From Games to Applications: Component reuse in Rescue Robots

Holger Kenn and Andreas Birk

School of Engineering and Science
International University Bremen
Campus Ring 12, D-28759 Bremen, Germany
h.kenn@iu-bremen.de
http://www.faculty.iu-bremen.de/kenn/

Abstract. Component-based software engineering is useful for embedded applications such as robotics. However, heavyweight component systems such as CORBA overstrain the ressources available in many embedded systems. Here, a lightweight component-based approach is used to implement the system software of the so-called CubeSystem, CubeOS. Since 1998, CubeOS and its component system have been successfully used in various areas from industry projects over RoboCup-related research to edutainment applications. Many of the components used in RoboCup soccer have been carried over in the implementation of the IUB Rescue robots, demonstrating the potential for software reuse.

1 Introduction

Component-based software engineering has emerged in recent years as a widely used approach to simplify software reuse. It relies on a large base of reusable software components and an integrating framework for those components [21]. Such frameworks for application software are CORBA [20] or JavaBeans [22] which alreay have been used in Robocup[16].

The main benefit of software component approaches are the possibility to reuse components within a framework. This leads to a significantly reduced development time both for implementation and testing/debugging if a high number of tested components is readily available.

The same advantages of component-oriented software engineering can also be used in the design of software for embedded systems. With the tendency towards more and more complex embedded systems, handling this complexity in the software design process becomes more important and component-oriented approaches are one solution to this.

Lightweight component architectures try to limit the overhead that is created by the component infrastructure without losing the advantages of the component-oriented software engineering approach and maintaining a maximum of the protection features of a heavyweight component architecture. Examples of lightweight component architectures for implementing embedded operating systems are pebble [3] and eCos [13].

In this paper, a very simple lightweight component-oriented approach is described and it is shown how it has been applied to the design of the system and application

software for robots. The main design goals for this software system have been minimal performance overhead, very modest hardware requirements and the possibility for code reuse by multiple groups and projects.

The CubeOS system described here has been continously developed since 1998 and has been used in several research projects. Its main benefit was the reuse of existing component code for future projects. For example a large part of the low-level control software of the current robots platforms of the IUB Robocup Rescue [7] team is based on existing components of the VUB AI Lab Robocup Smallsize League [6] team.

2 Autonomous systems

In the recent years, research on autonomous systems, i.e. networked embedded devices has shown the need for reliable energy-efficient low-cost computing platforms. Where it is possible, such as in systems used in the RoboCup [14] Middle-Size Robot League, this computing platform mostly consists of embedded PC hardware, running commercial or free general-purpose operating systems. However, for applications where the physical size and the energy sources of a device are restricted even further, these PC-hardware-based approaches are of limited use. Several other available platforms such as Lego Mindstorms [17] have limited compute ressources that restrict their use to Edutainment applications [2]. The need for a small and energy-efficient extendible platform led to the development of the CubeSystem [10]. Apart from its use as a standalone controller [6] the cube system can also be used in combination with PC hardware to execute realtime control tasks [8, 7].

Together with this new hardware platform, a new approach to system software design based on lightweight components has been pursued. For autonomous systems, the system software has to provide only limited services

- Standard operating system functions such as concurrent thread execution, inter-thread communication and synchronization, time measurement and realtime clock services.
- Interface code for sensor- and actuator devices ranging from simple i/o functions to complex software for computer vision applications
- Ad-hoc network communication service between multiple systems using various communication interfaces, e.g. wired bus systems, radio communication etc.
- Mechanisms that allow the extension of the system by third parties to enable the integration of new hard- and software.

Many of the features mentioned here are available in commercial operating systems for deeply embedded devices. Unfortunately, these come at a significant cost and/or restrictive license conditions. However, in order to benefit from the component-oriented software engineering approach through code reuse and to be able to use CubeOS for various projects with different licensing requirements, an open-source approach was chosen for the implementation of CubeOS. Other Projects such as eCos [13] later followed a similar approach.

3 A look at the CubeSystem hardware platform

The RoboCube hardware platform [10] is using the CPU32 Core [18] as CPU in the M68332 MCU [19]. It is a 32-bit CISC architecture without MMU or cache. Additionally, the MCU contains functions such as local memory, serial I/O, timer functions and programmable chip selects that make it suitable for the design of embedded systems. The RoboCube hardware platform extends the MCU with at least 1 Mbyte of Flash-ROM and 1 Mbyte of S-RAM. These components form the CPU board.

The CPU board is about 8×8 centimeters in size and about two centimeters in height. It has two stacking connectors a the edges that carry all bus and power signals so that multiple boards with similar connectors can be stacked together. These form a CubeSystem. In a cube system, exactly one CPU board must be present.

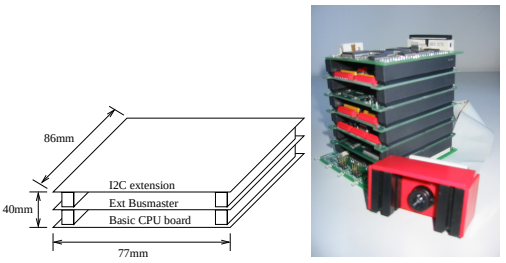

Fig. 1. The physical layout of a RoboCube stack of boards

Apart from the CPU board, there are various extension boards available such as memory boards (4Mbytes of SRAM), Bus Controller Boards (2x UART, 2x I^2C) and I/O Boards (24 8-bit A/D inputs, 6 D/A outputs and 16 bidirectional digital I/Os on one I/O Board).

The users can develop application-specific extension boards. The most basic application-specific extension boards are so-called base boards that implement application-specific power supply and I/O. Such application-specific boards have been developed for various projects [8, 7, 9, 5].

From this description of the hardware, it can be seen that the platform is quite flexible, not only in the way that users can use different hardware modules of similar functionality but that the hardware platform can be extended with completely new functions. The system software has to accommodate this by supporting the user in developing software that works with custom hardware at ease. An unfortunate feature of the CPU is that it does not contain a memory management unit, i.e. it is unable to offer memory protection services that could be used to separate components, i.e. preventing intentional or unintentional manipulation of memory used by a different component.

4 The component design of CubeOS

According to [23], a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Applications are created by combining components from possibly different origins. From this it can be concluded that a software component is a unit of independent deployment and of third-party composition. Unlike an object, i.e. an instance of a class in object-oriented languages, a component has no persistent state in itself, so there are no multiple instances of components, either a component is available or not, but it is not available several times.

This illustrates the strong correspondence between software modules and software components. A software module (such as a C object-code module) can implement the code for a software component. But the module itself is not sufficient to form a component since it does not necessarily implement well-defined interfaces, e.g. it does not protect its internal variables.

CubeOS uses a simple approach based on object modules and the standard C linker to form a component system.What is needed to make a software module a software component? Three requirements have to be met:

1. In order to separate the interface of the module from its inner structure, it has to be made clear whether an object belongs to the interface or to the implementation.(However, for grey-box testing purposes, it is still advisable to export the inner structure of the modules.)
2. Since the C linker identifies every object with an unique name, it has to be made sure that no two objects use the same name for any interface or implementation object.
3. From the two former steps, it is clear that a component system can be formed by linking the modules together if each module (and the application program) uses the appropriate interface objects of the other modules. However, this is not sufficient to make sure that the modules can be used independently. To ensure this, modules of an appropriate size have to be defined and their interfaces have to be documented.

CubeOS implements the first and the second requirement by prefixing every object name of a component with the component name, e.g. `KERN_schedule()` is the interface to the kernel scheduler. Internal objects get an additional `_` , e.g. the `KERN` component implements the process table in a private array: `extern struct process _KERN_ptable[];` . This has been considered good practice for coding C-modules but can be used in the same way in a component-based design process.

Unfortunately, neither the C language nor the hardware used support access-protection. Therefore, the method of marking the objects thorugh this naming scheme helps the user to observe the acces rules.

The third requirement cannot be met within the programming language itself but is a requirement for the implementation. The only option for a component system is to encourage the users of making their components reusable by providing an adequate set of tools that make it simpler to do so.

CubeOS uses the software documentation system “Doxygen” for this purpose. Although originally designed to document object-oriented software, Doxygen can be used to document component systems as well. For this, documentation groups with the same

name as the component are used. Doxygen includes the documentation in the source code by using specially formated comments.

The graph that is shown in Figure 2 illustrates the relation of the KERN component with other components. This graph can automatically be created by the build system from the object code and is helpful for documentation and error analysis.

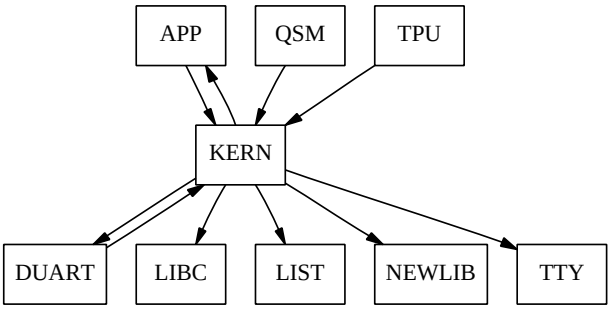

Fig. 2. The graph representing the component interdependencies of the KERN component with arrows pointing from the origin of the call to the called component. The LIBC and NEWLIB interface components represent the calls to the C library, the APP component represents the application program, in this case the CUBEOS test library. The call from KERN to APP is the initial call of `main()`. Note that calls between other components such as calls from APP to NEWLIB are not shown in this graph.

Often, it is necessary to interface legacy code such as standard or mathematical libraries. Examples of such legacy libraries that have been ported to CubeOS are the XDR libraries and the open-source JPEG compression library.

5 Successful applications of CubeOS in RoboCup

The CubeOS has been used in various robotics applications (figure 3), ranging from educational activities [2, 5] over basic research [1] to industrial applications [8, 7] . In all projects, the usage of CubeOS proved to be beneficial both in respect to the fast development time of the overall systems as well as in respect to the stability and reliability of the systems. Moreover, the use of the component-oriented approach enables code-sharing between many of the projects mentioned that goes beyond the operating system itself. For example, a number of general-purpose mobile robot control components has been implemented that implement PID motor control, odometric pose tracking and high-level motion commands.

One application is for example within the Small Robots League of RoboCup, the world championship of robot soccer [14, 15]. The CubeOS has been used on robot teams from the Vrije Universiteit Brussel (VUB) and recently from the International University Bremen (IUB). The teams participated in various tournaments, including RoboCup

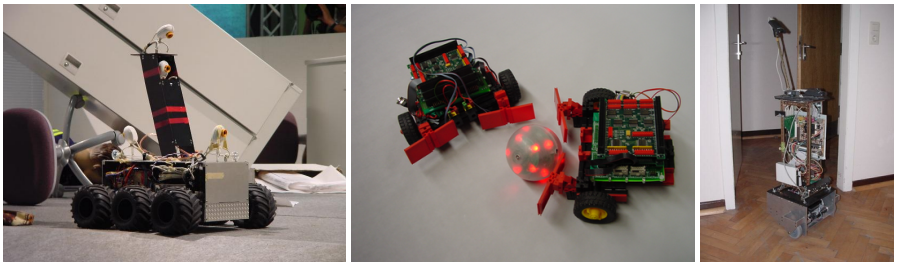

Fig. 3. Left: One of the IUB rescue robots performing in the NIST testing arena during RoboCup 2002 in Fukuoka, Japan. Center: Two robots designed and programmed by high school students for a robotics competition. Right: The inside core of the RoboGuard base, a commercial semi-autonomous robot for surveillance applications.

World Championship'98 in Paris, RoboCup World Championship'99 in Stockholm, the RoboCup European Championship 2000 in Amsterdam [12, 11].

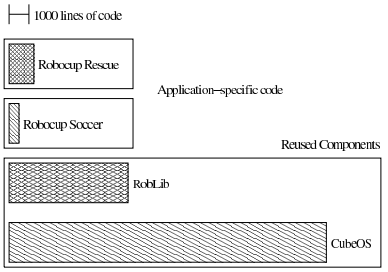

Fig. 4. This diagram illustrates that most software used for the IUB Rescue Robots has been reused from the soccer robots.

When in 2002 a seemingly completely different task within RoboCup was pursued by the IUB team, namely the creation of rescue robots, it turned out that due to the design of CubeOS, most of the code from the soccer robots could indeed be reused. The concrete numbers are shown in Figure 4. It shows that 97.8 % of the code were be reused.

6 Conclusion

CubeOS demonstrates the use of a lightweight component-oriented design approach for the design of both system and application software on multiple embedded mobile robot platforms used for Robocup research. This represents an advantage over the use of heavyweight component architectures in embedded systems since it reduces ressource

usage that is critical for embedded applications such as mobile robotics but still gives the benefits of component-oriented software engineering such as code reuse. The component model of CubeOS is very simple, thereby simplifying code reuse of existing legacy code and the from-scratch implementation of reusable components. CubeOS and its component system have been successfully used since 1998 in various areas from industry projects over RoboCup robotics research to edutainment applications.

References

1. Holger Kenn Andreas Birk and Luc Steels. Programming with behavior processes. *International Journal of Robotics and Autonomous Systems*, 39:115–127, 2002.
2. Minoru Asada, Raffaello D'Andrea, Andreas Birk, Hiroaki Kitano, and Manuela Veloso. Robotics in edutainment. In *Proceedings of the International Conference on Robotics and Automation, ICRA'2000*, 2000.
3. John Bruno, Jose Brustoloni, Eran Gabber, Avi Silberschatz, Christopher Small. Pebble: A Component-Based Operating System for Embedded Applications In *Proceedings of the USENIX Workshop on Embedded Systems* , 1999.
4. D. (Daniele) Bovet and Marco Cesati. *Understanding the Linux kernel*. O'Reilly & Associates, Inc., 2000
5. Andreas Birk, Wolfgang Günther, and Holger Kenn. Development of an advanced robotics-kit for education and entertainment of non-experts. In *1st International Workshop on Edutainment Robotics*, 2000.
6. Andreas Birk and Holger Kenn. Heterogeneity and on-board control in the small robots league. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, number 1856 in LNAI, pages 196 – 209. Springer, 1999.
7. Andreas Birk and Holger Kenn. A control architecture for a rescue robot ensuring safe semi-autonomous operation. In Gal Kaminka, Pedro U. Lima, and Raul Rojas, editors, *RoboCup-02: Robot Soccer World Cup VI*, LNAI. Springer, 2002.
8. Andreas Birk and Holger Kenn. Roboguard, a teleoperated mobile security robot. *Control Engineering Practice*, in press, 2002.
9. Andreas Birk, Holger Kenn, Martijn Rooker, Agrawal Akhil, Balan Horia Vlad, Burger Nina, Burger-Scheidlin Christoph, Devanathan Vinod, Erhan Dumitru, Hepes Ioan, Jain Aakash, Jain Premvir, Liebald Benjamin, Luksys Gediminas, Marisano James, Pfeil Andreas, Pfingsthorn Max, Sojakova Kristina, Suwanketnikom Jormquan, and Wucherpfennig Julian. The iub 2002 smallsize league team. In Gal Kaminka, Pedro U. Lima, and Raul Rojas, editors, *RoboCup-02: Robot Soccer World Cup VI*, LNAI. Springer, 2002.
10. Andreas Birk, Holger Kenn, and Thomas Walle. On-board control in the robocup small robots league. *Advanced Robotics Journal*, 14(1):27 – 36, 2000.
11. Andreas Birk, Thomas Walle, Tony Belpaeme, Johan Parent, Tom De Vlaminck, and Holger Kenn. The small league robocup team of the vub ai-lab. In *Proc. of The Second International Workshop on RoboCup*. Springer, 1998.
12. Andreas Birk, Thomas Walle, Tony Belpaeme, and Holger Kenn. The vub ai-lab robocup'99 small league team. In *Proc. of the Third RoboCup*. Springer, 1999.
13. The eCos open-source embedded operating system See `http://sources.redhat.com/ecos/`
14. Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proc. of The First International Conference on Autonomous Agents (Agents-97)*. The ACM Press, 1997.

15. Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada. The robocup synthetic agent challenge 97. In *Proceedings of IJCAI-97*, 1997.
16. Gerhard K. Kraetzschmar, Hans Utz, Stefan Sablatng, Stefan Enderle, and Gnth er Palm. Miro - Middleware for Cooperative Robotics. In *Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, Proceedings of RoboCup-2001 Symposium, volume 2377 of Lecture Notes in Artificial Intelligence, pages 411-416*, Berlin, Heidelberg, Germany, 2002. Springer-Verlag.
17. Henrik Hautop Lund and Luigi Pagliarin Robot Soccer with LEGO Mindstorms LNCS 1604, 1999
18. Motorola, Inc. CPU32 Reference Manual, 1996 see `http://www.motorola.com`
19. Motorola, Inc. 68332 Users Manual, 1995 see `http://www.motorola.com`
20. The Object Management Group (OMG) see `http://www.omg.org`
21. Ian Sommerville Software Engineering, 6th Edition Addison Wesley, 2001
22. JavaSoft, Sun Microsystems, Inc. JavaBeans Components API for Java, 1997
23. Clemens Szyperski. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 1999.