ARS ADVANCED ROBOTIC SYSTEMS INTERNATIONAL

Robotic Software Integration Using MARIE

Carle Côté; Yannick Brosseau; Dominic Létourneau; Clément Raïevsky & François Michaud
Université de Sherbrooke, Department of Electrical Engineering and Computer Engineering,
Sherbrooke (Québec), Canada,
{Carle.Cote, Yannick.Brosseau, Dominic.Letourneau, Clement.Raievsky, Francois.Michaud}@USherbrooke.ca

Abstract: *This paper presents MARIE, a middleware framework oriented towards developing and integrating new and existing software for robotic systems. By using a generic communication framework, MARIE aims to create a flexible distributed component system that allows robotics developers to share software programs and algorithms, and design prototypes rapidly based on their own integration needs. The use of MARIE is illustrated with the design of a socially interactive autonomous mobile robot platform capable of map building, localization, navigation, tasks scheduling, sound source localization, tracking and separation, speech recognition and generation, visual tracking, message reading and graphical interaction using a touch screen interface.*
Keywords: *software integration environment, autonomous robotics system, middleware, rapid prototyping.*

1. Introduction

Robotics and artificial intelligence are system sciences aiming at developing and integrating the necessary capabilities for making systems work in real applications. Rapid and specialized progress in the variety of associated domains makes integration a very challenging objective. Different development platforms, design practices, communication protocols, programming preferences and methods, operating systems, industrial or field requirements are driving robotics developments, increasing the complexity of integration. The design of a highly sophisticated mobile robot requires the integration of capabilities usually developed independently, such as localization and mapping, navigation, visual tracking, speech recognition, signal processing, planning, human-robot interface, just to name a few. The appropriate use of heteroclite software components developed in different contexts is therefore essential for efficient scientific and incremental progress of the autonomous robotics field, avoiding reinventing what is made available by others, and focusing efforts towards new discoveries.

However, designing a development and integration software environment for robotic systems is not an easy task. Because software development is a necessary process in autonomous mobile robotics, it is becoming more and more important to assist developers in their scientific and engineering work. Many existing programming environments, like Player (Vaughan, R. et al. 2003), CARMEN (Montemerlo, M. et al. 2003), CLARity (Nesnas I.A.D. et al. 2003), OROCOS (Orebäck, A. & Christensen, H. I., 2003) SmartSOFT (Schlegel, C., 2003), MIRO (Utz, H. et al. 2002), ADE (Andronache, V. & Scheutz, M., 2004) and RCS (Gazi, V. et al. 2001), are all proposing different approaches for robotics system development and integration. Most of them are incompatible with each other for different reasons (Orebäck, A. & Christensen, H. I., 2003), such as the use of specific communication protocols and/or mechanisms, different operating systems, robotics platforms, architectural concepts, programming languages, intended purpose, proprietary source codes, etc. This leads to code replication of common functionalities across different programming environments, and to specific functionalities being often restricted to one programming environment. Being able to create shared software infrastructures among the robotics community, based on common requirements and objectives, is clearly an important goal to reach in order to avoid effort duplication (Woo, E. et al. 2003). Identifying common requirements and objectives is challenging in the current context considering that the robotics field is still in an early exploration phase. As a solution it could be valuable to reuse existing programming environments and interconnect them through a common middleware framework to benefit from their respective approaches, instead of having to choose only one of them.

It is with that in mind that we created MARIE (for Mobile and Autonomous Robotics Integration Environment), our middleware framework oriented towards development and integration of new and existing software for robotic systems (Côté, C. et al. 2004). It is designed according to three main software requirements:

1) Reuse available solutions. Integration of existing software components is difficult considering that they are typically developed independently, following their own

set of requirements (e.g. timing, communication protocol, programming language, operating system, objectives, and applications). Reusability in this context is challenging but crucial for the evolution of the field, avoiding becoming experts in all the related areas that must be integrated. This implies that software components reusability and integration are critical, and should be addressed at the very beginning of the development process of a robotic software system.

2) Support multiple sets of concepts and abstractions. From high-level decision-making developers to perceptual processing and motor controllers designers, or from system analysts to testers, experts from many fields and with different objectives have to contribute concurrently on the same system. To cope with multidisciplinary software development teams, multiple sets of concepts and abstractions need to be supported.

3) Support a wide range of communication protocols, communication mechanisms and robotics standards. Actually, there is no unified protocol available, and no consensus has yet emerged from the robotic software community on standards to adopt. This suggests that it might still be too soon to make such choices, with the field having to explore a great variety of ideas, application areas (each one having its own set of constraints, e.g., space, military, human-robot interaction) and to cope with continuously evolving computing technologies. Consequently, being able to interchange communication protocols mechanisms and robotics standards easily, without major code refactoring, means longer life cycle for actual and future implementations.

This paper presents how MARIE follows these software requirements to create a distributed component-based middleware framework that facilitates reusability of applications, tools and programming environments when building integrated and coherent robotics systems. The paper is organized as follows. Section 2 explains how MARIE's software architecture supports key concepts for robotics software systems development. Section 3 presents MARIE's application design framework. Section 4 describes the use of MARIE in the development of a socially interactive and autonomous mobile robot operating in real life settings, followed in Section 5 with results and observations regarding this implementation.

2. MARIE's Software Architecture

MARIE's software architecture is based on three design choices made to address the software requirements presented in Section 1.

2.1. Component Mediation Approach

To implement distributed applications using heterogeneous components, MARIE adapted the Mediator Design Pattern (Gamma, E. et al. 1994) to create a Mediator Interoperability Layer (MIL). The Mediator Design Pattern primarily creates a centralized control unit (named Mediator) interacting with each colleague (referred to as components) independently, and coordinates global interactions between colleagues to realize the desired system. In MARIE, the MIL acts just like the mediator of the original pattern, but is implemented as a virtual space where components can interact together using a common language (similar to Internet's HTML for example). With this approach, each component can have its own communication protocol and mechanism as long as the MIL supports it. It is a way to exploit the diversity of communication protocols and mechanisms, to benefit from their strengths and maximize their usage, and to overcome the absence of standards in robotic software systems. It also promotes loose coupling between components by replacing a many-to-many interaction model with a one-to-many interaction model. In addition to simplify each component communication interface, loose coupling between components increases reusability, interoperability and extensibility by limiting their mutual dependencies and hiding their internal implementation. By using a virtual space approach, the MIL's design reduces the potential complexity of managing large number of centralized components, as observed with the original pattern. This is mainly attributed on having limited centralization to communication protocols and mechanisms, and leaving most of the functionalities decentralized.

2.2 Layered Architecture

Supporting multiple sets of concepts and abstractions can be achieved in different ways. MARIE does so by adopting a layered software architecture, defining different levels of abstraction into the global middleware framework. Three abstraction layers are used to reduce the amount of knowledge, expertise and time required to use the overall system. It is up to the developer to select the most appropriate layer for adding elements to the system. At the lower level, the *Core Layer* consists of tools for communication, data handling, distribute computing and low-level operating system functions (e.g., memory, threads and processes, I/O control). The *Component Layer* specifies and implements useful frameworks to add components and to support domain-specific concepts. The *Application Layer* contains useful tools to build and manage integrated applications using available components, to craft robotic systems.

2.3 Communication Protocol Abstraction

Component functionalities can often be used without any concerns with the communication protocols, as they are typically designed to apply operations and algorithms on data, independently of how data are received or sent. This eases components interoperability and reusability by avoiding fixing the communication protocol during the component design phase. Ideally, this choice should be made as late as possible, depending of which components

need to be interconnected together (e.g., at the integration phase or even at runtime). Therefore, a communication abstraction framework, called port, is provided for communication protocols and component's interconnections.

3. MARIE's Application Design Framework

Robotics application development with MARIE is based on reusable software blocks, called components, which implement functionalities by encapsulating existing applications, programming environments or dedicated algorithms. Components are configured and interconnected to implement the desired system, using the software applications and tools available through MARIE. Four types of components are used in the MIL: Application Adapter (AA), Communication Adapter (CA), Application Manager (AM) and Communication Manager (CM). AA's role is to interface applications with the MIL and to make them interact with each other. CAs are communication logic link components that ensure communication between other components by adapting incompatible communication mechanisms and protocols, or by implementing traditional routing communication functions. For example, a CA could link an AA providing data at a fixed rate with an AA requiring it asynchronously. AM and CM are system level components that instantiate and manage components locally or across distributed processing nodes. They can, for instance, restart a component not responding for a while, or can move components from one processing node to another to avoid CPU overloads. Interconnections using port communication abstraction are illustrated in Fig. 1 with small red box in components.

MARIE's component frameworks and software tools that manage and abstract useful functionalities (e.g., interconnections and communications, threads and processes, management function such as init, start, stop, pause, resume, restart, abort and quit, distributed system management, data filters and conversions, static and dynamic configurations, event handling) support the creation of components. Note however that although the use of these component frameworks and software tools is highly encouraged to save time and efforts, MARIE is not limited to them. Developers can use the best solution to integrate software applications and interconnect components by having the possibility to extend or adapt existing components and available frameworks. MARIE's underlying philosophy is to complement existing applications, programming environments or software tools, and therefore it is to be used only when required and appropriate.

Fig. 1 illustrates in an example how software applications can be integrated and interconnected in the MIL. Application A represents an integrated application directly linked with the implementation of its AA (e.g., a library or an open source application). When an application is integrated using an AA, it can use the MIL communication protocols to exchange data with any other components, as it is the case for providing data to Application Adapter 2 and Application Adapter 3. Application B interacts with other applications in two different ways. The first one needs Application Adapter 1 to transmit data to Application Adapter 2, which convert them into a specific communication protocol not supported by the MIL to make them available to Application B. The second one is used to send back data to Application Adapter 3, using a communication protocol supported by the MIL allowing direct interconnection with any components. However, Application B and Application Adapter 3 do not use compatible communication mechanisms or protocols, requiring a CA to interface them. Application Adapter 3 implements functionalities directly in the MIL by encapsulating them in a "stand-alone" AA (e.g., a graphical user interface implemented in the AA directly). Application C is already able to communicate with Application B. Therefore, no interconnection through the MIL is required.

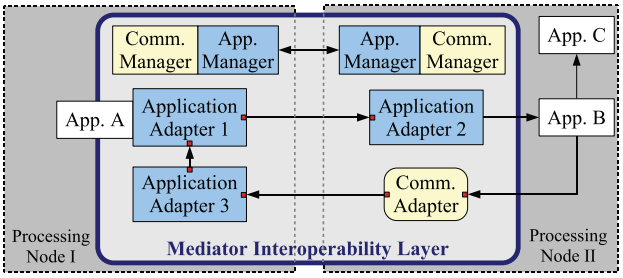

Fig. 1. MARIE's application design framework

4. Using MARIE to Design a Socially Interactive Autonomous Mobile Robot

Spartacus (Michaud et al. 2005) is a socially interactive mobile robot designed to operate in real life settings. The robot must be capable of autonomous navigation and localization, extract visual information from the world (such as reading messages, tracking people), localize, track and separate sound sources for enhanced speech recognition and dialogue interaction, provide graphical information through the touch screen interface, and schedule tasks (Beaudry et al. 2005) to satisfy temporal constraints. Integration of the decision-making processes to implement such scenario is done following a computational architecture called Motivated Behavioral Architecture (MBA) (Michaud et al. 2005).

MARIE is used for the software implementation of Spartacus' decision-making processes. Spartacus implementation requires 45 components (~50 000 lines of code) composed of 26 AA, 17 CA and two external applications. Processing is distributed over three on-board computers (Pentium M 1.6 GHz) and one external laptop (Pentium M 2.0 GHz). Except for the two external

applications, component interconnections are all sockets-based using *Push*, *Pull* and *Events* dataflow communication mechanisms (Zhao, Y. 2003) with XML encoding for data representation. The two external applications use their own communication protocols.

AAs are used to interface the different software applications required for decision-making by the robot. *Mailboxes*, *Splitters*, *SharedMap* and *Switches* are the four types of CA used. A *Mailbox* serves as a data buffer between asynchronous components. A *Splitter* forwards incoming data to multiple outputs. A *SharedMap* is a push-in/pull-out key-value data structure used to store system states that can be accessible by many components. A *Switch* sends only one of its inputs to an output port.

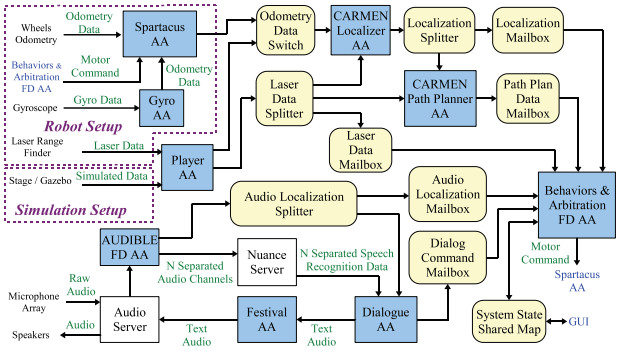

Fig. 2. Partial software implementation design of Spartacus' decision-making processes using MARIE

Partial implementation of Spartacus' decision-making processes is illustrated in Fig. 2. It covers sensing and acting in simulation and real robot setups, localization, path planning, sound source localization, tracking and separation, speech recognition and generation, and part of the computational architecture responsible for the robot's navigation and interactions capabilities.

In the real robot setup, SpartacusAA combines wheels odometry and gyroscopic (through GyroAA interfacing a gyroscope installed on Spartacus) data, and pushes the result at a fixed rate (10 Hz) to its interconnected component. Laser data is collected by PlayerAA, interfacing the *Player* library specialized for sensor and actuator abstraction (Vaughan et al. 2003), supporting the SICK LMS200 laser range finder installed on Spartacus. PlayerAA pushes data at a fixed rate (10 Hz) to connected components. In the simulation setup, odometry and laser data are both collected with PlayerAA, as generated using *Stage* (2D) or *Gazebo* (3D) simulators (Vaughan et al. 2003).

CARMEN Localizer AA and CARMEN Path Planner AA integrate *CARMEN* (Carnegie Mellon Robot Navigation Toolkit), offering path planning and localization algorithms (Montemerlo et al. 2003). CARMEN's localization algorithm relies on a pre-loaded map of the environment. Such map is created for Spartacus using an application based on the *Pmap* library (URL: http://www-robotics.usc.edu/~ahoward/pmap) for 2D mapping (not shown in Fig. 2). CARMEN's path planning algorithm also depends on dynamic goals received from an external component (not shown on Fig. 2).

RobotFlow (RF) and *FlowDesigner* (FD) programs (Côté et al. 2003) are used to implement Behavior & Arbitration FD AA, handling part of the MBA decision-making architecture. RF and FD are two modular data-flow programming environments that facilitate visualization and understanding of the robot's control loops, sensor, and actuator processing. In this component, RF/FD programs implement behavior-producing modules arbitrated using a priority-based approach. It uses data coming from different elements such as localization, path plan, audio localization, dialogue command and system states. Behavior & Arbitration FD AA uses an asynchronous pull mechanism to get its data, requiring the use of Mailbox CA components. It generates motor commands at a fixed rate (5 Hz).

The Audio Server and Nuance Server are the two external applications not integrated as MARIE components. Audio Server is interfacing a RME Hammerfal DSP Multiface sound card, and Nuance Server is interfacing *Nuance*, a speech recognition application (URL: http://www.nuance.com). DialogueAA is a "stand-alone" AA that manages simultaneous conversations with people. This is made possible with the use of AUDIBLE FD AA interfacing AUDIBLE (Valin et al. 2005), the sound sources localization, tracking and separation algorithms implemented with RF/FD and using a microphone array installed on Spartacus. It generates a number of separated audio channels that are sent to Nuance Server and behavior-producing modules. Recognized speech data is sent to Festival AA, interfacing the speech generation application *Festival* (Taylor et al. 1998), through the Dialogue AA. DialogueAA also provides data to the behavior-producing modules.

The global execution of the system is asynchronous, having most of the applications and AAs pushing their results at a variable rate, determined based on the computation length of their algorithms when triggered by new input data. Synchronous execution is realized by having fixed rate sensors readings and actuators commands writings.

5. Results and Observations

Although MARIE was used in smaller implementations, Spartacus represented its first real integration challenge and is still ongoing work. This section presents preliminary results and observations regarding MARIE's software requirements and design architecture based on our work on Spartacus.

MARIE is coded in C++ (~10 000 lines of code) and uses ACE (Adaptive Communication Environment) library for the transport layer (TCP/IP, Shared memory, etc.) and low-level operating system functions (threads and processes, memory access, timers, operating systems

interoperability, real-time support, etc.) (Schmidt, D.C. 1994). Although ACE met Spartacus' implementation needs, MARIE does not rely on this specific transport layer library.

MARIE shows interesting capabilities for software integration and team development. At the peak of Spartacus' software development process, eight software developers (seven specialists and one integrator) were working concurrently on the system. Most of them only used the AA component framework (*Component Layer*) to create their components, conducting unit and black-box testing with pre-configured system setups (*Application Layer*) given by the integrator. Communication protocols and operating system tools for component and application developments (*Core Layer*) were added by the core specialist when required. Components were incrementally added to the system as they became available. It took around eight days, spread over a four weeks period, to have a fully integrated system. Nine existing specialized applications/libraries were integrated together to build the complete system: Player/Stage/Gazebo, Pmap, CARMEN, Flowdesigner/RobotFlow, AUDIBLE, Nuance, Festival, DECIBEL, QT3 and OpenCV. Each of these applications required different integration strategies. For instance, Nuance is a proprietary application with a specific and limited interface. Integrating Nuance in an AA was challenging because its execution flow is tightly controlled by Nuance's core application, which is not accessible from the available interface. To solve this problem, we created an independent application that uses a communication protocol already supported by the MIL. CARMEN, on the other hand, is composed of small executables communicating through a central server. CARMEN's integration was realized by creating an AA that starts several of these executables depending on the required functionality, and on data conversion from CARMEN's to MIL's format. Having flexible component frameworks and port's communication protocol abstraction allowed us to adapt application specificities such as external threads execution, dynamic bindings, independent protocols and timing.

Choosing XML data representation for common language communication in the MIL was based on implementation simplicity and ease of debugging. Although it was sufficient for most of the system communication needs, we clearly observed that this solution was not sufficient to support communication-intensive data like audio and vision within MARIE. To avoid taking precious time to support optimized protocols for audio and video, we decided to use FD that already provides those protocols.

Regarding component interoperability, being able to change between simulation and robotic setups with only few system modifications gave us the possibility to do quick simulation and integration tests. Nearly 75% of system functionalities were validated in simulation and were used as is in the real world setup. In both simulated and real setups, configurations of components receiving laser and odometry data are exactly the same, abstracting data sources and benefiting from components modularity. Moreover, component interoperability can be extended with MARIE to do things like porting a computational architecture on robotic platforms from different manufacturers and with heterogeneous capabilities, or evaluating performances of algorithms implementing the same functionality (e.g., localization, navigation, planning) using the same platform and experimental settings.

Distributing applications across multiple processing nodes was not difficult with MARIE, having chosen socket as the transport mechanism. We initially used a shared memory transport mechanism to accelerate communication between components on the same computer. Changing from one transport mechanism to another was transparent using port's communication protocol abstraction and supporting shared memory interconnection in the MIL. Since no noticeable impact was observed over the global system performances using either of them, we chose to exclusively use socket transport mechanism. It allows us to quickly change distributed components configuration.

Meeting Spartacus' integration needs using MARIE rapid-prototyping approach highlighted three interesting consequences on robotics system development. First, it revealed the difficulty of tracking decisions made by the system simply by observing its behaviors in the environment, something that was always possible with simpler implementations. The system reached a level of complexity where we needed to develop a graphical application, called logviewer, to follow on-line or study off-line the decisions made by the robot. This suggests that creating analysis tools and supporting them in the integration environment can play a key role in working with such a highly-integrated system. The second observation emerged from the number of components involved in the software architecture. Manually configuring and managing the system with many components executed on multiple processors, is an error-prone and tedious task. In this context, MARIE would greatly benefit having GUI and system management tools to build, configure and manage components automatically. Third, regarding design optimization, being able to quickly interconnect components to create a complete implementation, without focusing on optimization right away, revealed to be a good strategy to identify real optimization needs. Such an exploration strategy led to quickly reject some applications, software designs or component implementations without investing too much time and efforts. For Spartacus, we originally thought that tighter synchronization between components would be necessary to obtain a stable system and support real-time decision-making. Having connected all of Spartacus' components together, we observed that performances were appropriate with the

processing power available as long as we did not overloads the computers with too many components. Noticing that, we decided to wait before investing time and energy working on component synchronization, to focus on Spartacus' integration challenge.

6. Conclusion

MARIE is a middleware framework oriented towards developing and integrating new and existing robotic software. It shows interesting characteristics as a rapid-prototyping approach to create robotics systems, and tries to overcome the current lack of standards in the autonomous robotic field. We believe that frameworks like MARIE are important because they are more than just engineering tools: they are part of the scientific process of studying and designing autonomous systems considering their influences on implementation capabilities and on the discovery of new needs.
In future work, testing will be performed on Spartacus to validate MARIE's architectural design and implementation. We are currently working on identifying and implementing tools to measure real-time performances, and on software metrics to quantify MARIE's computational overhead.

Acknowledgment

F. Michaud holds the Canada Research Chair (CRC) in Mobile Robotics and Autonomous Intelligent Systems. Support for this work is provided by the Canada Research Chair program, the Natural Sciences and Engineering Research Council of Canada and the Canadian Foundation for Innovation. MARIE is an open-source project available at http://marie.sourceforge.net.

7. References

Andronache, V. & Scheutz, M. (2004) ADE - A tool for the development of distributed architectures for virtual and robotic agents. *Proceedings of the AT2AI-04 Symposium at the 17th European Meeting on Cybernetics and Systems Research*.

Beaudry, É.; Brosseau, Y.; Côté, C.; Raïevsky, C.; Létourneau, D.; Kabanza, F. & Michaud, F. (2005). Reactive planning in a motivated behavioural architecture. *Proceedings American Association for Artificial Intelligence Conference*, pp. 1242-1247.

Côté, C.; Létourneau, D.; Michaud, F.; Valin, J.-M.; Brosseau, Y.; Raïevsky, C.; Lemay, M. & Tran. V. (2004). Code reusability tools for programming mobile robots. *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.1802-1825.

Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gazi, V.; Moore, M.L.; Passino, K.M.; Shackleford, W.P.; Proctor, F.M. & Albus, J. (2001). *The RCS Handbook: Tools for Real-Time Control Systems Software Development*. Wiley 298 p.

Michaud, F.; Brosseau, Y.; Côté, C.; Létourneau, D.; Moisan, P.; Ponchon, A.; Raïevsky, C.; Valin, J.-M.; Beaudry, É. & Kabanza, F. (2005). Modularity and integration in the design of a socially interactive robot. *Proceedings IEEE International Workshop on Robot and Human Interactive Communication*.

Montemerlo, M.; Roy, N. & Thrun, S. (2003). Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. *Proceedings. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2436–2441.

Nesnas, I.A.D.; Wright, A.; Bajracharya, M.; Simmons, R. & T. Estlin. (2003). CLARaty and challenges of developing interoperable robotic software. *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2428–2435.

Orebäck, A. & Christensen, H. I. (2003). Evaluation of architectures for mobile robotics. *Autonomous Robots*, Vol. 14, pp. 33–49.

Schlegel, C. (2003). A component approach for robotics software: Communication patterns in the OROCOS context. *18 Fachtagung Autonome Mobile Systeme (AMS)*, pp. 253-263.

Schmidt, D.C. (1994). ACE: an object-oriented framework for developing distributed applications. *Proceedings of the 6th USENIX C++ Technical Conference*. Cambridge, Massachussetts.

Taylor, P. A.; Black A. & Caley R. (1998). The architecture of the festival speech synthesis system. In *The Third ESCA Workshop in Speech Synthesis*, pp. 147-151, Jenolan Caves, Australia.

Utz, H.; Sablatnög, S.; Enderle, S. & Kraetzschmar, G. (2002). MIRO – Middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 4, pp. 493–497.

Valin, J.-M. (2005). Auditory system for a mobile robot. Ph.D. thesis, Department of Electrical Engineering and Computer Engineering, Université de Sherbrooke.

Vaughan, R.T.; Gerkey, B.P. & Howard, A. (2003). On device abstractions for portable, reusable robot code. *Proceedings. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2421–2427.

Woo, E.; MacDonald, B. A. & Trépanier, F. (2003). Distributed mobile robot application infrastructure. *Proceedings. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1475–1480.

Zhao, Y. (2003). A model of computation with push and pull processing. Master's thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley.