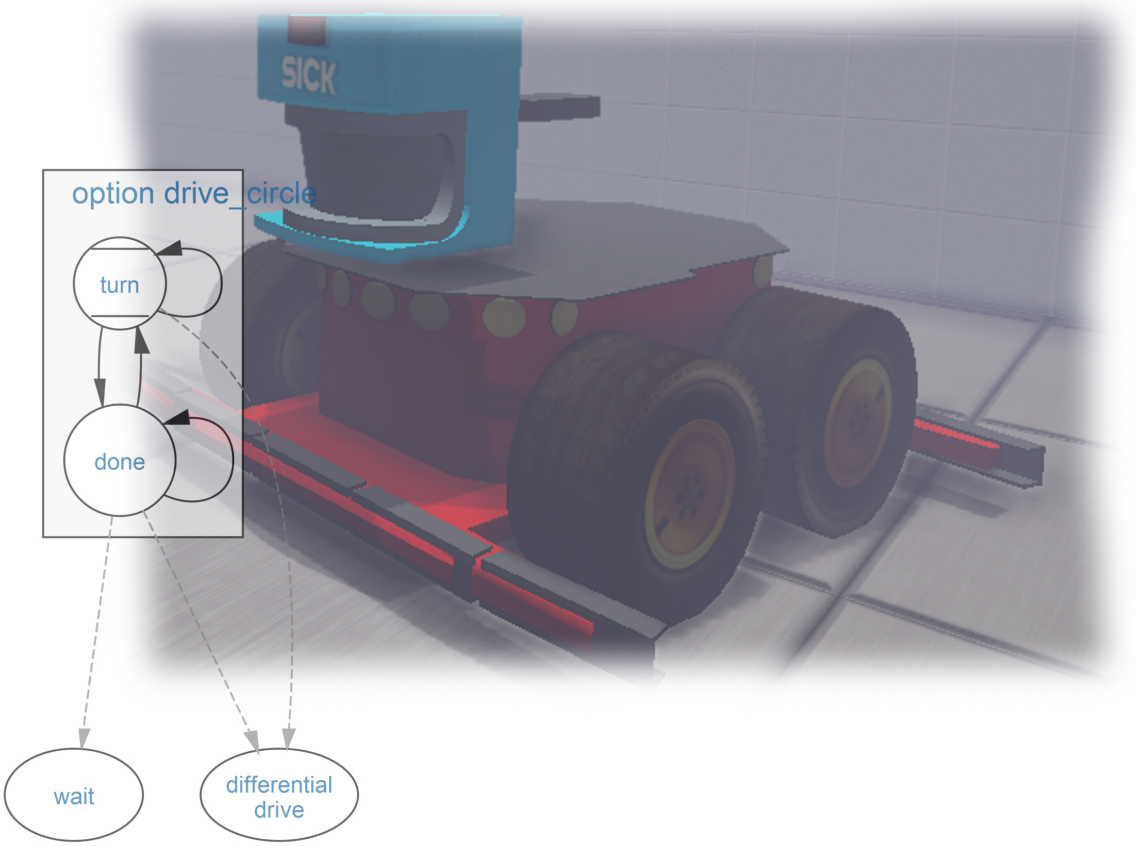
Combining Robocup Rescue and XABSL

Maarten P. de Waard



UNIVERSITEIT VAN AMSTERDAM

Bachelor project: Final Thesis

Combining Robocup Rescue and XABSL

Maarten P. de Waard
5894883

Bachelor thesis
Credits: 6 EC

Bsc. Artificial Intelligence

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
dr. A. Visser

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 26th, 2012

Abstract

In this research, a product will be introduced, that combines the Extensible Agent Behavior Specification Language (XABSL) with any program, capable of having a socket connection. A use of this product is shown, by combining it to the rescue project on the University of Amsterdam, using *UsarCommander*, a program designed to control one or more robots, in a virtual rescue operation.

1 Introduction

The research will be focussing on combining the *Extensible Agent Behavior Specification Language* (XABSL) with any program capable of making a socket connection. In particular, the focus will lay on virtual rescue operations, otherwise known as the *RoboCup Rescue League*. Using a behavior specification language will make it possible to separate specification of behaviors from implementation.

Currently, the focus of research in the rescue missions is mainly on creating smart implementations of sensors. Most of the actual controlling of robots is done by hand, using programs that forward the camera images of the robots to a human operator controlling them. Some of these operators use simple behaviors to help them, like for example making the robot automatically traverse a path to a specified point. This kind of simple task can be called a behavior.

An improvement that can be made in these behavior controlled robots, is in the specification of which behavior should be selected in a certain situation. and how the behavior is executed. This can be done by creating behavior-controlled robots, that can autonomously select the best behavior to activate on a certain moment, and using their sensors as input, can choose the right way to navigate.

This research will make use of XABSL, a behavior specification language, which makes it possible to separate the specification of a behavior, from the implementation.

Currently, not many behavior-controlled exploration algorithms exist. An exception is path finding on challenging terrain [3] . This research will result in a method to easily adjust and improve the behavior of any robot in any robot commanding program, especially focussing on UsarCommander, the program used by the UvA Rescue team[1].

2 XABSL

XABSL is a programming language, created to easily describe behaviors for autonomous agents based on hierarchical finite state machines [2]. It is the software that has been used by the German robotic soccer team to specify their robots' behaviors. The team won in 2008, and the years after that.

[1] Teamdescriptionandmoreat:http://www.jointrescueforces.eu/wiki/tiki-index.php

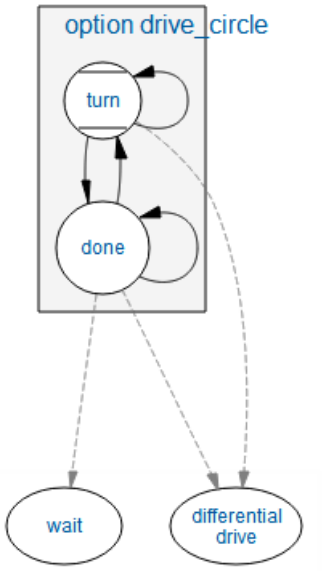


Figure 1: An example of a figure generated by the XABSL compiler, from XABSL code.

2.1 Difference with other specification possibilities

There are many alternatives to XABSL, when it comes to defining a robot's behavior. This section will discuss some of the alternatives, that had to be considered before choosing XABSL.

2.1.1 POSH

POSH [1] is a very similar alternative implementation of a Behavior Specification Language. Posh is defined as a *Behavior Oriented Design*, which is created from a combination of *Object Oriented Design* (OOD, used by object oriented programming languages like Java and C++) and *Behavior Based Artificial Intelligence* (BBAI). The BBAI-part of it is the decomposition of intelligence as subtasks called *acts*. Examples of acts are knowing your position and planning a route. From OOD, BOD takes building your behavior in an object-hierarchy and an agile and iterative development process.

POSH is an abbreviation of Parallel-rooted, Ordered, Slip-stack Hierarchical, which, freely interpreted, stands for something that enables a user

to specify an agent's goals and priorities. This is actually quite similar to XABSL, because it makes certain decisions, and then selects an action to be executed by an external program. There are some important differences though:

- POSH is designed to be used by non-programmers. This means the interface is easy, colorful and simple, whereas XABSL prioritizes complex capabilities, making the specification somewhat more complex for non-programmers.
- Where XABSL has a close coupling with the perception stream of the robot, POSH has no variable management, enabling the system to be a lot easier to use, but also maximizing the complexity of the specified behaviors to a lower maximum than XABSL offers.

2.1.2 The next thing

I don't know yet, let's see tomorrow

2.2 XABSL specification

XABSL makes use of four components: Agents, Options, states and Basic behavior. The following subsections will explain what these are, and how they can be used.

2.3 Agents

An XABSL agent is a rooted acyclic graph, containing all the behaviors for one agent. In this research, one agent will equal one robot.

- *Agents:* A rooted acyclic graph, containing all the behaviors for one agent. Several of these agents can be created, all having their own graph and thus their own behavior.
- *Options:* Complex agent behavior. Each option is on itself a finite state machine, containing several states.
- *States:* 'Actions' that can be either active or inactive. At least one state is always active. Each option has an initial state, being the first state to be active. States are connected with each other by decision trees, deciding what state or option is activated next.
- *Basic behavior:* At every leaf of an option (so, every state with no other states to reference to) a basic behavior is activated. This is a small piece of C++ code, that influences actions of the agent.

2.3.1 Motivation to use it

Using FSM's for behavior is an easily comprehensible method to specify behavior. The advantages of XABSL are that tools are delivered to make a hierarchy documentation for your website (or anything else). For example, the FSM in figure 2 is automatically generated from XABSL code.

Using this representation, tweaking the behavior should become an easier task resulting in better results for autonomous exploration.

This section will be expanded with the following:

3 RoboCup Rescue

3.1 Description

The project used in cooperation with the application, is UsarCommander. UsarCommander[2], originally developed by Bayu Slamet, and extended by Arnoud Visser and many others. This program takes care of connecting to USARSim (the simulator used in the Robocup) and makes the user able to easily get sensor data from the robots in it. It is also possible to control the robots with several types of behavior, like corridor-following, obstacle-avoidance, or tele-operation. The last of which enables the operator to manually control the robots by hand, using an interactive human interface.

Over time, the system has been expanded with many subprojects, for example one implementing a SLAM (Simultaneous Localisation And Mapping) algorithm, to make an accurate map from the sensor data of several robots [4]. All the information used and produced by these subprojects can be accessed by other subprojects, resulting in an ideal environment for creating new robot-controlling applications.

3.2 Motivation to use it

The main reason to use this program, instead of any other, to interface my software with USARSim is that it has so many features. The presence of many subprojects in the code, makes it possible to make a very efficient autonomous exploration algorithm interfacing with the subprojects at hand. Without using UsarCommander all the needed software should be taken from somewhere else, or implemented solely for this purpose.

Other software for this purpose is available too.

but since this is a bachelor thesis on the University of Amsterdam, and this is the software used by it in the RoboCup, this is the logical choice.

[2] Available at http://www.jointrescueforces.eu/wiki/tiki-index.php

4 Approach

4.1 Interfacing both programs

Since the UsarCommander is written in Visual Basic, and the basic behaviors of XABSL are written in C++, a bridge should be made. This is done by creating a Dynamic Link Library (DLL). This DLL contains the needed functions of the C++ program, making them accessible for Visual Basic. The bridge works both ways, so Visual Basic can offer output symbols to the XABSL Engine, while the engine can offer input symbols to the agent.

4.2 Creating a succesful hierarchy

This section will tell about the FSM hierarchy I will make for autonomous exploration

5 Results

This section will contain results, hopefully in the form of explored maps, numbers of victims found, etc.

6 Conclusion

References

[1] C. Brom, J. Gemrot, M. Bıda, O. Burkert, S.J. Partington, and J.J. Bryson. Posh tools for game agent development by students and non-programmers. In *The Nineth International Computer Games Conference: AI, Mobile, Educational and Serious Games*, pages 126–133, 2006.

[2] M. Lötzsch. Xabsl website, 2003.

[3] H. Seraji and A. Howard. Behavior-based robot navigation on challenging terrain: A fuzzy logic approach. *Robotics and Automation, IEEE Transactions on*, 18(3):308–321, 2002.

[4] B. Slamet and M. Pfingsthorn. Manifold-slam: a multi-agent simultaneous localization and mapping system for the robocup rescue virtual robots competition. *Master in Artificial Intelligence at the Universiteit van Amsterdam*, 11, 2006.