

A Fire Control Architecture for Future Combat Systems

Dr. Malcolm Morrison, Dr. Joel Sherrill, and Ron O'Guin
OAR Corporation

Deborah A. Butler
U.S. Army Aviation and Missile Command

The Future Combat Systems (FCS) program is developing a versatile, reconfigurable system of systems capable of performing a wide range of missions for the U.S. Army. In support of the FCS, the Fire Control-Node Engagement Technology (FC-NET) program is developing a versatile, modular fire control software architecture capable of satisfying the flexibility and rapid mission reconfiguration requirements of the FCS. The FC-NET software architecture achieves its flexibility through domain-centric software components that encapsulate weapon-specific hardware devices and algorithms. This article illustrates the structure and organization of the architecture using the Munitions domain as an example. A side benefit of developing and testing the FC-NET architecture is the production of reusable artifacts that offer potential cost and schedule savings on future FCS evolution and system integration efforts.

The U.S. Army's Future Combat Systems (FCS) program is developing a network-centric ensemble of systems capable of performing a range of missions, from warfighting to peacekeeping. Capabilities envisioned for FCS include direct and indirect fires, air defense, reconnaissance and surveillance, transport, and resupply [1]. Current FCS vehicle concepts range from unmanned aerial vehicles to manned 20-ton wheeled or tracked vehicles to small robots that weigh only a few pounds [2].

Developing systems that bring the FCS vision to reality requires the collaborative efforts of numerous communities. Architecture descriptions of the FCS will enable those communities to quickly and efficiently engineer, procure, and deploy advanced systems. The Fire Control-Node Engagement Technology (FC-NET) software architecture is being developed by the U.S. Army Aviation and Missile Research, Development, and Engineering Center as a foundation for fire control systems that support FCS. The essential characteristic of FCS is mission adaptability, and FC-NET provides a fire control software architecture that is equally adaptable.

Fire control in general encompasses all operations required to apply fire on a target [3]. Fire control systems are categorized as either tactical or technical. Tactical fire control systems are Command, Control, and Intelligence systems that focus on the planning and evaluation aspects of fire control. Tactical fire control systems are responsible for such activities as identifying targets based on data from multiple sources, prioritizing targets, assigning specific weapons for use against specific targets, and assessing damage after engagements.

Technical fire control systems are normally embedded in a weapon system and focus on the computational and mechanical operations required for that weapon system to hit a specific target with a specif-

ic munition. Technical fire control systems are responsible for interacting with tactical fire control systems to obtain information about targets. This information may be used to direct fire or to further refine firing solutions using organic sensors under the control of the technical fire control system.

Once sensors have acquired a potential target, the technical fire control system assists in tracking the target until a decision is made to engage and fire upon the target. Platforms and vehicles provide mobility for aiming weapons and sensors. Technical fire control systems augment the soldier's capability, enabling the soldier to fire on more targets both more quickly and more accurately. Figure 1 illustrates this role of technical fire control systems as force multipliers for the individual soldier.

FC-NET is a technical fire control software architecture. The architecture's flexibility ensures that FC-NET-based technical fire control systems can readily interact with tactical fire control systems and fully

exploit the information that the tactical systems provide. However, the architecture's structure emphasizes the technical actions required to place munitions on targets.

As used in the remainder of this article, the term *fire control* should be interpreted as meaning *technical fire control*.

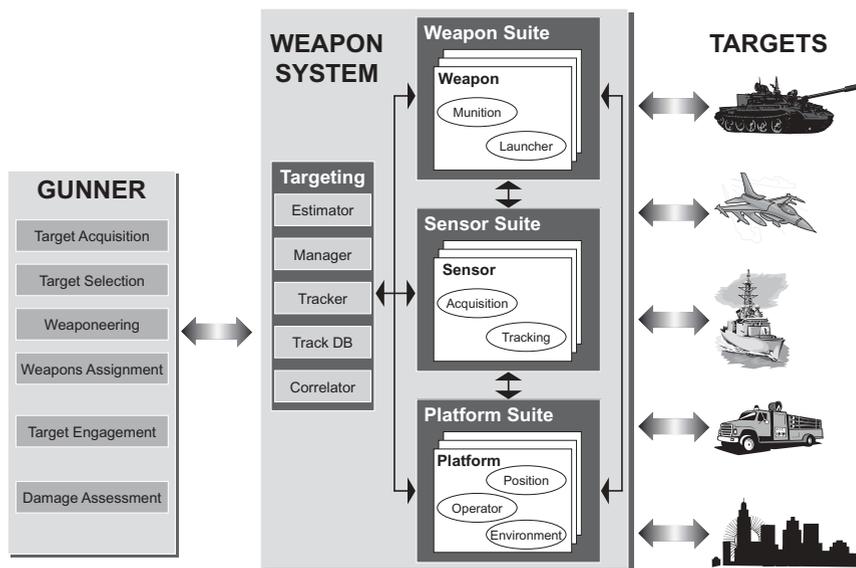
The FC-NET Architecture

FC-NET captures technical fire control functionality in a modular software architecture that provides a *plug-and-fight* capability for FCS. The following sections provide a brief description of how the architecture was developed from domain models, how it is represented as a set of software components, and how it is expected to evolve.

Domain Modeling

A domain is defined by systems that perform similar missions. For example, bullets, rockets, and missiles all belong to the Munition domain. A domain model is an abstraction of the systems within a

Figure 1: Technical Fire Control Is a Force Multiplier



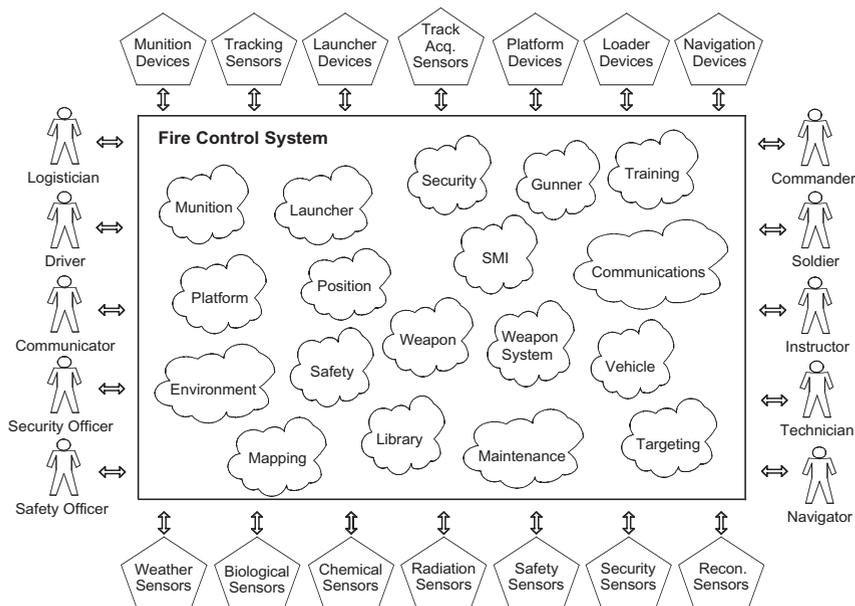


Figure 2: Fire Control Domain and Interactions

domain. A domain model is built for the purpose of understanding the domain's requirements, information, and processes. The top-level domain for the FC-NET architecture is military fire control systems. Therefore, the FC-NET architecture domain model is a model of the requirements, information, and processes associated with military fire control systems.

The FC-NET domain modeling activity examined the information required by a fire control system to execute a fire mission. Since fire control systems are composed of a variety of sensors and electro-mechanical devices, it was critical to classify the devices based upon the information they provide and the roles they play. This focus on devices helped draw a strong boundary between the problem space addressed by the FC-NET architecture and that of subsystems with which a fire control system must interact. Examples of such subsystems include battlefield management, avionics control, vehicle control, and user interfaces.

Figure 2 identifies the roles and devices that interact with the top-level fire control system domain. Within the fire control system domain, a number of other lower-level domains (or sub-domains) have been identified such as Weapon and Munition. Each of these lower-level domains represents a significant area of functionality and serves a coherent role in a military fire control system. Each lower-level domain is further decomposed into units called components.

Components

A component is a logically coherent, distributable unit of software composition.

The behavior of a domain emerges from the interactions among its components and between its components and the external environment. The following criteria were used to determine if an area of functionality should be encapsulated into a single component:

- **Logically Coherent.** Is there a collection of functional capabilities that should be grouped because of interdependencies?
- **Distributable.** Could this functionality be distributed?
- **Vendor Provided.** Could a vendor provide this functionality as a standard component?
- **Special Expertise.** Does a subject-matter expert normally write this functionality?
- **Updateable.** Could updates to this functionality occur independently of updates in other areas of functionality?
- **Optional.** Is this functionality only required in some systems?

The following example makes using these criteria clearer. A computationally intensive algorithm for target recognition might execute on a separate processor from software controlling the sensor devices. In this case, it would be desirable to be able to make sensor control and target recognition separate components to provide flexibility to the system designer in assigning these to separate processors. It is also likely that a subject-matter expert would provide the target recognition algorithm. The subject matter expert would normally not be responsible for the software interfacing with sensor devices.

Ideally, the manufacturer of the sensor device would provide this functionality. Finally, it is reasonable to expect improvements to both target recognition algorithms and sensor devices. It is likely that these would occur independently of one another.

In this example, the criteria suggest that target recognition capabilities and sensor device control be placed in separate components. The separation of functionality illustrated by this example is supported by FC-NET's adherence to an open system model with standard component interface definitions.

The FC-NET architecture identifies components within each of the lower-level fire control domains. The specifications for individual components are defined in terms of the attributes they possess, the services they offer, and the asynchronous notifications they may generate.

Attributes may be thought of as data elements. However, they are only accessible to clients of the enclosing component via services that access or modify the attributes. These accessing services are known informally as *get* and *set* and implicitly exist for every attribute of a component. In multi-threaded environments, accessing services are assumed to be thread-safe and provide atomic access to attributes.

Services define operations that can be performed by the component. Services may modify the values of attributes and may invoke the services of other components. Services may perform computations or effect changes in the external environment. For example, a fire control system must provide an Aim service that results in parts of a weapon system being physically repositioned so that munitions can be fired at targets.

Notifications are asynchronous messages generated by a component when an event of interest occurs. Other components subscribe to receive those notifications in which they have an interest. For example, the Department of Defense defines a hang fire as a non-desired delay in the functioning of a firing system [3]. A hang fire occurs when a weapon is fired but the munition does not physically leave its launcher. A notification is used to pass knowledge of this event to other components that need to know when a hang fire occurs.

For ease of modeling, the FC-NET domain model classifies components within each domain into one of five categories: functional controls, knowledge stores, device groups, algorithm containers, and façades.

Functional controls encapsulate sets of

functionality that have a similar purpose and perform a specific role within a fire control system. Since functional control components have specific roles, they are named according to their role. Functional controls always have services but sometimes do not have attributes or notifications.

Knowledge stores encapsulate groupings of related information. Each knowledge store component provides controlled access to its information, even in the face of concurrency. In general, knowledge stores are global repositories of system data. All knowledge stores are named according to the type of data they encapsulate. Knowledge stores tend to have attributes and implicit accessing services for reading and modifying the attributes, but rarely have any explicit services or notifications.

Device groups encapsulate collections of sensors or controllers that are used for similar functions. Specific devices are not defined within a device group. Since device groups represent hardware components, they are named according to the generic hardware that they represent. Device groups tend to have attributes, services, and relatively many notifications.

Algorithm containers encapsulate computations that are highly complex, likely to be system-specific, and typically written by subject-matter experts. In the fire control domain, examples include weapon/target pairing, target identification, and ballistic computations. Algorithm containers tend to contain very few services and no attributes or notifications. Algorithm containers normally operate in demand mode – executing only in response to requests from other components or as a result of system events.

Façades provide high-level interfaces that make complex subsystems easier to use by encapsulating the data and functionality of the subsystems. A façade is especially useful when the subsystem is highly complex, adheres to another domain model, or is produced by an outside organization. Although façades tend to have moderately complex interfaces, the implementation of a façade usually consists of straightforward mappings between subsystem and façade attributes, services, and notifications. An example of a façade in the FC-NET architecture is the Vehicle component. The Vehicle component is responsible for presenting the fire control system with a unified interface to all the subsystems that are part of (or are attached to) the vehicle on which the weapons are mounted. Common examples of such vehicle-mounted subsystems include power plant control and monitoring systems,

speed and position sensors, meteorological sensors, and nuclear, biological, and chemical detection systems.

Examples of these component categories are provided in a later section of this article that presents a decomposition of the Munition domain.

Architecting for the Future

FCS is being designed as a highly adaptable and flexible fighting platform, but it is still largely conceptual. The modular, component-based modeling approach described earlier provides the architectural adaptability and flexibility required if FC-NET is to support an evolving FCS. Although the exact physical nature of FCS is not known with certainty at present, initial concept definitions hint at possible configurations.

Possible FCS Configurations

For this discussion, a *weapon* is defined as the composition of a platform, one or more launchers, and one or more munitions. Weapons may be fixed or mobile. If mobile, the weapon platform must be mounted on some type of vehicle. The overall movement capability of a weapon depends on both the movement capabilities of a vehicle and the articulation capabilities of a platform. Likewise, sensors used to acquire and track targets require movement capabilities similar to the weapons they help control. Conceptual depictions of FCS to date embody, at minimum, the three different weapon and vehicle configurations shown in Figure 3.

Figure 3(a) reflects the simplest case where a weapon and its sensors are bore-sighted along the same line. In this configuration, all sensors and weapons effectively are aimed at the same point and moved at the same time. Thus, aiming at and tracking a target with a sensor results in the weapon aiming at and tracking that same target.

A step up on the complexity scale has each sensor or weapon located on a single vehicle but capable of independent movement. This configuration is shown in Figure 3(b). When weapons and sensors are mounted on the same vehicle, their actions must be coordinated to avoid interference and ensure proper aiming and alignment.

The most complex and most flexible configuration allows sensors and weapons to have completely independent movement. This situation is depicted in Figure 3(c) and reflects the growing utilization of unmanned ground and aerial vehicles. Coordination of sensors and weapons is still an issue, but the coordination may need to be effected across significant distances.

Personality Modules

Isolation of the fire control system from the underlying hardware configuration is handled in FC-NET by using Personality Modules (PMs). Similar to the device drivers used by popular operating systems, PMs encapsulate device-specific hardware characteristics. A PM implements an architecture-defined interface to the fire control system. The PM translates abstract fire control commands into device-specific commands understood by the attached device. Figure 4 (see next page) presents an example.

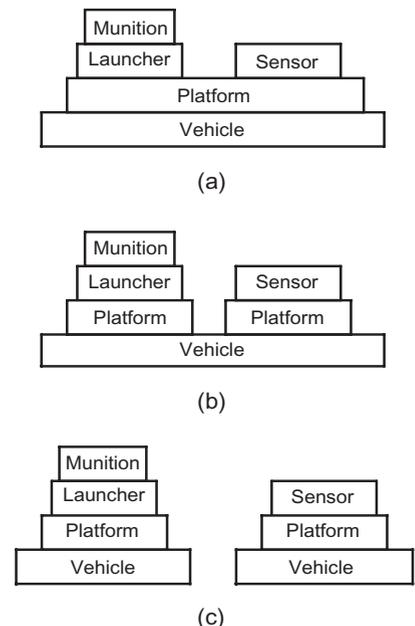
The fire control system sends a *Fire* command to the software component that controls a missile launcher. The software component passes the fire command to the launcher hardware through a launcher-specific PM. The PM translates the fire command into a sequence of relay activation commands that control electrical signals to the launcher at the connector pin level.

Fire Control Foundation Classes

Although weapon system physical configurations may vary widely, the fundamental operations required to place fire on targets remain relatively constant. Any fire control system must acquire and track targets, compute firing solutions, and deliver munitions against targets. The FC-NET exploits this functional consistency wherever possible to support the FCS in all of its anticipated weapon configurations.

The modular, domain-oriented structure of the FC-NET architecture promotes commonality at an abstract level. The architecture is designed to encourage and

Figure 3: Possible FCS Configurations



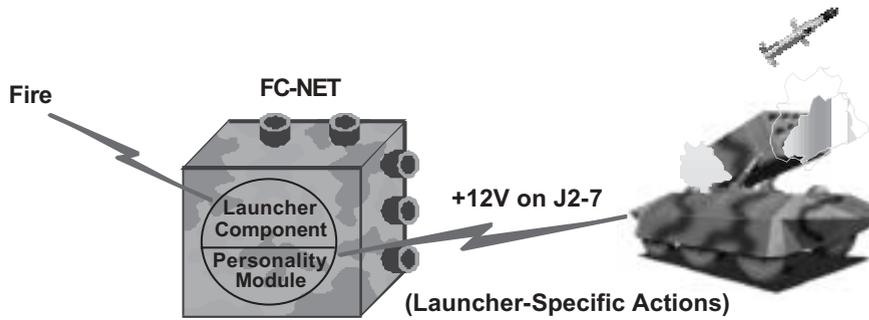
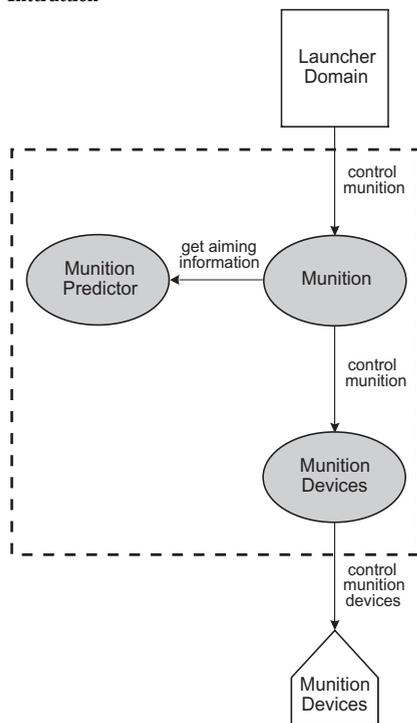


Figure 4: Personality Module Example

accommodate reuse. The architecture itself with its defined interfaces and services can be reused, as can implementations of components written in conformance with the architecture. As the FC-NET architecture is developed, a reference implementation of the architecture and a set of fire control foundation classes will be created along the lines of Microsoft Foundation Classes (MFC) [4]. An example of a potential fire control foundation class is geospatial position. Targets have positions, weapons have positions, and munitions have positions during their flight from weapon to target. A reusable position foundation class has been defined that provides position-related services such as transformations between coordinate systems. The position foundation class can be instantiated or extended by any fire control system implementer.

Just as the MFC are tightly coupled to Microsoft's Document-View application

Figure 5: FC-NET Munition Domain Interaction



architecture, the FC-NET fire control foundation classes will be tightly coupled to the FC-NET fire control architecture. Unlike the MFC, however, the FC-NET will be a non-proprietary open system portable to a variety of operating systems and central processing unit families.

Sample FC-NET Architecture Domain

Concepts discussed in earlier sections are illustrated by examining the part of the FC-NET architecture that addresses munitions. The FC-NET view of the Munition domain appears in Figure 5. The figure shows that the Munition domain contains three interacting software components: Munition, Munition Predictor, and Munition Devices. These components interact in turn with munition-related hardware devices and with other software components in the Launcher domain. Similarly, the Launcher domain is also composed of interacting software components that interface to hardware devices and to software components in yet other domains in the architecture.

Munition is a functional control component that is responsible for munition targeting and launching functions. Example services provided by this component include activating, arming, and launching a munition. The Munition component is responsible for sending notifications of events that occur during the course of its operations. For example, this component generates notifications when a munition is launched or stowed. These particular notifications normally originate in the Munition Devices component and the Munition merely propagates the notifications out to other components in the system. The Munition component also maintains information about munition attributes. This information includes basic characteristics, configuration information, and dynamic state. Example attributes include munition type and model number. Guided munitions may have attributes for laser designator code or waypoint list.

Munition Devices is a device group component that provides an interface for controlling the devices and sensors associated with a munition. It is used by the Munition component to access a munition's physical hardware. This component provides many of the same abstract services as the Munition component, such as aim, arm, and launch, but implements these services at a lower, hardware-aware level. The Munition Devices component generates the same notifications as the Munition component but does so at a lower level.

Munition Predictor is an algorithm container component that provides an interface for determining the necessary aiming conditions required to ensure that the munition's kill zone intersects the target. The Munition Predictor is used by the Munition component to compute the aim point for a munition. Like all algorithm containers, the Munition Predictor has no attributes and generates no notifications. The only services provided are computation of munition flight time, aim angle, and lead.

Design Trade-offs

Every software design involves trade-offs, and the FC-NET is no different in this regard. The FC-NET fire control architecture reflects the hardware structure and operational environment of modern weapon systems. The architecture is highly modular and assumes a hierarchical control model among components. The current version of the architecture defines 47 components, along with some additional data types and support services that collectively capture the behavior of 18 domains.

At first glance, this breadth and level of decomposition might seem excessive. Although this observation might hold true for many current fire control systems, the FC-NET was not designed for current fire control system. The FC-NET was designed for future fire control systems. In particular, it was designed for the FCS fire control system. The architecture was driven by the need to support the highly adaptable and reconfigurable weapon system that the FCS will be. Of particular concern is the potential for future weapon systems to be composed from cooperating, autonomous robots or physically distributed subsystems.

There are costs associated with such a modular and hierarchical architecture. Architectural simplicity may be reduced, as may implementation efficiency. The munition example presented earlier showed three independent, interacting components that together provide the necessary domain behavior. Other device-centered domains such as platforms and launchers also have

separate functional control and device group components. For simple devices, one could argue that this is design overkill. Given a dumb launcher device that does nothing more than transfer firing voltages to its mounted missiles, multiple components could introduce needless inefficiencies. Performance penalties can be incurred due to cross-component communications. Added complexity and risk can be incurred if it is necessary to locally cache data needed by multiple components.

Contrast this dumb launcher example with that of an intelligent launcher device with an autoloader, both mounted on an autonomous robot. Cross-component communications and local data caching become inevitable consequences of the hardware configuration. The architectural complexity that seemed excessive for the first example actually provides a smoother implementation path for the second.

Although complexity and inefficiency are legitimate concerns, current trends in processing speed and communication bandwidth ameliorate these disadvantages. Computer hardware gets faster and cheaper every year. The IBM PowerPC 750FX is a 32-bit processor that operates at speeds up to one gigahertz and is representative of high-end processors being used in new embedded systems [5]. Powerful processors can be teamed with Flash memory to provide large amounts of primary storage and virtual file systems. Advanced processors such as the PowerPC family have dramatically increased the capabilities of recent embedded systems, and the trend toward more powerful hardware is expected to continue.

In the case of distributed fire control systems, communications among physically independent components can have a major impact on overall system efficiency. The speed and reliability of network data transmission is almost always less than the speed and reliability of data transmission within a single processing unit. It is anticipated that distributed communication architectures such as Real-Time CORBA [6] will continue to mature and grow in reliability, performance, and usability to the point where communications issues cease to be a major performance concern for the architecture.

In light of these disadvantages, what advantages does the FC-NET architecture provide? Although we believe that there are potentially many, this article focuses on only three: adaptability, interchangeability, and vendor-independence.

An adaptable architecture allows fire control systems to be easily modified,

extended, or reconfigured in the face of changing requirements. Strategic defense policy formulation requires long-term planning. Long term in this context involves time horizons of 10 or more years as exemplified by Joint Vision 2010 [7] and Joint Vision 2020 [8]. Unfortunately, it is impossible to predict with any certainty what will happen to hardware and software technologies over the same period. It is important to design any new fire control system in such a manner that new technologies can be readily incorporated as they become available. The highly modular character of the FC-NET architecture provides numerous locations where new technologies can be inserted into the system with minimal impact on other components in the fire control system.

An architecture that provides easy interchangeability of components allows fire control systems to be readily modified, extended, or reconfigured to better meet current requirements. Individual components of the fire control system can be swapped out to incorporate improved algorithms, more efficient component implementations, or more sophisticated decision aids that enhance the weapon system's ability to engage targets. As commercial technologies mature and are adopted by the Department of Defense, new components that utilize these technologies can replace older components based on more expensive military technologies.

For example, if a system moves from MIL-STD-1553B-based communications to Transmission Control Protocol/Internet Protocol (TCP/IP), the FC-NET architecture ensures that this change only impacts boundary components involved with hardware device communications. Most importantly, component interchangeability supports the plug-and-fight capability of the FCS to host different combinations of weapons at different times. For example, a FCS might be quickly reconfigured by exchanging a non-line-of-sight gun for a complement of Netfires missiles and a remote armed reconnaissance robot. The FC-NET is designed to accommodate distributed fire control systems, where fire control software components physically reside in weapon hardware. The required software components are then automatically incorporated into the fire control system when the weapon hardware is inserted into the weapon system.

An architecture that provides vendor-independence allows fire control systems to be composed of components created

by different vendors. Vendors can work independently to produce fire control components that interoperate with components provided by other vendors or the government, using whatever in-house expertise or methodologies provide their competitive advantage. Since every component developer writes to defined component interfaces, integration costs are reduced and subcontracting becomes an effective means of incorporating best-of-breed technology into a fire control system.

Program Status

An initial version of the FC-NET software architecture has been produced [9]. The FC-NET program is in the first phase of a five-phase 50-month effort that will refine the software architecture through application to four different weapon systems constructed from five different weapons.

Each weapon system will be composed of two or more weapons and will consist of some combination of real and simulated hardware. The weapons selected for these systems will be representative of the types of weapons the FCS expects to mount. Example weapons include the Low Cost Precision Kill Missile, Common Missile, Compact Kinetic Energy Missile, and an Objective Crew Serve Weapon. Each weapon system will feature realistic gunner interaction by utilizing an integrated crew station provided by the Army's Tank and Automotive Research, Development, and Engineering Center.

The first phase of the program culminated in a June 2003 demonstration of a weapon system that operates in a simulated environment. It is expected that the FC-NET reference implementation and fire control foundation classes described earlier will be produced as by-products of implementing the different weapon systems developed during the course of the program. One intent of the demonstration projects was to expose opportunities for improving the current architecture. Early implementation experience has already resulted in many minor changes to component interfaces. More substantial changes to the architecture will be considered at the conclusion of each program phase.

Conclusion

The FC-NET fire control software architecture provides the flexibility needed to support the FCS and other new and evolving weapon systems. This flexibility is achieved through the encapsulation of functionality in well-defined software

components and the isolation of hardware characteristics in PMs. Fire control foundation classes and a reference implementation of the architecture will be developed in conjunction with this program. System integrators can exploit these reusable artifacts to achieve cost and schedule economies when developing fire control systems for new configurations of the FCS.◆

References

1. United States. Defense Advanced Research Projects Agency. "DARPA FCS Overview." Washington: DARPA, 26 Mar. 2002 <www.darpa.mil/tto/programs/fcs.html>.
2. McElwee, J., and J. Gully. "Future Combat Systems: Partnering for Rapid Innovation and Transformation." Boeing Integrated Defense Systems, 4 Apr. 2002 <www.boeing.com/defense-space/ic/fcs/bia/mcelwee.zip>.
3. United States. Joint Force. DoD Dictionary of Military and Associated Terms, Joint Publication 1-02. Washington: Director for Operational Plans and Joint Force Development (J-7), 12 Apr. 2001 (as amended through 15 Oct. 2001) <www.dtic.mil/doctrine/jel/doddict>.
4. Prosiase, J. Programming Windows With MFC. Redmond, WA: Microsoft Press, 1999.
5. IBM Microelectronics Division. "PowerPC 750FX Product Brief." IBM Corporation, Apr. 2002 <www.3.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_750XF_Microprocessor>.
6. Schmidt, D. C., and F. Kuhns. "An Overview of the Real-Time CORBA Specification." IEEE Computer. June 2000 <www.cs.wustl.edu/~schmidt/PDF/orc.pdf>.
7. United States. Joint Chiefs of Staff. Joint Vision 2010. Fort Belvoir, VA: Defense Technical Information Center, July 1996 <www.dtic.mil/jv2010/jv2010.pdf>.
8. United States. Joint Chiefs of Staff. Joint Vision 2020. Fort Belvoir, VA: Defense Technical Information Center, June 2000 <www.dtic.mil/jv2020/jvpub2.htm>.
9. U.S. Army Aviation and Missile Research, Development, and Engineering Center. FC-NET Architecture Description, v. 1.1. U.S. Army AMCOM, June 2002.

About the Authors



Joel Sherrill, Ph.D., is director of Research and Development for OAR Corporation with 15 years experience in the design, development, and fielding of real-time embedded applications in a variety of military, commercial, and research domains. As a principal author and current maintainer of the open-source real-time operating system RTEMS, he has been deeply involved in numerous RTEMS-related efforts including the GNAT/RTEMS validation. Sherrill is a founding member of the Steering Committee for the Free Software Foundation's GNU Compiler Collection.

OAR Corporation
4910-L Corporate Drive
Huntsville, AL 35805
Phone: (256) 722-9985
Fax: (256) 722-0985
E-mail: joel.sherrill@oarcorp.com



Ron O'Guin is executive vice president for OAR Corporation with 25 years experience in the development of real-time operating systems, real-time applications, visual simulations, and weapon systems trainers. As a principal author of the open-source real-time operating system RTEMS, he has been deeply involved in numerous RTEMS-related development efforts. O'Guin has an extensive background in missile system research and is a principal developer of the Fire Control-Node Engagement Technology (FC-NET) Technical Fire Control Architecture. He currently serves as the software manager for the FC-NET STO Program.

OAR Corporation
4910-L Corporate Drive
Huntsville, AL 35805
Phone: (256) 722-9985
Fax: (256) 722-0985
E-mail: ron.oguina@oarcorp.com



Malcolm Morrison, Ph.D., is a senior software engineer for OAR Corporation with 15 years experience as a developer of information and weapon systems, an educator, and consultant. His focus has been on software process management impacts. Morrison has served as a full-time faculty member at the University of Alabama in Huntsville and Salisbury University in Maryland. He is a developer of the Fire Control-Node Engagement Technology Technical Fire Control Architecture.

OAR Corporation
4910-L Corporate Drive
Huntsville, AL 35805
Phone: (256) 842-6937
Fax: (256) 722-0985
E-mail: malcolm.morrison@oarcorp.com



Deborah A. Butler is an electronics engineer with the Aviation and Missile Research Development and Engineering Center with more than 15 years experience in the design, development, and fielding of real-time military embedded applications. As an experienced hardware and software designer, Butler has contributed to the successful design, development, and demonstration of programs such as the Future Missile Technology Integration Program, Long Range Fiber Optic Guided missile, Future Artillery Loiter Concept, Low Cost Precision Kill, and Compact Kinetic Energy Missile. She has an extensive background in missile system research and development and is the program manager for the Fire Control-Node Engagement Technology Science and Technology Program.

U.S. Army Aviation and
Missile Command
ATTN: AMSAM-RD-MG-NC
(Deborah A. Butler)
Redstone Arsenal, AL 35898-5000
Phone: (256) 876-1303
Fax: (256) 842-9476
E-mail: deborah.butler@rdcc.
redstone.army.mil