Emergent Architecture Design

Joe Harrison Harm Griffioen Dereck Bridie
Paul van der Knaap Natasa Radenovic

May 22, 2015

1 Introduction

In this document we will go over the details of how the system is going to be built and how our group will design the software architecture of the connector part of our context project.

2 Design goals

Throughout the project, the contributors will keep certain design goals in mind when creating code for our project.

2.1 Extensibility

Keeping in mind that the Tygron Engine is not final and may be changed by Tygron in the future, we will design our Connector in such a way that features can easily be added, removed or modified when the Tygron Engine changes.

2.2 Performance

The Connector should have low latency, so that agents can ask for environment information and be able to process this information without our Connector being the bottleneck.

2.3 Code quality

The Connector's code will be of high-quality, which means that code will be dynamically and statically tested. To check the quality of our code we will be using various tools which give indications of code quality.

Our code repository is split up into multiple branches. One branch, **master**, is the main branch in which code will be stored that is working. Every change to the code in the master branch must be manually reviewed and approved by a majority of us before being allowed to be pulled to the master branch.

CheckStyle is a development tool that helps programmers write Java code that adheres to a coding standard. This will help us create code that has uniform code styling, keeping into account white space, method length, documentation, and so on.

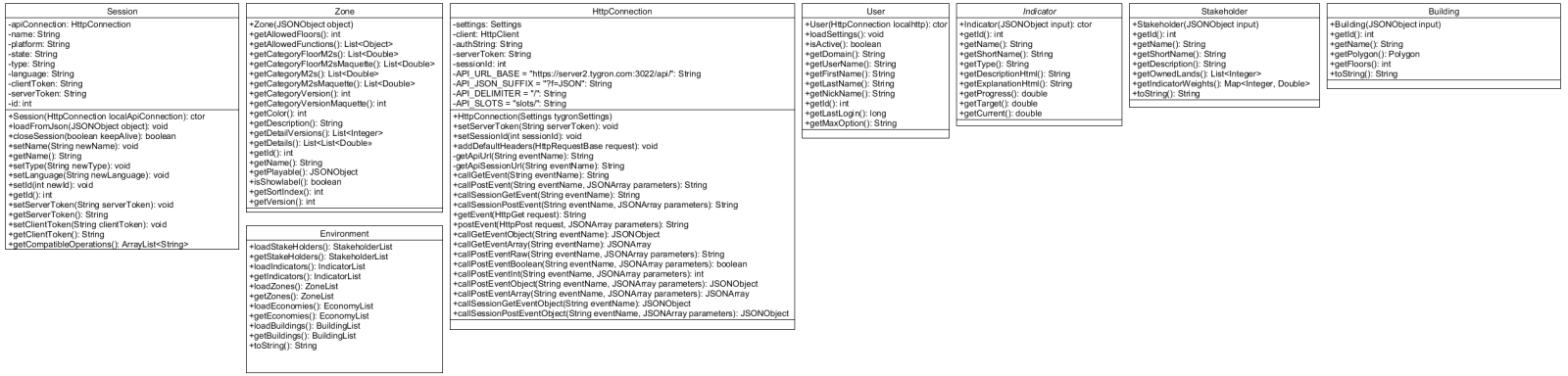
PMD and **FindBugs** are source code analyzers which finds common programming flaws and reports them.

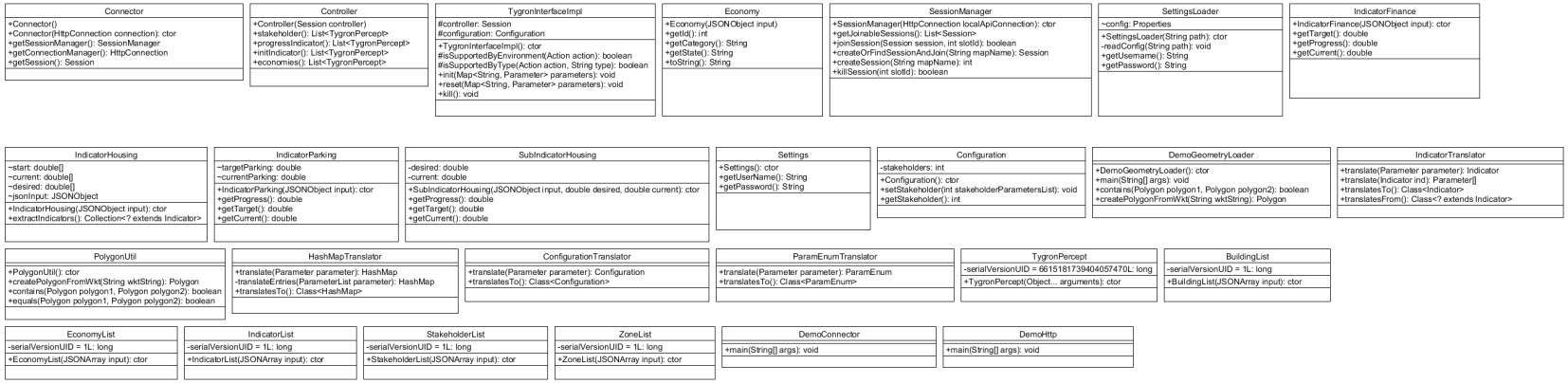
JUnit is a testing framework which allows programmers to write repeatable tests. In conjunction with **Cobertura**, this can be used to check the percentage of code accessed by tests.

Javadoc will be used to generate documentation which will be used to describe code.

All of these tools will be run together using **Travis** and **Jenkins** Continuous Integration. These tools will automatically generate reports based on latest versions of code.

3 Class Diagram



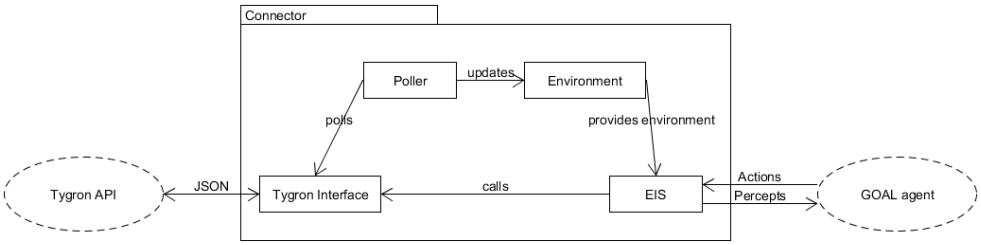


4 Software architecture views

4.1 Subsystem decomposition

This part describes our Connector and how it is divided into subsystems.

Figure 1: Components diagram



4.1.1 Tygron Interface

The Tygron Interface is a subsystem that will communicate with Tygron's REST endpoint. This interface's responsibility is to translate functions to REST requests, and to translate REST responses back to Java objects.

4.1.2 Environment

The Environment is a buffer between the EIS and the Tygron Interface. It's responsibility is to keep a copy of data from the Tygron Interface which can be updated. GOAL agents will be able to access this data via the EIS. The Environment is necessary because otherwise the GOAL agent will be have to wait for the duration of the API request.

4.1.3 EIS

We will also develop an interface according to the Environment Interface Standard. This interface is what provides GOAL agents with percepts which will provide GOAL agents with the required information to perform and exposes functions that the agent can call.

4.1.4 Poller

It's job is to poll our JSON interface with some interval, and the results of the polling is used to update the Environment.

4.2 Hardware/software mapping

Our software will communicate over the internet with Tygron's server. This communication will be layered on HTTP, using the REST protocol.

4.3 Persistent data management

Our connector does not have the responsibility of storing persistent data, so it does not have external files or databases.

4.4 Concurrency

Our Environment is a shared resource. It is updated by one subsystem and read by another, therefore we are not concerned about deadlocks.

5 Glossary

Cache
: A collection of data which duplicates other data

EIS
: Environment Interface Standard, a standard which facilitates connecting agents to environments

GOAL
: Goal based agent programming language, used for programming intelligent agents

JSON
: JavaScript Object Notation, a data representation format

Percept
: A chunk of data that a GOAL agent receives from it's environment

Polling
: Sampling an external service

REST
: Representational State Transfer, a software architecture which is used for creating web services