Architecture design

WhySoSerious

June 17, 2016

Contents

1 Introduction

This document provides a high-level description of the final product and the systems it uses. The final product is a combination of the virtual human and the EIS connector. The virtual human will operate autonomously in an environment within the Tygron Engine and is developed in the GOAL Agent Programming Language. GOAL uses a knowledge model with percepts and beliefs. A GOAL agent can perceive things in the environment and those can be stored as percepts. It can also make use of percepts and earlier beliefs to infer indirect information about the environment and store them as beliefs.
The virtual human has the role of DUWO, DUWO is a housing corporation for students that aims to provide many rental houses and rooms for students at a relatively affordable price. The Tygron-EIS connector connects the virtual human with the Tygron system and is developed in Java.

In the first section the design goals of the project are defined. In the second section the software architecture views of the project are discussed. Among those views are subsystem decomposition, hardware/software mapping, persistent data management and concurrency.

1.1 Design goals

The design goals can be split in four different aspects. These are performance, availability, code quality and reliability. In the following sections the four different aspects are elaborated.

1.1.1 Performance

The virtual human should not be overwhelmed with all the information it gets from the environment. When this does happen it is possible that the virtual human is busy for too long determining what kind of action it should perform. The virtual human is then acting on old information which can be imprecise. This behavior is unacceptable.

1.1.2 Availability

It is important to deliver a working system every week so Tygron can see what the plans are for the next sprint and what has been implemented last sprint. If Tygron does not like a new feature it can be removed from the plan, instead of spending a lot of useful time extending this feature.

1.1.3 Code quality

All code within a method or module should be understandable in about one minute, if this is not the case there should be comments that explain the code so it is understandable in one minute. To ensure all code follows the same code conventions, checkstyle[1] is used. This is only possible for the Tygron-EIS connector since the virtual human is developed in GOAL and not Java.
The virtual humans code should be divided into coherent modules with descriptive names. From the names of those modules and their comments it should be clear what their use is.

1.1.4 Reliability

It is very important to continuously test the code. Travis CI is used for continuous integration, results from the latest build of the Tygron-EIS connector can be found here. Travis CI is also used for the

[1] http://checkstyle.sourceforge.net/

virtual human, results can be found here.
Test driven development should be used to ensure that requirements are understood before developing a new feature. Testing methods that are used are unit tests, integration tests and regression tests. This helps finding bugs and unwanted behavior.

2 Software architecture views

The software architecture views explain how the different parts of the system are interconnected. The system is divided into subsystems in the subsystem decomposition and the hardware/software mapping shows how the subsystems connect with each other.

2.1 Subsystem decomposition

An user accessing the Tygron engine does so with client software on his or her computer. This client communicates with the Tygron servers where all of the processing is done. The virtual human will communicate with the simulation via the Tygron-EIS connector[2]. This connector gives the agent percepts which it will process into beliefs. Based on these, the virtual human will return actions to the EIS connector sending them through the Tygron API directly into the Tygron servers.

- Servers

  The servers are managed by Tygron and this is where the simulation takes place, the clients use the data that the Tygron server provides to visualize the simulation.

- Tygron API

  The API is part of the server software, and allows a user to incorporate functionality from another application within their own application, e.g. to send instructions and/or display data[3]. The Tygron-EIS connector uses this API to connect the virtual human to the server.

- Tygron-EIS connector

  This is a piece of software that handles the connection between the Tygron API and the GOAL language.

- GOAL agent

  The GOAL agent is a program written in GOAL that can simulate an entity with intelligence. In this project it simulates a human stakeholder within the Tygron game.

for a visual representation take a look at figure 3

[2] `https://github.com/Denpeer/tygron`
[3] `http://support.tygron.com/wiki/REST_API`

2.2 GOAL agent modules

Our goal agent has 4 main modules that decide how the agent will act with 3 of those modules representing the current state of the budget. Every cycle the main module is activated first. This module executes the commands that should be should be executed regardless of the budget. The main module will then decide which submodule will run.

Since the agent does not have any place to build student housing at the start of the session, the agent adopts the goal 'goalDemolish'. When this goal is adapted the agent will run the "insufficient space" module. When entering this module the agent will demolish what it doesn't need to improve its indicators, like gardens.

When there is sufficient space, the budget current budget is the measure to decide which module the agent will run. A specific threshold is set to a certain percentage of the target budget. This percentage is currently set to 120%. When our budget is above this threshold the "sufficient budget" module will run. This module will focus on gaining as much land and building as many student houses as possible. When our current budget goes below the threshold, the "low budget" module will run instead. This module will only build or trade when this gives us a significant increase in our total indicator score. Lastly the "insufficient budget" module will not build or buy anything and will even try to sell land if this doesn't affect our total indicator score in a significant way.

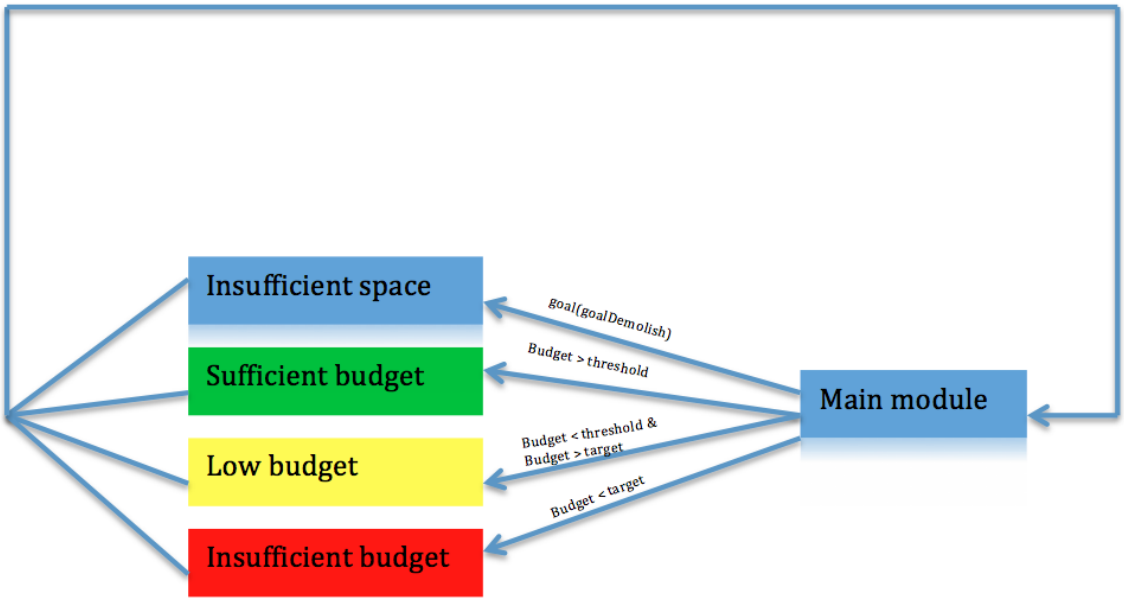

Figure 1: A graph of the different modules working within our agent.

2.3 Connector

2.3.1 Percepts

Percepts in the connector are the Prolog representations of internal Java Data structures. This means each percepts needs its own Translator, a class extending Java2Parameter⟨Class⟩ where Class is the

internal Data structure that is represented. An instance of this Translator will have to be added in the j2p. It will then automatically be inserted in the installTranslators method, which will be called on initialization.

When the Translator has been added, the percept has to be added to a switch in ContextEntityEventHandler.notifyListener. This case should fetch the represented data from the Tygron SDK, and call createPercepts with it. It will then generate the percepts using the translator.

2.3.2 Actions

Adding a custom action is now possible by adding the constructor of a new custom action in the ActionContainter class. A new custom action should implement the CustomAction interface. The CustomAction interface has two functions call and getName:

The call function is called when the agent wants to perform this custom action. It returns a percept, the virtual human sees this percept the next cycle and can use it for furthing processing. The getName function returns a string, this string is the name of the action that the agent has to call to perform this action.

Arguments for a custom action should be handled in the call function. There can be zero arguments or in theory infinite arguments.

Actions that are defined by the Tygron SDK are also available to the virtual human. These actions can be found in ParticipantEventType.java. Each action is properly documented, an explanation of the action is given and all arguments that are needed are given with a explanation.

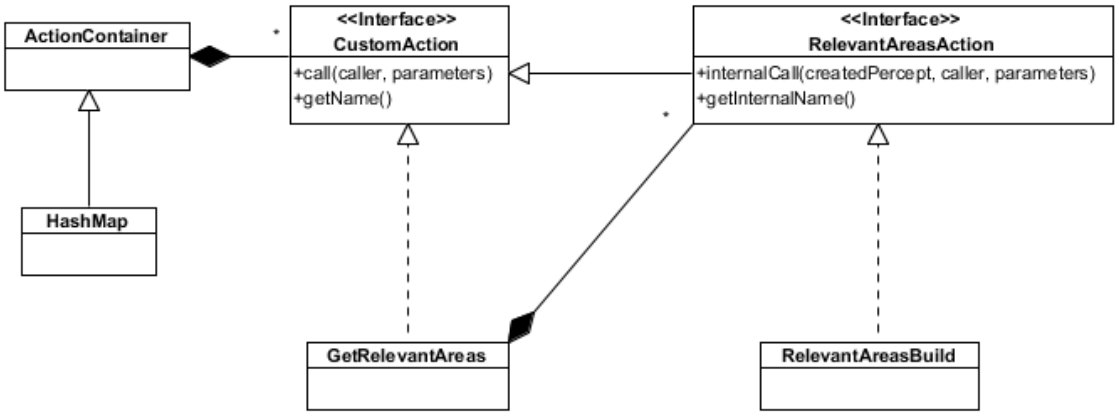

Figure 2: The architecture of our ”custom actions” implementation.

2.4 Hardware/software mapping

The Tygron API runs on Tygron’s servers, while the Tygron-EIS connector and the virtual human are both on the user’s computer. The Tygron-EIS connector uses the Tygron SDK to communicate with the Tygron API. The Tygron SDK and the Tygron API communicate with each other using JSON. The Tygron API runs on the Tygron Server, which is the final back-end of the entire Tygron system. The Tygron Engine connects with the Tygron Server, so it is possible to join a session a virtual human

is in as well. This way it is possible to view the virtual human's actions in real-time. In figure 3 an illustration of the connection between all parts of the process are shown.

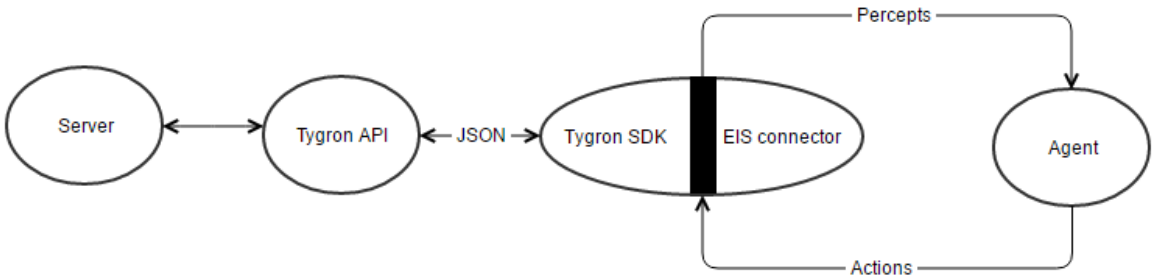

Figure 3: A graph of the interaction between the server and the agent.

2.5 Persistent data management

As the software will be a real-time virtual human, which will operate in short sessions and have no memory of any previous session, there will be no persistent data within the beliefs of the agent. All persistent data is the game environment state which will be kept at the Tygron server at all times, and the source code, that is kept on GitHub.

2.6 Concurrency

The virtual humans will have to be able to communicate with each other. This means that we will have to work together with the other teams in our context to develop a language which all of the virtual humans can use and understand. Virtual humans should also remember their actions so they won't get stuck in a loop. They should also be able to run at the same time, from different computers, and be up to date about what other virtual humans say.

The virtual human also has to be able to adapt to other virtual humans. If the Municipality rejects a certain plan, the virtual human has to abandon that plan, instead of trying it again every cycle. If the virtual human doesn't, it may deadlock. To address this, the virtual human will remember it when the Municipality rejects something, and will try other plans instead of continuously trying the same plan.

The Tygron-API doesn't allow a piece of land to be offered to multiple stakeholders at once, this means that every stakeholder has to correctly and quickly handle request so that it's possible for a stakeholder to iterate over other stakeholders when selling a piece of land.

Interaction between a stakeholder and the municipality requires a bit more attention. Since the municipality has to allow all of our building actions we have to take their interests into account. This means occasionally building a park to improve their environment indicator.

Our getrelevantareas action implemented into the connector will detect new land and return any new area's on which we can build. New land without parks or student housing will be demolished when there is a shortage of land.

3 Glossary

- API

  API means application programming interface. Programs use an API to communicate with external software, usually servers.

- SDK SDK means software developer kit.

- GOAL

  A programming language created at the TU Delft that enables effective programming of logical agents.

- GitHub

  GitHub is a website that enables storage and concurrent development of code for programming teams.

- Tygron

  Tygron is the company that developed the Tygron Engine[4], which is used in this project.

- Tygron Engine

  A program run on the Tygron server that enables multi-user simulated city planning with customizable roles.

- Virtual human

  Virtual human is another term for the GOAL agent that is being developed for this project.

- Unit test

  A unit test tests a single component to verify its behaviour.

- Integration test

  An integration test tests whether multiple components are able to cooperate as expected.

- Regression test

  A regression test tests whether new bugs have been introduced due to updated code.

[4] http://www.tygron.com