Software Modularity for Mobile Robotic Applications

Eric Colon[1,2], Hichem Sahli[2]
1 Unmanned Ground Vehicles Centre
Royal Military Academy
Avenue de la Renaissance 30
B-1000 Brussels, Belgium

2 Vrije Univeristeit Brussel
Electronics & Information Processing Dept.
VUB-ETRO, Pleinlaan, 2
B-1050 Brussels, Belgium

ABSTRACT

This paper reports on ongoing work in the specification and development of a modular software control framework for mobile robots. In order to allow flexibility, a systematic analysis of requirements has been conducted and their consequences on the framework architecture are reported in this paper. We also show that the communication requirements of the framework can be fulfilled by the free CORBA implementation library ACE_TAO. Finally, the concept of robotic control pattern is introduced and one of these patterns, the mobility pattern, is presented.

Keywords: software modularity, robotic control pattern, distributed control, control framework, CORBA

1 INTRODUCTION

Many researchers in robotics are confronted with the same problem: they have at their disposal many excellent algorithms but due to the lack of standard it is almost impossible to easily reuse those bricks into new applications. Existing programs have to be modified, translated, ported, or simply (!) completely rewritten from scratch when changing/updating the robotic platform. What is needed is a software framework that enables agile, flexible, dynamic composition of resources and permits their use in a variety of styles to match present and changing computing needs and platforms. Since a couple of years, some researchers have begun to work in this way.

At the Unmanned Ground Vehicle Centre (UGV-C), we are dealing with command and control applications including tele-monitoring, tele-operation (including shared, traded and supervised control), collaboration (between users), for single and multi-robots (of the same or different models). The present efforts are justified by the fact that most of the tools developed by the research community deals with autonomous robots (MCA, DCA, MIRO, GeNoM, ...). Note that we are not dealing with hard real-time control like the Orocos project (www.orocos.org). It addresses higher level components like planning and user interaction.

In order to clarify the text we give the following definitions:

A **framework** is a reusable, "semi-complete" application. It provides generic modules which generally need to be customised and extended in function of the application.

An **architecture** is an instance of the framework. It is composed by selected modules which are customised and completed by application specific modules.

After the presentation of the architecture requirements, a short overview of current distributed programming technologies is given. More specifically, we give an insight of the more and more popular ACE-TAO software library. The requirements are related to the capabilities of this library. Afterwards, robotic application patterns are introduced and their utility is discussed. The remainder of this paper describes the "mobility" control pattern.

2 FRAMEWORK REQUIREMENTS

The first step in any software development project is the collection of requirements from the intended user groups. In fact, we can distinguish different categories of users and according to their position in the development and utilisation process they have different needs. Lars Peterson distinguishes five groups (end user, application programmer, module programmer, interface programmer and hardware designer) and provides for each category their expectations of the framework [1].

We started from the review of typical tele-robotic applications [2] to extract the following characteristics:

- Integration of different robotic systems,
- Concurrent control of several robots,
- Universal GUI,
- Shared controlled between several users,
- Integration of user's tailored algorithms.

In this comparison we observed that no application possesses all the capabilities listed above. In the reminder of this section we detail these characteristics and we will see that they have consequences at different abstraction level of the framework.

- Integration of different robotic systems

The robot may be controlled by a computer or a microcontroller. In order to be able to connect and control existing robots, we need to be able to call functions of the robots' native libraries. These libraries are generally written in C or C++.
The framework should allow the use of native libraries directly or through wrappers.

In case of microcontrollers, we need to link them with a more capable computer in order to integrate the robot into the framework. This generally happens through serial ports. This means that we must be able to communicate through this kind of connection. New robots can be developed in such a way that they take into account the characteristics of the framework.

- Concurrent control of several robots

Many applications require the utilisation of several identical or different robots. The framework should allow this possibility in a natural (native) way. This necessitates the ability to write distributed and multi-threaded processes and consequently the possibility to easily communicate between and to synchronise processes.

- Universal GUI

The GUI should be as simple as possible for the user, and could ideally be used by a non-robotic engineer. The GUI must have three main characteristics: consistency (same appearance on different platforms), simplicity (easy to use easy to learn) and efficiency. The interface should enable the user to drive all interactions during the specification of the mission with simplicity and with flexibility, with no need for memorizing any commands, but it should guaranty the consistency of all dialogs. It should avoid the error possibilities, and, when this goal was not achieved, it should provide mechanism for error recovery.

The GUI needs a platform-independent language, in order to be compatible with any type of computer (UNIX, Linux, Windows or Mac). It should allow the display of: video, audio, text, 2D vector graphics, raster graphics, 3D and to be connectable to haptic interface.

Most of existing interfaces are very specific, and need to be re-compiled for any type of mission, especially when different robots or several robots are involved. The GUI should adapt automatically to the system capabilities. This requires the use of flexible programming language and implementation solution.

- Shared controlled between several users

In complex applications it is sometimes required to divide the workload between different people or between a user and autonomous agents. The framework should allow the access to several users at the same time and the distribution of tasks between different agents (software or users). This requires the management of user access and use policy, but also the coordination between different control modules.

- Integration of user algorithms.

It should be possible for local or remote users to customise the application by providing their own control algorithms. This will constitute one of the basic capabilities of the famework and requires run-time (remote) configuration capabilities. Furthermore, the added components should be able to run on different computer systems, requiring portability.

All these functionality requirements can be regrouped under the generic "flexibility" qualifier, which exhibits additional aspects: distribution, modularity, configurability, portability, scalability and maintainability.

Such a framework relies on distributed computing and hence on a very robust communication library. This aspect is examined in the next section.

3 DISTRIBUTED COMPUTING

Distributed computing models allow components to be spread across multiple computers to yield the benefits of increased scalability, better performance and varying degrees of component reuse. Distributed applications rely on network communications. Different techniques can be use to deploy such an application:

- Sockets,
- Middleware (DCOM, Corba, RPC, Java-RMI),
- Intelligent networks (Jini,...).

After a systematic comparison of distributed computing technologies (DCOM, CORBA, Java-RMI, Jini, ...), we have finally opted for the CORBA standard. The open source communication framework ACE_TAO (http://www.cs.wustl.edu/~schmidt/TAO.html) has been selected to develop our robotic control framework. TAO is a freely available, open-source, and standards-compliant real-time implementation of CORBA that provides efficient, predictable, and scalable quality of service (QoS) end-to-end. TAO is built on top of ACE (Adaptive Communication Library). Other robotic projects are also based on ACE_TAO: Miro (http://smart.informatik.uni-ulm.de/MIRO/) and Smartsoft for Orocos (www1.faw.uni-ulm.de/orocos/).

We examine hereafter what TAO provides to satisfy the control software requirements introduced in section 2:

- Integration of different robotic systems: use of native libraries (C/C++): ACE_TAO is written mainly in C++, which is the de facto language of most robot libraries.

- Concurrent control of several robots: distributed and multi-threaded processes, easy communication and process synchronisation. This is the reason of the existence of ACE.

- Universal GUI, flexible programming language and implementation solution: the choice of ACE_TAO does not restrict the GUI developments. GUI can be based on C++ toolkits or on interpreted language. Furthermore, CORBA communication is straightforward in Java.

- Shared control between several users requires management of access and use policy as well as coordination between control modules: this is not directly linked to the choice of ACE_TAO. However, the network capabilities could facilitate the implementation of such functions.

- Integration of user algorithms: requires run-time configuration capabilities, portability. By modifying the server registration data in the naming/trading services different servers (functionalities) can be selected at run-time.

- Flexibility:
  - Distribution: This is the core function of CORBA.
  - Modularity: by using OO programming and interfaces through IDL, capabilities can be divided among several components.

- Configurability: as explained above, the naming/trading could contribute to solve the configurability issues.
- Portability: ACE_TAO is an universal library that can be used on almost all platforms.
- Scalability: ACE_TAO provides many components tailored to applications involving many processes.
- Maintainability: ACE_TAO is based on many standard design-patterns. CORBA is an industry standard.

- Performance and efficiency: ACE_TAO has been developed for time-critical applications used in aviation and medicine. Many years of development and improvement have lead to very efficient software implementation. ACE_TAO provides many RT components (RT Event coms, RT-CORBA, ...).

4 APPLICATION PATTERNS

The framework architecture defines how the different components are integrated into the framework and how they are interrelated. Framework components are divided into:

- Structural components: these offer the basic services that will be used by other components (Name server, Time server, configurator, supervisor, ...).
- Application components: the building blocks of an application (navigation, vision, ...).

In order to identify reusable components, we introduce the robotic control patterns concept (a **pattern** is a proven design solution to a general problem). The robotic control patterns are classified from the simplest one to the most complex one:

- Direct control
- Monitoring
- Sensor data processing
- Direct tele-control
- Assisted tele-control
- Autonomous navigation
- Multi-robot coordination
- Multi-user cooperation

This classification should help us to identify basic components for each pattern and to make common components as generic as possible. Considering complex application patterns at design time allows integrating their requirements during the design of generic services. Due to the restricted paper length, we onlt present the simplest control pattern.

5 THE DIRECT CONTROL PATTERN

In this control pattern, we assume the user wants to remotely control a mechanical system that we call robot service. The system represented by the robot service can be a mobile robot, a manipulator or another mechanism capable of physical motion. The user controls this robot through a customized GUI which is downloaded as part of the service or by using an haptic interface like a 3D joystick or a master arm. The use of mice and keyboards are included in

the GUI interaction schema. We don't consider in this pattern visualization, is it part of the monitoring pattern. We assume the user has a visual contact with the robot.

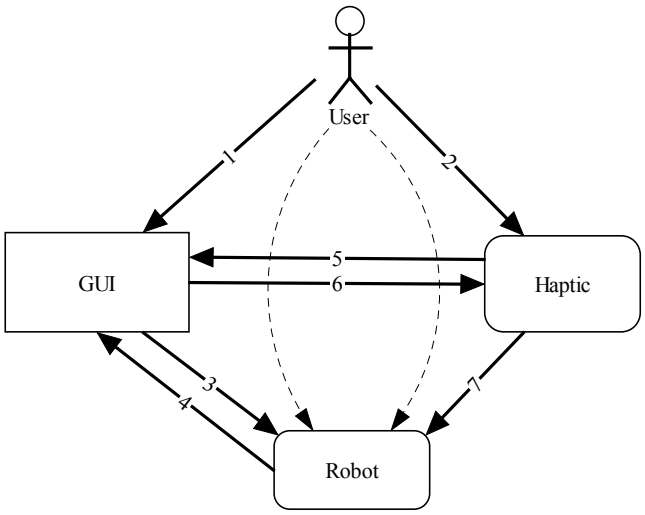

Fig. 1 Direct Control pattern

1. The user interacts with the GUI to configure the services
2. The user interacts with the Haptic device
3. Configuration and commands are sent to the robot service
4. The robot sends information about its status
5. The haptic device sends info about its status
6. Configuration commands are sent to the haptic device
7. Commands from the Haptic devices are sent to the robot

Dash lines represent virtual control paths.

It is obvious that some kind of mapping between user commands and robot commands must exist. For example, the motion speed is generally made proportional to the user inputs. User inputs can be directly transformed into speed commands for the different degrees or freedom or be combined in some ways for complex motions. This mapping could be performed either by a module included in the GUI or in the haptic service (downloaded at run-time as a proxy for the robot service) or by the robot service itself.

An important part of this pattern concerns mobility. In the next figure, the control is divided in different modules that act as filters. They are explained below.

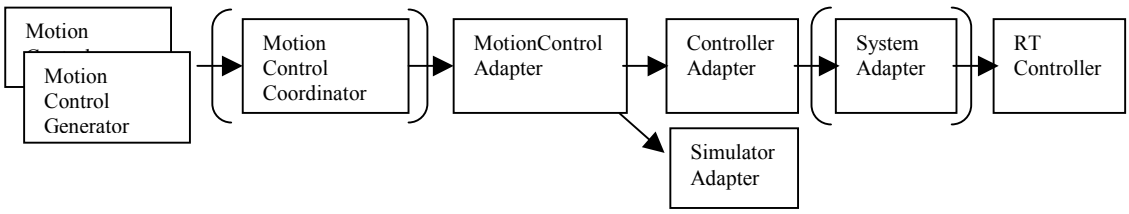

Fig. 2 Mobility modules

Motion Control Generator (MCG)
Function: generate kinematical command values
Input: keyboard, file, joystick, GUI, process (trajectory generator)
Output: normalised or raw kinematical commands

Motion Control Coordinator (MCC)
Function: fuses inputs coming from different Motion Control Generator (subsumption, fusion, sharing)
Input: Motion commands from (possibly) different Motion Control Generator
Output: Coordinated Motion commands

Motion Control Adapter (MCA)
Function: It transforms kinematics commands into motion commands taking into account the kinematics model of the system
Input: high level order coming from a MCG (directly or through a MCC)
Output: motion commands sent to the Controller Adapter (via network or Shared Memory)

Controller Adapter (CA)
Function: This component is an adapter between the framework and the real controller. It is specific for each controlled system.
Input: motion commands from the Motion Control Adapter
Output: actuator commands sent to the RT controller

System Adapter (SA)
Function: This component is used to abstract hardware components.
Input: actuator commands
Output: Hardware related

Simulator Adapter (SimA)
Function: system behaviour visualisation before real use.
Input: actuator commands
Output: graphics, data files,...

This is obviously a functional decomposition. The component granularity must be further evaluated and some functionalities could be aggregated into components in order to reduce the network load. For instance we could use a component that manages the serial port and that uses a queue if shared access is required. Furthermore, it should be transparent to other components. Modularity can also be reached at the programming libraries level using Object Oriented techniques

6 PRELIMINARY RESULTS

A serial server has been developed using the TAO library. A client reads console inputs and sends corresponding commands to the RMASerial server. This communicates through the serial port to another program simulating a microcontroller. It reads translation and rotation inputs and adapts these values to the Hunter kinematics. The right and left speeds values preceded by the command code are sent to the RMASerial server. In this first demonstration

application, the Motion Control Generator and the Motion Control Adaptor have been grouped. In a real application, it should be better to split those modules in order to increase flexibility.

It has been verified that the RMASerial server can process function calls and transfer data to/from the serial port concurrently. TAO provides several concurrency models. In our tests the default configuration has been used, it is a single-threaded, reactive model. One thread handles requests from multiple clients via a single Reactor [3]. It is appropriate when the requests take a fixed, relatively uniform amount of time and are largely compute bound. This is the case in this application where the response of the remote serial client program is immediate and is determined by a 50 ms timer. The single thread processes all connection requests and CORBA messages. Application servants need not be concerned with synchronizing their interactions since there is only one thread active with this model. [4]

7 CONCLUSION

Developing modular control software requires a systematic and detailed analysis of the applications requirements. The definition and description of robotic application patterns allows us to identify common components and to take into account complex requirements. Furthermore, adopting programming standards like CORBA and the choice of open-source software is the only way, according to us, to reach this software modularity. CORBA has a steep learning curve but offers many benefits for writing heavy distributed applications. Off course much of the work is still to be done and we need to develop real and usefull modules in order to prove the faisability of this framework design.

REFERENCES

[1] *A Framework for integration of Processes in Autonomous Systems*, Lars Peterson, Doctoral dissertation, 2002, University of Stockholm, Sweden

[2] *Telematics Applications in Automation and Robotics - TA2001*, July 2001, Weingarten, Germany

[3] *Pattern Languages of Program Design*, Jim Coplien and Douglas C. Schmidt, Addison-Westley, 1995, ISBN 0-201-6073-4

[4] Configuring TAO's components, Douglas C. Schmidt, http://www.cs.wustl.edu/~schmidt/ACE_wrappers/TAO/docs/configurations.html.