

# Towards a Pattern Language for Networked Embedded Software Technology Middleware<sup>1</sup>

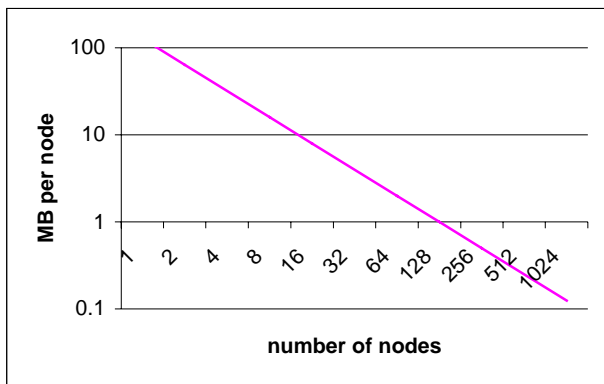
Venkita Subramonian and Christopher Gill,

Department of Computer Science, Washington University, St. Louis, MO

David Sharp, The Boeing Company, St. Louis, MO

## 1 Introduction

Networks of newly available inexpensive Micro Electro-Mechanical Sensors (MEMS) and actuators are increasingly being used in fields like advanced avionics and space systems, to reduce overall costs while allowing closer interaction between software systems and the physical world. The kinds of applications this approach supports, such as vibration suppression and material integrity assessment, require managed interactions between the physical components and the information system components, on very short time scales at a large number of physically distributed locations. This has prompted collocation and embedding of processing and control software within, or in close proximity to, fine-grained MEMS components, the number of which may vary anywhere from  $10^2$  to  $10^5$  computing nodes. Distributed Real-Time and Embedded (DRE) systems at this level of scale have been termed Networked Embedded Software Technology (NEST) [1] systems.



**Figure 1: Node Memory vs. Number of Nodes**

Constraining the complexity of software for distributed computation and communication among such large numbers of nodes motivates the need for

middleware in NEST systems. Common Off The Shelf (COTS) Distributed Object Computing (DOC) middleware such as CORBA [2] or Java RMI [3] may be optimized through techniques such as minimal feature profiles [4] or automated feature subsetting, to operate in smaller resource niches. Below a certain level, however, we expect alternative kinds of middleware to be appropriate: a key challenge in building interoperable middleware frameworks for NEST systems will be to resolve design forces *across* varying levels of scale.

A comparison of the number of nodes to the relative memory allocation at each node is useful to illustrate the levels of scale for NEST systems. For example, consider a single standard low-cost COTS computer with 128MB of memory. Partitioning that amount of memory among various numbers of nodes results in a static resource allocation curve like the one shown in Figure 1.

For the purposes of our investigation, we are focusing mainly on configurations involving ~300 nodes, each with ~500KB of memory. However, in this research we will also consider the design and implementation implications for larger numbers of nodes, each with a correspondingly smaller footprint. Other resources, such as network bandwidth, are expected to observe similar static resource allocation curves, due to price and performance trade-offs. We plan to study the implications of canonical design points for NEST along the partitioning curves of all relevant resources.

The remainder of this paper is organized as follows: Section 2 presents details of a common middleware framework we are building, and two example applications for that framework; Section 3 documents key design forces that are expected to influence the framework; Section 4 lists well-known patterns that are candidates to address some of the key design forces, and documents areas

<sup>1</sup> This work is funded under the Networked Embedded Software Technology (NEST) program by DARPA contract F33615-01-C-1898

where there is a necessity to identify new patterns; Section 5 examines related work, and suggests opportunities for additional pattern mining from these approaches; Finally, Section 6 presents our conclusions and plans for future work.

## 2 Open Experimental Platforms

The DARPA Networked Embedded Software Technology (NEST) [1] program seeks to develop and apply novel techniques for software design and implementation in the kinds of MEMS environments described in Section 1. The goal of the NEST program is to enable fine-grain fusion of physical and information processes in the manner described in Section 1.

Pursuant to this goal, the NEST program supports development of hardware and software test-beds, known as Open Experimental Platforms (OEPs). The OEPs are intended to (1) generate challenge problems for the program, (2) provide a software and hardware experimentation platform for hosting research products, and (3) support integration, testing, and evaluation of individual research technologies with respect to the stated challenge problems. Central themes of the NEST program are:

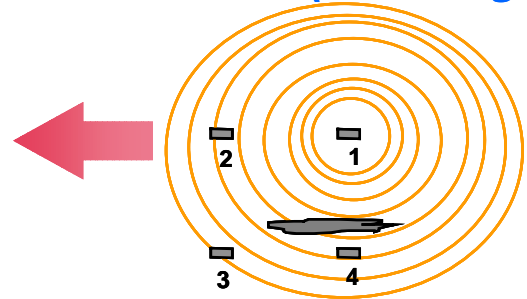
- *Coordination* services - fault tolerance, self-stabilizing protocols, data exchange, and synchronization
- *Synthesis* services - time-bounded solutions for distributed constraint satisfaction

The individual research technologies supported by the NEST program include services for application-independent coordination, time-bounded synthesis, and service composition and adaptation, that are being developed to address the principal challenges of *embedding*, *distribution*, and *coordination* of NEST applications [1].

There are currently two OEP efforts within the NEST program: (1) a Wireless Mobile Platform OEP being developed at the University of California, Berkeley [5], and (2) the Smart Structures OEP, being developed by the Boeing Company, which in contrast to the first OEP employs *wired* sensor-actuator networks.. Our work is part of the latter effort, and the discussion in the remainder of this paper pertains specifically to the Smart Structures OEP.

One target application of the Smart Structures OEP is Active Acoustic Attenuation (AAA) in enclosed cylindrical structures (*e.g.*, cryogenic tanks with thermal protection systems). Currently, passive acoustic reduction is achieved by adding vibration-damping materials to the structures, which has limited use in applications with significant weight/volume constraints. The AAA approach will use embedded, distributed, closed-loop sensor-actuator systems to minimize internal acoustic effects. It will provide active damping of both structural vibrations and internal acoustic modes, resulting in improved ratios of acoustic reduction to weight/volume. AAA is thus applicable to a range of structures, *e.g.*, launch vehicle fairings, cargo containers, aircraft fuselages, submarines, and automobiles.

### Acoustic Waves (kHz Range)



**Figure 2: Active Damage Interrogation**

A second application of the Smart Structures OEP is Active Damage Interrogation (ADI) monitoring systems. These systems help in active monitoring for damage to a critical structure, *e.g.*, the tail of an aircraft, as Figure 2 illustrates. The structure will typically have embedded or bonded piezoelectric transducers that continually transmit sensor measurements to an information processing system. The processing system in turn determines the extent of damage, if any, and takes the appropriate action by sending signals back to the actuators embedded in the structure, to effect the necessary action to control or further quantify the extent of damage.

### 3 A Taxonomy of Design Forces for the NEST Smart Structures OEP

The following design forces have been identified as playing key roles in the development of NEST systems. For each design force we describe the potential design and implementation impact on middleware for the NEST Smart Structures OEP.

#### 3.1 Distribution of Control

Several approaches to distributing control among the nodes are possible. One approach involves highly autonomous nodes with desired emergent behaviors, and little or no communication among the nodes. The benefit of this approach is that it reduces networking resource requirements, but it may do so at a cost of (1) reduced control of the exhibited behaviors, and (2) possibly increased computational resource requirements to support fully autonomous operation.

A second approach uses a peer-to-peer model, where the nodes have a lesser degree of autonomy, and perform increased communication to achieve desired coordination among the nodes. In particular, coordination and synthesis services could be implemented through message passing protocols or similar peer-to-peer communication techniques, implemented in middleware. This would improve control over the exhibited behaviors, and would still allow local autonomy, but could increase middleware configuration complexity in the face of the kinds of heterogeneous processing described in Section 3.5.

A third approach is to reduce local autonomy further, and concentrate control and coordination functions within a smaller number of dedicated higher-capacity nodes, interspersed with a much larger number of lower-capacity nodes. Assuming these higher-capacity nodes are sufficient to sustain resource-optimized COTS DOC middleware, this approach has the benefits that (1) the most effective kind of middleware can be deployed at each level of scale, (2) patterns and pattern languages can be applied to address design forces crossing the different levels of scale, and (3) the performance benefits of local autonomy still can be maintained within groups of lower-capacity nodes, compared to *e.g.*, a completely centralized model of control.

#### 3.2 Resource Management

The majority of nodes in a NEST system are typically subject to strict resource constraints. As noted in Section 1, these constraints apply to many resources, including CPU processing cycles, memory, network bandwidth, and power consumption. To achieve desired end-to-end properties such as real-time sensor-actuator control, resource management must span multiple nodes, leading to similar architectural implications to those discussed in Section 3.1

Furthermore, resources must be managed in support of multiple properties, *e.g.*, both timeliness and fault-tolerance. Middleware frameworks provide a place to reify key patterns related to each property, and to resolve the design forces stemming from each property, through application of additional patterns. Multi-property resource management requires that we identify and leverage *pattern languages* for building NEST middleware.

#### 3.3 Fault Tolerance

It is important to build redundancy into critical systems, so that they are robust in the face of various kinds of failures. Different forms of redundancy are appropriate for different types and degrees of failure handling requirements.

For example, passive replication [6] requires lower overhead to maintain, but does not offer the time-bounded exact recovery capabilities supported by more expensive forms like active replication [7]. In NEST, the design forces arising from closed loop interactions between the physical and information system components will guide such choices.

#### 3.4 Time Synchronization

Since processing is distributed across multiple nodes, synchronization may be needed among the activities carried out by the individual nodes. For example, in time-triggered architectures [8] each node conducts activities based on a local clock. In this case, clocks might be kept synchronized with clocks for all other nodes. Different forms of synchronization are possible, ranging from every node having a time source (*e.g.*, GPS time [9]) to a single time source being used to synchronize all clocks via a messaging protocol (*e.g.*, NTP [10]).

Alternatives to system-wide time-triggered approaches include merging local time consistency (*e.g.*, locally time-triggered execution) with

message-based distributed consistency mechanisms such as logical clocks to ensure consistent causal cuts across the distributed system, and support greater scalability of the system as a whole. Providing configurable strategies for different synchronization approaches in middleware will allow each application to receive services it needs without paying overhead for those it does not need.

### 3.5 Heterogeneous processing

Two main forms of heterogeneous processing are of interest. First, different nodes at the same level of scale could be used for different purposes. For example, sensors for vibration detection might be connected both to (1) nodes running vibration cancellation control laws, and (2) nodes running materials integrity assessment algorithms.

Second, as described in Section 3.1, nodes at different levels of scale and with different resource levels could be used for different purposes, *e.g.*, ubiquitous lower-capacity nodes for local control laws, and sparse higher-capacity nodes for coordination and synchronization. For both inter-level and intra-level heterogeneity, achieving configurations of middleware services that are consistent with application-level and resource-level design forces must be a major focus of our research.

### 3.6 Group membership

In addition to the static node grouping issues considered in Sections 3.1—3.5, NEST nodes could also join and leave other groups dynamically. This capability addresses design forces common to fault-tolerance (*e.g.*, replication groups), security (*e.g.*, consensus groups), timeliness (*e.g.*, synchronization groups), and high-performance (*e.g.*, autonomous messaging domains).

At the same time, group membership services require new design forces to be resolved, *e.g.*, membership privileges and consensus mechanisms. By applying design patterns to resolve these forces, we would be able to provide additional control spanning multiple levels of scale.

### 3.7 System Level Configuration Management

Several kinds of system configuration management are needed. To support the static and dynamic capabilities described in Sections 3.1—3.6, end-to-end configurations must be created.

Representing configuration strategies and parameters explicitly, and providing configuration services to achieve configurations goals implicitly, increases the applicability of NEST middleware, while reducing the complexity of managing diverse configurations.

### 3.8 Safety Criticality

Safety-critical systems need explicit assurance about system behavior and properties over time. This requirement generally translates into a need for proof or demonstration of one or more of the following: (1) sufficient test coverage, (2) *a priori* bounds on system behavior, or (3) invariants on system properties, and the conditions under which the system will operate. While our research does not target specific assurance techniques for safety criticality, design forces identified by individual researchers as leading to greater assurability will be considered in our pattern language.

## 4 Towards a Pattern Language

There is a significant body of patterns literature available, with potential application to NEST middleware. Taken together, the sets of design forces described in Section 3 intersect with design forces from the areas of *distributed*, *real-time*, *embedded*, and *pervasive* computing to define the context for a NEST OEP middleware pattern language. This section documents some of the well-known patterns from these areas and in conjunction with Section 5, suggests additional opportunities for mining relevant previously undocumented patterns. We defer discussion of how to compose the clusters of patterns from each subsection to Section 6, where we draw preliminary conclusions and identify areas of future work.

### 4.1 Distribution of Control

A canonical pattern in the area of control distribution is *Broker* [11]. DOC architectures such as CORBA employ this pattern to decouple clients making service requests from servants processing those requests. Other key patterns for architectural decomposition [11], such as *Layers* for vertical decoupling, *Pipes and Filters* for horizontal decoupling, and *Blackboard* for temporal and logical decoupling, must be applied as well.

In addition to architecture-level patterns, key strategic patterns for communication, concurrency,

synchronization, and configuration [12] must be applied to realize effective middleware frameworks. Therefore, we plan to leverage frameworks that reify these patterns as our starting point for the Smart Structures OEP middleware implementation, notably through aggressive subsetting of the ACE [13] and TAO [14] frameworks according to design forces from the NEST Smart Structures OEP.

#### 4.2 Resource Management

A number of patterns have been identified to deal with resource constraints such as CPU speed, power usage, memory size, and network bandwidth. Some of these patterns have been mined [15] from Jini technology [16]. *Lookup* [17] and *Locate & Track* [18] patterns are used to find services in an ad-hoc network. *Lazy Acquisition* [19], *Evictor* [20][21], *Leasing* [22], memory allocation and data structure patterns [23] [24] deal with efficient usage of memory in memory-constrained systems. Finally, *Pick & Verify* [25] avoids the need for a centralized manager to allocate resources in a distributed environment.

#### 4.3 Time Synchronization

Time-based coordination of actions in a distributed environment is needed in many domains. Techniques for time synchronization like heartbeat, probe/echo, and broadcast algorithms are described in [26]. New design paradigms like Time-triggered Message-Triggered objects (TMO) [27] integrate time and message triggering approaches. These different techniques are promising candidates to be mined for patterns, as we examine their applicability to the NEST Smart Structures OEP.

#### 4.4 Fault Tolerance

Patterns like *Distributed Observer*, *Recoverable Distributor*, *Distributed Scheduler* and *Distributed Lock Manager* are described in [28]. The Fault-Tolerant CORBA specification [29] uses the concepts of *Replication Manager*, *Fault Detector* and *Fault Notifier*: these abstractions and the approaches they support appear promising as potential sources of Fault-Tolerance patterns.

#### 4.5 Heterogeneous processing

The processing power of each NEST node may be different. To achieve consistency in the software that we are running on each node, it might be useful to identify what subset of the software can be run

on these resource-constrained nodes. The OMG minimumCORBA specification [4] discusses ideas on how to profile (subset) an ORB so that it can run in an embedded environment.

Another approach would be to abandon homogeneity of software, and focus instead on the minimal feature set needed in the context of each individual node. We are currently investigating the question of whether a minimal footprint CORBA ORB profile is sufficient for the NEST Smart Structures OEP, or whether possibly heterogeneous context-sensitive subsetting is more likely to result in an effective middleware framework.

#### 4.6 Group Membership

Multicast group membership protocols can be investigated to determine whether any of these could be used in an embedded environment. Bartoli [30] and Moser, *et al.*, [31] discuss various group membership approaches. We plan to mine these approaches for patterns that are composable with other patterns to form an integrated pattern language for the NEST Smart Structures OEP.

#### 4.7 System level reconfiguration

Dynamically reconfigurable networks have been of great interest in the mobile computing community. Although as we mention in Section 2 we are not focusing on wireless applications, leveraging work in this area appears promising. Concepts like *explicit assumption* and *intelligent adaptation* are described in [32]. Self-evolving software systems are also described in [33]. We plan to investigate whether it is possible to glean fundamental patterns from this work as well.

## 5 Related Work

Several areas of related work are relevant to the NEST Smart Structures OEP middleware framework. Some of these areas, notably Jini, have been mined for patterns previously, though perhaps not from a NEST OEP middleware perspective. The rest of this section is structured as follows: Section 5.1 describes Jini [16] and its potential for additional pattern mining; Section 5.2 describes the JavaSpaces [34] framework; Section 5.3 considers Real-Time Java [35][36] issues; Finally, Section 5.4 examines the OMG Smart Transducers RFP [37] and Response [38].

## 5.1 Jini Technology

The Jini [16] architecture lets programs use services in a network without knowing anything about the wire protocol that each service uses. For example, one implementation of a service might be XML-based, and another RMI-based, and a third CORBA-based. A service is identified by its programming API, declared as a Java programming language interface.

As noted in Section 4.2, many patterns have been mined from the Jini environment. We believe additional strategic and tactical patterns can be found through further investigation of (1) the ability of the Jini approach to address NEST-specific design forces, and (2) implementation details that shed light on the tactical patterns relevant to the NEST Smart Structures OEP middleware.

## 5.2 JavaSpaces

The JavaSpaces [34] service specification provides a distributed persistence and object exchange mechanism for code written in the Java programming language. It reifies the *Blackboard* [11] pattern, serving to decouple objects both temporally and logically. Clients can perform simple operations on a JavaSpaces service to write new entries, lookup existing entries, and remove entries from the space. Using these tools, also it is possible to write systems that use flows of data to implement distributed algorithms. Our NEST Smart Structures OEP middleware will likely focus on more explicitly coupled communication between objects, although JavaSpaces patterns will be integrated, as appropriate, into our design.

## 5.3 Real-Time Java

The Real-Time Specification for Java (RTSJ) [35] addresses revisions, extensions, and modifications to the Java programming language and its semantics, to support real-time system predictability, performance, and capabilities. It addresses real-time issues such as scheduling, memory, synchronization, asynchronous events, and physical memory access. In addition, further extensions in support of real-time distributed systems are under study by the Distributed Real-Time Specification for Java (DRTSJ) [36] Expert Group. We plan to mine both the RTSJ and the DRTSJ for our work on the NEST Smart Structures OEP.

## 5.4 OMG Smart Transducers

The OMG Smart Transducers RFP [37] and Response [38] focus on devices comprising sets of smaller devices clustered together—e.g., sensors and actuators, a micro-controller, a communication controller and associated software for signal conditioning, calibration, diagnostics, and communication. The RFP and response seek to provide a smart transducer interface that supports a standardized set of services to operate, configure and diagnose the transducer device, and encapsulate the internal complexity of the smart-transducer. Encapsulating failure modes, providing external services, and achieving small delay and minimal jitter over low bandwidth channels, are all areas of the Smart Transducers work where we plan to mine patterns for the NEST Smart Structures OEP.

## 6 Conclusions and Future Work

We have presented a number of patterns that appear relevant to the NEST Smart Structures OEP middleware, and a preliminary clustering of those patterns along categories of design forces. We plan two areas of future work to move to a coherent pattern language. First, we plan to establish a nascent pattern language from which to expand. Second, we plan to evolve that pattern language by reifying it as a middleware framework, and iteratively applying new patterns to address unresolved design forces at each step.

## 7 References

- [1] DARPA BAA #01-06, *Networked Embedded Software Technology (NEST)*, [www.darpa.mil/ito/Solicitations/CBD\\_01-06.html](http://www.darpa.mil/ito/Solicitations/CBD_01-06.html)
- [2] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG document formal/2001-02-33
- [3] JavaSoft, *Java Remote Method Invocation*, [java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html](http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html)
- [4] Object Management Group, *minimumCORBA submission*, OMG document orbos/98-08-04
- [5] David Culler, *Secure Language-Based Adaptive Service Platform (SLAP) for Large-Scale Embedded Sensor Networks*, NEST Kickoff Meeting June 5, 2001, [www.cs.berkeley.edu/~culler/next/](http://www.cs.berkeley.edu/~culler/next/)

- [6] R. Guerraoui and A. Schiper, *Software-Based Replication for Fault-Tolerance*, IEEE Computer, 30 (4), 1997, pp. 68-74
- [7] K. H. Kim, C. Subbaraman, *Fault-tolerant real-time objects*, Comm. of the ACM, 40 (1), 1997, pp. 75 - 82
- [8] C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, C. Temple, *Time-Triggered Architecture - (TTA)*, Advances in Information Technologies: The Business Challenge, IOS Press, 1997, pp. 758-765
- [9] B. Hofmann-Wellenhof, H. Lochtenegger, J. Collins, *Global Positioning System, Theory and Practice*, 2<sup>nd</sup> ed., Springer-Verlag, New York, 1993
- [10] *Network Time Protocol, version 3: Specification, Implementation, and Analysis*, RFC 1305, March 1992
- [11] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, *Pattern-Oriented Software Architecture—A System of Patterns*, Wiley, 1996.
- [12] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture (Vol. 2): Patterns for Concurrent and Networked Objects*, Wiley, 2000.
- [13] *ADAPTIVE Communication Environment (ACE)*, [www.cs.wustl.edu/~schmidt/ACE.html](http://www.cs.wustl.edu/~schmidt/ACE.html)
- [14] *The ACE ORB ( TAO )*, [www.cs.wustl.edu/~schmidt/TAO.html](http://www.cs.wustl.edu/~schmidt/TAO.html)
- [15] The Jini Pattern Language Workshop, OOPSLA Conference, Minneapolis, MN, USA, Oct. 15-19, 2000, [oopsla.acm.org/oopsla2k/fp/workshops/01.html](http://oopsla.acm.org/oopsla2k/fp/workshops/01.html)
- [16] Sun Microsystems, *Jini Network Technology*, [www.sun.com/jini/](http://www.sun.com/jini/)
- [17] M. Kircher, and P. Jain, *Lookup*, European Pattern Languages of Programs conference, Kloster Irsee, Germany, July 5-9, 2000.
- [18] B. Cohen, *Locate & Track Pattern*, The Jini Pattern Language Workshop, OOPSLA conference, Minneapolis, Minnesota, USA, October 15-19, 2000.
- [19] M. Kircher, *Lazy Acquisition*, European Pattern Languages of Programs conference, Kloster Irsee, Germany, July 4-8, 2001.
- [20] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*, 1999, Addison Wesley, pp. 570-588.
- [21] P. Jain, *Evictor*, Pattern Languages of Programs conference, Allerton Park, IL, September 11-15, 2001
- [22] P. Jain and M. Kircher, *Leasing*, Pattern Languages of Programs conference, Allerton Park, IL, Aug. 2000
- [23] J. Noble, and C. Weir, *Small Memory Software*, Addison-Wesley, 2000
- [24] M. Kircher and P. Jain, *Ad Hoc Networking Pattern Language*, European Pattern Languages of Programs conference, Kloster Irsee, Germany, July 4-8, 2001.
- [25] M. Wischy, *Pick & Verify Pattern*, The Jini Pattern Language Workshop, OOPSLA conference, Minneapolis, Minnesota, USA, October 15-19, 2000.
- [26] Gregory R. Andrews, *Paradigms for process interaction in distributed programs*, ACM Computing Surveys. 23(1) (Mar. 1991), pp. 49 – 90.
- [27] K.H. Kim, *Real-Time Object-Oriented Distributed Software Engineering and the TMO Scheme*, Int'l Jour. of Software Engineering & Knowledge Engineering, Vol. No.2, April 1999, pp.251-276
- [28] N. Islam and M. Devarakonda, *An Essential Design Pattern for Fault-Tolerant Distributed State Sharing*; Communications of the ACM 39(10), 1996, pp. 65 – 74
- [29] Object Management Group, *Fault Tolerant CORBA Specification*, OMG Document orbos/99-12-08.
- [30] A. Bartoli, *Group-Based Multicast and Dynamic Membership in Wireless Networks with Incomplete Spatial Coverage*; Mobile Networking Appl. 3(2), Aug. 1998.
- [31] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, C. A. Lingley-Papadopoulos, *Totem: a Fault-Tolerant Multicast Group Communication System*; Communications of the ACM 39(4), Apr. 1996
- [32] J. Inouye, J. Binkley and J. Walpole, *Dynamic Network Reconfiguration Support for Mobile Computers*, Third annual ACM/IEEE International Conference on Mobile Computing and Networking, Sept–., 1997
- [33] Chrysanthos Dellarocas, Mark Klein and Howard Shrobe, *An Architecture for Constructing Self-Evolving Software Systems*, Proceedings of the Third International Workshop on Software Architecture, 1998
- [34] Sun Microsystems, Inc., *JavaSpaces Technology* [java.sun.com/products/javaspaces/index.html](http://java.sun.com/products/javaspaces/index.html)
- [35] Java Community Process, JSR 1, *Real-Time Specification for Java*
- [36] Java Community Process, JSR 50, *Distributed Real-Time Specification for Java*
- [37] Object Management Group, *Smart Transducers RFP*, orbos/00-12-13, Dec 15, 2000
- [38] Object Management Group, *Smart Transducers Interface*, orbos/2001-06-03, Jun 18, 2001