Towards an Architecture for Extreme P2P Applications

Nadia Shalaby  John Zinky
BBN Technologies
10 Moulton St., Cambridge, MA 02138
nadia@bbn.com  zinky@bbn.com

ABSTRACT
The scope of the peer-to-peer (P2P) paradigm has expanded beyond the research arena and has become ubiquitous in commercial, industrial and military applications. This ubiquity, however, comes at the cost of significant handicap in design and development of large-scale, reliable, complex realtime applications, as they do not fit into readily available optimized P2P solutions, such as file distribution, grid computing, or Pub/Sub message-passing networks. Rather, these applications necessitate custom development, a high-risk, time consuming and expensive process. We approach this gap by categorizing the problem space into an application taxonomy, and identifying a new class, which we call Extreme P2P applications. Extreme applications are characterized by cross-cutting dimensions of severe QoS requirements, variable resource constraints, evolution during deployment, inherent human participation during operation, and small market share – to name a few. Such characteristics contribute to their development being a significant challenge. We address this challenge by first, proposing a novel architecture for Extreme Applications, and second, by introducing a newly re-architectured, comprehensive middleware, Cougaar, as a suitable platform for its implementation. We demonstrate the suitability via an architectural mapping, and show how the novel two-tier Cougaar architecture addresses the contextual domain of many such applications by being backward compatible with existing distributed and P2P systems.

KEY WORDS
Peer-to-Peer, Applications, Distributed Agents, Real-Time Systems, Middleware, Development Cycle

1. Introduction
Several optimized P2P solutions are available for specific application domains, such as file distribution, grid computing, or publication/subscription (pub/sub) message-passing networks. However, many modern distributed applications in the commercial, industrial, military and research domains inherently need multiple types of coordinations, have extreme system constraints, stringent QoS requirements, evolve in the course of their operation, or intrinsically require human participation in their operation. In these cases, the application's implementation will need custom development because the range of requirements is not met by any one optimized framework. Custom development of large-scale distributed P2P systems has proved to be a high-risk, time consuming and expensive process. This identifies the need for a middleware platform to create customized P2P applications quickly and cheaply, in a modular and incremental fashion, much like CORBA did for client/server applications, Globus [1] did for Grid applications, or J2EE did for web services.

Such a middleware platform would have complex requirements, desired to be sufficiently versatile, functional, and efficient in its development and operation, so as to provide the necessary framework for a wide range of sophisticated applications. To this day, the nature of these applications and their inter-relationships is not well understood. This work is a step in this direction. To this end, we formulate a taxonomy of distributed and P2P applications, defining classes with progressively more extreme constraints. This allows us to study and design their architectures and analyze inter-relationships at different levels of the system hierarchy.

Our analysis from the resource and application perspectives leads us to choose multi-agent systems as a basis for a unified middleware platform. A coarse architectural mapping between applications with extreme constraints and the multi-agent paradigm further emphasizes the suitability of this choice.

We advocate a new, two-tiered Cougaar [2], as a suitable candidate for a unified multi-agent middleware platform, which provides two major advantages. First, its architecture is a natural match to address the needs of the Extreme subclasses. Second, we demonstrate how its intrinsic architectural modularity allows backward compatibility with existing distributed and P2P applications or their integration into new applications. Cougaar's candidacy is superior to other multi-agent systems for several reasons: it is mature, open source, modular, customizable, embeddable, and has a wide range of support for the various development phases, from domain programming to runtime services. Moreover, it has exhibited scalable, survivable and secure performance for a range of Extreme applications.

The remainder of this paper is organized as follows. We first situate the work in Section 2, where we analyze the structure of P2P applications and henceforth define a taxonomy of application classes. We propose a novel architecture for Extreme applications in Section 3. This is followed by an overview of a new two-tier Cougaar architecture in Section 4, and a demonstration of how the enhanced Cougaar is a natural fit for Extreme applications.
from the architectural standpoint in Section 5. In Section 6 we discuss Cougaar’s backward compatibility with legacy applications. Case studies of Extreme and S3 applications are mentioned in Section 7. Section 8 concludes the paper.

2. Taxonomy of P2P Applications
It is always simpler to tackle a narrow problem than provide a more general solution. This is why for the most part, support for P2P applications has targeted a specific type of peer coordination. We believe that the space of P2P applications has expanded and matured to the point where the lack of a unified middleware platform is taking a toll on cost, performance and, in some cases, deploying the technology in a timely fashion altogether. Therefore, a more ambitious solution is in order. To provide a unifying middleware framework appropriate to the scale and heterogeneity of the space of P2P applications, we need to carefully define the scope and characteristics of the problem.

A systems view of the P2P applications architecture is depicted in Figure 1, portraying system components with richer functionality going up the layering hierarchy. The operating system (OS), network stack, and file system are classic systems abstractions built on the cyber resources of the CPU, network and persistent storage, respectively. At the next layer, service abstractions translate the resource abstraction API to a richer client/server interface. Grid services, such as Globus [1], are layered on OS processes, and in turn export job queues to their clients. Message services, such as web servers, translate the socket API to a higher level protocol such as RPC or HTTP. Likewise, database services, such as Oracle, use the file abstraction and export database languages such as SQL. Note that all three types of service abstractions also cross over to make use of the other two resources, albeit in a more lightweight fashion -- those interactions are depicted by thin arrows.

With the advent of P2P applications, and to facilitate and attract usage, another overlay was built on these services, namely P2P substrates. Chord [3], Pastry [4] and PlanetLab [5] are such examples in the research arena. Akamai and Napster are two examples of commercial P2P overlays. The goal of these overlay substrates is to offer richer language support and interoperability with underlying heterogeneous systems and services. Finally, P2P applications, such as PAST [6], Scribe [4], Kazaa [8], and Logistics Decision Support System (LDSS) [9] are all built on their respective overlay substrates. Note that applications are built on only one such overlay, whereas each overlay can host any number of different applications, in contrast to the lower levels of the system hierarchy. Having identified the structure of P2P applications, we now need to study the space of these applications with respect to their domain and resource requirements.

In recent years, the commercial, military and research domains have seen the emergence of a class of real-time distributed P2P applications with scalability, survivability (reliability and performance) and security requirements. We will refer to these as the S3 applications, where applications fall into this class only if mandating all three requirements. Examples are disruption tolerant networks, content distribution, fund exchange between banks, information assurance applications for resource protection, and travel reservation systems that need to arbitrate with individual airline systems in a P2P fashion. Within this class of S3 applications, we identify a particular subclass with more stringent requirements, with respect to both, the application functional requirements, and the system resource constraints. We refer to this subclass as Extreme Applications, characterized by the italicized parameters of all of the following twelve dimensions, as depicted in Figure 2:

Application Functional Requirements (addressed at the programming/development level):
1. Communication paradigm: P2P vs. client server
2. Development cycle: the application is in (partial) operation while still at its build and deployment phase, vs. the classic waterfall development cycle of implementation, installation, configuration and tuning -- in essence, all four phases are now required to happen quasi-simultaneously.
3. Process and requirements during operational lifespan: evolving vs. fixed
4. Human participation level: inherent human participation to perform discovery, execute specific processes and specify the automation requirements, on the sensor, model, cognitive levels vs. none

Figure 2: Scope of Extreme Applications: Extreme in All Sets

<table>
<thead>
<tr>
<th>Business</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 1:** Underlying System Structure of P2P Applications
5. Cross-cyber resource load: the application requires CPU, network and storage capabilities from heterogeneous resources vs. predominantly CPU, or network or storage

System Resource Constraints (exhibited at runtime):
6. Distributedness of cyber resources: distributed across wide area (WAN) vs. local area (LAN) vs. centralized
7. Data plane speed: realtime vs. online vs. batch
8. Survivability in reliability and performance: exigent vs. non-critical
9. Security trust level: insider threat or malicious intruders vs. compartmentalized threat vs. trust all
10. Scalability: more than 10,000 core nodes vs. 1000s, 100s or 10s.

Business Environment (organizational constraints):
11. Market Share: small vs. medium vs. large
12. Integration Environment: new functionality with legacy system integration vs. standalone vs. stovepipe

Examples of Extreme applications are field modifiable systems required to prevent evolving malicious attacks, UAV surveillance on mobile sensor platforms, systems for disaster relief operations, proactive and location-aware content distribution systems and continual ISP and telephone network providers service management and optimization.

We have thus formulated a taxonomy of distributed applications with progressively extreme requirements and constraints, depicted in Figure 3. As we zoom in on this taxonomy of application classes, the demands on a supporting middleware framework are progressively escalated. The middleware should be rich enough to enable quick and cheap development of the most sophisticated Extreme applications, as well as provide adequate support for the less sophisticated S3, P2P and distributed applications, thereby offering a unified middleware platform for the entire class.

Two comments are in order. First, many other partially overlapping subclasses can be defined based on the twelve dimensions identified above. We are however primarily interested in the Extreme applications that satisfy all twelve. Second, applications that address a single over-constrained resource make up three important subclasses, and are historically interesting as motivators for P2P development. We refer to them as Message-based, Grid-based and DB-based when they are Network-bound, CPU-bound and disk-bound, respectively. Such uni-resource based applications span the distributed, P2P and S3 hierarchy, as shown in Figure 3.

3. Architecture for Extreme Applications

Extreme applications systems simultaneously perform a distributed physical task, while adapting the scope, configuration and implementation of that task in realtime. These systems tend to naturally partition into two planes. A data plane that actually does the task, and a management plane that assesses the external situation and controls how to perform the task. For example, in standard networking infrastructure, the routers and links perform the task of moving data packets, while the network management system monitors the health and status, changes configuration, generates security policies, and optimizes resource consumption. The data plane is designed to work in the short-term without advice from the management plane, but cannot adapt to long-range issues and will eventually fail. The extreme version of a network infrastructure is when higher level “mission” requirements are changing quickly and the underlying data plane must be reconfigured to meet these mission requirements in realtime. Thus, Extreme applications place only dynamic reconfigurability requirements on the data plane, while deferring most of the hard adaptation functionality to the management plane. Consequently, an architecture for extreme applications must be based on the following principles: (1) a natural separation of data plane and management plane; (2) hierarchical levels of management control; (3) decoupling of systemic constraints from the application requirements; (4) a flexible and distributed computational model, with first class coordinations between the computational entities.

We propose a novel architecture for Extreme applications, which adheres to these principles, and is portrayed in Figure 4. A traditional data plane spans the systems hierarchy levels from the cyber resources at the bottom, interfacing to a stack-like executor of the protocols and services at various levels, all the way to the distributed applications at the top. The interesting component of the data plane is the presence of distributed sensor agents, such as Error! Reference source not found.Error! Reference source not found. gathering resource status (systemic constraints), as well as application and QoS status (dynamic application requirements), at the lower and higher levels of the systems hierarchy, respectively. These sensor agents are instrumental in driving the control plane.

The central architectural feature of the control plane is its layered nature, spanning three control loops with increasing levels of intelligent functionality. The first, a rapid but simple sensor-based control loop, collects the application status and resource status from the sensor agents, and produces an optimized policy for the data plane in real time (milliseconds to seconds). The second is a slower but more robust model-based control loop, expecting the sensor agents to collect resource and application trends, and using statistical pattern matching or rule-based techniques, predict an expected situation to modify the real-time optimizer agents. This loop operates within the seconds to minutes range, thereby being more dispensable in situations of extreme system duress. It
does, however, need to react to unforeseen conditions in a timely manner. The third, slowest and most complex **cognitive** control loop, operates within minutes to days, expects to collect detected **resource and application patterns** from the sensor agents, and via an adaptive and cognitive learning process drive the model-based loop by producing either a Hidden Markov Model (in the statistical pattern matching approach) or inference rules, thereby adjusting the situation predictor agents. The architecture assumes that if the model-based loop cannot react in minutes, the cognitive loop contributes its knowledge.

Note that the sensor agents, as well as the control loop agents of all three levels are distributed across the system, which in turn requires the interface between the latter agents and the sensor agents to be conducted in a distributed fashion. This distributed interaction is a complex problem which we abstract into the notion of **coordination**. A suitable implementation needs to offer such coordination as a first class construct.

One of the goals of Extreme applications is to replace human involvement in the control loops with automation in an incremental and gradual manner. This necessitates support for adapting to the application’s evolution and learning the new evolved processes and requirements automatically. Our architecture is purposefully designed in the form of progressively more complex and intelligent levels of execution on the control plane so as to allow for human agent operation. As the application matures in its development and operational cycle, such human agents can gradually migrate to automated agents in an incremental fashion.

4. **Cougaar’s Two-Tier Architecture**

We re-architected Cougaar [1], an open source, agent-based middleware tool kit for developing distributed P2P applications, into a new two-tier architectural model so as to enhance P2P application development in a more modular, incremental and low-cost fashion. The key new architectural feature, depicted in Figure 5, is partitioning Cougaar into two layers: a **society of Agents** that encapsulate the application behavior, and an **Environment** that interfaces to the underlying resources and manages the coordination between agents. This separation decouples application development from the complex issues of how to distribute the application in the field. In contrast to the traditional approach to layering, which attempts to completely hide the system issues from the application, the agent/environment boundary is elastic, allowing adaptive behavior to be handled at the most appropriate place for a given situation. Such flexible partitioning is especially important for Extreme applications evolving in terms of both, application QoS requirements, and environmental resource constraints.

---

**Figure 4: Architecture of Extreme Applications**

**Figure 5: Two-Tier Cougaar Architecture**

Agents are **data-driven** entities, with a Pub/Sub API, sensing data from the Environment, processing it, storing it, and writing it out to the Environment. Physically, each Agent is located on a single host and its **state** is defined by its Blackboard. On the other hand, the Environment is **event-driven**, exporting a service-oriented API, reacting to Agents and external systems, with components that interact through services. The Cougaar runtime Environment is the place where Agents execute and are offered a view of “The World”. Physically, the Environment interfaces to the underlying resource infrastructure, distributed across multiple hosts, as well as the network enabling inter-host communication. An agent-based application is a society of Agents that runs within an Environment. The Environment interfaces to the external world and coordinates between Agents, while the Agents implement the application functionality.
An Agent can interact with another Agent via the Environment; essentially one Agent actsuates a change to the environment that another Agent can sense. Agents interact via coordinations --- specialized communication channels between a group of Agents, analogous to long-lived sessions with built-in behavior. Each Agent plays a role relative to coordination, and the coordination moves data between roles, based on the coordination rules. An Agent society is a group of agents interconnected by coordinations and united by a common purpose. A detailed description of the new Cougaar architecture is found in [1], where we enumerate many tools and techniques for incrementally developing, encoding, and evolving the adaptive behavior at the Agent and Environment layers alike.

5. Extreme Applications to Cougaar: Architectural Mapping
To demonstrate how the two-tier Cougaar platform is a natural candidate for implementing the architecture of Extreme applications, the reader is referred back to Figure 4. We consider four aspects of the architectural mapping.

(1) Separation of data and control planes: The depicted data plane could be easily imported from its own native infrastructure. Cougaar would provide the middleware platform for the progressively more intelligent control plane depicted in the architecture.

(2) Computational model: The oval shaped functional modules of the control plane are mapped to Cougaar agents. Each set of agents have significantly different processing Behaviors for each of the sensor agents, real-time optimizer agents, situation predictor agents, and cognitive learner agents. Inter-agent coordination is mapped to Coordinations. Likewise, these Coordinations at various levels of the control plane hierarchy, the Sensor-based loop, the Model-based loop and the Cognitive loop, as well as among Agent peers, have drastically different QoS requirements. Such a wide range of complexity necessitates the Cougaar-provided modular support for Coordinations as first class constructs.

(3) Decoupling of system and application: The application’s control plane resides on a Cougaar Environment which is inherently distributed (since the data plane is distributed). The control plane is primarily responsible for delivering the application-imposed QoS requirements within the Environment-encapsulated cyber resource constraints. This delivery adaptation for Coordinations is provided by the Cougaar Environment. Said another way, the Environment takes care of the systemic issues for implementing the Coordinations, such as maintaining seamless Coordinations in the face of host, network or storage disruption. For example, in Figure 4, the Environment provides services that monitor and control the management interfaces of the data plane, such as data flow statistics. Behaviors of optimizer agents on the control plane would then set policies to restrict the flow. On the control plane, the Environment services are purely there to provide coordination between agents. To illustrate, two different Coordinations would be used in the sensor based control loop, one to move the monitored flow statistics from the sensor agents to the optimizers, and another to communicate the policies from the optimizer back to the executor on the data plane.

(4) Hierarchical control plane: This hierarchy inherent in the Extreme application architecture is mapped onto Cougaar agent societies, where distinct agent societies performs the functions of each of the sensor, model and cognitive loops.

(5) Evolving degree of operational human involvement: The human factor, inherent in Extreme applications, and its degree, is simply mapped by identifying which of the control plane loops (sensor-based, model-based or cognitive) are implemented by human agents, and which by automated Cougaar agent societies. This hierarchical progression within the proposed architecture and the corresponding hierarchical modularity in Cougaar societies allows for a gradual transition towards automation, in an incremental fashion. In other words, as the operational and functional specifications become better understood at a particular control loop, the development cycle evolves and is shifted up to address the next level in the control loop hierarchy. This concept is further explored in more detail in [13].

6. Integration with Legacy Systems
In order for our middleware to serve as a unified framework for Extreme and other customized P2P applications, it must not only provide a development solution that is quick, cheap, modular and incremental, but also be compatible with existing legacy systems or their components. In addition to speed and cost advantages associated with easy incorporation of existing components, in practice, it is often a necessary requirement to integrate with existing highly tuned systems, built on a variety of different platforms, as was demonstrated in Figure 1. The two-tier Cougaar architecture strategically provides this capability by allowing for two types of components, as illustrated in Figure 5: native Cougaar components providing the entire service functionality within Cougaar, or wrapper components that map any imported service library to the Cougaar service API. In essence, Cougaar could be used during the integration process to “fill in” the missing functionality of differently designed existing systems, in a modular and incremental fashion, wrapping legacy services with a unified API.

Service libraries from the underlying runtime environment can be incorporated at various levels of the system hierarchy, aligned with the agent/environment boundary of the implementation. An example of a legacy application integrated by means of Cougaar is the Integrated Consumable Item Support (ICIS) Model [10], which was originally developed as a traditional client-server architecture.

Two functional pieces needed to be migrated into UltraLog [11]. The first, a depot availability, comprised a complex C-language program with a well-defined API,
was imported as a service library and encapsulated into a wrapper component performing all the necessary data translations between the library and Cougaar’s Environment framework. The second, a demand forecasting unit, was designed in a centralized fashion, incompatible with UltraLog’s high-fidelity, scalable logistics planning. Therefore, new components were created leveraging the legacy service algorithms into Cougaar’s distributed Environment.

7. Case Studies of Extreme Applications
Cougaar has been successfully used as a middleware platform for building a plethora of challenging Extreme applications, and has exhibited excellent performance [12], scalability and survivability results. In one occasion, the Ultralog application [11] employed 1100+ agents distributed across 100+ hosts, subjected to a barrage of stresses and attacks over a range of cyber resource failures totaling 45% infrastructure loss, and the application survived with more than 80% of its capability while still exhibiting more than 70% of its maximum performance. Ultralog also placed stringent requirements on its development cycle, and the engineering test team was able to conduct over 190 test runs over the course of one month during its operation.

The LDSS Supply and Distribution (S&D) system [9] is a work-in-progress, multi-user system distributed supply chain planning and execution tool that manages inventories, calculates projected consumption and creates a resupply plan for Future Combat System (FCS) Brigades. It is built on 104 Cougaar agents distributed across multiple nodes, in support of distributed multi-echelon command and control of 1250+ vehicles, containing 30 commodities with corresponding inventory for each. With visualization of projected unit-readiness and projected-versus-actual inventories at any level of detail, from aggregations over all Brigades down to individual platforms, the system currently executes the plan in 50 seconds and has been robustly used in half a dozen of military exercises.

We are in the process of researching developing Cougaar for management and operational optimization of large scale networks and processing centers. Our design employs a Cougaar society to create a hierarchical management plane to schedule compute jobs between specialized centers. The management plane in turn, creates scheduling and routing policies for and between the processing centers. Another Cougaar society would coordinate status and control data between centers, calculate local schedules, and optimize global routing policies. Initially, we are experimenting with the Cougaar management plane as a simulation running on a 30-node grid computer. As the algorithms mature, some of the simulated agents are being replaced by interfaces to real systems, to synchronize the simulation to real data. For example, transitioning actuator agents would enable the society to control the real data-plane. The target scale of the data plane is 100’s of centers, each with 100’s of specialized processing elements, scaling up to millions of elements in total.

7. Conclusion
The contribution of this paper is threefold. First, we define a taxonomy of P2P applications and identify application subclasses with progressively stringent constraints, namely S3 and Extreme classes of applications. Second, we propose a novel architecture for Extreme applications that leverages the extreme development process and the gradual phasing out of human involvement during the system’s operation. Finally, we present a new, enhanced two-tier Cougaar architecture and demonstrate how it serves as a middleware platform that is both, a natural fit for building Extreme and S3 applications, as well as backward compatible to support legacy components built on tailored subsystems.

In the future we would like to explore the architectural mapping of Extreme applications to Cougaar within an analytical model at various levels of the system granularity. The generated insight could then be used to prototype and build such novel Extreme applications that conform to the stringent requirements in a cost effective and efficient manner.

References
[6] A. Rowstron, P. Druschel, Storage Management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, 18th ACM SOSP, Banff, Canada, 2001, 188-201