Improvements in the native development environment for Sony AIBO

Csaba Kertész
Tampere University of Applied Sciences (TAMK), Research Department, Tampere, Finland
Vincit Oy, Tampere, Finland

***Abstract* — The entertainment robotics have been on a peak with AIBO, but this robot brand has been discontinued by the Sony in 2006 to help its financial position. Among other reasons, the robot failed to enter into both the mainstream and the robotics research labs besides the RoboCup competitions, however, there were some attempts to use the robot for rehabilitation and emotional medical treatments. A native software development environment (Open-R SDK) was provided to program AIBO, nevertheless, the operating system (Aperios) induced difficulties for the students and the researchers in the software development. The author of this paper made efforts to update the Open-R and overcome the problems. More enhancements have been implemented in the core components, some software methodologies were applied to solve a number of restrictions and the achievements are summarized here.**

***Keywords* — AIBO, Aperios, toolchain, Open-R, SDK, URBI.**

I. Introduction

AIBO [5] was ahead of its time and the robot hardware can be still competitive for robotics research, however, the main emphasize of the industry is on humanoids, drones, manufacturing and healthcare services these years. Simpler four-legged robots (e.g Pleo [10], i-Cybie [6]) compared to AIBO have been on the market with similar target audience though less than a million entertainment robots with total value of US$ 166 million was sold in 2011 [12]. Interpreting these numbers, this is a small business segment worldwide with low demand and people are not familiar with robots.

Pleo [10], developed by Ugobe and later Jetta, imitates a baby dinosaur and the owner needs to teach from a toddler stage until a mature. This robot has two processors, 18 motors, plenty of sensors (camera, touch sensors, microphones, infrared distance sensor etc.) and it is capable for locomotion. Actions can be programmed in Python language, but the image and sound processing is slow and unreliable while AIBO ERS-7 can capture medium resolution images with 30 Hz and 2 msecs compression time.

i-Cybie [6] resembles a dog like AIBO and it can autonomously recharge itself, however, the repairs are not easy because of the lack of the modular design. It was sold during 2005-2006 without development environment, but the community did modifications to the hardware (Super i-Cybie) with soldering a communication port near the CPU and installing a new bootloader. With these changes, i-Cybie can be programmed in C under Windows and the sensors are accessible, but the SDK was abandoned in pre-alpha state with frequent freezes and almost no documentation.

A South Korean company (Dongbu Robot) sells a robot dog [3], which has a similar hardware configuration to AIBO, but Genibo does not have an open, low level software development environment, making impractical for researches.

Currently, there is no such an advanced and highly sophisticated quadruped system on the market like AIBO. If the shortcomings of the software environment can be fixed, the robot can be used for upcoming research topics. Several efforts have been done by the author of this study to renew the potential of the programming tools for AIBO. The next chapter gives a general overview of the operating system (Aperios) and the software development environment for AIBO then the updates of the cross compilation toolchain and a higher level middleware engine are described in details. The last chapters present the general development practices before a conclusion and an acknowledgment chapter.

II. Aperios, Open-R SDK

The Aperios, a real time operating system, was developed by the Sony's Computer Science Laboratory for TV set top boxes and AIBO. While the scheduling produces low latency, it is a closed, proprietary operating system without a Unix-like environment and lacking on-demand binary loader (shared libraries) or a reliable console emulator via serial connection.

There is no modern multithreading; the applications run as Open-R objects [11], loaded during the boot process from a memory stick, and they can communicate to each other with an inter-object messaging system. One object corresponds to one individual thread and the system scheduler gives control for a program in every 33 milliseconds (30 Hz). When a runtime error (e.g segmentation fault, division by zero or memory corruption) happens, the robot crashes and a dump is written to the memory stick (emon.log) before the immediate shut down. In practice, the crash cause can be obtained from this file, but instead of a meaningful backtrace, only crash address in the memory is provided. The compiler optimizations along with the function inlining make almost impossible to determinate the exact place of the error in the source codes.

The native software development kit, called Open-R SDK, is primarily targeted the Linux based systems although there was a Cygwin based version for Windows systems. The

DOI: 10.9781/ijimai.2013.237

gcc/newlib/binutils toolchain does not contain standard C programming APIs (e.g socket communication, data and time management) though some replacements are available.

These restrictions are troublesome for the native C/C++ development on AIBO because the written codes must be "perfect" to avoid any crash or memory leaks in the limited, not upgradable resources (e.g ERS-7: 576 Mhz MIPS processor, 64 MB RAM).

The update process of the software development kit is discussed in the following chapter.

III. Toolchain update

The Linux flavor of the Open-R SDK was selected for upgrade with newer components because the Windows version would be extremely hard to update. The official toolchain relies on four components to build programs for AIBO:

1. Gcc 3.3.x: a compiler to build C/C++ sources into mipsel object code.
2. Binutils 2.15: linker tools to assembly static libraries.
3. Newlib 1.15: minimal C library for non-Linux systems.
4. Pre-built Open-R system libraries and header files (robot hardware, network communication, date, time and other replacement APIs).

These tools are outdated and the C++ template support in gcc 3.x was quite incomplete, therefore, typical source codes do not build with this compiler nowadays. Extra efforts are required for backward compatibility and maintaining the changes over time. The intention was to bring the modern softwares to AIBO when the author of this paper upgraded the core components of the toolchain.

The building process for AIBO has four steps:

Step 1. The sources are cross compiled into object codes for the mipsel target.

Step 2. A static library is assembled from the compiled object codes and the prebuilt Open-R libraries.

Step 3. A new C++ source file is created with some Open-R tools because the Aperios needs a special descriptor about the exported symbols.

Step 4. The generated file is cross compiled and linked together with the static library. The result file is executable on the robot.

Since the Open-R object loading is hard coded and encrypted into the operating system, to update the cross compiler and the linker tools, the newer versions have to produce a compatible binary for Aperios. These programs are evolved mostly together, thus the choices are determinated which gcc/binutils release pairs can be tried as well as the delta between the gcc 3.3 and the later compilers grown over time. Because the verification method of the different compilers is to boot up the robot dog with a cross compiled binary and check if the robot runs or shuts down unexpectedly, the selection was done with trial and error in this first phase.

The original toolchain was distributed by Sony with a build script and some patches against the vanilla sources of the binutils, gcc and newlib. The patches were updated for later versions and a working set of the components were found: gcc 4.1.2, binutils 2.17 while the newlib remained the same. The last remaining challenge was the crashing exception handling. After low level debugging of the cross compiler with printf() and sleep() commands, the memory address translations were fixed when the exceptions are rethrown.

The unmodified, prebuilt Open-R libraries were compiled with the standard C++ library (libstdc++) in the gcc 3.3. Two software built against different versions of libstdc++ can not be mixed during linking (undefined symbols, redefinitions), as a result, all gcc 3.3 specific symbols were renamed in the Open-R libraries along with a copy of the old standard C++ library. In practice, some static memory allocation overhead (~120kB) happen for the new toolchain and the prebuilt Open-R libraries will call the old standard C++ implementation, but it does not result any abnormal operation in the applications on the robot. After these all kinds of modifications, it is interesting that the toolchain can be bootstrapped and compiled with the latest gcc versions (4.4-4.6.1) under Ubuntu Linux without updates to the build patches.

Finally, some words about the auxiliary tools. A helper source file is created during Step 3 with an application called gensnap. This script was written by Sony in Perl to dump the symbols with their addresses into text form with readelf/objdump for analysis and it outputs the needed new source file for Step 4 in the building process without any validation. This stage was slow while the Perl is an interpreted language. The gensnap and a validator (gensnapval) tools were written by the author in C++ along with a program (crashparser) to read the crash dump from emon.log and show the demangled symbol of the crashed function as well as where the return address points.

After the reborn of the original AIBO toolchain was described in these sections, the next chapter presents the update attempt of a new version of an interpreter engine.

IV. URBI 2 port

The Universal Robot Body Interface (URBI) was developed by Jean-Christophe Baillie [1] and later by the Gostai company. An URBI engine usually runs on a robot with an interpreter to parse scripts written in urbiscript language to manipulate the robot actuators and query the sensors. Remote objects for the robot can be created on the computer side to execute the heavy computations on the PC, but the bandwidth, the quality and the latency of the network connectivity limit this configuration.

The original URBI engine was written for Aibo and it was updated until version 1.5. Since the AIBO brand has been discontinued by Sony and the platform is difficult for development, it was dropped from the supported platforms in URBI 2.x. The author of this paper attempted to backport the new engine to AIBO and these efforts led to the toolchain upgrade described in the previous chapter.

After about half year programming, an alpha version of the URBI 2 engine built with the new toolchain was finished for

AIBO, the unnecessary features were cut, a setjmp()-based coroutine implementation was done and the software was adopted to the AIBO specific APIs, resulting a ~200kB patch against the URBI 2.3. The urbiscript tests passed, the engine was run on the robot, but the main bottleneck was the performance to finish the port.

In the URBI 2.x branch, more and more language primitives of the urbiscript were written in the script itself and relied extensively on the bison based parser. The further porting was stopped because writing some language constructs back to the native code and maintaining it with the upcoming URBI releases needed many efforts and the acceptable performance was not guaranteed. The URBI 2.x was claimed to run on the Spykee [9] (CPU: Armel, 200 Mhz) and the ERS-7 has a more powerful processor (MIPS, 576 Mhz), but the parser overloaded the CPU in idle state after constructing a couple of pure objects in urbiscript when it should not have any overhead (e.g without objects the parser consumed about 800 µs/cycle opposed to 16 msecs/cycle with 20 objects). The performance penalty could come from the fragmented memory usage and many cache misses, but the URBI 2.x did not prove to suit for AIBO as an embedded system. Albeit it was not a clear success, the efforts were rewarded by a 3rd place on the URBI Open Source Content in 2010.

The next chapters describe details about the development practices used in a real project for AIBO.

V. Development practices

The open source AiBO+ project[7] [7]-[8] is an attempt to write an alternative artificial intelligence for the Sony ERS-7 from the grounds. The first step of the project was the selection of a development environment and since the URBI 2 engine was not a viable option (Chapter 4), the Open-R SDK was updated (Chapter 3) and chosen.

By the reason of the Open-R SDK is a cross compilation toolkit, the usual architecture of the computers (x86/x64) and AIBO (MIPS) are different. Thus the Open-R binaries are not executable on the computer where they were built whilst a rich set of the debug tools (e.g valgrind, gdb, code coverage) are available there. A solution was needed to test the implemented algorithms without a real robot. The architectural, module, singleton and observer design patterns [2] have been applied to define functionally separated modules, but maintaining the interconnections between the objects with events. The behaviors, the actuator controllers have been developed in distinct components and integrated with a glue layer to the robot. Except this latter part, all other codes can be compiled on the host machine and tests are defined to verify the correct results with mocked hardware functions.

Usually, the bugs can be revealed and fixed with the previously mentioned patterns easily, but the performance and a solid medium for troubleshooting are also important.

The next subchapters describe several compiler optimizations, and in addition, improved WiFi stability for debugging.

TABLE I
Hardware features of the Sony AIBO robots

Robot name	Processor	Memory	Wireless
ERS-110	R4300i aka VR4300 (100 Mhz, RISC, MIPS)	16 MB	No
ERS-2x0	RM5321 (192 Mhz, RISC, MIPS)	32 MB	802.11b (optional)
ERS-2x0A	RM5321 (384 Mhz, RISC, MIPS)	32 MB	802.11b (optional)
ERS-3xx	RM5321 (192 Mhz, RISC, MIPS)	32 MB	No
ERS-7	RM7000 (576 Mhz, RISC, MIPS)	64 MB	802.11b (built-in)

A. *Compiler optimizations*

The faster execution is the primary reason to use compiler optimizations, however, the restricted resources in embedded systems introduce some trade-offs. The Sony robots have 16-64 MB RAM and their processors are limited to 100-576 Mhz (Table 1). The memory consumption is a major challenge because the dynamic memory allocations done by an Aperios program (Open-R object) decrease the overall available memory in the system and they are not usable by other objects anymore. The freed heap memory can be reallocated by the same Open-R object again, but it does not increase the free system memory anymore.

The data and code segments of a binary are loaded to the system memory, therefore, both the lower memory utilization and the faster execution through the less processor cache misses can benefit from a smaller program size. On the other hand, the function inlining can boost the performance, but it raises the chance of the 1st and 2nd level cache misses and the compiled code will reserve more system memory after loading. This trade-off needs a clever compromise to balance the memory usage and the fast execution.

Many optimizations of the compiler (gcc) and the linker (binutils) were examined to achieve improvements on the size individual flags almost accumulated after combining them together (Table 2) and the size of the compiled Open-R objects were reduced by ~25-35 %. The flags had no negative side-effects on the stability, but less static memory allocations and they have been used in the development of the AiBO+ project for years now. Table 2 contains gcc options for the ERS-7 robot model which have not been tested for the ERS-2x0 series, but all flags should behave the same, except the processor tuning flags (-mtune=vr5000 -march=vr5000 for ERS-2x0/ERS-2x0A).

Albeit the minimal size is an important measure, the shorter execution time what really matters. The most aggressive compiler optimization for speed (-O3) inlines the functions heavily and increases the binary size, nevertheless, it is faster 1.5-2 times than the compiler optimization for size (-Os).

[7] The project web address: http://aiboplus.sf.net

TABLE II
COMPILER FLAGS FOR SMALLER EXECUTABLE SIZE (RM7000, ERS-7). EACH ROW SHOWS THE IMPACT ON THE BINARY SIZE OF SOME COMPILER FLAGS WHILE THE LAST ROW HAS THE FINAL RESULT WHEN ALL FLAGS ARE APPLIED WITH THE SUM OF THE INDIVIDUAL IMPACTS IN BRACKETS.

Option	Description	Impact on binary size (URBI2 server)	Impact on binary size (AiBO+ server)
-mtune=rm7000 -march=rm7000	Optimization for the CPU in ERS-7	-11.8 %	-8.5 %
-fdata-sections -ffunction-sections	Removes the unused binary data from the code and data sections	-23.4 %	-17.9 %
-fno-enforce-eh-specs	Skip the runtime check of the C++ exception specifications	-1.03 %	-0.99 %
-fno-threadsafe-statics	Skip the thread-safe initialization of local static variables		
All flags applied		-34.7 % (-38.23 %)	-24.60 % (-27.39 %)

B. Wireless connectivity

The wireless connection is the only useful and direct debugging tool for application development on AIBO. The 2x0 series can be equipped with an optional 802.11b WLAN card which is built-in into the ERS-7M1/2/3. After the boot process, the robot dog can connect to a WiFi network, a telnet session may be opened on the port 59000. The system messages from Aperios and the debug messages of the Open-R objects are printed to this console. A solid connection is essential in this situation and it was analyzed by Hemel et al [4] to compare the ad-hoc and interactive mode performances with TCP transmission. They found the ad-hoc mode less reliable and fast than the connection via an access point. The best transfer rates were achieved when the computer was connected to a wireless router or access point with wired connection while AIBO used its wireless LAN card.

A basic requirement is the quick and frequent transfers of small packets. The robot can not satisfy this expectation with TCP sockets which are slow to deliver packets over the network (>= ~40 ms) and the socket state may be stuck for several hundred milliseconds caused by retransmissions. Therefore, the UDP transport layer was chosen and a minimalistic protocol was built upon. The log messages, the robot state and the camera images are bundled into a packet and sent to the computer. By compressing the content, a smaller packet (~5 kB/datagram) can be transmitted in every 100 msecs whereto an acknowledgment response is received.

The UDP is a connectionless protocol, packets can be lost or their order changed and the WiFi in ERS-7 has stability issues to be improved. To ensure the utmost arrived packets, an algorithm was developed whose pseudo code is shown in Figure 1. The steps are executed every time when a new packet is received from the robot and lists of the missing and received packets are maintained to handle the cases of lost or reordered datagrams. In the AiBO+ project, an ERS-7 opens a UDP socket to the computer automatically, it constructs and sends the datagrams continuously. Each packet has a unique, incremented ID and if the identifier of the arrived packet on PC side is higher than expected, the non-received IDs between the previous packet and the current are marked as missing. When missing packets are detected, higher latencies are taken into account and the oldest lost packets are requested again in a rotating order. This heuristic algorithm can recover the connection in most situations, but a bigger congestion can happen with high number of missing packets (element_count (MissingPacketIDs) >= 50) and the missing IDs are considered lost. Despite this technique, sometimes the socket to AIBO can be disconnected by unrecoverable problems on the link or internet layers which are out of control inside the operating systems.

```
MissingPacketIDs = ∅ // Sorted vector (ascending)
RequestedPacketIndex = -1
ArrivedPacketIDs = ∅ // Sorted vector (ascending)

Function PacketArrived(Packet, NewPacketID)
  if (NewPacketID ∈ ArrivedPacketIDs)
      Return
  if (NewPacketID ∈ MissingPacketIDs)
      MissingPacketIDs = MissingPacketIDs \ {NewPacketID}
  else
  if (NewPacketID – 1 >= ArrivedPacketIDs[last] + 1)
      Iterate (ID = ArrivedPacketIDs[last] + 1 to NewPacketID - 1)
          if (ID ∉ MissingPacketIDs)
              MissingPacketIDs = MissingPacketIDs ∪ {ID}
  end
  if (element_count(MissingPacketIDs) > 50)
      MissingPacketIDs = ∅
      RequestedPacketIndex = -1
  end
  if (MissingPacketIDs ≠ ∅)
      MaxRotationIndex = min(element_count(MissingPacketIDs), 5)
      RequestedPacketIndex = (RequestedPacketIndex+1) mod
                             MaxRotationIndex
      Request packet with ID = MissingPacketIDs[RequestedPacketIndex]
  end
  ArrivedPacketIDs = ArrivedPacketIDs ∪ {NewPacketID}
  if (sizeof(ArrivedPacketIDs) > 100)
      ArrivedPacketIDs = ArrivedPacketIDs \ {ArrivedPacketIDs[0]
```

Fig. 1. Heuristic algorithm to improve the network connection stability between a computer and Sony AIBO ERS-7

The last chapter summarizes the conclusions of the paper and gives an insight in the future work.

VI. CONCLUSION

When the AIBO brand was discontinued by Sony, it created a gap in the market and none of the upcoming developments could fill the need for a sophisticated quadruped robot with software access to the low level hardware.

The work detailed in this paper improved the toolchain support for AIBO and a more robust network connectivity was achieved. The compiled binaries have less size by ~30 % with gcc 4.1.2, modern softwares can be built for ERS-7 and the memory utilization is made better. The results have been used in the AiBO+ project successfully in the past years and the old gcc 3.x based as well as the new updated gcc 4.x based toolchains can be downloaded by anybody from a Personal Package Archive (PPA) for Ubuntu Linux[8]. The URBI 2 was not been finished, but it provided important experiences.

The future work can include the examination of more compiler flags for further optimizations in speed and size, but the current results needs also an applicability check for the ERS-2x0/ERS-2x0A models whose CPU and optional wireless cards are different from the ERS-7 model.

ACKNOWLEDGMENT

Thanks to other AIBO community members for sharing their AIBO tricks (AiboPet) and the valuable help in the maintenance of the robot dogs used in the project (WolfBob, Pablo). The advices from employees of Gostai company were also appreciated while porting the URBI 2 engine to AIBO.

REFERENCES

[1] J.-C. Baillie, “URBI: Towards a Universal Robotic Low-Level Programming Language” in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS05), 2005.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Pearson Education, USA, 2004.

[3] Genibo SD – User Manual, Gongbu Robot, Bucheon, South-Korea, 2011.

[4] Z. Hemel, J. Loriente, and L. Raphael, “Measuring the Wireless Performance of an AIBO”, Trinity College, NDS Paper, March 2006.

[5] L. Hohl, R. Tellez, O. Michel, and A. J. Ijspeert, “Aibo and Webots: Simulation, Wireless Remote Control and Controller Transfer”, In: Robotics and Autonomous Systems, Vol. 54, Num. 6, 2006, p. 472-485.

[6] i-Cybie Quick Start Owner ’s Guide, Hasbro/Tiger Electronics, 2001.

[7] C. Kertész, “A synchronized system concept and a reference implementation for a robot dog”, 14th Finnish Artificial Intelligence Conference (STeP), Espoo, Finland, 2010.

[8] C. Kertész, “Dynamic behavior network”, IEEE 10th Jubilee International Symposium on Applied Machine Intelligence and Informatics (SAMI), Herl'any, Slovakia, 2012.

[9] A. A. Kist, A. Maxwell, P. Gibbings, R. Fogarty, W. Midgley, and K. Noble, “Engineering for primary school children: learning with robots in a remote access laboratory”, 39th SEFI Annual Conference: Global Engineering Recognition, Sustainability and Mobility (SEFI 2011), Lisbon, Portugal, 2011.

[10] Pleo Programming Guide, Ugobe, 2008.

[11] F. M. Rico, W. R. Gonzalez-Careaga, J. María, J. M. C. Plaza, and V. M. Olivera, “Programming Model Based on Concurrent Objects for the AIBO Robot”, Journal: Actas de las XII Jornadas de Concurrencia y Sistemas Distribuídos, Spain, 2004.

[12] World Robotics: Service Robots, IFR (International Federation of Robotics), Statistical Department, Frankfurt am Main, Germany, 2012.

Csaba Kertész received his BSc degree in Computer Sciences focused on Artificial Intelligence from Budapest Tech and MSc in Computer Sciences from University of Szeged, Hungary. He is a Lead Engineer at Vincit Oy and works for Tampere University of Applied Sciences (TAMK) in Technical Lead/AI Specialist role in Tampere, Finland. His research interests include image processing in robotics, behavior-based systems and specialization in Sony AIBO robot dog.