

# 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

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Design goals . . . . .	3
<b>2</b>	<b>Software architecture views</b>	<b>5</b>
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
<b>3</b>	<b>Glossary</b>	<b>9</b>
	<b>References</b>	<b>10</b>

# 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 chance 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

### 2.1 Subsystem decomposition

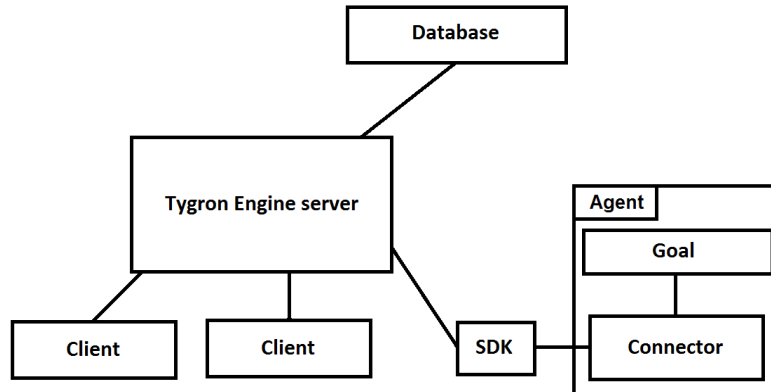


Figure 1: Diagram of the Tygron Engine connected to the database and the GOAL agent.

The architecture is divided into different subsystems. It consists of the database, tygron engine server, several clients and several agents. In this section we will explain what each part is and does.

- **The Database**

The database in the system is used to save all kinds of data. This database stores all the data that is used to build the environment like the roads, buildings, rivers, etc.

- **The Tygron Engine Server (TES)**

The Tygron Engine server is used to serve all the data to the correct clients and agents.

- **A Client**

A client is a human interacting with the Tygron Engine Software. There can be multiple clients per *TES*.

- **The SDK**

The SDK (Software Development Kit) is used to connect agents to the *TES*.

- **An Agent**

An agent is a computer controlled entity that acts like a client. There can be multiple agents per *TES*.

## 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 separate 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

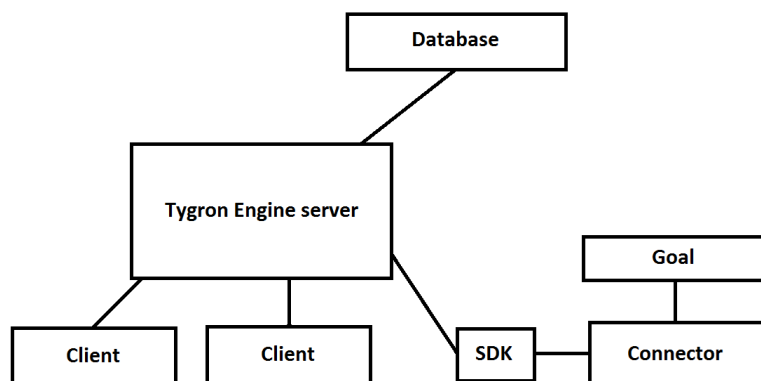


Figure 2: Diagram of the Tygron Engine connected to the database and the GOAL agent.

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. 2 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 2 shows the diagram of the Tygron Engine connected to the database and the GOAL agent.

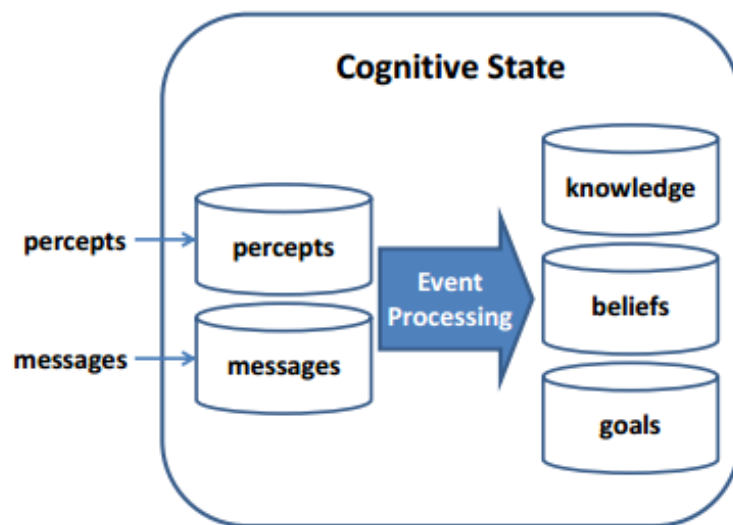


Figure 3: Updating the agent's database.

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's 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 3 (Hindriks, 2016) shows how the databases of the agent are updated.

## **2.4 Concurrency**

Each group in the Virtual Humans for Serious Gaming context project will develop a cognitive agent, so there will be 5 roles and therefore multiple agents. An agent itself does not make use of concurrency, but all the agents together will run parallel and are running in 1 environment. When there are multiple agents running, there is concurrency. A crucial thing to find solutions for city design and development projects in a game is communication between the agents. The agents have to be able to communicate and find a solution together that fulfil the goals of the agents as much as possible.



### **3 Glossary**

## References

Hindriks, K. (2016). *Programming cognitive agents in goal*. Retrieved from <http://ii.tudelft.nl/trac/goal>