Topic: Information Age Transformation

Title: Suitability of Agent Technology for Military Command and Control in the Future Combat System Environment

Thomas Potok[1], Laurence Phillips[2], Robert Pollock[2], and Andy Loebl[1]

[1]Oak Ridge National Laboratory
Post Office Box 2008, Mail Stop 6414
Computational Sciences and Engineering Division
Oak Ridge, Tennessee 37831-6414
Phone: 865-574-0834
Fax: 865-241-6211
potokte@ornl.gov

[2]Sandia National Laboratories, New Mexico
PO Box 5800, Mail Stop 0455
Advanced Information and Control Systems
Albuquerque, New Mexico 87185-0455
Phone: 505-845-8846
Fax: 505-844-9641
lrphill@sandia.gov

Point of contact: Thomas Potok

Suitability of Agent Technology for Military Command and Control in the Future Combat System Environment

Thomas Potok[1], Laurence Phillips[2], Robert Pollock[2], and Andy Loebl[1]

[1]Oak Ridge National Laboratory
Post Office Box 2008, Mail Stop 6414
Computational Sciences and Engineering
Division
Oak Ridge, Tennessee 37831-6414
Phone: 865-574-0834
Fax: 865-241-6211
potokte@ornl.gov

[2]Sandia National Laboratories, New Mexico
PO Box 5800, Mail Stop 0455
Advanced Information and Control Systems
Albuquerque, New Mexico 87185-0455
Phone: 505-845-8846
Fax: 505-844-9641
lrphill@sandia.gov

Abstract

The U.S. Army is faced with the challenge of dramatically improving its war fighting capability through advanced technologies. The focus of this paper is to assess the novelty and maturity of agent technology for use in the Future Combat System (FCS). FCS will rely on networked command and control (C^2) to transform from the historically centralized C^2 function. Achieving a networked C^2 capability will require breakthroughs in current software technology.

We have developed a set of software requirements for FCS based on military requirements for this system. We have then evaluated these software requirements against current computer science technology. From this analysis we find that existing technologies will not likely be sufficient to meet the networked C^2 requirements of FCS due to limitations in scalability, mobility, and security.

However, agent technology provides a number of advantages in these areas, mainly through much stronger messaging and coordination models. These advantages have the potential for significant improvements in scalability, mobility, and security. We believe that agent technology has the capability to support most of the networked C^2 requirements of FCS. However, we would recommend proof of principle experiments to verify the theoretical advantages of this technology in an FCS environment.

1 Introduction

The U.S. Army's new concept for the future combat system (FCS) describes forces that must be "flexible, effective and efficient multi-mission forces capable of projecting overwhelming military power worldwide" [1] across the full spectrum of engagement. This "system of systems" will include networked command and control (C^2) capabilities designed for future missions, which is a significant departure from the historically centralized C^2 system. Before this new networked C^2 capability can be achieved, several major technical challenges must be overcome. The goal of this paper is to highlight the significant new software requirements of such a system and to determine whether software agent technology is a suitable means of addressing these technical challenges.

The FCS C^2 system is a revolutionary approach to provide network-centric C^2 with dedicated battlespace visibility and support for a completely integrated intelligence, surveillance, and reconnaissance (ISR) capability. The system is to be built within an Objective Force consisting

of a family of autonomous and non-autonomous vehicles expected to assure command of a battlespace tens of kilometers wide, in three-dimensional space, vertically integrated, and effectively interoperable among allied and joint forces.

The complexity of the future war fighting environment will require that information be securely and reliably transmitted over dynamic and potentially unreliable virtual and physical networks. Data from a wide range of systems and sensors need to be fused, analyzed, and summarized to help support rapid and effective decision-making.

Creating software to manage this modern C^2 functionality provides a number of significant computer science challenges. For such a complex system to be developed within any reasonable time frame, improvements in software development productivity and quality are needed. Indeed, it is unclear whether the technology to create such a system is available today. However, many have suggested that agent technology and its emerging software development conventions and environment may provide the strongest capability for solving such a substantive development problem [2].

The goal of this paper is to address those technologies that seem suitable for building this C^2 environment for FCS, particularly agent technology. We begin (Section 3) with a background review of the networked C^2 challenge in an FCS environment, in the process also developing a set of software requirements for such a system. We then analyze the networked C^2 requirements against the current state-of-the-art non-agent-based software technology to develop a list of limitations in the current technology (Section 4). In Section 5, we review these limitations against agent technology and explore the potential of this technology. Section 6 describes briefly several current agent-based systems of particular relevance given FCS requirements. The final sections provide recommendations and conclusions on the suitability of agent technology in creating the environment for the envisioned C^2 of the Army's FCS.

2 Background

2.1 Command and Control (C^2) Evolution

According to U.S. Army leadership, the main enhancement of the FCS C^2 system is that it will be network-centric at its core [3]. Historically, C^2 has been centralized—i.e., intelligence has been sent to a central location where military decisions are generated and from which C^2 emanates. Typically, decision makers have relied on centralized C^2 structures and adequate time to make and transmit decisions. As the operational tempo of war increases to allow modern forces to succeed, the older concepts of C^2 become a liability to forces in the battlespace.

The concept of decentralized control and centralized command is not new, having been used by the Greeks, Trojans, and Romans, as well as in recent warfare. However, the revolutionary concept of networked command is so recent as to seem visionary and can now be considered only because of advances in information technology. This paper addresses how and when such technology can be applied given its limitations. In order to decentralize command, intelligence gathering and analysis must be available at lower levels in the military hierarchy [4]. Figure 1 depicts a notional information network of the sort required to support a decentralized C^2 environment.
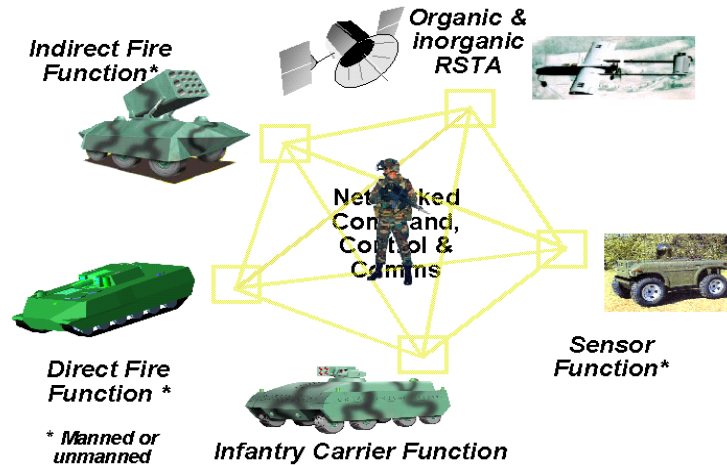
Figure 1 The FCS concept of networked command and control.

The FCS concept implies that data will be produced by a very large number of sources—every human and most machines involved in an FCS operation—and shared among a very large number of entities, vertically integrated, and so broadly federated as to define interoperability in a new venue.

2.2 C^2 Requirements

Although FCS requirements have not been fully defined at the time of this writing, information from a U.S. Army Training and Doctrine Command (TRADOC) briefing, reproduced as items 1–7 below, describes the functional requirements of the FCS C^2 system [5]. We use these requirements to develop a list of software capabilities that are required to support the FCS C^2 system. We then use these software capabilities as a basis for evaluation and comparison. Each numbered item from the TRADOC briefing, shown in italics, is followed by an analysis of the capabilities and behaviors the numbered item would demand of the software supporting it.

1. *Collect, display and disseminate a seamless, fully integrated, multidimensional, and tailorable common operating picture; and precision geospatial environment information layers (modifiable digital overlays) which support cognitive and dynamic mission planning/rehearsal, thus creating a real-time virtual decision making capability based on the commander's and battle staff's detailed "knowledge" of the friendly, enemy and physical environment.*

   To meet the first functional requirement, the software system must maintain a real-time, easy-to-understand, and accurate Common Operating Picture (COP). This implies that the volume of information distributed throughout the battlefield sensors and systems network must be rapidly and accurately integrated, then analyzed and organized to support military decisions. For a COP to be common, it must either be 1) produced in one place and distributed, or 2) produced wherever needed using distributed information. The first approach calls for centralized command, and becomes an obvious bottleneck, where delays or failure limit or prevent access to an up-to-date COP. The second approach has no such bottleneck. In

such a system, the FCS software system would act to provide the information needed to construct the COP over the C^2 network. There would be no central creation point whose destruction would prevent the COP from being formed, and the FCS system would degrade gracefully under component destruction or failure since no component or group of components is responsible for the COP. All FCS components would act to provide COP information to the network where any site with COP formation capability can produce its own COP.

2. *Enable battle command on the move supported by C4ISR architecture for continuous estimate of the situation on the move. Share integrated common operating picture to enable visualization and dissemination of tactical scheme by combined arms mission orders with graphic overlays. Changes in leadership that occur during battle will be automatically disseminated to appropriate levels with shared COP to enable continuity of command.*

This second functional requirement expands on the first by adding the capability of mobile command, decision making, and ISR. To meet this functional requirement, the system software must have the ability to move command securely from one future combat vehicle and/or commander to another. This type of command requires that FCS system software support the ability to deliver orders when one or more of the participants are moving. This function would also have to be tightly integrated with the physical C^2 network.

3. *Objective force units must contain a mission-centric, embedded information system that enables commanders to effectively lead during dynamically changing and offensive operations anywhere on the battlefield. This includes the following tasks.*

   a. *They must maintain situational understanding at all times. This is greater that just providing fused sensor data to provide the red and blue COP. It includes that capability to collaborate with subject matter experts, subordinate commanders and staff in real time in order to develop a complete appreciation of the situation.*

   b. *They must identify schemes of maneuver, opportunities, decisive points, terrain and weather updates, enemy vulnerabilities, and conceptualize solutions through accelerated collaborative planning, rehearsal and simulation.*

   c. *They must make reasoned decisions based on information available. The commander will be able to leverage intelligent agents in his information systems to assist him in filtering through the vast amount of information so that he only focuses on the most pertinent items to assist in his decision making process.*

   d. *Commanders will direct decisive action through communicating orders, intent and supporting operational graphics from the commander's battle command system.*

   e. *Commanders will synchronize maneuver, fires and RSTA [reconnaissance, surveillance, targeting and acquisition]*

Requirement 3 adds the concept of mission-centric situational understanding in a dynamic environment where the participants in command operations are not only mobile but also in different locations. To meet this requirement, the C^2 software and supporting ISR resources must be able to rapidly and accurately acquire and fuse mission-relevant data, then assist in analyzing and summarizing the data, and finally help to support command decisions.

4. *Commanders and battle staffs will leverage automated cognitive decision aids and real-time collaborative planning support tools to achieve knowledge-based course(s) of action*

development. Systems must be mobile, fully interoperable in the joint multinational, and interagency environment.

Requirement 4 poses a significant technical challenge in the area of decision support and security. We believe that commanders and their forces will use the most effective technology available to help plan and make decisions. However, many significant issues must be overcome in the area of decision support and collaborative planning [6]. In addition to this is the security challenge of sharing information at various levels of classification with various other joint and allied and even coalition forces, ensuring that it does not get corrupted by, or fall into the hands of, an enemy.

5. *[The mission-centric, embedded information system] will provide [a] digital 3D mapping tool for high terrain resolution to enable C2 of small unit tactical action in close, complex terrain; virtual rehearsals; and terrain analysis. Also allows visualization of inside buildings and subterranean dimension.*

Requirement 5 adds three-dimensional (3D) and geospatial visualization to the FCS C^2 system concept. These features will require the software to perform very complex data analysis, summarization, and transformation so that it can be viewed in a comprehensive and understandable way. Creating two-dimensional (2D) images of large amounts of data is a difficult problem; 3D portrayal dictates significant additional complexity.

6. *[The mission-centric, embedded information system] will enable continuous mission planning from alert through deployment to employment. Support continuous mission planning, rehearsal, battle command, and ability to integrate into gaining theater command during movement by air, land, and sea.*

This sixth functional requirement is closely aligned with the second requirement, command on the move, and the fourth requirement, real-time collaborative planning support and course-of-action development. This item adds no new software requirements to the FCS C^2 system. It emphasizes that the other requirements must be met continuously, regardless of transport mode, beginning at first alert and ending some time after force stand-down and postmortem mission analysis.

7. *Enable command and control needed to synchronize fire, maneuver, and RSTA in real time to close with and destroy the enemy.*

In an environment where command and control are decentralized, it becomes necessary to coordinate and synchronize activities. This requirement's use of the word "synchronize" implies temporal requirements and constraints for all C^2 functions. We assume that it must be possible to include these concerns during planning and course-of-action development, although this is not explicitly stated.

2.3 C^2 Requirements Analysis

To satisfy the requirements as analyzed above, the networked FCS C^2 concept will need to be based on significant software technology advances in scalability, mobility, and security. The emerging FCS concept of C^2 activities will no longer be performed in a centralized manner, but over a dynamic network of moving vehicles, and will be dependent on a vast array of sensors to gather data from the battlefield. This new C^2 network will be created in an ad hoc fashion, with nodes entering and leaving the network at unpredictable times. The C^2 system must be highly

reliable and highly secure. The battlefield sensor information, vital to C^2, will be broadcast from potentially thousands of locations. This proposed FCS C^2 network must be able to process this information rapidly and deliver the right information to the right locations and people at the right time.

As developed above, this system provides a number of new software challenges that we have summarized in the following list:

1. *Distributed computing* over an unreliable, ad hoc, dynamic physical network

2. *Fault tolerance* over a system in which, at any given time, it is unclear what nodes are available within the network

3. *Network security and accessibility.* Warfighters will need immediate access to the network, but adversaries need to be prevented from accessing or corrupting it.

4. *Data fusion.* Data from a wide range of systems and sensors will need to be correctly related

5. *Information analysis and summary* of enormous amounts of data from the C^2 network on the basis of user needs

6. *Decision support.* A network capable of supporting C^2 decision making

7. *Software development improvements* to reduce the complexity and risk in creating the proposed system

Figure 2 provides a schematic mapping the TRADOC FCS functional requirements to the expected software requirements. Clearly, this is not an exhaustive list of C^2 requirements. However, we believe that the list is representative of the challenges placed on software of the networked C^2. In the next section, we evaluate how software technologies are equipped to meet these challenges.

TRADOC Requirements	Distributed Computing	Fault Tolerance	Security	Mobile Code	Information Fusion	Information Analysis Summary	Decision Support	Software Productivity
Common Operational Picture	X	X	X		X	X	X	X
Mobile Command	X	X	X					X
Mission-Centric IS	X	X			X	X	X	X
Decision Support/Planning			X		X	X	X	X
3D Visualizations						X		
Continuous Mission Planning						X	X	X
Synchronized C^2	X	X	X		X			

Figure 2 A mapping of TRADOC FCS functional requirements to expected software requirements.

3   State-of-the-Art Software Technology

Software development methods have been transformed over the years from structured analysis methods, where processing and data were kept separate [7], to object-oriented methods, where processing and data are combined into software entities called objects [8]. Object technology has been further enhanced with distributed capabilities, allowing an object on one system to communicate with objects on other systems [9]. There is also the capability for an object to be transmitted across a trusted network and executed on another computer, a technique commonly known as mobile code [10].

What we must consider at this point is whether the development of software technology has reached a stage of sophistication that will allow it to meet the seven FCS C2 software requirements listed above. If so, the use of any less mature technology would be ill-advised. A full analysis of these very broad requirements is beyond the scope of this paper. Instead, we provide a very general review of the state-of-the-art in relation to these requirements and note some obvious limitations with respect to the FCS environment. These limitations will then be assessed against the capabilities of agent technology (Section 5 below).

3.1   Distributed Computing

Distributed computing or ubiquitous computing is the vision that devices ranging from super computers to nanoscale processing units will be able to communicate and act in concert to solve problems. The distributed computing approaches widely in use today include the Common Object Request Broker Architecture (CORBA) [11], the Distributed Component Object Model (DCOM) [12], and Remote Method Invocation (RMI) [13]. Each of these approaches provides a way of executing a software function needed by one computer on a different computer. To be executed remotely, this functionality places a number of constraints on the software. For example, assume that a source object[1] is attempting to execute some function on a target object; for this to happen, the source object must have the capability to resolve the network and computer memory address of the target object. Next, the source object must have detailed prior knowledge of the functions (methods) and parameters available on the target object, as well as the expected return information. There are also assumptions that these remote functions will be accessed synchronously and that the network connections are available and permanent. If any of these assumptions does not hold, then these distributed interactions will fail [14].

It is very unlikely that all of these above assumptions can be relied on in the dynamic FCS environment. Therefore, a C^2 system build on the current distributed object models is unlikely to succeed without significant enhancement.

The communication topology of the current distributed computing models is another potential limitation. This topology is typically a client-server model, in which the client sends a request to a server and then waits for a response. In the FCS C^2 network, messages will need to be drawn from a richer model of interaction than the client-server model, since the structure and stability of the network is likely to change at any time. A message may need to be broadcast to several sites, relayed by several objects, retransmitted, or postponed, depending on the nature and status of the network, which is a very complex challenge for the client-server model.

---

[1] For ease of discussion, we will refer to software programs or functions as objects.

3.2 Fault Tolerance

Fault tolerance is concerned with making a distributed system more reliable by handling faults within the system. A great deal of work has been done in this area, culminating in formal fault tolerant models. These systems are usually described as having the properties of safety and "liveness." Safety properties consist of the set of acceptable system configurations, or invariants, defining the operations that are legal within a distributed system. "Liveness" describes the notion of the progress of a task within the distributed system. For example, safety properties may require that an FCS vehicle cannot fire on friendly troops, while a liveness property may require that a friendly troop notification will arrive at the appropriate FCS vehicle or force warrior.

Ideal fault tolerance provides that all safety and liveness properties are guaranteed to be satisfied within a software program. When neither safety nor liveness properties are guaranteed, the software program has no fault tolerance. If only safety properties can be guaranteed, then the program will not violate system invariants but may not complete the task—i.e., the system will not fire if a friendly troop notification has not been received, or in other words, the system is failsafe. If only liveness properties are met, the system will fire, and may find out when the notification arrives that friendly troops were fired on [15].

The key to fault tolerance is redundancy and the ability to detect and correct faults. These concepts are mainly design principles that need to be enforced during the construction of software. However, there are some practical technology limitations to fault tolerance based on current distributed computing models. The client-server model, as described above, limits the capability for message redundancy within a distributed system. A client passes a message to a server and waits for a response. If the client, the message, or the server suffers a fault, the transaction will fail. This can significantly limit the fault tolerant capability of current technology in an FCS environment.

3.3 Security

Security ensures that data can be safely transmitted within the FCS system. The nodes within the system can be authenticated, and data securely communicated. Existing security systems tend to be static; consequently, security policies and mechanisms are very difficult to change once the systems are installed. With systems that support a ubiquitous and/or mobile computing environment, the fundamental problem that arises is to provide security that is expressive and flexible enough to satisfy the specific needs of diverse applications [16].

Security operations are typically based on a security policy that defines which operations are proper and should be allowed. A security policy usually specify access, accountability, authentication availability, maintenance, violations reporting and response, and support information about interaction with entities that are either unknown or known but non-local. If such a policy can be enforced, and there are no violations, the system is secure by definition. The goal is to create software than can enforce such a policy.

There is certainly existing software that meets some of the demands outlined above, but nothing capable of supporting the size, distribution, and lifecycle requirements that will flow from the relevant FCS scenarios. Software protecting individual computers—firewalls, intrusion detection systems, password mechanisms, Public Key Infrastructures (PKI), and so on would make FCS operations, relatively, but not absolutely secure in the sense outlined above. To meet FCS

demands, it is imperative that the FCS security system be unified, policy-based, and dynamic. Current COTS systems are relevant but only marginally capable of meeting these requirements.

3.4 Mobile Code

The term "mobile code" typically refers to a capability whereby a combination of data, code, and execution state is sent to another machine and executed on that machine through a general virtual machine. The virtual machine may take the form of a distributed system layer, such as CORBA, or as a computational environment, such as the Java Virtual Machine. Currently, there are three design paradigms for a mobile code system: (1) a code-on-demand system allowing code to be transmitted to the data, (2) a remote evaluation system allowing code and data to be moved to another system, and (3) a mobile agent system allowing code, data, and state[2] to be moved to another system [17].

FCS levies very demanding requirements for mobile code. There is no guarantee that any node in the C^2 network will be available at any one time. Therefore, the design paradigms represented in 1 and 2 above provide limitations if the source node is no longer available to hold the code or state of a mobile transaction. The third paradigm, mobile agents, will be discussed in the next section.

Security—most notably, how to prevent malicious software from entering a system—is a major issue with mobile code. A typical solution is to prevent state from being sent with the code—i.e., mobile code is generally executed in a very narrow computational space where the target memory is not accessible and can only communicate with the source system. It appears that this approach may not be viable in an FCS environment.

3.5 Information Fusion

Fusing data from different sources is a difficult problem. The most promising technique for doing so appears to be the use of a metadata tag language such as Extensible Markup Language (XML) [18]. With this approach a common ontology or set of XML tags is developed. Then specific data is tagged using this common ontology and can then be combined with data from other sources [19]. Kim argues that ontologies will be best for reducing uncertainty, while XML will be most effective in reducing the complexity of the shared data [20].

This approach shows great promise. Unfortunately, tagging data does not necessarily ensure that the data can be fused. There are many examples where it is technically impossible to fuse data derived from different relative scales or with differing assumptions. The ultimate goal of data fusion is for the software to understand and manipulate the data, which has been an open issue for decades.

3.6 Information Analysis and Summary

After data are fused, there is likely to be a need to analyze the data for a wide variety of reasons. Typically, this analysis will result in reducing the size of the data being analyzed. This provides for faster processing and transmission of the data. There are a number of mathematical

---

[2] State is a description of a partially completed process, including the values of all program variables and which step of the process is the next to be executed. State information is necessary in order for another computer to complete a process that another has begun.

techniques for analyzing and reducing data—feature extraction, dimensionality reduction, principle component analysis, and cluster analysis, to name a few. These topics are orthogonal to state-of-the-practice software methods but are very important to addressing the networked C^2 challenge of FCS.

3.7 Decision Support

After data has been gathered, fused, and analyzed, this information would typically be used to make military decisions. A number of decision-support methods and systems can be used to perform this task. As with information analysis, decision support models are not dependent on the state-of-the-practice software methods, yet are very important to addressing the networked C^2 challenge of FCS.

3.8 Software Development Productivity

The proposed FCS networked C^2 functionality will be very large and particularly complex by today's standards. The engineering effort to assemble such a resource is challenging in both effort and risk. Object-oriented methods have been shown to produce simpler designs and provide a greater capability for reuse than other methods. However, object-oriented technology has not been shown to improve software development productivity in a commercial environment [21]. While simpler designs are clearly desirable in building new software systems, the need for improved productivity is a significant concern as well.

3.9 Software Development Challenges Posed by FCS

As is apparent from the preceding discussion, a number of challenging software requirements that must be met to build any networked C^2 system, much less the proposed FCS concept. We have analyzed the functional requirements to produce a reasonable set of software characteristics needed to create this system. We have then analyzed these software requirements to understand the key technology challenges posed by these requirements, see Figure 3. From this figure, the distributed computing requirement poses the greatest software challenge for the new FCS system, while information fusion, information summary and analysis, and decision support are tangential to software technology advances.

Software Limitations	Distributed Computing	Fault Tolerance	Mobile Code	Security	Information Fusion	Information Analysis Summary	Decision Support	Software Productivity
Higher-level Interfaces	X			X				
Asynchronous Interaction	X							
Sporadic Network Support	X	X	X					
Security			X	X				
Peer-to-peer Models	X	X						
Software Productivity								X

Figure 3 A mapping of the software requirements to the limitations of the current software technology

Our analysis indicates six keys software challenges in building this system:

1. **Providing higher-level interfaces to distributed objects.**
2. **Allowing asynchronous object interaction.**
3. **Providing message support for sporadic network connections.**
4. **Providing secure object communication and information system operation.**
5. **Providing support for richer peer-to-peer programming models.**
6. **Increasing software development productivity.**

In the next section we evaluate the suitability of agent technology against these six challenge areas.

4 Agent Technology

Agent technology is an evolving paradigm that strives to create software that can mimic certain human behavior. Agents are typically described as possessing human characteristics, for example, agents are normally considered to be autonomous, adaptable, social, knowledgeable, mobile, and reactive to name a few [22]. The focus of much discussion about agents is on the characteristics of agents. While this can be a very useful abstraction for discussing agents, it does not provide a strong means of objective comparison. For the purposes of this paper, we are more interested in the computer science novelties of the technology; therefore, we will limit the discussion of characteristics, and focus strongly on the comparative benefits of agent technology.

There are many proposed and deployed agent architectures. A representative architecture by Sycara et al. [23] proposes planning, communication and coordination, scheduling, and execution monitoring of agent activities. In this architecture, the agents have access to shared information, typically implemented through a coordination model that can be domain specific or domain independent. Another architecture description is offered by Griss et al. [24] who provide a broad description of a general agent architecture where the architecture provides facilities for locating and communicating with moving and unconnected agents, and for gathering information about groups of agents. This architecture provides services that include support for mobility, security, management, persistence, and naming of agents.

These architectures and most others highlight the communication and control aspects of agent systems, which are typically provided by a general messaging paradigm where one agent can communicate with one or several other agents. This messaging approach encapsulates the messages that agents send and receive [22]. Object-oriented methods popularized the concept of data encapsulation, which provides for simple software functions to access an object's data. These functions, not direct data access, are then used to retrieve and update this data. This capability limits the software that must change when minor changes are made to the data. The agent paradigm extends encapsulation from data to messages sent among agents. This capability is provided through agent coordination models [25]. These models define how agents communicate among themselves, and can be seen as coordinating communication based on the time a message is sent (temporal) or the names of the target agents (spatial). These models provide the ability for communication that is encapsulated and asynchronous with the use of

blackboards, and tuple space models and associated pattern-matching, such as Linda [26]. Agents that use a blackboard or Linda type coordination model provide a level of indirection for agent communication. In other words, an agent sends a message to a blackboard, and those subscribers to the blackboard retrieve the message. The agent that sent the message may have no idea who actually receives it. This concept allows for asynchronous and encapsulated communication among a collection of connected or disconnected agents, a capability that currently not available in non-agent systems.

Another aspect of agent messaging is that these messages are typically written in an agent control language [27] (ACL) such as KQML or the FIPA ACL. These languages provide a structured means of exchanging information and knowledge among agents. ACLs provide support for a higher-level communication protocol that currently does not exist with distributed objects.

We will now review in detail how suitable agent technology is for the software development challenges posed by FCS.

4.1   Higher level interfaces to distributed objects

Agent technology is based on a flexible messaging scheme and agent control languages. Agents conceptually are connected to blackboards, not other agents. The encapsulation of messages allows for an agent interfaces to change, requiring only minor modifications to a blackboard, not to all calling agents [22]. This capability provides for a more robust interface than is currently available in distributed object systems.

Another advantage of agent messaging is that ACLs provide the ability to pass propositions, rules, actions, and states among agents. This means that messaging is not merely a way of activating a function on a remote agent, but provides a way of sending information to another agent. The agent can then decide what to do about this information, if anything. This information can be used to describe what requirements need to be met for an agent to take action, what states the sender and receiver will be in after the action takes place, or what states the agents will be in when the overall transaction is complete [27]. Information sent from one agent to another may also be informative or declarative, having nothing to do with instructing the receiving agent to take action.

The challenge of implementing such an agent interface is selecting both a messaging architecture and an ACL. Currently there is not a universally accepted messaging architecture or ACL. For an agent system to take advantage of this high-level interface, there must be very specific and precise specifications on how agents will communicate, and on the precise syntax of the ACL.

4.2   Asynchronous object interaction

Griss et al. [24] points out that agent systems typically have simple interfaces, and derive capability from loose coupling and asynchronous messaging. This capability of asynchronous messaging is results from the ability of a message to be sent to and retrieved through a loosely coupled temporal agent coordination model. Cabri et al. [25] reference two coordination models that provide asynchronous agent communication. The first model is a blackboard-based model that provides a shared area where agents can send and retrieve messages. A message is posted to a blackboard by an agent, and other agents have the ability to read the message posted by that agent. The sending agent's identifier is used by other agents to determine whether to retrieve the

message. A blackboard-based system can be considered asynchronous; however, knowledge of the agent identifiers is required. The second model is based on a Linda coordination model approach. These models define a messaging protocol which is made up of a tuple of information, for example a tuple may include the data format, the date of creation, the classification, or a list of keywords. These tuples are then placed in a shared area, such as a blackboard. Agent can access these messages, not based on agent identifiers, but on a query of the tuple information, i.e., an agent may retrieve all messages created yesterday with the "Taliban" keyword. This type of model is asynchronous, and does not require knowledge of the agent identifier.

Both of these types of models are mature, and widely used in agent systems today. They provide the type of asynchronous behavior that is required by the FCS system. Clearly, a system that uses a single blackboard for all agent communication is exposed to security and performance failures. An operational agent system would require multiple blackboards supporting redundancy to provide a more fault tolerant system.

4.3   Message support for sporadic network connections

Providing software that can effectively function over a faulty network is a very challenging, if not impossible problem. The advantage that agent technology provides in this type environment is the flexibility and redundancy of the communication paths among agents, and the ability for agents to change location. Vogler et al. [28] propose a distributed transaction model using a two-phase commit protocol to verify that an agent message has been delivered. This very well known approach can provide a means of ensuring that an agent transaction has successfully or unsuccessfully completed. The agent coordination model must support the ability for an agent to store undelivered messages within the agent, or support the ability to rollback the transaction, if synchronous transactions are required. If a transaction has not completed successfully, then a number of network or graph theory algorithms can be used to determine a viable path through the network, and the transaction can be attempted again, or the agent can move to another location and try again. If a physical path cannot be found then the transaction is not possible.

The messaging architecture and mobility of agents can be effectively used to communicate over a sporadic network, however, there is a point where the network can degrade to a point where agent communication is no longer possible. Distributed transaction protocols (DTP) are very useful for verifying the success of transactions, and can be used to ensure network security, however, adding this capability can limit the performance of the overall system.

4.4   Secure Communication and Information Operations

As Abadi [29] notes, it is practically impossible to construct a truly secure information system. Communications are secure if transmitted messages can be neither affected nor understood by an adversary, likewise, information operations are secure if information cannot be damaged, destroyed, or acquired by an adversary.

Most agree that security in a distributed system should be enforced through system wide security policies. There policies are often static, and difficult to modify and enforce with existing technology [16]. Agents have demonstrated that they can enforce a security policy defining what must be done and what must not be done when information is moved (including communication), stored, created, or destroyed. Agent technology is valuable in this context because it provides multiple, standalone, persistent processes that can act at high speeds to

ensure that all the rules are always followed. Encapsulated instructions concerning what actions to take under what circumstances enables agents to execute very complex operations, enabling agents to participate in complex collaborative security protocols such as key updating and multiparty authorization.

There is no overt reason agents cannot be designed to provide a very secure information management system within the FCS environment. The challenge for FCS is in defining the FCS system-wide security policy and designing agents able to enforce it without undo complexity or performance limitations.

4.5   Peer-to-peer programming models

Through the use of blackboard and Linda type coordination models, the programming model of agents can be very general. Any number of agents can send messages to one or many blackboard(s), and any number of agents can receive messages from one or many blackboard(s). This provides the building blocks to create virtually any network topology that can be defined, and allows for very broad scalability of the network. Care must be taken in defining the bandwidth, messaging rates, and processing requirements of the network. These topologies will require tuning to enhance fault tolerance and performance.

4.6   Increasing software development productivity

There are indications that agent technology may provide some software development productivity improvement [24]. While there does not appear to be any empirical evidence to support this to date, the proposed theory is that agents increase the level of software reuse. Agents are software components that have their messaging, functionality, and location encapsulated, which is believed will increase the level of software reuse, thus increasing productivity. Likewise, if standard messaging protocols and ACLs can be defined, the agent development teams may require less communication overhead since the interfaces are far richer than with traditional programming.

5   Specific Agent Projects and Technological Readiness

In this section we briefly review a handful of existing agent-based systems that appear to address FCS C^2 requirements. This analysis provides a brief glimpse into the state-of-the-practice of agent technology. This review looks at the published reports of the systems, not the actual systems themselves. The assessment of technology readiness level (TRL), see Appendix A, is performed strictly from the open literature, and may not accurately reflect the TRL level of the actual system.

5.1   Cooperating Agents for Specific Tasks (CAST)

Principal investigator: Kenneth Whitebread
Affiliation: Lockheed Martin Advanced Technology Laboratories
URL: http://www.atl.external.lmco.com/indexlist.html

Lockheed Martin has significant experience developing agent-based systems for military applications. We focus here on Cooperating Agents for Specific Tasks (CAST), which is affiliated with the DARPA Control of Agent-Based Systems (CoABS) program. The CAST system performed C^2 functions for Theater Air Missile Defense (TAMD) during USN Fleet

Battle Experience. The CAST system manages large amounts of distributed information and provides COP and situation awareness data in the TAMD domain supporting naval C^2 of surveillance and strike assets. CAST does not support large numbers of distributed information sources and links, and scaling properties are unknown. However, Lockheed Martin Advanced Technology Laboratories also developed and deployed the Domain Adaptive Information system (DAIS) with the Army 201st Military Intelligence Brigade. DAIS was built to query heterogeneous databases over unreliable low-bandwidth networks. Although it is safe to say that neither of these systems would be capable of meeting FCS C^2 requirements, according to their information, both perform aspects of these requirements very well and both are at high TRL: CAST is TRL 7 and DAIS is TRL 9.

5.2 Dartmouth Agent (D'Agent) Multidisciplinary University Research Initiative (MURI) Demonstration

Principal investigator: Robert Gray
Affiliation: Dartmouth College
URL: http://actcomm.thayer.dartmouth.edu/

The D'Agent MURI demonstration focused on a small number of distributed agents deployed in support of low-intensity-conflict urban operations, specifically location and arrest of a specific individual. The agents operated within a dynamic network maintaining two-way C^2 connectivity among mobile soldiers and a static command post in a realistic outdoor urban environment. The commercial off-the-shelf (COTS) hardware used in the demonstration would not serve in an FCS mission environment, and it is not clear whether the software would scale; the number of participants in the three demonstrations have been in the low tens of individuals. However, good measures of performance and logs were taken, the entirety of which can be seen online at the above URL. This work falls at TRL 6. Achievement of TRL 7 would require mission-relevant hardware and a more realistic Military Operations in Urban Terrain (MOUT)–like test environment.

5.3 Standard Agent Architecture (SAA) Development Program

Principal investigator: Steven Goldsmith
Affiliation: Sandia National Laboratories Advanced Information Systems Laboratory (AISL)
URL: http://www.aisl.sandia.gov/

Sandia's Advanced Information Systems Laboratory (AISL) has focused on providing agent technology to cooperatively manage and protect complex operations on critical data. The Standard Agent Architecture (SAA) program is unusual in that it uses no COTS agent technology but instead relies on a unique framework constructed in-house from first principles. SAA agents use KQML and HTML to communicate with non-SAA entities. Recent work is aimed at in-house deployment of the Boxer cybersecurity application that will detect specific types of otherwise undetectable anomalous transactions in high-volume TCP/IP traffic (TRL 5). Initial deployment will field only a few agents; however, Boxer is designed for expansion. AISL will also demonstrate C^2 of a mixed collective of nonrobotic agents, robots controlled by on-board agents, and semiautonomous non-agent robots near the end of 2002 (TRL 4). AISL has demonstrated multi-agent execution of several advanced cryptographic algorithms specifically designed to protect against stealthy penetration and individual system failure or cooption (TRL

4). When deployed, the Boxer system will be at TRL 6 (not technically TRL 7 because neither the hardware nor the personnel are military), but Boxer will be providing operational information to computer security operations personnel in an operational environment.

5.4 UltraLog Program

Principal investigator: Mark Greaves (program manager)
Affiliation: DARPA/IXO
URL: http://www.ultralog.net/; http://www.cougaar.org/sitemap.html

UltraLog is a DARPA program whose expressed goal is to improve the reliability and robustness of the Cougaar architecture by eventually deploying at least 1000 simultaneously functioning agents providing military logistics support in a major regional contingency. The primary contractor providing the Cougaar architecture and most of the development is Bolt, Beranek, and Newman (BBN). We would place UltraLog at TRL 6 or 7; there is room for interpretation as to whether the demonstration environment is an "operational" environment.

In any case, UltraLog at this time is focused on logistics, and is able to construct an operational plan to move large quantities of material to a given location. This involves several dozen distributed agents (i.e., the agents are not co-located) trading information about constraints, capabilities, commitments, and so on to arrive at a workable plan. This work begins to show that agent systems large enough to support FCS operations are possible. The agents are general-purpose with specializing behavior provided by "plug-ins," which are code modules written by the application programmers. BBN has also done substantial work to prepare Cougaar-based agents for FCS-like operation of unattended sensors and battlefield logistics.

5.5 Virtual Information Processing Agent Research (VIPAR)

Principal investigator: Thomas E. Potok
Affiliation: Oak Ridge National Laboratory
URL: http://www.csm.ornl.gov/~v8q/Homepage/Projects/vipar.htm

The VIPAR project uses the Oak Ridge National Laboratory (ORNL) Oak Ridge Mobile Agent Community (ORMAC) to address challenges facing the intelligence community for the U.S. Pacific Command (USPACOM). ORNL has used ORMAC to develop agent-based systems for the U.S. 6th Fleet, the Defense Logistics Agency, Lockheed Martin, and the Department of Energy. ORMAC is a blackboard based agent framework that uses FIPA compliant messaging, and supports full agent mobility.

The VIPAR system quickly gathers and organizes massive amounts of information, up to 10,000 documents, then distills that information into a form directly and explicitly amenable for use by an intelligence analyst. This system is deployed and in use at USPACOM. The USPACOM commander in chief Admiral Blair calls VIPAR "A tremendously successful project" where "Software agents … lead to substantially improved analytical products." The USPACOM Science and Technology Advisor calls VIPAR "a grand slam home run!" the "first time we've seen information discovery and knowledge management software working at HQ USCINCPAC operationally." This system is at TRL level 9, however, it only addresses a small part of the C^2 requirements for FCS.

6 Discussion

The analysis in this paper begins by deriving a set of software requirements for the FCS networked C2 system based on a set of TRADOC functional requirements. This set of software requirements is not an exhaustive set for C2, however, from a military point of view provides a credible and representative list of the challenges awaiting the software designers of FCS.

A comparison of these requirements with the capabilities of existing technology is very revealing. Several of the limitations of existing technology bring into question whether it is capable of producing a C2 system for FCS. The main limitations of existing technology are low-level interfaces, synchronous interactions, requirements for continuous network availability, limited redundancy, and limited productivity improvements. Clearly, the current technology would require major enhancements to be able to support an FCS environment.

Reviewing the limitations of existing technology against agent technology, we are able to assess the suitability of agent technology in the FCS environment. This assessment highlights the main strength of agent technology within an FCS environment, which are the messaging and coordination models that agents use. These models enable better solutions to the FCS challenges than do existing technology. The issue however, is to determine whether the theoretical capabilities of these models can be realized in practice.

We provide a brief review of some relevant agent work in related areas. This is a paper analysis that may not fully represent the actual systems, however, there appears to be ample evidence that agent systems have been used to solve some of the problems faced by FCS.

There are two main questions that this analysis raises, 1) should FCS be built on enhancements to existing technology or on an agent architecture? 2) Is agent technology mature enough to be used for a project the size and complexity of FCS? The first question deals more with an economic analysis than a technical analysis. If current technology is enhanced to solve some of its limitations, the resulting system will most likely look like existing agent systems. It does not make much sense to reinvent what already exists. The maturity of agent technology is an issue. There is not a reference agent system that supports the complexity or scale of the proposed FCS system. On the other hand, it is pretty clear that existing technology will not be able to solve this problem. Looking strictly at the success of the FCS project, it would appear that agent systems will perform at least as well as traditional systems, but with the promise of doing much better. Therefore, we recommend the use of agent technology for the FCS C^2 system.

There are some issues not related to software that must be addressed as well—namely, security, information analysis and summary, and decision support. Agent technology can clearly support these tasks, but the technology does not explicitly provide these capabilities, and these are challenging problems. If these problems cannot be adequately solved, regardless of whether or not agent technology is used, the FCS system will be limited.

We recommend the use of prototypes and experimentation with agent technology to reduce the software development risk of FCS, specifically in the areas of scalability, mobility, and security. The resulting information will provide a clearer picture of the expected benefits of agent technology.

7   Conclusion

The U.S. Army is transforming through advanced technologies to significantly improve its war fighting capability. The Army is looking for technologies that can provide dramatic improvements over existing capabilities, yet are reliable enough to provide a fielded system. Our study assesses both the potential improvements, and the reliability of using software agent technology for the network-centric C^2 portion of FCS. The emerging FCS concept of C^2 activities is a dynamic network of moving vehicles that will gather and analyze data from a vast array of battlefield sensors. This ad hoc network will have vehicles entering and leaving the network at unpredictable times. This system must be highly reliable and highly secure, with the ability to scale to process massive amounts of data. This proposed FCS C^2 network must be able to process this information rapidly and deliver the right information to the right locations and people at the right time.

Achieving a networked C^2 capability will require significant advances in existing software technologies. Key experts have proposed agent technology as a potential solution to this challenge. To analyze the capabilities of agent technology, we have developed a set of software requirements of FCS based on military requirements. These requirements are then reviewed against the current computer science literature to highlight limitations and challenge areas. These challenge areas are then reviewed against agent technology to illustrate the comparative benefits of this technology in an FCS environment.

From this analysis we find that the networked C^2 requirements of FCS are beyond the capabilities of existing technologies in scalability, mobility, and security. Agent technology provides a number of significant advantages in these areas, due to much stronger messaging and coordination models, and theoretically is much better suited to the FCS challenge that is existing technology. There are some mature agent systems that meet some of the requirements of FCS, but there is currently no single agent system that meets the scale and complexity proposed by FCS.

In summary, agent technology will clearly perform at least as well as traditional technology in an FCS environment, but with the promise of solving a number of existing technology limitations. Our theoretical and system level analysis shows that agent technology has the capability to support the significant networked C^2 requirements of FCS, requirements that likely pose unachievable challenges with current technology. In other words, agent technology is the best technology, perhaps the only technology, for delivering a viable C^2 system for FCS. To further strengthen this analysis, we recommend proof of principle experiments to verify and validate the results of this analysis.

8   References

[1]   Col. William Johnson, Program Manager, "Future Combat Systems," DARPA/Army Collaborative Future Combat Systems Demonstration Program, at http://www.arpa.mil/ tto/programs/fcs.html, accessed 4/30/02.

[2]   T. M. Carrico, "Vision and Concepts: Agent-Based Command and Control for FCS," *The UltraLog White Paper Series*, Darpa Technical Report.

[3]   DARPA/Army Collaborative Future Combat Systems Demonstration Program, "FCS Public Briefings," at http://www.arpa.mil/fcs/public.html, accessed 4/30/02.

[4]   T. Lee and S. Ghosh, "Simulating Asynchronous, Decentralized Military Command and Control," *IEEE Computational Sciences & Engineering* 3, no. 4 (1996): 69–79.

[5]   U.S. Army Training and Doctrine (TRADOC) Briefing, given at Eatontown, N.J. FCS Integrated Study Team Workshop, December 2001.

[6]   G. Fischer and J. Ostwald, "Knowledge Management: Problems, Promises, Realities, and Challenges," *IEEE Intelligent Systems* 16, no. 1 (2001): 60–72.

[7]   T. Demarco and P. J. Plauger, *Structured Analysis and System Specification*, Prentice Hall, New York, 1985.

[8]   G. Booch, *Object-Oriented Design with Applications,* Benjamin/Cummings Publishing, Redwood City, Calif., 1991.

[9]   R. S. Chin and S. T. Chanson, "Distributed, Object-Based Programming Systems," *ACM Computing Surveys* 23, no. 1 (1991).

[10]   T. Thorn, "Programming Languages for Mobile Code," *ACM Computing Surveys* 29, no. 3 (1997).

[11]   See the Object Management Group's (OMG's) CORBA web site, at http://www.corba.org, accessed 4/30/02.

[12]   M. Horstmann and M. Kirtland, "DCOM Architecture," July 23, 1997, at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomarch.asp, accessed 4/30/02.

[13]   "Java Remote Method Invocation - Distributed Computing for Java," White Paper, at http://java.sun.com/marketing/collateral/javarmi.html, accessed 4/30/02.

[14]   K. Geihs, "Middleware Challenges Ahead," *IEEE Computer* 34, no. 6 (2001): 24–31.

[15]   F. Gartner, "Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments," *ACM Computing Surveys* 31, no. 1 (1999): 1–26.

[16]   Z. Liu, P. Naldurg, S. Yi, R. Campbell, and M. Mickunas, "Pluggable Active Security for Active Networks," in *Proceedings of the Twelfth IASTED International Conference on Parallel and Distributed Computing and Systems* (PDCS 2000), November 2000.

[17]   A. Fuggetta, G. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Transactions on Software Engineering* 24, no. 5 (1998): 342–361.

[18]   Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006.

[19]   T. Potok, M. Elmore, J. Reed, and N. Samatova, "An Ontology-based HTML to XML Conversion Using Intelligent Agents," in *Proceedings of the 35th Hawaii International Conference on System Sciences*, January (2002).

[20]   H. Kim, "Predicting How Ontologies for the Semantic Web Will Evolve," *Communications of the ACM* 45, no. 2 (2002): 48–54.

[21]   T. Potok, M. Vouk, and A. Rindos, "Productivity Analysis of Object-Oriented Software Development in a Commercial Environment," *Software—Practice and Experience* 29, no. 10 (1999): 833–847.

[22]   N. Jennings, K. Sycara and M. Wooldridge "A Roadmap of Agent Research and Development" *International Journal of Autonomous Agents and Multi-Agent Systems* 1 no. 1 (1998): 7-38.

[23]   K. Sycara, A. Pannu, M. Williamson, and D. Zeng, "Distributed Intelligent Agents," *IEEE Expert* 11, no. 6 (Dec. 1996): 36-46

[24]   M. Griss and G. Pour, "Accelerating Development with Agent Components," *IEEE Computer* 34 no. 5 (May 2001): 37-43.

[25]   G. Cabri, L. Leonardi, and F. Zambonelli, "Mobile-Agent Coordination Models for Internet Applications," *IEEE Computer* 33, no. 2 (Feb 2000): 82-89

[26]   D. Gelernter and N. Carriero, "Coordination Languages and Their Significance," *Communications of the ACM* 35 no. 2 (Feb. 1992): 96-107.

[27]   Y. Labrou, T. Finin, and Y. Peng, "Agent Communication Languages: The Current Landscape," *IEEE Intelligent Systems* 14 no. 2 (March 1999): 45-52.

[28]   H. Vobler, T. Kunkelmann, and M. Moschgath, "An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions," *Proceedings of the International Conference on Parallel and Distributed Systems* (1997): 268-274.

[29]   M. Abadi, "Secrecy by Typing in Security Protocols," *Journal of the ACM* 46, no. 5 (Sept 1999): 749-786.

[30] J. Mankins "Technology readiness Levels: A White Paper"; *Advanced Concepts Office, NASA Office of Space Access and Technology*; (April 1995); http://see.msfc.nasa.gov/see/WorkShop/TRL%20Descriptions.doc