Aibo Project 2004 - German Team Report

Brammert Ottens bottens@science.uva.nl
Aron Abbo aabbo@science.uva.nl
Peter Johan van der Meer pjmeer@science.uva.nl
Manfred Stienstra mstnstra@science.uva.nl

January 29, 2004

University of Amsterdam

Abstract

This document describes an evaluation of the German Team software framework which was built for the Four Legged League, RoboCup 2003. It also includes a comparison to the Carnegie Mellon University framework.

1 Introduction

For us, playing soccer is something we learned as a kid. Some became better at it than others, but we can all follow the ball, predict where it is going and kick it in the right direction. We can even play a match against each other.

Since 1997 the academic community took up the task to create robots with the same soccer capabilities as us humans. Every year the RoboCup games are held, so researchers can test their teams in a real environment. One of the leagues in The RoboCup is the Sony four legged league. In this league, teams of four Aibo robots play soccer against each other, and since a couple of years it is customary for the competitors to release their code after the games. This way people can use the work done by others and learn from each other.

The Dutch have competed in some of the other leagues before, but a Dutch team has yet to enter the four legged league. This is about to change, and in order to avoid unnecessary work, the Dutch team wants to adopt an architecture from another team to use as a starting point.

Our task is to analyse the architecture used by the German Team 2003[5]. This analysis will deal with the modularity of the architecture, the functionality, the compatibility, the ease of understanding, the adaptability, the utilities and the documentation.

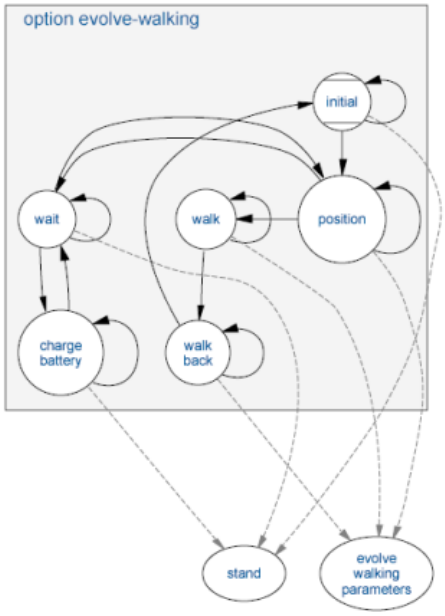

Figure 1: Evolve walking agent, a state machine example

After this we have to compare our findings with the findings of the other group, which analysed the CMU team framework. In other words, we have to say which one is better suited as a starting point for the Dutch team. And last but not least, we will also point out some parts of the software that can be improved.

We have decided to concentrate on the architecture of the framework and the tools. Information about installation details, modules and algorithms can be found in our technical document[7] and in the German Team technical document[1].

2 Architecture

2.1 Functional architecture

The main goal of the German Team is creating a team of aibos that can play soccer and solve challenges created by the RoboCup organisation. In order to achieve these goals the robot can use one of 10 different behaviours. More information on the different behaviours can be found in the documentation of the German Team which can be generated from the source code in Visual Studio. The basic idea behind all the behaviours is that they are modelled as finite state machines, see figure 1. In each state the robot will perform one of the basic behaviours. These basic behaviours are for example a behaviour

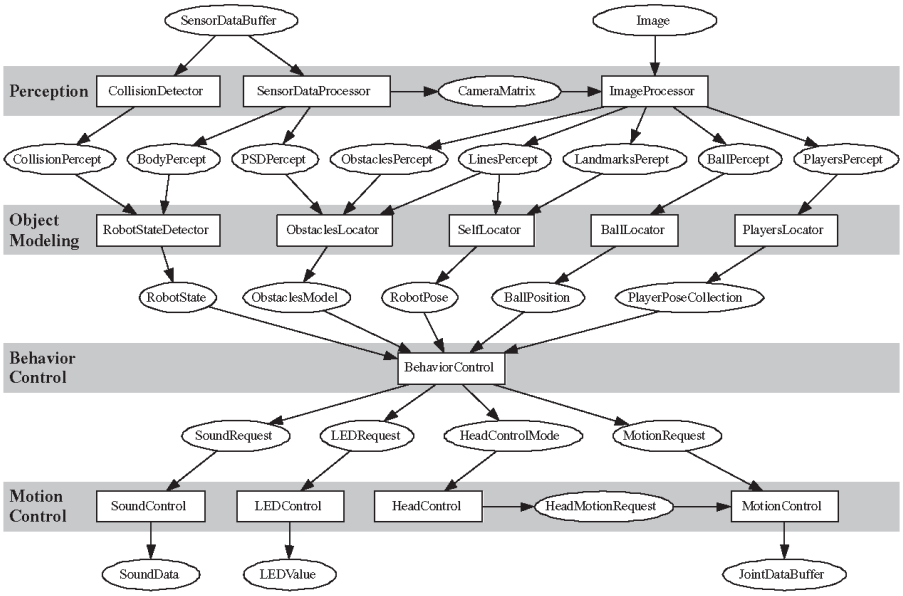

Figure 2: The German Team architecture

that lets the robot walk to a certain point on the field without hitting obstacles, kick a ball or do nothing. Again more information on the different basic behaviours can be found in the generated documentation.

For the soccer playing agent, there are four different sub-agents: goalie, striker, defensive supporter and offensive supporter. The role of the goalie is assigned once each game, but the rest of the robots can change roles during the game. The robot that can reach the ball the fastest will be the striker. The other robots will be supporters. Every robot sends the information necessary to perform the role assignment over the WLAN to all the other robots every cycle. If, for some reason, the WLAN is down all three robots will become striker.

The information send by the robot to the other robots also contains its own location. Knowledge of other robots' positions is useful for avoiding collisions and for tactical planning. Each robot determines it's own location in the world. This is the most accurate. If a player cannot receive the position of another player he will use his vision. If a robot cannot see the ball for some time, the (estimated) ball position will also be communicated. The level of coöperation is similar to that of the CMU team.

2.2 Operational Architecture

The German Team has chosen a hierarchical architecture, as can be seen in figure 2.

All the decisions are made by the module behaviour control. Each rectangle in the picture has it's own classes in the implementation. The ovals are packages of data that can be sent to other modules. Whether the modules are truly independent of each other, will be discussed below.

2.3 Implementation Architecture

2.3.1 Modules

A process consists of a grouping of modules. Modules are implemented to perform different tasks and have well defined interfaces. Typical tasks for modules are reading the data from sensors, extracting features from the sensor data, taking decision based on these features and making the robot act upon decisions. Because the interfaces are well defined it's easy to replace a module, which allows a convenient way of testing several solutions for a task. The code in general is highly modular, functions are well separated or grouped into an appropriate class. This also allows a good maintainability of the code.

Most of the functionality of the robot can be found in the modules, however the robot behaviour is described in another way. The Extensible Agent Behaviour Specification Language XABSL[4] is an XML based behaviour description language that is executed by the runtime system XabslEngine on the Aibo. The engine parses and executes the intermediate code that was generated from XABSL documents. XABSL uses hierarchies of behaviour modules, called options. These contain state machines for decision making. A rooted directed acyclic graph, also called the option graph is used to activate and parameterise one of the basic behaviours, which is then executed. The terminal nodes of the graph are called basic behaviours. Beginning from the root option, each active option has to activate and parameterise another option on a lower level in the graph or a basic behaviour. The symbols from the XML specification are used to describe the options and states to the variables and functions of the agent platform. While options and their states are represented in XML, the basic behaviours are written in C++. XABSL allows for creation of complex behaviour patterns in a declarative manner which are easy to extend. It's also possible to integrate behaviour control with learning facilities based on parameters.

2.3.2 Main Processes

The program consists of two processes that run concurrently, the motion process and the cognition process. The communication between these two processes uses Aperios message queues. The German team has written receiver and sender classes, and a process can be both a sender and a receiver. The information that is send is wrapped into packages.

It is rather easy to implement another process structure. The Germans just recently started using this process layout. How you can change the process layout is described in the German Team technical report[1].

Communication There is one package for the communication from the Cognition process to the Motion process, and one package for the communication from the Motion process to the Cognition process. All these packages are streams from one process to another, so they can easily be adapted or replaced if other information has to be sent. There is a config file that describes communication links between the processes. This is just plain text, so it is easily changed, and is not dependent on the Aibo used.

For the communication between the operating system and the processes the German team also uses the sender/receiver method. The cognition process receives sensor information from the operating system in two different packages. One for the image taken by the camera and one for the rest of the sensors. The Motion process sends the joint angles to the operating system through a special sender. In this sender all the names of the joints are defined, so if the names of the joints change with a new Aibo, this sender is the only part of the code that has to be adjusted. If there is a change in the number of sensors, a change also has to be made in one other class. Of course the implementation of the motion has to be changed too, but that is inherent to a change in joints and is not caused by the used architecture.

Synchronisation is achieved by a blocking mechanism. The Cognition process awaits the sensor data. It only jumps into main when it has received all the sensor data. The motion process awaits a request by the operating system for new joint angles. For the between process information there is no blocking. The processes don't await new information, but work with the old until they receive new information. This new information is received at the start of the main process. All the processes will always work with the most recent information because of this construction.

The Function of the Processes The cognition process will first analyse the sensor data and decide what kind of behaviour is appropriate, after this the Motion process will convert this into a motion the robot can perform. The performance of a module in one process is not dependent on the implementation of a module in the other process. As long as the interface stays the same, nothing bad will happen if one of the modules is changed drastically. No functions from one module in one process are called by another module in the other process.

The cognition process consists of the perception layer, the object modelling layer and the behaviour control. The Motion process consists of the motion control layer.

There are two more processes, the debugger and the logger, but these

processes are only for development and are not put into the release version. The communication also uses the sender/receiver method. We will discuss the debugging in more detail later on.

Remarks If you look at the code it is easy to understand what is happening. The processes just load some modules, execute these modules and send the data. All the real work is done in the modules themselves. The only problem that can be foreseen is that it might be possible for the communication between the processes to become a bottleneck when more data is to be send between the processes. A shared memory structure might then be faster, despite the locking procedures that than have to be implemented. Because the processing power could increase in future Aibos the streams will probably become the bottleneck rather than the overhead from the locking mechanisms.

3 Platform Interface

In this section we will give a short description of the hardware and operating system requirements of the German Team soccer code.

3.1 Operating System Independence

There is a very rigid and explicitly stated combination of OS and software that is needed to compile. These requirements are:

- Microsoft Windows 2000/XP
- Microsoft Visual C++ 6.0 SP5
- Cygwin 1.3.22: A Unix-emulator for windows.
- CygIPC 1.13-2 : Provides Inter-Process Communications services for Cygwin.
- GTK+ 1.3: A multi-platform toolkit for creating graphical user interfaces.

Compiling modules for the Aibo means we have to use a cross compiler, and Sony provides a cross compiler only for Cygwin. The German team decided at some point to use Windows specific libraries for the tools, so they had to use a Windows IDE (in this case Visual Studio). In order to still use a single environment to build the system, they had to create a number of makefiles and scripts. These scripts prove to be a little shaky. Right now, only the simulator Robot Control (which will be described under Tools) is compiled by Visual Studio; the other segments of the code are sent

to Cygwin for compilation. For this part of the code, Visual Studio can also be bypassed altogether by running the build scripts directly.

The required programs and program versions are, as said before, quite rigid. A different version of one of the needed software components could result in hard-to-fix compilation errors and crashes during runtime.

Another factor that complicates the platform independence is Depend. This is a speed optimised preprocessor, written in C, which can calculate the dependencies in a certain setup of modules. Since the memory on the Aibo is quite limited, it is critical to be able to determine which source files are needed and which are not.

Several assumptions are made about the OS and the file system layout that are not documented: In fact, there is little or no documentation available for Depend at all.

3.2 Robot and Simulation Independence

One of the main goals of the architecture of the German Team was to maintain platform independence between robots and simulations. They were not only motivated to do this because a simulation can greatly speed up testing and debugging: Other reasons were that it also improved modularity, and before 2003 there team also had to take into account that there was a non-disclosure agreement in effect for the Open-R SDK. Thus, they had to keep the modules that communicated with Open-R strictly separated from the other ones.

4 Tools

4.1 Requirements

If you want to develop for a certain platform, the right tools are essential. There are a few tools which would really speed up the development, namely a debugger, a profiler and a tool to inspect higher level behaviour. Because the Aibo has limited output functionality we probably want to do this remotely. If some task proves to be cumbersome, you might want to automate this. We also would like to be able to replay entire games and, because uploading code to the Aibo is time consuming, running modules on our PC in a simulator. The final requirement is the need for configuration assistance; the number of parameters in the Aibo can be extensive and assistance in the form of a tool can be very helpful.

The German Team has written a number of tools for these purposes; RobotControl, SimGT2003, Router and the Motion Net Code Generator[1]. We will now look at how they perform these tasks.

4.2 Interaction

In order to receive information about the state of modules, we need a process to handle the communication about the debug information. The framework has two types of requests, a *debug request* and a *solution request*. The requests are issued by software on the PC and routed to the correct module by the debug process, and the data received from the module is routed back to the PC. Debug requests turn debug keys on and off, solution requests turn entire modules on and off. If a debug key is turned on, a module will write associated values to the message queue in the debug process.

4.3 Debugging

RobotControl allows us to look at the output the modules give when certain debug keys are activated. The advantage of this method is that it allows us to look at all the values in the code, even complete images. The disadvantage is that we can't monitor arbitrary values, so we need to create a debug key for each value we want to look at and program it into the debugging framework. In case of a crash we have to fall back on a Perl script provided by Sony for evaluating stack dumps.

4.4 Profiling

RobotControl also enables us to profile the various modules; this is done by a slightly modified version of the debug key solution. The time between two C++ Macro calls is sent to RobotControl by the process which starts the module. RobotControl allows us to view these timings and it can calculate the mean of a arbitrary number of measurements. The disadvantage in this method is again the lack of flexibility.

4.5 Simulating

Both RobotControl and SimGT2003 use an inverse kinematics simulator to visualise a game with Aibos. SimGT2003 is primarily a simulator and RobotControl is more an observation tool. SimGT2003 can be used to simulate entire games, RobotControl can be used to playback log files from the Aibo. Both programs function pretty well. SimGT2003 has a very important shortcoming, it doesn't do collision detection. This means it can't simulate kicks and Aibos can move freely trough each other. The level of detail in SimGT2003 is very high, it can even emulate the buttons on the Aibo. This makes porting the simulator to a new Aibo more difficult because the button actions have to be reprogrammed.

The user interface for both tools is not always clear, for example in a particular dialog box a long list of buttons is presented without dividing

them in functional groups. In another dialog a drop-down menu is shown without the parameter you're changing next to it.

4.6 Automated Processes

Because the behaviour control uses XML files that need to know about the movements, the movements have to be in XML too. The movement control works by calling methods on the movement module. In order to work efficiënẗly with these two representations the German Team has written a tool called the *Motion Net Code Generator*. This tool converts a common joint angle representation to C++ source files and XML descriptions, which can be used in the various components.

The scenarios in the SimGT2003 simulator can be scripted in the console command language, this allows for automated tests.

4.7 Configuration and Calibration

Most of the configuration is done with plain text configuration files, some of the files can be created and maintained with the RobotControl tool. You can save folders with configuration files in the Build directory and copy them with a copy tool to the memory stick. This allows you to maintain multiple configuration profiles and upload them to an Aibo. The copy script doesn't look very robust and in some cases didn't work for us.

Calibration is done by creating a colour table for the Aibo, this colour table has to be in YUV space. RobotControl provides two tools to create this colour table. One in HSI, optimised for fast creation of the colour table, and on in YUV, optimised for precision.

5 Documentation

The documentation available for the German Team framework consists of a technical document, several papers and API documentation. The technical document is a complete overview of the framework.

In Visual Studio it's possible to generate an HTML API reference manual by making use of Doxygen. This generated manual also allows visualisation of the relations between the various elements by means of *include dependency graphs*, *inheritance diagrams* and *collaboration diagrams*. Comments in the code are formatted for parsing by Doxygen and since the documentation is extracted directly from the source code, only one source of documentation has to be maintained.

The technical report gives an overview of all the modules, the tools and gives in depth explanation about the behaviour XML, installation, sender/receiver functionality and the streams library. The documentation

for SimGT2003 and RobotControl is more a user manual than technical documentation.

The documentation handles the complete scope of the framework, and is of good quality. Especially the API documentation is very clear and the use of the images is a very helpful addition.

6 German Team framework vs. CMU framework

Now that we've looked at some of the strengths and weaknesses of the German Team software, we can make a comparison to the CMU software. We had a talk with the group examining the CMU software, read their evaluation[6] and came up with the following differences.

6.1 Modularity

The German Team software is very modular, the modules are separated at conceptual level and have very little inter modular constraints. The CMU software seems to be a lot less modular. It still has modules, but these modules call parts of other modules.

The lack of modularity makes it hard to implement other wanted features of the framework: such as easily replaceable modules or having multiple modules being able to perform the same task.

A drawback of rigid modularity might be the ease of communication. Because the communication channels are predefined, you have to fit all the communication into these channels. There is, for example, no nice way to bypass the communication for a high speed interface.

The German Team uses a separate module for behaviour and evaluates behaviour programs during execution, this makes implementation and activation of new behaviours very easy. The German Team behaviour is expressed in XML, the CMU software has behaviour expressed in C++ code. The only drawback of evaluating the behaviour programs in runtime is that it costs a little processor power, but the expressiveness is the same as C++ code. It even makes dynamic role changing possible.

6.2 Maintenance and Changes

Sometimes changes need to be made to the code. For example, Sony might release a new model, or the rules of the RoboCup could change. It's important that these changes can be incorporated into the framework without too much effort. The modularity of the code makes this a little easier, but it turns out some things take quite some time to adjust. Another factor in the maintainability is the size of the code base. The German Team software has a very large portion of the code expressed in XML. Code, header files and XML are around 300,000 lines, while the CMU software only has

around 100,000 lines. This difference is mostly because of SimGT2003 and RobotControl.

When we were looking at the code, we found some assumptions about the robot which turned out not to be true any more for the new Aibo. The biggest problem people will have porting to the new Aibo will be the change in the joints in the neck and head. In the ERS-210, the Aibo had a tilt in the neck and a roll and pan in the head. The ERS-7 has a second tilt in the head instead of the roll. The assumption that the head could tilt, roll and pan was found in multiple modules, from the math through the image processing to the data structures in the localisation modules. Without this assumption the code would become too general and probably slow. The CMU software makes the same assumption.

6.3 Tools

Both frameworks have basic tools for debugging and monitoring the Aibos. The main difference between the two is the interface to the tools. While CMU uses command line tools, the German Team uses graphical user interfaces. The German Team has a visualisation component which makes the choice for a GUI a logical one. The CMU seems to have taken a more ad hoc attitude, creating small command line tools to perform tasks which turn out to be tedious. Unfortunately the CMU tools don't match the German Team tools in functionality and quality.

One of the drawbacks of using a GUI is platform independence: it's much more difficult to build a GUI that works on multiple operating systems than creating command line tools which do the same. The type of interface on programs is a matter of taste. Although a graphical simulator is a big advantage of the German Team framework.

6.4 Ease of Learning and Documentation

The first impression we got from the documentation of both frameworks was that the German Team documentation was much better. Later on, the other group found out the documentation was spread out over a large number of sources. Both code bases are well documented, the German Team has a little advantage on this point because it can generate HTML documentation from source.

The learning curves for both frameworks aren't very steep. If we had to choose, we would vote in favour on the German Team framework because the technical document is a very good way to get to know the system.

The documentation about installing the compilation software and compiling the code is quite clear and provides a step-by-step explanation of the actions needed to get the Aibos up and running.

7 Conclusions and Possible Improvements

7.1 Conclusion

The German team have a couple of important strengths. The code is clean and is highly modular. Graphical simulators are included that can be used for monitoring the Aibos and debugging purposes. Elaborate documentation is written and the code itself is well documented also, allowing a fast learning curve for people unfamiliar with the project. XABSL facilitates creating complex extensible agent behaviour solutions instead of just using C++.If we compare the software package of the German team to that of the CMU team, the German teams software package is better in the end.

7.2 Improvements

There are a couple of items we feel that can be improved.

Some of the vision algorithms used seem rather crude[3] at first sight. No colour-invariant features are used in general. Scan lines with a distance between them are being used to detect objects by looking at colour sequences of pixels. This is caused by real-time requirements. Calculating and scanning lines, edge finding, colour classification and recognising objects is computationally expensive if done frequently on the 400Mhz MIPS processor used in the ESR-210. The new model Aibo the ESR-7 has a 567Mhz MIPS processor which allows space for testing new algorithms.

Inter process-communication is achieved by using message-queues. If more communication would take place (e.g. from image-processing) this could become a bottleneck. A shared memory structure might be faster, despite the locking mechanisms that have to be used.

Compiling the code for the Aibo was done using the Cygwin Unix emulator and a MIPS cross-compiler supplied for that platform. It would have been easy if the MIPS cross-compiler had also been supplied for other platforms (Unix/Linux) as you really have to know what you are doing if you want to make one yourself. In order to facilitate generation of dependencies between source files and binaries a simple pre processor called Depend was used. A little more documentation would have been useful as the source code had to be in a native windows directory. It's easy to make the mistake of supplying a Unix directory, in which case the Depend program doesn't function properly.

References

[1] Thomas Röfer, E.A. Technical document RoboCup 2003, 2003, http://www.robocup.de/germanteam/GT2003.pdf.

[2] Röfer, T., Dahm, I., Düffert, U., Hoffmann, J., Jüngel, M., Kallnik, M., Lötzsch, M., Risler, M., Stelzer, M., Ziegler, J., GermanTeam 2003, 2003, In: 7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence, Padova, Italy, 2004 / http://www.robocup.de/germanteam/tdp03.pdf.

[3] Jüngel, M., Hoffmann, J., Lötzsch, M., A real-time auto-adjusting vision system for robotic soccer, 2004, n: 7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence, Padova, Italy, 2004 / http://www.informatik.hu-berlin.de/

[4] Lötzsch, M., Bach, J., Burkhard, H.-D., Jüngel, M., Designing agent behavior with the extensible agent behavior specification language XABSL, 2004, In: 7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence, Padova, Italy, 2004 / http://www.informatik.hu-berlin.de/

[5] Röfer, T, An Architecture for a National RoboCup Team, 2003, In: Kaminka, G. A., Lima, P. U., Rojas, R. (Eds.): RoboCup 2002: Robot Soccer World Cup VI. Lecture Notes in Artificial Intelligence / http://www.tzi.de/kogrob/papers/robocup03.pdf

[6] Patrick de Oude, Tim van Erven, Jochem Liem, Tim van Kasteren, Evaluation of CMPack 2003, January 2004, http://gene.science.uva.nl/ poude/doas/paper_cmpack03.pdf

[7] Ottens, B., Abbo, A., van der Meer, P.J., Stienstra, M., Aibo Project 2004 - Technical Report, 2004, http://carol.science.uva.nl/ mstnstra/projects/2004/aibo