Emergent Architecture Design

GROUP: GOAL-Integration

May 12, 2015

1 Introduction

This group will create an integration of the Gamygdala engine in the programming language GOAL.

1.1 Goals

In this project we have multiple goals. Besides the goals between the different groups that can be found in the product vision document we also have goals that are group specific. We think that it is very important for us that we have good code quality. Also functionality is important with this we mean focusing on creating code that works well and that has the necessary features to bring this proof of concept to a good end.

1.1.1 Source code quality

We think that it is important to have readable code that is well written with the correct design patterns. Other software engineering principles we have learned so far should also be used appropriately and correctly. For example: Method sizes.

1.1.2 Functionality

By testing in different ways we want our code to have all the necessary features working well. Since our job is to create an integration, functionality is the most important goal for our team. It should be noted though that this is a proof of concept, so while we will focus on making sure the other groups receive the necessary features some of them might be implemented in a less nice way in order to be able to implement them quickly. An example of this is the emotion configuration file. Instead of adding the definitions of this file to the agent and MAS files we decided to use a separate file since this meant we could more easily create our own parser for it since adapting the files of GOAL would require learning a lot about ANTLR.

2 Software Architecture

When a user is creating a Virtual Human in the programming language GOAL, they should be able to use the Gamygdala engine in an easy and understandable way.

2.1 Programming languages

We will use Java. This is because GOAL is written in Java. Gamygdala is written in Javascript, but we will use a port to java so that we do not need to work with Javascript.Mostly we will be working in the source code of GOAL and find a way to build an integration of the engine in GOAL itself.

2.2 Tests

Testing of the software will be done mostly in the JUnit framework since we are writing Java code. The port between javascript and java also needs to be tested thoroughly, but this won't be done by this group.We aim for a high test coverage, our definition of done requires at least 75 % test coverage, but this is a minimum a higher test coverage is always welcomed of course. Also we will have continious integration during our project which means that everytime our project gets built, we get a report of tests but also source code quality discussed in the next section.

2.3 Code Quality

During this project our group wil use the pull-request method to develop code. This wil make sure that the quality is maintained since nothing can be merged to the Master without permission of atleast two other students. It works like this. You create your own branch of the master and then you start coding. When you are finished you push your code to the branch (which is a mirror of the master except for your new implementations). After this you make a request of your branch being pulled to the master (hence pull requests). Then your colleague students can see this on GitHub and comment,reply and approve or dissaprove. When atleast 2 other student finally agree after you made adjustments they can allow you to merge your branch with the master. It helps our group in 2 ways. First of all it makes sure we do code reviews and think about what other are creating and if it is helpful for our goals or not. Secondly, by using this method less merge conflicts will happen. By using tools like Checkstyle we can maintain readability,good line length and method size violations.

2.4 Packages

This is hard to think about from the beginning for our context project since we are using GOAL as our starting point. We will maintain the structure of GOAL and try to fit in our implementation. All the source code and Package of GOAL

can be found on their GitHub project.

Throughout the project we will keep updating this list when we discover new places we need to make adjustments in. So far we've discovered that we will need to modify the following packages (or already have made adjustments):

2.4.1 Runtime

The packages from the runtime repository:

goal.core.mentalstate: This needs to be modified to add the new database for the emotions. This is also were the database for the goals etc. is located. We will need to add a new class representing this emotion database and we will need to modify the MentalState class to add this database to the mental state of an agent.
goal.core.runtime: In this package we will need to modify the RuntimeManager. This class can be seen as the controller of GOAL it contains the other services like the MessagingService and the EnvironmentService. This class will need to be modified to add our Gamygdala instance to it. Here we will also be able to add all the agents to the Gamygdala instance and use our emotion configuration file to add all the predefined settings to the Gamygdala instance.
goal.core.runtime.service.agent: We will need to modify this package so that each cycle of GOAL the new goals are inserted in Gamygdala and to make sure that the emotions are updated according to the beliefs/percepts of the agents. For this we will specifically need to modify the RunState class.
goal.tools: The class PlatformManager resides in this package, this class has methods that retrieve the parsed version of MAS files and agent files. This means that we will need to modify this class to also retrieve the parsed version of the emotion configuration file.

2.4.2 Grammar

The packages from the grammar repository:

languageTools.parser.relationParser: This is a subpackage we have added to the parser package to parse our emotion configuration files. It contains the parser itself and the classes needed to store the information we parsed.
languageTools.exceptions.relationParser: A package we added to contain the exceptions thrown by our parser.
languageTools.program.agent.mas: We will need to modify the structure of the MAS file to include our emotion configuration file (in a similar way to how the MAS file contains the agent files). This specifically means adjusting the MASProgram class to contain this file. To achieve this a lot of other packages will need to be changed as well though (as outlined in some of the previous pacakges).

Glossary

Gamygdala An emotional engine created to simulate emotions for virtual identities[2].. 1

GOAL A programming language developped at TU Delft for creating Virtual Humans[1].. 1, 2

JUnit This is a framework to build testsuites in Java.. 1

Master The master branch of a repository is the main of repository. Your main stable version.. 2

Package a package it actually a map in which a part of your code is put. This is handy for keeping a good structure.. 2

Virtual Human A virtual human is an agent that uses logical rules to derive the wanted actions and can percept events from the environment. It is AI.. 1

References

[1] GOAL programming language `http://ii.tudelft.nl/trac/goal`

[2] Gamygdala emotion engine `http://ii.tudelft.nl/~joostb/gamygdala/index.html`

[3] ANTLR, another tool for language recognition `http://www.antlr.org/`