Proceedings of the 2001 IEEE
International Conference on Robotics & Automation
Seoul, Korea • May 21-26, 2001

Open Robot Control Software: the OROCOS project

Herman Bruyninckx*
Mechanical Engineering, Katholieke Universiteit Leuven, Belgium
http://www.mech.kuleuven.ac.be/~bruyninc

Abstract

This paper introduces the OROCOS project (www.orocos.org), that aims at becoming a general-purpose and open robot control software package. OROCOS follows the Open Source development model that has been proven to work in many other general-purpose software packages, such as Linux, Apache, Perl, or LaTeX. The paper focuses on the long-term vision of this start-up project, motivates which strategic and innovative design decisions are to be taken (a CORBA(-like) component architecture being the most important one, [14]), and lists other projects on which OROCOS could build. The success of OROCOS depends critically on how many researchers and engineers can be motivated to contribute code, documentation and feedback to the project.

1 Introduction

The robot industry is about half a century old, and has always been keen to adapt standards and high-tech evolutions from electronics and mechanics, especially materials, acutators, and electronic communication. However, the industry's current *software* practices are very similar to those in the computer business some 20 years ago: every manufacturer has its own proprietary software, algorithms, data structures, as well as programming languages. Efforts to define open data format standards or programming languages have had very minor success, such that software exchange possibilities are really extremely low.

There are many reasons for this lack of interoperability and openness in the robotics industry; some of the more important ones are: (i) all manufacturers operate in niche markets (that is, a niche in *scope*, certainly not in *size*!) such as manufacturing or assembly; (ii) the largest share of the market is covered by large corporate customers that make long-term investments and demand high reliability, and not by very price-sensitive individual consumers with rapidly varying tastes; (iii) the academic "customers," interested in the development of robotic systems with a much wider and ambitious scope, represent only negligable market share.

As a result, robotics is still an industry with a very high "user lock-in," leading to the corresponding huge investment threshold for newcomers and low cross-fertilization rate of innovations. This situation has only become worse after the worldwide mergers of the last ten years or so, with only a handful of general purpose manufacturers remaining. On the other hand, many relatively small companies have appeared, which offer a wide range of innovative (but mutually incompatible) robotics products, that are aimed at the (growing) academic, special-purpose or home markets; for example, RWI/iRobot [7, 16], Nomadic [13], Khepera/CyberBotics [8, 27], etc. The repercussions of this commercial situation on academic robotics institutes is that exchange of software research results is almost inexistant, and independent "benchmark" comparisons of results are impossible.

In contrast to what has happened in robotics, the *computer business* scene has gone through a radically different evolution during the last couple of years: the appearance and growing maturity of the *Linux* operating system and other Free Software and Open Source software packages (the GNU tools, Apache, Gnome and KDE, Perl, etc.), together with the massive employment of the Internet and its open communication standards (TCP/IP, HTML, CORBA, etc.), has drastically lowered the monetary threshold for newcomers and consumers, *and* (or rather, *because*!) it has allowed the easy, cheap, instantaneous, and worldwide exchange of software and data.

There is no reason why this "democratization process" of the computer industry could not be applied to the robotics area too. However, until now, no (proprietary or Open) general-purpose robot control software package exists. The goal of this paper is to present such an emerging and ambitious effort, that goes under the name of *OROCOS* (*O*pen *RO*bot *CO*ntrol *S*oftware).

*Postdoctoral Fellow of the F.W.O–Vlaanderen. The financial support by the Belgian State—Prime Minister's Office—Science Policy Programme (IUAP), and by K.U.Leuven's Concerted Research Action GOA/99/04 are gratefully acknowledged.

The following Sections explain the long-term goals of OROCOS, the modern software engineering principles the project will use, the Open Standards with which its software and data formats will work, the business model companies will be able to use to generate an income based on OROCOS (in synergy with the project's Open Source nature!), and the practical, Internet-based day-to-day management and organisation of the project.

2 Goals and vision

The OROCOS project is all about developing robot control software

- under Open Source license, [15]. That is, the source code is freely available for study, use and modification. The GPL license (GNU General Public License, [21]) is a major candidate because it has proven to work for Linux. However, it forces all "derived code" to be released under the GPL too. Linux Torvalds solved this problem for the Linux kernel by explicitly allowing proprietary code to be linked to the kernel code; another possibility are less stringent licenses such as the LGPL (Lesser However, OROCOS has a competitive advantage over proprietary libraries (simply because the latter don't exist) so that a more compliant license is not really necessary [22].
- with extreme modularity and flexibility. Such that (i) users can build their own system *à la carte*, and (ii) developers can contribute to the modules they are interested in, without the need to delve into the code of the whole system.
- of the highest quality, from both the technical, documentation, and software engineering points of view. The education and documentation aspects receive special attention within the project: the OROCOS documentation should not be limited to the classical user and reference guides for the software, but, due to its Open Source nature, it can serve as a basis for educational purposes too, e.g., classroom notes illustrated with the examples from the OROCOS code and OROCOS projects. By starting the project with an extensive design brainstorming phase, we want to increase the chances that documentation is available together with the code, or even before it!
- independently of (but if possible compatible with) commercial robot manufacturers. Initially, especially the smaller and/or service-oriented businesses will be interested in using the OROCOS code, and in contributing to it. In the long run, the OROCOS code should become an inevitable *de facto* "Open Standard," comparable to the acceptance of Linux in the mainstream computer industry. One of the major gaps in the current robot industry is the lack of a common programming API (Application Programming Interface), and trajectory generation and motion control libraries; an independent project such as OROCOS is ideal to suggest roadmaps to fill this gap.
- for all sorts of robotic devices and computer platforms. (Note that the large majority of Open Source projects are not at all limited to the Linux platform, but run on Windows, MacOS, Unices, etc.) The largest obstacle in this context are the "off the shelf" robot hardware systems, because their software architecture seldom provides sufficient openness. However, many of those systems have already been "hacked" by the robotics research community, and the results can easily be added as the "Open Hardware" part of OROCOS. As soon as OROCOS proves to be a viable and reliable software project, new robotic systems will adapt it as (one possible) default control environment.
- *localized* for all languages. That means that the software should be straightforward to configure in such a way that all dialogues with users take place in the users' preferred language.
- featuring software *components* for kinematics, dynamics, planning, sensing, control, hardware interfacing, etc. The meaning of "component" in this context corresponds to what is described in so-called "middleware" software (CORBA, DCOM, or RIM), [4], i.e., a software object that can dynamically be added or removed from a network and that offers its services through a neutral, programming language independent interface (such as CORBA's Interface Description Language IDL, [14]).

The project aims at becoming *much more* than just a copy of existing commercial robot controllers or robot simulation/programming packages: these commercial offerings are, up to now, always limited to niches in the market (although these niches often represent billion euros economies!) and are very difficult to extend, to combine, or to run on any hardware or over a network. The difficulties come not only from limits on the scope of the projects, but often also from the limitations imposed by commercial licenses.

The OROCOS project on the other hand, starts out from the opposite side: it wants to develop *shareable libraries* that are *platform independent*, implement *stand-alone component* examples that can be adapted and extended by end-users, and develop a *configurable run-time environment* from which to simulate and control all possible *distributed robotic systems*. The result should be able to appeal to *all* roboticists, whether they are students interested in extending the soccer server of RoboCup, [17], or a multinational team of engineers from different companies that have to get a five-finger robot hand work together with a redundant manipulator on a planetary rover under vision-guided teleoperation, where all parts come from different manufacturers.

These aims are very ambitious (but realistic), but they are only achievable in an Open Source effort: no proprietary developer can (or wants to, or has the man power to) cover the same broad scope. However, nothing prevents commercial companies from using (part of) the OROCOS software, or even contributing to it. It might be a cliché, but the Linux project is a proven example of a very ambitious undertaking that now receives contributions from both volunteers and professional software engineers, working in isolation or in the framework of multinational corporations.

The goals of this paper are: (i) to introduce the OROCOS project to the public, (ii) to explain the motivations behind its long-term strategy, (iii) to invite colleagues from all over the world to brainstorm about its *design*; and (vi) to raise the interest of "robot hackers" to contribute, review and extend code. Indeed, experience has learned that two requirements should be fulfilled before an Open Source project will succeed: (i) it should start with a *clear vision* of what it want to achieve; and then (ii) it needs a *critical mass of skilled contributors* to make that vision happen. Both requirements are surely fulfilled in robotics! The only(?) problem will be to manage these heterogeneous forces such that they share the same long-term goals.

3 Modules

A huge software project such as OROCOS should rely on a "divide and conquer" approach, in order to keep complexity manageable. The OROCOS code (and documentation!) base is to be divided into *modules*, or *libraries*. There are roughly three major types of modules:

- *Supporting modules*. That is, the software without functional robotics contents, but that is nevertheless needed to build a working robot control system. For example, 3D visualisation and simulation; numerical library; software component configuration tools; real-time operating system; inter-process communication; documentation writing tools; etc. These modules can almost completely be "recycled" from already existing Open Source projects, see Sect. 7.

- *Robotics modules*. That is, the software that implements specific robotics algorithms: kinematics and dynamics of (both general and specific) kinematic chains; servo controllers; Baysian and Neural Network estimators; motion planners for mobile robots, serial and parallel manipulators; etc. The Robotics modules make use of one or more of the *Supporting modules*.

- *Components*. That is, the CORBA objects with their IDL descriptions. These components are constructed out of the previous two types of modules, and they are the building blocks with which users "assemble" their (distributed) robot control software environment.

4 Software engineering approach

The Robotics and Component modules are the *innovative core* of the OROCOS project. "Innovative core" has two complementary meanings, i.e., in the areas of

1. *Research*: novel ideas and/or algorithms, i.e., not yet available in academic or commercial robot systems. Much novelty also lies in the ambitious *scope* of the applications.

2. *Software engineering*: many algorithms are, in one form or another, already in use in commercial robot systems for many years, but they cannot be accessed for improvement or reuse in other applications. However, pouring them in a sofware module that will function as, for example, a CORBA component, requires creative software engineering, and, in most cases, re-creation of the source code. This is an excellent occasion to apply modern software engineering principles, i.e., object-orientation, software patterns, and distributed components.

The common *design strategy* underlying the development of each OROCOS module is as follows: in a first step, the long-term vision is openly discussed by means of the OROCOS mailinglist; the developers look for possible *Software Patterns* that are transferable from other domains or have emerged in robotics over the last couple of decades; and, finally, implementation starts and is iterated. Of course, in practice these steps occur most often in parallel. Open Source projects have proven their enormous flexibility in this respect: whenever rational arguments exist to recreate code or redesign parts of the system, there are no marketing or "backwards compatibility" limits to keep the community from doing it. In contrast to the commercial software industry, backwards compatibility is not such a big constraint, because (i) all older versions remain available, and (ii) the cost to upgrade is marginal.

A large component-based software system such as OROCOS will profit maximally from two of the major evolutions in software engineering: object-orientation and software patterns. The latter in particular will get much emphasis, because it is still almost completely absent from the robotics research scene, while a project such as OROCOS is the perfect initiative to fill this gap in an innovative way. The following paragraphs explain the major features of both software engineering concepts, but the interested reader is refered to the abundant literature for more details.

1. *Object orientation.* A large software system is divided into smaller "objects," with well-described *interfaces*, that other objects can use. The objects themselves are responsible for *how* they implement these interfaces.

2. *Software patterns.* The development of a new software system becomes faster and more reliable, whenever one can find a part of the whole system that shows the same behaviour as previously implemented software systems, or as a well-known behaviours from everyday life in human society. Such a recognizable behaviour is called a "*pattern*," [5].

   Well-known examples that are immediately useful in a robotics context are [1, 5]: the *Model–View–Control* pattern for graphical user interfaces; the *Producer–Consumer* model for distributing data over different, dynamically changing clients on a network.

   A software pattern is a set of objects and methods, whose functionality and inter-relationships have been proven to work in common software practice or in real human life. A pattern exists in a possibly very different domain than the one in which the software system is to be implemented. It need even not exist in the form of a software module at all, as long as the expected behaviour of the whole system is intuitively clear to the human developer and/or user, because they recognize it from their everyday experiences. The goal of introducing Software Patterns is to have users recognize these patterns in the system at hand, by giving the pattern a suitable, intuitive name, and then to implement the code in such a way that it satisfies the (often implicit) expectations that users have in the context of the intuitively named pattern. That is, the name of a pattern brings to live a whole *context* within the user's mind, and this context makes it easier to (i) understand the functionality of the different methods implemented in the software that comes with the pattern, and hence (ii) extend and/or maintain the current implementation of the pattern, even for people that were not involved in the original coding.

   Very few of the Patterns that implicitly appear in many robotic systems have already been made explicit, which means that there is still a large unexplored research area to be covered by OROCOS.

5 Standards

Open standards (in programming languages, data formats, communication protocols, and documentation) are of paramount importance for efficient sharing of software and for its cooperative development. Only an Open Source project can guarantee the optimal use of, and respect for, open standards; commercial software producers are often tempted to introduce proprietary protocols to keep competitors out of the market.

Here is a list of some of the most important (and most powerful) standards that the OROCOS project will use:

- The CORBA IDL for object embedding, [14]. Through the IIOP (Internet Inter-Orb Protocol), all different implementations of CORBA are interoperable. CORBA also has a real-time extension, which is very important for robot control; TAO, [19] is an open source implementation.

- The structured document language XML [26] for configuration and data files.

- Modelica [12] for modeling of dynamical systems. Most of the commercial packages for the simulation of dynamical systems are co-developing and supporting this modeling format as their future open data format standard for the vendor-neutral exchange of models.

- DocBook [2] and LaTeX[9] for documentation. Both are platform-independent, Open Source, can be translated into many other formats, and as such they are the preferred text processing vehicles of the Linux Documentation Project too. DocBook is ideally suited for software manuals, while LaTeXis best for the advanced mathematics that occur everywhere in robotics.

The OROCOS does not want to impose a "standard" *programming language*. The philosophy behind this strategy is that the best tool for the job should be used. This can be C for real-time control, C++ for numerical algorithms, a scripting language such as Tcl/Tk for graphical interfaces or robot programs, etc. Anyway, the CORBA middleware is, by design, able to work in an heterogeneous programming language environment.

6 Business model

The previous Sections could give the false impression that only the robotics researchers in academia are to be interested in OROCOS. This is certainly not true, because all arguments behind proven business models in "classical" Open Source projects hold for OROCOS too:

- Robot service companies and robot users have less costs when they have to learn a "universal" programming paradigm and API. So, the investment threshold for OROCOS is lower than for proprietary alternatives.

- (Especially) small start-up businesses can focus on their core competencies (services or products) without having to invest enormous amounts of money in software.

- The above-mentioned companies (and many others that are mainly *users* of robot software) will pay developers to extend the existing code base with particular drivers, algorithms or features that they need for their own businesses. Through the open source license principle, this code will flow back into the major code tree, and can serve other people's needs too.

- The professional packaging and distribution of software is also a valid business model, as proven by the Red Hats, SuSEs and other players in the Linux market. Also professional documentation is a viable source of income.

- Companies that want to buy service contracts for their robots have more control: they are not "locked-in" by the proprietary software from one particular provider, and they can judge the quality of a given service company from looking at the code this company has already contributed to the project's code tree. In contrast to the commercial software world, Open Source developers are known by their individual names, because these occur in the copyright statements in the code or documentation.

The only "loss" resulting from Open Sourcing a large project is that no company will get tremendously rich from just writing and selling the software...

7 Supporting projects

OROCOS is very ambitious in its goals, but it can begin with an enormous head-start: almost all of its *Supporting Modules* (Sect. 3) are already available in other Open Source projects. Indeed, the re-use (of code, as well as of project management tools and experiences) is, almost by definition, the strong point of Open Source. Some (but certainly not all!) of the projects that OROCOS can built on are:

- *Octave*, [3], is a very rich numerical library, that uses the same foundations as, and in its scripting form is very compatible with, Matlab. It also contains ODE and DAE solvers, for the simulation of dynamical systems.

- *Real-Time Linux* [28] and/or *RTAI* [11], which are real-time kernel extensions to Linux; or the Linux-independent eCos [6].

- *Comedi*, [18], is a library for real-time device drivers, such as AD/DA cards.

- Various packages and open formats for 3D visualisation: VRML [25], Java3D [23], OpenGL with OpenInventor [20] or Coin3D [24], etc.

8 Organisation

Also for the practical organisation of a large software project such as OROCOS, one can profit maximally from the experiences and tools originating from other projects. In fact, a certain project management "Pattern" has arisen over the last couple of years, which describes the necessary components needed to successfully run an Open Source project: the project needs a small group of people that outline the strategic *vision*, and it needs a web page (www.orocos.org) for efficient *communication* with the core body of code developers. That web page carries a CVS code repository, one or more mailinglists (together with their archives!), a bug tracking tool (such as Bugzilla or GNATS), and lots of documentation (for example, in the form of an HOWTO guide, as is common practice in the Linux Documentation Project, [10]).

The major responsibilities of the project management team members are *not* in the first place writing code themselves, but rather: being responsive to contributing users; making their valid contributions to the code, the webpage, the documentation, etc., appear on the web page as soon as possible; keeping a low profile in (inevitable) "flame wars" on the mailing list while reminding participants of the common long-term goals; and being pro-active but moderate in the "advocacy" efforts.

9 Conclusions

Time has come to start with an Open Source robot control software project: the current commercial culture in robotics is still to "lock-in" users to proprietary software that uses no open software standards to work with code from other sources; the academic (and other!) robotics research community offers enough skilled users, not only to reach the critical mass needed for successful code writing, but also to provide long-term and long-reach design vision; there are many existing Open Source projects on which to build; and, last but not least, the concepts and practices behind running an Open Source project have been proven many times already, with, among many others, the Linux and Apache developments as prime examples.

This paper has introduced and motivated the long-term strategy behind OROCOS, the Open RObot COntrol Software project. This strategy encompasses innovative research subjects, as well as advanced software engineering goals.

References

[1] Patterns home page. http://hillside.net/patterns/.

[2] DocBook Technical Committee. Docbook. http://www.oasis-open.org/docbook/.

[3] J. W. Eaton. GNU Octave. http://www.che.wisc.edu/octave/.

[4] W. Emmerich. *Engineering distributed objects*. Wiley Chichester, 2000.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, Reading, MA, 1995.

[6] R. Hat. Embedded Configurable Operating System. http://sources.redhat.com/ecos/.

[7] iRobot. http://www.irobot.com/ir/index.htm.

[8] K-Team. http://www.k-team.com/.

[9] L. Lamport. LaTeX: A document preparation system. http://www.latex-project.org/.

[10] D. S. Lawyer. Linux documentation project. http://www.linuxdoc.org/.

[11] P. Mantegazza. RTAI: the Real-Time Applications Interface. http://server.aero.polimi.it/projects/rtai/.

[12] Modelica Association. Modelica: Language design for multi-domain modeling. http://www.modelica.org/.

[13] Nomadic Technologies. http://www.robots.com/.

[14] Open Management Group. CORBA: Common Object Request Broker Architecture. http://www.corba.org/.

[15] OpenSource.org. The open source page. http://www.opensource.org/.

[16] Real World Interface. http://www.irobot.com/rwi/.

[17] RoboCup. Soccer server. http://ci.etl.go.jp/~noda/soccer/server/.

[18] D. Schleef. Comedi: Linux control and measurement device interface. http://stm.lbl.gov/comedi/.

[19] D. C. Schmidt. TAO, The Ace Orb. http://www.cs.wustl.edu/~schmidt/TAO.html.

[20] SGI. Open inventor. http://oss.sgi.com/projects/inventor/.

[21] R. M. Stallman. GNU Public Licence. http://www.fsf.org/copyleft/gpl.html.

[22] R. M. Stallman. Why you shouldn't use the Library GPL for your next library. http://www.fsf.org/philosophy/why-not-lgpl.html.

[23] Sun Microsystems. Java3D. http://java.sun.com/products/java-media/3D/.

[24] Systems in Motion AS. Coin3D. http://www.coin3d.org/.

[25] The Web3D Consortium. The VRML repository. http://www.web3d.org/vrml/vrml.htm.

[26] W3C. XML: Extensible Markup Language. http://www.w3.org/XML/.

[27] Webots. http://www.cyberbotics.com/.

[28] V. Yodaiken and M. Barabanov. Real-time linux. http://www.rtlinux.org.