Architecture Design

VH-Danshal

1 Introduction

This document provides set of overviews relating the to the architecture of the system. This particular section will cover the design goals. Each view will provide a different high level description of system and its different subsystems.

1.1 Design goals

In order to maintain a high-quality system we have set the following design goals for our code.

1.1.1 Design goal: Performance

Our code should be able to run on most modern consumer pc's. This is possible because of the architecture of our product. The connector should handle all low level functions and the GOAL agent should provide high level artificial intelligence, simulating the decision making of a real human player.

1.1.2 Design goal: Availability

Our team uses Travis for continuous integration in order to make sure our main branch is always a working product. This way a prototype can be shown to our client each week. The client can then review the changes and help us improve on our product and implement additional features.

1.1.3 Design goal: Code quality

Our code should be of high quality so it is easy to modify and maintain by both us and future users. To reach this goal we make use of checkstyle, junit, FindBugs, PMD, Cobertura and Powermockito.

1.1.4 Design goal: Functionality

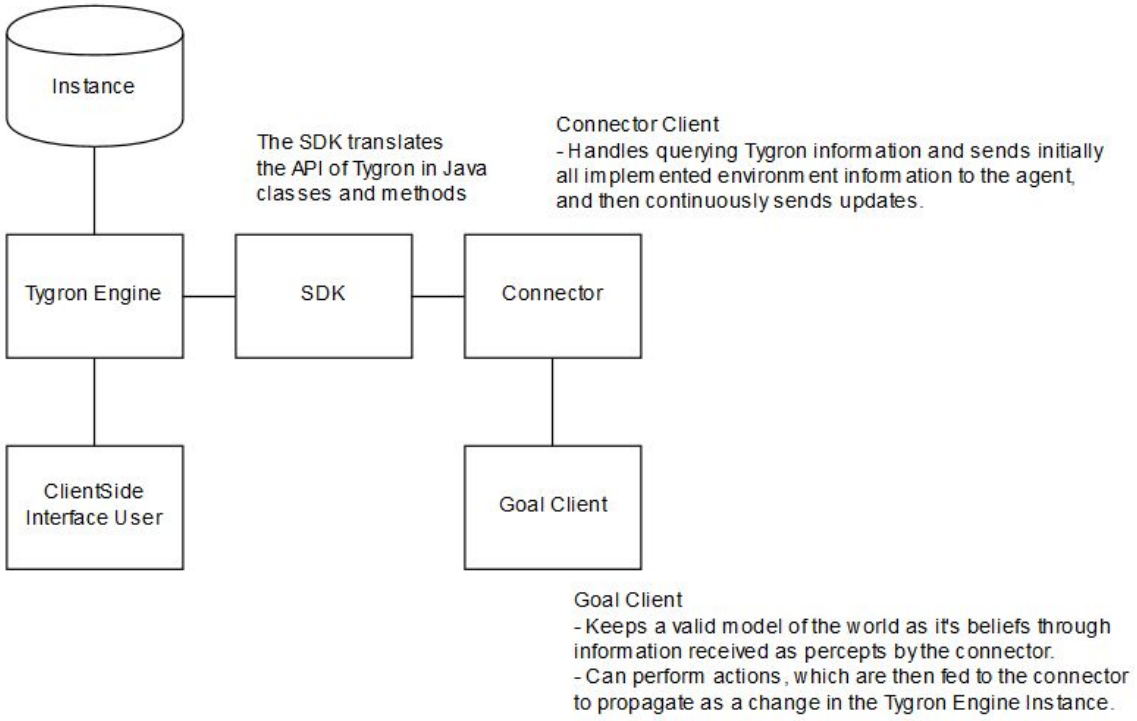
We implement an agent that can fulfill the role of municipality in the TU Delft environment within the Tygron engine. The bot will make decisions through knowledge of the environment and communication with other players and tries to change the environment to improve it in favor of the indicators of the municipality.

2 Software architecture views

2.1 Subsystem decomposition

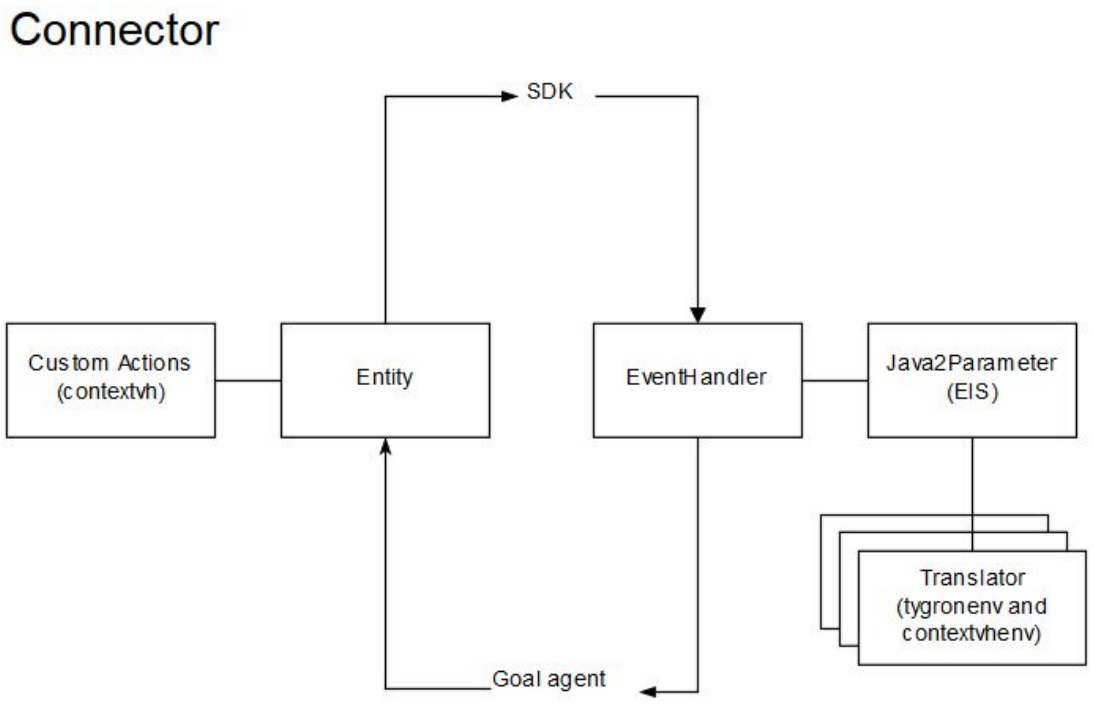
The following diagram reflects the architecture of the Tygron game & engine, as well the way in which users and AI have to connect to the engine. It shows what behaviours should be handled by what systems.

The given assignment states that the Goal Client, the agent, has to be build and connect to the SDK via the connector. The focus therefore lies on extending the current Connector and constructing the agent.



2.1.1 The Connector

The Connector is the link between the Goal agent and the SDK, where the SDK translates the API of the tygron Engine to be used in java through java objects and methods. It consists of the Eis module, the tygron environment and the contextvh environment. It allows translation from Java (used by the SDK) to GOAL (used by the AI) and back.



The Environment Interface Standard (EIS) is an interface on which an environment for a (multi) agent system can be developed. It manages agent pausing and termination, agent-environment interaction and agent-agent interaction.

The tygron environment is a premade module that uses the EIS interface to connect agents with the environment. It does the following. It sends events from the SDK (which received the events from the Tygron engine) to the agent in prolog syntax, and catches actions from the agent and feeds them back to the SDK. It already supports rudimentary functionality for a agent in the Tygron Engine. It also instantiates the connection with the server through the SDK.

The contextvh environment extends from the tygron environment and it is developed to help conceive the product asked for by the client. To make an interesting multi-agent simulation in the Tygron engine, a multitude of new features need to be added. New percepts are added and existing percepts are expanded upon and override the same percept in the tygron environment. Also apart from actions that are fed directly to the SDK, the contextvh also supports adding new actions. Logic within the contextvh are written for them, to make complex actions possible that are not directly implemented in the SDK. An action can be implemented for example where a complicated calculation needs to be performed that instead of letting the agent deduce it, it commands the environment to do the calculation and feed the result back.
To add additional functionality percepts a new entry in the translator folder can be made that extends from the tygron connector translators or implements the EIS parameter protocol. To add an action the custom action interface in the contextvh project can be used.

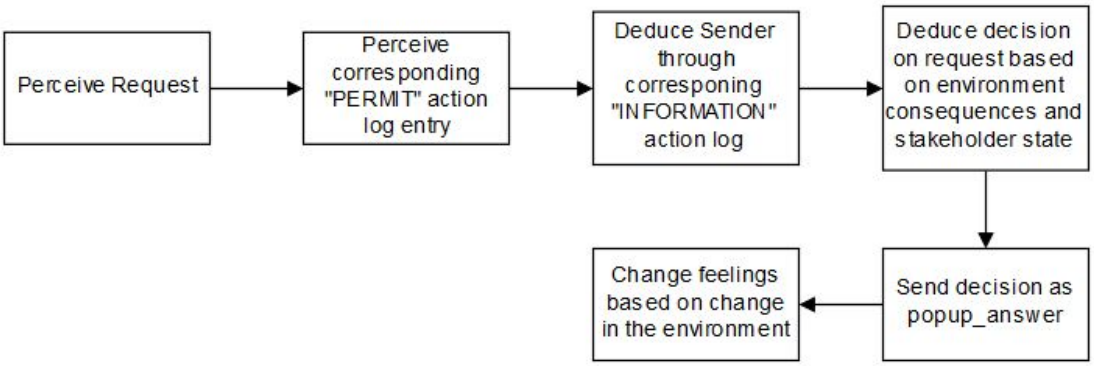
2.1.2 The Agent

The agent represents a stakeholder in the game. It needs to make informed decisions in order to advance its interests. The GOAL programming language is used to implement the stakeholder. GOAL, that currently supports Prolog syntax, has a declarative programming style, which fits agent programming.

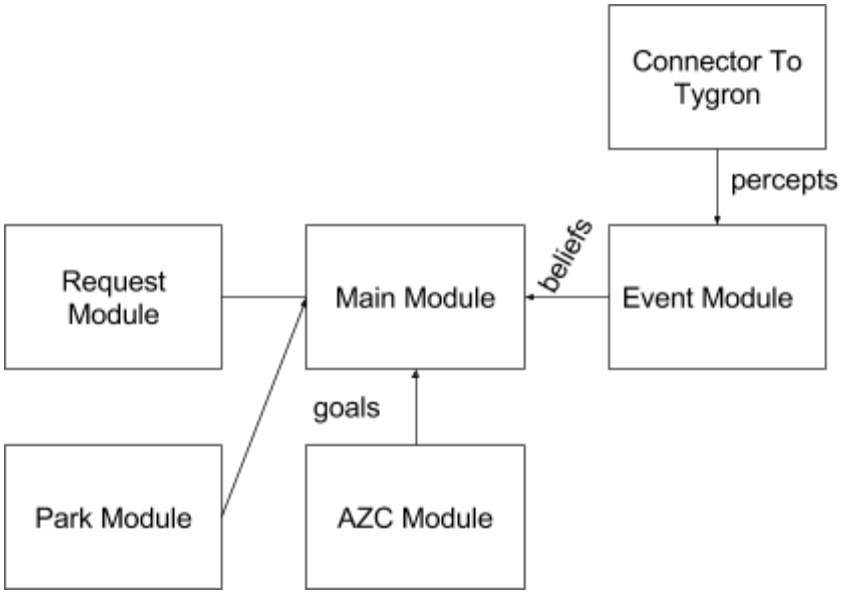
Our task is to implement the Municipality agent. Every agent has the goal of improving the environment to his requirements. These requirements are defined in Tygron engine and are called indicators. These indicators are about how livable the area is or how much profit will be made by a certain stakeholder.

Using indicators as our guide, there are two main tasks. The most important task is handling permits sent by other stakeholders and through strategically deciding on permits improving the environment to our indicators. In Dutch law, any new built structure is illegal if it has not been approved by the government. If a stakeholder wants to build something, the Municipality has to approve. Our agent is mainly focuses on handling these permits. It does this by perceiving a request and answering it with the request answer action.Also there is the task of building an AZC (asylum seeker centre) and the optional task of building parks. In the appendix the percepts and action used by the Municipality agent are documented.

Normally in such city planning processes there is a negotiation between stakeholders. If a stakeholder promises A the Municipality will allow B, where A would be something that helps the Municipality a lot and B has a small disadvantage. But it would be problematic to constrain such actions that such requests need to be received at the same time, which is out of the control of the agent. An option would be to create an external communication platform for agents to do such negotiations but instead of this, the Municipality is implemented as following to account for indirect risk reward scenarios. The municipality keeps state of all other stakeholders and deduces decisions on requests based on if stakeholders did well in the past. This decision making flow is shown in the following image.



Apart from reviewing permits we also have these additional tasks. We also have to reach our own goals by building parks and immigrant housing. This is done by using the get_relevant_area action to get area's that we own and where is room for such structures. We can then build with the building_plan_construction action.



The agent is structured in modules. The event module will handle all percepts and insert that information as beliefs. The park and azc module will evaluate the current situation and if it is deemed possible and necessary to build respective structures the action will be performed. In the module that handles requests called the negotiation module negotiations are implemented.

2.3 Hardware/Software Mapping

The only hardware component necessary for the connector and AI to run is a PC or laptop. The connector uses the SDK provided by Tygron, which is also present on the laptop. Using the SDK, the connector is able to produce a jar file on the laptop. The jar file can then be imported to the bot, which can then run from the PC or laptop as well.

While the agent runs, it does connect to the tygron server. The server and Tygron engine to run on different hardware components when compared to an AI. Communication between the AI and the server is via the Internet, using HTTP. Different agents can be run from the same computers or separate ones. An agent has it's own client instance and is loosely connected to their own connector environment instance, it's loose because they are different programs. These two instances have to be on the same machine. Each agent connector has it's own connection with the Tygron server .

2.4 Persistent data management

Persistent data is data that is stored even after the application is done running.
The agent and the connector both do not have to store persistent information. So there is no need for a database or external files.
We could make use of persistent data. For example if a number of sessions will be run with the agent it could store the available land at the start of the session in persistent memory. Because cities are changing in the matter of decades, it would be fine to not compute such expensive geometric computations every time at initialisation of the agent. This would be a worthwhile addition to increase performance.

2.5 Concurrency

There is no concurrency present within our agent, and because the SDK alleviates the low level server connection the connector does not address concurrency issues. An interesting cause for concurrency is when multiple agents work together in one Tygron instance. The Tygron server has to account for handling the requests in an orderly fashion, but this is far from the scope of our agent program.

3 Glossary

Goal Client & AI
These terms can be used interchangeably and refer to the artificial intelligence we have to build during this project. Our artificial intelligence will play the role of Municipality.

The Game & Session
These terms can be used interchangeably. They refer to the environment in which the AI needs to operate. The environment runs in sessions and represents a planning sessions in which multiple stakeholders participate.

Stakeholder
An entity connecting to the Tygron engine in order to attend a session. They can either be human or an AI.

4 Appendix

These are percepts and actions used by the Municipality agent. A list with additional list can be gotten from the eishub/tygron repository on github.

Percepts
Currently we have these basic percepts

- functions(<X>)
  Returns a list of all functions (creating buildings and structures) our bot can perform.
- settings(<X>)
- buildings(<X>)
  Returns some basic information about all buildings, this is used to percept when a building is successfully built
- stakeholders(<X>)
  Returns all stakeholders
- request(<type>, <type of request>, <Id>, <contentlinkid>, <[visible for this stakeholder]>, <[actionlogId]>, <multipolygon>, <price>,<area>, <[Answer(<AnswerId>, <Description>)]>)
  When a request is made to buy land or for approval of the construction of a building, this Is percepted for agents for which it is relevant. Generally the one who wants the action
- actionlog(<stakeholderid>, <description>, <Id>, <[Indiciator increase]>)
  Every action of a stakeholder that changes the indicators in the environment is perceived In a actionlog.

and the one who needs to agree with it. The Id is connected with the Actionlog percept. Only those requests of type “INTERACTION” and “INTERACTION_WITH_DATE” need to be

answered. Generally the possible answers are a Integer between 0 and 3, where 0 is Yes, 1 is No, 2 is take it for free, 3 is close the window, which is not really relevant for our agent.

Actions

- building_plan_construction(Id, Level, MultiPolygon)
  Builds a simple building with [level] floors
- popup_answer(<Id>, <AnswerId>) and popup_answer_with_date(<Id>, <AnswerId>, Number)
  With this a requests can be answered
- get_relevant_areas(<CallID>, <ActionType>, <Filters>)
  Get area's in the tygron engine that fit given criteria