Advanced Engineering Informatics 24 (2010) 62–75

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Programming multirobot applications using the ThinkingCap-II Java framework

H. Martínez-Barberá [a,*], D. Herrero-Pérez [a,b]

[a] Department of Information and Communications Engineering, University of Murcia, 30100 Murcia, Spain
[b] Department of Systems Engineering and Automation, University Carlos III of Madrid, 28911 Madrid, Spain

ARTICLE INFO

Article history:
Received 16 June 2009
Accepted 10 August 2009
Available online 26 September 2009

ABSTRACT

This paper presents a Java framework, ThinkingCap-II, for developing mobile multirobot applications, which has been successfully used in indoor, automotive and industrial robotics applications. It consists of a reference architecture that serves as a guide to make the functional decomposition of a robotics system, a software architecture that allows a uniform and reusable way of organising software components for robotics applications, and a communication infrastructure that allows software modules to communicate in a common way. A key aspect of this software architecture is that it allows code reusability by high level abstraction and a uniform way of accessing the characteristics of the sensors. In order to show the suitability of the framework, for both diverse complex platforms and multirobot applications, two case studies are discussed. One is an autonomous car-like vehicle which is guided by a manned vehicle, and the other an autonomous industrial vehicle which is member of a multirobot transportation system.

1. Introduction

The development of complex robotics applications involves diverse areas with different needs, such as data acquisition, signal processing, intelligent control, networking, etc. Large robotics projects involving development teams require efficient collaboration of their members. In the case of small or reduced development groups, leaving aside the economic factor, this becomes even more critical, because the development can be extended over non-bearable periods, and it is paramount to have fast prototyping and code reusability. To allow for these, the abstraction, organisation and design of the software components are mandatory. In addition, it is also necessary to produce working, robust and reliable applications.

The software aspects of this issue have not been discussed in depth by the robotics community [1], probably because it is traditionally a software engineering topic. Nevertheless, the need for standard specifications that deal with the recurrent concepts and requirements of the robot software development is certainly a key issue, which would allow robotics software components to be shared, distributed and/or reused. Thus, several recent papers address robotics software surveys, analysis and comparisons (for instance, see [2,3]).

Traditionally, robotics software has basically been the implementation of a functional architecture which was focused for a specific problem or set of problems. In these cases, the software was divided into modules depending on their functionalities (like *TCA* [4], *AuRA* [5], *3T* [6], *BERRA* [7] and *Saphira* [8]) with most of them obviating the transparency in communications, portability and code reusability. These architecture usually group the data acquisition, the real-time reactive processing, and the computation of actuators to perform certain actions in a single software module, according to the real-time constraints of these applications.

Robotics frameworks focused on low-level problems, on the other hand, aim to provide an abstraction of the robotics platform (like *Player/Stage*, *URBI*, *Open-R* and *OROCOS*), which facilitates the separation of robot control and real-time control of effectors, and then the reusability of software upon this abstraction. The level of abstraction usually depends on the complexity of the hardware platform for which they are designed. *Player* [9] is intended to command wheeled robots, proposing a mechanism for the data to flow between sensor, processors and actuators. *URBI* (*Universal Robotics Body Interface*) [10] is designed to command more generic platforms (like humanoids, animal-like and wheeled robots) and although complex commands can be written, its kernel is low level in essence. *Open-R* (*Open architecture for Robot entertainment*) [11] provides a common interface for object-oriented components to facilitate the modularisation, communication, portability and design, and a layered architecture for hardware adaptation, system services and applications. This framework has been used by quadruped and biped entertainment robots of the Sony Company. *OROCOS* (*Open RObot COntrol Software*) [12] proposes defining a generic robot by the specification of components which are completely decoupled of communications and, hence, control flow and data flow are established outside of components. This framework is oriented to provide the user with the necessary tools to assemble the global functionality of the robot depending on the functionalities needed.

* Corresponding author.
E-mail addresses: humberto@um.es (H. Martínez-Barberá), dherrero@um.es, dherrero@ing.uc3m.es (D. Herrero-Pérez).

1474-0346/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.aei.2009.08.011

Table 1
Summary of selected frameworks properties.

	Research	Industrial	Prototyping	Language	Data flow	Functional
Player	Widespread			C++/Clients		
URBI	Accademia	Limited	Excellent	URBI/Clients	Fixed	FSM
Open-R	RoboCup	Full		C++	Configurable	
OROCOS	EU Project	Full		C++	Configurable	
TeamBots	Limited		Excellent	Java	Fixed	Reactive
MissionLab	Military	Full	Adequate	C++	Fixed	AuRa

From the software point of view, it would be useful to define a generic robot and the separation of the problems to be addressed. Then it would be possible to build tools to allow for productivity in the robotics development cycle (like *MissionLab* [13], and *URBI* [10]).

In recent years there has also been an increasing trend in multi-platform support, either in the development of the whole software framework (i.e. *TeamBots* [14] written entirely in Java) or in allowing clients developed in other programming languages, which can be interpreted, scripted or compiled (i.e. both *Player*/*Stage* and *URBI* support remote clients written in different programming languages like Java, Python, etc).

When faced with the development of robotics applications for different domains, platforms, sensors and actuators using a reduced development team, as our research group does, **productivity** is of paramount importance. In addition, if any of the developments is to become a commercial or industrial product, **robustness** is also a must. We have summarised the important properties of selected frameworks regarding the productivity and robustness goals in Table 1. The *Research* column identifies the most relevant base users in the research community. The *Industrial* column identifies which frameworks are intended to be used for commercial or industrial applications, and what degree of industrial grade has been reached. The *Prototyping* column evaluates the simplicity or ease for fast prototyping, which is directly related to man-hours effort. The *Language* column shows the language for implementing and using the framework. In some cases, there are available clients for additional languages. The *Data Flow* column identifies if the data flow is fixed at compilation time or can be configured at runtime. The *Functional* column identifies which kind of functional architecture the frameworks are related to, if any. Blank fields mean that the column property is not applicable to the corresponding framework.

Because we are concerned with productivity, fast prototyping is a property that is considered more than necessary. *URBI*, *TeamBots* and *MissionLab* provide good support for this, but only *MissionLab* and *URBI* can be qualified as industrial grade. On the other hand, general usability is assured by *Player*/*Stage*, *Open-R* and *OROCOS*, but only at a platform level. In addition, Java multi-platform development has become a standard feature in business, with many productivity and development tools readily available. For this reason, we have developed a Java software framework for robotics applications that tries to keep productivity and robustness as its main goals, while adopting the many interesting features of the above-mentioned frameworks.

This paper presents a software framework for developing autonomous robots applications in diverse domains, like laboratory robots, automotive and industrial vehicles. The main goal of the framework is to allow a high productivity while obtaining robust code, which is suitable for commercial or industrial applications. Some important features of this framework are the modularisation of functionalities, i.e. implementation of algorithms independently of functional architecture, the flexible information exchange mechanism for communications, which facilitates the reconfiguration of the functional architecture and the integration of new skills, and the execution specified at runtime by configuration files, which allows to customise the modules that are instantiated, the functional architecture and the platform where it is running (hardware abstraction). This Java framework has been successfully used in different applications like laboratory robots, soccer-playing robots, industrial robots and autonomous vehicles.

The paper is organised as follows. The Section 2 describes the characteristics and design criteria of the software robotics framework, which is later used to control very different platforms. The Section 3 analyses and discusses the most important features of the proposed framework. The Section 4 describes two case studies: an autonomous car-like vehicle guided by a manned leader, and a team of industrial vehicles which are used to achieve transport tasks in a cooperative way. Finally, some conclusions are presented.

2. The ThinkingCap-II framework

ThinkingCap-II (TC-II) is a Java framework for developing mobile robot applications.[1] It is a joint effort between the University of Murcia, Spain, and the University of Örebro, Sweden, and it is based on previous work on *ThinkingCap* [15,16] and *BGA* [17] architectures. The framework consists of a reference cognitive architecture (largely based on *ThinkingCap*) that serves as a guide to make the functional decomposition of a robotics system, a software architecture (partially based on *BGA*) that allows a uniform and reusable way of organising software components for robotics applications, and a communication infrastructure that allows software modules to communicate in a common way, regardless of whether they are local or remote.

2.1. Functional architecture

Although the *TC-II* framework is functionally architecture-free, we have developed most of our applications, like the case studies described below, following a functional architecture based on *ThinkingCap* [16]. It consists of two-layer architecture (Fig. 1) for controlling mobile robots, one layer for reactive processes and the other for deliberative processes. It can be viewed as a stripped down instance of the *3T* architecture [6]. The blocks group the different functionalities present in typical mobile robotics systems (navigation, perception, control and planning), in which sensing and acting are a must. An important role is played by a centralised data structure called *Local Perceptual Space* (LPS), taken from the *Saphira* architecture [8]. It is a geometrically consistent robot centric space which consists of a collection of *Local Perceptual Objects* (LPOs). These LPOs model the local environment of the robot, and take into account the a priori information (map) and the currently perceived information (sensors) in a coherent way.

This architecture has been implemented and used in different types of robots and has shown good capabilities as an abstract guideline to organise the software which has to be run in an implementation of the abstract model of a module of the framework.

[1] Additional information can be found at http://robolab.inf.um.es/tc2.