Application of the Distributed Field Robot Architecture to a Simulated Demining Task

Matt Long, Aaron Gage, Robin Murphy and Kimon Valavanis
Center for Robot-Assisted Search and Rescue
University of South Florida
Tampa, Florida

***Abstract*— As mobile robot teams become more complex, it is necessary to develop a control architecture to manage the resources present in the team. The Distributed Field Robot Architecture (DFRA) is a distributed, object-oriented implementation of the SFX hybrid robot architecture that allows for dynamic discovery and acquisition of robot resources and the seamless integration of humans and artificial agents in the robot team. This paper introduces the DFRA and details its application to a high-fidelity demining scenario using a heterogeneous team of ground and aerial robots.**

***Index Terms*— distributed robot architectures, demining, heterogeneous agents, recruitment**

I. Introduction

Mobile robots are being used for an increasing array of tasks, from military reconnaissance to planetary exploration to urban search and rescue. As robots are deployed in increasingly complex domains, teams are called upon to perform tasks that exceed the capabilities of any particular robot. Effective management of robot teams requires a consistent means of discovering, controlling, and monitoring robot capabilities, both by human operators and artificial agents within the system.

Existing robot architectures, such as those described in [1] and [11] tend to be initially designed for single-robot systems. Parker notes in [14] that while there have been numerous advances in multi-agent robotics, these have been primarily focused on providing a specific capability to a robot team. Typically, this involves adding a distributed or multi-agent capability to an existing architecture. For example, Matarić used the subsumption architecture to examine social rules in multi-robot systems[9], as have Matarić, Sukhatme and Østergaard[10] for distributed task planning in real and simulated "emergency handling." In a similar vein, Parker has also utilized the ALLIANCE architecture[13] to explore distributed fault tolerance using a hazardous waste cleanup scenario. Similarly, the DARPA Software-Enabled Control (SEC) program is aimed at improving the control systems for autonomous aerial vehicles, and several relevant works for UAVs have come from this project, such as [6], [15], [17]. While the work of the SEC community uses a standard distributed middleware layer, little has been said about the application to multiple UAVs — the implication being that current systems are limited to a single UAV. By retrofitting single-robot systems with distributed capabilities, existing systems are limited in flexibility, extensibility, and robustness to partial failures.

The Distributed Field Robot Architecture (DFRA) is an implementation of the SFX managerial architecture [11] that enables seamless interaction of software objects and agents throughout a fully decentralized team of robots. Under the DFRA, the capabilities of each robot (such as sensors, effectors, and schemas) can be dynamically discovered and used by the robots themselves or by human operators. The DFRA has been applied to a simulated demining task on a team of ground and aerial robots, yielding positive results. We claim that the DFRA is a suitable distributed architecture for a team of heterogeneous robots. This claim is supported in the following ways: discussion of the implementation of a single-agent reactive behavior, discussion of the implementation of a deliberative, distributed recruitment system, and through the demonstration of a heterogeneous team of two ground vehicles and one aerial vehicle performing an outdoor simulated beach demining scenario.

The rest of this paper is organized as follows. Sec. II describes the Distributed Field Robot Architecture in detail. Sec. III presents a description of an application domain (a simulated demining task), and details the robot team used in field tests, along with the functionality implemented to meet this task. Sec. IV describes examples of systems that have leveraged the architecture to provide high-level control over a team of robots. Sec. V provides detail on the demonstration used to validate this work. Finally, Sec. VI provides a summary and closing remarks.

II. Architecture

The approach taken in this work is to extend the Sensor Fusion Effects (SFX) hybrid architecture to add a distributed layer that takes inspiration from the concept of a *persona* from psychology [7]. SFX serves as the base for the cognitive model, with portions of each of the relevant components extended to the distributed realm. These portions are a representation of the capabilities, attributes and knowledge

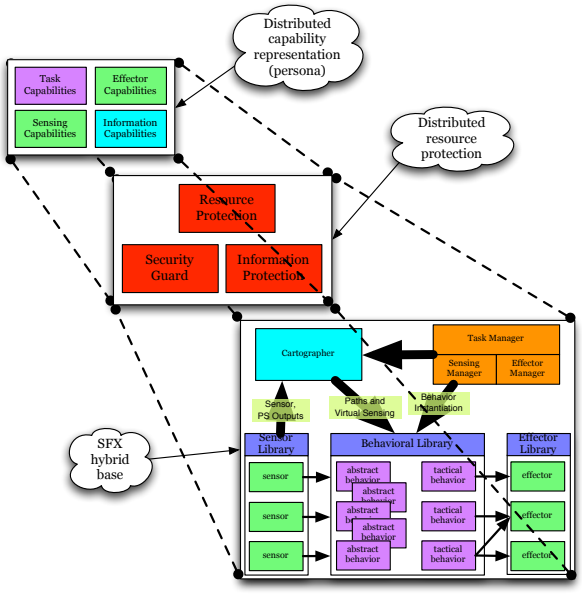

Fig. 1. Extending the original architecture by adding a distributed persona.

of the robot as they appear to the distributed system— the robot persona. The overall architecture is shown in Fig. 1.

The approach utilizes the Jini distributed architecture for the underlying middleware layer and the Java programming language and runtime environment for implementation and execution. Seven key constraints influenced the design of this system:

Behavior-based and deliberative support: The architecture must support common robotic paradigms. Behavior-based control has historically worked well for low-level, time-sensitive control, while the deliberative approach is geared toward learning, artificial intelligence and processes with weaker time constraints. This requirement is met by the inclusion of the SFX hybrid deliberative-reactive architecture as a base. SFX is discussed in more detail in Sec. II-A.

Open standards: Robot hardware and software platforms do not typically have an immensely long life — several different models of robots currently used have been discontinued by the manufacturer or the manufacturer has gone out of business. Because of this, it is important to build on a base that is open, flexible and extensible. While specific robot hardware is beyond the scope of this work, an important working requirement is that the software be built on open standards and on open source if possible. Java, Jini, XML and other core technologies are common, with large user bases and active development. Further discussion of Java and Jini can be found in Sec. II-B and Sec. II-C.

Fault tolerant: Both the overall system and individual modules should be reliable in the face of hardware faults, software errors and network problems. The use of Jini as the foundation aids in system-level fault-tolerance, while the use of SFX incorporates prior work on robot fault recovery [8].

Adaptable: The system should be able to adapt to its operating environment. Because the system is based on a Java foundation, software portability is not an issue as long as all services correctly implement specified interfaces. However, modules need to adapt to allow the network environment as a whole to function — to be good "network citizens". This may involve limiting communication and message passing to maintain sufficient bandwidth for critical services (such as live video) to function correctly.

Longevity: An ultimate goal of this system is longevity. A robot should not have to be taken out of service for the installation of changes and updates. To support this, components need to be modified, administered, logged and maintained at runtime. Dynamic class loading, a feature of the Java language, addresses this.

Consistent programming model: The implementation should abstract the locality of objects. The same method should be able to access local or remote services without sacrificing error handling or performance. While this constraint is primarily of concern for implementation, it does impact the approach taken and the conceptual model of how services are located, acquired, and used.

Dynamic system: The system should be dynamic rather than static and should be able to flexibly accommodate new sensors, effectors, or other components. This also implies that clients will be able to discover the services that are needed at runtime and adapt to the addition and removal of services over time. For a client in a general distributed computing environment, the salient characteristics of a service are the capabilities and attributes of the service. This holds as well for robotics. For example, if a robot has two identical cameras providing color images (the capabilities of the sensors), then a client will not have a preference between which camera provides an image, all else being equal. However, if the two cameras are mounted in different locations on the robot (the attributes of the sensor) then there may be differences in the client's preference for service. Thus, a service should provide a listing of its various capabilities and attributes to clients in the distributed system, allowing a client to make intelligent choices related to available services. Since the design constraints require adaptation to a changing environment, these capabilities and attributes must be changeable as the system evolves or services fail.

These constraints are addressed by the application of three key technologies: the SFX architecture, Java, and Jini. Each of these is discussed below.

A. SFX Base Architecture

SFX is a managerial architecture by Murphy[11] designed to incorporate and enhance sensor fusion. As a managerial architecture, SFX has both deliberative and reactive components. The primary component of the reactive layer is

the *behavior*. A behavior maps from some sensing percept generated by a perceptual schema to a template for motor output, known as a motor schema. A perceptual schema processes sensory data and other percepts to generate a representation of what is sensed. For example, a perceptual schema may process a camera image to generate a mine percept representing the location of the closest mine in the image.

Once a percept is generated, the behavior passes the information to a motor schema. The motor schema incorporates the control necessary to act on the percept. This may involve actions ranging moving the robot to orienting a pan-tilt unit to achieve a better view. Behaviors can act and react rapidly, allowing the robot to operate in real time.

While reactive components operate rapidly, they do not have the ability to plan or even to maintain past state. Deliberative components executing at a slower pace do have this ability, however, and many are incorporated in the SFX architecture. One example is the recruitment agent described in Sec. IV-B.

B. Java Implementation Language

The Java programming language and runtime environment serve as the foundation for this distributed system. Java is a strong choice as a base for an architecture that can control multiple robot platforms for five reasons:

- **Platform-independence** Java is an interpreted language that can be executed on any platform that runs the Java Virtual Machine (JVM).
- **Strong typing** Java is a strongly typed language, so it is possible to specify via interfaces certain actions that a class must implement. With this contract it is possible to interact with objects of the class in a known manner. Strong typing aids in system development and with error handling during system execution [16].
- **Library support** There are many software libraries available for Java that provide various functionality. Some of the most important for this work: 1) JDOM, an XML parser, 2) Java3D, a vector math and 3D visualization package, 3) a Java-MATLAB bridge and 4) a Java-CORBA library to communicate with robot control software.
- **Dynamic class loading** Dynamic class loading is a critical benefit of the Java platform, especially in a distributed scenario. Dynamic class loading allows a Java program to load classes at runtime. This enables the use of classes that may not even have been written when the Java program was started. In a distributed environment programs or services may run for extended periods of time. Robots may move around their environment and may wish to share information or code with programs on other robots. The ability to do this dynamically is vital.
- **Performance** Since Java is a byte-compiled language, it has traditionally been considered slow. Java runs through a virtual machine that interprets the bytecode stream and generates the native machine instructions to perform each operation. This interpretation step reduces performance. However, modern virtual machines include a just-in-time (JIT) compiler that compiles basic blocks of Java code to machine code the first time the block is executed. Subsequent executions will use the newly-compiled code rather than re-interpret the bytecode.

C. Jini Distributed Layer

Middleware frameworks [3] are abstractions of a distributed computing environment. These frameworks allow software developers to more easily extend system infrastructures into a distributed environment. This is because the middleware is "in the middle", between the operating system or network services and the application layer, abstracting the details of the system-specific networking and other low-level code. Jini is an example of a middleware framework [2], which has a goal of providing for spontaneous networking of services — connecting any device to any other device in the network. Jini consists of a set of specifications that describe an operational model for a Jini network and how components interact within the system. The use of a standard middleware layer such as Jini has a benefit — systems built on top of the middleware layer automatically inherit the attributes of a distributed system.

There are four primary benefits to Jini that are heavily used in this approach. First, Jini provides protocols that enable services to dynamically adapt to the network and computing environment. Second, Jini also provides a form of naming service, called the lookup service, which allows advertisement of services and availability to potential clients. Third, Jini also provides a distributed event system. Using this, a client can register interest in a service, and can be notified when that service becomes available. Finally, Jini uses a leasing mechanism to handle partial failures. This mechanism enables a client can obtain a lease on a service; when the lease expires, the client can either stop using the service or attempt to renew the lease.

The DFRA uses modular services to implement all of the capabilities of the robot, including sensors, effectors, and behaviors. The architecture also supplies high-level service managers for the coordination and functional integration of low level modular services. Modules are exported to a distributed run-time system as services with certain attributes and types. Services can then be searched for (using a distributed-object lookup service) based on functional attributes rather than details of actual implementation or physical location. This architecture allows a decoupling of client and server, providing an interface (proxy) to the requesting process in a modular fashion regardless of where the

requested service physically resides or how it is implemented at the local level. More details on the implementation of these services are provided in the next section.

III. Implementation

The DFRA has been implemented in the Java programming language using Jini to provide distributed functionality. This implementation provides robustness to partial failures that are characteristic of distributed systems and allows for the seamless addition or removal of capabilities (represented as *services*) from the robot team.

A. Domain and Task

A task domain for the DFRA was beach demining, where a heterogeneous team of robots is needed and where current architectures are insufficient. In this domain, a team of robots explores a beach for mines, finding a clear path for other vehicles or personnel to safely land and progress inland. The domain is complex, requiring systematic distributed control of a team of robots, but also requires the capability to communicate within the team and with operators at a mission control station.

An example of the cooperative mine detection task is shown in Fig. 2. In this example, two ground robots and an aerial robot perform a mission in an outdoor environment, searching for mines in a region. One ground robot performs a raster search of an area, while the other assists the VTOL with identification of a potential target. The DFRA has been successfully applied to this domain, and the hardware used in this demonstration is listed in Sec. III-B. Results from the demonstration are included in Sec. V.

B. Hardware

The DFRA has been successfully applied using a team of mobile robots: two types of robots, unmanned ground vehicles (UGVs) and unmanned aerial vehicles (UAVs), were used together for the mine detection task. These robots can be seen in Fig. 3. The UGVs were two identical iRobot ATRV Jr. platforms, each with a sensor suite containing a color camera, Sick planar laser rangefinder, Motorola M12 GPS receiver, electronic compass, internal odometry, and software to monitor wireless Ethernet connectivity.

The UAVs were a family of helicopters: Thunder Tiger's Raptor 30, 60, and 70 platforms were used, in addition to a Rotomotion Bergen that is capable of semi-autonomous flight. The Raptor 60 was alternately configured with two payload sleds. The first payload carried a pan-tilt unit for a camera, a wireless video transmitter, and batteries. This payload enabled a human operator to control the pan-tilt unit from the ground, allowing the UAV to be used for aerial reconnaissance. The second payload, which could also be equipped to the Bergen UAV, contained a camera on a fixed mount, a GPS receiver, and an onboard computer that provided realtime telemetry data back to the ground control station via wireless Ethernet.

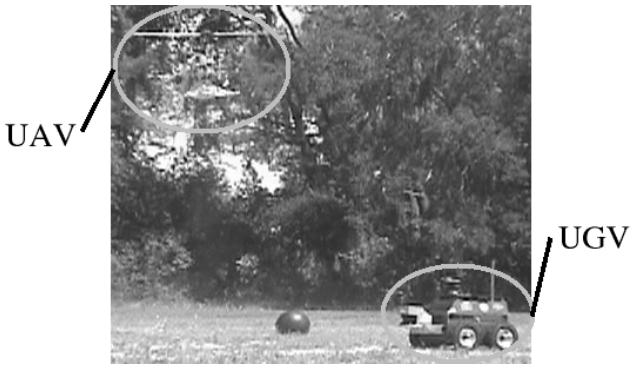

Fig. 3. Partial robot team. A UAV is visible in the upper left, and a UGV is visible to the lower right. A simulated mine, a large black sphere, is visible on the ground.

Sec. V discusses a demonstration that was performed in this domain using two ground robots and two aerial vehicles. The demonstration serves as an outdoor validation of the software architecture in a less controlled environment. Sec. III-C describes the services available on each ground robot during the demonstration.

C. Services Implemented

The distributed software system is implemented using version 1.4 of the Java programming language. The virtual machine used is the J2SE Hotspot Virtual Machine (VM). The core VM and classes (java.lang, java.io, etc.) are used as a base. The eXtensible Markup Language (XML) is used for configuration files and the architecture uses a free Java XML parser to read and write these files. Java3D is used to implement coordinate transforms and vector math, while collection classes such as HashSet and HashMap are used pervasively in the implementation. Jini is used for the distributed system architecture, and it in turn relies on Remote Method Invocation, Java Remote Method Protocol and the Java Extensible Remote Invocation for remote procedure calls. The use of pre-written, standard libraries reduced development time and increased the reliability of the system. The full list of services available is enumerated below:

1) The **GPS sensor** that generates latitude and longitude readings in WGS-84 format. The GPS was used primarily for navigation.
2) A **compass sensor** that provides heading readings in degrees. Compass data is fused with GPS location and odometry to provide pose data.
3) A **laser rangefinder service** provides range readings in meters from the front of robot used for obstacle avoidance.
4) An **odometry sensor** which produces the robot position in robot coordinates. Odometry is primarily used to

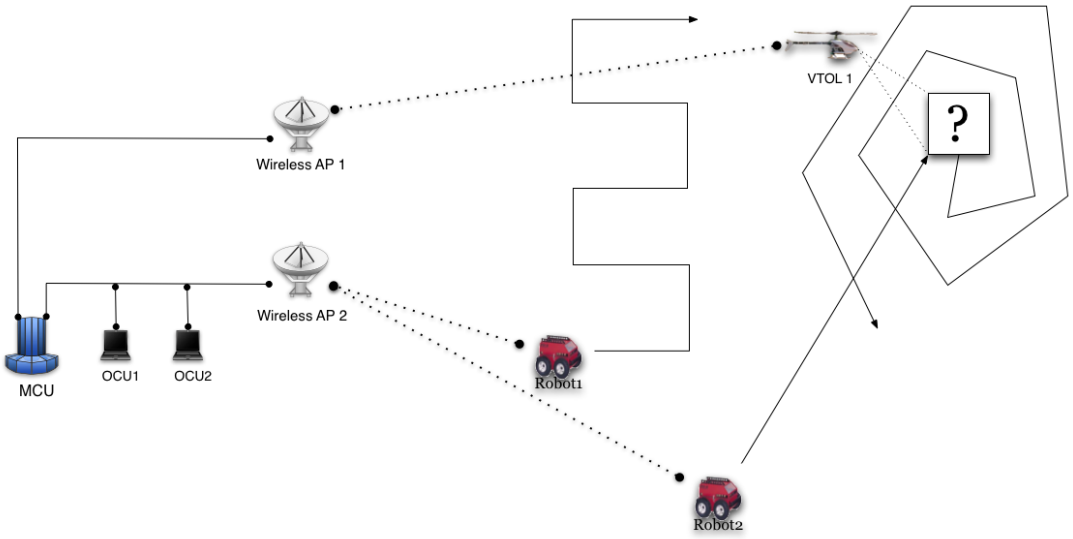

Fig. 2. The target mission scenario

filter and update GPS readings.

5) A **color camera service** that generates color images from in front of the robot.
6) A **FLIR sensor service** that generates thermal images from the front of the robot.
7) A **communication connectivity sensor** that detects connectivity with a base station. The communication services allow the robot to detect and recover from communication failure.
8) A **drive motor effector** which processes movement commands to drive the robot and provides the robot's motive force.
9) The **self-pose perceptual schema** which generates a pose percept by sensor fusion of GPS, compass and odometry data.
10) An **angle-error perceptual schema**, angle error percepts generated from target and robot poses. The angle error is fed into the motor schema's fuzzy-logic-based control scheme.
11) Filtered laser percepts processed by a sliding window filter are returned from the **filtered laser perceptual schema**. This schema removes noise and grass from the laser scan.
12) A vision-based **mine detector perceptual schema** to detect potential mines as required by the task domain.
13) The **communication status perceptual schema** monitors connectivity over time to determine overall communication state. If the state denotes failed communication for an extended period, recovery can be activated.
14) The **guarded movement motor schema**, a MATLAB-based motor schema to generate motor commands, with built-in obstacle avoidance. Internally, the MATLAB control uses a fuzzy-logic based control detailed in [12].
15) The **move-to-goal behavior** is the primary workhorse of the system and is described in Sec. IV-A.
16) The **raster search script** generates and coordinates a mine search in a raster pattern.
17) A **spiral search script** generates and coordinates a mine search in a biologically-inspired spiral pattern [5].
18) A **communication recovery script** to return a robot to base after communications loss. This script is an example of failure recovery using the DFRA architecture.
19) For affective recruitment, each robot has a **recruitment agent** that mediates recruitment requests.

Given this list of implemented functions, the next section describes the two representative examples in greater detail: the move-to-goal behavior and the recruitment agent. A demonstration of these functions in a mine-detection task is presented in Sec. V.

IV. Examples

This section describes two examples of DFRA services in greater detail. An example of a reactive service is the waypoint-following behavior described in Sec. IV-A. The architecture also supports deliberative agents, such as the recruitment agent in Sec. IV-B. These two examples serve to show the interaction of the reactive, deliberative and distributed layers of the architecture.

A. Implementing a Waypoint-Following Behavior

This subsection describes a behavior that moves the robot to a goal location specified as a GPS coordinate. The implementation of this service validates the behavior-based,

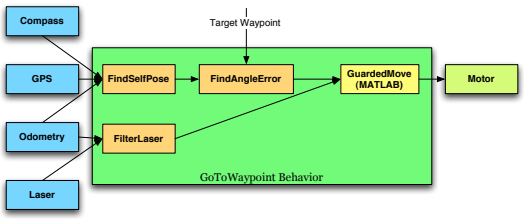

Fig. 4. Design of a waypoint-following behavior

reactive portion of the SFX architecture and also illustrates the dynamic lookup capabilities of the distributed layer.

A schematic of the behavior is shown in Fig. 4. The behavior uses four sensors: a GPS that continually updates and returns the current location of the robot in world coordinates, a compass that continually reports heading, odometry that provides the location of the robot in robot coordinates, and a rangefinder that determines distance from the sensor to obstacles over a 180° horizontal scan. Data from all four sensors is subject to noise, so additional perceptual schemas were implemented. The laser readings are filtered via a sliding-window algorithm to reduce spot noise from long grass. Current pose of the robot is generated by fusing data from the GPS, compass and odometry. This fusion generates pose information more rapidly than the 1 Hz GPS update rate by providing intermediate results based on odometry and compass readings. The pose is then compared to the target pose to generate an angle error. This, along with obstacle information, is passed to a MATLAB-based motor schema, which in turn generates motor commands for the drive system. The behavior operates at roughly 10 Hz, generating a new movement command approximately every 100 milliseconds.

During service activation, the behavior uses the distributed layer of the architecture by requesting each of the services listed above. This lookup repeatedly scans the services available in the persona of all robots in communication range, filtering the list of services based on the information and location constraints. In the lookup of the GPS service, for example, the filter removes all services that are not located on the requesting robot and that do not provide GPS position data. All other service requirements proceed in a similar manner. If the discovered service is not active, the process repeats recursively until all service requirements are met. While this behavior does not use services on remote robots, the mechanism to do so remains the same.

The implementation of this service highlights two key aspects of the system: services are discovered and requested dynamically and the system supports reactive behaviors.

B. Distributed Agents for Affective Recruitment

Gage[4] has used the DFRA to demonstrate *affective recruitment* within a team of heterogeneous robots. Affective recruitment operates at the deliberative layer of the architecture, and the recruitment algorithm leverages the distributed layer for communication.

Affective recruitment uses an emotional model to delineate when a robot will respond to a request for help sent by another agent. The recruitment model uses an affective variable to model the increasing SHAME associated with ignoring a help request. The recruitment protocol has been implemented in DFRA as modules representing agents in the distributed layer that utilize the distributed aspects of the system in the following ways:

- Each agent and a simulation GUI locate recruitment agents using the Jini lookup service. As new agents are found, the lookup service notifies each client agent about the newly discovered agent.
- Message passing is accomplished through the remote event mechanism.
- The recruitment agent determines the location of the robot by accessing the robot's GPS service and retrieving the GPS coordinates. In contrast to the behavior above, this does share sensor data between different robots.
- The Java and Jini exception mechanisms and recruitment protocol logic are used to handle partial failure.

The recruitment agent described above demonstrates the utility of both the distributed and deliberative layers of the architecture. The deliberative layer can monitor information from the distributed layer.

V. Demonstration

The Distributed Field Robot Architecture (DFRA) was tested in two field trials at the NAVSEA Coastal Systems Station (CSS) on July 1 and September 9, 2004. The purpose of these tests was to demonstrate the successful integration of a MATLAB-based motor controller, affective recruitment [4], and scripts [11] for performing a multi-robot mine detection task using the DFRA.

The demonstration used software operating both on the robots and in a mission control tent. Human operators interacted with the system through two types of user interfaces. The first interface was available through Operator Control Units (OCUs), laptop computers that each provided detailed information about a particular platform. The OCUs for the UGVs provided controls for teleoperation, live video from the robot's camera (at a rate of 10Hz when the teleoperation controls were used, and 1Hz otherwise), latitude and longitude, heading, communications status, output from the robot's mine-detection perceptual schema, and the robot's index into its active script, as shown in Fig. 5. The OCU was intended to provide a robot operator with enough information

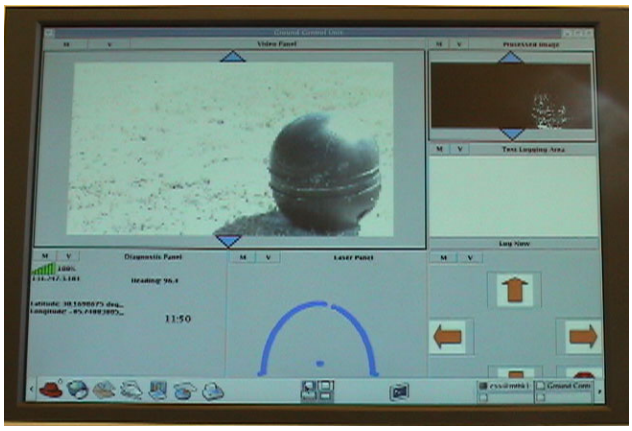

Fig. 5. Operator Control Unit user interface, providing robot-specific information.

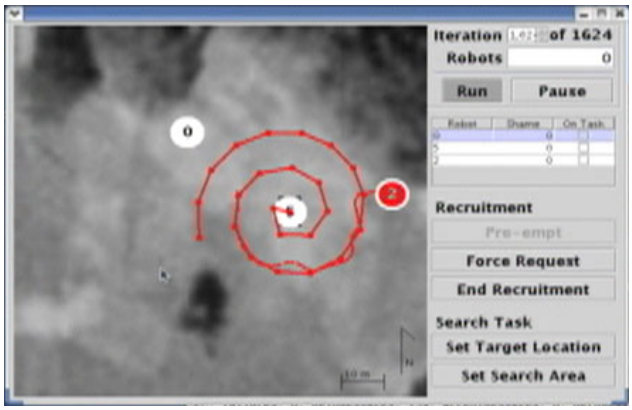

Fig. 6. Mission Control Unit user interface, providing a high-level overview of the robots in the demining scenario.

to effectively monitor the robot's status and to intervene manually if required. However, the OCU was not intended to perform mission- or team-related functions: this role was reserved for the Mission Control Unit (MCU), described below. The MCU and OCUs operated concurrently, each securing services from the robots to suit the role of the user interface.

The MCU was a single user interface that provided a high-level overview of the robot team, including the UGVs and a UAV carrying a telemetry payload, and the overall mission. Each robot that had communications contact with the MCU was displayed as an icon on an *a priori* satellite map, including its current goal waypoints and a trace of its recent positions. The MCU provided the human mission planner with the following five functions. First, the operator could select the corners of an area to search on the satellite map, and each selected robot would generate a series of waypoints to perform a raster scan over the area while using a perceptual schema to detect mines. Next, the operator could provide a single waypoint as a new home position for the robot, and if idle, the robot would attempt to reach that point. Third, the operator could pre-empt a robot from its search task. The remaining functions were related to robot recruitment: the operator could force the selected robot or robots to request assistance at their location, or to disregard their recruitment status and return to nominal operation. The MCU is shown in Fig. 6.

The experimental scenario for the robot team was as follows. A UAV with a telemetry payload would fly over a minefield, using an onboard camera and offboard image processing to identify possible mines. When an object was detected that resembled a landmine, an agent representing the UAV initiated an affective recruitment request to call an idle UGV to the UAV's position. The recruited UGV would compute a series of waypoints in the shape of a spiral, starting at the UAV's position and circling outward. In this manner, it was possible to bring the UGV near the UAV using coarse GPS data, and for the UGV to perform a search to acquire the suspected mine. When the UGV found the mine, the human operator was alerted, such that they could then investigate more closely using teleoperation from the OCU. This scenario was carried out twice in the field with an aerial vehicle successfully recruiting an idle UGV.

VI. Conclusions

This paper has presented the Distributed Field Robot Architecture and its application to a simulated demining task. DFRA is a distributed, decentralized implementation of the SFX architecture that provides consistent access to the capabilities of a team of robots in the form of Jini *services*. The DFRA is a flexible and extensible framework, into which MATLAB motor controllers and affective recruitment, a form of task allocation, have been integrated. The DFRA has been successfully demonstrated in a simulated demining task using real robots, in which an unmanned aerial vehicle observed a possible mine and recruited an idle unmanned ground vehicle to investigate more closely. The DFRA has been shown to be suitable for distributed teams of heterogeneous robots, including both ground and air vehicles, in a high-fidelity, real-world task. It is expected that the DFRA will be extended to support underwater vehicles, additional sensors, and new behaviors without any loss of flexibility or performance.

Acknowledgments

This work was supported by ONR Grant N00014-03-1-0786 and DOE Grant DE-FG02-01ER45904. The authors would like to thank NAVSEA Panama City for hosting the demonstration scenario and Colleen Cleveland for her assistance in preparing media for this paper.

REFERENCES

[1] Ronald C. Arkin. *Behavior-Based Robotics*. The MIT Press, May 1998.

[2] Ken Arnold, Bryan O'Sullivan, Robert W. Scheifler, Jim Waldo, Ann Wollrath, and Bryan O'Sullivan. *The Jini Specification*. Addison-Wesley Pub Co, 1999.

[3] Philip A. Bernstein. Middleware: a model for distributed system services. *Communications of the ACM*, 1996.

[4] Aaron Gage and Robin R. Murphy. Affective recruitment of distributed heterogeneous agents. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 14–19, July 2004.

[5] A.T. Hayes, A. Martinoli, and R.M. Goodman. Distributed odor source localization. *IEEE Sensors Journal*, 2(3):260–271, 2002.

[6] Bonnie S. Heck, Linda M. Wills, and George J. Vachtsevanos. Software enabled control: Background and motivation. In *Proceedings of the American Control Conference*, 2001.

[7] C. G. Jung. The relations between the ego and the unconscious, 1928.

[8] M. T. Long, R. R. Murphy, and L. E. Parker. Distributed multi-agent diagnosis and recovery from sensor failures. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[9] Maja Matarić. Minimizing complexity in controlling a mobile robot population. In *Proceedings of the IEEE Conference on Robotics and Automation*, 1992.

[10] Maja J. Matarić, Gaurav S. Sukhatme, and Esben Østergaard. Multi-robot task allocation in uncertain environments. *Autonomous Robots*, 14(2–3):255–263, 2003.

[11] Robin R. Murphy. *Intro to AI Robotics*. MIT Press, 2000.

[12] A. L. Nelson, L. Doitsidis, M. T. Long, K. P. Valavanis, and R. R. Murphy. Incorporation of MATLAB into a distributed behavioral robotics architecture. In *to appear in Proceedings of the IEEE / RSJ Conference on Intelligent Robots and Systems (IROS-2004)*, 2004.

[13] L. Parker. Alliance: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[14] L. E. Parker. *Distributed Automous Robotic Systems 4*, chapter 1, pages 3–12. Springer-Verlag Tokyo, 2000.

[15] James L. Paunicka, David E. Corman, and Brian R. Mendel. A CORBA-based middleware solution for UAVs, 2001.

[16] Bjarne Stroustrup. *The Design and Evolution of C++*. Addison-Wesley Publishing Company, 1994.

[17] L. Wills, S. Kannan, B. Heck, G.Vachtsevanos, C. Restrepo, S. Sander, D. Schrage, and J. V. R. Prasad. An open software infrastructure for reconfigurable control systems. In *Proceedings of the 2000 American Control Conference*, pages 2799–2803, 2000.