openSDK - An Open-source Implementation of OPEN-R (Short Paper)

Nuno P. Lopes
nuno.lopes@ist.utl.pt

Pedro U. Lima
pal@isr.ist.utl.pt

Institute for Systems and Robotics
Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisbon, Portugal

ABSTRACT

This paper describes openSDK, an open-source implementation of Sony's AIBO development kit (OPEN-R). openSDK is capable of running unmodified AIBO programs (only a recompilation is necessary) on a standard computer, using a simulator at full frame rate (currently only USARSim is supported) or on a different robotic hardware platform. openSDK also offers standard debugging facilities for AIBO programs.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Applications; I.2.9 [**Artificial Intelligence**]: Robotics; D.4.9 [**Operating Systems**]: Systems Programs and Utilities

General Terms

Design, Experimentation, Languages

Keywords

robot simulation, robot virtual machine, AIBO, OPEN-R

1. INTRODUCTION

The development cycle time of applications designed for Sony's AIBO robots has always been long and painful, because of the lack of good debugging tools and of the time it takes to deploy the binaries to the robot and restart it. Some four legged league RoboCup teams (e.g. German Team) have developed mechanisms to help them running their own code on a standard computer, but they had to duplicate some parts of the code, worsening the maintenance problem. Most important is that they still would have to run the code on the AIBO to test the low-level code that was duplicated (and thus not run on the PC).
openSDK mitigates this problem by implementing Sony's AIBO API (named OPEN-R) and by allowing to run code designed for the AIBO on a standard computer, without requiring modifications to the application (only a recompilation is necessary). Moreover, openSDK is able to emulate the full AIBO platform (currently only ERS-7 is supported) with the help of an external simulator.
openSDK also eases the transition to new platforms, by enabling to run code designed for the AIBO on a new robotic platform, without modifications to the code. The process is explained later on this paper.
Although the RoboCup committee has decided to end the four-legged league and thus has shorten the openSDK lifetime, openSDK still constitutes an example of what and how can be done with future platforms and simulators/emulators.
This paper is organized as follows: in section II we give an overview of the implementation of openSDK, in section III we present the results of our experiments using openSDK, in section IV we present the known limitations of openSDK and related tools that were used, and in section V we present the conclusions of this work, as well as possible ideas for future works.

Cite as: openSDK - An Open-source Implementation of OPEN-R (Short Paper), Nuno P. Lopes, Pedro U. Lima, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 1207-1210.

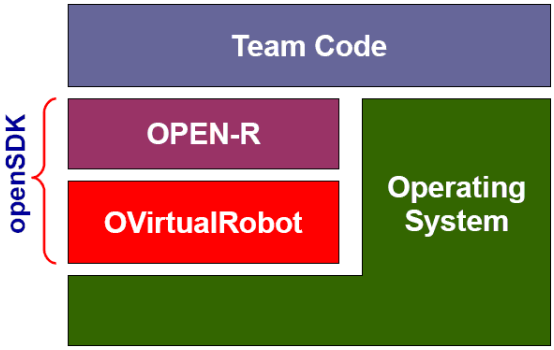

Figure 1: OPEN-R architecture

2. OPENSDK

This section provides an overview of the openSDK architecture and its implementation details.

2.1 Architecture

The openSDK architecture is itself based on OPEN-R and Aperios (AIBO's Operating System). It follows the same event-driven architecture that characterizes the AIBO platform. The top-level architecture of OPEN-R is shown in Fig. 1. What openSDK does is to implement both OPEN-R API and OVirtualRobot (the sensor input and joint actuator interface) layers. It also provides an OPEN-R module loader and a generic toolchain for compiling the code to run on openSDK.

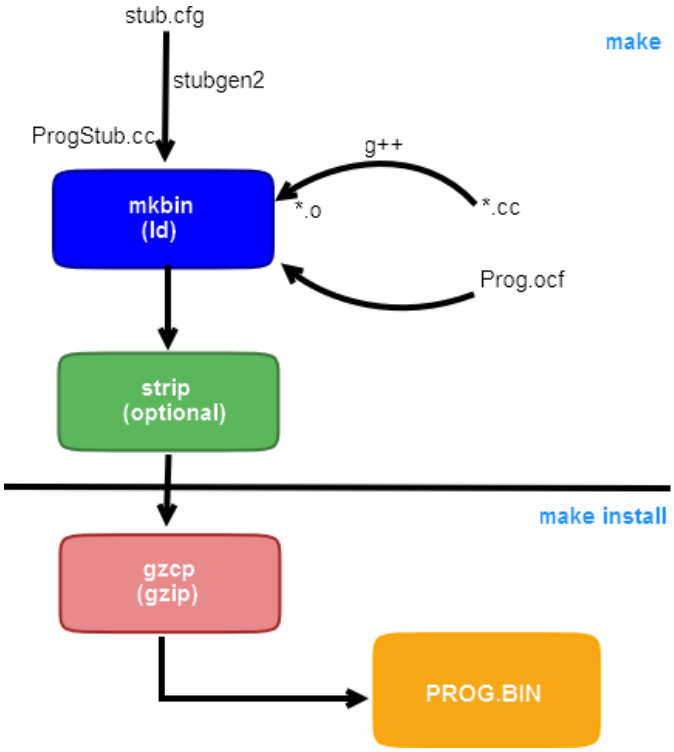

Figure 2: overview of the build process

2.2 Build Process

The typical build process of an AIBO program is shown in Fig. 2. openSDK is able to compile the code without much modifications to the Makefiles (typically it is only needed to change the OPEN-R root directory variable), by providing implementations of all the tools involved in the build process. The compiler and linker are provided directly by the system, while stubgen was implemented to generate specific stubs for openSDK (that are not compatible with the stubs generated by Sony's stubgen). The strip part is often disabled (and the developer is encouraged to do so), so that debugging tools can generate useful information (e.g. stack traces). The strip command is also provided by the system. The *.ocf files are simply ignored, because they are not relevant on Unix systems. The code is always run in no-TLB mode (one process, multiple threads: one per OPEN-R object) and in user mode and without heap or stack memory limits (except the ones imposed by the Operating System), regardless of what the ocf file says.

2.3 Operating System APIs

OPEN-R isn't the only available API for AIBO developers. The Aperios Operating System also exports some standard functions (e.g. open, read, etc...) into userland. However some of these functions perform differently from the ones found in most Unix systems (e.g. case-insensitive file system and no relative paths allowed). In order to emulate the AIBO platform, it was needed to replace some of these system functions with AIBO's equivalents. This is done at link-time, using ld's `-wrap` argument to redirect some function calls to openSDK wrappers. However, openSDK doesn't wrap (at least yet) C++ functions calls (e.g. `fstream::open()`). This means that the C++ file system related function calls aren't case-insensitive.
`strtok()` is also wrapped, to make it use local thread storage through the usage of the non-standard `strtok_r()` function. This effectively eliminates potential race-conditions between OPEN-R objects (which do not occur in the AIBO because the memory space of the objects is separated).

2.4 Module Loader

Each AIBO module is compiled to a `.BIN` file, which is nothing more than a gzipped Dynamic Shared Object (DSO). So the module loader decompresses these files at run-time and loads them using standard APIs (i.e. `dlopen()`).
Each module is run on a different thread, which resembles the no-TLB mode of the Aperios Operating System (the memory space is shared across modules). In order to guarantee that no symbol or variable clash occur (because two modules may have two variables with the same name), a GNU libc specific `dlopen()` trick is used (`RTLD_DEEPBIND`). More details on this feature are provided in [1].
After loading the module, a special entry-point function (generated by the stubgen2 script) `__start_module()` is called. This function is responsible for running the `DoInit()`, `DoStop()`, etc.. methods and for delivering the messages to the correct event handlers.

2.5 OVirtualRobot: Sensors and Joints Data

openSDK supports sensor data injection through sockets (Unix sockets are used for performance reasons). It also supports joint data export in real-time in the same manner. Data import/export is done using a client/server architecture and the clients can be changed at run-time without loss of data. For example, one can feed a recorded AIBO camera video or just a fixed image. One can also send the joint values to a real robot or simply print them to the screen, etc...
The sensor data is sent to the robot at fixed intervals (in the case of the AIBO ERS-7 it sends new data - four frames - every 32 ms). If no fresh data is available (for example, USARSim currently provides data at a lower frequency: five times per second only), the old data is sent instead. This is done to guarantee the timings of the AIBO platform, which many programs rely on.
OVirtualRobot is the module that makes it possible to run AIBO code on either a simulator or on a new robotic hardware platform. By replacing this module (or just read/write data to the current one), one can run the code on a new platform without much trouble.

2.6 Simulation

openSDK also supports robot simulation, using the USARSim [9] simulator with the AIBO module [10]. USARSim runs on top of the industrial game engine of Unreal Tournament 2004. It is able to feed camera images and sensor data to the simulated robot, as well as feeding joint data commands back to the simulator.
A sample camera image (as seen by the robot) simulated by USARSim can be seen in Fig. 3.
Currently running multiple robots in USARSim with live image feeding is not yet supported. This is due to the USARSim image server being only able to feed images from one robot at a time. There is some ongoing work to properly fix this. More details about the problem and possible solutions can be found in [10].

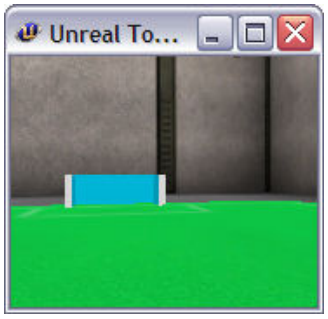

Figure 3: USARSim camera image simulation

2.7 Thread Safety

Thread safety is an important issue, and thus great care has been taken to insure that openSDK is thread-safe. All global message queues and resources are protected with mutexes. Local object resources aren't protected because there isn't concurrency inside the OPEN-R objects.
Also read the "Operating System APIs" section above for a note about `strtok()` thread-safety.

3. RESULTS

This section provides an overview of our experiments using openSDK with AIBO programs used in the RoboCup context.

3.1 SocRob-4LL

SocRob-4LL [2] is our team's code. The low-level part is based on Team Chaos [3] code base. A sample run of openSDK with our code can be seen in Fig. 4.
At first the code didn't compile with gcc 4.1 because it is much stricter and standard conformant than the old gcc version used by Sony's AIBO toolchain (gcc 3.3.2). After patching the errors spotted by gcc, the code would still not run (it was segfaulting). After diagnosing the problem, it was found that '`cout « "string"`' was the problem. After removing all references to it, the code run just fine. It is still under investigation if it is a gcc/libstd++ bug or if it is some problem in openSDK.
Another interesting problem we had was that once the code was crashing in the real robots, but not in openSDK. After investigation it was found that the problem was a CPU Floating-Point exception that was being triggered (related with unwanted mathematical operations with NaNs). Investigation on how to report these problems in openSDK (or simply crash it) in a portable way is yet to be done (possibly use the CPU configuration registers and make the Operating System trigger a SIGFPE signal).
openSDK proved extremely useful when implementing the communication protocols (it was only tested in openSDK and then it worked without modifications in the real robots as expected) and for debugging memory-related problems (including memory leaks and crashes). Valgrind [4] with its Memcheck tool [6] was the tool of choice when debugging the problems and it worked very well when using `-chroot=no` (described below).
With a Pentium M 2.0 Ghz (single core) we were able to run up to two robots on the same machine when forcing the camera images frame rate at 30 fps. When not forcing this high frame rate (and thus serve images at the rate

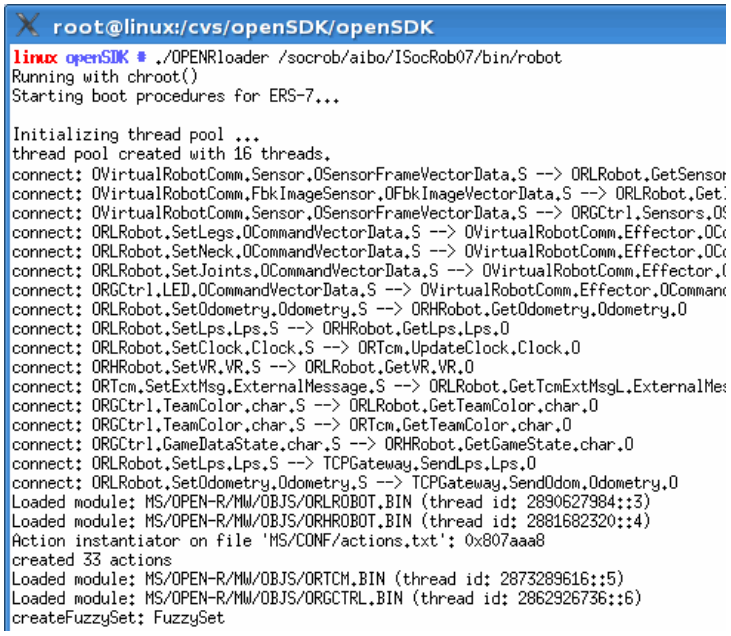

Figure 4: SocRob-4LL code running on openSDK

sent by the USARSim image server), we were able to run up to five robots on the same machine. The USARSim image server frame rate was always lower than 9 fps. USARSim was running on a Pentium 4 2.0 Ghz computer and the two computers were connected with a short 100 Mbps ethernet cable (to reduce latency). While a lower frame rate can be useful to run more robots on the same computer, it can produce unexpected behaviors, because some programs heavily rely on the correct timings of the sensor and image data event delivery.

3.2 CMU CMPack 2004

CMU's CMPack 2004 [8] code is also one of the major players in the RoboCup. However, due to the aging factor it won't compile cleanly with a recent gcc version (i.e. 4.0 onwards). After patching the problems and relaxing the new gcc diagnostics and restrictions (with `-fpermissive`), the code run without much problems under openSDK.

3.3 Tekkotsu

Tekkotsu [7] is a general-purpose development framework for the AIBO. Although the latest release at time of writing (3.0) didn't compile with gcc 4.1, the development CVS version compiled just fine, without any kind of modifications. However, it was not possible to run Tekkotsu under openSDK because of a "dirty" trick that it uses to speed-up the compilation time (it searches and replaces binary strings by "hand" in an object file). As a result, when trying to run Tekkotsu, openSDK would fail to load and report missing symbols, due to the broken ELF symbol table. Further investigation on how to fix the problem was not carried on.

4. KNOWN LIMITATIONS

During this work we found many bugs in the toolchain programs and in most debugging software we tried. Therefore we describe here the limitations we came across. We also describe some known limitations in openSDK itself.

4.1 Multiple Robots

Running multiple robots on the same machine with openSDK, while possible, poses some problems. The most obvious is the processing power (in particular the latency of commands, because of the real-time processing as insured by the AIBO Operating System), although with the arise of multi-core CPUs the problem should smooth down very quickly. Other problem is the network ports used by the robots, as most of them will bind to the same ports. This is important because the host Operating System (in this case, linux) will refuse to have multiple listeners on the same port. Some RoboCup teams (e.g. German Team [5]) already bind each of their robots to different ports, though.

4.2 Operating Systems

openSDK currently only supports the Linux platform. This means that we were not able to run the German Team 2004 [5] code due to the lack of support of the Cygwin platform.

4.3 Pthreads implementations

openSDK is a highly concurrent program and thus requires a good Pthreads implementation. We found a bug in the old LinuxThreads Pthreads implementation that didn't allow openSDK to run correctly. As this implementation is deprecated, it is advised to use the new NTPL Pthreads implementation (the default since glibc 2.4 and most recent linux distributions).

4.4 Debugging Tools

Most debugging tools used (Valgrind, GDB and Intel Thread Checker) have shown several limitations when instrumenting openSDK while running OPEN-R programs (all problems have already been reported to their authors). Some workarounds have already been added to the code, although most tools still fail miserably with the `chroot()` call that openSDK issues (to guarantee that it handles the absolute paths - `/MS/...` - correctly). It was added an option to openSDK to skip the `chroot()` call (`-chroot=no`), that should be used when debugging with one of the mentioned programs, otherwise they won't produce meaningful stack traces. Not using `chroot()` shouldn't be a problem as openSDK wraps the `open()` and `fopen()` function calls and handles the absolute paths automatically. openSDK isn't able (at least yet) to wrap C++ functions calls (e.g. `fstream::open()`), though.

5. CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this paper we have described an open-source implementation of OPEN-R that is able to run code designed for AIBO on a standard computer, as well as act as a simulator, or emulate the AIBO on a different robotic hardware platform, and all this without any modification to the sources (only a recompilation is necessary).
Although openSDK is still far from perfect, it already constitutes a great platform for all AIBO developers by reducing the development cycle time considerably.

5.2 Future Work

openSDK currently only supports Linux, but ongoing work to support Cygwin and MacOS is being done. Supporting AIBO models other than ERS-7 is also something we would like to accomplish. Sound input and output will be also looked at in the future.
In terms of simulation, we would like to be able to control more accurately the speed of the simulation (real-time and speedup/slowdown) and allow the user to control it. We also would like to add camera image filters to degrade the image fed by the simulator, to provide a more realistic environment.
We will also export some USARSim specific features into userland code and/or through an interface. These features include: ball and robots manual repositioning, as well as absolute location, orientation and velocity input for each robot (so that one can run a simulation without image processing, thus saving much computing power).
A full emulator of the AIBO platform based on an existent MIPS emulator (e.g. QEMU) is also being considered. This would allow running an AIBO memory stick without any kind of modifications (not even recompilations would be necessary) in another platform (including another legged robot).

6. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contribution of Marco Barbosa, who developed the initial network implementation (OPEN-R ANT) for openSDK and all the SocRob-4LL team for their feedback.
This work was supported by the Portuguese Fundação para a Ciência e Tecnologia under ISR/IST pluriannual funding through the POS_Conhecimento Program that includes FEDER funds.

7. REFERENCES

[1] U. Drepper. How To Write Shared Libraries 4.0, 2006.
[2] L. Iocchi, L. Marchetti, D. N. and. P. U. Lima, M. Barbosa, H. Pereira, and N. Lopes. SPQR + ISocRob - RoboCup 2007 Qualification Report, 2007.
[3] K. LeBlanc, S. Johansson, J. Malec, H. Martínez, and A. Saffiotti. Team Chaos 2004, 2004.
[4] N. Nethercote and J. Seward. Valgrind: A Framework for Heavyweight Dynamic Instrumentation. In *PLDI 2007*, 2007.
[5] T. Rofer, R. Brunn, I. Dahm, M. Hebbel, J. Hoffmann, M. Jungel, T. Laue, M. Lotzsch, W. Nistico, and M. Spranger. German team robocup 2004. 2004.
[6] J. Seward and N. Nethercote. Using Valgrind to detect undefined value errors with bit-precision. In *USENIX'05 Annual Technical Conference*, 2005.
[7] E. Tira-Thompson. A Rapid Development Framework for Robotics. Master's thesis, CMU, 2004.
[8] M. Veloso, P. Rybski, S. Chernova, D. Vail, S. Lenser, C. McMillen, J. Bruce, F. Tamburrino, J. Fasola, M. Carson, and A. Trevor. CMPack'04: Team Report, 2004.
[9] J. Wang, M. Lewis, and J. Gennari. A game engine based simulation of the NIST Urban Search & Rescue arenas. In *2003 Winter Simulation Conference*, 2003.
[10] M. Zaratti, M. Fratarcangeli, and L. Iocchi. A 3d simulator of multiple legged robots based on usarsim. In *Robocup Symposium*, 2006.