Application of Middleware and Agent Technologies to a Representative Sensor Network

William R. Otte, John. S. Kinnebrew,
Douglas C. Schmidt and Gautam Biswas
Institute for Software Integrated Systems (ISIS)
Vanderbilt University
2015 Terrace Place
Nashville, TN 37203

Dipa Suri
Advanced Technology Center (ATC)
Lockheed Martin Space Systems Company
3251 Hanover Street
Palo Alto, CA 94304

***Abstract*—**
Sensor networks are distributed real-time embedded (DRE) systems that often operate in open environments where operating conditions, workload, resource availability, and connectivity cannot be accurately characterized a priori. As with other open DRE systems, they must perform sequences of heterogeneous data collection, manipulation, and coordination tasks to meet specified system objectives. The South East Alaska MOnitoring Network for Science, Telecommunications, Education, and Research (SEAMONSTER) project illustrates many common system management and dynamic operation challenges in a representative sensor network, including adapting to changes in network topology, effective reaction to local environmental changes, and power management through system sleep/wake cycles. This paper discusses a case study for applying middleware and autonomous agent technologies from the Multi-agent Architecture for Coordinated Responsive Observations (MACRO) to these challenges in the SEAMONSTER sensor network.

I. Introduction

Today's scientists have an unprecedented advantage in studying and predicting weather, natural disasters, and climate change using information gathered from sensors around the globe, and quickly transmitted to central locations where significant computational resources are available for model building, data analysis, and data prediction. Unfortunately, the configurations and operations of individual sensor networks are often performed in an *ad hoc* manner, which impedes adding of new sensors, updating/modifying their software, and reconfiguring them to accommodate evolving conditions and changing science needs. These sensor networks are typically distributed real-time embedded (DRE) systems, and they face many of the system management and dynamic operation challenges that arise in DRE systems in other domains, such as shipboard computing [6] and fractionated spacecraft [2]. In these related domains, many of these challenges are being addressed through the use of *component middleware* [5], which automates remoting, lifecycle management, system resource management, deployment, and configuration. In large sensor nets, or sensor webs as they are sometimes referred to, a component middleware approach is even more critical to address the much larger assets and computational resources that need to be coordinated and managed to address weather, climate change, and disaster management problems.

Deployed hardware and sensors are also increasingly configurable and must operate in *open* environments where operating conditions, workload, resource availability, and connectivity cannot be accurately characterized *a priori*. The challenges presented by such open environments are only recently being addressed in DRE systems [15]. The combination of state-of-the-art component middleware with intelligent, local autonomy in application generation, resource allocation, and coordination provides a powerful solution approach to many system management and dynamic operation challenges facing sensor networks.

This paper presents a case study where a combination of middleware and autonomous agent technologies developed as the *Multi-agent Architecture for Coordinated Responsive Observations* [13] (MACRO) is applied to the *South East Alaska MOnitoring Network for Science, Telecommunications, Education, and Research* (SEAMONSTER) [3], a representative sensor network for monitoring of glacier discharges. The major challenges using these technologies for the SEAMONSTER project are presented along with the solution approach provided by MACRO software. Finally, we summarize important lessons learned from our initial application of MACRO software to SEAMONSTER hardware.

II. MACRO Architecture Testbed

Development of the MACRO architecture is currently being driven by the SEAMONSTER project, which provides a real-world platform for development and evaluation of middleware, resource allocation and control infrastructure, and multi-agent coordination techniques for smart sensor webs. This section describes the SEAMONSTER project, and the testbed we constructed at the Institute for Software Integrated Systems (ISIS) in Vanderbilt University using the same hardware and sensors deployed in SEAMONSTER to enable controlled studies of real-world SEAMONSTER use cases.

A. Overview of SEAMONSTER

SEAMONSTER is a NASA-sponsored smart sensor web project located in Juneau, AK. The SEAMONSTER sensor network supports collaborative environmental science with sensor data from the nearby glaciers and watersheds, which

includes timely production of relevant sensor data and goal-driven coordination within the sensor network. For example, autonomous increase in the rate of data collection for relevant sensors when an event like a glacial lake drainage is recognized.

B. Overview of the MACRO Framework

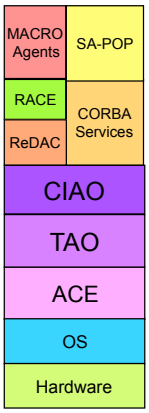

Fig. 1: MACRO Architecture

The Multi-agent Architecture for Coordinated, Responsive Observations (MACRO) architecture provides a powerful computational infrastructure for enabling the deployment and operation of large distributed sensor networks, i.e., a sensor web. The system comprises of two levels of agents: (i) the mission level, where agents interact with users to define science goals and then translate these goals into a set of prioritized tasks that have to be executed to achieve these goals, and (ii) resource level, where agents translate tasks into activities related to data collection, data analysis, and data communication. The resource agents use component middleware and novel services, such as the Spreading Activation Partial Order Planner (SAPOP) for supporting dynamic (re)planning/scheduling and the Resource Allocation and Control Engine (RACE) for resource allocation/control, to achieve the necessary local autonomy to react to changing conditions, while efficiently achieving mission goals with limited resources. The implementation of the agents is based on a state-of-the-art component middleware implementation of CORBA and the CORBA Component Model (CCM) to ensure interoperability across heterogeneous computing platforms, reduce development costs, and improve overall robustness and scalability. The agents operate on a quality of service (QoS)-enabled component middleware framework, shown in Figure 1 to ensure that a diverse set of science objectives can be met. This architecture helps overcome current limitations by facilitating real-time, reactive data acquisition, analysis, fusion, and distribution, i.e., a "smart sensing" capability in the sensor web context.

C. Overview of ISIS Smart Sensor Web Testbed

The ISIS Smart Sensor Web Testbed (ISISWEB) consists of hardware that is identical to that used in the field by the SEAMONSTER project. This hardware falls into three categories: (1) primary microservers, (2) adjunct microservers, and (3) sensor motes. The ISISWEB environment consists of two primary microservers, three adjunct microservers, and ten sensor motes.

1) ISISWEB Hardware: **Primary microservers.** The primary microservers serve as the primary gateway from the sensor network in the field to the outside world. These units, known as Vexcel/Microsoft microservers, are custom designed ruggedized cases which enclose a number of commercial off the shelf (COTS) components. The most significant features of the Vexcel/Microsoft microserver are a low-power 200 MHz ARM Single Board Computer (SBC), and a power conditioning subsystem (PCS) (also designed by Vexcel/Microsoft). The PCS consists of a micro-controller that mediates all power to devices in the microserver case, allowing the SBC to programatically determine battery state, enable and disable attached devices, and to indicate the length of sleep/wake cycles.

In addition to the SBC and PCS, the microserver case also contains a solar power regulator, a GPS unit, a wireless/Ethernet bridge, and a wireless signal booster. The case also has room for additional components, such as a wireless router. Power for the purposes of the testbed is provided by an adjustable bench-top power supply that allows simulation of low-power conditions.

Adjunct microservers. Adjunct microservers are inexpensive units intended to extend the range of the primary microservers. These units are inexpensive ARM based SBCs, which are currently Linksys NSLU-2 NAS devices, which have been re-flashed with Debian Linux and are affectionately known as "SLUGs." SLUGS do not have the fine-grained power control available to the primary microservers, but provide two USB ports for flash drives, or attached sensors. Power for the purposes of the testbed is provided by the stock power adapter, so low-power situations are not exercised on the SLUGs.

Sensor Motes. Sensor motes are extremely low power field sensors that serve as primary data sources for the primary and adjunct microservers. The SEAMONSTER project uses Moteiv tMote Sky units that have 8MHz microprocessors, and are programmed using Tiny-OS. The tMote units have on-board temperature, humidity, and light sensors as well as an expansion port and USB port that may be connected to external sensors. These motes communicate via 802.11.4 ad-hoc networks based on the ZigBee standard [1], with a mote that is directly connected to a microserver via USB acting as a base station. Power for the purposes of the testbed is provided by batteries, similar to the power mechanisms in the actual SEAMONSTER environment.

2) ISISWEB Topology: The ISISWEB testbed may be configured in one of two physical topologies that simulate different use cases in the SEAMONSTER environment. This section provides an overview of each of these topologies. For

the purposes of the testbed, both primary microservers are equipped with COTS wireless routers.

Topology with physical distribution. This topology is useful when it is possible to provide sufficient separation between groupings of microserver groupings to ensure they are unable to communicate wirelessly. As shown in Figure 2, this topology provides two separate microserver groupings. The

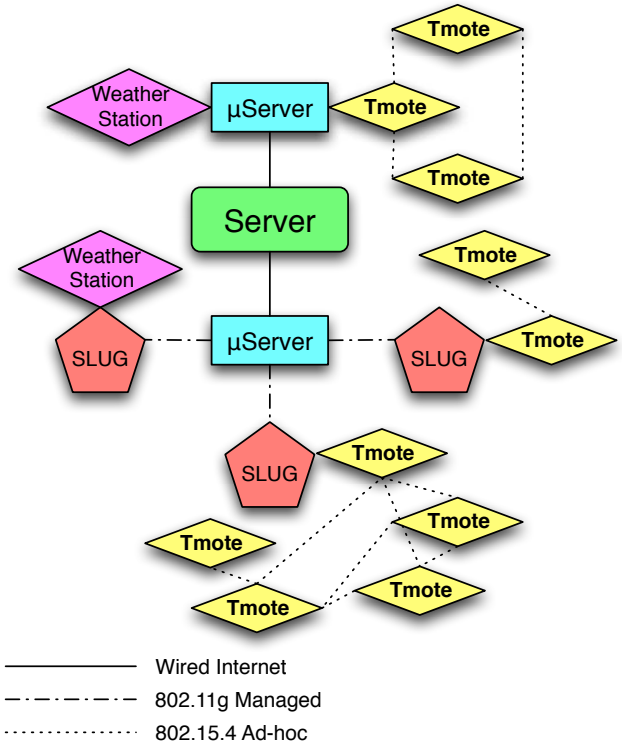

Fig. 2: ISISWEB Topology With Physical Distribution

first consists of a single primary microserver with attached sensors and a small tMote network. The second grouping consists of a single primary microserver, wirelessly connected to SLUGS which have attached tMote networks. Each primary microserver has a physical connection to the main server.

This first topology is ideal for exercising the autonomous operation and coordination of MACRO science agents. The independent groups of tMote sensors are controlled by their corresponding microservers, which must coordinate to produce relevant data products as environmental conditions change. Moreover, temporary loss of wireless links between microservers may require them to operate independently and autonomously with only the non-local information from earlier communications available.

Topology Without Physical Distribution. This topology is useful when sufficient physical separation of the microserver groupings is not feasible. This topology, shown in Figure 3, consists of a single primary microserver connected physically to the central server, with the other primary microserver connected to the first using a wireless distribution service (WDS). This topology effectively extends the range of wireless network proffered by the directly attached microserver. The SLUGs connect to this wireless network and the tMote units are organized in a single large mesh. An alternative configuration (which does not necessitate a wireless access point be included with the primary microserver) is to build the network in *ad-hoc* mode. This configuration decreases the power requirements of the wireless network, but means that nodes not within wireless range of one another (*e.g.*, the directly attached microserver and either of the SLUGS attached to the second microserver) may not communicate directly in a point-to-point fashion.

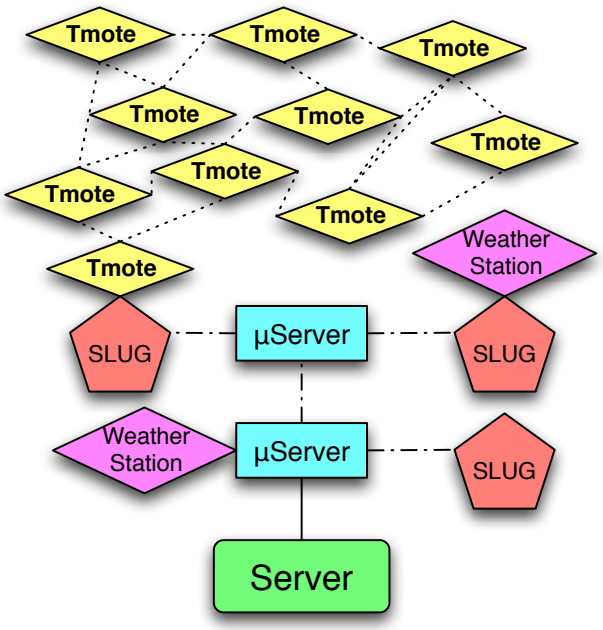

Fig. 3: ISISWEB Topology Without Physical Distribution

This second topology presents challenges similar to those presented by the previous topology, and also presents challenges to the deployment and configuration infrastructure, particularly if the network is constructed in *ad hoc* mode. These challenges arise from the potentially multiple hops required in communications from global planners or deployment infrastructure.

III. Overview of Middleware Technologies

This section describes the middleware technologies provided by MACRO that are used in the context of the SEAMONSTER case study described in Section II-A.

A. Overview of the CORBA Component Model

The CORBA Component Model (CCM) [10] is an extension to the Common Request Broker Architecture (CORBA) [7] that supports Component Based Software Engineering. CCM enhances reusability by allowing developers to focus only on application business logic, by abstracting away the details of communication and configuration. Components interact with

one another only through well-defined ports which include *facets* (provided interfaces), *receptacles* (required interfaces), and *event sources and sinks* (asynchronous publish/subscribe transport).

The CCM middleware used by MACRO is the *Component Integrated ACE ORB* (CIAO) [14], which is a quality of service (QoS)-enabled implementation of the Lightweight CCM (LWCCM) [8] specification built on top of *The ACE ORB* (TAO). CIAO provides a clear separation of concerns between *configuration logic*, specified at deployment time via XML-based meta-data, and *business logic*.

B. Overview of the Deployment And Configuration Engine

CIAO's deployment and configuration capabilities are provided by the *Deployment and Configuration of Component Based Systems* (DnC) [12] specification, which was created by the OMG in response to the need for generic and standard mechanisms for deploying component-based applications. The DnC standard contains a Platform Independent Model (PIM), which includes both a *data model* (*i.e.*, descriptions of components, component compositions, target domains, and associated configuration meta-data) and a *run-time model* (*i.e.*, a set of interfaces used to manage application life-cycle). This PIM is then mapped to a *Platform Specific Model* (PSM) for particular component middlewares. In this case, the CCM specification contains this PSM.

The DnC *run-time model* maps to a set of daemons that run at certain points in the *domain*, the collection of nodes and communication methods that comprise the target environment. Important elements of the *run-time model* include:

- **Node Manager**, which is a daemon that runs on all nodes in the domain that is responsible for deploying, configuring, and managing all components deployed to that node. This daemon also supports monitors necessary to report resource status on the node to the global planning agents. Each node in the sensor web will have a running Node Manager.
- **Execution Manager**, which is a daemon that coordinates the activities of all *Node Managers* in a given domain. This daemon is the primary point of control for the life-cycle of all component applications. Primary microservers with direct connections to the main server will have execution manager.
- **Target Manager,** which is a daemon that collates and reports resource availability in a given domain. Information is collected from resource monitors installed in individual *Node Managers*. Like the *Execution Manager*, this daemon will run on primary microservers with direct connection to the main server.
- **Repository Manager,** which is a daemon that maintains a collection of component meta-data and binary implementations. Individual *Node Managers* may contact nearby repositories to download binaries for components they are tasked to deploy, while planning agents may query the repository for information about components available for deployment. An instance of the *Repository Manager* will run on the primary server for use by the global planning agents, another instance will reside on primary microservers with direct connections to the main server for use by nodes in the field.

IV. Resolving Deployment and Configuration Challenges in Sensor Webs

This section describes our solutions to common problems that arose in the deployment and configuration of the SEAMONSTER sensor web.

A. Adapting to Changing Network Topology

Context. Sensor networks that reside in remote or inaccessible locations may suffer from frequent and unexpected changes to its network topology. These changes may be transient in nature, *e.g.*, due to limited resources such as power or temporary disruptions in wireless transmission, or permanent in nature, *e.g.*, due to damage to or destruction of physical resources.

Problem. Changes in network topology impacts the sensor web in two ways. First, loss of a particular node means temporary or permanent loss of the data stored on that node. The middleware used to facilitate implementation of the sensor web should be able to provide fault tolerant storage of data in an application independent manner. Moreover, the fault tolerance strategy may need to change when failures are detected, *i.e.*, if a node loses its mirror (a node selected to store a duplicate of its data), another node should be selected as a mirror for that node. Second, temporary or permanent loss of a wireless node may interrupt direct communication paths between nodes. As shown in Figure 4, there may be alternate paths that could be used for communication in the face of failed nodes. The middleware should be able to discover and

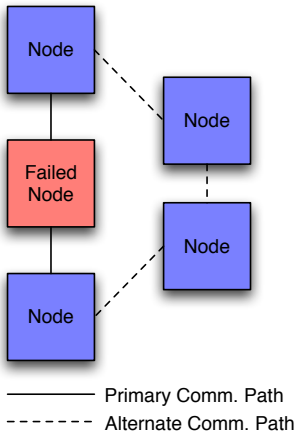

Fig. 4: Alternate Communication Paths with Failed Nodes

take advantage of these alternate paths.

Solution approach. To address the first problem, a natural approach is to make use of the asynchronous pub/sub communication ports available in the CCM middleware. Agents responsible for collecting and collating data from attached

sensors can publish noteworthy data to one of its event ports. The CCM middleware, using a pub/sub middleware such as the OMG Data Distribution Service (DDS) [11] or CORBA Real-time Notification Service [4], will transmit the data to clients connected by the DnC infrastructure. Should the global planning agent detect a failure of a data mirror, it can direct the DnC infrastructure to connect a different sink in the failed nodes instead, which shields the agents from where their mirror is located and enables seamless failover in response to faults.

The second problem is more challenging and is not directly addressed by either the CCM or DnC middlewares, as discussed in Section V.

B. Adapting to Changing Goals and Conditions

Context. Nodes in a sensor network often have a large number of observable phenomena in their area of interest. The type, duration, and frequency of observation of these phenomena may change over time, based on changes in the environment, occurrence of events in the environment, and changing goals and objectives in the science mission of the sensor web. Moreover, limited processing ability, storage, and network bandwidth limit the ability of these nodes to continually perform observations at the desired frequency and fidelity. In addition, the ability to observe certain phenomena could be influenced by changes in the environment, such as changes in the availability of sunlight to provide power due to inclement weather or variations due to time of year and season.

Problem. Dynamic changes in environmental conditions coupled with limited resource availability requires individual nodes of the sensor network to revise current operations and future plans. To handle these dynamic changes effectively, the nodes must be capable of some local, autonomous adaptation. Moreover, they must be able to adapt the local system in a goal-driven manner to maintain efficient and correct operation of the overall sensor network.

Solution approach. To address the problem of effective reaction to local changes, while respecting system-wide science goals, the nodes must be capable of intelligent, autonomous planning and action. This intelligent behavior is directed by local MACRO agents with functional knowledge of the system and software components. The local agent(s) use their planning and re-planning capabilities to adapt system functionality to local, dynamic changes, while prioritizing system activities when sufficient resources are unavailable to fully achieve all goals.

C. Ensuring Correct Re-Deployment After Reboot

Context. Limited availability of power, due to variations in weather that limits ability to re-charge attached batteries, motivates the need for power management. While powering down sensors, radios, and other attached devices is helpful, extreme low power conditions necessitates more aggressive measures. Moreover, to protect against "wedging," which is a situation where the operating system becomes unresponsive, it is advisable to reset the microservers periodically. Where possible, this approach involves large numbers of sleep/wake cycles, which consists of entirely powering down a microserver for periods of time, and performing a cold boot to restore functionality.

Problem. When the microserver returns from a sleep/wake cycle, *i.e.* when the boot process completes, agents and applications must be correctly re-deployed and connections between nodes must be correctly re-established. Correctly accomplishing these two tasks requires that (1) agent and application state be preserved across the reboot, and (2) deployment infrastructure state be preserved across reboots.

Solution approach. The current approach to solving this problem is to create all deployment as locality-constrained deployments. Locality-constrained deployments describe only components that reside on a single node and describe connections with components on other nodes with external references. This approach is in contrast to a global deployment plan, which describes components deployed to several nodes and describes connections as internal references, *e.g.*, refers to the connected components directly. This approach requires that each node have a complete DnC stack - *i.e.*, each node has both an Execution Manager and a Node Manager. Since this approach is less desirable from the standpoint of run-time footprint a superior approach is outlined in Section V.

V. Concluding Remarks

Deployment, configuration, and operation of distributed sensor networks is typically accomplished in an ad-hoc manner. This limits the ability of the system to evolution in its hardware makeup, science mission, and operating environment. This paper presents a case study that combines the MACRO agent architecture with CCM and its associated deployment and configuration infrastructure to the SEAMONSTER sensor web. In addition, we describe the ISISWEB testbed, which provides a realistic environment in which to evaluate the effectiveness of the agent-based approach.

We found that successfully applying the CIAO CCM middleware and deployment and configuration infrastructure to MACRO and SEAMONSTER requires the resolution of several challenging problems.

a) Resource Constraints: The strict resource constraints (*i.e.*, less than 64MB RAM and less than 266 MHz processors) were a significant hurdle due to the large memory footprint of CIAO components and deployment infrastructure. Previous development of CIAO has been driven largely in environments with few resource constraints, *e.g.*, more than a gigabyte of RAM and processors faster than two gigahertz.

While CCM is currently functional in the ISISWEB environment, its footprint limits the number of agents that can be concurrently deployed to a single node in the sensor network. Likewise, the easily saturated processors present problems in that it may be hard to provide reasonable deployment infrastructure responsiveness and deployment latencies.

Efforts are currently underway to reduce the footprint incurred by the CIAO middleware. In addition to careful analysis and refactoring of the code, generative specialization techniques are being investigated to prune middleware features not used by particular component implementations. Techniques for ensuring reasonable deployment responsiveness, such as using both OS priorities as well as QoS extensions to the CORBA and CCM specification are also being explored.

b) Infrastructure Fault Tolerance: The faulty nature of the SEAMONSTER environment (*i.e.*, frequent—and possibly unexpected—power cycling of nodes and the possibility that nodes lose power for long periods of time) motivates the need for better fault tolerance not necessarily of the CCM middleware (though that is useful), but of the deployment infrastructure itself. The approach currently taken to solve this problem in Section IV is coarse-grained and unnecessarily resource heavy.

Techniques for providing infrastructure level fault tolerance for the CCM deployment infrastructure are currently being investigated. These techniques, for node-level daemons, currently include both snapshot based, *i.e.* state is recorded on a non-volatile medium, and query based, *i.e.* global level daemons inform re-constituted node-level daemons of their proper state. Fault tolerance for global daemons presents more of a challenge, and may involve federation of these services to provide for active replication of state information.

c) Communications in sparse wireless networks: Situations where point-to-point communications are not possible between nodes (see Figure 4) present challenges to CIAO, which assumes the presence of point-to-point links between all nodes in the target environment. This challenge is currently avoided by using infrastructure-based wireless networks to provide routing between nodes. A more effective approach might be to leverage the CORBA Wireless Access and Terminal Mobility specification [9], which provides for features such as automatic route discovery and communications tunneling.

ACE, TAO, and CIAO are open source, and may be obtained from `http://download.dre.vanderbilt.edu`.

REFERENCES

[1] ZigBee Alliance. *ZigBee Specification*. ZigBee Alliance, q4/2007 edition, 2007.

[2] Owen Brown and Paul Eremenko. Fractionated Space Architectures: A Vision for Responsive Space. In *Proceedings of the 4th Responsive Space Conference*, Los Angeles, CA, 2006. American Institute of Aeronautics & Astronautics.

[3] D. R. Fatland, M. J. Heavner, E. Hood, and C. Connor. The SEAMONSTER Sensor Web: Lessons and Opportunities after One Year. *AGU Fall Meeting Abstracts*, pages A3+, December 2007.

[4] Pradeep Gore, Douglas C. Schmidt, Christopher Gill, and Irfan Pyarali. The Design and Performance of a Real-time Notification Service. In *Proceedings of the 10th Real-time Technology and Application Symposium (RTAS '04)*, Toronto, CA, May 2004. IEEE.

[5] George T. Heineman and Bill T. Councill. *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, Reading, Massachusetts, 2001.

[6] Patrick Lardieri, Jaiganesh Balasubramanian, Douglas C. Schmidt, Gautam Thaker, Aniruddha Gokhale, and Tom Damiano. A Multi-layered Resource Management Framework for Dynamic Resource Management in Enterprise DRE Systems. *Journal of Systems and Software: Special Issue on Dynamic Resource Management in Distributed Real-time Systems*, 80(7):984–996, July 2007.

[7] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 3.0.2 edition, December 2002.

[8] Object Management Group. *Lightweight CCM FTF Convenience Document*, ptc/04-06-10 edition, June 2004.

[9] Object Management Group. *Wireless Access and Terminal Mobility Specification*, OMG Document formal/2005-05-04 edition, May 2005.

[10] Object Management Group. *CORBA Components v4.0*, OMG Document formal/2006-04-01 edition, April 2006.

[11] Object Management Group. *Data Distribution Service for Real-time Systems Specification*, 1.2 edition, January 2007.

[12] OMG. *Deployment and Configuration of Component-based Distributed Applications, v4.0*, Document formal/2006-04-02 edition, April 2006.

[13] Dipa Suri, Adam Howell, Douglas C. Schmidt, Gautam Biswas, John Kinnebrew, Will Otte, and Nishanth Shankaran. A Multi-agent Architecture for Smart Sensing in the NASA Sensor Web. In *Proceedings of the 2007 IEEE Aerospace Conference*, Big Sky, Montana, March 2007.

[14] Nanbor Wang, Krishnakumar Balasubramanian, and Christopher Gill. Towards a real-time corba component model. In *OMG Workshop On Embedded & Real-time Distributed Object Systems*, Washington, D.C., July 2002. Object Management Group.

[15] X. Wang, D. Jia, C. Lu, and X. Koutsoukos. DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):996–1009, 2007.