User Manual

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 serves a user manual for the GMoDS-based Runtime Agent Role Interpreter. It contains information about how to obtain the interpreter's source code and executable binary. It also contains information about how to setup and run the Agent Architecture Demo that shows the Role Interpreter in action. Finally, it contains information about how to design your own Role Level Goal Models, Capabilities, and Goal Capability Maps.

2 Installation

2.1 Prerequisites

The GMoDS-based Runtime Agent Role Interpreter requires the following prerequisites in order to run:

- Java Runtime Environment 6.0 or higher

2.2 Required Files

To obtain the GMoDS-based Runtime Agent Role Interpreter, download the final GAA source archive from http://people.cis.ksu.edu/~kylhill/phase_3/gaa.zip. The only required files from within the archive are gaa.jar, and the contents of the configs, models, and scenarios directories. The other files within the archive are the project source code and documentation.

Alternatively, the GAA project and all of its dependencies can be checked out from the K-State CIS projects CVS server at projects.cis.ksu.edu/cvsroot/gaa. The setup and use of this CVS repository are outside the scope of this document.

2.3 Setup

Extract the final GAA source archive, gaa.zip, to the desired location on your computer. No other installation steps are required.

3 Running the Agent Architecture Demo

The GMoDS-based Runtime Agent Role Interpreter contains an agent architecture demonstration in addition to its basic Role Interpreter functionality. To run this demonstration, execute the following command on a command line from the directory where you extracted the project archive:

```
java -jar gaa.jar scenarios/WumpiWorldFull.xml configs/GoalCapabilityMap.xml
```

Figure 1 – Sample Demonstration Invocation

The first parameter to gaa.jar provides the path to the CROS environment description XML file. This file provides information describing the environment the agents will exist within,

as well as their starting locations and other parameters. Several sample scenarios have been provided within the scenarios subdirectory.

The second parameter to gaa.jar provides the path the Goal Capability Map description XML file. This file provides information to map specification goal and parameter names from Role Level Goal Models to actual capability method calls at runtime. Instructions for producing your own Role Level Goal Models and Goal Capability Maps are given in section 4.

Figure 2 – Agent Architecture Demonstration

A window similar to the one shown in Figure 2 should appear after invoking the given command. This window is the standard a CROS environment display for the Wumpi World scenario. This scenario was given as a final project to students in CIS-844 during the fall of 2010. The goal of this scenario is to gather all of the gold in the world and kill all of the Wumpi monsters that guard it. While carrying out these tasks, agents, represented by the robot icons in the lower-left must avoid falling into pit traps or being attacked by the Wumpi monsters. Agents are scored based on how efficiently they search the map, how many Wumpi are killed, and how much gold is retrieved.

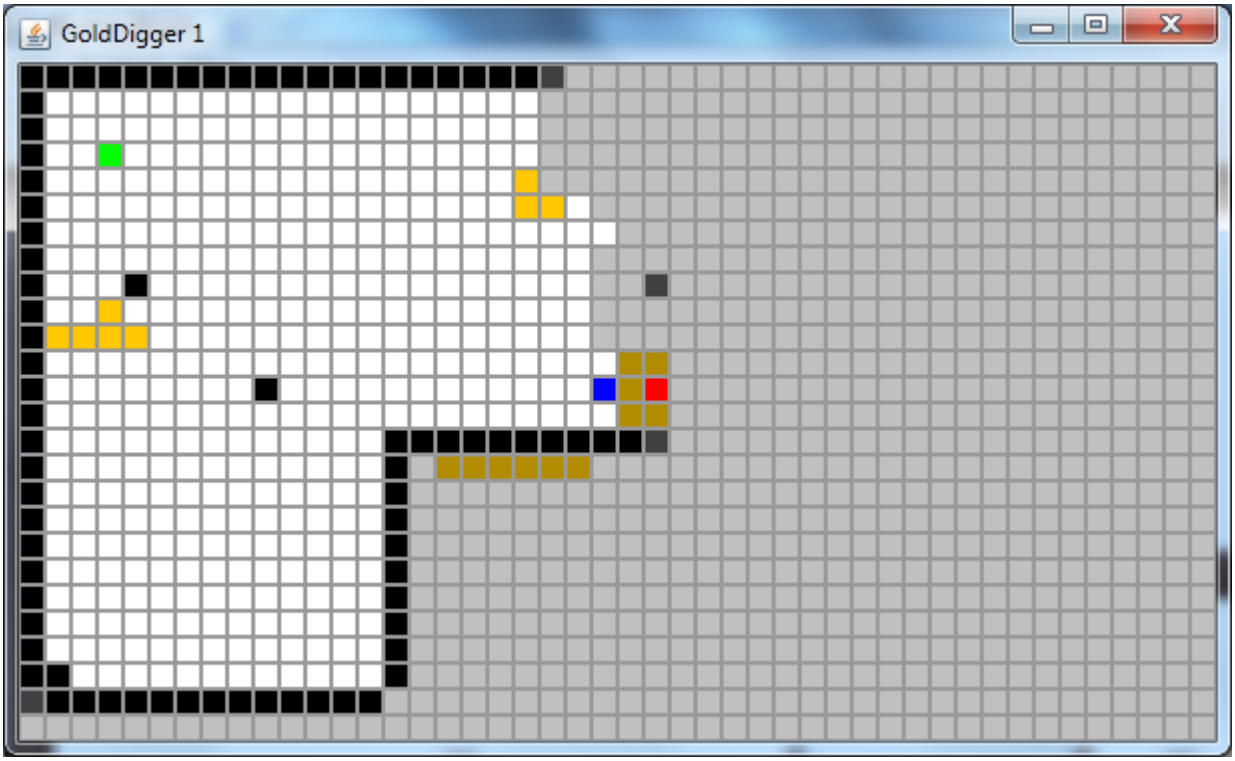

Figure 3 – Agent Map Display

Several other windows, similar to the one shown in Figure 3, should show up as well. These windows show what each agent thinks the world around them looks like. The key for this map is given in Figure 4.

Tile Color	Meaning
White	Searched Tile, Nothing Special
Light Gray	Unsearched Tile
Dark Gray	Unknown Obstacle
Black	Wall
Red	Wumpi
Light Orange	Possible Pit
Dark Orange	Possible Wumpi Danger
Yellow	Possible Gold
Green	Claimed Goal Tile
Blue	Agent, Active
Magenta	Agent, All Goals Complete

Figure 4 – Agent Map Display Key

All agent activity within this scenario is controlled by the Role Level Goal Models supplied within the models subdirectory of the archive. These goal models provide the instructions that agents are to carry out as they pursue their assigned goals. The mappings of these goals to agent capability methods are defined by the Goal Capability Map. The design and construction of these models is discussed in section 4.

4 Designing Models

The GMoDS-based Runtime Agent Role Interpreter is designed to modular and extensible. To achieve that aim, new Role Level Goal Models, Capabilities, and Goal Capability Maps can be supplied by users to enable them to use the Role Interpreter in any multiagent system that they choose.

4.1 Creating New Role Level Goal Models

New Role Level Goal Models can be created to specify new behavior for agents to perform while playing the role the model defines. Role Level Goal Models are simply regular GMoDS Goal Models whose root goal is a leaf-level goal in the top-level system Goal Model. The leaf-level goals of a Role Level Goal Model map directly to capability methods that the agent possesses. Role Level Goal Models can be constructed using the agentTool3 Diagram Editor just like any other GMoDS goal model.

Role Level Goal Models are no different from standard GMoDS Goal Models, except for the following restrictions:

1. The number and names of leaf-goal specification parameters must match with number and names of formal parameters of the methods to which they are to be mapped. If they do not match, runtime exceptions will be thrown when the Goal Capability Map is parsed at startup.

2. Special “true”- and “false”- prefixed event names must be used if conditional execution and, by extension, iteration, is desired. These events only possess special meaning if they originate from a leaf-level goal whose mapped method returns a boolean value. If they are used on a non-boolean method, then they have no special meaning. If the method returns true at runtime, only the true-prefixed events are executed. Likewise, if the method returns false at runtime, only the false-prefixed events are executed.

3. Only one instance goal in a “triggers-cycle” may be active at a time. This is due to the way GMoDS determines how specification goals are completed. A custom patch was applied to GMoDS for this project to allow execute of “triggers-cycles” under most cases. This only means that Role Level Goal Models cannot support parallelism within a Role if conditional execution is also required. Additional work on GMoDS could remove this restriction.

4. At most, only one new specification parameter may appear in an event leaving a given goal. It is assumed that this specification parameter represents the return value from the invoked method. Any and all previous instance and inherited parameters can be passed along as part of the event without any restrictions. There is no real way for the system to deterministically provide values for more than one new specification parameter. If this behavior is required, then leaf-level goals should be split up into smaller ones that only generate a single new event parameter when executed.

4.2 Existing Capabilities

The following existing capabilities are present in the system and may be used within Role Level Goal Models and mapped to by Goal Capability Maps.

4.2.1 Advanced Bazooka

The AdvancedBazooka capability extends the existing Bazooka capability with targeting capabilities and ammo counting.

Available Methods
fire(loc : LocationData) : boolean
getNextFiringLocation() : LocationData
hasAmmo() : boolean
hasFiringLocation() : boolean
setTarget(loc : LocationData)

Figure 5 – Advanced Bazooka Methods

4.2.2 Advanced Communication

The AdvancedCommunication capability extends the existing CommunicationsImpl capability. This capability can send or receive messages on specific channels, or send and receive broadcast messages.

Available Methods
broadcast(channelID : String, content : Object) : boolean
broadcastIncludeSelf(channelID : String, content : Object) : boolean
receive() : Boolean
removeChannel(channelID : String) : boolean
send(agentID : UniqueIdentifier, channelID : String, content : Object) : Boolean

Figure 6 – Advanced Communication Methods

4.2.3 Advanced Gold Grabber

The AdvancedGoldGrabber capability extends the existing GoldGrabber capability. This capability can grab gold, report if it has gold, and release gold.

Available Methods
grab() : boolean
putGoldIntoBin() : boolean
hasGold() : boolean

Figure 7 – Advanced Gold Grabber Methods

4.2.4 Advanced Movement

The AdvancedMovement capability wraps the more primitive RobotMovement and GPSImpl and extends them with advanced mapping and route planning capabilities. AdvancedMovement is used to determine the agent's current location, set goal movement locations, plan routes between locations, and then actually execute those routes.

Available Methods
`getCurrentLocation() : LocationData`
`getDestinationLcation() : LocationData`
`getOldLocation() : LocationData`
`isAtDestinationLocation() : boolean`
`move() : boolean`
`setDestinationLocation(loc : LocationData)`

Figure 8 – Advanced Movement Methods

4.2.5 Advanced Sensors

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.

Available Methods
`isGoldAtLocation(loc : LocationData) : boolean`
`isLocationSearchable(loc : LocationData) : boolean`
`isWumpiAtLocation(loc : LocationData) : boolean`
`isAtDestinationLocation() : boolean`
`readOthersMaps()`
`readSensors() : boolean`
`sendOthersMaps()`
`setLocationUnsearchable(loc : LocationData)`

Figure 9 – Advanced Sensors Methods

4.2.6 OMACS Interface

The OmacsInterface is a special capability that allows Role Level Goal Models 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.

Available Methods
`createReturnGoldGoal()`

Figure 10 – OMACS Interface Methods

4.3 Creating New Capabilities

New capabilities can be added to the system very easily. To make a capability that can be used within a Role Level Goal Model and Goal Capability Map, simply state that the capability implements the empty GaaCapability interface and then annotate all parameters of all methods that you wish to export to the Goal Capability Map with the custom @Name annotation. The name you give the parameter is the name that must be called out in the Goal Capability Map. As sample annotated method signature is provided below for reference:

```
public final boolean addChannel(@Name("channelID") final String
channelID, @Name("channel") final CommunicationChannel channel);
```

Figure 11 –Example Method Annotation

4.4 Creating New Goal Capability Maps

Creating new Goal Capability Maps is simple and straightforward. When you have finished creating a new Role Level Goal Model and any new capability classes, check all leaf-level goals and see if they already contain a mapping to capability method. If they do not, then you will need to provide such a mapping. If any mappings are missing, a runtime exception will be thrown indicating which mappings need to be established.

The sample XML file GoalCapabilityMap.xml provides mappings for all existing capabilities to goal names used by the Role Level Goals Models in within the project models subdirectory. If you wish to use different leaf-level goal names to map to existing capability methods, either rename the existing mapping's goal_name attribute, or provide a new mapping with a different goal_name attribute. Similarly, if you create a new capability class, you will need to construct mappings for each public method that you with to use from your RoleLevelGoalModels.

```
 <!-- AdvancedGoldGrabber -->
 <Capability class="AdvancedGoldGrabber"
package="edu.ksu.cis.macr.simulator.capabilities">
        <Method goal_name="grab" declared_name="grab"/>
        <Method goal_name="putGoldIntoBin" declared_name="putGoldIntoBin"/>
        <Method goal_name="hasGold" declared_name="hasGold"/>
 </Capability>
```

Figure 12 – Example Goal Capability Mapping

4.4.1 Goal Capability Map Schema

For easy reference, the Goal Capability Map Schema is provided below to aid in the construction of valid mappings. To check your Goal Capability Map, simply validate your XML file with the schema provided below. This will guarantee that the file will at least be parsed correctly when given the Role Interpreter.

Figure 13 - GoalCapabilityMap

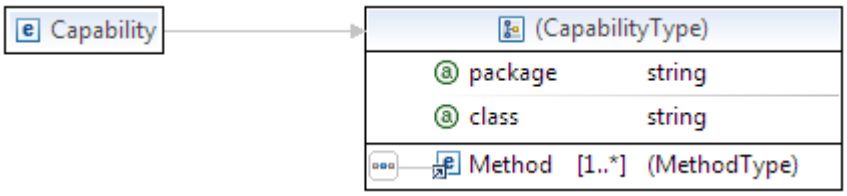

Figure 14 - Capability

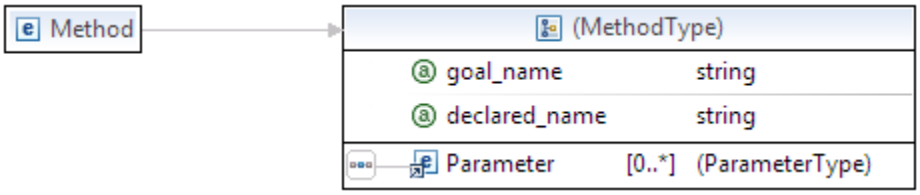

Figure 15 - Method

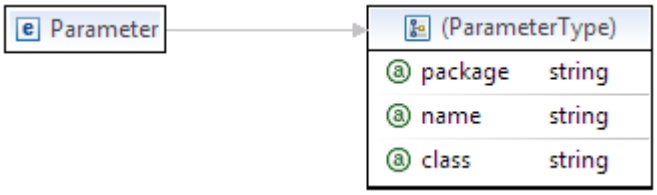

Figure 16 - Parameter

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="Parameter">
		<xs:complexType>
			<xs:attribute name="package" type="xs:string" use="required"/>
			<xs:attribute name="name" type="xs:string" use="required"/>
			<xs:attribute name="class" type="xs:string" use="required"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="Method">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="Parameter" minOccurs="0" maxOccurs="unbounded"/>
			</xs:sequence>
			<xs:attribute name="goal_name" type="xs:string" use="required"/>
			<xs:attribute name="declared_name" type="xs:string" use="required"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="GoalCapabilityMap">
		<xs:complexType>
			<xs:sequence>
```

```
                        <xs:element ref="Capability" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="Capability">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="Method" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="package" type="xs:string" use="required"/>
                    <xs:attribute name="class" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Figure 17 – The Goal Capability Map Schema