Distributed Mobile Robot Application Infrastructure

Evan Woo
EDS(NZ) Ltd
Email: EvanWoo at Ihug.co.nz

Bruce A. MacDonald   Félix Trépanier
EEE Department, University of Auckland, New Zealand
Email: B.MacDonald at Auckland.ac.nz

***Abstract*— A distributed mobile robot software application infrastructure is developed, improving integration and leverage between projects in a research environment. The resulting design includes a three layer CORBA based, service broker application architecture. A reference implementation and tests on B21r, LEGO Mindstorm and Khepera robots demonstrate the feasibility of the design.**

I. Introduction

Our distributed mobile robot software application infrastructure will improve the integration and leverage between projects, particularly in a diverse university or laboratory research environment with different robots, operating systems, programming languages, and researchers. It adopts a CORBA based application architecture comprising three layers: application, infrastructure services, and middleware.

The application layer contains components developed by researchers, which are registered as services. The infrastructure services layer is a broker, following the CORBA Trader specification. The broker provides the protocol between services as well as query facilities for clients. The middleware layer handles communications between services. CORBA allows a diverse range of languages and operating platforms. Our reference implementation includes robot services for the B21r, Khepera, and LEGO Mindstorm robots, the service broker, a remote robot control application, plus a web–enabled version. Real–time facilities are not yet included. Although there are issues to overcome, ORB compatibility and performance in a busy network, the application infrastructure provides a good framework for researchers.

A. Robot Programming Systems

Mobile robot researchers face difficulties developing large software systems. Much of the software infrastructure is proprietary, much is necessarily targeted at specific hardware, robot software development kits may be limiting, and there is a lack of open standards to promote collaboration, code reuse and integration.

There is much work on robot programming systems, but little coherence and little focus on the underlying architecture. Themes include client/server based tools [1], specific methods for producing robot software, for example graphically [2], reusable software architecture [3], portable application programming interfaces [4, in ADA], hierarchies of classes with attention to their interfaces [5], [6], fault tolerance [7], component interaction via a blackboard [6], mission programming aimed at end users [8, with three layers: dynamics, control and planning], layer architectures [9, also expects to support different paradigms depending on the application], real–time object–oriented automation [10, a simulator is used to check the behaviour and code is generated automatically], component based systems for real time programming [11, used for some commercial applications]. Many authors propose simulation and virtual reality systems to support the programming task.

A number of authors describe distributed robot programming systems, including architectures for: the separation of cognitive from reactive robot components [12], robot planning [13], components and patterns [14, in robot workcell programming], distributed development in which workflows are distinguished at three levels: user, system and execution [15], hybrid models for integrating decision and reactive levels [16], a layered, object–oriented client/server architecture for sensorimotor systems [17]. Orocos [18] focuses on object–oriented components and patterns. There is little emphasis on the system architecture. CORBA inspires the distributed communications and is used in some work, although there is concern about its performance. Miro [19] is an object–oriented robot middleware system based on ACE [20] and the associated real–time CORBA ORB, TAO. Miro is a layered client/server architecture that provides frameworks for common data structures, functionality, and communications, and has been implemented on a number of robots. MCA2 [21] focuses on reusable modules all with the same standardised interface, for robot control at the sensor/actuator level. Claraty [22] provides an architecture with functional and decision layers, and a set of functional components for JPL's mobile robot systems.

B. Objectives

Many authors describe systems with strong architectural decisions, necessary in software production settings. Our work focuses on reuse via an easily accessible *infras-*

tructure for researchers. Components are the eventual goal, but we usually don't know enough to create them until the research has matured to standard forms. The infrastructure must enable more experimentation with the system architecture than in a production environment.

Our aim is to design an application infrastructure for the development and operation of robot software, one that is open to new research ideas for robot programming. At the low level, the infrastructure should accommodate different languages and platforms, and promote collaboration, software reuse, and leverage between projects. At a higher level the infrastructure services should ideally be architecturally neutral so that we may experiment with different architectural styles. An earlier project developed an extendable framework for writing robot applications in C++ in Microsoft Windows, including facilities for distributing the application on more than one computer [23]. A substantial redesign was needed in order to provide full multithreading, to provide for other programming languages and platforms, to simplify distributed development, and to provide a consistent and structured development environment that would enable researchers to easily contribute to the overall robot system.

We first analyse the requirements for robot programming systems, then compare a number of specific existing systems. We then explain the preferred design based on a CORBA trading service.

II. Requirements

The analysis was based on the operational context, concerned with *system usage*, and system capabilities such as *scalability*, *manageability*, *performance*, *reliability*, *security* plus other observable system qualities, and the developmental context, concerned with aspects of system and application development, such as *design*, *coding*, *programming languages*, *development environment/tools*, *software reuse.* The resulting requirements are summarised below. A key factor is providing the flexibility for new researchers to easily develop and deploy useful software, within the period of a typical short research project, and for this the infrastructure must support a high level of abstraction of robot hardware, and the flexibility to use different languages and platforms. This makes a distributed infrastructure essential, and drives other requirements, such as code reuse, so that a researcher does not spend a significant time creating the infrastructure. For example, the time consuming practice of porting software to a special robot environment, should be avoided.

A. Operational context requirements:

RQ1: *Distributed software environment.*
RQ2: *Application centric not robot centric* — Hardware diversity prevents standardising onboard robot software; the architecture should standardise software interfaces, so an application can be assembled transparently.
RQ3: *Support parallel and distributed processing for any application components.*
RQ4: *Robot independence* — an important consequence of RQ2
RQ5: *Support applications with multiple robots.*
RQ6: *Preserve system and application integrity* — conflicting commands need to be resolved

B. Developmental context requirements:

RQ7: *Promote a high level of software reuse.*
RQ8: *Programming language independent.*
RQ9: *Platform independent.*

C. Trends in Mobile Robot Applications

A number of other significant trends support these requirements. There is much interest in the remote control of physical devices over networks, including the use of online teleoperated robots [24], [25], [26]. These require an infrastructure to support a distributed architecture, and solutions to time delays, limited sensory feedback, user interface design, real-time problems with the HTTP protocol, complex data communication, and dependencies on diverse web server and middleware technologies. A flexible and powerful distributed architecture is needed to support development of multirobot applications, and communications between robots. The use of disposable robots in dangerous situations, such as bomb disposal, demining, and chemical handling, requires a distributed architecture to control and manage robots remotely.

D. Current Robot Software Systems

A detailed study was carried out on four systems available in our labs. MROS [23] provides a non–preemptive architecture and communications across a dedicated serial link to a robot. A messaging service abstraction layer routes messages to the appropriate controller, hiding details from the programmer. Only non–preemptive multithreading is provided. The main controller cannot be distributed and could be a bottleneck as additional components are added. Only Windows 3.1 and C++ are supported. The API is not clearly delineated.

The K–Team Khepera robot provides cross–compilation tools, plugins for systems such as LabVIEW and Matlab, and a serial communications protocol. It enables any language to be used for remote control, but lacks debugging facilities. The lack of a defined software framework for remote application development limits code reuse.

The B21r has a substantial controller and networking support. Its extendable CORBA Mobility system provides for distributed applications, multiple robots and integration. Opportunity for reuse is provided by base framework

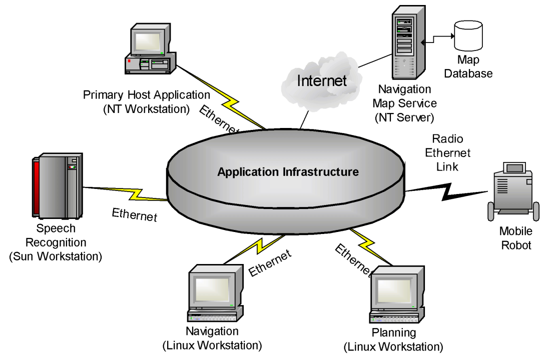


Fig. 1

THE APPLICATION INFRASTRUCTURE FACILITATES INTEGRATION

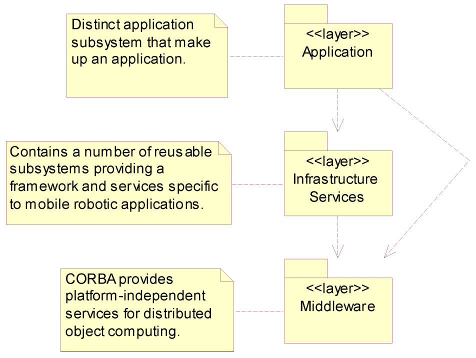

Distinct application subsystem that make up an application.
<<layer>>
Application
Contains a number of reusable subsystems providing a framework and services specific to mobile robotic applications.
<<layer>>
Infrastructure
Services
CORBA provides platform-independent services for distributed object computing.
<<layer>>
Middleware

Fig. 2

A LAYERING APPROACH TO THE MOBILE ROBOT APPLICATION INFRASTRUCTURE DESIGN

classes. A CORBA naming service is used to manage distributed software components. A graphical tool helps debug applications. The programming interface is defined in a language independent manner using open CORBA standards. The system is scalable, language and platform independent for applications using the existing framework, but extensions must be written in C++.

The LEGO Mindstorm provides for cross–compilation, or remote control via an IR link. Physical flexibility is provided by modular LEGO components. It lacks a standard software development toolkit, but users have developed tools: NQC [27], LeJOS, a cut down Java Virtual Machine [28], and LegOS, firmware for assembly language, C or C++ [29]. Some examples of the remote control program environments are pbFORTH [30] and Visual Basic [31]. Duplicated efforts are solving the same problem, and solutions developed in one language are unlikely to work with another environment. There are no standard ways of developing applications.

The analysis shows that each of the mobile robots surveyed uses different programming languages, tools, communication protocols, and proprietary APIs. As a result, little leverage and code reuse can be achieved. Researchers are required to possess good knowledge of robot hardware and operating system.

III. Distributed Software Technology

There are three classes of distributed service [32]: **Distributed system services** provide critical communications and data interchange services and are our focus in this work; **Application enabling services** provide applications with access to distributed services; **Middleware management services** provide a controlled runtime environment.

For distributed system services Message Oriented Middleware uses message queues for communication (eg. [33]), but does not remove integration, standardisation and compatibility problems, and its asynchronous nature is ill–suited to real time robotics. Distributed objects middleware extends object oriented systems by allowing objects to live across a heterogeneous network yet appear local. It provides synchronous, direct communications between objects, and so is more suited for programs that depend on each other for their actions.

A. Design Considerations

The three popular distributed objects middleware paradigms have similar approaches but different mechanisms: OMG's CORBA, Microsoft's DCOM, and Sun's Java/RMI [34]. CORBA was chosen based on our requirements, since: a) it supports a wide range of programming languages and platforms, b) there are many different implementations available so developers are not dependent on a single vendor's products, c) most CORBA products provide free academic licences, so it is particularly suited for research and development use, and d) it provides a richer set of auxiliary features.

IV. Design

Tests with an initial proof of concept LEGO Mindstorm implementation confirmed that CORBA technology could be applied effectively in our typical distributed robotics scenario [35]. The middleware initialisation time lag from client to server was 8mS for a local C++ client, 15 mS for a remote C++ client (10mS and 20mS respectively for a Java client). Method invocation between client and server was 1mS locally and 2mS remotely for C++ and Java clients. The components of a LEGO Robot Tank Remote Control application were transparently distributed (RQ1); the client and server components could be written in different programming languages (RQ8), and it was not reliant on a single operation platform (RQ9).

The goal of the second phase was to define a standard, common approach for CORBA enabled components, so the resulting architecture will satisfy the requirements. Fig. 1 shows an example scenario.

The design includes the three layers in Fig. 2. Application layer services will be created by researchers. The middleware layer contains the core CORBA subsystems. The infrastructure services layer is the project emphasis.

A. *Service Based Architecture (SBA)*

Distributed architecture models include: client/server, three tier, broker–based, and multiagent [36]. Strong coupling between client and server is undesirable for applications with many components, especially when some server components are dependent on other servers. As the application is scaled up, changes are difficult to manage, and performance bottlenecks may develop. Instead, clients should seek *services*. The details of where and how those services are provided should be immaterial to the client.

Rather than defining a server as a set of services behind a single, static interface, the design proposes a broker–based approach. Shown in Fig. 3, the broker has two types of client: those offering service, and those requesting service; a single client could both offer and request services. Once the broker has found the requested service, it returns the reference of the remote servant object, for direct communication with the client.

To maintain an open architecture the design used the CORBA Trading Service specification with a minimum of added functionality. The Trading Service requires a *service type* which includes the *service type name*, the IDL *interface type*, and a collection of additional *property types*. Each property type has a name, a value, and a mode (which specifies whether the property is mandatory and whether it is read–only). A service importer may specify criteria based on the desired service properties, so that the broker will return only the matching service offer(s). The service type supports inheritance, so *super types* may abstract common properties and functionalities.

A CORBA trader may create, modify and delete service types. Service offers may be exported, imported, modified, and withdrawn. The service type definition and offer management functionality is defined as CORBA IDL interfaces, and so are independent of the trading service implementation. Generally, ORB vendors will provide IDLs for accessing these functions, some provide a GUI tool for administrators. Service exporters register services with the broker, while service importers query the broker by a service search criterion, which is a boolean expression, and may include preferences and a policy.

The application infrastructure is summarised in Fig. 4. A researcher can create a mobile robot application using existing components deployed as services. An application component may query the service broker for a particular service name and properties, rather than requiring prior knowledge of the service implementation or deployment; the components of an application thus have loose couplings. Components can be "swapped" out without changing code or configurations. Furthermore, as this infrastructure promotes the separation of logic and physical robot

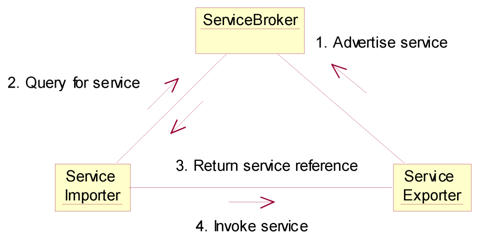


Fig. 3

COLLABORATION BETWEEN SERVICE BROKER, SERVICE EXPORTER, AND SERVICE IMPORTER.

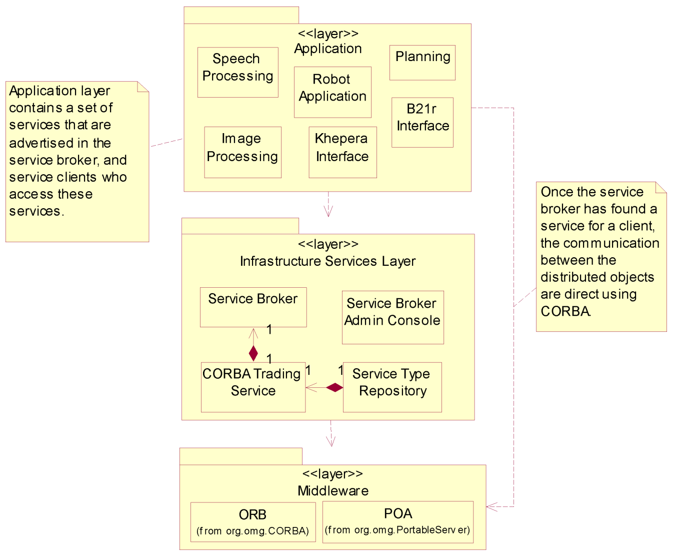

<<layer>>
Application
Speech Processing
Robot Application
Planning
B21r Interface
Image Processing
Khepera Interface
Application layer contains a set of services that are advertised in the service broker, and service clients who access these services.
<<layer>>
Infrastructure Services Layer
Service Broker
Service Broker Admin Console
CORBA Trading Service
Service Type Repository
1
1
1
1
Once the service broker has found a service for a client, the communication between the distributed objects are direct using CORBA.
<<layer>>
Middleware
ORB
(from org.omg.CORBA)
POA
(from org.omg.PortableServer)

Fig. 4

THREE TIER ARCHITECTURE.

control interfaces, (the latter as services), the infrastructure enables a robotic application to be independent from the make, model, and type of robot.

B. *Design Realization*

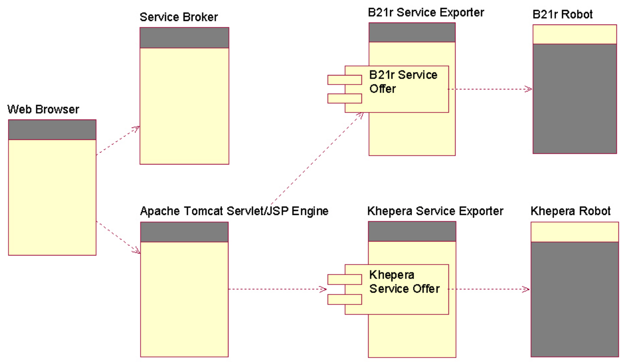
The implementation included the service broker, the robot interface service type definitions for the LEGO Mindstorm, Khepera, and B21r, and interface services. Also, a sample web based robot control application was created. A super service type is defined for the three robots, assuming that a basic mobile robot has two primary motors, connected to two wheels or caterpillars. This super service type contains IDL interface methods for controlling the motors and reading the sensors. The properties defined include whether the robot supports collision detection, how many horizontal layers of proximity sensors there are, the number of sensors per horizontal layer, the type of sensor at each horizontal layer, and the class of robot. An AccessControl service type is used to prevent conflicting robot commands [RQ6]; it has a method to lock a service and one to unlock a service.

The B21r robot interface service makes Mobility objects available to clients. The Khepera robot depends on a serial port interface and it is not possible to implement the interface service onboard. The interface service runs on an external computer to encapsulate the serial link and implement the necessary methods for the Khepera Interface service type, in this case in Java. The LEGO Mindstorm uses an infrared serial port link and its interface service was implemented with a similar class structure as the Khepera robot.

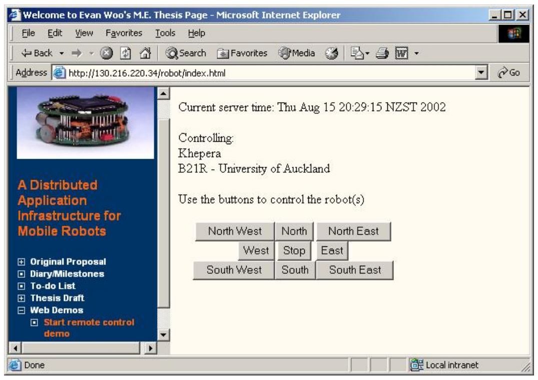
The ORBacus Trading Service was used as the service broker, as it is a full service trader. Additionally ORBacus provides source code as part of an academic license.

A sample application written in Java demonstrates the benefits of the distributed application infrastructure. It allows a user to control a mobile robot from a computer on the network. It can interact with any robots that implement the Basic Mobile Robot Interface service type, so that makes it compatible with the B21r, Khepera, and LEGO Mindstorm robots. The application provides a service query screen, and a control panel screen, with 8 directions and speed controls for any selected robot. The application demonstrates that the distributed infrastructure is application centric rather than robot centric [RQ3], and is independent of specific robots [RQ4]. The infrastructure also promotes software reuse since the three robot interface services can be reused. A web based version of this sample application was created using JSP and Servlet technology. Its user interface and deployment diagram are shown in Fig. 5. The user chooses from a list of mobile robots to control remotely, with a panel of navigation buttons. A user can simultaneously control the movement of the selected robots.

A



B


Welcome to Evan Woo's M.E. Thesis Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://130.216.220.34/robot/index.html Go
A Distributed Application Infrastructure for Mobile Robots
Original Proposal
Diary/Milestones
To-do List
Thesis Draft
Web Demos
Start remote control demo
North West
North
North East
West
Stop
East
South West
South
South East
Done
Local intranet

Current server time: Thu Aug 15 20:29:15 NZST 2002

Controlling:
Khepera
B21R - University of Auckland

Use the buttons to control the robot(s)

Fig. 5

EXAMPLE APPLICATION: A. DEPLOYMENT DIAGRAM OF SERVICES, WEB COMPONENTS AND HARDWARE; B. WEB INTERFACE.

V. EVALUATION

In terms of the software development context, the distributed application infrastructure [RQ1] supports and promotes a high level of software reuse [RQ7] because it allows the collaborating application services to be developed independently of programming language [RQ8] and operating platform [RQ9]. However, potential compatibility issues concerning different versions of CORBA specification and the relatively high learning curve associated with CORBA, could affect the reusability level and uptake of the infrastructure.

In the operational context, the design and implementation highlight the application centric nature [RQ2]; it is possible to create applications that are independent of the mobile robots' make and type [RQ4]. Service type inheritance increases reusability and robot independence, however more in–depth research and analysis are required to identify common services for most mobile robot programs. The example web based controller demonstrates the support for multiple robot applications [RQ5]. Further work is needed on integrity [RQ6]. The performance of CORBA was evaluated to help developers and researchers evaluate the impact on application performance when large amounts of data are passed between services. The time lags introduced by remote CORBA calls with parameter sizes of less than 1KB are less than 2ms, which is acceptable for distributed robot control [RQ3].

VI. CONCLUSION

Problems in mobile robot software development and operation are addressed in the context of a research environment. A design and implementation of a distributed software infrastructure are presented. The approach was to create an infrastructure specification for collaboration between components developed in different research projects, for different robots, including different languages for different operating environments; the architecture is inherently distributed. The design contains three layers: the application layer, infrastructure services layer, and middleware layer. A CORBA based trading service is used for the infrastructure services layer, to provide more flexibility for researchers and research implementations. This distributed software infrastructure provides a backbone for a broader robot programming environment. Although the infrastructure is in early development, our current robot visualisation tool project uses it to interface with robot services to graphically simulate robots' behavior [37].

VII. REFERENCES

[1] O. Kubitz, M. Berger, and R. Stenzel, "Client-server-based mobile robot control," *IEEE/ASME Trans. Mechatron.*, vol. 3, no. 2, pp. 82–90, June 1998.

[2] A. Bredenfeld and G. Indiveri, "Robot behavior engineering using DD–designer," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA '01)*, 2001, pp. 205–210.

[3] C. Kapoor and D. Tesar, "A reusable operational software architecture for advanced robotics," The University of Texas at Austin, Tech. Rep., 1998. [Online]. Available: http://www.robotics.utexas.edu/rrg/downloads/software/oscar/oscar.html

[4] M. G. Harbour, R. G. Somarriba, A. Strohmeier, and J. Jacot., "PINROB: A portable API for industrial robots," in *International Conf. on Reliable Software Technologies, Ada-Europa'98, Uppsala, Sweden, in Lecture Notes in Computer Science No. 1411*, June 1998, pp. 151–162.

[5] D. Miller and R. Lennox, "An object-oriented environment for robot system architectures," *IEEE Control Syst. Mag.*, vol. 11, no. 2, pp. 14–23, February 1991.

[6] B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, and M. Balabanovic, "A domain-specific software architecture for adaptive intelligent systems," *IEEE Trans. Software Eng.*, vol. 21, no. 4, pp. 288–301, April 1995.

[7] L. Parker, "Alliance: an architecture for fault tolerant multirobot cooperation," *IEEE Trans. Robot. Automat.*, vol. 14, no. 2, pp. 220–240, April 1998.

[8] E. Ċoste-Manière and N. Turro, "The MAESTRO language and its environment: specification, validation and control of robotic missions," in *Proc. IROS '97, Intelligent Robots and Systems*, vol. 2, 7–11 September 1997, pp. 836–41.

[9] K. Nilsson and R. Johansson, "Integrated architecture for industrial robot programming and control," *Robotics and Autonomous Systems*, vol. 29, no. 4, pp. 205–26, Dec 1999.

[10] L. B. Becker and C. E. Pereira, "SIMOO–RT — an object–oriented framework for the development of real–time industrial automation systems," *IEEE Trans. Robot. Automat.*, vol. 18, no. 4, pp. 421–30, August 2002.

[11] S. A. Schneider, V. W. Chen, and G. Pardo-Castellote, "The ControlShell component-based real-time programming system," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '95)*, vol. 3, 21–27 May 1995, pp. 2381–8.

[12] M. Piaggio, A. Sgorgbissa, and R. Zaccaria, "Ethnos: a light architecture for real-time mobile robotics," in *Proc IROS'99 Intell Robots & Sys's*, vol. 3, 1999, pp. 1292–97.

[13] S. Flavell, S. Winter, D. Wilson, and E. Hart, "Design of a distributed software architecture for an intelligent planning system," *IEE Proceedings – Control Theory and Applications*, vol. 143, no. 2, pp. 125–31, March 1996.

[14] J. Beck, J. Reagin, T. Sweeny, R. Anderson, and T. Garner, "Applying a component-based software architecture to robotic workcell applications," *IEEE Trans. Robot. Automat.*, vol. 16, no. 3, pp. 207–17, June 2000.

[15] O. Stasse and Y. Kuniyoshi, "PredN: achieving efficiency and code re-usability in a programming system for complex robotic applications," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '00)*, vol. 1, April 2000, pp. 81–7.

[16] S. Fleury, M. Herrb, and R. Chatila, "Gen^{en}oM: a tool for the specification and the implementation of operating modules in a distributed robot architecture," in *Proc. IROS '97, Intelligent Robots and Systems*, vol. 2, 7–11 September 1997, pp. 842–9.

[17] C. Schlegel and R. Wörz, "The software framework SMARTSOFT for implementing sensorimotor systems," in *Proc. IROS '99 Intelligent Robots and Systems*, vol. 3, 1999, pp. 1610–6.

[18] "Orocos." [Online]. Available: http://www.orocos.org

[19] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro — middleware for mobile robot applications," *IEEE Trans. Robot. Automat.*, vol. 18, no. 4, pp. 493–7, Aug 2002.

[20] D. Schmidt, "Adaptive communication environment (ACE)." [Online]. Available: http://www.cs.wustl.edu/~schmidt/ACE.html

[21] "Mca2." [Online]. Available: http://mca2.sf.net

[22] "CLARAty." [Online]. Available: http://robotics.jpl.nasa.gov/tasks/claraty/homepage.html

[23] J. Coleman and A. Marter, "Mobile robot operating system with position location," Department of Electrical and Electronic Engineering, University of Auckland," Project Reports (2), 1997.

[24] K. Goldberg, C. Sutter, J. Wiegley, and B. Farzin, "The Mercury project: A feasibility study for online robots," in [38].

[25] F. M. P. Saucy and F. Mondada, "KhepOnTheWeb: One year of access to a mobile robot on the internet," in [38].

[26] R. G. R. Simmons, S. Koenig, J. O'Sullivan, and G. Armstrong, "Xavier: An autonomous mobile robot on the web," in [38].

[27] D. Baum, *Dave Baum's Definitive Guide to LEGO Mindstorms*. APress, 2000.

[28] B. Bagnall, *Core LEGO MINDSTORMS Programming: Unleash the Power of the Java Platform*. Prentice Hall, 2002.

[29] M. L. Noga, "About legOS," 1998. [Online]. Available: http://www.noga.de/legOS/

[30] R. Hempel, "pbForth." [Online]. Available: http://www.hempeldesigngroup.com/lego/pbForth/homePage.html

[31] D. H. S. Hearne, "A course for programming the RCX with Visual Basic." [Online]. Available: http://emhain.wit.ie/~p98ac25/

[32] M. Bray, "Software technology review: Middleware," 1997. [Online]. Available: http://www.sei.cmu.edu/str/descriptions/middleware.html

[33] B. Dalton, "A distributed framework for online robots," in [38].

[34] C. Szyperski, *Component Software: Beyond Object–Oriented Programing*. Addison-Wesley, 1999.

[35] E. Woo, "A distributed application infrastructure for mobile robots," Master's thesis, University of Auckland, New Zealand, 2002.

[36] D. Brugali and M. E. Fayad, "Distributed computing in robotics and automation," *IEEE Trans. Robot. Automat.*, vol. 18, no. 4, pp. 409–420, August 2002.

[37] F. Trépanier, "A graphical simulation and visualisation tool for distributed mobile robotics applications," Master's thesis, University of Auckland, New Zealand, 2003.

[38] K. Goldberg and R. Siegwart, Eds., *Beyond Webcams: An Introduction to Online Robots*. MIT Press, 2002.