SPQR-RDK: a modular framework for programming mobile robots

Alessandro Farinelli, Giorgio Grisetti, Luca Iocchi

Dipartimento di Informatica e Sistemistica
Università "La Sapienza", Rome, Italy
Via Salaria 113 00198 Rome Italy
E-mail: <lastname>@dis.uniroma1.it

Abstract. This article describes a software development toolkit for programming mobile robots, that has been used on different platforms and for different robotic application. In this paper we address design choices, implementation issues and results in the realization of our robot programming environment, that has been devised and built from many people since 1998. We believe that the proposed framework is extremely useful not only for experienced robotic software developers, but also for students approaching robotic research projects.

1 Introduction

Research on developing autonomous agents, and in particular mobile robots, has been carried out within the field of Artificial Intelligence and Robotics from many different perspectives and for several different kinds of applications, and the development of robotic applications is receiving increasing attention in many laboratories. Moreover, robotic competitions (e.g. AAAI contexts, RoboCup, etc.) have encouraged researchers to develop effective robotic systems with a predefined goal (e.g. playing soccer, searching victims in a disaster scenario, etc.). These robots have been obviously used not only for these competitions, but also for experimenting the research techniques developed within robotic research projects. Moreover, mobile robots are also used for teaching purposes within computer science laboratories and often students are required to work and develop robotic applications on them[1].

This increasing population of robots in the research laboratories and the consequent need for developing robotic applications have started a process of design and implementation of robotic software, that aims in a special way at having a design methodology and a software engineering approach in the development of such applications, which integrates several functionalities and architectural choices that go beyond the scope of conventional robotic applications.

Furthermore, companies producing and selling mobile robots make available to their users development libraries and software tools for building and debugging robotic applications (e.g. Saphira for Pioneer robots [4], OPEN-R SDK for

[1] e.g. CMRoboBits Course at CMU http://www.andrew.cmu.edu/course/15-491/.

Sony AIBO [5], etc.). These tools are obviously platform dependent and thus they cannot easily be used for building multi-platform robotic systems, and also they usually lack some features that are required from a general purpose robot development toolkit. For instance, the OPEN-R SDK completely lacks facilities for remote monitoring the behavior of the robot. It just support wireless network communication among processes and all the remote information exchange must be explicitly coded. On the contrary, the Saphira environment, although it is specifically implemented for the Pioneer robots, has several facilities for building robotic applications and debugging them also by using a Pioneer simulator and allowing for a graphical display of the robot status.

Finally, a number of open source multi-platform robotic development environments have been realized. For example, OROCOS (Open RObot COntrol Software)[2] is an European project that has recently started with the objective of realizing a framework for developing robot control software under Real Time Linux. This project has many general goals, like independence to architectures used for connecting the components together, to robot platforms, to robotic devices, to computer platforms. The OROCOS project has a long time target and it is currently under development. *Player/Stage* [2] is also a general framework for controlling a robotic system. *Player* supports a wide range of devices, algorithms and viewers, that can be tested through *Stage*, a simulator able to work on complex multi robot scenarios. Each of these devices can be either a server or a client, allowing for a great flexibility in spreading the computation on different machines. However, *Player/Stage* provides only limited support for high level specification of user-defined modules and their interaction. *CARMEN*[3] comprises a set of independent utilities, that communicate with each other through the UNIX inter process communication facilities. This framework has been used for implementing a set of interesting algorithms, but it is mainly suited with the low level activities of the robots (such as navigation and exploration). Also the works in [7, 8] are focused on proposing robot middle-ware that are not specific to a given platform or to a particular application domain. In particular, the system presented in [8] is explicitly focused on the realization of soccer applications, while in [7] mostly low level interface issues are addressed.

In this paper we describe a Robot Development Toolkit (RDK) for modular programming of mobile robots. The toolkit we have realized includes a middle-ware that implements all the basic requirements for the development of a typical robotic application, a set of modules implementing the basic functionalities of the robot, and a set of tools that are useful for developing, monitoring and debugging the entire application. In particular the middle-ware implements an infrastructure for: task management, interfacing with the robot hardware, representation of the status of the robot, remote monitoring and debugging.

Our development toolkit is currently named SPQR-RDK, and is available to be used by robotic programmers[4].

[2] Orocos project, `www.orocos.org`.
[3] Carmen project, `www-2.cs.cmu.edu/~carmen/`
[4] Available from `http://www.dis.uniroma1.it/~spqr/`.

We are currently using our framework for developing different kinds of robotic applications: i) RoboCup soccer [3] ii) RoboCup Rescue [6] iii) RoboCare [1] - a project for developing a multi robot system for assistance of elderly people in a health care house. The development of these applications has given us a real testbed for evaluating the proposed RDK and, by a comparison with the development of similar applications by using a different development environment (in particular, we refer to the robotic soccer application with Sony AIBO robots by using OPEN-R SDK), we have experimented the effectiveness of our toolkit.

2 Design Choices

During the development of our RDK, we have identified a set of fundamental functionalities and a set of software requirements needed for our framework.

As our applications have been developed through the years by different people which were able to work at the application only for a limited period of time, *modularity* and *re-usability* appear to be the main issues to address: the proper division of the code in independent modules exchanging data inside a clear framework ensures to have a coherent software generation, resulting in highly modular and re-usable code. *Efficiency* is also a primary requirement, the middle-ware needed for running the modules must have a minimum overhead with respect to the entire application. Moreover, the hardware computational capabilities must always be considered, posing strict constraints on the implementation choices for our middle-ware; therefore most of the design choices that we have done (e.g. language, operating system, shared memory for information exchange) are motivated by this requirement.

As for functionalities we have identified three main issues to be addressed: i) **Remote Inspection Capability** ii) **Information Sharing** iii) **Common Robot Hardware Interface**.

Remote Inspection is a fundamental functionality for every robotic application. The Remote Inspection mechanism, should allow the developers to use a general mechanism for remote inspecting the internal status of the application, with limited network bandwidth and with minimum computational overhead with respect to the normal execution of the robotic application.

Another important problem that we have faced during our past developments has been the exchange of data among modules. A basic use of shared memory, without any data access policy, is not satisfactory because the management of all the shared data in the program can become very complex. Similarly, the use of message exchanging typically arises the same problems and may also affect modularity of the system, when a module is implemented by including the details of other interacting modules. Therefore, an important functionality for the RDK is an **Information Sharing** mechanism providing a uniform interface and a policy for sharing data among modules.

When dealing with several different types of mobile bases and sensing devices the independence of the application from the low level details of platforms and devices becomes an important issue. Hence, the development of a **Robot**

Hardware Interface has been detected as another important functionality: a uniform interface has to be defined between robot devices and user modules, and hardware configuration is described in a configuration file.

3 Software Architecture and Implementation of the Middle-ware

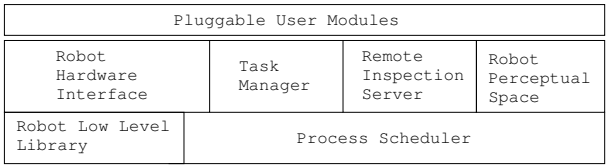

Fig. 1. Middle-ware Architecture Layered View

The RDK we are presenting in this article is based on a middle-ware that provides the basic functionalities for the development of robotic applications. This middle-ware is composed by a minimum set of modules, common to all the applications that can be developed within our framework. In particular, the middle-ware is made up by the following modules **Robot Hardware Interface**, **Task Manager**, **Robot Perceptual Space** and **Remote Inspection Server** as shown in Figure 1. In the following a description of each of those modules is given.

3.1 Robot Hardware Interface

The Robot Hardware Interface implements a level of abstraction with respect to the specific mobile base in use, providing the user with a common interface for accessing all the robotic platforms and devices. We decided to model this abstraction by exploiting the fact that usually each robotic platform comprises several sensors and actuators (devices), but only one mobile base. For robots and devices we implemented an abstract interface through a class hierarchy; in this way robots and devices of the same kind can be accessed through a common interface, and a user module can thus directly access the information and the services provided by a device, using the more general class needed. Moreover, by enforcing the abstraction on the robot hardware, it is possible to port all the written software on a new mobile base, simply by writing the low level interface.

The Robot Hardware Interface (RHI) module encapsulates the functionalities for accessing the mobile base and the on board devices and provides an abstraction for: i) *mobile robot kinematics*, by implementing the functions for reading odometry and for controlling the motion that are specific to a mobile

platform kinematics model (for example, distinguishing holonomic[5] mobile bases from unicycle-like[6] ones); ii) *mobile base connection*, by providing a standard way to access the mobile base and its specific control functions.

Each mobile base is generally equipped with various kinds of sensors and actuators like sonar rings, laser scanners, cameras, kickers (in the case of our soccer robots) and so on, that are generically defined as *Device*. These devices are connected to the robot and grouped in a set of hierarchical classes.

Both devices and robot drivers can be replaced by simulators or players of real data streams recorded before, allowing for off-line application development and debugging.

3.2 Task Manager

The Task Manager has been designed in order to allow the user to dynamically load his/her modules, to specify their execution features (i.e. execution period, scheduling policy, priority and so on) and to export the information to be shared among them.

A first feature of the Task Manager is to allow the users to easily define the scheduling policy of their modules by wrapping the Linux thread libraries.

Moreover, the Task Manager allows for the exchange of information among modules. When modules need to directly exchange information each other, the simplest solution is to couple them. However, this simple solution has the effect of limiting the software modularity and may results in cyclic references which are difficult to resolve in the linking phase.

Therefore, besides the mechanism of directly coupling two modules, the Task Manager offers another possibility to exchange information, by abstracting on the *type* of information. In fact, if a module needs data provided by some other module, it only needs to know *where* to read such data and *when* the data are available. On the other hand, a module that produces information can easily declare the kind of such information without knowing which user module will use it. This solution grants a complete independence among modules sharing data and it is possible to substitute a module with another, by only ensuring that the two modules produce the same kind of data. This mechanism has been used for sharing information among user modules, as well as between a device and a user module. Notice that the such mechanism requires the use of a shared memory thus limiting the spreading of computation on different machines. However, distributed robotic applications are currently not within the scope of our RDK and solutions explicitly designed for such applications are already provided in other programming frameworks (such as Times tool[7] or Charon[8]).

[5] An holonomic robot has three degrees of freedom in its motion.

[6] A unicycle robot has translational and rotational velocity bounded by a given kinematic law.

[7] http://www.timestool.com/

[8] http://www.cis.upenn.edu/mobies/charon/examples.html

3.3 Robot Perceptual Space

The Robot Perceptual Space (RPS) contains all the information known by the robot about the environment, and represents the current knowledge shared by all the modules in the application.

The RPS defines a uniform interface for accessing its data, thus similarly to the Task Manager provides a mean for information exchange. However, the semantics of the information contained in the RPS is different from the information shared through the Task Manager: RPS represents an updated snapshot of the robot perception of the environment, and the information contained in the RPS are specific to the robot application and thus generically useful for all the modules; the information exchanged by user modules through the Task Manager are instead parameters depending on the implementation of such modules and not on the characteristics of the environment.

3.4 Remote Inspection

The lessons learned from the past difficulties in debugging our software yielded to the design of a mechanism for remote control and debug that allows a module to generate and export information that can be received and displayed by a (graphical) remote client application (remote console). Information computed within the user modules are of different kinds and should be represented in different graphical forms: scanner readings, images, sonar readings, detected map features, position hypotheses, etc.

In facing the problem of building a debug interface, a key issue is to consider the high noise level and the latencies imposed by current wireless networks, therefore particular care has to be given in keeping low the bandwidth requirements. According to this consideration it has been designed a sharing mechanism that allows for a flexible run time selection of the information to inspect, in fact avoiding the differentiation of a release version from a debug one. We have thus chosen to implement a publish/subscribe mechanism for debugging information, in order to allow the user for selectively monitor the data of interest.

The Remote Inspection Server (RIS) defined in our middle-ware exports facilities for publishing information that can be monitored by remote clients. The publishing mechanism comprises two steps: The first one is *refresh*, where the RIS copies the information requested by at least one client in a local buffer. The second one is *transmission*, where the RIS performs the transmission of the buffered information to the clients. In this way network latency only affects the communication of the information to the remote host and not the efficiency of the publishing module on the robot. During the normal operation, when it is not needed to monitor the robot behavior in such a deep way, and clients do not request information to the robot, there is no overhead at all, since the Remote Inspection Server detects this situation and avoids useless computation.

4 Developing a robotic application: pluggable modules and supervisor tools

The development of a robotic application requires the realization of a set of modules implementing specific functionalities that must be appropriately connected together. The middleware realized for our RDK is suitable both for the realization and connection of the application modules.

It is interesting to notice that the composition of an application, in terms of which modules are activated and how they are connected, is simply described in a configuration file. Moreover, once we are satisfied with the robotic application in the virtual environment, our framework allows for an easy interchangeability of modules simulating the behavior of some sensors with actual sensor data interpretation modules, in order to make the application work on a real robot.

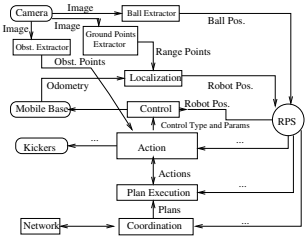

Fig. 2. Real robotic soccer application

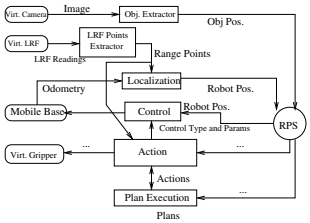

Fig. 3. Virtual cleaning application

As an example in figure 2 and 3 two instances of robotic applications obtained by implementing and connecting modules developed within our framework are shown.

5 Conclusions

In this paper we have presented a framework (SPQR-RDK) for developing modular multi-platform robotic applications, that has been designed for providing modularity, effectiveness and efficiency. This RDK allows a group of programmers to design and implement the modules composing a multi-platform multi-robot application, having both remote control and remote debugging capabilities, with a very small effort, by using a software engineering approach and by focusing on the semantics of the information exchanged among the modules. The main use of our framework is for people (mainly students) that want to develop a solution for a single topic or for a specific application (e.g. localization in an office-like environment, path planning with moving obstacles, multi robot coordination in a soccer domain, etc.), by using available modules for all the other capabilities of the robot. Our RDK provides these programmers with

an easy methodological tool for implementing the robotic application and also it allows for easily evaluating the specific application developed under different environment conditions and in comparison with different solutions.

The presented RDK has several advantages with respect to other robotic development libraries distributed by robot producing companies (e.g. Saphira [4], OPEN-R SDK [5], etc.), since it has been specifically designed for multi-platform applications. Furthermore, differently from other general-purpose robotic development tools, like the works in [7, 8] or the tools CARMEN and Player-Stage, our RDK provides in an integrated framework some important facilities, such as easy and efficient implementation of modular solutions to a specific robotic problem, remote control and inspection, information sharing, abstraction with respect to the mobile base and the connected devices, and a set of useful tools for developing typical robotic applications.

The SPQR-RDK is continuously increasing in the number of modules that are realized for the different applications that are currently under development within our group, but always maintaining the same middle-ware. This is an important achievement for our group since having several modules that can be combined for building different robotic applications with a small effort, allows for developing different solutions to common robotic problems and to evaluate them in several scenarios and in general to increase over time the quality and the effectiveness of the robotic applications developed.

References

1. S. Bahadori, A. Cesta, G. Grisetti, L. Iocchi, R. Leone, D. Nardi, D. Oddi, F. Pecora, and R. Rasconi. Robocare: an integrated robotic system for the domestic care of the elderly. In *In Proceedings of Workshop on Ambient Intelligence AI*IA-03*, Pisa, Italy, 1995.
2. B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *In Proc. of the Int. Conf. on Advanced Robotics (ICAR 2003)*, pages pp. 317–323, Coimbra, Portugal, June 30 - July 3 2003.
3. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: A challenge problem for ai and robotics. In *Lecture Note in Artificial Intelligence*, volume 1395, pages 1–19, 1998.
4. K. Konolige, K.L. Myers, E.H. Ruspini, and A. Saffiotti. The Saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1):215–235, 1997.
5. Sony. Open-r sdk, `http://www.jp.aibo.com/openr/`.
6. S. Tadokoro and et al. The robocup rescue project: a multiagent approach to the disaster mitigation problem. *IEEE International Conference on Robotics and Automation (ICRA00), San Francisco*, 2000.
7. H. Utz, S. Sablatng, S. Enderle, and G. K. Kraetzschmar. Miro - middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, 18(4):493–497, 2002.
8. Hui Wang, Han Wang, C. Wang, and W. Y. C. Soh. Multi-platform soccer robot development system. In *RoboCup 2001: Robot Soccer World Cup V*, pages 471–476, 2001.