Auton Robot (2007) 22:101–132
DOI 10.1007/s10514-006-9013-8

Development environments for autonomous mobile robots: A survey

James Kramer · Matthias Scheutz

Received: 2 July 2006 / Revised: 16 October 2006 / Accepted: 20 October 2006 / Published online: 1 December 2006

Abstract *Robotic Development Environments* (RDEs) have come to play an increasingly important role in robotics research in general, and for the development of architectures for mobile robots in particular. Yet, no systematic evaluation of available RDEs has been performed; establishing a comprehensive list of evaluation criteria targeted at robotics applications is desirable that can subsequently be used to compare their strengths and weaknesses. Moreover, there are no practical evaluations of the usability and impact of a large selection of RDEs that provides researchers with the information necessary to select an RDE most suited to their needs, nor identifies trends in RDE research that suggest directions for future RDE development.

This survey addresses the above by selecting and describing nine open source, freely available RDEs for mobile robots, evaluating and comparing them from various points of view. First, based on previous work concerning agent systems, a conceptual framework of four broad categories is established, encompassing the characteristics and capabilities that an RDE supports. Then, a practical evaluation of RDE *usability* in designing, implementing, and executing robot architectures is presented. Finally, the *impact* of specific RDEs on the field of robotics is addressed by providing a list of published applications and research projects that give concrete examples of areas in which systems have been used. The comprehensive evaluation and comparison of the nine RDEs concludes with suggestions of how to use the results of this survey and a brief discussion of future trends in RDE design.

Keywords Robotics · Programming environment · Comparison · Software

J. Kramer (✉) · M. Scheutz
Artificial Intelligence and Robotics Laboratory, Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA
e-mail: jkramer3@nd.edu

M. Scheutz
e-mail: mscheutz@nd.edu

1 Introduction

Robots, unlike many software agents, operate under real-world, real-time constraints where sensors and effectors with specific physical characteristics need to be controlled. To facilitate research in autonomous robotics and help architecture designers in managing the complexity of embodied agents, several robot development environments (RDEs) have been developed that support various aspects of the agent development process, ranging from the design of an agent architecture, to its implementation on robot hardware, to executing it on the robot.

While several frameworks for comparing agent systems have been proposed, some of them specifically for RDEs (see Section 3), there is currently no survey available that (1) provides a set of conceptual RDE *features* comprehensive enough to serve as a basis for a conceptual evaluation that does justice to the specific aims with which most RDEs have been developed, (2) applies the conceptual criteria *systematically* to a large selection of RDEs, (3) augments the theoretical evaluation with a practical *usability* evaluation that includes architecture design, implementation, and execution within each RDE on a robot, with special emphasis on ease of use and performance, (4) includes the *impact* of the RDE in terms of categorized published work using it, an indicator of an RDE's prevalence in and influence on the robotics field, and (5) provides a principled way of combining the three evaluations (conceptual, practical, and impact) to obtain an

overall measure of how well an RDE can be adapted to the particular needs of researchers choosing among available systems or RDE developers considering future directions of system development.

This paper addresses all five points. Starting with a set of constraints used for the selection of RDEs to be examined (including a rationale for excluding certain RDEs), Section 2 introduces nine general purpose, freely available RDEs. Section 3 reviews previous work concerning agent system and RDE comparisons, establishing four categories of criteria corresponding to typical stages of application development for autonomous mobile robots. The RDEs are then systematically evaluated according to the criteria in Section 4. Section 5 contains a practical evaluation based on designing, implementing, and running a simple architecture and some more complex architectural components in each RDE. The subsequent discussion in Section 6 ties together the conceptual and practical evaluations and augments them with one possible evaluation of the impact of each RDE, also suggesting a principled method for using the results of this survey by both researchers and RDE developers. Section 7 summarizes the results and extrapolates to identify some future trends in RDE development.

2 Autonomous mobile robot systems

A complete accounting and systematic comparison of all RDEs is clearly impossible within the confines of a survey paper, not only because of the number of RDEs available and the release of new systems, but also due to the scope of robotics as a discipline. To make the task manageable, a group of qualifying constraints is used to limit the selection to a specific subset of representative RDEs. First, we consider only open source packages unencumbered by licensing costs and available for free download. CyberBotics *Webots* (Michel, 2004; Webots, 2005), *White Box Robotics* (WhiteBoxRobotics, 2005), and Evolution Robotics' *ERSP* (ERSP, 2004) are excluded as commercial packages. Also excluded are BERRA (Lindstrom et al., 2000) and CLARAty (Volpe et al., 2001; Nesnas et al., 2003, 2006) due to download unavailability. Systems are also required to generalize beyond specific hardware platforms, but provide more specificity than a general framework. So, while Lego *Mindstorms* (LEGO, 2005) is a widely-used robotics platform with many related packages available, we do not consider it (or projects such as *CotsBots* (Bergbreiter and Pister, 2003; CotsBots, 2005) or *Modular Controller Architecture* (MCA2, 2005)) due to specificity in relation to a single platform or custom hardware construction. Conversely, LAAS's GenoM (Fleury et al., 1997; Mallet et al., 2002; Fleury and Mallet, 2004) is excluded as a framework for the generic definition of robot components. Finally, there must be at least one cohesive application developed in the system (i.e., a repository of components is not considered for inclusion). To our knowledge, this requirement is not met by *Orocos* (Bruyninckx, 2001; OROCOS, 2005), *The Rossum Project* (Lucas, 2004), *Nomadic* (Sprouse, 2005), *Dave's Robotic Operating System* (Austin, 2004), the *Open Automation Project* (Walters, 2003), or *YARP* (Metta et al., 2006). Similarly, this excludes some projects that, at the time this research was begun, were either just being developed (e.g., Orca Brooks et al., 2005; Orca, 2005) or in a pre-release stage (e.g., the RObotics and Learning Environment (ROLE) (Heckel, 2005)).[1]

Given the above constraints, nine RDEs have been selected,[2] listed in Table 1 The following synopses give an overview of the systems' use and operation, including a broad system description, the stated purpose of the system, the platforms on which it runs, the release version, and a summary of notable features. To characterize the strengths of the systems more completely, the end of each subsection lists publications from particular robotics research subareas, determined by the presentation groupings established in the 2001–2005 *Proceedings of the IEEE International Conference on Robotics and Automation* (ICRA), which have been divided into three categories:

- *Single robot*: SLAM, Planning/Navigation, Learning, Hierarchical Behavior, and Education
- *Human-Robot Interaction*: Task Allocation, Learning, and Assistive Robotics
- *Multi-robot*: Sensing, Exploration, Mapping, Localization, Planning, Coordination, Formation, and Task Allocation

Only a single publication represents a sub-area; citations are also used in evaluating an RDE's *impact* in Section 6.

2.1 TeamBots

TeamBots (Balch, 2004; Balch and Ram, 1998) (which supersedes JavaBots) is a Java-based collection of application programs and Java packages for multi-agent mobile robotics research. Although it is no longer under active development (the most recent version available, 2.0e, was released in April 2000), it is included due to its appearance in both Jia et al. (2004) and Orebäck and Christensen (2003) and because it has found wide use in both research and education. The main author of TeamBots is now affiliated with the laboratory that develops MissionLab (see Section missionlab-desc); for the above reasons, this description will be brief.

A highly touted feature of TeamBots, stemming from a strict separation of hardware interfaces and control code, is

[1] Exclusion of the listed systems is only indicative of not meeting the specified constraints; further examination is encouraged.

[2] Almost all of the selected RDEs are under constant revision and more recent versions might be available.

Table 1 RDEs selected for this survey

RDE	Originating institution	More information
TeamBots, v.2.0e	Carnegie-Mellon University	http://www.teambots.org/
ARIA, v.2.4.1	MobileRobots, Inc	http://robots.mobilerobots.com/
Player/Stage, v.1.6.5, 1.6.2	University of Southern California	http://playerstage.sourceforge.net/
Pyro, v.4.6.0	Bryn Mawr College Swarthmore College University of Massachusetts SRI International	http://emergent.brynmawr.edu/pyro/?page = Pyro
CARMEN, v.1.1.1	Carnegie-Mellon University	http://carmen.sourceforge.net/
MissionLab, v.6.0	Georgia Institute of Technology	http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab/
ADE, v.1.0beta	University of Notre Dame	http://ade.sourceforge.net/
Miro, v.CVS-March 17, 2006	University of Ulm	http://smart.informatik.uni-ulm.de/MIRO/
MARIE, v.0.4.0	Université de Sherbrooke	http://marie.sourceforge.net/
FlowDesigner, v.0.9.0		http://flowdesigner.sourceforge.net/
RobotFlow, v.0.2.6		http://robotflow.sourceforge.net/

the use of the same control code both in simulation and for an actual robot. While the only robot platforms supported are Probotic's Cye and Nomad 150 robots, there are many example simulation environments and control systems available. The simulator was developed to be extremely flexible, supporting multiple, heterogeneous robot platforms and control systems simultaneously. In addition, the *Clay* package allows hierarchical behavior specification, specifically targeting schema-based control. Inter-robot communication is supported via sockets and serial ports only. A notable inclusion is the Java CMU-Vision package, which supports frame captures and blob-tracking.

TeamBots publications include those from the hierarchical behavior (Balch, 2000) and education (Balch, 2002) subareas.

2.2 Advanced robotics interface for applications (ARIA)

ARIA (MobileRobots, Inc., 2005; LaFary and Newton, 2005), the base software that comes packaged with the purchase of MobileRobots (*neé* ActivMedia) robots, is a set of C++ classes available for free download. At the lowest level, ARIA provides *system architecture* capacities; that is, software that describes the structure of a robot (including its sensors, effectors, and physical specifications) and implements the low-level interaction between software and hardware components. At a higher level of abstraction, it also includes some sensory interpretation functionality, basic actions (analogous to behaviors), and an elementary action resolver.

Although freely available, ARIA is a product of MobileRobots, Inc. and thus only supports MobileRobots platforms, using *robot parameter files* as the means of defining the characteristics of a robot. This includes information about the robot body (e.g., the robot's radius), the sensors (e.g., the number and position of sonar), and the effectors (e.g., the maximum velocity). The parameters are used by ARIA for various calculations (e.g., the "RobotRadius" parameter is used to determine the robot's turn limits). In support of distributed computing, ARIA provides the *ArNetworking* package as a wrapper around socket communications. In addition, the Simplified Wrapper and Interface Generator (SWIG, 2004) development tool is used to provide Java and Python support.

Supporting software is available, but is limited in some cases by licensing or purchase requirements. The *MobileSim* 2-dimensional simulator, a modified version of Player/Stage's Stage simulator (see Section 2.3), is freely available. A demo version of the ActivMedia Color Tracking Software (ACTS) is available for free download, but has restricted functionality that disallows integration with ARIA (not true of the licensed version). Additional open source software includes the *ArSpeech* components that provide interfaces to Sphinx speech recognition and both Festival and Cepstral speech production packages, *SonARNL* for sonar-based localization, *Mapper3 Basic* for map creation and editing, and *VisLib* for single camera object tracking. Also available, but restricted to license and/or purchase are *MobileEyes* (which provides a remote robot display and control GUI), *ARNL* (which provides laser-based mapping and localization), and *Mapper3* (which augments the basic mapper package with laser support and automated map creation from sensor logs).

There are no publications available from projects that have used ARIA.[3]

[3] While publications exist for ARIA's ancestral software, the authors were explicitly requested to not refer to it.

2.3 Player/stage

The Player/Stage (Gerkey et al., 2001, 2003, 2005) project is designed to be a *programming interface*, specifically avoiding being a development environment. Rather than treating a robot as the primary unit of agency, it instead focusses on *devices*, or the various sensors and effectors. A "collection" of devices is typically, although not necessarily, located on a single robot. Supported platforms include MobileRobots, RWI/iRobot's RFLEX-based, Segway, Acroname, Botrics, Evolution Robotics, and K-Team robots, while components that are packaged along with the download include vector field histogram goal-seeking/obstacle avoidance, adaptive Monte-Carlo localization, and a wavefront propagation path planner and interfaces for ACTS (see Section 2.2), CMVision, Festival, a service discovery mechanism, and others.

In this software, *Player* refers specifically to the device and server interface. Devices are independent of one another and "register" with a Player *server* to become accessible to *clients*. Each client uses a separate socket connection to a server for data transfer, thus preserving concurrent operation of devices and the servicing of multiple requests. Minimal constraints are placed on the use of devices; in a very real sense, usage is only a *communication protocol*, leaving the client the freedom (and by extension, the work) of designing and implementing a control architecture. *Stage*, which is the second part of the software, is a device simulator. Client control code uses the same programming interface when used in conjunction with either simulated or physical devices.

A stated design goal of the device model used in Player/Stage is the separation of *interface* and *function*. The fact that servers communicate via a socket interface means that client programs can be written using any language with socket support. According to Gerkey et al. (2003), currently available libraries include those written in C, C++, Tcl, Python, Java, and Common LISP. Due to the prevalent use of socket communication, the Player system inherently fits the distributed computing paradigm. Client code is able to operate on any host that has network connectivity, enabling location independence. A side effect of the device model and its networked basis is that combinations of devices can be formed to create novel types of agents (e.g., one composed of only sonar devices from many different robots). An additional feature of the device model is that the frequency of sensor and effector updates are independent, providing clients the ability to make full use of the data generated by devices that operate at a high frequency, while not being hindered by those that are slower.

Stage is a graphical, 2-dimensional simulator that models devices in a user defined environment. Specifically designed to support research in multi-robot systems through its use of socket-based communication, it also forms the foundation for ARIA's MobileSim simulator. In addition to Stage, a high-fidelity, 3-dimensional simulator called *Gazebo* is available. In both cases, client code uses the same interface on real robots as in the simulator. The authors mention that the device model makes it easy to simulate non-existent devices (for instance, a type of sonar that penetrates walls to some extent) for further research in device design and use.

Player/Stage publications include those from the SLAM (Wolf and Sukhatme, 2005), learning (Provost et al., 2004), education (Matarić, 2004), HRI task allocation (Tews et al., 2003), multi-robot sensing (Jung and Sukhatme, 2002), multi-robot exploration (Howard et al., 2002), multi-robot mapping (Howard et al., 2004), multi-robot localization (Howard et al., 2003), multi-robot planning (Howard et al., 2004), multi-robot coordination (Jones and Matarić, 2004), multi-robot formation (Fredslund and Matarić, 2002), and multi-robot task allocation (Gerkey and Matarić, 2002).

2.4 Python robotics (Pyro)

Pyro (Blank et al., 2003, to appear; Pyro, 2005) is a robot programming environment aimed at, but not limited to, educational purposes, leading to specific choices in its design. One goal is to provide a top-down approach to the design of controllers, insulating students from low-level details of implementation while preserving access to the low-level if it is desired. Some of the abstractions include: range sensors, robot units, sensor groups, motion control, and devices, which encapsulate lower levels. This includes "wrapping" Player/Stage (see Section 2.3) and ARIA (see Section 2.2) functionality, so that any component written for those systems are also available to Pyro users. A large selection of platforms are supported, including K-Team Kheperas and Hemission, MobileRobots Pioneer, Handyboard, Sony Aibo, and all robots supported by Player/Stage (see Section 2.3).

Educational modules exist to demonstrate control paradigms (e.g., neural networks, evolutionary algorithms, vision processing, and reactive, behavior-based, finite state machines, etc.). Python, an interpreted language, was chosen as the basis of the system due to its ease of use for beginning students, while permitting more knowledgeable designers to write more advanced code. While it is acknowledged that using an interpreted language leads to slower operation, the trade-off between usability and performance is consciously made. Construction of graphical visualization of robot operation are explicitly supported through the use of pre-defined facilities and Python's OpenGL interface. Another goal is to design control code that operates on many different robots with no modification. An example of this is the use of "robot units" that replace traditional measurements such as meters.

Pyro publications include those from the learning (Blank et al., 2002), education (Blank et al., to appear), and HRI task allocation (Desai and Yanco, 2005) subareas.

2.5 Carnegie mellon robot navigation toolkit (CARMEN)

CARMEN (Montemerlo et al., 2003a, b) is an open source collection of (mobile) robot control software written in the C programming language that is meant to provide a "consistent interface and a basic set of primitives for robotic research". Oriented towards single robot control, it uses a three layer agent "architecture," in which the first layer is the hardware interface, providing low-level control and integration of sensor and motion data, the second layer is concerned with basic robot tasks such as navigation, localization, object tracking, and motion planning, and the third layer is the user-defined application, which relies on the primitives of lower layers. Modularity is a primary concern, supported by the Inter-Process Communication System (IPC) communication protocol/software (discussed in more detail below). Besides supporting a number of robot platforms (including MobileRobots, Nomadic Technologies Scout and XR4000, Segway, iRobot ATRV, ATRVjr, and B21R) and navigation primitives (map-making, Monte-Carlo particle filter localization (Thrun et al., 2000), and Markov decision process path planning (Konolige, 2000)), CARMEN also provides configuration tools, a simulator, and graphical displays and editors.

IPC (Simmons, 1994, 2004) provides high-level support for connecting processes using TCP/IP sockets and sending data between processes, including opening and closing sockets, registering, sending, and receiving messages, which may be anonymous publish/subscribe or client/server type communications. The IPC library contains functions to marshall (serialize) and unmarshall (de-serialize) data, handles data transfer between machines with different Endian conventions, invoke user-defined handlers when a message is received, and invoke user-defined callbacks at set intervals. In essence, IPC performs a function similar to a naming service for components; besides providing the means to define message abstractions used for communication over a network, it also encourages extensibility (in that components are self-contained) and fault-tolerance (in that failure of a component ceases communication, but does not actively interrupt other components in the system).

CARMEN components generally take the form of a single executable, such as **pioneer** (for a MobileRobots Pioneer robot), **laser** (for a SICK laser range finder), or **localize** (for robot localization using a pre-made map). Particular platform definitions are contained in "base" specifications, which are then abstracted to a generic "robot" configuration that includes basic parameters such as body length and width, sonar offsets, maximum velocities, etc. Parameters for each component are stored in a human readable text file repository, but a graphical editor can be used to modify parameters at run-time. In addition, each component relies on a set of IPC message definitions to which other components can subscribe, allowing component distribution through an IPC server.

CARMEN publications include those from the SLAM (Thrun et al., 2000), learning (Osentoski et al., 2004), HRI assistive robotics (Pineau et al., 2002), and multi-robot coordination (Simmons et al., 2000) subareas.

2.6 MissionLab

MissionLab (MacKenzie et al., 1997; MissionLab, 2003) is a set of software tools for executing military-style plans using individual or teams of real and simulated robots. Developed as part of the DARPA Mobile Autonomous Robot Software (MARS) project, the main stated goal is to control the motion of robots in highly dynamic, unpredictable, and possibly hostile environments. Collaboration and coordination of robot teams is based on the *Societal Agent* theory, which views abstract "assemblages" of agents as agents themselves and whose behavior, in turn, is the aggregate of coordinated "primitive" behaviors of "atomic" agents. Assemblages are hierarchical, while behavior coordination is achieved through finite state automata (either competitive or temporally sequenced) or vector summation cooperation.

The *Configuration Description Language* (CDL) is used to recursively define abstract societal agents (called *configurations*), usually accomplished using the graphical *CfgEdit* tool. A configuration can be bound to a specific set of robots and devices; robot choices include MobileRobots Pioneer, iRobot ATRVjr and Urban, Evolution Robotics ER-1, and Nomad 150/200. CDL is compiled to *Configuration Network Language* (CNL) code, which is then compiled to C++ code and finally compiled to machine code, resulting in a robot executable. The executable contains a communication module (called *HClient*) to interface with an *HServer*, an abstract control interface used for all robot hardware via IPT communication software. (IPT supports distributed computing and is related to the IPC communication software, described in Section 2.5; both are derived from the *Task Control Architecture* (TCA, Simmons, 1994) project.) Developers also have the option of using the higher level *Command Description Language* (CMDL) to describe robot missions, which is a mechanism for providing high-level input to robot behaviors.

As a primary concern of MissionLab is usability, the graphical interface is quite extensive, allowing non-experts to write control code without any programming. Logging consists of writing a robot's position, velocity, heading, and the current state of the robot with respect to time to a disk file, while debugging toggles are used to display program output

to a console. A unique feature relative to other systems is the inclusion of "Motivational Variables" (anger, fear, hunger, curiosity) that simulate emotionality. Developers can also assign user-defined "Personalities" to robots. Finally, there is an extensive set of components available with Missionlab, including a case based reasoner, Q-learning, graphical behavior building tool, D* Lite planner, and human/robot interaction interfaces.

MissionLab publications include those from the learning (Arkin et al., 2003), hierarchical behavior (MacKenzie and Arkin, 1993), HRI task allocation (MacKenzie and Arkin, 1998), multi-robot planning (Endo et al., 2004), multi-robot coordination (MacKenzie et al., 1997), multi-robot formation (Balch and Arkin, 1999), and multi-robot task allocation (Arkin et al., 1999) subareas.

2.7 APOC development environment (ADE)

ADE (Andronache and Scheutz, 2004a, b; Scheutz, 2006) is a programming environment that combines (1) support for developing and implementing agent architectures with (2) the infrastructure necessary for distributing architectural components. An explicit goal is to combine features of multi-agent systems (by treating architectural components as "agents" in a MAS-sense) with those of a programming environment and toolkit for complex agent design and implementation. ADE is a Java implementation of the APOC (Activating, Processing, Observing Components) (Scheutz and Andronache, 2003; Scheutz, 2004) universal agent architecture framework, which provides arbitrary levels of (possibly hierarchical) component abstraction and interconnection. Communication among ADE components relies on Java's Remote Method Invocation (RMI) facilities. ADE provides infrastructural components for an enhanced naming service, connection mediation and monitoring, security features (access control and authentication), and the ability to store the run-time state of the system, which in turn allows for the detection and recovery from component failures.

While ADE is limited to MobileRobots robots and Arrick Robotics' Trilobot, a set of abstractions for typical robotic sensors and effectors provide the means for extending support to other platforms. Configuration files can take the form of either text or XML files and include both an abstract architecture description and/or the run-time specification of component distribution. Graphical representations of individual components exist, accessible via a distributed, multi-user GUI, which provides a view of the complete agent architecture and the means to control individual components. Logging facilities allow any component in an ADE system to write to multiple files. ADE provides several predefined components including facilities for behavior definition, vision processing, speech recognition and production, a general-purpose rule interpreter, a Prolog interface, and "wrappers" to incorporate external software. It also includes a Java implementation of a Player (see Section 2.3) client that interfaces with the Stage 2-dimensional robot simulator and other Player/Stage components, in addition to an interface to the simulator packaged with the now defunct (Saphira Konolige et al., 1997; Konolige, 2002) system.

ADE publications include those from the planning/navigation (Kramer and Scheutz, 2003), hierarchical behavior (Scheutz and Andronache, 2004), HRI task allocation (Scheutz et al., 2004), assistive robotics (Scheutz et al., 2006), multi-robot sensing (Andronache and Scheutz, 2004a), and multi-robot coordination (Scheutz, 2006).

2.8 Middleware for robots (Miro)

Miro (Utz et al., 2002; Miro, 2005) is a distributed, object oriented framework for mobile robot control that is meant to facilitate heterogeneous software integration and foster portability and maintainability of robot software. Core components have been developed in C++ for Linux based on Common Object Request Broker Architecture (CORBA) technology using the adaptive communication environment (ACE, Schmidt, 1994) as its communication framework. Due to the programming language independence of CORBA, further components can be written in any language and on any platform that provides CORBA implementations.

Miro currently supports three platforms: iRobot B21, MobileRobots Pioneer, and the custom-built Sparrow. Abstraction interfaces include odometry, motion, rangesensor (sonar, infrared, bumper, laser), stall, video, pantilt, GUI buttons, and speech. Components exchange data based on subscriptions, which allow for event driven notification. Defined messages include those for odometry, rangesensor (scanevent, groupevent, bunchevent), sonar, infrared, bumper, stall, and GUI buttons. Miro includes a "behavior engine" for reactive behavior specification, which allows hierarchical decomposition of timed, event and task behavior sets into "policies". There are two types of policy transitions, local and global, that can be edited via a graphical interface; global policies preempt behaviors, local do not. The configuration of hardware, data subscriptions, and logging specification is stored in XML files. Two types of logging are defined, "log levels" and "log categories", that allow developers to vary the granularity of log data, while a graphical LogPlayer allows the replay of logged data.

Miro publications include those from the SLAM (Kraetzschmar et al., 2004), planning/navigation (Kraetzschmar et al., 2000), learning (Fay et al., 2004), hierarchical behavior (Utz et al., 2005), HRI assistive robotics (Gassull, 2001), multi-robot sensing (Utz et al., 2004), and multi-robot coordination (Utz et al., 2004) subareas.

2.9 Mobile and autonomous robotics integration environment (MARIE)

MARIE (Côté et al., 2004, 2006; Côté, 2005) is a programming environment that is specifically designed with the integration and distribution of robot applications, components, and tools in mind. For brevity, "MARIE" is used throughout this description and the rest of the survey to signify the MARIE software **and** the related FlowDesigner (Valin and Létourneau, 2004) and RobotFlow (Michaud and Létourneau, 2004) packages (described below), unless clarification is necessary. MARIE is implemented in C++ and the integration aspect of MARIE proper uses (but is not dependent upon) the Adaptive Communication Environment (ACE, Schmidt, 1994) communication framework. Following the *mediator* design pattern (Gamma et al., 1994), MARIE provides a centralized component that connects a variety of (possibly different) software. There are four functional components: application adapters, communication adapters, communication managers, and application managers. Application adapters act as proxies between the central component and applications. Communication adapters translate data between application adapters, while communication managers create and manage the links. Finally, application managers coordinate system states and configure and control system components on any one processing node. In keeping with the aim of integrating software, components have been developed for Player/Stage (see Section 2.3), CARMEN (see Section 2.2), and ARIA (see Section 2.2).

FlowDesigner is a data-flow processing library coupled with a graphical display that allows developers to create reusable "software blocks" linked together in a (possibly hierarchical) network. Available libraries include support for signal processing, audio processing (DSP), vector quantization, neural networks, fuzzy logic, an Octave plug-in, and RobotFlow. RobotFlow is a mobile robotics toolkit for FlowDesigner that includes support for MobileRobots Pioneer2 robots and other hardware devices, behaviors, finite state machines, vision processing (color training, tracking, etc.) and the interfaces for use with MARIE.

MARIE publications include those from the planning/navigation (Beaudry et al., 2005), education (Michaud, 2005), HRI assistive robotics (Labonté et al., 2005), multi-robot localization (Rivard, 2005), multi-robot coordination (Guilbert et al., 2003), and multi-robot formation (Lemay et al., 2004) subareas.

3 A conceptual framework for comparing RDEs

Several comparisons of agent systems and agent development environments have been proposed in the recent literature. For software agents, they are typically concerned with various aspects of multi-agent systems (MAS), including comparing *agent platforms* (Altmann et al., 2001; Nguyen et al., 2002; Laukkanen, 1999; Nowostawski et al., 2000; Ricordel and Demazeau, 2000), *agent development kits* (Bitting et al., 2003), *mobile agent systems* (Silva et al., 2001), or *agent environments* (Eiter and Mascardi, 2002). There are also comparisons of *general agent systems* and *agent architectures per se* (Sloman, 1998; Logan, 1998; Sloman and Scheutz, 2002). Comparisons that concern robotic agents in particular have addressed *mobile robotic architectures* (Orebäck and Christensen, 2003) and *robot programming environments* (MacDonald et al., 2003; Jia et al., 2004; Biggs and MacDonald, 2003). Common to all is the need to establish an appropriate set of *criteria* that serves as a basis for the comparison. Clearly, the choice of criteria is critical, for, as pointed out in Ricordel and Demazeau (2000), "any criteria is relevant to a specific outside need."

We briefly review some of this prior work to situate our proposed evaluation criteria, giving a general overview of the conceptual breakdown in each and why each proves insufficient for the purposes of this paper. To avoid ambiguities and equivocations among the different terms used, we will adhere to the following terminology for the rest of this paper:

- *Platform:* the hardware on which an application will be executing; this includes the sensors, actuators, computers, operating system(s), and other hardware or software intimately tied to hardware.
- *Component:* a functionally independent part of an agent or system.
- *Architecture:* the structure and interaction of components; if necessary, a distinction will be made between system and agent architectures.
- *Agent:* the sum of the software and hardware required for an individual robot to perform its task. In particular, we will not consider infrastructure or strictly software agents (e.g., a *naming service* or *communication agent*) as agents *per se*, as is done in the field of multi-agent systems. These are instead considered functional *components* that are part of the broader environment or application.
- *Programming environment:* the tools, infrastructure, and components that are not left for implementation by the developer. The term *system* will be used interchangeably in this context.

The most general and, for our purposes, pertinent, framework is Eiter and Mascardi (2002). Although founded in MAS research, the classification is intended to be comprehensive, establishing a framework for all types of agent systems. Additionally, the authors provide a practical method for *choosing* an appropriate system for a task selected by an application designer. Criteria are divided into five categories: (1) *agent attitudes*, (2) *software engineering support*, (3) *implementation concerns*, (4) *technical issues*, and (5)

economical aspects.[4] While the *software engineering*, *implementation*, and *technical issues* categories usually have a prominent role in discussions of RDEs, the *agent attitudes* aspect is often omitted–not because it is unimportant or ignored, but because features therein often form the task, or object, of investigation. Yet, according to Eiter, the attitudes category is comprised of features that discriminate between agent and non-agent software: they are either *basic* (i.e., "close to the very core of agenthood") or *advanced* (i.e., "desirable but not of central interest"). Hence, an agent development environment (and by extension an RDE) should, at least in part, be evaluated with respect to the degree to which it supports these attitudes. While Eiter and Mascardi's categorization is comprehensive, it lacks some details of considerable importance for evaluating RDEs. In fact, this is explicitly acknowledged with the disclaimer, "other features and criteria should be taken into account" for the unique issues that arise in the development of *physical agents* (e.g., support for devices, real-time operation, etc.).

In their framework proposal, Jia et al. (2004) isolate three high-level categories for analysis of an RDE: (1) *openness*, (2) *abstraction*, and (3) *modularity*. *Openness* refers to extensibility: a programming environment should support the addition and evolution of hardware and software. *Abstraction* forms the basis on which *openness* is built, providing a well-defined application programming interface (API) that allows a developer to work at a level beyond the hardware (see also Vaughan et al., 2003). Different from *abstraction*, which is focussed on hardware, *modularity* concerns software, promoting good design and reusability. While these three categories address the design and implementation of autonomous mobile robotic applications (as demonstrated by their in-house development of the *Frontier-1* robot), they are too general to address specific concerns of RDEs (e.g., real-time support, hardware-dependence of a robotic platform, debugging tools, etc.).

MacDonald et al. (2003) give a detailed and comprehensive description of RDE features in three categories: (1) *robot programming* (both at the system and task level, which enable programmers to describe robot behavior), (2) *infrastructure* (which supports the execution of behavior descriptions), and (3) *human-robot interaction* (HRI, which allows interaction with the robot programming area; see also Biggs and MacDonald, 2003). The proposed features will be largely included in our comparison, but there are some issues concerning the analysis, organization, and application to various aspects of RDEs. For one, the boundaries of the categories overlap to such an extent as to be unclear. For instance, *infrastructure* conflates the facilities provided by the environment with both the *programming* and the *agent architecture* categories. Similarly, the broad scope of the *HRI* (human/robot interaction) category largely overlaps the robot programming category, yet contains individual features that are too specific for a general system comparison (i.e., excluding systems that are not especially intended nor designed for HRI). Moreover, the proposed categorization is not structured in a way that is easily amenable to a systematic comparison (e.g., conceptually different items are subsumed under the rubric "robot programming").

The study closest in intent to this survey is Orebäck and Christensen (2003), which attempts to establish the characteristics of a "common software architecture" for mobile robot systems. In particular, seven categories (*hardware abstraction*, *scalability*, *overhead*, *control model*, *software*, *tools and methods*, and *documentation*) are proposed as a basis for comparing RDEs, covering an extensive range of features. However, while the proposed framework is generally suitable, the actual comparison is limited to only three RDEs (Balch, 2004; Konolige et al., 1997), and BERRA (Lindstrom et al., 2000)) and does not adhere strictly to the conceptual framework. Rather, criteria are grouped into six areas that mostly, but not always, correspond to the categories as defined, in some instances leaving out or introducing new criteria.

While all of the above studies agree that the main purpose of an RDE is to provide appropriate tools and abstractions that help the agent designer, they fail to provide a comprehensive, yet succinct conceptual framework that allows for a systematic comparison of RDEs. Based on the three typical stages in the development process of a robotic agent architecture[5] (*design*, *implementation*, and *execution*), we propose four categories of criteria for RDE comparison, categorized in terms of:

F1: *Specification*, which includes formalisms, methodologies, and design tools,
F2: *Platform support*, which is related to the hardware and its low-level interface (e.g., the operating system),
F3: *Infrastructure*, which refers to components and capabilities that are part of the RDE, but not the "agent architecture proper," and
F4: *Implementation*, which includes aspects of application development (including predefined components used in an agent architecture).

Of the four categories, three reflect features relevant to specific development stages (e.g., *specification* features are

[4] Eiter's *economical aspects* category will not be considered here, except for the *documentation* criterion, as the selected RDEs are both open source and research-oriented. Related considerations, such as the cost of application development, RDE maintenance or modification, training, etc. are, however, addressed by the usability evaluation in Section 5.

[5] Our stages are similar to Ricordel and Demazeau (2000), although we subsume the *analysis* category as part of the *design stage*.

central to *design*, *implementation* features pertain to implementation, as the name suggests, and *platform* features play a role in the execution). The fourth category, *infrastructure*, is added to explicitly distinguish aspects of an RDE that are separate and distinct from the agent architecture (e.g., distribution mechanisms that are integral to system operation, yet usually transparent to the agent designer).

We note in advance that the four categories are comprised of features that an RDE objectively has or does not have, with an emphasis on the software engineering aspects of its functional characteristics and capabilities. These criteria alone are not sufficient for a full evaluation of an RDE and are supplemented with additional criteria in Sections 5 and 6. The different types of evaluation can be distinguished by an identifying prefix; criteria in this section are denoted by **F**, followed by a category and item number. It is crucial to note that even though the expanded criteria list provides a comprehensive foundation for RDE evaluation, it is impossible in principle to address every concern a developer might have. A remedy for the situation is discussed in Section 6.

F1: Specification

The specification of a robotic agent or application occurs in the design stage and concerns issues such as the application domain(s), software engineering, and determination of an appropriate agent architecture. To preserve the focus on RDEs, the criteria presented are somewhat broad, but are sufficient to address the prevailing concerns.

F1.1: *Architectural Primitives.* An RDE provides various types of predefined functional component and/or knowledge primitives useful in robotic applications (e.g., behaviors, methods of control, tasks, objects, etc.), or the means to create, organize, and manipulate them.

F1.2: *Software engineering.* Software engineering support promotes the creation of high-quality software. Enabling modularization and code reuse, it can be accomplished through the use of stated design principles, explicit frameworks or tools, methodologies, or formalisms, and includes application verification, prototyping, and the abstractions mentioned in Jia et al. (2004), Orebäck and Christensen (2003) and Vaughan et al. (2003).

F1.3: *Architecture neutrality.* An RDE may be associated with a particular theoretical foundation that promotes a specific agent/application architecture, separate from implementational concerns. Alternatively, it may be *architecture neutral*, leaving the choice to the designer or even providing the means to compare application implementations using different agent architectures.

F2: Platform support

Robotic applications necessarily incorporate real-world sensors and effectors; thus, they require a more diverse set of hardware than software-only systems. The principles of *abstraction*, *modularity*, and *openness*, as put forth in Jia et al. (2004), are of particular importance to this category, promoting application use across varying platforms.

F2.1: *Operating system.* An RDE may be compatible with one or many operating systems, but must be compatible with the designer's choice. This can become a major obstacle when certain libraries or components are implemented only for a particular operating system.

F2.2: *Hardware support.* "Hardware support" refers to the variety of sensors and effectors that are available in an RDE, such as cameras, sonar, and laser devices. Since the number of standard (that is, common and non-custom) devices is limited and widely used on different platforms, we will refer instead to particular robot manufacturers. In support of increased modularity, ease of device modification, and addition of custom devices, a hierarchy of device abstraction is often specified, allowing control code to be easily ported and executed on different robots.

F2.3: *Simulator.* Simulation of the physical world allows developers to test applications, model currently unavailable hardware, and replay actual application execution. Simulators can be low- or high-fidelity, approximating an environment to some lesser or greater degree, and can also be two- or three-dimensional. Some simulators have the ability to include multiple robots in a single simulation or to mix real and simulated robots in an environment.

F2.4: *Configuration method.* The configuration of a robot is often changed to meet the demands of various applications. This information may be incorporated into the source code (requiring compilation to effect changes) or in configuration files that can be easily modified, either with a text editor or a graphical interface.

F3: Infrastructure

Infrastructure refers to RDE functionality that affects multiple components (or the system as a whole) and is not tailored to individual architectural components, application domains, or particular stages of application/agent development. For example, *logging facilities* can be used with any or all components, are often invaluable as debugging tools during the implementation stage, and provide records of an execution instance for later performance analysis. In some

cases, however, it may be impossible to determine whether a feature is due to a specific component or part of the infrastructure by function alone. For instance, the graphical representation of components might be implemented on an *ad hoc* basis, removing it from consideration as infrastructure. A system must provide generic mechanisms that supply these capabilities for them to be considered as infrastructure.

F3.1: *Low-level communication*. Inter-process communication (such as memory mapping, pipes, or sockets), basic networking protocols (such as UDP, TCP/IP, etc.), and mid-level protocols (such as IIOP or RMI) are part of the system infrastructure. These capabilities are often dictated by the platform being used, as their availability is contingent on the operating system and/or programming language.

F3.2: *Logging facilities*. Log files of application operation can be used for debugging, repetition of an application execution, or gathering performance statistics. Logging mechanisms can have various levels of flexibility, including fixed (which generally captures all data produced by components) vs. configurable data content, local vs. remote logging, file name selection, single vs. multiple data streams and/or files, or the ability to start and stop logging at run-time.

F3.3: *Debugging facilities*. While logging facilities can suffice for basic debugging, robust debugging tools can be invaluable during application implementation. Such tools can range from low-level code editors, to mid-level graphical representations of sensors and effectors, to high-level graphical behavior or task modification, possibly allowing run-time suspension, modification, and restarting.

F3.4: *Distribution mechanisms*. Distribution mechanisms, as part of the infrastructure, are required for multi-host applications. Typically, distribution capabilities are enabled by middleware (e.g., Poggi et al., 2002), either as a generic component implementation framework (such as CORBA, 2005, SOAP, 2003, etc.) or in the form of particular components (such as an *agent naming service*, *directory facilitator*, *broker agents*, or other components that provide similar functionality).

F3.5: *Scalability.* As robotic applications grow in scope and capabilities, a developer must be concerned with how an RDE handles increasing complexity. The term "scalability" can refer to many different aspects of a system, some of which are addressed more specifically by other criteria. For instance, *architectural primitives* (criteria F1.1) and a *high-level language* (F4.1.2) includes facilities for managing complex actions and behaviors, *software engineering* (F1.2) takes into account modularization that promotes system organization, while *distribution mechanisms* (F3.4) encompasses mechanisms used to add computational hosts. Additional concerns might include the overhead involved with message passing, both within a single host and among connected hosts, task allocation for multi-robot applications, or other concerns. Scalability is used here in a broad sense as a general system property, inclusive of the above.

F3.6: *Component mobility.* "Mobility" refers to the potential to relocate components at run-time. In robotic applications, however, it is somewhat constrained by possible dependence on the location of the requisite hardware. When an application is distributed across many hosts, component mobility can be used for dynamic resource allocation or run-time system reconfiguration, assuming there are mechanisms that allow reconnection to data sources. Ultimately, these capabilities would be automatic, adjusting operation with a changing computing environment.

F3.7: *System monitor/management*. A system monitor displays the status of multiple application components, often in graphical form. An extension of simple monitoring can allow for the management of the components' operation, ranging from starting and stopping to adjustment of parameters. Such extensions are often implemented as part of individual components, which are treated separately as an *implementation characteristic* in Subsection crit-impl-char and do not qualify as part of the infrastructure.

F3.8: *Security.* An application executing on a single robot may not need any security mechanism, but distribution across many hosts raises such concerns. Predefined components for encryption, authentication, and access control can be available for ready integration into applications. (A related discussion of security concerns in the multi-agent system RETSINA can be found in Singh and Sycara (2004).)

F3.9: *Fault-tolerance*. Repeated failures of both hardware and software are common in robotic applications. The system infrastructure may incorporate generic mechanisms for *failure detection*, or be structured such that disruptions due to failed components do not halt the entire application. Extending this concept, mechanisms for *failure recovery* may exist that enable components to automatically recover from failures with no outside intervention (for instance, see Melchior and Smart, 2004).

F4: Implementation

In practice, an important reason for selecting a particular RDE is to facilitate the implementation of an agent architecture. We subdivide implementation features into two areas:

(1) *implementation characteristics*, which are somewhat abstract and refer to implementation concerns that are not predefined components, and (2) *predefined components*, which perform some specific function that can be directly incorporated into an architecture.

F4.1: Implementation characteristics

F4.1.1: *Programming language*. Architecture implementation necessitates the use of programming languages, such as C or Java. An RDE that is itself implemented in the particular language used for the application guarantees compatibility; however, an RDE may also supply interfaces or wrappers that interface easily with other languages.

F4.1.2: *High-level language*. Some programming environments integrate higher-level languages, such as The Behavior Language (Brooks, 1990), COLBERT (Konolige, 1997), or GRL (Horswill, 2000) for behavior description or ACL (FIPA-ACL, 2002) or KQML (Mayfield et al., 1996) for agent communication. These high level languages can be used within an agent architecture (e.g., to facilitate data transfer between components) or in multi-robot applications.

F4.1.3: *Documentation*. The usability of an RDE is greatly enhanced by the inclusion of well-documented code and user manuals that may include the system's API specification, answers to frequently asked questions, trouble-shooting guides, instructions concerning custom extensions, etc.

F4.1.4: *Real-time operation*. Real-time constraints are often critical in designing and operating robot architectures. Real-time capabilities of an RDE are generally dependent on the operating system and/or programming language.

F4.1.5: *Graphical interface*. An RDE may supply preimplemented graphical interfaces that enhance individual component visualization during application execution, including displays related to various sensors, effectors, behaviors, robot control, navigational plans, etc. Additionally, RDEs may define a standardized method of adding such displays.

F4.1.6: *Software integration*. RDEs may provide tools that facilitate the integration of external software, either at the component level (e.g., a localization routine) or a complete application-as-component (e.g., speech production), greatly enhancing development time and effort. A notable development in this area is "wrappers" for components of other robotic systems that promote the integration, sharing, and reuse of components.

F4.2: Predefined components

Predefined components are analogous to software libraries; since the list is open-ended and will most assuredly expand in the future, we deviate from the format used thus far and give a necessarily incomplete list of common components with corresponding citations. Furthermore, the list assumes a fairly high-level viewpoint, necessary to maintain an acceptable level of commonality among systems.

Currently, most RDEs include predefined components for *map-making* (**F4.2.1**), *localization* (**F4.2.2**, e.g., Thrun, 2003), *route planning* (**F4.2.3**, e.g., Konolige, 2000), *speech recognition* (**F4.2.4**), *speech production* (**F4.2.5**), and *vision processing* (**F4.2.6**, with various capabilities such as blob tracking, edge detection, motion tracking, etc.). Some less common components are *rule interpreters* (**F4.2.7**, e.g., JESS, 2003 or Sloman, 2002), *planners* (**F4.2.8**, e.g., Maes, 1990; Jensen and Veloso, 1998; Stentz, 2002), *neural networks* (**F4.2.9**, e.g., Koker et al., 2004), and *machine learning* (**F4.2.10**, e.g., Vijayakumar et al., 2002; Russell, 2004). Even less common, and therefore not included in the evaluation criteria, are support for *instruction/teaching* (e.g., Skubic and Volz, 1998; Bentivegna and Atkeson, 2002), *human robot interaction facilities* (e.g., Fong et al., 2003), *affect* (e.g., Pfeifer, 1988; Moshkina and Arkin, 2003; Scheutz et al., 2006), and *coordination mechanisms* (e.g., Hoff and Bekey, 1995; Chaimowicz et al., 2003; Dias and Stentz, 2003).

4 RDE feature criteria evaluations

For each of the RDEs in Section 2, a value has been assigned for the criteria from Section 3, determined using the system's documentation and verified based on usage experience (a synopsis of experimental implementations and the usability evaluation is provided in Section 5). Three types of assignments are made: (1) *binary*, signified by a blank for *no* and ✓ for *yes*, (2) *ternary*, signified by □ for *not supported*, ⊟ for *partially supported*, and ⊞ for *well supported*, and (3) *listings*, which are text descriptions. Table 2 shows the values assigned to each system for each criteria, while further explanation is given in the text. The following shorthand column headings are used to designate particular systems: **TB**–TeamBots, **AR**–ARIA, **P/S**–Player/Stage, **Py**–Pyro, **C**–CARMEN, **ML**–MissionLab, **AD**–ADE, **Mi**–Miro, and **MA**–MARIE.

F1: Specification

F1.1 *Architectural primitives:* To attain a *somewhat supported* value, a system must provide at least one form of robot control. Systems that provide additional,

Table 2 Feature criteria evaluation by RDE

Category	Criteria	TB	AR	P/S	Py	C	ML	AD	Mi	MA
Specification F1	F1.1 Architectural primitives	⊞	⊟	□	⊞	⊟	⊞	⊞	⊞	⊞
	F1.2 Software engineering	⊟	⊟	⊟	⊟	⊟	⊞	⊞	⊟	⊟
	F1.3 Architecture neutrality	√	√	√	√	√	√	√	√	√
Platform F2	F2.1 Operating system	J	U,W	U,W	U,W	U	U	J	U,W	U
	F2.2 Hardware support	□	□	⊞	⊞	⊞	⊟	□	⊟	⊞
	F2.3 Simulator	⊟	⊟	⊞	⊞	⊟	⊟	⊟	⊟	⊞
	F2.4 Configuration method	□	⊟	⊟	⊟	⊞	⊞	⊞	⊟	⊞
Infrastructure F3	F3.1 Low-level communication	S	S	S	S	I	I	R	C	S
	F3.2 Logging facilities	□	⊞	⊞	⊟	⊞	⊟	⊞	⊞	⊟
	F3.3 Debugging facilities	□	⊟	⊞	⊞	⊞	⊞	⊞	⊞	⊞
	F3.4 Distribution mechanisms	□	⊟	⊟	□	⊟	⊟	⊞	⊞	⊞
	F3.5 Scalability	□	⊟	□	□	⊟	⊞	⊞	⊟	⊟
	F3.6 Component mobility	□	□	⊟	□	⊟	⊟	⊞	⊟	⊟
	F3.7 Monitoring/Management	□	□	⊞	⊞	□	⊞	⊞	□	⊞
	F3.8 Security	□	⊟	□	□	□	□	⊞	□	□
	F3.9 Fault-tolerance	□	□	⊟	□	⊟	⊟	⊞	⊟	⊟
Implementation F4	F4.1.1 Programming language	Java	C++	C++	Pyth	C	C++	Java	C++	C++
	F4.1.2 High-level language	√	√		√		√		√	√
	F4.1.3 Documentation	□	⊞	⊞	⊞	⊞	⊞	⊟	□	⊟
	F4.1.4 Real-time operation									
	F4.1.5 Graphical interface	□	□	⊞	⊞	⊟	⊟	⊞	⊟	⊞
	F4.1.6 Software integration	□	□	⊟	⊞	⊟	□	⊟	⊟	⊞
	F4.2.1 Map-making		√	√	√	√		√		√
	F4.2.2 Localization	√	√	√	√	√		√	√	√
	F4.2.3 Route planning	√	√	√		√	√	√		√
	F4.2.4 Speech recognition		√	√	√			√	√	
	F4.2.5 Speech production		√	√	√			√	√	√
	F4.2.6 Vision processing	√	√	√	√		√	√	√	√
	F4.2.7 Rule interpreters						√	√		√
	F4.2.8 Planners						√			√
	F4.2.9 Neural networks				√				√	√
	F4.2.10 Learning	√			√		√			

likely more complex, methods of robot control receive a *well supported* value. Player/Stage does not provide any predefined control methods, following their policy of providing only the framework for implementing robot control and so receives a *not supported* value. ARIA provides a set of basic actions and an elementary priority-based action resolver. CARMEN provides a Markov decision process planner as part of

its navigation component. Each receives a *somewhat supported* value. The rest of the systems are considered *well supported*. TeamBots provides schema-based motor control, finite state machine (FSM) sequencing, and hierarchical behaviors via the *Clay* behavior configuration system. Pyro provides both subsumption and fuzzy blending of behaviors, while MissionLab provides schema-based control, behavior sequencing and artifacts. ADE provides access to a general-purpose rule interpreter, schema-based and subsumption-based behavior primitives, a Prolog interface, and a distributed neural-network style component model based on APOC (Scheutz and Andronache, 2003). Miro includes a custom "behavior engine", based on that introduced in Brooks (1991). MARIE, via the RobotFlow and FlowDesigner packages, provides hidden Markov models, fuzzy blending, FSMs, an interface to Octave software, and other primitives.

F1.2 *Software engineering:* To attain a *somewhat supported* mark, an RDE must explicitly state design principles, be implemented using an object oriented programming language (e.g., C++ or Java), or make use of a high-level object language (e.g., CORBA). An explicit theoretical foundation yields a *well supported* mark. TeamBots, Player/Stage, ARIA, CARMEN, Pyro, MARIE, and Miro are of the former type, while the use of *Societal Agent* theory in MissionLab and the APOC formalism in ADE provides the basis for receiving a *well supported* value.

F1.3 *Architecture neutrality:* All the systems under consideration are neutral with regard to agent architectures, although MissionLab has a strong association with the AuRA architecture (Arkin and Balch, 1997) and CARMEN has been described by its authors as an example of the 3T hybrid architecture (Montemerlo et al., 2003b). However, neither enforces the use of the associated architecture, and can therefore be considered agent architecture neutral.

F2: Platform support

F2.1 *Operating system:* Compatible operating systems have been determined according to information from system documentation and do not necessarily discount those not listed. If a system, such as TeamBots or ADE, is implemented in Java, the assumption is made that it will execute on any computer platform for which a Java Virtual Machine of the required type is implemented. Similarly, no differentiation is made among the various "flavors" of UNIX, although each system has at least been tested in a Linux environment. Player/Stage, ARIA, and Pyro run on both UNIX and Windows. CARMEN, MissionLab, Miro, and MARIE run on UNIX systems. Letter codes in Table 2 are as follows: J = Java, U = UNIX, W = Windows.

F2.2 *Hardware support:* Hardware support, as used here, refers to specific robot manufacturers/platforms. We assume a relatively limited pool of sensors and effectors are used across platforms (such as SICK LMS lasers), although all systems allow specification of custom sensors and/or effectors. To attain a *somewhat supported* value, a system must support at least three different platforms; more than five earns a *well supported* value. ARIA supports only MobileRobots robots, ADE supports MobileRobots and Arrick Trilobot, TeamBots supports Cye and Nomad 150 robots, Miro supports MobileRobots, iRobot B21, and their in-house Sparrow platforms, MissionLab supports MobileRobots, iRobot, Evolution Robotics ER-1, and Nomad robots, MARIE supports MobileRobots platforms natively, in addition to all platforms available through ARIA, Player/Stage, and CARMEN via its adapters, Player/Stage supports MobileRobots, iRobot, Segway, Acroname, Botrics, Evolution Robotics, and K-Team platforms, CARMEN supports MobileRobots, Aibo, Nomadics, iRobot, and Segway platforms, and Pyro supports MobileRobots, Aibo, Cye, iRobot, Khepera, Nomad, and Segway platforms.

F2.3 *Simulator:* To attain a *somewhat supported* value, an RDE must at least provide a low-fidelity, 2-dimensional simulator (which may or may not support multi-robot simulations). To attain a *well supported* value, an RDE must provide a high-fidelity, 3-dimensional simulator that supports multi-robot simulations and may be used to model robotic mechanisms. CARMEN includes a 2-dimensional simulator that supports low-fidelity single-robot simulation that can, with some manipulation of the IPC communications, be used for multi-robot simulations. TeamBots, ARIA, MissionLab and ADE provide low-fidelity, multi-robot simulators. MissionLab also supplies a low-fidelity 3-dimensional simulator (although the manual states that its use will halt the system). A major component of Player/Stage, as indicated by its name, is the Stage 2-dimensional simulator, which is low-fidelity and supports multiple robots. Also available is the *Gazebo* high-fidelity 3-dimensional simulator, which elevates Player/Stage to *fully supported* status, along with Pyro and MARIE, which provide interfaces to Stage, Gazebo, and the ARIA and CARMEN simulators.

F2.4 *Configuration method:* A system, such as TeamBots, that embeds configuration in source code has a *not supported* status. If a system stores configuration in a text file (possibly XML), it receives a *somewhat supported* value. Player/Stage, ARIA, Pyro, and Miro all use text files, of which Miro supports XML. A

system that provides a graphical means of accessing and modifying configuration settings gets a *well supported* value, which includes CARMEN, MissionLab, and ADE. MARIE also receives a *well supported* value due to the graphical interfaces included with FlowDesigner and RobotFlow; MARIE itself uses XML configuration files.

F3: Infrastructure

F3.1 *Low-level communication:* All systems considered provide socket support and common networking protocols. TeamBots, Player/Stage, ARIA, and Pyro use direct socket connections as their primary method of communication. CARMEN and MissionLab use IPC and IPT, respectively, which adds a level of abstraction to general TCP/IP sockets. ADE uses Java's RMI, while Miro relies on CORBA's IIOP. MARIE makes use of shared memory or sockets, relying on ACE for the latter. Letter codes in Table 2 are as follows: S = Socket, I = IPC/IPT, R = RMI, C = CORBA IIOP.

F3.2 *Logging facilities:* All systems provide some means of monitoring component operation as console output or graphical display, which forms the baseline for the value assignment (i.e., a *not supported* value). To gain a *somewhat supported* value, at the very least a system must supply a predefined logging facility; to gain a *well supported* value, a system must allow for remote data capture, run-time starting and stopping of logging, and dynamically configurable data capture that can be recorded in one or more files in one or more locations. TeamBots provides only simple console/graphical output. Logging in ARIA, Player/Stage, CARMEN, ADE, and Miro is *well supported*, while logging in Pyro, MissionLab, and MARIE is *somewhat supported*.

F3.3 *Debugging facilities:* To attain a *somewhat supported* status, a system must allow non-simulated application interruption and restart in conjunction with the ability to obtain information about component data. To qualify as *well supported*, a system must allow run-time suspension, modification, and replacement of arbitrary components. TeamBots is the only RDE receiving a *not supported* value. ARIA receives a *somewhat supported* value. Player/Stage, Pyro, CARMEN, MissionLab, ADE, MARIE, and Miro all qualify as *well supported*, but MissionLab and MARIE excel due to their integrated and extensive graphical interfaces. MissionLab is unique in that it also uses the included case-based reasoner to analyze a "mission" after completion to identify the source of operational errors. Also notable is ADE's ability to dynamically compile and replace components at run-time using the ADE class loader.

F3.4 *Distribution mechanisms:* To be elevated from a *not supported* to *somewhat supported* value, a system must include a component that functions as middleware. To qualify as *well supported*, an agent framework that treats components as independent agents is required. Neither TeamBots nor Pyro provide middleware mechanisms and receive a *not supported*. The IPC and IPT software used by CARMEN and MissionLab and the Player server in Player/Stage act as a centralized naming service, while the *ArNetworking* package provided in ARIA fills a similar function. ADE, MARIE, and Miro each specifically incorporate enhanced middleware functionality as part of their infrastructure, earning a *well supported* value.

F3.5 *Scalability:* To use "scalability" as a general reflection of an RDE's properties, a combination of criteria from categories F1, F3, and F4 (specification, infrastructure, and implementation) is used. To earn a *well supported* value, a system must provide scalability support in all categories (as defined below); a *somewhat supported* value indicates support in any two categories, while support for a single category or none at all receives a *not supported* value. In the specification category, an RDE must have a *well supported* value for either *architectural primitives* or *software engineering* (criteria F1.1 or F1.2). For support in the infrastructure category, a system must earn at least a *somewhat supported* value for *distribution mechanisms* (F3.4), while the implementation category is comprised of satisfaction of at least one of *high-level language*, *rule interpreters*, or *planners* (F4.1.2, F4.2.7, and F4.2.8, respectively).

F3.6 *Component mobility:* To receive a *somewhat supported* value, an RDE must provide architectural components to operate independently of one another in addition to continuing system operation when a component is removed, restarted, and reconnects. To attain a *well supported* value, mechanisms must be in place that can perform this task automatically. TeamBots, ARIA, and Pyro all use a fixed run-time system architecture that does not allow mobility and so receive a *not supported* value. The portability of devices in Player/Stage allows manual component relocation at run-time, as do the modules in CARMEN and the object structure found in MARIE and Miro, while MissionLab provides mechanisms to upload robot executables to remote hosts. Each of these systems receives a *somewhat supported* status. ADE provides mechanisms for saving state, automatic component start-up, and automatic component re-location due to detected failures, earning a *well supported* value.

F3.7 *System monitoring/management:* To gain *somewhat supported* status, an RDE must provide an interface

that gives access to all components in the system architecture. To gain *well supported* status, an RDE must also provide mechanisms to manage all components. (Note that graphical representations of a robot's sensors, effectors, or other individual components do not qualify as *infrastructure*; see the *Graphical Interface* in Section 4). None of TeamBots, Miro, nor CARMEN provide coherent system-wide facilities. ARIA supplies the MobileEyes GUI for robot display and control, but source code is not freely available and thus earns a *not supported* status. Pyro, Player/Stage, MissionLab, ADE, and MARIE all have graphical interfaces that not only display component status, but also allow component control.

F3.8 *Security:* To gain a *somewhat supported* value, a system must provide a way to securely authenticate components. To gain a *well supported* value, an RDE must also provide access control and encryption. None of TeamBots, Player/Stage,[6] Pyro, CARMEN, MissionLab, Miro, or MARIE use security mechanisms. MARIE and Miro both might inherit security features from their use of ACE for component communication, but do not exploit its availability. ARIA provides authentication services as part of the ArNetworking package, earning a *somewhat supported* value. ADE explicitly addresses all three aspects of security (encryption, authentication, and access control).

F3.9 *Fault-tolerance:* To achieve *somewhat supported* status, a system must isolate components such that failure of a single component does not cause the entire application to fail. To receive *well supported* status, an RDE must also provide mechanisms in support of failure recovery. None of the TeamBots, ARIA, or Pyro RDEs provide component isolation. Player/Stage, CARMEN, and MissionLab, through their reliance on IPC software, each isolates components, while MARIE and Miro's use of ACE objects serve the same purpose. It is worth noting that MissionLab also incorporates a case-based reasoning wizard for the purpose of repairing a mission post-execution (Moshkina et al., 2006) and that ACE implements the Fault Tolerant CORBA specification, although neither MARIE nor Miro have yet incorporated it. ADE\ provides both fault detection and fault recovery at the component level through its use of heartbeats between `ADEServers` and clients.

[6] While the Player server in Player/Stage can optionally be set to require authentication, it is explicitly acknowledged that the authentication is *not* for security, as keys are passed in plain text.

F4: Implementation

F4.1: System implementation characteristics

F4.1.1 *Programming language:* Both TeamBots and ADE are written in Java, while CARMEN and Pyro are implemented in C and Python, respectively. Player/Stage, ARIA, MissionLab, MARIE, and Miro are implemented in C++.

F4.1.2 *High-level language:* To qualify as supporting a high-level language, an RDE must supply a structured method for controlling a robot (e.g., a behavior or agent communication language). TeamBots supplies the Clay behavior hierarchy, ARIA provides action specification via the *ArAction* class, MissionLab provides both CDL and CMDL, Pyro and MARIE supply both a set of foundational behavior classes and finite state automata, and Miro provides a "behavior engine" for behavior specification. None of Player/Stage, CARMEN, nor ADE\ provide a high-level language.

F4.1.3 *Documentation:* To attain a *somewhat supported* value, an RDE must have well documented source code and a publication outlining its use. If an RDE also supplies a manual that describes how to use the system (including installation instructions, guidelines for developing applications and extending capabilities into new areas, solutions to common problems, and example code), it receives a *well supported* value. While TeamBots, ADE, Miro, and MARIE all provide some level of documentation, both web-based and in source code, it is either incomplete or they do not provide finished manuals that detail their use. Player/Stage, ARIA, CARMEN, and MissionLab all have complete and detailed manuals available, while Pyro provides the equivalent through its extensive online documentation.

F4.1.4 *Real-time operation:* None of the systems directly provide real-time support, although MissionLab has the mechanisms in place for use with a purchased license of proprietary software from Honeywell.

F4.1.5 *Graphical interface:* To obtain a *somewhat supported* value, an RDE must supply graphical interfaces for visualizing component operation or designing control code without actual programming. To receive a *well supported* value, an RDE must provide both items just mentioned, in addition to a standard method for creating new displays. Only TeamBots does not provide a graphical display for a robot at run-time (although it does supply a graphical simulator facility), and so receives a *not supported* value. While the MobileEyes GUI is available with ARIA,

source code is not freely available and thus has to be classified as *not supported*. CARMEN provides *ad hoc*, component specific graphical interfaces, but does not provide standard methods for adding visualization nor graphical control code tools, and so receives a *somewhat supported* value. MissionLab and Miro provide interfaces that allow developers to design control code without actual programming, but do not provide a standard method for defining new displays, also earning them a *somewhat supported* value. Player/Stage, ADE, Pyro, and MARIE all provide both implemented displays and a standardized method of creating new displays, earning *well supported* values.

F4.1.6 *Software integration:* To attain *somewhat supported* status, an RDE must provide a standard API or mechanism for incorporating "outside" software, generally using socket connections (with the recognition that translation code will always have to be written). Providing additional codified facilities that interface with other RDEs, thereby allowing their software to be "dropped into" the environment, elevates the status to *well supported*. Neither TeamBots, ARIA, nor MissionLab provide such standard APIs or mechanisms. Player/Stage and CARMEN provide such APIs (for their devices and modules, respectively), Pyro and ADE explicitly include steps to "wrap" external software, MARIE supplies a variety of APIs and mechanisms for integration, while Miro relies on writing TAO interfaces in Interface Device Language (IDL), used to produce C++ code. Each receives at least a *somewhat supported* value. MARIE and Pyro also provide translation facilities such that components written for CARMEN, Player/Stage, or ARIA can be used and earn a *well supported* value.

F4.2 Predefined components

As mentioned earlier, any list of predefined components is open-ended and therefore necessarily incomplete. We limit this list to components that are commonly available and only list the RDEs that include them. Furthermore, no quantitative evaluation is given; the intent is not to establish a full taxonomy, but to provide a high-level indication of system functionality. It should also be noted that both ARIA and MissionLab provide some of the following components so long as they are licensed; due to the limitation of this survey to open source software, such components have been excluded.

F4.2.1 *Map-making:* ARIA provides the *Basic Mapper* software, which can be used to manually construct maps. Player/Stage, Pyro, CARMEN, ADE, and MARIE all include map-making facilities, which are combined with localization.

F4.2.2 *Localization:* TeamBots provides a landmark-based localization component, while Miro provides particle filter localization. ARIA provides sonar-based localization, but full localization facilities must be purchased. Player/Stage, Pyro, CARMEN, ADE, and MARIE all include localization facilities, which are combined with map-making.

F4.2.3 *Route planning:* TeamBots and ADE provide schema-based navigation, ARIA supplies a navigator integrated with its localization package, Player/Stage provides a wavefront propagation route planner, CARMEN uses a Markov decision process planner, MissionLab relies on geometric map analysis and an A* graph search, and MARIE integrates Player/Stage and CARMEN navigation components.

F4.2.4 *Speech recognition:* Player/Stage, ARIA, ADE, Pyro, and Miro all provide speech recognition support through integration of outside software such as Sphinx (Sphinx, 2004) or Sonic (Pellom and Hacioglu, 2003).

F4.2.5 *Speech production:* Player/Stage, ARIA, ADE, Pyro, MARIE, and Miro all provide speech production support through integration of outside software such as Festival (Festival, 2004).

F4.2.6 *Vision processing:* MissionLab and Miro have basic image/video capture capabilities, but Miro also provides stereo image capture and many video filters. TeamBots includes CMVision software, which can capture images and perform blob detection. ARIA has two vision packages available, the ActivMedia Color Tracking Software (ACTS) and VisLib. ACTS is a blob detection package, while VisLib includes image filters, blob detection, and object tracking. Player/Stage supports both ACTS and CMVision. Pyro includes image/video capture, blob, edge, and motion detection, assorted filters, and stereoscopic tools, implemented in C++ for speed reasons. ADE\ includes both an ACTS interface and custom blob detection, object tracking, and face/emotion detection. MARIE, via the RobotFlow software, provides custom image capture, blob detection, movement detection, text/symbol extraction routines, and supports OpenCV.

F4.2.7 *Rule interpreters:* ADE includes an interface to POP-Rulebase (Sloman, 2002) and Prolog, while MARIE and MissionLab both include custom rule interpreters.

F4.2.8 *Planners:* While item F4.2.3 specifically covers navigation planners, these are considered too task-specific to qualify under the general rubric of

"planner". MissionLab and MARIE both supply generic planners.

F4.2.9 *Neural networks:* Pyro, Miro, and MARIE all include neural network software.

F4.2.10 *Learning:* TeamBots provides both reinforcement and Q-learning components, Pyro has a reinforcement learning module, while MissionLab includes integrated case based reasoning and Q-learning components. We do not include neural network software under the *learning* heading, as it appears as a separate category above.

5 RDE usability evaluations

The systematic comparison of RDEs with respect to their supported features based on a conceptual framework is one important part of an RDE evaluation. Another important part is RDE *usability*, for the extent to which an RDE can be easily installed and used in research is ultimately a decisive factor for its adoption. Yet, surprisingly, there is only one previous study (Orebäck and Christensen, 2003) that provides a practical RDE evaluation. And while a robotic architecture was actually implemented and executed on a robot in Orebäck and Christensen (2003), their study is very limited in scope (only three RDEs were evaluated according to a small set of criteria based on a single application) and does not provide a methodology for systematic comparisons and subsequent evaluations that ties together conceptual, practical, and impact factors. Consequently, the conclusions (Orebäck and Christensen, 2003) arrived at have limited applicability.

We believe that a comprehensive evaluation needs to encompass at least the three categories of usability criteria:

U1: *Installation.* Basic steps required to obtain a usable system, evaluated in terms of the required time and effort.

U2: *Basic usability.* Implementation and execution of a simple "lowest common denominator" architecture for RDE comparison, focussing on "low-level" sensor and effector access and allows for an investigation of architectures that reside on a single host.

U3: *Advanced usability.* Usage of individual, predefined, "high-level" components that would commonly form sub-architectures of a complex, distributed architecture; an effort is made to explore uncommon (and possibly unique) "high-level" system features.

The following subsections describe the three categories in more detail. As was done with features in Sections 3 and 4, a set of criteria is defined and subsequently evaluated. Due to the variability of each criteria's subject, value meanings are specified per item; in general they can be interpreted as: ☐ for *below average*, ⊟ for *average*, and ⊞ for *above average*. While an attempt has been made to adhere to a ternary value assignment, a value of *na* is used to indicate that a specific item was not examined in sufficient depth to assign a value for some reason (e.g., incompatible hardware, difficulties with prerequisite software, etc.). *na* values will not be included in an RDE's score. Results are shown in Table 3, where the following shorthand column headings are used to designate particular systems: **TB**–TeamBots, **AR**–ARIA, **P/S**–Player/Stage, **Py**–Pyro, **C**–CARMEN, **ML**–MissionLab, **AD**–ADE, **Mi**–Miro, and **MA**–MARIE.

All systems were installed on at least two computers out of a selection of five: two laptops, two desktops, and the onboard PC of an ActivMedia PeopleBot P2DXe robot (shown on the right in Fig. 1). None of the computers were the same make and model, with varying CPUs (850 MHz Pentium III, 1.3 GHz and 2.0 GHz Pentium M, 2.3 GHz Pentium 4, and a 1.8 GHz AMD Athlon) and memory capacities (from 128MB-1GB), although all used Linux (either Debian or Fedora distributions) running a 2.6.x kernel. Various supporting hardware included microphones, speakers, a Firewire camera, and both wired and wireless Ethernet networking. All non-simulated experiments were conducted on the robot, which also has a pan-tilt unit, sonar, bumpers, and a SICK LMS200 laser range finder.

U1: Installation

Prior to actually using an RDE, it must be properly installed. Since we believe that installation difficulties might often be a deterrent for potential users, we give "Installation" its own category and criteria.

U1.1 *Documentation: Installation documentation* refers specifically to how well the documentation described the installation process, including required preparatory steps and supporting software, minimum system specification, a clearly laid-out sequence of instructions, a list of known or potential issues, inclusion of mailing list or contact addresses, and references to further information. Satisfying more than five of the above requirements receives a ⊞ value, three or four receives a ⊟ value, while less than three receives a ☐.

U1.2 *Non-RDE Installation:* In all cases, installation required additional supporting software. In some cases, this is limited to a single package (e.g., an adequate Java system), while in others, a large set of additional software is required to enable all available features. A value of ⊞ indicates that installing non-RDE software (including, if necessary, determining what supporting software was required, actual compilation and installation, and any needed debugging) took less than three hours, a ⊟ indicates less than two days, while a ☐ indicates more than two days. Due to the level of detail

Table 3 Usability Criteria Evaluation by RDE

Category	Criteria	TB	AR	P/S	Py	C	ML	AD	Mi	MA
Install	U1.1 Documentation	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞
	U1.2 Non-RDE installation	⊞	⊞	⊞	⊞	⊞	⊟	⊟	□	□
	U1.3 RDE installation	⊞	⊞	⊞	⊞	⊟	□	⊞	□	⊟
	U1.4 installation usability	⊞	⊟	⊞	⊞	⊟	□	⊟	□	□
Low-level	U2.1 documentation	□	⊟	⊞	⊞	⊞	⊟	⊟	⊟	⊟
	U2.2 architecture implementation	□	⊟	⊞	⊞	⊞	⊟	⊞	□	□
	U2.3 architecture execution	⊞	⊞	⊟	⊞	□	⊞	⊞	⊟	□
	U2.4 graphical tools	□	⊟	⊟	⊞	⊟	⊞	⊞	⊟	⊟
	U2.5 overhead (memory, CPU)	⊞†	⊟	⊟	⊟	⊟	⊟†	⊞	⊟†	⊞
	U2.6 "low-level" usability	□	⊟	⊞	⊞	⊞	⊟	⊟	□	□
High-level	U3.1 documentation	*na*	⊟	⊞	⊞	⊞	⊟	⊟	□	⊟
	U3.2 predefined components	*na*	⊟	⊞	⊞	□	⊞	⊟	⊟	⊞
	U3.3 task implementation	*na*	⊟	⊞	⊞	⊟	⊞	⊞	⊟	⊟
	U3.4 distribution	*na*	⊟	⊟	□	⊟	⊞	⊞	⊟	⊟
	U3.5 graphical tools	*na*	□	⊟	⊟	□	⊞	⊟	□	⊞
	U3.6 system integration	*na*	□	⊟	⊞	⊟	⊞	⊞	⊟	⊞
	U3.7 "high-level" usability	*na*	□	⊞	⊞	⊟	⊞	⊟	□	⊟

a † indicates that execution of the "low-level" architecture was done in simulation due to difficulties running it on the robot.

FUNCTION simple_architecture
$turning, forward = 0$
while true **do**
  $R \leftarrow getRanges()$
  for all $r \in R$ **do**
    $turning = turning + \frac{\alpha \times cos(r_{angle})}{r_{distance} \times r_{distance}}$
    $forward = forward - \frac{\alpha \times sin(r_{angle})}{r_{distance} \times r_{distance}}$
  end for
  $forward = forward + \beta$
  $setVelocity(forward, turning)$
end while

Fig. 1 *Left:* The "simple" architecture algorithm implementing a wander behavior with obstacle avoidance. At each time step, a set of polar range readings $R = (r_1, r_2, \ldots, r_n)$ is obtained, where $-\pi \leq r_{angle} \leq \pi$ ($r_{angle} = 0$ is straight ahead) and $r_{distance}$ is relative to the center of the robot. The rotational velocity *turning* is calculated by summing the x component $cos(r_{angle})$ of polar readings, divided by square of the distance $r_{distance}$ to account for obstacles, multiplied by some system-dependent scalar α. The translational velocity *forward* is calculated similarly, using the y component $sin(r_{angle})$, subtracted from the total to make it repulsive, then adding a constant β for default forward movement. *Right:* The robot on which experiments were performed

necessary to cover the variety and scope of additional software packages, we do not address many of the related issues encountered, although some additional information is given as part of criteria U1.4.

U1.3 *RDE Installation: RDE installation* refers to the steps necessary to have a usable system, assuming all supporting software has been installed. Values are the same as non-RDE installation (U1.2): a ⊞ value indicates that installation took less than three hours, a ⊟ indicates less than two days, and a □ indicates two or more days were required to attain a usable system.

U1.4 *Installation Usability: Usability*, as related to the installation procedure, is the overall (and ultimately subjective) impression of the experience. Values are assigned relative to the other systems, thus three systems each receive □, ⊟, and ⊞ values. The following notes provide selected information (presented in no particular order) gathered during the implementation process that help explain the evaluations:

- Pyro has a bootable "LiveCD" available, which should avoid installation issues altogether. However, actual robots rarely have a CD drive, making this irrelevant for non-simulated use. The packages in the Pyro `yum` repository conflicted in some cases, but manual installation was done without issue.
- The version of MissionLab available required the use of `gcc` version 3.2 or below and related libraries, which, due to its age and incompatibility with current versions, was the cause of time-consuming installation issues.
- Installation of ACE/TAO software, required for Miro and MARIE, was particularly time-consuming, particularly due to an initial misconfiguration that required multiple manual de- and re-installation. Miro requires a particular ACE/TAO configuration installation, which is not documented.
- Installation of supporting packages for ADE includes a hardware interface for Java, Player/Stage for simulation, a secure shell client and server for distribution, and assorted other packages to attain the full complement of system functionality.
- Few, if any, installation issues were experienced with TeamBots, ARIA, Player/Stage, and CARMEN. The issues that were encountered mostly concerned platform configuration (e.g., appropriate privileges and permissions, default hardware settings that differed from the particular configuration in use, etc.).

U2: Basic usability

Basic usability in this context means to be able to implement and execute a simple robotic architecture to be able to test a minimal set of capabilities supplied by each RDE. The implemented architecture consists of a basic wander behavior that incorporates obstacle avoidance. Only motor control and range finder sensors were accessed, in as direct a manner as possible, providing a "lowest common denominator" for RDE comparison. The basic algorithm, which uses a potential field method, is shown on the left side of Fig. 1. Note that because the robots available to the authors are not supported by TeamBots, the architecture was implemented but execution could only be done in simulation. Similarly, MissionLab and Miro were also run in simulation due to repeated failures. Each time an installation was not successful, the error was located, fixed, and another attempt was made. Once simulated architecture execution was possible, a few additional attempts were made to run the architecture on the robot; when these also failed, simulated results were deemed acceptable. Systems that were only evaluated in simulation are marked with a † in Table 3.

U2.1 *Documentation:* In relation to the "low-level" architecture, *documentation* refers to information that enhances basic usability (e.g., APIs, example code, a frequently asked question list, etc.). A □ value indicates that it was difficult to find information concerning either basic functionality (e.g., how to send a command to the robot) or a solution to a relatively simple problem (e.g., how to start architecture execution). A ⊟ value indicates that information was available but required some effort to locate, while a ⊞ value indicates that very little effort was required to find installation and implementation information.

U2.2 *Architecture implementation: Architecture implementation* refers to the process of writing the program that performs the wander behavior with obstacle avoidance. A □ value indicates that implementing the simple architecture was a major undertaking (requiring more than two days), a ⊟ value indicates that substantial effort was involved (requiring more than 5 hours), and a ⊞ value indicates that implementation required an expected amount of effort (less than 5 hours). It is important to note that because the simple architecture was a low-level implementation that circumvented or avoided the abstractions and/or tools supplied by some RDEs (e.g., TeamBots' Clay system, ARIA's predefined actions, or MissionLab's behavior libraries), the value is not necessarily indicative of what might be considered "normal" usage, which is considered instead as part of "high-level" usability.

U2.3 *Architecture execution: Architecture execution* refers to the effort required to start and stop a fully implemented architecture (including both the control code and supporting software, if they are separate). A □ value indicates a sequence of more than four steps,

perhaps requiring multiple terminal connections that each require separate initialization. A ⊟ value indicates that two to four steps are required, sometimes mitigated by the GUI. A ⊞ value indicates that startup and shutdown requires a single step.[7]

U2.4 *Graphical tools:* For the "low-level" architecture, *graphical tools* refer to the non-command line interface presented by an RDE. A value of ⊞ is given if the system provides both a display of basic operational information and a graphical manner of starting and stopping architecture execution. A value of ⊟ is given if an RDE provides either, while a value of □ indicates that neither are provided.

U2.5 *Operational overhead: Operational overhead* refers to the memory and CPU (M and C, respectively) resources used during architecture execution.[8] Initial conditions were kept constant by rebooting the computer for each system, turning the swap file off, then executing the architecture three times, recording measurements at one second intervals. Each execution run is divided into two phases: (1) *startup* (denoted by a subscript S), demarcated by the time just prior to system startup until robot movement is first detected and (2) *execution* (denoted by a subscript E), which begins when robot movement is detected, continuing for 90 seconds.

Figure 2 shows the average (with standard deviation bars) and maximum values across all three runs. The average values form the basis for calculating evaluation scores by dividing the range in thirds that are assigned values of 0 for the top third, 1 for the middle third, and 2 for the lower third. More specifically, M_S and M_E receive: <20 MB $=2$, <40 MB $=1$, and >40 MB $=0$. For C_S and C_E, $<33\% = 2$, $<66\% = 1$, and $>66\% = 0$. The values shown in Table 3 are the sum of M_S, M_E, C_S, and C_E, where a total of 6 or better receives a ⊞, a 4 or 5 receives a ⊟, and 3 or less receives a □. It is interesting to note that, in most cases, the operational overhead displays the classic memory vs. CPU usage tradeoff in both the startup and execution phases.

U2.6 *"Low-level" usability: Usability*, as related to the "low-level" architecture, is the overall (and ultimately subjective) impression of the experience. Values are assigned relative to the other systems, thus three systems each receive □, ⊟, and ⊞ values. The following notes provide selected information (presented in no particular order) gathered during the implementation process that help explain the evaluations:

- Execution for TeamBots, MissionLab, and Miro were performed in simulation (denoted by a † in Table 3) due to unsupported laser hardware in TeamBots and difficulties in hardware communication for MissionLab and Miro.
- Although RobotFlow supplies components for interfacing with a Pioneer and SICK lasers, execution for MARIE used a Player server as the hardware interface so that MARIE functionality was included in the performance measurements.
- Inclusion of an easily accessible simulator was extremely useful in implementing and debugging an architecture; Player/Stage and CARMEN are particularly strong in relation to simulator integration, while Miro was relatively difficult to access.
- The structure of TeamBots is such that implementations that deviate from the included software (e.g., non-Clay behaviors or unsupported hardware) are very difficult to program.
- CARMEN and MissionLab require some knowledge of IPC messages to implement custom components or write routines that access the available components.
- Player/Stage, Pyro, and ADE provide a good balance of high-level component abstractions while also preserving accessibility to low-level sensor and effector interfaces.
- Player/Stage and Pyro provide a varied base of example code combined with relatively easy configuration and relevant documentation.
- Both MissionLab and CARMEN provide excellent documentation; MissionLab has a clear and complete user manual, while CARMEN's documentation is structured and organized in a manner that made it easy to find desired information.
- Although ARIA provides very good documentation, the "low-level" nature of sensor and effector access used in this architecture deviates from the standard methods addressed therein, leading to a more difficult implementation.
- While both the FlowDesigner and MissionLab GUIs are extensive and polished, some low-level tasks are easier to perform using a text editor (in other words, the GUI was actually a hindrance in some respects).

U3: Advanced usability

While the "low-level" architecture provides a lowest common denominator for subjective RDE evaluation, the implementation of what we will refer to as "high-level

[7] We do not consider placing a sequence of commands in a shell script for execution as a single step.

[8] While disk space usage and bandwidth are important, neither is considered. We exclude disk space due to the variability of packages required, while bandwidth is not addressed due to the single-host nature of the "low-level" architecture.

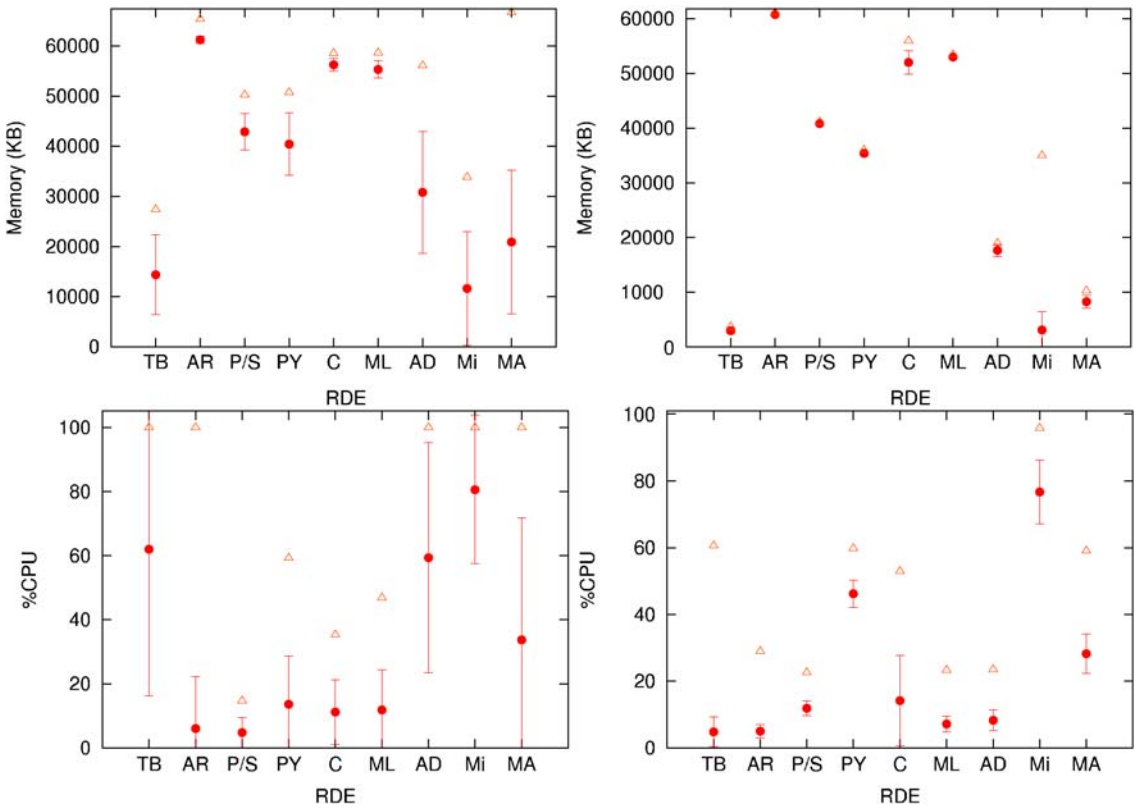

Fig. 2 Operational overhead of each RDE executing the "low-level" architecture. *Top Row:* Memory usage (in KB), showing average (with standard deviation bars) and maximum values. The left-hand figure shows the startup phase, while the right shows the execution phase. *Bottom Row:* CPU usage (as a % of system load), again showing average and maximum values. As with the memory overhead, startup is on the left and execution is on the right

subarchitectures" provides a basis for evaluating an RDE under expected normal usage (although with a focus on distributed computing which becomes necessary due to architecture complexity and real-time processing constraints). Because there are few areas in which all RDEs provide the same capabilities in the same way, the underlying idea is to obtain an estimate of the effort required to distribute a single complex architecture over at least two hosts. A model example of such an architecture is DIARC (Scheutz et al., 2006), which provides a foundation for experiments in human-robot interaction.

The "fundamental" components examined are: (1) vision processing (monocular, that uses blob-detection), (2) speech recognition, (3) speech production, and (4) the provided robot control primitives, although attempts were made to use of some (possibly unique) components and capabilities (e.g., MissionLab's selection of behaviors, ADE's autonomous distribution, etc.). The implementation consisted of installing, configuring, and testing individual components, then connecting at least two of them across a network as an indication of ease of distribution, with the assumption that connections among the other components will require similar effort. No attempt was made to implement these tasks in TeamBots, due to both the limited number of predefined components and because the robots available to the authors are not supported.

U3.1 *Documentation:* In relation to the varied tasks, *documentation* refers to information concerning the advanced RDE functionality (e.g., explanation of available components and their use, example code, etc.). In particular, the focus was on the components required for the varied tasks (vision, speech recognition, speech production, and robot control) and their distribution. A ☐ value indicates either missing or incomplete information for at least two basic functionalities. A ⊟ value indicates that information was either missing or incomplete for only one basic functionality or that the supplied documentation for any basic component was minimal or unclear. A ⊞ value indicates that documentation was complete, easy to locate, and highly readable.

U3.2 *Predefined components: Predefined components* refers to the integrated capabilities included with an RDE. A ☐ value indicates that an RDE was either (1) missing at least one component required to implement all "fundamental" functionality for the envisioned architecture (robot control, vision, speech production, and speech

recognition) or (2) did not provide at least two additional components for each missing "fundamental" component. A ⊟ value indicates that the basic components were available or at least two other components were available for each missing basic component. A ⊞ value indicates that the RDE comes packaged with more than four predefined components beyond what is required for a ⊟ value, in addition to other tools or functionality. Of particular note are Player/Stage, Pyro, MissionLab and MARIE: Player/Stage supports the largest number of devices, Pyro and MissionLab include various additional functionalities (e.g., Pyro includes working examples from Russell and Norvig (2002), MissionLab includes a case-based reasoner for post-execution analysis), while MARIE provides many signal processing and vision tools via the RobotFlow and FlowDesigner packages.

U3.3 *Task implementation: Task implementation* refers to the effort required to implement the subarchitectures, both in terms of accessing individual components and their distribution. As with the "low-level" architecture, a □ value indicates that implementation was a major undertaking (requiring more than two days), a ⊟ value indicates that substantial effort was involved (more than five hours), and a ⊞ value indicates that implementation required an expected amount of effort (of less that five hours). It is imperative to note that the "high-level usability" case made use of the abstractions and/or tools supplied by some RDEs (which sometimes proved detrimental), which may account for differences from the value given for the "low-level" architecture.

U3.4 *Distribution: Distribution* refers to the ease of locating components across hosts once the distribution mechanisms have been implemented, accounting for both startup/shutdown procedures and relocation of components. A ⊞ value indicates that the system provided facilities for automatic login and component startup/shutdown, in addition to providing the means to reconfigure component location. A ⊟ value indicates that only one of those specifications was met, and a □ value indicates that neither is supported. Of note is that RDEs in which components must be implemented with network capabilities (e.g., components in CARMEN or Miro and servers in Player/Stage or ADE) all have a relative advantage.

U3.5 *Graphical tools:* In relation to the "high-level subarchitectures," *graphical tools* refers to both the presentation of an integrated display and the means to graphically implement robot control procedures. This differs from the "low-level" architecture in that (1) an emphasis is placed on integration, such that architecture display and system control is consolidated and (2) a user does not have to write low-level programs. A value of ⊞ is given if an RDE provides both, a value of ⊟ if only one is provided, and □ if neither is provided.

U3.6 *System integration: System integration* refers to the separate issues of (1) easily connecting and controlling components in a complex architecture and (2) coherent system usage in terms of providing a consistent interface to the complete system that allows access to and control of individual components. A ⊞ value is assigned if both objectives are met, a ⊟ if only one is satisfied, and a □ if neither.

U3.7 *"High-level" usability:* As with the "low-level" architecture, *usability* is the overall (and subjective) impression of the experience of using each RDE. Values are assigned relative to the other systems, thus three systems each receive ⊟ and ⊞ values, while two receive a □ value (because TeamBots was not evaluated in terms of the high-level tasks). The following notes provide selected information (presented in no particular order) gathered during the implementation process that help explain the evaluations:

- Both ARIA and MissionLab have additional components that were not considered here as they require licensing and were not included in the downloadable package.
- While ARIA's documentation is in general very good, the distribution package, `ArNetworking`, lacks complete documentation. For some basic tasks, MARIE's documentation is sparse, limited to "node" listings, while ADE and Miro are comparably spotty.
- Pyro's interface is well integrated, allowing access to various conceptually separate parts of an application (i.e., the *server*, *robot*, *devices*, and *"brain"*), while also providing the ability to enter Python commands at runtime. However, non-graphical usage is not quite as polished (for instance, hanging when exiting the system).
- The user interface provided by MissionLab is especially suited to task specification implemented by non-programmers, matching its objective of providing a high-level view of robot control.
- Systems that require the strict use of abstractions in support of their component model (e.g., the ACE/TAO in Miro and FlowDesigner networks in MARIE) or, to a lesser extent, require some level of abstraction removed from actual source code (e.g., IPC/IPT communications in CARMEN and MissionLab and RMI in ADE) can be either beneficial or detrimental to some degree. For instance, Miro's requirement that all components are CORBA objects makes component integration and distribution extremely simple, but the implementation of an arbitrary component is made more difficult.

6 Discussion

The evaluations presented in Sections 4 and 5 provide the foundation for comparing RDEs, both at a conceptual level and from a practical perspective. Augmented with an impact evaluation (to be described below), we envision the results of this survey being useful to the robotics community in at least two ways: (1) by providing *researchers* with a practical means of selecting an RDE that will most closely match their requirements and (2) by giving RDE *developers* an overview of the innovations being made in other systems, possibly suggesting improvements and extensions to their own system.

6.1 Researchers

To perform a comprehensive comparison of the RDEs presented, three separate measures are given: (1) a summary of the evaluations from Section 4 concerning an RDE's *features*, (2) a summary of the evaluations from Section 5 concerning an RDE's *usability*, and (3) an estimate on the influence an RDE has had on the robotics field (i.e., its *impact*), which is gauged by the breadth of publications from different research areas (as listed at the end of each description from Section 2) and the number of other RDEs that provide interoperability interfaces with a system. A researcher examining a group of RDEs to find one that best fits their needs might conduct evaluations using either *qualitative* or *quantitative* measures.

A qualitative evaluation is highly contingent on the user's purpose; on the one hand, very specific capabilities may be required, while on the the other, needs may be highly abstract or only loosely defined. For instance, an application designer who has a large body of already written Octave software and desires to use it with a robot might look at the system descriptions in Section 2 and find that MARIE already has an Octave plug-in. Conversely, educators establishing an "Introduction to Robotics" class might consider the items from the usability evaluation in Section 5 to be of overriding importance; an examination of the values given to the *documentation* criteria (U1.1, U2.1, and U3.1) might lead them to limit attention to Player/Stage, Pyro, and CARMEN.

More likely, however, is that a *mixture* of characteristics is desired. For example, a developer might require a system that provides a simulated environment and a fair level of distribution facilities, with a preference for a system that supplies extensive GUI capabilities and usability oriented towards non-programmers. An examination of Section 4 would lead to considering criteria F2.3 (*Simulator*), F3.4 (*Distribution Mechanisms*), F4.1.2 (*High-level Language*), and F4.1.5 (*Graphical Interface*), while Section 5 would U2.4 (*Graphical Tools*) and U3.1–U3.7 (*High-level Usability*), making MissionLab the most likely choice.

To arrive at a quantitative evaluation, an assignment of values to criteria must be made, which can then be applied to a selection (or all) of the criteria. To arrive at numerical comparison scores, the three categories of criteria mentioned above are retained, yielding a *feature* score F, *usability* score U, and *impact* score I, which can be summed to give a *total* score T. Within each category, values of 0 or 2 are assigned to binary-valued criteria and values of 0, 1, or 2 to ternary-valued criteria (listing features are not scored).[9]

In formal terms, the selected RDEs form the set $S = \{TB, AR, P/S, Py, C, ML, AD, Mi, MA\}$ and are assigned a score that is the sum of the category scores F, U, and I. Given a number of individual criteria within each category (e.g., $F_{1.1}$ denotes *Architectural Primitives*, while $U_{1.1}$ denotes *Installation Documentation*), category scores are comprised of the sum of individual criteria values F_i, U_j, and I_k, where m, n, and o are the total number of feature, usability, and impact criteria and $1 \leq i \leq m$, $1 \leq j \leq n$, and $1 \leq k \leq o$. In the simplest case, where all criteria are given equal weight, the RDE comparison scores are given by the formula:

$$T_{s \in S} \leftarrow \sum_{i=0}^{m} F_i^s + \sum_{i=0}^{n} U_i^s + \sum_{i=0}^{o} I_i^s$$

Should a quantitative evaluation that weights some categories or criteria more or less than others be desired, weights can be assigned at both a coarse-grained level (for each category, α, β, and γ) and a fine-grained level (for each criterion within a category, W_{F_i}, W_{U_j}, and W_{I_k}). The resultant formula for establishing the total comparison scores is:

$$T_{s \in S} \leftarrow \alpha \sum_{i=0}^{m} W_{F_i} F_i^s + \beta \sum_{i=0}^{n} W_{U_i} U_i^s + \gamma \sum_{i=0}^{o} W_{I_i} I_i^s$$

where $\alpha + \beta + \gamma = 3$, $\sum W_{F_i} = m$, $\sum W_{U_i} = n$, and $\sum W_{I_i} = o$. This method will give an objective comparison in that scores are not biased for or against any particular RDE, although the choice of features and their assigned weights are based on the particular application requirements.

The quantitative evaluation that does not weight any criteria or category more than another follows, supplemented by a discussion of each category score and the totals. The tabulated scores are shown in Tables 4, 5, and 6, respectively, and summarized in Table 7.

MARIE and ADE have the highest score (44 and 43, respectively) in terms of the *Feature* score F. The *Implementation* category contributes more than half of the F score, exerting the largest influence. This is quite acceptable, as it corresponds to the expectation that the purpose of an RDE

[9] Binary and ternary values range from 0 to 2 so as to not introduce a bias towards ternary criteria.

Table 4 The raw *Feature* score F and %, broken down by categories from Section 4 for each RDE

	Feature category									
	Specification (6)		Platform (6)		Infrastructure (16)		Implementation (30)		Total (58)	
RDE	Score	%	Score	%	Score	%	Score	%	Score	%
TB	5	83	1	17	0	0	10	33	16	28
AR	4	67	2	33	6	38	14	47	26	45
P/S	3	50	5	83	9	56	17	57	34	59
Py	5	83	5	83	5	31	22	73	37	64
C	4	67	5	83	8	50	10	33	27	47
ML	6	100	4	67	10	63	15	50	35	60
AD	6	100	3	50	16	100	18	60	43	74
Mi	5	83	3	50	9	56	14	47	31	53
MA	5	83	6	100	10	63	23	77	44	76

is to provide appropriate tools and abstractions that facilitate application development. MARIE and Pyro have the highest implementation scores (23 and 22), indicative of their wide selection of components (partially due to its interoperability with other RDEs) and advanced GUI capabilities. Pyro is assigned the third highest F score (37) due to the *Infrastructure* category, which accounts for over 25% of the final score. ADE, which has a score of 18 in the implementation category, ends up with the highest F score (43) due to the infrastructure category, as it the only RDE that earns a *well supported* value for each criterion therein.

Of particular note is the fact that the RDEs with the highest scores (Pyro, ADE, and MARIE) all have explicit interoperability interfaces with other systems. While providing a boost in total feature score, we also note that this makes them reliant, to some degree, on the availability of the other systems for certain features, in addition to potentially affecting their stability (in that changes to the other RDEs may impact their operation). We also note again that comparing RDEs in terms of F alone does not provide a full picture of evaluation, leaving out aspects such as *system usability*, discussed next.

As noted in Section 3, an appropriate set of criteria must be considered to serve as the basis for comparing RDEs. Not only do applications have markedly different characteristics that may impact the designer's choice of RDE, but users tend to have different needs and working styles. The *Usability* score U attempts to address the practical aspects of RDE usage by adding criteria relevant for the actual implementation and execution of robotic architectures (in particular, the two classes of tasks described in the previous section), the results of which are summarized in Table 5.

Pyro, Player/Stage, ADE, and MissionLab have the highest usability scores (30, 28, 26, and 24, respectively). From a usability point of view, this indicates that each has fulfilled their stated purpose: Pyro is aimed at novice users for educational purposes, Player/Stage is a flexible and adaptable programming interface, ADE combines robotic development with a MAS infrastructure, while MissionLab provides military personnel with non-programming methods of controlling robots. On the other hand, the low score given to Miro can be attributed to its reliance on the ACE/TAO communication framework[10] and incomplete documentation. In addition, it is necessary to point out that both ARIA and MissionLab's scores would be higher if the restriction to open-source components was lifted.

Table 5 The raw *Usability* score U and %, broken down by categories from Section 5 for each RDE (a † indicates some criteria were not included)

	Usability category							
	Installation (8)		"Low-level" (12)		"High-level" (14)		Total (34)	
RDE	Score	%	Score	%	Score	%	Score	%
TB†	8	100	4	33	*na*	*na*	12	35
AR	7	88	7	58	4	29	18	53
P/S	8	100	9	75	11	79	28	82
Py	8	100	11	92	11	79	30	88
C	6	75	8	67	6	43	20	59
ML	3	38	8	67	13	93	24	71
AD	6	75	10	83	10	71	26	76
Mi	2	25	4	33	4	29	10	29
MA	3	38	4	33	10	71	17	50

It is interesting to note that MissionLab and MARIE have the widest discrepancy in score between usability categories, each scoring relatively highly for "high-level" usability but low on the "low-level" architecture. Personal experience determined that much of the difference can be attributed to predefined components (both their number and usage) and their integration into a cohesive user interface. Both provide a comprehensive graphical method for connecting

[10] The impact of ACE/TAO is acknowledged in the user manual thusly: "The CORBA environment and the Miro framework seem to raise the bar for an easy entry into robot programming. While this can hardly be denied they facilitate tremendously the task of writing distributed programs."

Table 6 The raw *Impact* score *I* and % for each RDE, the sum of "Application Area References" citations from Section 2 and the other RDEs that provide interoperability interfaces

Application area	TB	AR†	P/S	Py	C	ML	AD	Mi	MA
SLAM			√		√				√
Planning/Navigation							√	√	√
Learning			√	√	√	√			√
Hierarchical behavior	√					√	√		√
Education	√		√	√				√	
HRI — task allocation			√	√		√	√		
HRI — learning									
HRI — Assistive Robotics					√		√	√	√
Multi-robot Sensing			√				√		√
Multi-robot Exploration			√						
Multi-robot Mapping			√						
Multi-robot Localization			√					√	
Multi-robot Planning			√			√			
Multi-robot Coordination			√		√	√	√	√	√
Multi-robot Formation			√			√		√	
Multi-robot Task Allocation			√			√			
Research areas (out of 16)	2	0†	12	3	4	7	6	6	7
Interoperability facilities	0	2	4	0	1	0	0	0	0
Total score (out of 20)	2	2†	16	3	5	7	6	6	7
Total %	10	10†	80	15	25	35	30	30	35

a † indicates some criteria were not included.

Table 7 The *Total* comparison score *T* and % for each RDE, comprised of the sum of feature *F*, usability *U*, and impact *I* scores. The left-hand columns under the *Removed* † *Criteria* heading do not include criteria unevaluated for any RDE, while columns under the *All Criteria* include all criteria, using a value of zero for unevaluated items

	Score															
	Removed † criteria								All criteria							
	F (58)		*U* (20)		*I* (4)		Total (82)		*F* (58)		*U* (34)		*I* (20)		Total (112)	
RDE	Raw	%	Raw	%	Raw	%	Raw	%	Raw	%	Raw	%	Raw	%	Raw	%
TB†	16	28	12	60	0	0	28	34	16	28	12	35	2	10	30	27
AR†	26	45	14	70	2	50	42	51	26	45	18	53	2	10	46	41
P/S	34	59	17	85	4	100	55	67	34	59	28	82	16	80	78	70
Py	37	64	19	95	0	0	56	68	37	64	30	88	3	15	70	63
C	27	47	14	70	1	25	42	51	27	47	20	59	5	25	52	46
ML	35	60	11	55	0	0	46	56	35	60	24	71	7	35	66	59
AD	43	74	16	80	0	0	59	72	43	74	26	76	6	30	75	67
Mi	31	53	6	30	0	0	37	45	31	53	10	29	6	30	47	42
MA	44	76	7	35	0	0	51	62	44	76	17	50	7	35	68	61

components, but suffer from either not providing a graphical interface to all parts of the system (e.g., MARIE requires shell scripts for startup/shutdown and the definition of communication channels) or by their orientation towards very high-level tasks.

Finally, an oblique way to determine the strengths of an RDE is to establish an *Impact* score *I* that reflects the influence it has had on the robotics field. The number of research areas in which there are publications serve as an indicator of successful usage, as does the recognition that widely used systems are most likely to have other RDEs provide interoperability interfaces. Table 6 summarizes the robotics research subareas and citations given in Section 2 for each RDE, in addition to giving a count of the number of RDEs that interoperate with it. We reiterate that a single publication is used to satisfy research in any subarea, simply to provide an idea of the breadth of research areas in which it has been used; we also note the likelihood that some relevant publications were not included, as the particular RDE used is sometimes not mentioned in a publication. Because criteria are all either

binary in nature or a simple count, total scores are a simple sum of items, deviating slightly from the previous convention of assigning 2 points to a √. Player/Stage clearly has had the largest impact, reflected by the fact that its score (16) is more than double that of the next highest system. It is also necessary to point out that ARIA's impact score is deceptively low (indicated by the † symbol), due to the fact that the authors were asked not to include references to its ancestral software.

Two overall comparison scores $T_{\dagger}$ and T_{all} are shown in Table 7, where $T_{\dagger}$ does not include criteria that were unevaluated for any RDE and T_{all} does. Overall scores are the sum of the feature F, usability U, and impact I scores. Player/Stage has the highest total score T_{all} (78 out of a possible 112), partially due to having the highest I score, even though it had the fifth highest F score and second highest U score. The next three highest scoring RDEs (ADE with 75, which had the highest F score; Pyro with 70, which had the highest U score; and MARIE with 68, which had the second highest F score) are all relatively new systems; in addition to providing some level of interoperability interfaces with other RDEs (thus capitalizing on prior innovations), we believe that part of their score can be attributed to identifying areas of application development that can be improved, partially based on the examples of already established RDEs (discussed in more depth in the next section). Of note is that when the unevaluated criteria are removed, the top four RDEs (ADE, Pyro, Player/Stage, and MARIE, respectively) remain the same.

We reiterate that the total scores $T_{\dagger}$ and T_{all} may not reflect the particular requirements of a particular person or group and that while the evaluations here are comprehensive, they necessarily miss some criteria that may be important for a specific designer or application. Such items can be added at will to further refine and customize the evaluations, adjusting the evaluation formula given earlier.

6.2 RDE maintainers and developers

The selected RDEs in this survey are, as defined by the constraints of system selection, open source projects. While their availability is of obvious benefit to users, individual RDE maintainers can also potentially reap some benefit by examining other systems. Hopefully, this will facilitate the transfer of techniques and tools (e.g., Vaughan et al., 2003; Montemerlo et al., 2003b; Hattig et al., 2003; Howard and Roy, 2004) across environments and promote progress in the field of robotics as a whole. Using the *Feature* and *Usability* comparison scores from the previous section as a basis (the *Impact* score is not considered, as it is not directly controlled by RDE maintainers), it becomes possible to not only make specific improvement suggestions, but also to make some high-level points. We note here that no suggestions are made for TeamBots because it is no longer under active development.

To begin, we examine the *Feature* scores shown in Table 4 by discussing each category. In the *Specification* category, all RDEs score at least 50%, while 8 of the 9 score 67% or higher, indicating that all supply adequate support for application design. Considering the *Platform* category, only two systems score less than 50%: TeamBots, which is no longer being actively developed, and ARIA, which has been developed in support of a proprietary platform and thus has unique objectives. Increasing the hardware support in MissionLab, ADE, and Miro would yield scores of 67% or higher for all of the remaining systems, such that all could be considered to have adequate platform support. In terms of the *Implementation* category, only CARMEN and MissionLab score below 50%. Recalling the information found in Table 2, it is evident that the score is substantially due to supporting a limited number of predefined components (although it is important to note that some components are available with MissionLab if licensed). ARIA's score is similarly affected by licensing issues, in that an integrated GUI is available; inclusion would put its score at about 60%. ADE, Miro, and MARIE all have inadequate documentation, which would improve their *Feature* score, while also increasing their *Usability* scores. The last category factoring into the feature score is *Infrastructure*, discussed very briefly here due to its inclusion in Section 7. ARIA, Pyro, and CARMEN all score 50% or below; again, ARIA's licensing requirements affect its score to some degree, leading to a *not supported* value for the *Monitoring and Management* criterion. The most prevalent unsupported criterion is *Security*, which only ARIA and ADE support at all. As noted earlier, Player/Stage, Miro, and MARIE all have the potential to easily incorporate security (Player/Stage by utilizing its authentication mechanism and Miro/MARIE by leveraging the ACE/TAO framework). The next least supported criteria are *Component Mobility* and *Fault-tolerance*, related to *Distribution Mechanisms*. Suggestions for improvements in these areas is beyond the scope of this survey, and we again refer to Section 7 for more.

Three categories make up the *Usability* score, *Installation*, the "low-level" architecture, and "high-level" subarchitectures. Three RDEs are at or below 50% in installation (MissionLab, Miro, and MARIE), three in the "low-level" category (TeamBots, Miro and MARIE), and three in the "high-level" category (ARIA, CARMEN, and Miro). Discounting unavailable software, ARIA's individual scores are well distributed among criteria, indicating that while each could be improved, a good overall mix is established. MissionLab's installation score is low due to its reliance on older versions of `gcc`; a new release would most likely greatly improve its score. As with the *Implementation* category mentioned above, CARMEN would benefit greatly

from additional predefined components. Miro's framework, relying as it does on CORBA, has great potential in terms of usability; however, additional tools and documentation are required to "lower the bar" that, as they say in their user manual, has been placed quite high. In MARIE's case, the lowest individual criteria scores are concentrated in the "low-level" architecture. In particular, the necessity of manually specifying component connections lowers usability, as does what might be considered a high learning curve.

From the above, one broad point that should be clear is that interoperability tools are of great utility, allowing one RDE to incorporate any component functionality developed in another RDE. Interoperability has been discussed in the literature (e.g., Baum et al., 2002; Nesnas et al., 2003; Côté et al., 2005), and while no single technique has been accepted, MARIE's foundations suggest a direction for future standards. While Pyro provides a large base of interoperability tools, MARIE's stated intention is to provide a well-grounded framework that is not only compatible with existing systems, but also provides the conceptual basis for adding interoperability in the future. The benefits of this approach are apparent when considering the available pre-defined component list; components available in Player/Stage, ARIA, and CARMEN are also available in Pyro and MARIE.

Finally, to gain acceptance, an RDE must pay attention to usability and the quality of its user and developer interfaces (see Steinfeld, 2004 for a general treatment). An area that is receiving increasing attention is robotics education. In addition to opening up the field to new people, an RDE that caters to novices should, almost by definition, promote usability. Pyro is particularly strong in this respect. Another aspect of usability concerns those who are not interested in going beyond the functionality already provided by an RDE, but rather use the already established components to implement their own applications without programming. In this sense, the FlowDesigner package (related to MARIE) provides a graphical method for defining data flow throughout an application. Taking this a step further, MissionLab provides graphical tools not concerned with robot particulars at all, but only with their actions. Additionally, the MissionLab developers have conducted many usability studies (e.g., MacKenzie and Arkin, 1998; Collins et al., 2000; Endo et al., 2004; Moshkina et al., 2006) in conjunction with system development.

7 Conclusion and outlook

This survey has evaluated nine RDEs with respect to an extensive set of relatively common criteria supporting the development of robotic applications. Results were then compiled and used to compare the systems according to three types of score (*Feature*, *Usability*, and *Impact*), providing robotic architecture designers with information useful in picking an RDE for themselves. Finally, the comparisons provided the foundation for suggesting potential areas of improvement to RDE maintainers based on features currently found in other systems. In conclusion, we extrapolate from the results and attempt to identify some likely future trends.

The comparison of different RDEs suggests that common features will increasingly be expected in all systems, strengthened by the interoperability mechanisms found in some recent systems (e.g., Pyro and MARIE). In addition to creating a set of (possibly *de facto*) standards, this will lead to an increasing number of predefined components that can be expected in any given RDE. Furthermore, we expect the list of predefined components given in Section 3 to continue to expand, both in relation to high-level functionality (e.g., various types of robot control) and more specific low-level functionality (e.g., "vision processing" will split into separate categories such as monocular vs. stereo vision processing). We feel that a similar trend will develop in relation to RDE *infrastructure* (see Section 3), such that users expect inclusion of a suite of tools that implement various non-architectural functions. This suspicion is borne out by a cursory examination of the origination of RDEs. Early systems (e.g., TeamBots, 1998) provide little in the way of infrastructure: an application in TeamBots is the sum of the Java classes that implement it. Player/Stage (2001) incorporates a minimal amount of infrastructure; the authors acknowledge and deliberately reject this trend, making the system "free from the computational and programmatic overhead that is generally associated with the practical application" of such mechanisms (Gerkey et al., 2003). More recent RDEs, such as MARIE (2004) and ADE (2004), explicitly incorporate substantial infrastructure into their design and use, with the stated aims, respectively, of improving interoperability and distribution.

The necessity of providing infrastructural interoperability and distribution is illustrated by a quote from the authors of the GRACE project (Simmons et al., 2003): "One of the more difficult parts of the Challenge for us was determining how to integrate a vast amount of software that had been developed by the participating institutions, mostly on different hardware platforms." Such mechanisms should immediately bring to mind multi-agent systems (MAS) research, which has found particular traction in the robotics field in the form of multi-robot applications (such as the citations listed at the bottom of Table 6; (see also Sycara and Zeng, 1996; Altmann et al., 2001; Dias and Stentz, 2003; Gerkey and Matarić, 2004). We suggest that it will be critical for future RDEs to incorporate other aspects of MAS research, including, but not limited to security (e.g., Singh and Sycara, 2004) and system-wide management facilities (such as those discussed in Bellifemine et al. (1999) and Sycara et al. (2003)).

A final trend we expect to take shape in RDEs in the future is the prominent promotion of *autonomic computing* functionality (e.g., Bantz et al., 2003). On the one hand, we expect improvements in low-level system characteristics that are transparent to users such as *fault tolerance* (e.g., Varakantham et al., 2002; Long et al., 2003; Melchior and Smart, 2004). ADE, for example, already explicitly incorporates features for monitoring, relocating, and restarting of components integrated into its infrastructure. Moreover, MARIE and Miro can potentially take advantage of recent advances in middleware, e.g., the Fault Tolerant CORBA specification (see Chapter 23 in CORBA, 2005), which incorporates mechanisms that promote robust system operation. On the other hand, we expect that development of high-level AI techniques that enhance a robot's apparent intelligence will increasingly find inclusion in RDEs, like the tools found in MissionLab. We expect that robot learning (e.g., Russell, 2004; Blank et al., 2005), findings from human-robot interaction (HRI) research (e.g., Salter et al., 2005; Fong et al., 2006; Moshkina et al., 2006; Scheutz et al., 2006), and the study of social robotics (e.g., Bruce et al., 2002; Breazeal, 2003) will become commonplace.

In sum, we believe that the increase in capability of robotic applications will soon require extensive infrastructure support, with expanding development of support for autonomic computing in the future. Such tools and techniques will be used not only for the development and debugging of robotic architectures, but also for the execution and maintenance of robotic architectures as part of application deployment. If true, the choice of one RDE over another will be based on more than just the development support offered, but increasingly on the features it provides for the long-term operation of robotic applications. Furthermore, and perhaps more significantly, the integration of system infrastructure with the development of intelligent robotic architectures will lead to robots that display ever greater levels of autonomy.

References

Activmedia robotics mobilerobots developer support. 2005. http://robots.mobilerobots.com/.

Altmann, J., Gruber, F., Klug, L., Stockner, W., and Weippl, E. 2001. Using mobile agents in real world: A survey and evaluation of agent platforms. In T. Wagner (Ed.), *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS at the 5th International Conference on Autonomous Agents*. ACM Press, Montreal, Canada, pp. 33–39.

Andronache, V. and Scheutz, M. 2004a. ADE—a tool for the development of distributed architectures for virtual and robotic agents. In *Proceedings of the 4th International Symposium "From Agent Theory to Agent Implementation."*

Andronache, V. and Scheutz, M. 2004b. Integrating theory and practice: The agent architecture framework APOC and its development environment ADE. In *Proceedings of AAMAS* 2004.

Arkin, R. and Balch, T. 1997. AuRA: principles and practice in review. *JETAI*, 9(2–3):175–189.

Arkin, R., Collins, T., and Endo, T. 1999, Tactical mobile robot mission specification and execution. *Mobile Robots XIV*, pp. 150–163.

Arkin, R., Endo, Y., Lee, B., MacKenzie, D., and Martinson, E. 2003, Multistrategy learning methods for multirobot systems. In *Proceedings of the 2nd International Workshop on Multi-robot Systems*, Washington, DC, pp. 137–150.

Austin, D. 2004. Dave's Robotic Operating System. http://dros.org/.

Balch, T. 2000. Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, 8(3):209–238.

Balch, T. 2002. Teambots Proposal. http://www.cs.cmu.edu/ trb /robocupjr/.

Balch, T. 2004. *Teambots*. http://www.teambots.org/.

Balch, T. and Arkin, R. 1999. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 20(5).

Balch, T. and Ram, A. 1998. Integrating robotics research with JavaBots. In *Working Notes of the AAAI* 1998 Spring Symposium.

Bantz, D., Bisdikian, C., Challener, D., Karidis, J., Mastrianni, S., and Mohindra, A. 2003. Autonomic personal computing. *IBM Systems Journal*, 42(1):165–176.

Baum, W., Bredenfeld, A., Hans, M., Hertzberg, J., Ritter, A., and Schönherr, F. 2002. Integrating heterogeneous robot and software components by agent technology. *Robotik 2002 Leistungsstand - Anwendungen - Visionen - Trends*, pp. 655–660.

Beaudry, E., Brosseau, Y., Côté, C., Raïevsky, C., Létourneau, D., and Kabanza, F. 2005. Reactive planning in a motivated behavioural architecture. In *Proceedings American Association for Artificial Intelligence Conference*, pp. 1242–1247.

Bellifemine, F., Poggi, A., and Rimassa, G. 1999. JADE—a FIPA-compliant agent framework. In *Proceedings of the 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-agents*. London, pp. 97–108.

Bentivegna, D. and Atkeson, C. 2002. *Learning How to Behave from Observing Others* (SAB02 Workshop on Motor Control in Humans and Robots: on the interplay of real brains and artificial devices).

Bergbreiter, S. and Pister, K. 2003. CotsBots: An off-the-shelf platform for distributed robotics. In *Proceedings of 2003 IEEE*/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), Vol. 3, pp. 1632–1637.

Bergbreiter, S. and Pister, K. 2005. *CotsBots*: An Off-the-shelf Distributed Robot Platform. http://www-bsac.eecs.berkeley.edu/projects/cotsbots/.

Biggs, G. and MacDonald, B. 2003. A survey of robot programming systems. In *Proceedings of the Australasian Conference on Robotics and Automation*. Brisbane, Australia.

Bitting, E., Carter, J., and Ghorbani, A. 2003. Multiagent systems development kits: An evaluation. In *Proceedings of the 1st Annual Conference on Communication Networks and Services Research (CNSR* 2003). Moncton, Canada, pp. 101–107.

Blank, D., Kumar, D., and Meeden, L. 2002. A developmental approach to intelligence. In S. Conlon (Ed.), *Proceedings of the Thirteenth Annual Midwest Artificial Intelligence and Cognitive Science Society Conference*.

Blank, D., Kumar, D., Meeden, L., and Marshall, J. 2005. Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture. *Cybernetics and Systems*, 36(2).

Blank, D., Kumar, D., Meeden, L., and Yanco, H. 2003. Pyro: A python-based versatile programming environment for teaching robotics. *Journal on Educational Resources in Computing*, 3(4):1–15.

Blank, D., Kumar, D., Meeden, L., and Yanco, H. to appear. Pyro: A python-based versatile programming environment for teaching robotics. *ACM Journal on Educational Resources in Computing (JERIC)*.

Breazeal, C. 2003. Towards sociable robots. *Robotics and Autonomous Systems*, 42(3–4):167–175.

Brooks, A., Kaupp, T., Makarenko, A., Oreback, A., and Williams, S. 2005. Towards component-based robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*, pp. 163–168.

Brooks, R. 1990. *The Behavior Language: User's Guide* (Tech. Rep. No. AIM-1227). Massachusettes Institute of Technology.

Brooks, R. 1991. Intelligence without representation. *Artificial Intelligence Journal*, 47:139–159.

Bruce, A., Nourbakhsh, I., and Simmons, R. 2002. The role of expressiveness and attention in human-robot interaction. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Bruyninckx, H. 2001. Open robot control software: The OROCOS project. In *Proceedings of IEEE* International Conference on Robotics and Automation (ICRA) 2001, Vol. 3, pp. 2523–2528.

Bruyninckx, H. 2005. *The OROCOS* project. http://www.orocos.org/.

Chaimowicz, L., Cowley, A., Sabella, V., and Taylor, C. 2003, ROCI: A distributed framework for multi-robot perception and control. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Vol. 3, pp. 266–271.

The CMU sphinx group open source speech recognition engines. 2004. http://cmusphinx.sourceforge.net/html/cmusphinx.php. The Sphinx Group at Carnegie Mellon University.

Collins, T., Arkin, R., Cramer, M., and Endo, Y. 2000. Field results for tactical mobile robot missions. In *Unmanned Systems 2000*, Orlando, FL.

Common object request broker architecture (CORBA/IIOP). 2005. http://www.omg.org/technology/documents/corba_spec_catalog.htm. Object Management Group.

Côté, C. 2005. *Mobile and Autonomous Robotics Integration Environment (MARIE)*. http://marie.sourceforge.net/.

Côté, C., Brosseau, Y., Létourneau, D., Raievsky, C., and Michaud, F. 2006. Robotic software integration using MARIE. *International Journal on Advanced Robotics Systems*, 3(1):55–60.

Côté, C., Létourneau, D., Michaud, F., and Brosseau, Y. 2005. *Software Design Patterns for Robotics: Solving Integration Problems with MARIE*. Submitted for workshop to ICRA2005.

Côté, C., Létourneau, D., Michaud, F., Valin, J., Brosseau, Y., and Raievsky, C. 2004. Programming mobile robots using RobotFlow and MARIE. In *Proceedings IEEE/RSJ International Conference on Robots and Intelligent Systems*.

Desai, M. and Yanco, H. 2005, Blending human and robot inputs for sliding scale autonomy. In *Proceedings of the 14th IEEE International Workshop on Robot and Human Interactive Communication*. Nashville, TN.

Dias, M. and Stentz, A. 2003, A comparative study between centralized, market-based, and behavioral multirobot coordination approaches. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Vol. 3, pp. 2279–2284.

Eiter, T. and Mascardi, V. 2002. Comparing environments for developing software agents. *AI Communications*, 15(4):169–197.

Endo, Y., MacKenzie, D., and Arkin, R. 2004, Usability evaluation of high-level user assistance for robot mission specification. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(2):168–180.

ERSP 3.0 Robotic Development Platform. 2004. http://www.evolution.com/products/ersp/. Evolution Robotics.

Fay, R., Kaufmann, U., Schwenker, F., and Palm, G. 2004. Learning object recognition in a NeuroBotic system. In H. Groß, K. Debes, and H. Böhme (Eds.), *3rd Workshop on Selforganization of Adaptive Behavior SOAVE* 2004. Dusseldorf, VDI, pp. 198–209.

The Festival Speech Synthesis System. 2004. http://www.cstr.ed.ac.uk/projects/festival/. Centre for Speech Technology Research.

FIPA ACL Message Structure Specification (SC00061G). 2002. http://www.fipa.org/specs/fipa00061/. Foundation for Intelligent Physical Agents.

Fleury, S., Herrb, M., and Chatila, R. 1997. Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *International Conference on Intelligent Robots and Systems, IEEE*, Vol. 2, pp. 842–848.

Fleury, S. and Mallet, A. 2004. LAAS Open Software for Autonomous Systems. http://softs.laas.fr/openrobots/tools/genom.php.

Fong, T., Kunz, C., Hiatt, L., and Bugajska, M. 2006. The human-robot interaction operating system. In *Proceedings of the ACM Conference on Human-robot Interaction (HRI2006)*, ACM.

Fong, T., Nourbakhsh, I., and Dautenhahn, K. 2003. A survey of socially interactive robots. *Robotics and Autonomous Systems*, 42:143–166.

Fredslund, J. and Matarić, M. 2002, A general, local algorithm for robot formations. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-Robot Systems*, 18(5):837–846.

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

Gassull, G. 2001. *Communication Services and User Interfaces for Tele-operating Mobile Robots via the Internet*. Master's thesis, University of Barcelona and University of Ulm, Neuroinformatics.

Gerkey, B., Howard, A., and Vaughan, R. 2005. Player/Stage. http://playerstage.sourceforge.net/.

Gerkey, B. and Matarić, M. 2002, Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-Robot Systems*, 18(5):758–768. (Also Technical Report IRIS-01-399).

Gerkey, B. and Matarić, M. 2004. Are (explicit) multi-robot coordination and multi-agent coordination really so different? In *Proceedings of the AAAI Spring Symposium on Bridging the Multi-agent and Multi-robotic Research Gap*, pp. 1–3.

Gerkey, B., Vaughan, R., and Howard, A. 2003. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics*. Coimbra, Portugal, pp. 317–323.

Gerkey, B., Vaughan, R., Støy, K., Howard, A., Sukhatme, G., and Matarić, M. 2001, Most valuable player: A robot device server for distributed control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Wailea, Hawaii, pp. 1226–1231.

Guilbert, N., Beauregard, M., Michaud, F., and de Lafontaine, J. 2003. Emulation of collaborative driving systems using mobile robots. In *Proceedings IEEE Conference on Systems, Man, and Cybernetics*, pp. 856–861.

Hattig, M., Horswill, I., and Butler, J. 2003, Roadmap for mobile robot specifications. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Vol. 3, pp. 2410–2414.

Heckel, F. 2005. ROLE Robotics Development Environment. http://www.cse.wustl.edu/ fwph/role/.

Hoff, J. and Bekey, G. 1995. An architecture for behavior coordination learning. In *IEEE International Conference on Neural Networks*.

Horswill, I. 2000. Functional programming of behavior-based systems. *Autonomous Robots*, 9(1):83–93.

Howard, A., Matarić, M., and Sukhatme, G. 2002. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots Special Issue on Intelligent Embedded Systems*, 13(2): 113–126.

Howard, A., Matarić, M., and Sukhatme, G. 2003. Putting the 'I' in 'Team': An ego-centric approach to cooperative localization. In *IEEE International Conference on Robotics and Automation*. Taipei, Taiwan, pp. 868–892.

Howard, A., Parker, L., and Sukhatme, G. 2004, The SDR experience: Experiments with a large-scale heterogenous mobile robot team.

In *9th International Symposium on Experimental Robotics 2004*, Singapore.

Howard, A. and Roy, N. 2004. *Robotics Data Set Repository (RADISH)*. http://radish.sourceforge.net/index.php.

Jensen, R. and Veloso, M. 1998, Interleaving deliberative and reactive planning in dynamic multi-agent domains. In *Proceedings of the AAAI Fall Symposium on Integrated Planning for Autonomous Agent Architectures*, AAAI Press.

JESS - the expert system shell for the java platform. 2003. http://herzberg.ca.sandia.gov/jess/. Sandia National Laboratories.

Jia, J., Chen, W., and Xi, Y. 2004. Design and implementation of an open autonomous mobile robot system. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA) 2004*, Vol. 2, pp. 1726–1731.

Jones, C. and Matarić, M. 2004, Automatic synthesis of communication-based coordinated multi-robot systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sendai, Japan, pp. 381–387.

Jung, B. and Sukhatme, G. 2002, Tracking targets using multiple robots: The effect of environment occlusion. *Autonomous Robots*, 13(3): 191–205.

Kaupp, T. 2005. Orca Robotics. http://orca-robotics.sourceforge.net/.

Koker, R., Oz, C., Cakar, T., and Ekiz, H. 2004. A study of neural network based inverse kinematics solution for a three-joint robot. *Robotics and Autonomous Systems*, 49(3–4):227–234.

Konolige, K. 1997. COLBERT: A language for reactive control in saphira. In *Proceedings of the German Conference on Artificial Intelligence*. Freiburg, Germany, pp. 31–52.

Konolige, K. 2000. A gradient method for realtime robot control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotic Systems (IROS)*.

Konolige, K. 2002. *Saphira Robot Control Architecture* (Tech. Rep.). Menlo Park, CA, SRI International.

Konolige, K., Myers, K., Ruspini, E., and Saffiotti, A. 1997. The Saphira architecture: A design for autonomy. *Journal of Experimental & Theoretical Artificial Intelligence: JETAI*, 9(1):215–235.

Kraetzschmar, G., Gassull, G., and Uhl, K. 2004, July. Probabilistic quadtrees for variable-resolution mapping of large environments. In M. I. Ribeiro and J. Santos Victor (Eds.), *Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*.

Kraetzschmar, G., Sablatnög, S., Enderle, S., Utz, H., Simon, S., and Palm, G. 2000, Integration of multiple representations and navigation concepts on autonomous mobile robots. In H. Groß, K. Debes, and H. Böhme (Eds.), *Workshop SOAVE-2000: Selbstorganisation von Adaptivem Verhalten*, Vol. 10/643. Ilmenau, Germany, VDI Verlag.

Kramer, J. and Scheutz, M. 2003. GLUE—a component connecting schema-based reactive to higher-level deliberative layers for autonomous agents. In R. Weber (Ed.), *Proceedings of the 16th International FLAIRS Conference*, AAAI Press, pp. 22–26.

Labonté, D., Michaud, F., Boissy, P., Corriveau, H., Cloutier, R., and Roux, M. 2005. *Evaluation Methodology of User Interfaces for Teleoperated Mobile Robots in Home Environments* (Submitted to IEEE International Conference on Robotics and Automation).

LaFary, M. and Newton, C. 2005. Aria html Documentation.

Laukkanen, M. 1999. *Evaluation of FIPA*-Compliant Agent Platforms. Unpublished master's thesis, Lappeenranta University of Technology.

LEGO.com Educational Division—Mindstorms for Schools. 2005. http://www.lego.com/eng/education/mindstorms/default.asp. LEGO.

Lemay, M., Michaud, F., Létourneau, D., and Valin, J. 2004. Autonomous initialization of robot formations. In *IEEE International Conference on Robotics and Automation*.

Lindstrom, M., Orebäck, A., and Christensen, H. 2000, BERRA: A research architecture for service robots. In *Proceedings of International Conference on Robotics and Automation (ICRA)*, Vol. 4, pp. 3278–3283.

Logan, B. 1998. Classifying agent systems. In B. Logan and J. Baxter (Eds.), *Proceedings of AAAI-98 Conference Workshop on Software Tools for Developing Agents*. Menlo Park, California, American Association for Artificial Intelligence.

Long, M., Murphy, R., and Parker, L. 2003. Distributed multi-agent diagnosis and recovery from sensor failures. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3:2506–2513.

Lucas, G. 2004. *The Rossum Project*. http://rossum.sourceforge.net/.

MacDonald, B., Yuen, D., Wong, S., Woo, E., Gronlund, R., and Collett, T. 2003. Robot programming environments. In *ENZCon2003 10th Electronics New Zealand Conference*. University of Waikato, Hamilton.

MacKenzie, D. and Arkin, R. 1993, Nov. Formal specification for behavior-based mobile robots. *Mobile Robots VIII*, pp. 94–104.

MacKenzie, D. and Arkin, R. 1998. Evaluating the usability of robot programming toolsets. *The International Journal of Robotics Research*, 17(4):381–401.

MacKenzie, D., Arkin, R., and Cameron, J. 1997. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–52.

Maes, P. 1990. Situated agents can have goals. In P. Maes (Ed.), *Designing Autonomous Agents*. MIT Press, pp. 49–70.

Mallet, A., Fleury, S., and Bruyninckx, H. 2002. A specification of generic robotics software components: future evolutions of GenoM in the Orocos context. In *International Conference on Intelligent Robotics and Systems*, IEEE.

Matarić, M. 2004, Robotics education for all ages. In *Proceedings, AAAI Spring Symposium on Accessible*, Hands-on AI and Robotics Education.

Mayfield, J., Labrou, Y., and Finin, T. 1996. Evaluation of KQML as an agent communication language. In M. Wooldridge, J. P. Müller, and M. Tambe (Eds.), *Proceedings on the IJCAI Workshop on Intelligent Agents II: Agent Theories, Architectures, and Languages*, Springer-Verlag, Vol. 1037, pp. 347–360.

Melchior, N. and Smart, W. 2004. A framework for robust mobile robot systems. In D. W. Gage (Ed.), *Proceedings of SPIE: Mobile Robots XVII*, Vol. 5609.

Metta, G., Fitzpatrick, P., and Natale, L. 2006. YARP: Yet another robot platform. *International Journal on Advanced Robotics Systems*, 3(1):43–48.

Michaud, F. 2005. *Engineering Education and the Design of Intelligent Mobile Robots for Real Use* (Submitted to International Journal of Intelligent Automation and Soft Computing, Special Issue on Global Look at Robotics Education).

Michaud, F. and Létourneau, D. 2004. *Robotflow: Open Source Robotics Toolkit for Flowdesigner*. http://robotflow.sourceforge.net/.

Michel, O. 2004. Webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1): 39–42.

Miro - Middleware for Robots. 2005. http://smart.informatik.uni-ulm.de/MIRO/index.html. Robotics Group, University of Ulm.

Missionlab v6.0. 2003. http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab/. Mobile Robot Laboratory.

Modular Controller Architecture. 2005. http://mca2.sourceforge.net/.

Montemerlo, M., Roy, N., and Thrun, S. 2003a. *CARMEN*, Carnegie Mellon Robot Navigation Toolkit. http://carmen.sourceforge.net/.

Montemerlo, M., Roy, N., and Thrun, S. 2003b. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit. In *IROS 2003*. Las Vegas, NV, Vol. 3. pp. 2436–2441.

Moshkina, L. and Arkin, R. 2003. On TAMEing robots. In *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 4, pp. 3949–3959.

Moshkina, L., Endo, Y., and Arkin, R. 2006. Usability evaluation of an automated mission repair mechanism for mobile robot

mission specification. In *Proceedings of the ACM Conference on Human-robot Interaction (HRI2006)*, ACM.

Nesnas, I., Simmons, R., Gaines, D., Kunz, C., Diaz-Calderon, A., and Estlin, T. 2006. CLARAty: Challenges and steps toward reusable robotic software. *International Journal on Advanced Robotics Systems*, 3(1):23–30.

Nesnas, I., Wright, A., Bajracharya, M., Simmons, R., and Estlin, T. 2003. CLARAty and challenges of developing interoperable robotic software. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Vol. 3, pp. 2428–2435.

Nguyen, G., Dang, T., Hluchy, L., Balogh, Z., Laclavik, M., and Budinska, I. 2002. *Agent Platform Evaluation and Comparison* (Tech. Rep.). Bratislava, Slovakia: Pellucid 5FP IST-2001-34519.

Nowostawski, M., Bush, G., Purvis, M., and Cranefield, S. 2000. Platforms for agent-oriented software engineering. In J. Dong, J. He, and M. Purvis (Eds.), *Proceedings of APSEC 2000*. IEEE Computer Society Press, pp. 480–488.

Orebäck, A. and Christensen, H. 2003. Evaluation of architectures for mobile robotics. *Autonomous Robots*, 14(1):33–49.

Osentoski, S., Manfredi, V., and Mahadevan, S. 2004. Learning hierarchical models of activity. In *IEEE/RSJ* International Conference on Robots and Systems (IROS 2004).

Pellom, B. and Hacioglu, K. 2003. Recent improvements in the CU SONIC ASR system for noisy speech: The SPINE task. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Pfeifer, R. 1988. Artificial intelligence models of emotion. In V. Hamilton, G. H. Bower, and N. H. Frijda (Eds.), *Cognitive Perspectives on Emotion and Motivation, Volume 44 of Series d: Behavioural and Social Sciences*. Kluwer Academic Publishers, Netherlands, pp. 287–320.

Pineau, J., Montemerlo, M., Pollack, M., Roy, N., and Thrun, S. 2002. Towards robotic assistants in nursing homes: challenges and results. In T. Fong and I. Nourbakhsh (Eds.), *Workshop Notes (WS8: Workshop on Robot as Partner: An Exploration of Social Robots), IEEE International Conference on Robots and Systems*. IEEE, Lausanne, Switzerland.

Poggi, A., Rimassa, G., and Turci, P. 2002. What agent middleware can (and should) do for you. *Applied Artificial Intelligence*, 16(9–10): 677–698.

Provost, J., Kuipers, B., and Miikkulainen, R. 2004. Self-organizing perceptual and temporal abstraction for robot reinforcement learning. In *AAAI-04 Workshop on Learning and Planning in Markov Processes*.

Pyro, Python Robotics. 2005. http://emergent.brynmawr.edu/pyro/?page = Pyro. Python Robotics.

Ricordel, P. and Demazeau, Y. 2000. From analysis to deployment: A multi-agent platform survey. *Engineering Societies in the Agents World*. Springer-Verlag, Vol. 1972, pp. 93–105.

Rivard, F. 2005. *Localisation relative de robots mobiles op*érant en groupe (Tech. Rep.). Mémoire de maîtrise, Département de génie électrique et de génie informatique, Université de Sherbrooke.

Russell, R. 2004. Mobile robot learning by self-observation. *Autonomous Robots*, 16(1):81–93.

Russell, S. and Norvig, P. 2002. *Artificial Intelligence: A Modern Approach*, 2 ed., Prentice Hall.

Salter, T., Michaud, F., Dautenhahn, K., Létourneau, D., and Caron, S. 2005. Recognizing interaction from a robot's perspective. In *Proceedings IEEE International Workshop on Robot and Human Interactive Communication*, pp. 178–183.

Scheutz, M. 2004. *APOC—An Architecture for the Analysis and Design of Complex Agents* (Ed.) (Forthcoming In Darryl Davis, editor, *Visions of Mind*).

Scheutz, M. 2006. ADE—steps towards a distributed development and runtime environment for complex robotic agent architectures. *Applied Artificial Intelligence*, 20(4–5).

Scheutz, M. and Andronache, V. 2003. APOC—a framework for complex agents. In *Proceedings of the AAAI Spring Symposium*, AAAI Press, pp. 18–25.

Scheutz, M. and Andronache, V. 2004. Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems. *IEEE Transactions of System, Man, and Cybernetics Part B: Cybernetics*, 34(6).

Scheutz, M., Andronache, V., Kramer, J., Snowberger, P., and Albert, E. 2004. Rudy: A robotic waiter with personality. In *Proceedings of AAAI* Robot Workshop, AAAI Press, pp. forthcoming.

Scheutz, M., Schermerhorn, P., Kramer, J., and Middendorff, C. 2006. The utility of affect expression in natural language interactions in joint human-robot tasks. In *Proceedings of the ACM conference on human-robot interaction (HRI2006)*, ACM.

Schmidt, D. 1994. The ADAPTIVE communication environment: An object-oriented network programming toolkit for developing communication software. In *12th Annual Sun Users Group Conference*. San Francisco, CA, pp. 214–225.

Silva, A., Romao, A., Deugo, D., and Silva, M. da. 2001. Towards a reference model for surveying mobile agent systems. *Autonomous Agents and Multi-Agent Systems*, 4:187–231.

Simmons, R. 1994. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43.

Simmons, R. 2004. *Inter process communication (IPC)*. http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/TCA/www/ipc/.

Simmons, R., Apfelbaum, D., Fox, D., Goldmann, R., Haigh, K., and Musliner, D. 2000. Coordinated deployment of multiple heterogeneous robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Simmons, R., Goldberg, D., Goode, A., Montemerlo, M., Roy, N., and Sellner, B. 2003. GRACE: an autonomous robot for the AAAI robot challenge. *AI Mag.*, 24(2):51–72.

Simplified Wrapper and Interface Generator. 2004. http://www.swig.org/.

Singh, R. and Sycara, K. 2004. *Securing Multi Agent Societies* (Tech. Rep. No. CMU-RI-TR-04-02). Robotics Institute, Carnegie Mellon.

Skubic, M. and and Volz, R. A. 1998. Learning force-based assembly skills from human demonstration for execution in unstructured environments. In *Proceedings of International Conference on Robotics and Automation (ICRA98)*, pp. 1281–1288.

Sloman, A. 1998. What's an AI toolkit for? In B. Logan and J. Baxter (eds.), *Proceedings of the AAAI-98 Workshop on Software Tools for Developing Agents*, pp. 1–10.

Sloman, A. 2002. *Help Poprulebase*.

Sloman, A. and Scheutz, M. 2002. A framework for comparing agent architectures. In *Proceedings of UK Workshop on Computational Intelligence*, pp. 169–176.

SOAP version 1.2. 2003. http://www.w3.org/TR/soap12/. W3C XML Protocol Working Group.

Sprouse, J. 2005. *Nomadic.sourceforge.net*. http://nomadic.sourceforge.net/.

Steinfeld, A. 2004. Interface lessons for fully and semi-autonomous mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA) 2004*, Vol. 3, pp. 2752–2757.

Stentz, A. 2002. CD*: A real-time resolution optimal re-planner for globally constrained problems. In *Proceedings of AAAI* 2002, p. 605.

Sycara, K., Paolucci, M., Velsen, M.V., and Giampapa, J. 2003. The RETSINA MAS infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1):29–48.

Sycara, K.P. and Zeng, D. 1996. Coordination of multiple intelligent software agents. *International Journal of Cooperative Information Systems*, 5(2/3):181–212.

Tews, A., Matarić, M., and Sukhatme, G. 2003. A scalable approach to human-robot interaction. In *IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, pp. 1665–1670.

Thrun, S. 2003. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel (Eds.), *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann, San Francisco, CA, USA, pp. 1–35.

Thrun, S., Fox, D., Burgard, W., and Dellaert, F. 2000. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1–2):99–141.

Utz, H., Kraetzschmar, G., Mayer, G., and Palm, G. 2005. Hierarchical behavior organization. In *Proceedings of IROS 2005*. Edmonton, Canada.

Utz, H., Sablatnög, S., Enderle, S., and Kraetzschmar, G. 2002. Miro—middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, 18(4):493–497.

Utz, H., Stulp, F., and Mühlenfeld, A. 2004. Sharing belief in teams of heterogeneous robots. In D. Nardi, M. Riedmiller, and C. Sammut (Eds.), *RoboCup-2004: The eighth RoboCup Competitions and Conferences*, Springer Verlag.

Valin, J. and Létourneau, D. 2004. Flowdesigner. http://flowdesigner.sourceforge.net/.

Varakantham, P., Gangwani, S., and Karlapalem, K. 2002. On handling component and transaction failures in multi agent systems. *SIGecom Exch*, 3(1):32–43.

Vaughan, R., Gerkey, B., and Howard, A. 2003. On device abstractions for portable, resuable robot code. In *Proceedings of IROS 2003*, Las Vegas, Nevada, pp. 2121–2427.

Vijayakumar, S., D'souza, A., Shibata, T., Conradt, J., and Schaal, S. 2002. Statistical learning for humanoid robots. *Autonomous Robots*, 12(1):55–69.

Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., and Das, H. 2001. The CLARAty architecture for robotic autonomy. In *Proceedings of the 2001 IEEE Aerospace Conference*.

Walters, D. 2003. Open automation project (OAP). http://oap.sourceforge.net/.

Webots 5. 2005. http://www.cyberbotics.com/. Cyberbotics.

White box robotics. 2005. http://whiteboxrobotics.com/. White Box Robotics.

Wolf, D. and Sukhatme, G. 2005. Mobile robot simultaneous localization and mapping in dynamic environments. *Autonomous Robots*, 19(1):53–65.

James Kramer is pursuing his Ph.D. in Computer Science and Engineering at the University of Notre Dame, South Bend, IN, where he previously earned his M.Sc. in 2005. His primary interests concern the role of infrastructure in agent architectures, focussing on the use of reflective mechanisms in integrating system and cognitive architectures, particularly concerning their use in failure detection and recovery. These ideas are being implemented in the APOC Development Environment (ADE), of which he is a primary developer.

Matthias Scheutz received the M.Sc.E. degrees in formal logic and computer engineering from the University of Vienna and the Vienna University of Technology, respectively, in 1993, and the M.A. and Ph.D. of philosophy in philosophy at the University of Vienna, Austria, in 1989 and 1995 respectively. He also received the joint Ph.D. in cognitive science and computer science from Indiana University Bloomington in 1999. He is an assistant professor in the Department of Computer Science and Engineering at the University of Notre Dame and director of the Artificial Intelligence and Robotics Laboratory. He has over 80 peer-reviewed publications in artificial intelligence, artificial life, agent-based computing, cognitive modeling, foundations of cognitive science, and robotics. His current research interests include agent-based modeling, complex cognitive and affective robots for human-robot interaction, computational models of human language processing in mono- and bilinguals, distributed agent architectures, and interactions between affect and cognition.