2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Building a Software Architecture for a Human-Robot Team Using the Orca Framework

Tobias Kaupp, Alex Brooks, Ben Upcroft and Alexei Makarenko

Abstract— **This paper considers the problem of building a software architecture for a human-robot team. The objective of the team is to build a multi-attribute map of the world by performing information fusion. A decentralized approach to information fusion is adopted to achieve the system properties of scalability and survivability. Decentralization imposes constraints on the design of the architecture and its implementation. We show how a Component-Based Software Engineering approach can address these constraints. The architecture is implemented using Orca – a component-based software framework for robotic systems. Experimental results from a deployed system comprised of an unmanned air vehicle, a ground vehicle, and two human operators are presented. A section on the lessons learned is included which may be applicable to other distributed systems with complex algorithms. We also compare Orca to the Player software framework in the context of distributed systems.**

I. INTRODUCTION

This paper presents the design and implementation of a software architecture for a human-robot team engaged in an information-gathering task. Multiple robots cooperate with multiple human operators to fulfill the following tasks:

- collecting information using on-board robotic sensors and human perception respectively, and
- probabilistic information fusion to build a multi-attribute map of the environment.

All aspects of the system including human-robot interactions are restricted by the requirement of *decentralization* [7]. A fully decentralized approach has consequences for the design of the architecture and its implementation. In this paper, we address the problem of designing a robot software architecture which can incorporate human operators and cope with the consequences of a decentralized solution.

Information gathering refers to the problem of measuring, locating or mapping a spatially-distributed phenomenon which can change over time. Using human-robot teams is a promising approach to address the problem. Observations made by robots and humans are likely to be complementary in terms of sensor modality, uncertainty and robustness. Robotic sensors perform well in low-level descriptions such as geometric properties. In contrast, human operators can be valuable for higher-level tasks such as object recognition. Presence of such complementary information sources offers an opportunity for effective information fusion.

Application areas where human operators are likely to remain at the center of operations include search and rescue, bush fire fighting, and defence. Human involvement makes it possible to access human perceptual information to building rich environment models beyond simple geometric descriptions.

A Component-Based Software Engineering approach is adopted to address the problem of implementing a fully decentralized architecture including human and robotic entities. Orca is an open-source software project that implements a component model[1]. We will explain why we consider it well-suited to build a decentralized human-robot architecture.

The architecture is implemented as part of an outdoor information gathering mission involving four platforms: an unmanned air vehicle (UAV), a ground vehicle, and two human operators. All platforms observe the environment and fuse their observations probabilistically into a common belief [20]. Features in the environment are trees, sheds and stationary cars. Two types of information about the features are fused into a multi-attribute map: their 3-dimensional geometric position and their class (e.g."tree").

II. RELATED WORK: HUMAN-ROBOT ARCHITECTURES

There are many roles humans and robots can play when cooperating in teams. For instance, humans can interact with robots on a supervisory, peer-to-peer or mechanic level [18]. Peer-to-peer interaction is often applied to *human-robot teams* which are formed to achieve a task collaboratively [5][3]. For systems involving many robots and many human operators such as ours, only peer-to-peer interaction guarantees scalability [19][15].

A number of architectures for human-robot cooperation have been suggested recently [5][19][3]. Fong *et al.* introduce the Human-Robot Interaction Operating System (HRI/OS) as an interaction infrastructure [5]. The HRI/OS includes a set of interaction services and support for dialogue. The focus is on *operational tasks* such as collaborative seam welding and inspection. No support for data fusion is included and the dialogue mechanism does not scale to many-to-many interactions.

Scalability issues are explicitly addressed in Tews' work [19]. Here, scalability refers to the communication bandwidth between humans and robots which needs to be kept low for larger numbers of entities in the system. They propose that the amount of communication increases with decreasing robot autonomy, tighter human-robot coupling, and the number of robots. They suggest that a large-scale

All authors are with the ARC Centre of Excellence for Autonomous Systems (CAS), The University of Sydney, Australia. Corresponding author: `t.kaupp@cas.edu.au`

[1]`http://orca-robotics.sourceforge.net`

interaction mechanism must allow for many-to-many, one-to-many, and one-to-one interactions. Additionally, it must allow heterogeneous robot teams and different levels of human-robot coupling. The architecture is server-client based which is comparable to our component-based approach but it relies on a centralized server. Data fusion is not addressed in their work.

Bruemmer *et al.* developed a control architecture for a mixed team of air/ground vehicles and human operators similar in scope to our system [3]. Like us, they argue that a common substrate or *collaborative workspace* is required to maintain awareness of the environment and the mission objectives. Even though demonstrating a simple scenario of collaborative perception, the focus of their architecture is on control and adjustable autonomy.

Some of the architectures discussed above are specifically designed for a particular task or a set of tasks. Inherent coupling between parts of the architecture can cause problems when generalizing to other systems one might want to build. This is the lesson learned after our experiences with implementing previous architectures [2] and a motivating factor for our component-based approach. Rather than building specific inflexible architectures, they should be pieced together from basic and reusable software components [1]. Restricting component interaction to a set of well-defined interfaces allows this.

III. HUMAN-ROBOT TEAMS FOR DATA FUSION

This section explains what is meant by a decentralized fusion architecture and why human operators are considered to be an integral part of it.

A. Decentralized Data Fusion

The objective of the human-robot team is to build a common environment representation. Physically, robots and operators can be seen as forming a heterogeneous *Sensor Network* (SN). The components of the SN can be organized using different topologies, e.g. hierarchical or centralized distributed. The usage of a decentralized architecture has several advantages over other solutions:

- *Scalability*: the network can grow to an arbitrary number of components. This includes software components as well as embodied components (humans and robots).
- *Survivability*: no component of the system is mission-critical, so the system is survivable in the event of run-time loss of components.
- *Modularity*: all components can be implemented and deployed independently.

These advantages require that (1) no central services, facilities or components exist, and (2) no global knowledge of the network topology is needed. Our *Decentralized Data Fusion* (DDF) algorithms fulfill those requirements [7]. Scalability of the DDF algorithms has been addressed before [7][15] and is not the focus of this work.

An example of a decentralized system represented as a Unified Modeling Language (UML) component diagram is shown in Figure 1(a). Each software component (capitalized throughout the rest of the paper) has a set of provided and/or required interfaces visualized as filled circles and open semicircles respectively. Interfaces can be thought of as a contract between components to achieve inter-operability. Data is communicated between components through these interfaces.

The example shows how SENSORs and USER INTERFACEs send preprocessed observations in the form of *likelihoods* to NODEs via the `Fusing` interface. The NODEs run the DDF algorithm whose task is to maintain a consistent probabilistic estimate of the environment state. To ensure that the estimate is common to all NODEs they have to communicate with each other via the `Linkable` interface. This involves keeping track of previously communicated information to avoid *rumor propagation* [7]. The network of NODEs builds the backbone of the system, the DDF network.

B. Roles of Human Operators

In a decentralized system, the numeric relationship of humans and robots is potentially *many-to-many* [15]. Together with the other requirements of decentralization, this imposes constraints on the interaction of human operators with the SN. In general, there are two types of information which can be queried or submitted by operators: information related to the *environment* and information related to the SN's *components*. An example of environment information is the location of a feature. An example of component information is the health status of a robot.

Only environment information is scalable with respect to the size of the network, i.e. querying or submitting environment information is independent of how many components are deployed [15]. To ensure scalability, the main objectives of human interaction with the SN have to be related to environment information, namely: (1) to present the user with the global world state (operator as *information sink*); and (2) to allow human operators to contribute environment information to the network (operator as *information source*).

Other human-robot interactions which refer to component information are non-scalable but can be useful in a practical system. A summary of all objectives is given in [11]. The objectives are realized as a USER INTERFACE component exposing relevant interfaces as shown in Fig. 1(b).

IV. APPROACH

This section first presents the requirements of the system to be built. Then, Orca is introduced as a software framework which is considered well-suited to address those requirements.

A. System Requirements

The choice of employing a decentralized architecture including human operators results in requirements for the system design. Here, we present the *non-functional* requirements which include *qualities* and *constraints* of the system [16]. The requirements are organized into three categories: architecture, algorithms and implementation:

1) Architecture

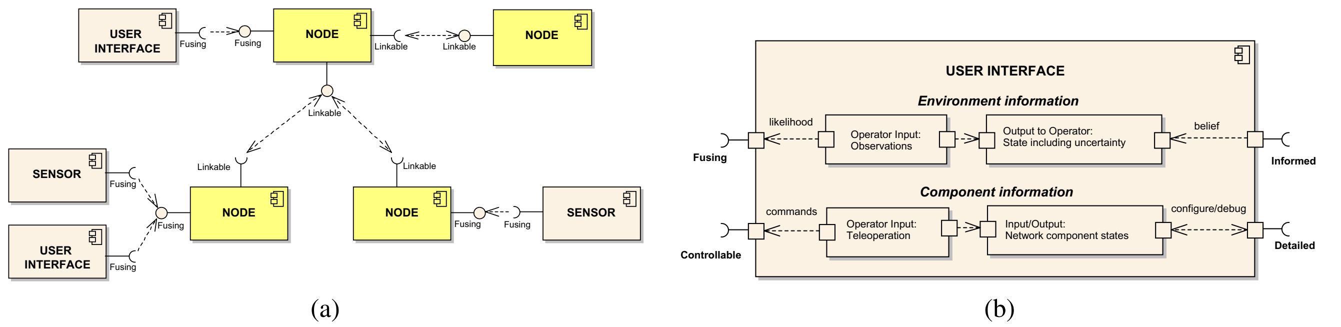

Fig. 1. (a) A decentralized data fusion architecture: SENSORs and USER INTERFACEs submit likelihoods to NODEs which form a DDF network (highlighted). Arrows indicate the direction of data flow. (b) A USER INTERFACE with four interfaces and corresponding data types is used to fulfill the requirements of human interaction with a SN.

- ability to incorporate both robotic and human entities
- support of fully distributed system scalable to many software components
- flexibility in host/platform deployment options

2) Algorithms

- ability to handle both synchronous high-frequency observations from robots and asynchronous infrequent observations from human operators
- capability to let independently developed, complex algorithms interact with each other
- incorporation of real-time processing

3) Implementation

- no central communication mechanism allowed but standard communication infrastructure must be used
- robustness to failure of software components and communication infrastructure

B. Component-Based Software Engineering

Orca is an open-source software project which applies Component-Based Software Engineering (CBSE) principles to robotics [1]. A CBSE approach has the following benefits:

1) *Modularity*: the software engineering benefits of having a modular system with controlled, explicit dependencies only.
2) *Replaceability*: the ability to build flexible systems in which individual components can be developed independently and replaced.
3) *Reusability*: the ability to build more reliable systems by incorporating components which have been tested across multiple projects.

Orca supports modularity and replaceability by providing (1) the means for defining and implementing interfaces, such that components developed independently can inter-operate and (2) the infrastructure to let components communicate with each other. Orca's basic philosophy is to impose as few constraints as possible, referred to as *design minimalism* [1]. Orca also supports reusability by maintaining a component repository.

C. Orca's Elements

The Orca project has evolved over several years and the current version is Orca2. The architecture under consideration was implemented in the previous version, Orca1. The CBSE principles are valid for both while implementation details vary.

Orca fundamentally consists of two parts: (1) the *infrastructure* to define interfaces and support the communication of components, and (2) a *component repository* of reusable components.

1) Infrastructure:

a) Middleware: Software acting as intermediary between application components is referred to as *middleware*. Orca1 was designed not to prescribe a specific middleware implementation but to let the user choose at compile-time. This has the disadvantage that no standard Interface Definition Language (IDL) can be used. To simplify interface implementation, Orca1 defined a set of *communication patterns* as methods for transferring data, and *objects* as the messages to be transferred [1].

b) Utilities: Orca provides a number of utilities to facilitate the configuration, deployment and monitoring of software components. They are especially useful for larger, more complex systems. Table I lists the utilities implemented as part of Orca1 and the problems they address.

2) Component Repository: Orca provides an online selection of open-source, inter-operable, re-useable and well-documented robotic components.

D. Why is Orca Suitable for our Problem?

We consider Orca sufficiently flexible to address the system requirements listed in Sec. IV-A:

a) Architecture level: Orca's design minimalism allows architectures ranging from single vehicles to distributed sensor networks. Especially useful is the flexibility of deploying components in any host/platform configuration (see also Sec. VI). Data fusion algorithms are computationally expensive and several host computers may be required to distribute the load. This can be achieved by deploying individual components on different hosts.

b) Algorithm level: no prescriptions are made for structuring the internals of a component. This implies that any type of algorithm can be implemented as long as it exposes

TABLE I

UTILITIES PROVIDED BY ORCA1 AND THE PROBLEMS THEY ADDRESS

Problem	Service/utility	Description
Component composition and configuration	`Gorca`	Graphical tool to define an architecture (similar to Matlab's Simulink)
Component deployment	`orcad`	Daemon automatically deploys, starts and stops components on hosts
Remote monitoring of components	`orcad`	Daemon can be queried for component status information

the relevant interfaces. Orca's communication patterns can accommodate both synchronous high-frequency and asynchronous infrequent data. Real-time processing can be incorporated if the high-frequency loop is internal to a component.

c) Implementation level: many commercial middleware options can not be used out of the box because they often rely on central services (e.g. CORBA's naming service). Orca1 provides the decentralized middleware implementation CRUD[2] which uses standard ethernet as a physical layer. To address some of the robustness issues of a larger system, Orca offers several services and utilities (see Table I).

V. EXPERIMENTS

A. Software Architecture

Fig. 2 shows the software architecture of the SN deployed for our experiments as a UML deployment diagram. All components expose provided and/or required interfaces to interoperate with other components. Some components connect to hardware shown as *artifacts*. Components are run on a number of host computers visualized by 3d-boxes. Hosts are part of a physical platform which can be a robot, a human operator, or a fixed station. Platform boundaries are indicated with dashed lines. The backbone of the system, the DDF network, is highlighted. For clarity, deployed components and interfaces related to monitoring and logging are not shown. In total, up to 20 components were running at any given time.

The figure shows the flexibility of a CBSE approach in terms of deployment options. Computationally demanding components are run on separate hosts. In contrast, components connected to hardware typically produce data at a high rate which uses up bandwidth and can cause delays if communicated. Instead, they are colocated to the component which processes their output data. On the other hand, for platforms with limited computational capabilities onboard such as the UAV, it is possible to run the components on a ground station and just communicate raw data.

B. Implementation of Robotic Components

Preexisting components from the repository which were reused for this project included IMAGESERVER, USER INTERFACE and logging/replaying facilities. The other robotic components, the LOCALISER, SENSOR and NODE needed to be implemented from the ground up. LOCALISER connects to GPS/INS hardware and provides an estimated pose of the platform. The pose estimate is sent to the SENSOR which extracts features from captured images [13]. Together with the platform pose information, it can compute a likelihood of the feature position in a global coordinate system. The SENSOR also encodes the visual properties of the features which are related to their class (e.g. "tree"). SENSORs communicate their likelihoods to NODEs which fuse them with prior beliefs. They communicate the resulting posterior to other NODEs to maintain a common belief.

[2] `http://crud.sourceforge.net`

C. Implementation of Operator Roles

All operator roles visualized in Fig. 1(b) were implemented within the project. To communicate with the SN, operators carried tablet PCs with attached handheld GPS units. Platforms were either teleoperated (UAV) or manually driven (ground vehicle). Debugging and monitoring of the software components played an important role when preparing for the final demonstration.

To submit and receive environment and component information, the USER INTERFACE was implemented graphically. Human operators were able to contribute observations of feature properties by drawing and clicking mechanisms. Observed feature properties included geometric and feature class information. Several information exchange patterns can be identified which show how robots and human operators cooperate effectively in the information fusion context. An evaluation of these results is presented in [10].

D. What Worked

The ability of the CBSE approach to build flexible systems in which individual components can be developed independently was crucial in this multi-developer project. The breakup of the system into functional components and clear interface definitions allowed the independent development of the algorithmic internals of each component. Once they had been tested individually and considered to be mature, the system was composed and tested as a whole. This modular development strategy proved to be successful in practice.

E. What Didn't Work

An important lesson learned from this project was the difficulty to scale up to many components. System composition, configuration and deployment with up to 20 components on seven hosts and five platforms soon became a tedious task. Even though `Gorca` and `orcad` were designed to address this problem, they were not sufficiently mature.

A number of problems were related to unreliable wireless ethernet connections. Communication failures are not unexpected when dealing with platforms deployed over several hundred square meters and regularly losing line of sight to each other. While DDF algorithms are designed to cope with failing communication links, it made it difficult to assess the overall state of the system.

Other faults were related to the algorithms themselves and the interaction between components. Detecting and localizing

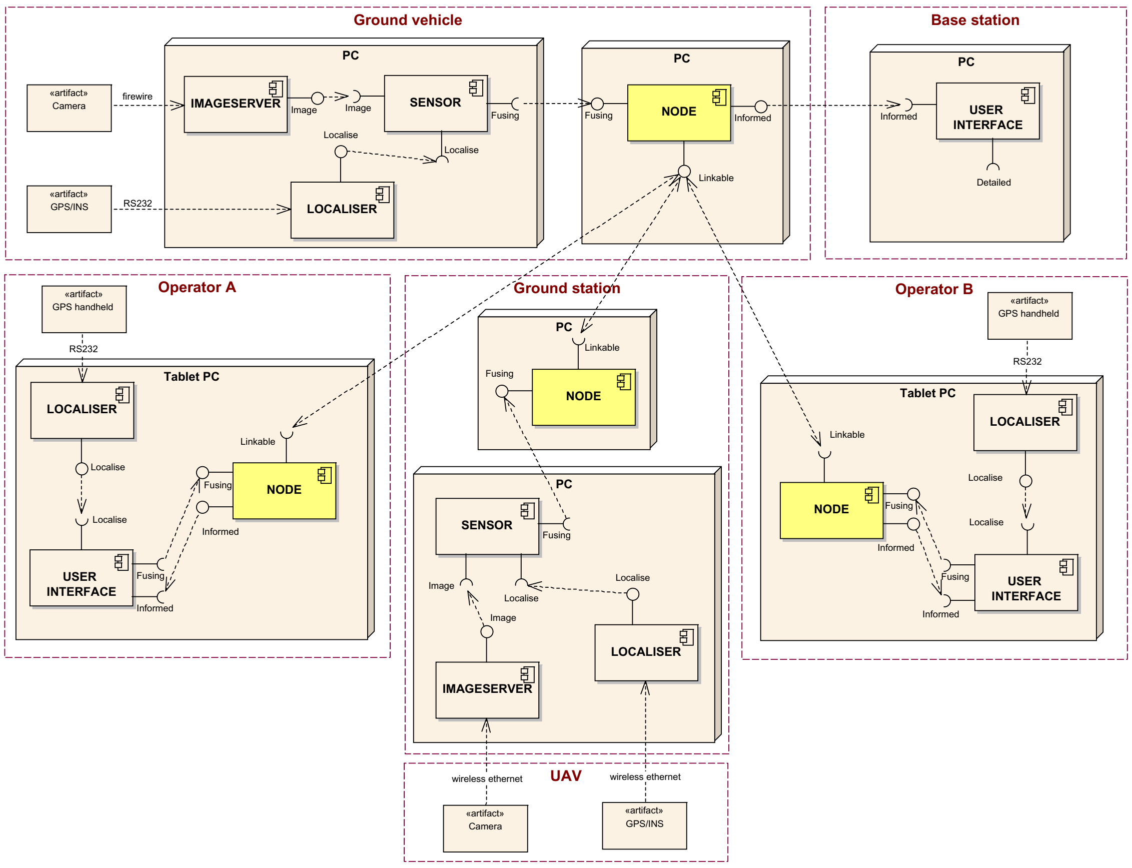

Fig. 2. The system architecture as a UML deployment diagram: components run on hosts which are part of a physical platform (robot, human operator, or fixed station). Some components are connected to pieces of hardware shown as artifacts. The DDF network is highlighted. Note the flexibility in component deployment options.

faults was problematic because, as individual component reliability increased, components would try to detect and recover from faults internally. While this behavior is desirable, the result was that overall system performance would mysteriously degrade. Even when faults were detected, it could be difficult to determine which component was at fault or even on which host the problem was occurring.

Another difficult-to-debug problem was the coupling between publishers and subscribers. Two problems can occur when communication is unreliable or clients are slow. Firstly, slow clients can delay the publishing component's thread, causing problems that appear to be related to the publisher's algorithm. Secondly, a slow client can starve faster clients, causing problems that appear to be related to the faster clients.

VI. Comparison to the Player Framework

The two most popular frameworks for implementing robotic architectures are CARMEN [17] and Player/Stage [6]. While CARMEN was designed specifically for single-robot systems, Player has been applied to distributed systems as well [9][12]. Since Player has become the de-facto standard for mobile robotics systems, we use it as a benchmark for comparison.

Player was not designed with CBSE principles in mind but is component-based in the following sense: a monolithic server houses a set of modular devices. Client components can be developed using libraries which exist in many languages. These components communicate with the server via Player's custom middleware. The most important difference to Orca is the delineation of client and server space which results in different communication mechanisms: (1) *client-server* which is the standard mechanism, (2) *inter-client* which is not defined by Player and must be implemented by the user, and (3) *inter-server* which has been introduced as part of Player 2.0 [4]. Player servers can subscribe to each other, but there cannot be any circular dependencies.

The Player model works well if there are modules exclusively exposing either provided or required interfaces and option (1) can be used naturally. For distributed systems, it is likely that *mixed* components exist with both provided and required interfaces. Consider the system shown in Fig. 3: a HARDWARE component provides `RawData` to a FEATUREEXTRACTOR which sends `Features` to a NODE which produces a `Representation`. For bandwidth and computation reasons, HARDWARE and FEATUREEXTRACTOR are colocated on the same host whereas NODE runs on a separate host.

Both FEATUREEXTRACTOR and NODE are mixed components. If implemented in client space, Player's middleware cannot be used as shown in Fig. 3(a). If implemented in

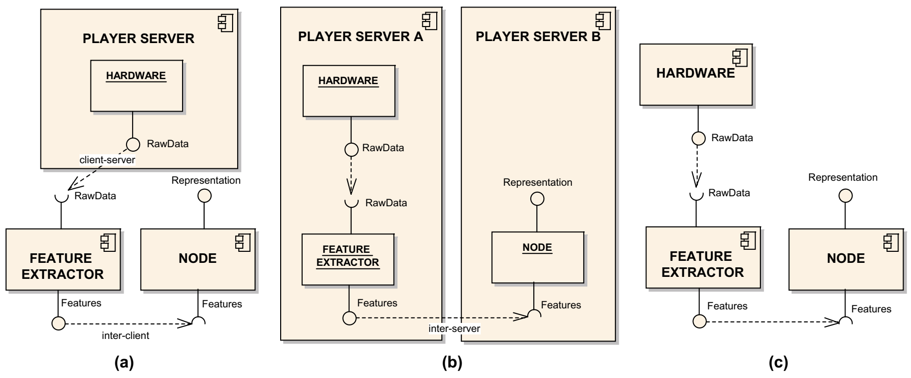

Fig. 3. A three-component system with Player (a)-(b) and Orca (c): (a) inter-client communication is custom, (b) inter-server communication works but dependencies have to be resolved on server initialization, (c) Orca has only one type of communication mechanism.

server space as shown in Fig. 3(b), Player server B needs to subscribe to a previously started Player server A on initialization. This results in a fixed startup sequence of Player servers which quickly becomes impractical when deploying a larger distributed system. Compare this to Orca's more natural approach where only one type of communication mechanism exists as shown in Fig. 3(c). Components can be developed without consideration of a distinction between client and server space, and deployed arbitrarily at run-time.

Kranz *et. al* present a recent effort of making Player a standard in the ubiquitous computing domain [12]. Their server-side implementation of a FEATUREEXTRACTOR inspired the example presented in this section. They do not address the aforementioned problem of distributing mixed components.

VII. Conclusion

The Orca framework proved to be successful to build a human-robot architecture constrained by the system requirement of decentralization. Many lessons were learned from this project which are incorporated into the design of Orca2 [14]. The major change is the adoption of the commercial middleware package Ice [8]. Besides a robust middleware implementation, Ice provides services and utilities addressing most of the problems listed in Sec. V-E. Orca2 is currently at a development stage where it can be used to implement architectures such as the one presented in this paper.

Acknowledgment

This work is supported by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales State Government.

References

[1] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Orebaeck. Orca: A component model and repository. In Davide Brugali, editor, *Principles and Practice of Software Development in Robotics*. Springer, 2006.

[2] A. Brooks, A. Makarenko, T. Kaupp, S. Williams, and H. Durrant-Whyte. Implementation of an indoor active sensor network. In *9th International Symposium on Experimental Robotics 2004*, Singapore, 2004.

[3] David J. Bruemmer and Miles C. Walton. Collaborative tools for mixed teams of humans and robots. In *NRL Workshop on Multi-Robot Systems*, pages 219–229, Washington, DC, USA, 2003.

[4] T.H.J. Collett, B.A. MacDonald, and B.P. Gerkey. Player 2.0: Toward a practical robot programming framework. In *Australasian Conference on Robotics and Automation (ACRA 2005)*, 2005.

[5] T. Fong, C. Kunz, L Hiatt, and M. Bugajska. The human-robot interaction operating system. In *Int. Conf. on Human-Robot Interaction (HRI '06)*, 2006.

[6] B. Gerkey, R. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Int. Conf. on Advanced Robotics*, pages 317–323, 2003.

[7] S. Grime and H.F. Durrant-Whyte. Data fusion in decentralized sensor networks. *Control Eng. Practice*, 2(5):849–863, 1994.

[8] M. Henning. A new approach to object-oriented middleware. *IEEE Internet Computing*, 8(1):66–75, 2004.

[9] A. Howard, L.E. Parker, and G.S. Sukhatme. Experiments with large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *Int. Journal of Robotics Research*, 25(5):431–447, 2006.

[10] T. Kaupp, B. Douillard, B. Upcroft, and A. Makarenko. Hierarchical model for fusing information from human operators and robots. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'06)*, Beijing, China, 2006.

[11] T. Kaupp, A. Makarenko, S. Kumar, B. Upcroft, and S. Williams. Humans as information sources in sensor networks. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'05)*, Edmonton, Canada, 2005.

[12] M. Kranz, R.B. Rusu, A. Maldonado, M. Beetz, and A. Schmidt. A player/stage system for context-aware intelligent environments. In *Proceedings of the System Support for Ubiquitous Computing Workshop (UbiSys 2006)*, 2006.

[13] S. Kumar, F. Ramos, B. Upcroft, and H. Durrant-Whyte. A statistical framework for natural feature representation. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'05)*, Edmonton, Canada, 2005.

[14] A. Makarenko, A. Brooks, and T. Kaupp. Orca: Components for robotics. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'06)*, Beijing, China, 2006.

[15] A. Makarenko, T. Kaupp, and H. Durrant-Whyte. Scalable human-robot interactions in active sensor networks. *IEEE Pervasive Computing*, 2:63–71, 2003.

[16] R.A. Malan and D. Bredemeyer. Defining non-functional requirements. Technical report, Bredemeyer Consulting, 2001.

[17] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2436–2441, 2003.

[18] J. C. Scholtz. Human-robot interaction: Creating synergistic cyber forces. In *NRL Workshop on Multi-Robot Systems*, pages 177–184, Washington, DC, USA, 2002.

[19] Ashley D. Tews, Maja J. Mataric, and Gaurav S. Sukhatme. A scalable approach to human-robot interaction. In *IEEE Int. Conf. on Robotics and Automation (ICRA '03)*, Taipeh, Taiwan, 2003.

[20] B. Upcroft, M. Ridley, L.L. Ong, B. Douillard, T. Kaupp, S. Kumar, T. Bailey, F. Ramos, A. Makarenko, A. Brooks, S. Sukkarieh, and H.F. Durrant-Whyte. Multilevel state estimation in an outdoor decentralised sensor network. In *10th International Symposium on Experimental Robotics 2006 (ISER '06)*, 2006.