

Using QoS-Adaptive Coordination Artifacts to Increase Scalability of Communication in Distributed Multi-Agent Systems

John Zinky, Sarah Siracuse, Richard Shapiro

BBN Technologies

jzinky@bbn.com, ssiracus@bbn.com, rshapiro@bbn.com

Abstract

Coordination Artifacts in Multi-Agent Systems (MAS) offer a coherent abstraction of communication among agents. Coordination Artifacts are first-class entities that encapsulate the coordination activity outside of the agents themselves. The separation of domain knowledge into agents, and communications knowledge into Coordination Artifacts, allows the decoupling of processing from coordination. Pulling out coordination allows for adaptation to systemic constraints and changing Quality of Service (QoS) requirements. Tailoring the QoS-adaptation to the kind of coordination and to the availability of underlying communication resources increases the scale of inter-agent communication, both in terms of the range of communication bandwidths supported and the propagation delays. In this paper, we extend the notion of Coordination Artifacts to include adaptation to QoS requirements. We illustrate how to construct QoS-adaptive Coordination Artifacts in an existing MAS infrastructure (Cougaar). Finally we show how Coordination Artifacts can be used to increase scalability in an agent-based distributed control application.

1. Introduction

Coordination artifacts in Multi-Agent Systems (MAS) offer a coherent abstraction of communication among intelligent agents. Specifically, Coordination Artifacts offer (i) a context that permits the modeling of the environment in which agents communicate, and (ii) a coordination medium in which to mitigate the behavior of those interactions. [1,10] This is an *objective* coordination model [1,4,13], where the inter-agent communication is external to the intelligent agents themselves. Coordination artifacts are *first-class entities* on a par with agents within the system, providing at runtime a coordination infrastructure or service, which the agents can exploit. [4, 10] Separating coordination from processing allows Coordination Artifacts to deal with systemic issues of inter-agent communication. A

Coordination Artifact's set of inherent properties (specialization, encapsulation, malleability, and controllability [1]), map well to the requirements needed to make scalable MAS-based societies for controlling physical systems. In contrast to their middle-agent counterparts, these specialized properties of a Coordination Artifact are typically not found in a *subjective* MAS [4, 14], comprised solely of intelligent agents, because "agents are generally supposed to be autonomous, pro-active, situated entities that interact by means of a general-purpose and high-level communication language." [9]

In this paper, we extend the definition of Coordination Artifact to add *QoS-adaptive* features. We then apply our design of this artifact to the problem of making scalable control societies for physical systems. We will use an existing agent environment (Cognitive Agent Architecture, *Cougaar* [15]) as an architectural example to illustrate the construction of Coordination Artifacts and to show their respective differences in implementation. We will end with a brief discussion about how *Cougaar* can be extended to make the construction of QoS-adaptive Communication Artifacts.

Scalability of coordination has many dimensions, apart from an increase in the number of agents [3]. The task the society is expected to accomplish affects the choice of coordination strategy. Real-world tasks tend to have complicated interactions that are not found in abstract tasks. Also, the properties of the solution dictate the kind of coordination used. Adding QoS requirements to the solution properties puts demands on the robustness and overhead of mechanisms used in the coordination. Scaling agent societies into more demanding tasks and solutions shows the limitations of coordination, especially those that use only subjective mechanisms. We will use agent societies that control a physical environment as an example of an application that demands scaling in all dimensions, including the size of the agent population, task complexity, and the solution's QoS requirements.

The remainder of this paper is outlined as follows: Section 2 describes the existing forms of inter-agent communication in MAS and the varying degrees of coordination. Section 3 describes the details, requirements, and features of a Coordination Artifact, and their association with existing middleware components. Section 4 discusses the implementation of Coordination

Artifacts. Section 5 goes on to define a MAS-based control-plane in which agents operate, and describes the physical nuances to which a Coordination Artifact could be specifically applied. Section 6 extends Coordination Artifacts to be QoS-adaptive. Section 7 shows how to construct such a reactive Coordination Artifact in an existing QoS-middleware: the Cognitive Agent Architecture (Cougaar [15]). Finally, Section 8 concludes with suggestions for improvements to QoS-adaptive Coordination Artifact design and implementation.

2. Inter-Agent Communication in MAS

We begin with a definition of coordination in complex agent systems. The classic definition of coordination is “managing and constraining interaction”. [10] In MAS, this is aptly termed *agent coordination context*, which is defined as “a means to model and shape environment in agent systems.” [10] Two complementary forms of coordination exist among agent systems, those being *subjective* and *objective*, defined as coordination *inside* and *outside* of the agents, respectively. [4] Subjective coordination describes a model whereby the agents directly observe behavior of other agents and form their own policies of communication. FIPA Agent Communication Language (ACL) [7] aligns itself with subjective coordination. In objective coordination, agents observe the behavior not of a particular other agent, but of the working environment in which agents exist. [9] Both of these communications models are used in agent systems; it is an engineering choice as to which to use in the varying MAS contexts. Figure 1 visualizes the difference between models.

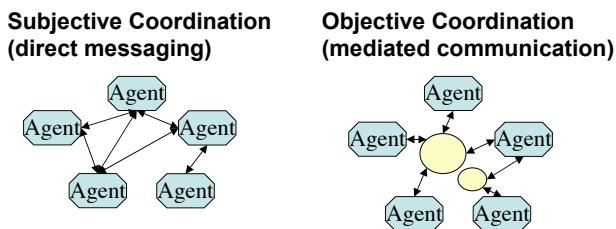


Figure 1: Existing Coordination Models in MAS

Scaling becomes a critical issue when choosing between coordination models. For example, if each agent interacts with every other agent, the number of paired interactions will grow quadratically with the number of agents [3]. Creating a scalable coordination medium involves the following: maximizing the management of agent interactions [9], shaping and constraining the agents’ interaction space, embedding a coordination policy in some constructive building block that we can build upon. It is within this context that we present the Coordination Artifact as a viable incarnation of effective coordination.

3. Definition of the Coordination Artifact

Coordination Artifacts are infrastructure abstractions that can be conceived as “embodied entities specialized to provide a coordination service in a MAS.” [1] Their primary function aims at reifying and managing agent-communication events via a computational model that contains these key features: specialization, encapsulation, malleability, and controllability; properties that are foreign (and sometime contrary) to agents. [1] “The most obvious embodiment of the notion of artifact for agent coordination is represented by a dedicated abstraction, provided at design time by the coordination model, and enacted at runtime by the corresponding coordination infrastructure” [9] Their specialization and necessary specialized entity, makes Coordination Artifacts a “first-class entity” of a MAS, complete with its own set of structural properties. In the basic model, a Coordination Artifact is characterized by having (i) a *usage interface*, describing physical acts of their interaction schema, (ii) a set of *operating instructions*, describing the use of the artifact and interaction semantics, and (iii) a *coordination behavior specification*, in which to describe the artifact’s formal behavior. [1] Figure 2 illustrates the structure of a Coordination Artifact.

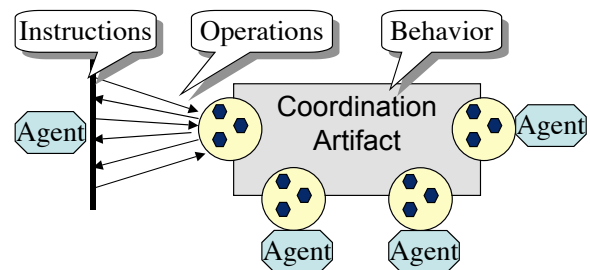


Figure 2: Coordination Artifact Characteristics

Agents interact with the artifact by performing operations on Coordination Artifacts. The operations are constrained by the operating instructions. If the operating instructions are followed, the Coordination Artifact will exhibit some coordinating behavior. An agent can use Coordination Artifacts to interact with other agents or with the environment itself. As such, the agent becomes a hub of activity for processing information presented to and extracted from communication artifacts. The agent does not have to worry about the mechanics of coordination but can concentrate on its domain processing. Figure 3 shows an agent’s various Coordination Artifacts.

4. Implementation of Coordination Artifacts

Objective coordination has many implementations in MAS. TuSCoN [8,11] is a typical implementation of Coordination Artifacts that is based on the use of *tuple-centers*. TuSCoN is a descendant of Linda, which made

popular the distributed *Tuple-spaces* paradigm. [19] Tuples are typed structures with a vector of typed data fields. Many distributed clients can add or remove tuples from a logically centralized tuple space. TuSCoN extends this model into a tuple-center, which adds Prolog-like rules to process tuples as they are added and removed from a center. The rules become the behavior for the center. The tuple-center works well for agent-to-agent coordination, but does not directly support QoS-adaptation.

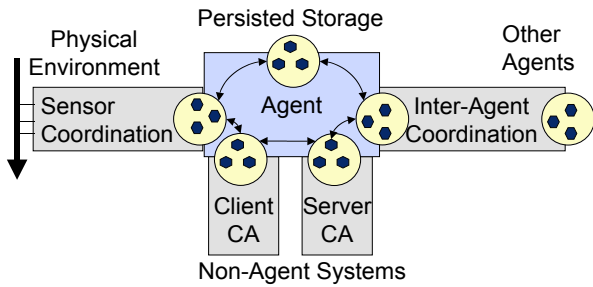


Figure 4: Agent Processing Data from Coordination Artifacts

In practical terms, coordination with the physical environment involves interfacing agents to existing software libraries. The libraries give access to some feature of the physical environment, and are usually too complicated or proprietary to modify. The libraries offer a service through which a client can observe or affect the physical environment. When the library is implemented using a component-based system such as Java Beans, the services are managed through the use of a Service Oriented Architecture (SOA). While an SOA offers an advantage over traditional language-based libraries, they are still too low level to effectively implement agents directly, for example, because of threading issues involving synchronous calls versus asynchronous callbacks. So an additional glue-like layer is needed to convert from SOA services to the software style used to implement the agent, such as a blackboard. Thus, a Coordination Artifact must encapsulate three layers, the physical environment, the existing interface libraries, and infrastructure glue. Figure 4 shows how Coordination Artifacts can be implemented to interface between various kinds of external environments.

The generic nature of Coordination Artifacts makes them ripe as an abstraction in agent-based middleware: they can potentially unify the interface between inter-agent and agent-to-environment communication. The blackboard-like interface to tuple-centers is a good model for the internal implementation of an agent that interacts with multiple Coordination Artifacts. The implementation of the coordination behavior is less clear. While TuSCoN's general programming language approach is elegant for defining abstract behaviors, it offers little pragmatic support for interfacing to grungy physical environments. To expound upon issues for implementing coordination to physical systems, we will next describe a MAS-based control society.

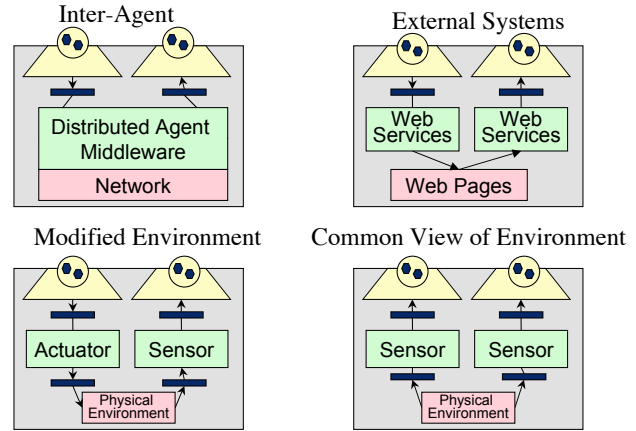


Figure 5: Diversity of Coordination Artifacts

5. Coordination Artifacts in a MAS-based Control Plane

Given that Coordination Artifacts can be designed and engineered for each specific interaction-scenario, and that artifacts have the potential for dynamic adaptation and responsiveness to environmental complexities [1], they exhibit the properties needed to construct control societies for a physical environment. We will call a society of agents that monitors and controls a physical environment a *MAS-based control plane*.

The physical environment can be a classic embedded control system, such as one used on a manufacturing floor. But more likely a MAS-based control plane will be used in a more dynamic environment, such as for inventory of a logistics system or for controlling pools of computer and network resources.

In scalable-agents systems today, there almost always exists some layer of control, external to the intelligent agents themselves. A *MAS-based control plane* is some set of control structures, either in middleware or external to the system, meant to interface between agents and the requirements/demands of their physical environment. Such a control plane must (i) adapt an economy-oriented view of the environment [9] taking into account the costs of bottlenecks in each architectural layer – possibly using sensors (i.e., serialization, bandwidth and propagation limits), (ii) provide enough architectural support that agents can exploit, (iii) use its structures to allow its system to scale effectively.

Figure 5 shows a typical hierarchical MAS-based control plane. The low-level sensor and actuators must coordinate with the physical environment and their manager agents. Management agents must integrate sensor data from many sensor agents and send control advice out to many actuator agents. All the coordination has soft real-time constraints and must be robust in the face of failures and overload. Also, the relative roles agents play is long-lived.

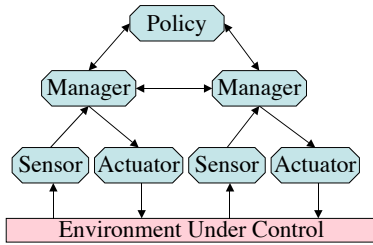


Figure 5: Hierarchical MAS-based Control Plane

6. QoS-Adaptive Coordination Artifacts

Extending Coordination Artifacts to make them *QoS-adaptive*, enables them to react to the constraints imposed by the physical environment. This may require specializing a generic coordination pattern into a customized component. This context-specific Coordination Artifact uses specialized knowledge about the expected coordination behavior, and run-time knowledge about the resource constraints to pick the best implementation strategy to meet the situation. To support this QoS-adaptation, we propose extending Coordination Artifacts to include:

- Addition of *facts and roles*; information concerning the values in a tuple-center and the domain-based role of an interaction with a Coordination Artifact, respectively.
- Addition of *systemic properties*, which inform the artifact about the constraints of the physical environment, potentially making it capable of dynamically changing the operating environment.
- *High-level services*, including data structure translation.
- A Coordination Artifact *life cycle*, which makes explicit the time horizon for QoS-Adaptations.

Addition of facts and roles: In encouraging the further specialization of the agent interface to Coordination Artifacts, we introduce two extensions. *Roles* are a typed portal into an artifact that assigns its operations to a specific agent function. *Facts* add types and attributes to tuples to define the QoS requirements for handling the tuple. In this design, the artifact's tuple center is further organized and divided, providing control over data flow between roles played by agents. Figure 6 illustrates the partitioning.

The advantage of partitioning an artifact's tuple space into *pools of facts* for each role is threefold. First, the complexity of instructions and operations are reduced to focus on a specific type of agent interaction with the Coordination Artifact. This interaction can have explicit QoS requirements associated with both the pool and the facts. Second, a role's pool of facts is local to the agent, so the direct interactions with the Coordination Artifact are not burdened with typical distributed system issues.

In some sense, the local pool of facts is a cache or proxy for the logical centralized tuple-space. Third, the Coordination Artifact can implement the coordination behavior by controlling the data flow between pools of facts. Since these data flows are distributed, and hence susceptible to physical constraints, they can have custom implementations using standard QoS-adaptive middleware [20,21].

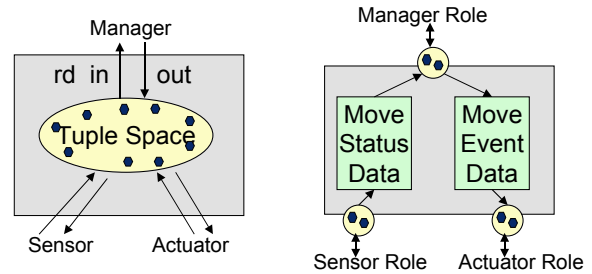


Figure 6: Tuple-Space vs. Fact-Role Implementations

Addition of systemic properties: Figure 5 illustrates the diversity of QoS-requirements between the roles in a Coordination Artifact. These diversities are apparent when dealing with agent systems involving communities of agents and chain-of-command style systems, i.e., managers and subordinates. For example, in Figure 6 a manager role may be aggregating sensor data from multiple sensor roles and disseminating control commands out to actuators roles. The sensor dataflow only worries about the current state of the sensors, and thus can discard old sensor observations if the path between the sensors and the manger gets congested. The command stream between the manger and the actuators must be reliable, and all control events must get through in their intended order.

High-level services including data structure translation [5]: This feature stems from the communication function of an artifact, which is to move data from one role to another. Both sides implicitly interpret the data in the same way. Using the improved partitioning structure, both agents could use the same non-standard representation and be able to communication data without double conversion. Also, this process could be used in a semantic translation of a data structure from one ontology to another ontology.

Coordination Artifact life cycle: Creating a life cycle for an artifact, which is standard upon creation time, allows ease of duplication/alteration of artifacts in MAS. In systems with a pre-existing middleware, adding such features to a Coordination Artifact allows ease of application, which you will see in section 7. The stages of this timeline are as follows: *Specification time:* a skeleton of the artifact is generated. *Implementation time:* the artifacts are further specialized to a specific type of coordination and application domain. *Creation time:* the Coordination Artifact is created at runtime and has a unique identifier and name. *Bind time:* an agent binds itself to an artifact's role. *Invocation:* the agent performs

operations on an artifact's role, within the constraints imposed by the role's instructions. *Unbind-time*: when the agent is done, it unbinds from an artifact's role. *Destroy time*: unnecessary Coordination Artifacts are removed from the system. This life cycle allows patterned creation and use of Coordination Artifacts. The life cycle is further explained in the next section, which shows an example implementation of QoS-adaptive coordination objects using pre-existing agent-based middleware.

7. QoS-Adaptive Coordination Artifact Implementation in an Existing MAS

We show how QoS-adaptive Coordination Artifacts can be created using an existing, open source, java-based agent architecture. The Cognitive Agent Architecture (*Cougaar*), has been used in the construction of large-scale distributed agent-based systems. [15,16] Cougaar provides a comprehensive infrastructure for developing robust societies of distributed agents, with security and scalability mechanisms built into this infrastructure. There exist within Cougaar, specific components in the middleware whereby the enhancements described to Coordination Artifacts can be modeled, made, and applied. These include a blackboard-based data representation, and message routers. [18] We explain each with references in the context of the related architecture component.

Cougaar agents communicate via a publish-and-subscribe mechanism. Each agent has at its core a membership-transactional blackboard with predicate-based publish/subscribe semantics. Logically, the blackboard is an arbitrary collection of objects. Communication plugins, called *Logic Providers*, subscribe to the blackboard and look for objects that should be transferred to other agent's blackboards. Logic Providers come in many different types, each with domain-specific behavior for moving objects between blackboards. For example, a simple *relay* Logic Provider might merely copy an object to all agents in a community. The *task/allocation* Logic Provider will request an allocation from a remote agent, but also predict the allocation before it officially arrives, allows the allocation to be rescinded, and recovers if either agent fails. Logic providers use a Cougaar service called Message Transport Service (MTS) to send messages between agents. The MTS takes care of all the communications details, and Logic Providers treat the MTS as a reliable message transport.

To interface to non-agent systems or to physical sensors, Cougaar supports an underlying Service Oriented Architecture. Cougaar is component-based middleware. Cougaar's interface is closely modeled on Java Beans BeanContext API. [17] Cougaar adds additional layers of insulating, or containing, objects in between each pair of interacting Components: Binders between parent and child Components, and ServiceProxies between server and client objects. These insulating objects can be anything from simple forwarding objects, merely delegating calls

to higher-level objects, or can implement arbitrarily complex adaptive behavior. Inter-component communication happens via *services*. The implementation of a service is provided by a *service provider*, which registers its services with a *service broker* at any level of the component hierarchy. A container controls which services are offered at its layer and can inherit services from its parent container or can override services for its children [18].

Figure 7 shows how plugins can be composed into a logical Coordination Artifact. The blackboard fulfills the function of the pool of facts for each role. Blackboard objects can be tagged with a role and a community. This effectively partitions the blackboard into pools associated with each logical Coordination Artifact. The plugins convert the underlying infrastructure components from an SOA model into a blackboard model. The infrastructure components interface to the non-agent system, the physical environment, or other agents. These components form a kind of connectionless Coordination Artifact, since they can be generated and removed dynamically, and they are not explicitly named.

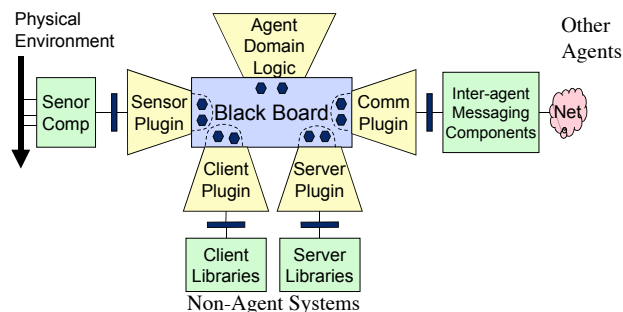


Figure 7: Cougaar Component Model

The QoS adaptation happens at each component layer. For example, the blackboard object can be tagged with a QoS requirement for a time to live. If a Logic Provider cannot send a message to the remote agent in time, the object is removed from the blackboard and the message is flushed from the message transport. Cougaar infrastructure has many components to help implement QoS-adaptation. [16] Some common examples follow. The Cougaar *service discovery* mechanism is a mechanism for finding an agent that offers a service. Through service discovery, other services can be contracted upon the request of any agent with access to the service. The distributed *Yellow Pages Service* in Cougaar helps to prevent bottlenecks and to balance workload. An agent's information can be contained in multiple YPServers for retrieval later. The *Metrics Service* provides information regarding resource consumption and performance measures. The Metrics Service especially provides necessary systematic

information to be used by the sensors in a QoS-adaptive Coordination Artifact, potentially initiating some level of control over its data

QoS-adaptive Coordination Artifacts increase scaling of communications in several dimensions. First, using Coordination Artifacts allows for groups of agents to communicate as a whole instead of pair-wise. Second, the QoS requirements are used to optimize the use of constrained resources, allowing larger societies to run in the same environment. Third, the MAS system will operate over a wider range of physical environments.

8. Conclusions

The possible applications of Coordination Artifacts are far-reaching in Multi-Agent Systems, while the objective nature of their coordination model makes them especially desirable to agent systems constrained by physical parameters in their environments. In such environments, Coordination Artifacts truly benefit from being resource-aware. By varying degrees of design in their construction, Coordination Artifacts can become equipped to offer agent systems more than a means of mere communication alone, but offer survivability features. Our hope is that the agent-development community will consider the extensions to Coordination Artifacts proposed in this paper, and their applications to agent coordination models and Multi-Agent Systems.

In the future, we plan to add explicit, long-lived Coordination Artifacts. Having a complete life cycle includes code generation of skeletons for glue plugins, blackboard objects, and Logic Providers. The coordination objects would be composed at runtime based on the underlying physical constraints. When the agent invokes the connection artifacts, the artifacts' components will dynamically adapt to QoS issues, as they currently do for logical Coordination Artifacts. First-class support for Coordination Artifacts will make programming coordination easier and will help societies scale to more dynamic physical environments.

9. Acknowledgements

The work described here was sponsored, in part, by the DARPA UltraLog contract #MDA972-01-C-0025.

10. References

[1] A. Ominici, A. Ricci, M. Viroli. "Coordination Artifacts: Environment-based Coordination for Intelligent Agents." *AAMAS'04*.

[2] D. D. Corkill. Collaborating software: "Blackboard and multiagent systems & the future." In *Proceedings of the International Lisp Conference*. 2003.

[3] E. H. Durfee. Scaling up agent coordination strategies. *IEEE Computer*, 34(7), July 2001.

[4] A. Ominici, A. Ricci, M. Viroli, G. Rimassa. "Integrating Objective & Subjective Coordination in Multi-Agent Systems" *AAMAS'04*.

[5] Jintae Lee, Thomas W. Malone, "Partially Shared Views: A Scheme for Communicating among Groups that use different Type Hierarchies" *ACM Transactions on Information Systems*, January 1990, pp 1-26.

[6] A. Ominici and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277-294, Nov. 2001.

[7] FIPA Agent Control Language Specification <http://www.fipa.org/repository/aclspecs.html>

[8] TuCSon <http://lia.deis.unibo.it/research/TuCSon/>

[9] A. Ominici, S. Ossowski, A. Ricci, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Chapter 14. Kluwer Academic Publishers, June 2004. "Coordination Infrastructures in the Engineering of Multiagent Systems."

[10] A. Ominici. "Towards a Notion of Agent Coordination Context." *Process Coordination and Ubiquitous Computing*, Chapter 12, CRC Press, October 2002.

[11] A. Ominici, F. Zambonelli. "TuCSon: a Coordination Model for Mobile Information Agents." *1st International Workshop on Innovative Internet Information Systems (IIS'98)*, Pisa, Italy, June 1998.

[13] M. Schumacher. "Objective Coordination in Multi-Agent System Engineering - Design and Implementation," Volume 2039 of *LNAI*. Springer-Verlag, Apr. 2001.

[14] Wooldridge, M. J., and Jennings, N. R. (1995). "Intelligent Agents: Theory and Practice." *Knowledge Engineering Review*, 10(2), pages 115-152.

[15] Cognitive Agent Architecture. <http://www.cougaar.org/>

[16] "Cougaar Developer's Guide" v. 11.4. http://cougaar.org/docman/view.php/17/133/CDG_11_4Final.pdf

[17] <http://java.sun.com/products/javabeans/index.jsp>

[18] John Zinky, Richard Shapiro, Sarah Siracuse. "Complementary Methods for QoS Adaptation in Component-based Multi-Agent Systems." *The 2004 IEEE First Symposium on Multi-Agent Security and Survivability*.

[19] Nicholas Carriero, David Gelernter "Linda in Context" *Communication of the ACM*, Apr 1989.

[20] Quality Objects (QuO) <http://quo.bbn.com>

[21] George Heineman, Joseph Loyall, and Richard Schantz, "Component Technology and QoS Management," *International Symposium on Component-based Software Engineering (CBSE7)*, May 24-25, 2004.