Bazooka capability with targeting capabilities and ammo counting. When a new target is set on the capability, a set of prioritized firing locations is generated. The agent should navigate to each of these, in turn, and try to fire upon the target Wumpi. At the first targeting location the agent can fire from, the agent should take the shot and then sense to see if the target Wumpi has been killed.

3.2.3.2 AdvancedCommunication

The AdvancedCommunication capability extends the existing CommunicationsImpl capability. No new abilities are added through this extension, but all public methods are annotated so that they can be executed through the GoalCapabilityMap. This capability can send or receive messages on specific channels, or send and receive broadcast messages. It is used by this system to synchronize map information between agents.

3.2.3.3 AdvancedGoldGrabber

The AdvancedGoldGrabber capability extends the existing GoldGrabber capability. It does not add any new abilities, but simply annotates the GoldGrabber’s public methods so that they can be executed through the GoalCapabilityMap. This capability can grab gold, report if it has gold, and release gold.

3.2.3.4 AdvancedMovement

The AdvancedMovement capability wraps the more primitive RobotMovement and GPSImpl and extends them with advanced mapping and route planning capabilities. Together with the AdvancedSensors, these capabilities make up the core of the capabilities used by the demonstration agent architecture. AdvancedMovement is used by every Role to determine the agent’s current location, set goal movement locations, plan routes between locations, and then actually execute those routes.

3.2.3.5 AdvancedSensors

The AdvancedSensors capability internally wraps the Breeze, Smell, and Sparkle sensors. It also encapsulates the sonar capability. It is responsible for aggregating sensor input from a variety of sources, interpreting it, and then storing it in the agent’s map database. It is also responsible for sharing this map information with other agents, and merging their map data with the agent’s map data.

3.2.3.6 OmacsInterface

The OmacsInterface is a special capability that allows RoleLevelGoalModels to communicate with the rest of the OMACS system. It currently only allows for the construction and assignment of a single ReturnGoldGoal. However, in the future it could be extended to provide generic OMACS interface capabilities, such as role assignment, failure reporting, and reorganization. All of these tasks could be executed through existing RoleLevelGoalModels.

3.3 Role Interpreter

3.3.1 Description

The Role Interpreter component is the heart of the system. It is responsible for parsing the role plan definitions, constructing a runtime GoalTree, executing that GoalTree through the GoalCapabilityMap, and then reporting the status of the role execution to the rest of the system. The module consists of four major classes: The RoleLevelGoalModelImpl, the RoleInterpreterImpl, the GoalCapabilityMapImpl, and the GoalCapabilityMapFactory. The package exports the RoleLevelGoalModel, RoleInterprerter, and GoalCapabilityMap interfaces to the rest of the system.

At startup the system constructs a GoalCapabilityMap for each Agent instance using the GoalCapabilityMapFactory to parse the mapping XML. When the agent goes to execute its assigned role, it will lazily initialize a RoleInterpreter, which, in turn, will lazily initialize a new RoleLevelGoalModel by parsing the goal definition using the GMoDS parser. The agent then repeatedly calls execute() on the RoleInterpreter. Each time execute is called, an active goal from the RoleLevelGoalModel is invoked through the GoalCapabilityMap. This causes the capability method that is mapped to the given goal name to be executed through the Java reflection API.

3.3.2 Class Diagram

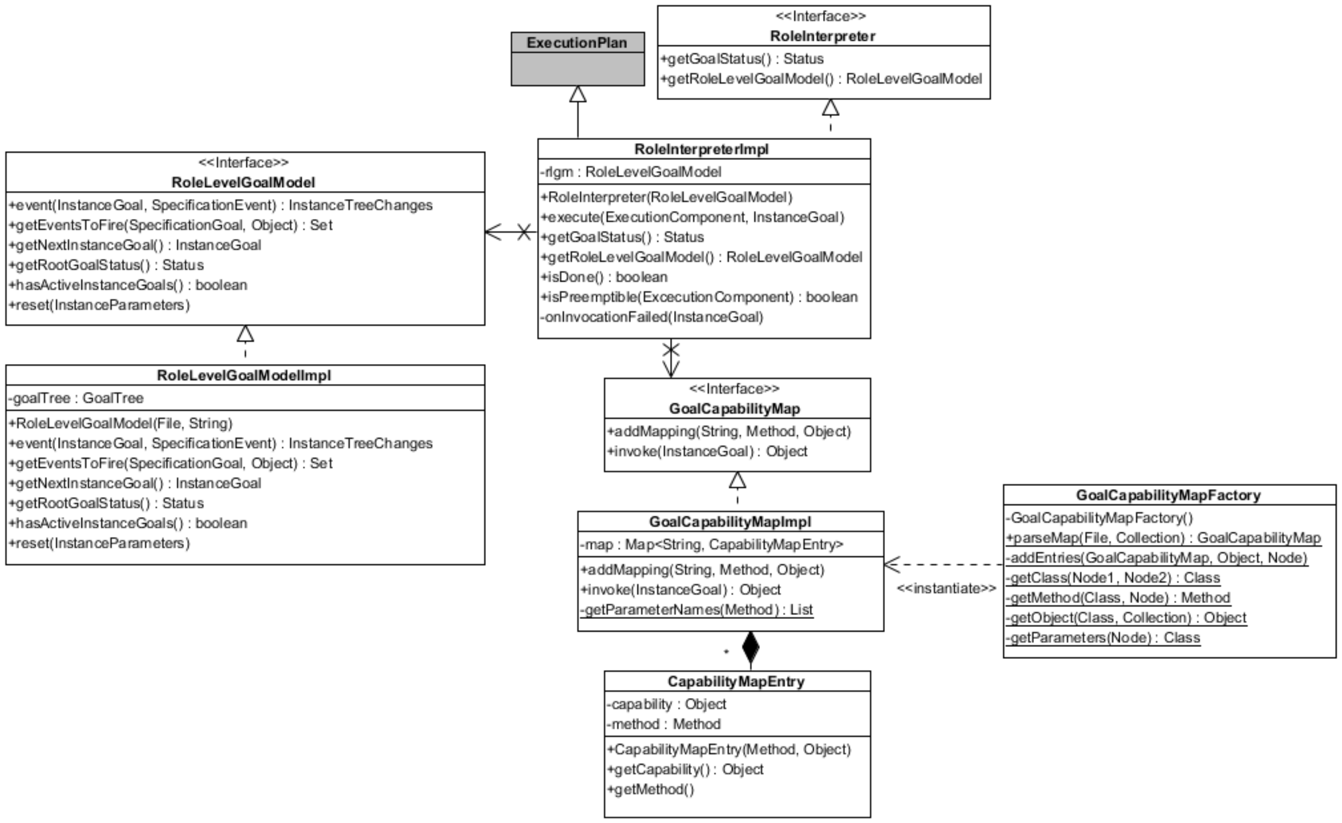
ExecutionPlan
<<Interface>>
RoleInterpreter
+getGoalStatus() : Status
+getRoleLevelGoalModel() : RoleLevelGoalModel
RoleInterpreterImpl
-rlgm : RoleLevelGoalModel
+RoleInterpreter(RoleLevelGoalModel)
+execute(ExecutionComponent, InstanceGoal)
+getGoalStatus() : Status
+getRoleLevelGoalModel() : RoleLevelGoalModel
+isDone() : boolean
+isPreemptible(ExcecutionComponent) : boolean
-onInvocationFailed(InstanceGoal)
<<Interface>>
RoleLevelGoalModel
+event(InstanceGoal, SpecificationEvent) : InstanceTreeChanges
+getEventsToFire(SpecificationGoal, Object) : Set
+getNextInstanceGoal() : InstanceGoal
+getRootGoalStatus() : Status
+hasActiveInstanceGoals() : boolean
+reset(InstanceParameters)
RoleLevelGoalModelImpl
-goalTree : GoalTree
+RoleLevelGoalModel(File, String)
+event(InstanceGoal, SpecificationEvent) : InstanceTreeChanges
+getEventsToFire(SpecificationGoal, Object) : Set
+getNextInstanceGoal() : InstanceGoal
+getRootGoalStatus() : Status
+hasActiveInstanceGoals() : boolean
+reset(InstanceParameters)
<<Interface>>
GoalCapabilityMap
+addMapping(String, Method, Object)
+invoke(InstanceGoal) : Object
GoalCapabilityMapImpl
-map : Map<String, CapabilityMapEntry>
+addMapping(String, Method, Object)
+invoke(InstanceGoal) : Object
-getParameterNames(Method) : List
<<instantiate>>
GoalCapabilityMapFactory
-GoalCapabilityMapFactory()
+parseMap(File, Collection) : GoalCapabilityMap
-addEntries(GoalCapabilityMap, Object, Node)
-getClass(Node1, Node2) : Class
-getMethod(Class, Node) : Method
-getObject(Class, Collection) : Object
-getParameters(Node) : Class
*
CapabilityMapEntry
-capability : Object
-method : Method
+CapabilityMapEntry(Method, Object)
+getCapability() : Object
+getMethod()

Figure 4 – Role Interpreter Package Class Diagram

3.3.3 Class Descriptions

3.3.3.1 RoleLevelGoalModelImpl

The RoleLevelGoalModelImpl is a façade for the GMoDS GoalTree. It exposes a specialized set of methods that are used by the RoleInterpreter to manipulate the GoalTree during Role execution.

The getNextInstanceGoal() method is used to return an arbitrary leaf-level goal from the set of active goals. This is used by the RoleInterpreter to decided which goal should be invoked from its execute method.

The getEventsToFire() method is one of the most important methods in the entire system, as it allows conditional execution to take place. It uses the return value from recently invoked goal, along with the set of outgoing events from a given goal and decided which ones should be fired. The exact semantics of this method are defined in the formal specification.

3.3.3.2 RoleInterpreterImpl

The RoleInterpreterImpl brings the Role's RoleLevelGoalModel and the Agent's GoalCapabilityMap together to actually perform Role execution. It is also responsible for reporting the status of the top-level goal that the agent is attempting to achieve by playing this Role.

3.3.3.3 GoalCapabilityMapImpl

The GoalCapabilityMapImpl is a mapping of goal names to CapabilityMapEntry objects. It essentially provides a mapping from GMoDS goals to actual Java methods. When the invoke() method is called with a given InstanceGoal, the GoalCapabilityMap is used to map that InstanceGoal to a specific method on a specific capability object. In addition, the GoalCapabilityMap takes care of matching up the InstanceParameters of an InstanceGoal with the formal parameters required by the method it is trying to invoke.

3.3.3.4 CapabilityMapEntry

The GoalCapabilityMapEntry is a simple tuple object that represents a Method-Object pair in the GoalCapabilityMap. These entries are not really used outside of the GoalCapabilityMapImpl.

3.3.3.5 GoalCapabilityMapFactory

The GoalCapabilityMapFactory class is a factory class that is responsible for constructing a new GoalCapabailityMap from a given XML file and a list of capability objects. It takes care of parsing the XML file, matching capability classes with runtime objects, and determining which methods on those objects should be invoked by which GMoDS goal names. This XML file is formally defined in the GoalCapabilityMap.xsd file.

3.3.4 Interface Specification

GoalCapabilityMap

Signature	`addMapping(s : String, m : Method, c : Object)`
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 : InstanceGoal) : Object`
Purpose	Invokes the given InstanceGoal (method) on the object that maps to the given goal's name in the map.
Pre-conditions	The given InstanceGaol is not null. A mapping whose key matches the InstanceGoal's name that takes the given InstanceGoal's 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	`getRootStatus() : Satus`
Purpose	Returns the execution status of the root goal
Pre-conditions	None

Post-conditions	ACTIVE is returned if the root goal is still being pursued. ACHIEVED is returned if the root goal is achieved. FAILED is returned if the root goal was failed to be achieved.
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	A leaf-level InstanceGoal from the set of active InstanceGoals whose preconditions have been met has been returned.

RoleInterpreter

Signature	`getGoalStatus()`
Purpose	Returns the execution status of the root goal
Pre-conditions	The agent and goal are not null. The given goal is a top-level organizational goal.
Post-conditions	ACTIVE is returned if the root goal is still being pursued. ACHIEVED is returned if the root goal is achieved. FAILED is returned if the root goal was failed to be achieved.
Signature	`getRoleLevelGoalModel() : RoleLevelGoalModel`
Purpose	Returns the RoleLevelGoalModel associated with this interpreter
Pre-conditions	None
Post-conditions	The RoleLevelGoalModel associated with this interpreter has been retured

3.4 Roles

3.4.1 Description

The Role component's responsibility is to provide an adapter between the Role representation used by OMACS in WumpiWorld, and the Role Interpreter. The parent class, AbstractGaaRole defines a method to return a statically defined role priority (to make the agent architecture as simple as possible). The public interface exported by this component, the GaaRole, provides a method to return the RoleLevelGoalModel associated with a Role.

Roles encapsulate the behavior an agent should perform while pursuing its assigned goal. They are parameterized by instance goals and their instance parameters. In our system, role behavior is entirely defined by the RoleLevelGoalModel. No additional code is used to specify behaviors.

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.

3.4.2 Class Diagram

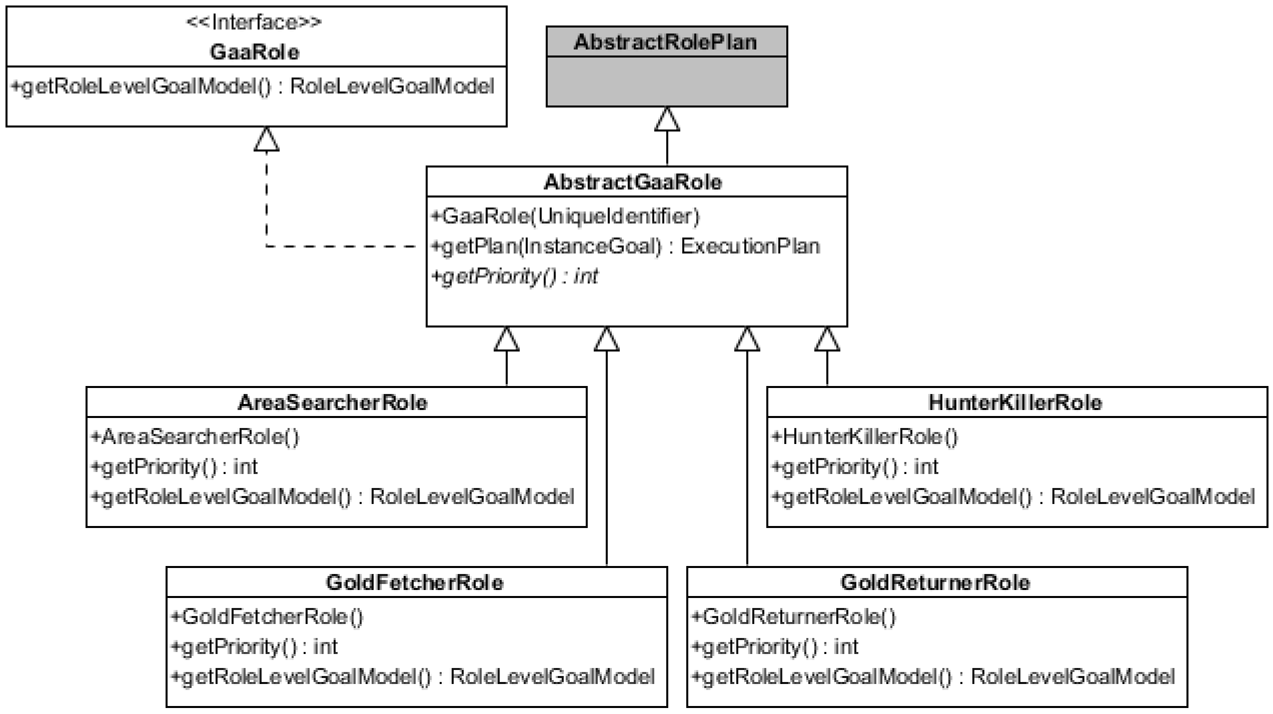
<<Interface>>
GaaRole
+getRoleLevelGoalModel() : RoleLevelGoalModel
AbstractRolePlan
AbstractGaaRole
+GaaRole(UniqueIdentifier)
+getPlan(InstanceGoal) : ExecutionPlan
+getPriority() : int
AreaSearcherRole
+AreaSearcherRole()
+getPriority() : int
+getRoleLevelGoalModel() : RoleLevelGoalModel
HunterKillerRole
+HunterKillerRole()
+getPriority() : int
+getRoleLevelGoalModel() : RoleLevelGoalModel
GoldFetcherRole
+GoldFetcherRole()
+getPriority() : int
+getRoleLevelGoalModel() : RoleLevelGoalModel
GoldReturnerRole
+GoldReturnerRole()
+getPriority() : int
+getRoleLevelGoalModel() : RoleLevelGoalModel

Figure 5 – Role Package Class Diagram

3.4.3 Class Descriptions

AbstractGaaRole – The AbstractGaaRole is the base role. It defines the method that is used to create a static priority mapping between roles for the demonstration architecture. In addition, it contains code to lazily initialize the RoleInterpreter whenever the getPlan() method is called. This method will call the abstract getRoleLevelGoalModel() method to lazily initialize the RoleLevelGoalModel that is associated with the concrete Role class that has been instantiated.

AreaSearcherRole – The AreaSearcherRole is the lowest priority role in the system. It is defined by the AreaSearcherRole.goal model. It is responsible for navigating the agent to a previously unsearched location and then searching it for Wumpi and Gold.

GoldFetcherRole – The GoldFetcherRole is the second highest priority role in the system. It is defined by the GoldFetcherRole.goal model. It is responsible for navigating the agent to a location that contains gold and picking it up. This role will instantiate a new ReturnGold goal if gold is successfully picked up.

GoldReturnerRole – The GoldReturnerRole is the highest priority role in the system. It is defined by the GoldReturnerRole.goal model. It is responsible for navigating the agent back to the gold drop off location, the bin. It is also responsible for making sure that the agent actually releases the gold into the bin successfully.

HunterKillerRole – The HunterKiller role is the third highest priority role in the system. It is defined by the HunterKillerRole.goal model. It is responsible for generating a set of firing locations for the given target Wumpi and then making sure that the agent actually fires at the Wumpi once a suitable firing location is reached.

3.4.4 Interface Specification

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.