Software Design Patterns for Robotics : Solving integration problems with MARIE

Carle Côté, Dominic Létourneau, François Michaud and Yannick Brosseau
Department of Electrical Engineering and Computer Engineering
Université de Sherbrooke
Sherbrooke (Québec) CANADA, J1K 2R1
{*Carle.Cote, Dominic.Letourneau, Francois.Michaud*}*@USherbrooke.ca*

I. Introduction

The robotics field is evolving quickly. The diversity of technologies available and the software solutions developed in the various research centers throughout the world is in fact increasing the complexity of integration. Rapid and simple use of heteroclite components developed in different projects is essential for efficient progression of the autonomous robotics field. Just like sharing experimental results through scientific publications, sharing code should be part of the scientific process in robotics, allowing other to benefit from the work without having to reimplement everything. Too often not using available software components is caused by the inherent complexity to integrate different incompatible programs because they were designed for specific platforms, operating systems or communication protocols and conventions.

MARIE (Mobile and Autonomous Robotics Integration Environment) is designed and developed with these particular considerations in mind. This paper presents three design patterns applied in MARIE that helps resolving recurrent integration problems observed in robotics software development.

II. Open Integration Environment

A. Context

Building robotic applications is often complex and generally requires a lot of knowledge in various fields such as probabilistic navigation algorithms, simultaneous localization and mapping, planning, speech recognition, audio and vision processing, etc. Being able to reuse already available solutions is one key to accelerate developments and to avoid having to become expert in each of those fields. This implies that software components integration is critical and should be addressed at the very beginning of the development process.

Unfortunately, components integration gets more complex as many developers are contributing software, each having different software development skills, goals and interests. Most of these developers are considered valuable resources with specialized knowledge and limited time. Being able to minimize their integration efforts to reuse their work becomes critical.

B. Problem

Three important constraints characterize software development in robotics :

- Robotic components are typically developed independently, considering their own set of requirements (e.g. timing, communication protocol, programming language, operating system, etc.), their own integration environment and their own specific objectives and applications. Those components are often available "as is", with not much support from their developers and with minimal documentation. Reusability in this context is difficult to obtain, although it is considered crucial for the evolution of the field.
- Evaluating characteristics of each individual component has not been shown sufficient to predict how they will behave in a robotic system. Being able to work with complete robotic systems at all time becomes a critical issue in development process.
- Developers play different roles in the development process, and each role has its own view of the system determined by the abstraction level of their contributions. Here are descriptions of five typical roles :
  - Core Architect programs low-level functionalities related to operating system issues (I/O, threads, process, communications, etc.).
  - Framework Architect designs and develops framework architectures to support domain specific concepts for upper levels of abstraction.
  - Component Integrator implements and integrates new components in the integration environment based on available frameworks.
  - Application Integrator creates applications based on available components.
  - Tester tests and calibrates the developed applications.

Being able to cope with these constraints and their related concepts is an important issue to accelerate robotic software developments.

C. Forces

- Reuse and exploit already available software capabilities for robotic control and decision-making processes, coming from different fields.
- Accelerate software developments.

- Increase interoperability between existing softwares and programming environments.
- Reduce the complexity of creating complete robotic systems.
- Minimize developer's knowledge and expertise required to use complete robotic systems.
- Minimize developer's integration time and efforts.
- Share implementations of robotic systems to accelerate new developments.

D. Solution

Developing a software integration environment is a well known approach to solve integration issues. The concept consists of creating a standardized framework to integrate existing or new components together. This approach inspires MARIE's design by decentralizing integration management and having developers integrate their work and reuse other's work through the integration environment, without being aware of each other.

In order to support different developer's roles, MARIE follows a layered architecture to build the integration environment :

- The Core Abstraction layer consists of tools for communications, data handling and operating systems issues. It is an open architecture that does not specifically determine how to build robotic systems.
- The Components Creation and Management layer, built on top of the Core Abstraction layer, specifies and implements useful frameworks to add new components and to support domain specific concepts.
- The Application Builder Abstraction layer consists of integration tools to build applications using available components, without knowing the exact nature of the component's programming.

Using many abstraction layers reduce the knowledge, the expertise and the time required to use the overall system, by letting each developer choose which abstraction layer is more appropriate for its contribution. The Application Builder Abstraction layer is designed for high-level users with minimal knowledge in robotics systems, but also for specialized developers to rapidly integrate, test and validate their own components in a complete robotic system.

III. Distributed Mediator

A. Context

To make progress in various fields of activities in robotics, software development needs to integrate more and more components coming from different fields, and adapt them to different application contexts. Reusability, extensibility, adaptability, modularity and interoperability are therefore important software attributes when designing new software components and facilitate their integration.

B. Problem

One approach to obtain such software attributes when designing components is to use standards for frameworks, interfaces and data representation. Unfortunately, current developments in robotic are not based on those kind of standards as they are still being developed and adopted by the community. Each component is created considering its own development context, i.e. using different programming languages, operating systems, data representations, communication interfaces, communication protocols, etc. Integration of such heterogeneous components is often difficult and need to be simplified.

C. Forces

- Reuse and exploit already available libraries and programming environments.
- Accelerate developments.
- Increase reusability, extensibility, adaptability, modularity and interoperability of heterogeneous components.
- Overcome the lack of standards to develop applications based on heterogeneous components.
- Avoid re-implementing available components in integration frameworks.
- Exploit the diversity of communication protocols and mechanisms to benefit from their advantages and maximize their usage.

D. Solution

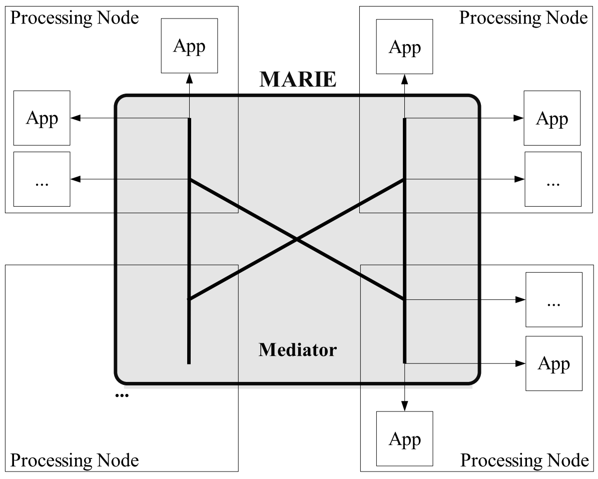

Fig. 1. MARIE's adaptation of the mediator pattern for distributed system.

MARIE proposes a different approach to overcome the lack of standards in robotic sofware systems by adopting the Mediator Design Pattern model [1] in order to create distributed applications using heterogeneous components. The Mediator Design Pattern primarily creates a centralized control unit (named mediator) interacting with each colleague (components/applications) independently, and coordinates global interactions between colleagues to realize the desired system. MARIE's adaptation of the Mediator Design Pattern is illustrated in Fig. 1. It decouples components functionalities by interfacing them using communication links managed by the mediator. Detailed description of the solution can be found in [2] and detailled implementation description can be found at http://marie.sourceforge.net.

IV. Abstracting Communication Protocols from Component Functionalities

A. Context

Creating a distributed application typically signifies that each component interacting within a software application has at least one communication protocol in common. In the actual robotic software development context, there is no unified protocol available, and no consensus has emerged yet from the robotic software community.

B. Problem

Having to support a wide range of communication protocols is often perceived as a difficult task knowing that each communication protocol has its own set of requirements and specificities that need knowledge and expertise to be handled appropriately. Since software component developers are often specialized in specific robotic fields and are not necessarily communication protocol experts, this typically leads to the situation where only one protocol is supported, reducing components interoperability and reusability. This shows how important it is to decouple communication protocols management from specialized functionalities, in order to let component developers focus on their contribution without being aware of the details of communication protocols.

C. Forces

- Decouple the communication protocol management from the components functionalities to limit the impact of having to choose one communication protocol over another.
- Ease components interoperability and reusability.
- Avoid having to choose, at the design phase of the components development, which communication protocols to support. Ideally, this choice should be made as late as possible in the development process, depending of which components need to be interconnected together (i.e., the integration phase or even at runtime).

D. Solution

Components functionalities can often be used without any concerns on the communication protocols, as they are typically designed to apply operations and algorithms on data, independently of how data are received or sent. Abstracting data management gives the opportunity to develop specialized functionalities independently of the communication protocols. Here is a description of the class involved in MARIE's solution showed in Fig. 2 :

- **Client**. An instance of a component that can execute specialized functionalities. It uses objects implementing Communication Interface to send and receive data. It also instantiates Concrete Port objects needed to communicate with other components.
- **Communication Interface**. Common interface for sending or receiving data.
- **Abstract Port**. Abstract Port object implements Communication Interface to be used as a communication link between components. Port Abstract object are used to handle data formatting and data managing before sending them to Communication Strategy Abstract object responsible of the communication execution.
- **Concrete Port**. Port objects are used to decouple how data are managed and represented from how they are communicated. They serve as an intermediate between components and are responsible for communications issues.
- **Abstract Communication Strategy**. Abstract objects responsible for communication protocol execution handling.
- **Concrete Communication Strategy**. Concrete instances of Communication Strategy abstract responsible for communication protocol execution such as Socket, Shared Memory, I/O streams, CAN bus, serial port, etc.

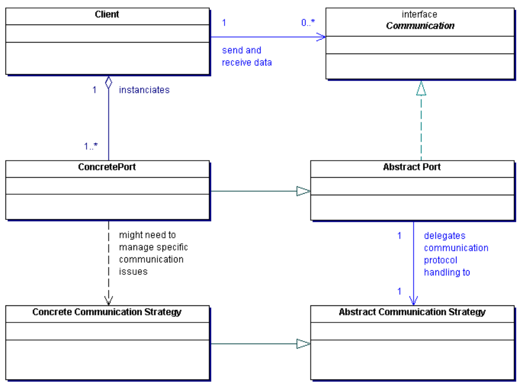

Fig. 2. Port abstraction used in MARIE.

Acknowledgment

F. Michaud holds the Canada Research Chair (CRC) in Mobile Robotics and Autonomous Intelligent Systems. Support for this work is provided by the Natural Sciences and Engineering Research Council of Canada, the Canada Research Chair program and the Canadian Foundation for Innovation.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[2] C. Cote, D. Letourneau, F. Michaud, J.-M. Valin, Y. Brosseau, C. Raievsky, M. Lemay, and V. Tran, "Code reusability tools for programming mobile robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.