Architecture Design

Michiel van den Berg   4391039   michielvandenb
Stefan Breetveld   4374657   sbreetveld
Timo van Leest   4423798   timovanleest
Daan van den Werf   4369556   djvanderwerf
Job Zoon   4393899   jzoon

April 29, 2016

Contents

1 Introduction **3**
1.1 Design goals 3

2 Software architecture views **5**
2.1 Subsystem decomposition 5
2.2 Hardware/software mapping 6
2.2.1 Tygron 6
2.2.2 Our project 6
2.3 Persistent data management 7
2.4 Concurrency 8

3 Glossary **9**

1 Introduction

In this document we'll describe the architecture of our product for the Virtual Humans context project. It should give insight in the design of our system. First we will identify our design goals, which should represent the things that we find to be important for this project.

1.1 Design goals

Manageability

We're working with multiple project groups and have to maintain the same code for the connector. It's important that we keep this code manageable, enabling every group to make modifications as needed, without breaking other parts of the code.

Performance

From our previous MAS project we know that goal code can get really slow when you don't take performance into consideration while coding. We should try to keep our design as efficient as possible to prevent these kind of problems.

Scalability

Although we're developing our agent for just one scenario in this project, there's a change it will be used in larger scenarios and we should design so that this would be possible if ever needed.

Reliability

Since our agent will be used to replace a missing player in a game it's important that this agent is reliable. If during a play session with multiple users the agent fails and the game has to be restarted this is not only frustrating for the players, but can also cost a lot of money since some of the players are well paid employees.

Securability

The agents login to the system of Tygron and have a direct connection to their server running during a game. We should keep this in mind while designing the system.

Availability

There should be a working version at all times, so that when a player is missing for a game the game can be played anyway. Not knowing if an agent is available could cause a whole game to be unplayable because of a missing player.

2 Software architecture views

asdf

2.1 Subsystem decomposition

asdf

2.2 Hardware/software mapping

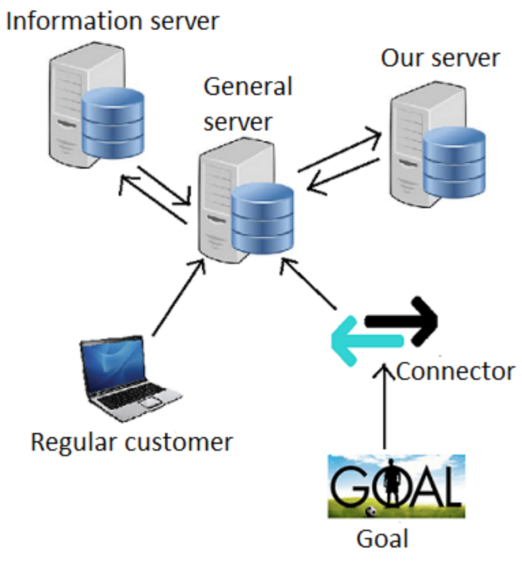
In this section the hardware/software mapping Tygron uses will be explained, as well as the way our agent will be connected to this.

2.2.1 Tygron

Tygron has its own server, from which it communicates with all other entities. For each group of users, there is a seperate server which contains their world and on which those users can work. The general Tygron server knows these other servers and will redirect the user to the particular server. So, when a customer wants to connect to Tygron, it will first connect to this general server. Also, this general server has contact with other servers which are not connected to Tygron. For example, when a customer wants to build a part of a city to play on, the general server will connect to some other servers that give information about this place, like where the buildings are and what the use of those buildings is.

2.2.2 Our project

In our project we have our own server with all TU Delft students. But there is another thing that needs to be different: the way we connect to that server. Since we don't use humans, but agents to play the game, those agents need to connect in some way to the server. To do so we have a connector between our instance and the server. This connector will use the goal code we wrote and will translate this into actions in the game. This way our agent can play the game. The next picture illustrates this.



2.3 Persistent data management

The data management for our project envelops multiple things. We have a part where the data for the game is stored and the data we need for running the agent. The first part is done by Tygron. When a new project is created it is stored in the Tygron database to which the clients can connect via the Tygron server. Any changes made in a session are then updated to the database, which are visible to all the other clients. Figure 1 shows the diagram of the Tygron Engine connected to the database and the GOAL agent.

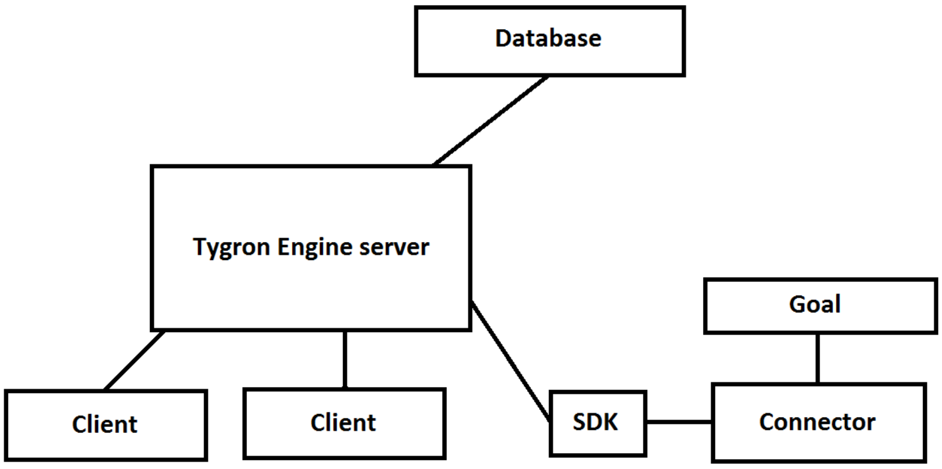


Figure 1: Diagram of the Tygron Engine connected to the database and the GOAL agent.

In GOAL the data the agent uses is stored in multiple databases. The agent has a knowledge base which is the same for every environment that he might find himself in. This data is simply stored as GOAL code in the agent his GOAL files. The rest of the data that the agent uses is dynamic and his state is updated constantly according to the situation that the agent is currently in. Our agent maintains two different databases of facts. One database called the goal base consists of things the agent wants to achieve. The other database is called the belief base and consists of facts that the agent believes are true (now). Figure 2 shows how the databases of the agent are updated.

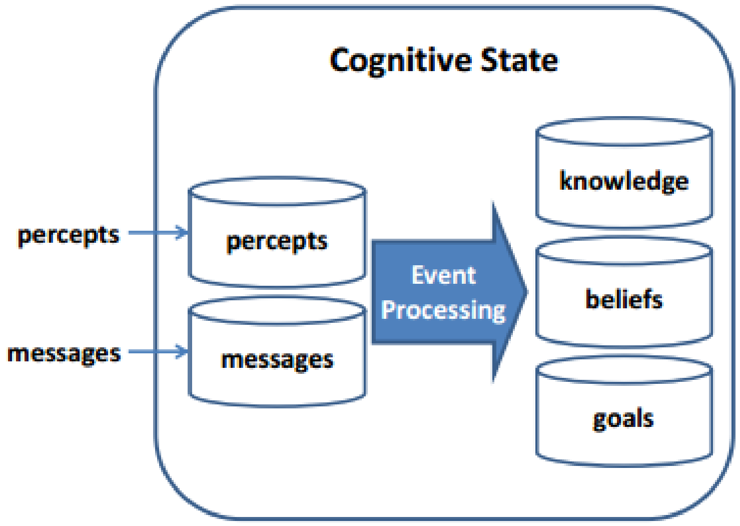


Figure 2: Updating the agent's database.

2.4 Concurrency

3 Glossary