A VISUAL AND PORTABLE TOOL FOR ROBOT PROGRAMMING

Juan A. Fernández, Javier González
System Engineering and Automation Department, University of Málaga
Campus Teatinos - Complejo Tecnológico
29071 Málaga (Spain)
e-mail: {jafma,jgonzalez}@ctima.uma.es

Abstract: Programming software for complex robotic systems requires powerful tools that allow the programmers to focus on the functionality of their application without getting lost in low-level details (communication software, operating systems, etc), while guaranteeing robustness, flexibility, and efficiency. In previous works a software integrating system called NEXUS was presented that achieves these characteristics. This paper presents a visual programming tool called BABEL that fulfills the NEXUS specifications and in addition exhibits multilanguage and multidistribution features. It decouples the OS, communication network, and hardware-dependent parts of the modules of the application from their intended functionality. Copyright © 2002 IFAC.

Keywords: Mobile Robots, Distributed Robotic Control Systems, Software Specification, Robot Programming, Programming Environments.

1. INTRODUCTION

Developing modern robotic applications involves the production of robust, efficient, and flexible (upgradable, maintainable, reusable) software. Although this aspect of a robotic application is seldom the focus of study, its influence in the productivity of a robotic research team is of great importance. Several issues arise in this field that affect the results of research in a robotic work:

- The integration of multidisciplinary work into a coherent, efficient application.
- The reuse of software in future research and in other robotic projects.
- Obtaining robust software even after integrating software modules from different programmers with different sensibilities and styles.

...
platforms, and still require a lot of programming not directly related to the robotic algorithms that are the core of research (for example, code for communicating the modules of the application, for accessing the special low-level operating system characteristic, etc.).

Other kind of tools for implementing robotic applications are based on control architectures that allow the researchers to configure sets of functional modules, communicating them and reuse them in future applications. Some known architectures are TCA (Simmons et al., 1990) and NASREM (Albus and Quintero, 1990). They run on existing operating systems. Their main drawbacks are their restricted models of control (for example, hierarchical or centralized).

A more evolved approach consists of software integration packages that let the programmer free for choosing any type of control architecture while integrating the software components in a comfortable fashion, usually providing tools for automatic generation and validation of code. NEXUS (Fernández and González, 1998) and G\textsuperscript{em}M (Fleury et al., 1997) can be included in this category. These systems are highly flexible and, specially NEXUS, designed for being portable and open (platform, OS, and vendor-independent). However, they still are rigid since they require to use specific programming languages and include non-changeable distribution packages in order to spread the application modules over a network of computers.

One next natural stage in the evolution of robotic programming tools seems to be the improvement of portability, distribution capability, and integration of very different programming languages and tools; the main objective is letting the researcher to focus on the robotic problem that is to be solved, and not on hardware or software dependent issues, such as communication between modules or generation of code for linking software written in different languages and operating systems.

This paper studies the need for a programming tool that takes a step in that direction. In this line, BABEL, a visual environment for assisting in the design and generation of the code of the modules of a robotic application, is introduced. It is prepared for abstracting the user from different programming languages, operating systems, and distribution packages. It provides the programmer with a visual environment focused on the design characteristics of each module, and minimizes the effort developed for low-level tasks that are out of the scope of the robotic research. In addition, BABEL is based on the NEXUS specifications for a robotic software application, which facilitates to obtain robust, efficient, and flexible code (Fernández and González, 1998).

The paper is organized as follows. Section 2 identifies the features that a robotic application should exhibit and how they are captured by the NEXUS specification. Section 3 analyzes the contributions of BABEL, and section 4 illustrates its use through the design and implementation of a simple robotic application for a mobile robot. Section 5 offers some conclusions of this work.

2. SPECIFICATIONS FOR PORTABLE, DISTRIBUTED ROBOTIC SOFTWARE

We have identified the following basic characteristics that a robotic software application should exhibit (Fernández and González, 1998; Fernández and González, 1999):

-Portability. Any program should be able to run on any platform with minor changes in its internal structure.

-Upgradeability and Reusability. Any part of the program should be able to extend its capabilities without affecting the current application, or to be substituted by other programs with a similar functionality.

-Interoperability and Distribution. The different software modules of an application should be able to exchange data, and to run in different machines from which each one obtains the maximum efficiency.

-Efficiency. The achievement of certain efficiency requirements should be guaranteed, i.e., real-time response. From a different perspective, the programming process should also be efficient in the sense that it should be performed in minimum time and with minimum effort.

-Robustness. Current robotic applications require a high degree of robustness, since they are often used in real, critical situations.

An integration software system called NEXUS has been presented that provides all these characteristics (for more detail in NEXUS specifications and how NEXUS is implemented, see (Fernández and González, 1999)). It has been designed for reducing both the cost of development of, as well as further modifications to, complex robotic software applications. In short, NEXUS is a system that combine all the software modules of a robotic application on a possibly distributed hardware system.

In NEXUS, an application is defined as an integration of different software modules that provide services to each other, aimed at accomplishing a given robotic task. The modules hide their implementation details: only the services and the format of their input and output parameters are known by other modules. This basic scheme guarantees many of the characteristics needed in an open system: portability, since the modules access NEXUS and the OS through general and simple interfaces that can be adapted to very different hardware platforms and operating systems; upgradeability, since a change in any module
that does not change its functionality (its set of services) does not affect the rest of the application; and reusability, since any module can be plugged into another application that needs its functionality without depending on its implementation details. The following are the main specifications of NEXUS that have been followed in BABEL.

2.1 Services

Each service of a module is defined through four parameters: its input data, its output data, its characteristics, and its priority with respect to other services of the same module (the implementation of this latter feature depends on the distribution package being used, since it is in charge of launching the routines for serving each service request).

The characteristics of a service specify:

-Its functionality (whether it is a service for consulting (query), updating (modifier), or monitoring (monitor) the module status)

-Its duration (whether the requests last until the architecture is deactivated (permanent), or not (temporary); whether the service has to be requested (non-monitor) or is launched automatically at module starting (monitor)).

-Its concurrency (whether several requests can be served at the same time or not (non-reentrant or non-reentrant)).

Any service contains a subset of these characteristics. This allows the programmers to implement a wide range of services, for example those that run periodically, or those that execute only when a given asynchronous signal is received.

2.2 ICE modules

ICE modules, or simply modules, are the main components of the applications. Each module is executed on a single computer, and therefore all the services that it provides are served from that machine. The possibility of distributing the ICE modules among the computers of the robotic system permits optimal exploitation of the available hardware resources. For example, one computer in the system may be more appropriate for image processing than others, so the ICE modules which offer services related to visual perception should execute in that machine.

An ICE module consists of three parts:

-Interface: The interface part of the module defines the services implemented in the module and provides the rest of the application with the public information about the functionality of the module.

-Code: This part contains the code of the routines that supply services and two special routines for both initializing and finishing the internal status of the module (internal static variables, configuration of the module, etc.).

-Error-recovery code. This part deals with the errors that the module may find during its execution. Some of them can be overcome by the module, while others must be propagated to the module services requesters. The mechanisms provided to deal with errors are described elsewhere (Fernández and González, 1999).

Each ICE module can be assigned a priority relative to the other modules of the application. The implementation of this feature depends on the operating systems on which the module is executed.

2.3 Communications

The distribution package is in charge of carrying information among the different components of the robot system. If the source and destination modules of an information package are not being executed on the same computer, the communication is routed to the destination machine, and then it is directly sent to the suitable module.

Two types of communications are distinguished in the NEXUS specifications: messages, that contain an arbitrary amount of data (for example, requests to any component of the system, or replies to these requests) and events, that do not contain any information and are sent to all the modules of the application at the same time. With these basic types of communications, both synchronous and asynchronous concurrent processing can be implemented. Messages are used to request and reply service requests, while events are used for asynchronous operation. For example, if there is a module called "Battery Sentinel" offering a monitor service which supervises the charge level of the batteries of a mobile robot, an event can be defined for informing the other modules about whether the charge falls below a certain level.

3. BABEL: A TOOL FOR VISUAL ROBOTIC PROGRAMMING

This section presents a new tool, called BABEL, intended for the visual programming of robotic applications. It has been constructed over the NEXUS specifications, and therefore, it maps the above requirements (see section 2) to certain programming language, distribution package, and operating system (the so called "dependence triple"). In addition, it can generate code automatically and is as independent as possible from the election made on the dependence triple.

1 "ICE" stands for the three parts of a module in NEXUS: Interface, Code, and Error-recovery code.
Basically, BABEL is a tool for assisting in generating code for the components of the robotic application. This code can be produced for different programming languages and several communication packages (such as CORBA (Henning and Vinoski, 1999) or NEXUS). The main features of BABEL include:

- **Visual Programming.** It provides a user-friendly integrated development environment for designing the application, specifying the public interfaces of the modules and their services, and writing the code of the service routines (snapshots are shown in fig. 2).

- **Automatic Code Generation.** The programmer needs only to provide the formal specifications of his/her modules and the code of the service routines. The software for making up the modules as complete executable programs, and for integrating these program into the robotic application is automatically and transparently generated by BABEL.

- **Multi-Language.** BABEL is prepared for generating code for different programming languages. Currently it is ready for producing C++ and C code, and it is being extended with JAVA code. It also provides the possibility of linking with existing libraries, possibly written in different languages.

- **Multi-Distribution.** The programmer can also specify the communication package that will be utilized for communicating the modules of the application throughout the network. Currently, the ACE+TAO CORBA\(^2\) (Schmidt, 2000) and NEXUS implementations are admitted.

- **Multi-Platform.** The OS-dependent software (multithreading, console output, disk management, timing, etc) is accessible through the use of BABEL macros that are independent on the platform being used. These macros are translated into the appropriate language constructs when the code for the module is generated.

As previously commented, BABEL follows the NEXUS specifications, thus allowing the programmer to implement robust, flexible, and efficient robotic software. Moreover, the programmer does not need to write any code referring the underlying OS, the distribution package, or the internal management of incoming or outcoming service requests: he/she only focus on programming the routine of each service.

This highly improves the productivity and allows multidisciplinary researchers to fruitfully participate in complex robotic projects. In the following section, an example of the use of BABEL is described.

### 4. PROGRAMMING WITH BABEL

In order to illustrate the suitability of BABEL for designing and generating robotic applications, an example is described which is a simplified version of an application we are implementing for our mobile robots RAM-2 and SENA. An overall scheme of the application is shown in fig. 1. The task consists of moving the robot following a given geometric trajectory. For accomplishing that, four important modules are integrated: the Motor Steering module is in charge of setting and reading the velocity and curvature of the robot, the Laser Scanner module reads the measurements from a frontal laser that yields distances to obstacles in the environment, the Trajectory Generation module generates a geometric trajectory on a given map of the environment, and the Trajectory Tracking module drives the robot along that trajectory, stopping it if there is any unexpected obstacle.

![Fig. 1. Scheme of the software application that allows a mobile robot to follow a trajectory in its environment.](image)

The following screenshots show how the Motor Steering module is designed and programmed in BABEL. Fig. 2a shows the main features of the module. A priority can be given to each module in order to set some real-time constraints between the modules of an application\(^3\). The code generation parameters allows the programmer to choose the distribution package (generic CORBA, ACE+TAO CORBA, NEXUS), the programming language (C++, C), and the compiler that will be used for building the executable file corresponding to the module (MS Visual C++, GNU gcc). On the left of the BABEL screen several buttons serve to open text editors for entering the startup and ending code of the module (the ones executed when the module is started and finished, respectively, see section 2.2), plus the code for defining global structures and including external programming objects.

---

\(^2\) ACE+TAO CORBA is a real-time implementation of CORBA.

\(^3\) As commented in section 2.2.
Fig. 2b shows the groups of services created for this module: the Internal services group include a monitor that recovers data from the hardware of the robot; the Modifying services group include services for changing the internal status of the module, that is, for setting the speed and curvature of the robot; the Reading services group are for consulting the internal status of the robot. In fig. 2b, the monitor service “Motor monitor” is deployed, showing that it is a regular service (it could also be a handler for receiving inter-module real-time events, also defined in the NEXUS specification -see section 2.3-), and its characteristics are monitor and permanent, which implies that there is no input or output data. On the left-down part of the screen the characteristics of the service can be set. They include specifications for services that modify the internal status of the module (modifier), monitor (monitor, permanent), services that can be requested more than once in parallel (reentrant), services that can be requested from web pages (web), services that open graphical interfaces (graphical), and services that run at the highest available priority.

Fig. 2. Example of programming of a robotic module. a) Top-Left: design of the principal characteristics of the module. b) Top-Right: groups of services provided by the module to other modules of the application, and design of a monitor service. c) Bottom-Left: design of the service for changing the robot’s speed. d) Bottom-Right: macro assistant for writing the code of one of the services.

---

4 The service is not requested by any other module, but launched automatically by this module at startup, and it does not return until the module terminates.
(prioritized). Also, a button opens a text editor for writing the code of the service routine.

Fig. 2c shows other service: the “Set Curvature” one. It has an input value, the new curvature for the robot.

As shown in figure 2c, the input and output data for a service of a module that uses CORBA distribution packages are written in IDL, the language for specification of data provided by CORBA (Henning and Vinoski, 1999). Fig. 2d illustrates how the code is edited for a service (in this example, the “Set Curvature” service).

The code of the service routines must be written in the language specified for the module (C++ in this case), and the platform-dependent constructs are entered by BABEL macros. These macros can be filled in using an assistant. The figure shows a menu with some of the macros implemented. In particular, the macro for requesting a service from other module is highlighted.

Once the code is provided for all the services, BABEL is ready for generating automatically all the code needed for building an executable file that can communicate with other modules via the selected distribution package (ACE+TAO in this case).

As shown in the figures, BABEL allows the programmer to focus almost exclusively on the functionality of his/her module, minimizing the amount of code to write. In addition, since the NEXUS specifications are met, the robustness and flexibility of the application are guaranteed.

3. CONCLUSIONS

In complex robotic projects, developing software that achieves certain characteristics (robustness, flexibility, efficiency) is an important issue. This paper has introduced a new tool for facilitating the design and implementation of robotic software applications that fulfills these requirements. This is due to some restrictions that it imposes on the software: modularly, what assures upgradability, flexibility, and robustness, standardization, what assures portability, and simplicity, which leads to an efficient design phase without losing all the possibilities that should be available to any robotic application. Some of these features are provided since BABEL implements the NEXUS specifications, which were designed with those objectives in mind. In addition, BABEL contributes with a visual programming environment that allows the programmer to focus on his/her algorithms, and with automatic code generation for different languages, operating systems, and communication packages.

REFERENCES


3rd International Symposium on Robotics and Manufacturing Burnaby, Canada.


