Emergent Architecture Design

Blocks World for Teams

BW4T Context Project

Delft University of Technology

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
	Valentine Mairet	4141784
	Xander Zonneveld	1509608
	Ruben Starmans	4141792
	Calvin Wong Loi Sing	4076699
	Martin Jorn Rogalla	4173635
	Jan Giesenberg	4174720
	Wendy Bolier	4133633
	Joost Rotheweiler	4246551
	Joop Aué	4139534
	Katia Asmoredjo	4091760
	Sander Liebens	4207750
	Sille Kamoen	1534866
	Shirley de Wit	4249259
	Tom Peeters	4176510
	Tim van Rossum	4246306
	Arun Malhoe	4148703
	Seu Man To	4064976
	Nick Feddes	4229770

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

TU Delft Delft University of Technology

CONTENTS

Abstract

1 INTRODUCTION

In this chapter, we will explain our goals for this project. We split them up into several tasks that work towards achieving our design goals, which include but are not limited to: reliability, modifiability and ease of use.

1.1. DESIGN GOALS

For this project we were given a large codebase. This codebase had grown complex and convoluted. The task for us was to refactor the code 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. By analyzing the product owner's final goals, we decided that a modular design which can be easily expanded or modified was actually desired.
Our goal is to have most, if not all core parts of the environment and agents built using interfaces. This allows for easy addition and modification of elements in the environment or agent behaviour. Creating these additional features would then be done by creating new classes utilizing one of these set up 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 Client and Server** - Currently the client and server 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 classes 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 separate 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 unfamiliarity 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.

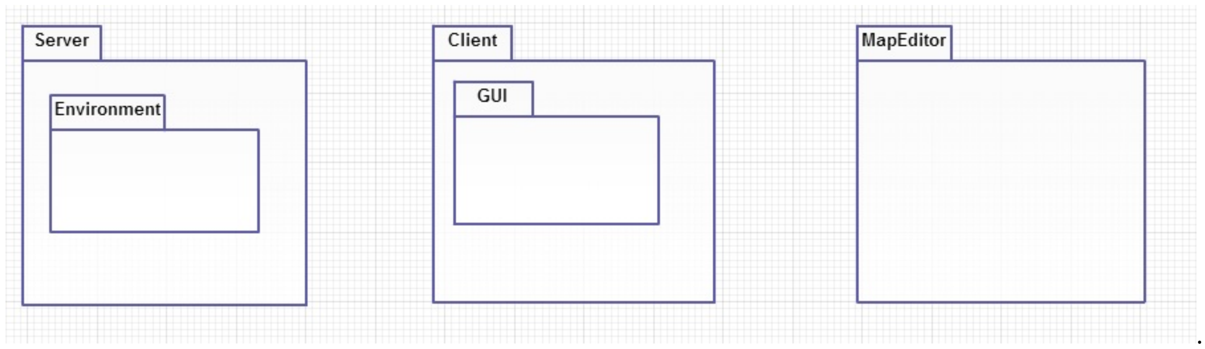
2

Software Architecture Views

In this chapter we will explain our software architecture views. First we will talk about the subsystem decompostion. After that we will talk about the software and hardware mapping of this subsytems. Then we will talk about persistent data management. The last subject is concurrency.

2.1. Subsystem Decomposition

Our systems consist of three parts: the server, client and map editor. Within these three subsystems, other subsytems can be defined. These subsystems will become clearer over time.



2.2. Hardware/software mapping

In this project there is no need to discuss hardware. While we do have a Client-Server relation, we run it on a single computer, so for all practical purposes there is no need for different hardware. It is, however, possible for clients to connect to a remote environment.

The software architecture is shown in chapter 2.1. We are planning on refactoring the code to this structure first, because there are currently many illogical connections between client and server.

2.3. Persistent Data Management

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

At this moment, the log files are raw dumps of the actions of a bot (might be human, might be agent). These actions vary from walking to a spot in the map to interacting with an item.
Currently, there is no use for these log files besides debugging, but the log files are too poorly organised and contain insufficient information to be of significant value for this.
In the future, these log files might be used to reconstruct a series of actions or events. You would be able to feed a log file to 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 certain decisions, or why it did not do what you expected it to do.

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

- The size of the map (an x and y value)
- The bot entities (including an (x,y) starting position)
- The zones (including the name of the zone, the colour of the boxes inside, its neighbouring zones and an (x,y) value)
- The sequence of coloured blocks to be collected (or alternatively set to be random via a seperate boolean)
- Whether multiple bots are allowed in one room (boolean)
- Whether the number of blocks in the rooms are randomized or not (boolean)

Because these XML files can be rather difficult to write by hand, there is currently a tool which creates a map for you according to certain parameters. This tool will however always generate the maps in a grid formation, with all rooms being the same size, which results in the maps always looking rather similar.
The map editor which we would like to construct is going to create such XML files with only drag and drop actions, so that maps become more realistic as any shape is possible, not just grid formations.

2.4. CONCURRENCY

As stated in section 2.2, the BW4T-Server needs to accept connections from various BW4T-Clients. The following decisions have been made regarding client priorities:

- Every connected client is allowed to start the debugging process and/or control it.
- The agents' 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 is that the server will select each agent in a random order and allow them to perform the requested action.

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

Glossary

Concurrency Concurrency is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other. 5

Debugging The act of finding and correcting errors in a program. 5

GUI A graphical user interface, allowing the user of a system to manipulate the data within the system with ease. 3

Javadoc Documentation of the code. 3

Log Files Files that are generated by a system to allow the user/programmer to find errors or check to flow of the program. 5

Map Editor A map editor allows the user to generate an environment for the system. 5

Maven Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. 3

Plugin Architecture A plugin architecture is a part of a larger system that allows to easyly change, adapt or implement new features, possibly even at runtime. 3

Unit Tests These are test run on small subsets of the software usually at the class level. 3

XML Files Extensible Markup Language, a well-known language used to represent data structures. 6