Emergent Architecture Design

Blocks World for Teams

BW4T Context Project

Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

EMERGENT ARCHITECTURE DESIGN

BLOCKS WORLD FOR TEAMS

by

BW4T Context Project

in partial fulfillment of the requirements for the completion of

TI2805: Context Project
of the Bachelor of Computer Science and Computer Engineering

at the Delft University of Technology,

Architecture Design Team:	Daniel Swaab	4237455
	Martin Jorn Rogalla	4173635
	Jan Giesenberg	4174720
	Sander Liebens	4207750
	Sille Kamoen	1534866
	Shirley de Wit	4249259
	Tom Peeters	4176510

An electronic version of this document is available at https://github.com/MartinRogalla/BW4T/.

TU Delft Delft University of Technology

Contents

1 Introduction **2**
1.1 Design Goals 2

2 Software Architecture Views **3**
2.1 Subsystem Decomposition 3
2.2 Software Mapping 3
2.3 Persistent Data Management 3
2.4 Concurrency 4

3 Glossary **5**

Bibliography **6**

1

INTRODUCTION

1.1. DESIGN GOALS

For this project we were given a large codebase. This codebase had grown complex and convoluted and the task for us was to refactor it to reduce complexity, make it more manageable and give it a Plugin Architecture. On further inspection, however, it was revealed that an exact Plugin Architecture wasn't precisely what the product owner wanted, but simply a modular design that can easily be expanded or modified was desired.
To this end the goal is to have most if not all core parts of the environment and agents built using interfaces, thus allowing easy addition of new elements in the environment or agent behaviour by creating a new class utilising one of these interfaces, similarly these classes could then later be easily removed without causing parts of the system to start malfunctioning or breaking entirely.

For the refactoring of the code a number of tasks need to be completed:

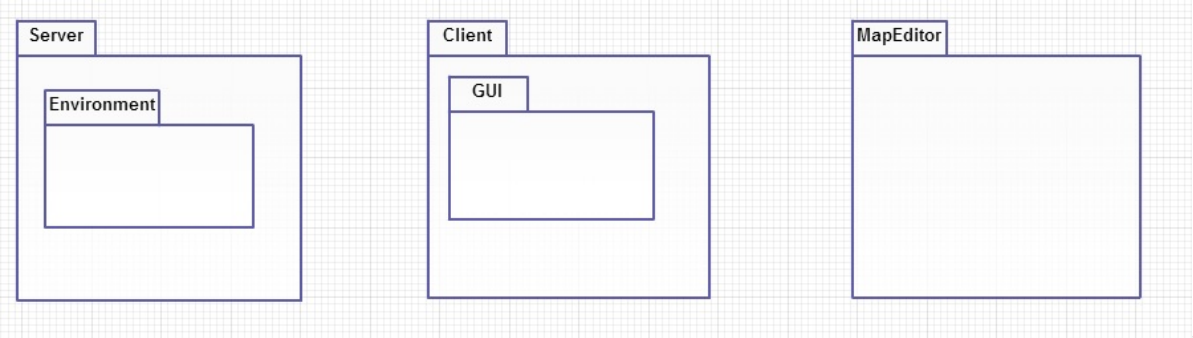
- **Split up codebase into multiple structures** - Currently the Client and Server and GUI all share the same codebase, this makes it hard to keep a proper overview as it isn't always clear at a glance which class belongs to which structure, and some are even shared entirely. We want to split these structures into individual projects to make them easier to work with.
- **Convert to Maven project** - There are currently no Unit Tests present in the codebase, therefore in addition to splitting the codebase we wish to turn these seperate projects into Maven projects including Unit Tests so we can easily verify that the codebases are still working correctly after having made changes.
- **Improve and add to the GUI** - There is much room for improvement for the GUI. We want to make it more user-friendly in addition to adding a whole new GUI for the creation of maps via an easy to use drag-and-drop system.
- **Clean code** - There is a lot of unused code present, due to the complexity of the codebase it can be hard to tell which pieces of code are unused.
- **Fully utilize Repast** - Repast is a library that has been imported to act as the environment for the agents to act in, however due to some missing functionality and unfamiliary with Repast at the initial time of implementation Repast isn't properly utilized, many things that have been implemented manually can also be handled by Repast.
- **Improve Javadoc** - Javadoc's are sometimes lacking in information or occasionally not present at all.
- **Remove code duplication** - There are a number of instances of code duplication present in the code which can make future updates harder to manage.

2

Software Architecture Views

2.1. Subsystem Decomposition

Our systems consist of three parts: the server, the client and the mapeditor. Within this three subsystems, other subsytems can be defined. These subsystems will become more clear over time.



2.2. Software Mapping

2.3. Persistent Data Management

In the Blocks World for Teams project there is no need to use databases.
The only persistent data are log files and Environment files.

At this moment, the log files are raw dumps of the actions a bot (might be human, might be agent). These actions vary from walking to a spot in the map to picking up an item.
There is not much to do with these log files. Finding something the bot has done is a heavy task as you first need to restructure the file and seek through all the lines.
In the future, these log files might be used to reconstruct events of a bot. Then you could launch the log file into the client, which will run the bot according to the lines in the log file.
This could become very useful if you want to analyse why a bot made some decision, or why it did NOT do what you expected it would do.

The map files are big XML files, which consist of a decleration of the map.
The XML file needs to contain at least the following:

- The area of the map (in x and y value).
- The bot entities (including a starting position in x and y).

- The zones (including the name of the zone, the colour of the boxes inside, its neighbouring zones and an x and y value).
- But also the goal sequence (containing six colours).
- Whether this sequence has to be random (boolean).
- Whether multiple bots are allowed in one room (boolean).
- And whether a random number of blocks is in the rooms (boolean).

These XML files have to be very precise for the server to set up an environment. The MapEditor, we would like to construct, is going to create such XML files with only drag and drop actions, so that maps become more realistic then just rooms in a row. We could have a centered room, which is surrounded by others, or one room bigger than the other.

2.4. CONCURRENCY

As stated in section ..., the BW4T-Server needs to accept connections from various BW4T-Clients. The following decisions had to made regarding client priorities:

- Every connected client is allowed to start the debugging process and/or control it.
- The agent's actions, percepts, etc. which are time dependent are offered by the server in a first-come first-serve basis.
- The agents should be indistinguishable and should not suffer from any effects due to being from a different client.

Concurrency will mostly be handled by a tick system. Every time a tick is executed, an agent performs a certain action which is allowed by the server within that specific time-frame. If a non-shareable atomic action is requested by an agent, the server will look up in the configuration what it is supposed to do. The default behaviour in this situation will be that the server picks one of the agents randomly and allows them to perform the action.

The key here is that each action is assigned a specific timing(number of ticks). In the previous versions of BW4T, it would still be possible for a client to complete an action even though the debugger had halted the game. By splitting up every action in separate ticks, the world should be much more predictable and should enhance debugging simplicity.

3

GLOSSARY

BIBLIOGRAPHY