9th International Command and Control Research and Technology Symposium Coalition Transformation: An Evolution of People, Processes and Technology to Enhance Interoperability

Integrating Battlefield Objects of C4ISR Systems by Using CAPS

Dr. Meng-chyi Harn
(Point of Contact)

Department of Information Management
China Institute of Technology

245, Academic Rd. Sec. 3, Nankang, Taipei,
Taiwan 115, Republic of China

+886-919-297-147 (Tel)
+886-2-2785-2811 (Fax)

harn@cc.chit.edu.tw

Cheng-hang Wang

Department of Computer Science and Engineering
Yuan Ze University

135, Far-East Rd., Chung-Li, Taoyuan,
Taiwan 320, Republic of China

+ 886-2-2222-2137 ext. 8479 (Tel)
+ 886-2-2226-5896 (Fax)

chwang@saturn.yzu.edu.tw

Integrating Battlefield Objects of C4ISR Systems by Using CAPS*

Dr. Meng-chyi Harn

Department of Information Management
China Institute of Technology

245, Academic Rd. Sec. 3, Nankang, Taipei,
Taiwan 115, Republic of China
+886-919-297-147 (Tel)
+886-2-2785-2811 (Fax)

harn@cc.chit.edu.tw

Cheng-hang Wang

Department of Computer Science and Engineering
Yuan Ze University

135, Far-East Rd., Chung-Li, Taoyuan,
Taiwan 320, Republic of China
+ 886-2-2222-2137 ext. 8479 (Tel)
+ 886-2-2226-5896 (Fax)

chwang@saturn.yzu.edu.tw

Abstract

The purpose of this study is to integrate the battlefield objects of a C4ISR system by using a heterogeneous integration tool, called Computer-Aided Prototyping System (CAPS). We explored the whole picture and the prototype of the R. O. C. military C4ISR system and found what the difficulties and the best solutions were. In the physical domain, we specified the battlefield object into five types: military people, weapon systems, navigation systems, platform sensors, and communication links. Then, we modeled the physical battlefield objects as several software objects that had an operator with properties, and the relationship among these battlefield objects as a data stream. The requirements evaluation of a C4ISR system can be carried out by the rapid prototyping method. We have created and integrated 230 large-grain prototypes of C4ISR subsystems, which are going to be developed urgently in Taiwan, R. O. C. We discussed the theorem and practice of battlefield object integration in this paper.

Keywords: C4ISR Systems, Battlefield Object Integration, Software Object Integration, Computer-Aided Prototyping System (CAPS), Software Evolution.

*This research was supported in part of the R.O.C. National Science Council under the grant number 90-2213-E-123-001.

1. Introduction

This study focuses on the theorem and practice on the battlefield object integration of a specified R. O. C. military *Command, Control, Communication, Computer, Intelligence, Surveillance, and Reconnaissance (C4ISR) system* by using a heterogeneous integration tool, called *Computer-Aided Prototyping System (CAPS)*. The C4ISR system is a kind of real-time embedded system that is difficult to be developed without a powerful tool, especially to integrate thousands of battlefield objects including military people, weapon systems, navigation systems, platform sensors, and communication links [Harn *et al*., 2002]. The requirements description of C4ISR systems is always changed in accordance with the national strategic thinking, commander's operational needs, battlefield resources allocations, and so forth. Therefore, it is very hard to firm the user's requirements and specifications during the development process of a command and control system [Luqi, 1992].

In Taiwan, no one has ever conducted similar projects to integrate huge military C4ISR system before. We explored the whole picture and the prototype of the R. O. C. military C4ISR system and to find what the difficulties and the best solutions were. We got together over 200 military officer students working with our group in Army College and National Defense Management College, National Defense University, Taiwan, R. O. C.

When we went through this project, we faced the several same research issues that had been in existence in the process of the general business system integration. The integration of the military C4ISR system is quite different from the business system because the C4ISR system needs the framework that allows operational, system, and technical architecture to be defined in terms of formal specifications that evolve via consensus-based forums [Harn *et al*., 1999b]. In this project, we addressed the issues that focused on the system integration of military applications only. We have created and integrated 230 large-grain prototypes of C4ISR subsystems, which are going to be developed urgently.

The open issues faced are as follows:

- It is not easy to integrate the C4ISR systems without a project leader and a system engineer who have experience in the system integration even though they use a powerful integration tool.
- The battlefield objects are not only too numerous but also too complex to be integrated into an expectable system because the huge amount and complexity of battlefield objects limit the system function.
- It is very hard to obtain a military operation model from the physical domain to the cognitive domain through the knowledge understanding process.
- There is not a reliable and adaptive tool to integrate a huge and complicate system, especially for the real-time embedded system such that the formalization of system integration has to depend on the inadequate tool that provides partial functions.
- The additional issues in the system integration, such as weapon system replacement, real-time data communication, business process reengineering, command and

control process changing, and so on, are elicited when we undertake this project further.

The solutions proposed to the above issues are the rapid prototyping method and the formal model that have been performed well into CAPS. The CAPS is a set of integrated software engineering tools for creating real-time prototyping systems. Such prototypes are useful for requirements analysis, feasibility studies and systematic development of embedded systems [Luqi, 1993].

The primary objective of the CAPS system is to assist military project managers and designers and rapidly evaluate requirements for real-time control software using executable prototypes. Other important objectives include testing and integrating completed subsystems through evolutionary prototyping [Harn *et al*., 1999c]. The CAPS provides a capacity to quickly develop functional prototypes so that feasibility can be verified early in the software development process; furthermore, the code generator of CAPS increases productivity at a very low cost [Luqi, 1997].

Other functionalities include:

- Supporting integration of large complex systems from requirements to maintenance,
- Formulating/Validating requirements via a prototype demonstration with user's feedback, and
- Assessing feasibility of real-time system designs.

The benefits of using evolutionary prototyping [Harn *et al*., 1999b] supported by CAPS include:

- Reducing project failure risks,
- Validating systems that meet the stakeholder needs,
- Lowering reliability and maintenance costs,
- Reducing system integration problems, and
- Improving responsiveness of changing requirements.

2. Battlefield Objects

Many C4ISR Systems in the world are created by different purposes that may be to the military or nonmilitary applications. Whatever the military and nonmilitary C4ISR systems are, thousands of system objects are hard to be managed. On the research view for creating a military C4ISR application, we classified the battlefield objects into five types: *military people*, *weapon systems*, *navigation systems*, *platform sensors*, and *communication links* in the physical domain [Harn *et al*., 2002].

The *battlefield objects* in a C4ISR system specified are modeled as a set $O = P \cup W \cup N \cup S \cup C$ where

- P denotes a set of *military person objects*,
- W denotes a set of *weapon system objects*,
- N denotes a set of *navigation system objects*,

- *S* denotes a set of *platform sensor objects*, and
- *C* denotes a set of *communication link objects*.

The military people using the C4ISR system include different system end users who directly or indirectly receive the intelligence from the different platform sensors of surveillance and reconnaissance, manipulate the equipments of computers for receiving information and making decision, handle the weapon systems according to the direction of navigation systems, command other military people via computers with communication links and control all of the battlefield objects. We can define each battlefield object by a set as follows:

- The *military people* can be denoted a set $P = \{P1, P2, \ldots, Pp\}$, where p is a number of the military person objects that are involved to the C4ISR system and can be classified into many groups;
- The *weapon systems* can be denoted a set $W = \{W1, W2, \ldots, Ww\}$, where w is a number of the weapon system objects that are handled by Army, Navy and Air Force people and can be classified into many types;
- The *navigation systems* can be denoted a set $N = \{N1, N2, \ldots, Nn\}$, where n is a number of the navigation system objects that direct weapon systems via communication links and can be classified into many types;
- The *platform sensors* can be denoted a set $S = \{S1, S2, \ldots, Ss\}$, where s is a number of the platform sensor objects that monitor any battlefield objects via communication links and can be classified into many types; and
- The *communication links* can be denoted a set $C = \{C1, C2, \ldots, Cc\}$, where c is a number of the communication link objects that may assemble the relative equipments with the data transformation function and can be classified into many types.

A specified C4ISR system is a huge combination of thousands of battlefield objects that is very difficult to be firmed. What kinds of battlefield objects would be selected? How to combine these objects? These issues depend on the user's requirements of a specified C4ISR project.

Due to the adjustment of user's requirements coming from national strategic thinking, commander's operation needs, battlefield resources allocations, military organizational architecture and so forth, some of the new battlefield objects would be swap into the specified C4ISR system, some of the original battlefield objects would be swap out of this C4ISR system, and some of the original battlefield objects would be modified. The integration mechanism of battlefield objects of a C4ISR system, which includes *swap_into*, *swap_out_of*, *modified* and *unchanged*, can be shown as Figure 1. The change from the C4ISR system representing the ith version to the C4ISR system representing the $(i+1)$st version can be described in terms of set operations by the following equations:

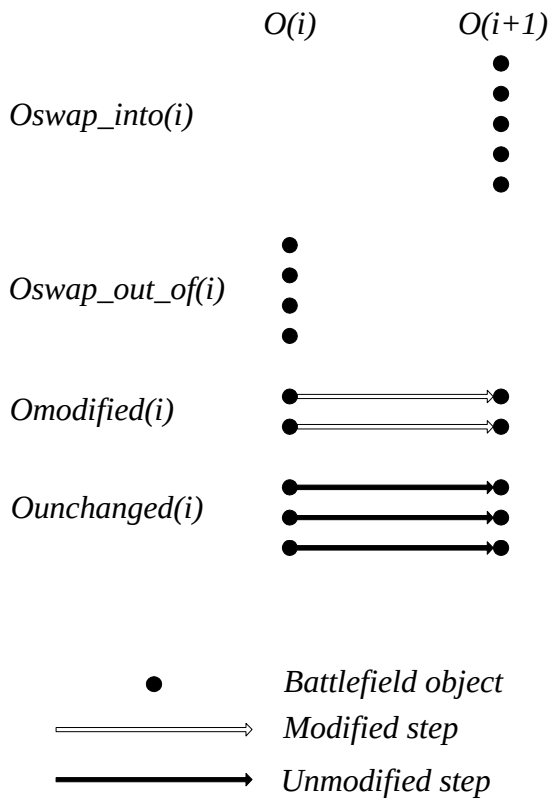

Figure 1: The integration mechanism of battlefield objects of a C4ISR system

$O(i+1) = O(i) + \Delta O(i)$
$O(i+1) = P(i+1) \cup W(i+1) \cup N(i+1) \cup S(i+1) \cup C(i+1)$
$O(i) = P(i) \cup W(i) \cup N(i) \cup S(i) \cup C(i)$
$\Delta O(i) = \Delta P(i) \cup \Delta W(i) \cup \Delta N(i) \cup \Delta S(i) \cup \Delta C(i)$ where

$\Delta O(i) = Oswap\_into(i) - Oswap\_out\_of(i) \cup Omodified(i)$ where

Oswap_into(i): The set of the new battlefield objects of the C4ISR system to be added to *O(i)*.

Oswap_out_of(i): The set of the original battlefield objects of the C4ISR system to be removed from *O(i)*.

Omodified(i): The set of the removed battlefield objects of the C4ISR system to be modified and added to *O(i)*.

$\Delta P(i) = Pswap\_into(i) - Pswap\_out\_of(i) \cup Pmodified(i)$ where

Pswap_into(i): The set of the new military person objects of the C4ISR system to be added to *P(i)*.

Pswap_out_of(i): The set of the original military person objects of the C4ISR system to be removed from *P(i)*.

Pmodified(i): The set of the removed military person objects of the C4ISR system to be modified and added to *P(i)*.

$\Delta W(i) = Wswap\_into(i) - Wswap\_out\_of(i) \cup Wmodified(i)$ where

Wswap_into(i): The set of the new weapon system objects of the C4ISR system to be added to *W(i)*.

Wswap_out_of(i): The set of the original weapon system objects of the C4ISR system to be removed from *W(i)*.

Wmodified(i): The set of the removed weapon system objects of the C4ISR system to be modified and added to *W(i)*.

$\Delta N(i) = Nswap\_into(i) - Nswap\_out\_of(i) \cup Nmodified(i)$ where

Nswap_into(i): The set of the new navigation system objects of the C4ISR system to be added to *N(i)*.

Nswap_out_of(i): The set of the original navigation system objects of the C4ISR system to be removed from *N(i)*.

Nmodified(i): The set of the removed navigation system objects of the C4ISR system to be modified and added to *N(i)*.

$\Delta S(i) = Sswap\_into(i) - Sswap\_out\_of(i) \cup Smodified(i)$ where

Sswap_into(i): The set of the new sensor platform objects of the C4ISR system to be added to *S(i)*.

Sswap_out_of(i): The set of the original sensor platform objects of the C4ISR system to be removed from *S(i)*.

Smodified(i): The set of the removed sensor platform objects of the C4ISR system to be modified and added to *S(i)*.

$\Delta C(i) = Cswap\_into(i) - Cswap\_out\_of(i) \cup Cmodified(i)$ where

Cswap_into(i): The set of the new navigation system objects of the C4ISR system to be added to *C(i)*.

Cswap_out_of(i): The set of the original navigation system objects of the C4ISR system to be removed from *C(i)*.

Cmodified(i): The set of the removed navigation system objects of the C4ISR system to be modified and added to *C(i)*.

Therefore, the change from *P, W, N, S* and *C* representing the *i*th version to *P, W, N, S* and *C* representing the *(i+1)*st version can be described in terms of set operations by the following equations:

$P(i+1) = P(i) + \Delta P(i)$

$W(i+1) = W(i) + \Delta W(i)$

$N(i+1) = N(i) + \Delta N(i)$

$S(i+1) = S(i) + \Delta S(i)$

$C(i+1) = C(i) + \Delta C(i)$

The elements of the above sets of *P(i+1), W(i+1), N(i+1), S(i+1)* and *C(i+1)* are integrated by the super set of *O(i+1)*. The number of the battlefield objects depends on the scale of the specified C4ISR project that can be supervised by the mechanism of the *Version Control and Configuration Management (VCCM)* of CAPS.

3. Software Objects

The battlefield objects in the physical domain can be transferred to as a specification or a program in the C4ISR system development and evolution process [Harn, 1999a]. We can

simulate the physical battlefield object as an *operator* with *properties* and the relationship among the physical battlefield objects as a *data stream*. In the [Luqi, 1989], Luqi explored the general class of objects subject to version control and viewed each type of software object – such as a specification or a program – as a subclass of the general class "*versioned-object*." Each subclass provides additional operations and properties relevant to each kind of software object.

In order to discuss the integrating of C4ISR system changes made to a prototype, we have to introduce the mathematical model of the change process provided by her study group and shown in Figure 2 [Dampier, 1994].

If S is the intended final version of the software system, then each successive iteration of the prototype can be viewed as an element of a sequences $S(i)$, where $lim\ i \rightarrow \infty\ S(i) = S$.
Each prototype $S(i)$ is modeled as a graph $G(i) = (V(i), E(i), C(i))$, where

(1) $V(i)$ is a set of vertices. Each vertex can be an atomic operator or a composite operator modeled as another graph.
(2) $E(i)$ is a set of data streams. Each edge is labeled with the associated variable name. There can be more than one edge between two vertices. There can also be edges from an operator to itself, representing state variable data streams.
(3) $C(i)$ is a set of timing and control constraints imposed on the operators in version i of the prototype.

The prototype designer repeatedly demonstrates versions of the prototype to users, and designs the next version based on user requirements.
The change from the graph representing the ith version of the prototype to the graph representing the $(i+1)$st version can be described in terms of graph operations by the following equations:
$S(i+1) = (V(i+1), E(i+1), C(i+1)) = S(i) + \Delta S(i)$
$\Delta S(i) = (VA(i), VR(i), EA(i), ER(i), CA(i), CR(i))$ where

$V(i+1) - V(i) = VA(i)$: The set of vertices to be added to $S(i)$.
$V(i) - V(i+1) = VR(i)$: The set of vertices to be removed for $S(i)$.
$E(i+1) - E(i) = EA(i)$: The set of edges to be added to $S(i)$.
$E(i) - E(i+1) = ER(i)$: The set of edges to be removed from $S(i)$.
$C(i+1) - C(i) = CA(i)$: The set of timing and control constraints to be added to $S(i)$.
$C(i) - C(i+1) = CR(i)$: The set of timing and control constraints to be removed from $S(i)$.

The + operation above is defined as follows:
$V(i+1) = V(i) \cup VA(i) - VR(i)$
$E(i+1) = E(i) \cup EA(i) - ER(i)$
$C(i+1) = C(i) \cup CA(i) - CR(i)$

Figure 2: A change process model of a software system

Because of the need of the successive iteration process of the prototype, a requirement evolution model that is a kind of a schematic model for the analysis process, was proposed in [Berzins et al., 1997] for computer-aided prototyping. The requirement evolution model was modified from the *Issue-Based Information System (IBIS) model* [Conklin and Begeman, 1988] and considered that the software object via the evolution process is not limited to a specification or a program only. We have to trace the source of the specification and program for obtaining the rationale of evolution.

As a result, the meaning of a software object was extended to the *software evolution objects* that have seven types of *software evolution components*: *criticisms, issues, requirements, specifications, modules, programs* and *optimizations,* and eight types of *software evolution steps*: *software prototype demo, issue analysis, requirements analysis, specification design, module design, software prototype integration, software product demo* and *software product implementation,* for the software evolution process [Harn, 1999a].

The *software evolution objects* in a C4ISR system specified are modeled as a set $X = Y \cup Z$ where

- Y denotes a set of *software evolution components* and
- Z denotes a set of *software evolution steps.*

The *software evolution components* in a C4ISR system specified are modeled as a set $Y = C \cup I \cup R \cup S \cup M \cup P \cup O$ where

- C denotes a set of *criticisms,*
- I denotes a set of *issues,*
- R denotes a set of *requirements,*
- S denotes a set of *specifications,*
- M denotes a set of *modules,*
- P denotes a set of *programs,* and
- O denotes a set of *optimizations.*

The *software evolution steps* in a C4ISR system specified are modeled as a set $Z = C' \cup I' \cup R' \cup S' \cup M' \cup P' \cup O' \cup Q'$ where

- C' denotes a set of *software prototype demo steps,*
- I' denotes a set of *issue analysis steps,*
- R' denotes a set of *requirements analysis steps,*
- S' denotes a set of *specification design steps,*
- M' denotes a set of *module implementation steps,*
- P' denotes a set of *software prototype integration steps,*
- O' denotes a set of *software product demo steps,* and
- Q' denotes a set of *software product implementation steps.*

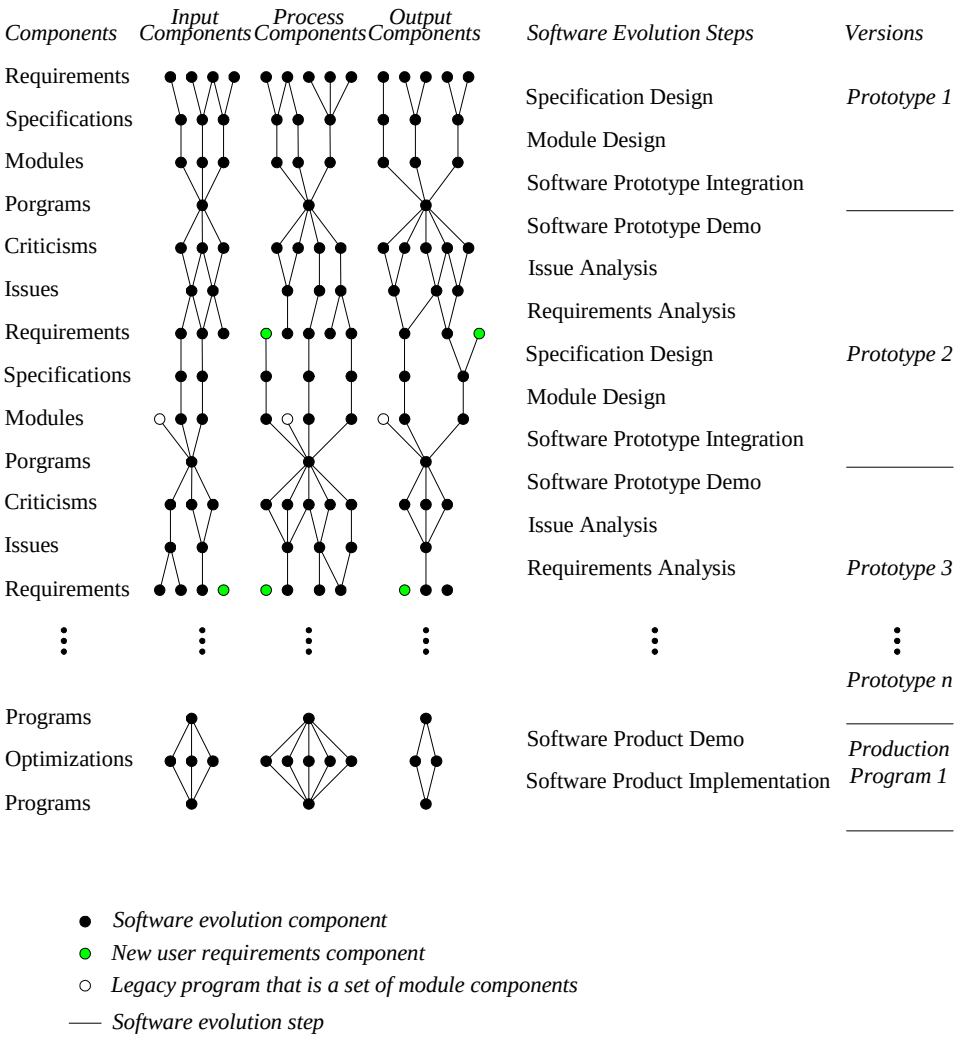

Figure 3: Software evolution steps with their related components

In the C4ISR system development and evolution process, the change from the C4ISR system representing the *i*th version to the C4ISR system representing the *(i+1)*st version can be described in term of set operations by the following equations:

$X(i+1) = X(i) + \Delta X(i)$

$X(i+1) = Y(i+1) \cup Z(i+1) = (C(i+1) \cup I(i+1) \cup R(i+1) \cup S(i+1) \cup M(i+1) \cup P(i+1) \cup O(i+1)) \cup (C'(i+1) \cup I'(i+1) \cup R'(i+1) \cup S'(i+1) \cup M'(i+1) \cup P'(i+1) \cup O'(i+1) \cup Q'(i+1))$

$X(i) = Y(i) \cup Z(i) = (C(i) \cup I(i) \cup R(i) \cup S(i) \cup M(i) \cup P(i) \cup O(i)) \cup (C'(i) \cup I'(i) \cup R'(i) \cup S'(i) \cup M'(i) \cup P'(i) \cup O'(i) \cup Q'(i))$

$\Delta X(i) = \Delta Y(i) \cup \Delta Z(i) = (\Delta C(i) \cup \Delta I(i) \cup \Delta R(i) \cup \Delta S(i) \cup \Delta M(i) \cup \Delta P(i) \cup \Delta O(i)) \cup (\Delta C'(i) \cup \Delta I'(i) \cup \Delta R'(i) \cup \Delta S'(i) \cup \Delta M'(i) \cup \Delta P'(i) \cup \Delta O'(i) \cup \Delta Q'(i))$ where

$\Delta X(i) = Xswap\_into(i) - Xswap\_out\_of(i) \cup Xmodified(i)$ where

Xswap_into(i): The set of the new software evolution objects of the C4ISR system to be added to *X(i)*.

Xswap_into(i): The set of the original software evolution objects of the C4ISR system to be removed from *X(i)*.

Xmodified(i): The set of the removed software evolution objects of the C4ISR system to be modified and added to *X(i)*.

$\Delta Y(i) = Yswap\_into(i) - Yswap\_out\_of(i) \cup Ymodified(i)$ where

Yswap_into(i): The set of the new software evolution components of the C4ISR system to be added to *Y(i)*.

Yswap_out_of(i): The set of the original software evolution components of the C4ISR system to be removed from *Y(i)*.

Yswap_modified(i): The set of the removed software evolution components of the C4ISR system to be modified and added to *Y(i)*.

The definition in details is shown in Appendix A.

$\Delta Z(i) = Zswap\_into(i) - Zswap\_out\_of(i) \cup Zmodified(i)$ where

Zswap_into(i): The set of the new software evolution steps of the C4ISR system to be added to *Z(i)*.

Zswap_out_of(i): The set of the original software evolution steps of the C4ISR system to be removed from *Z(i)*.

Zswap_modified(i): The set of the removed software evolution steps of the C4ISR system to be modified and added to *Z(i)*.

The definition in details is shown in Appendix B.

4. Object Integration of C4ISR Systems

4.1 Project Organization and Preparation

The C4ISR system selected by our project is a *huge-grain* program [Luqi, 1997] involving module composition. We spent almost one year to conduct this project by nine project teams. Each project team had one project leader and 24 project members in charge of 25 *large-grain* programs that included one integrated meta-program and 24 individual programs.

We assigned project team A, whose members came from Army, Navy, and Air Force, to integrate the C4ISR systems of Army, Navy, and Air Force. We assigned project team B to I to create 200 Army C4ISR systems because all of the project members served in the Army. We had one project chairman and 4 project technical instructors to direct the 225 project members and work with them for four hours once a week. Most of the 225 officer students, whose ranks were lieutenant commanders or lieutenant colonels, were operational officers and part of them served in logistics units.

They didn't have too much knowledge about prototyping the C4ISR systems before joining to our project. At the beginning of this project, we gave them several relative classes about the C4ISR systems, for examples: Digitizing Battlefield Management,

Development and Implementation of the C4ISR Systems and Evolution of the C4ISR Systems. The major contents of these classes include the following subjects:

1. Digitizing Battlefield Management:
    - Network Centric Warfare [Alberts *et al.*, 1999]
    - Understanding Information Age Warfare [Alberts *et al.*, 2001]
    - Power to the Edge [Alberts *et al.*, 2003]
    - Information Warfare and Security [Denning, 1999]
2. Development and Implementation of the C4ISR Systems
    - Command and Control Theory
    - Real-time Embedded Systems
    - Requirement Engineering and Rapid Prototyping
    - Computer-Aided Prototyping System
3. Evolution of the C4ISR Systems
    - Relational Hypergraph Model [Harn, 1999a]
    - Version Control and Configuration Management
    - Automated Software Engineering
    - Computer-Aided Software Engineering System [Harn, 1999a]

After three months, we taught them to manipulate the *Heterogeneous Integration System (HIS)* that was a building tool for creating real-time embedded system and was a PC version of CAPS executed in Windows platform. At the same time, they collected the operational requirements of the C4ISR system from user views and discussed the operational architecture with their previous senior officers and colleagues.

4.2 Operational Architecture

We carried out the Army C4ISR system primarily and integrated with the Navy and Air Force C4ISR systems that had already been operated for many years by a meta-system that is a top-level Joint Operational Command and Control System. We have designed interfaces with the glue and wrapper methodology [Luqi, 1997] to integrate different heterogeneous platforms including hardware, software, database and network protocol, to enhance the interoperability among the numerous objects. Through the process of discussion and optimization by each project team, there were nine projects/systems we classified as follows:

- Joint Operational Command and Control System (JOCCS)
- Decision Support System of Operation Area Commander (DSSOAC)
- Battle Command System (BCS)
- Intelligence Surveillance and Reconnaissance System (ISRS)
- Air Defense System (ADS)
- Fire Support and Coordination System (FSCS)
- Disaster Control System (DCS)
- Operational Service System (OSS)
- Personnel Information Integration System (PIIS)

The hierarchical operational architecture is built by the above systems and combined by five types of battlefield objects. Each system is a kind of distributed computer system that includes the distribution of the related software and data, and integrates the other battlefield objects via interfaces. The distributed computer system in the above systems connects to the relative upper/lower level and large/small-scale distributed computers that are embedded in their command and control systems.

In our project, the battlefield object was simulated as a prototype program with a series of relative evolutionary objects. We obtained the requirements of the operational architecture by rapid prototyping method and automated software engineering technology. CAPS can spread out the whole picture and automatically generate the *Prototype System Description Language (PSDL)* codes including the operators and data streams with properties.

4.3 An example: Fire Support and Coordination System (FSCS)

FSCS is one of the Army C4ISR systems in our project. Table 1 is provided, briefly and partially, to illustrate the battlefield objects of FSCS configuration management in the physical domain.

Table 1: The Battlefield Objects of FSCS

Types\Objects	The Battlefield Objects of FSCS
Military People	*Fire-coordinating Officer, Fire-supporting Director, Artilleryman, Navy Liaison Officer, Air Force Liaison Officer, Chemistry Officer, Communication and Information Officer, Object-obtaining Officer, Object-analyzing Officer, Air Force Support Director, J2 Air Operational Officer, Air Control Director, Army Air Force Officer*
Weapon Systems	*Field Artillery, Rocket Forces, Armor Forces, Gunnery, Fighter-bomber, Hunting Helicopter*
Navigation Systems	*CNS, Satellite Navigation GPS*
Platform Sensors	*Battlefield Surveillance and Reconnaissance Radar, Coast Acquisition Radar, Searching and Ranging Radar, Weather Radar, ATHS*
Communication Links	*High-rate Receive Links, High-rate Transmit Links, Low-rate Receive and Transmit Links, Tadiran Communications*

FSCS is functionally decomposed into six subsystems, such as object-obtaining subsystems, fire-supporting subsystem, fire-coordinating subsystem, air-supporting subsystem, air-controlling subsystem and fire-coordinating command and control subsystem, which are incorporated by a large-scale computer shown as Figure 4.

In the graphic user interface of HIS, each subsystem of FSCS is considered as a software object that is a set of lower-level software objects. An example of the fire-supporting

subsystem is shown as Figure 5. These software objects model the battlefield objects of the physical domain. The software object's enlargement and refinement of a C4ISR system depend on the optimization mechanism of operational requirements analysis including the consideration of economic, technical, and legal feasibility.

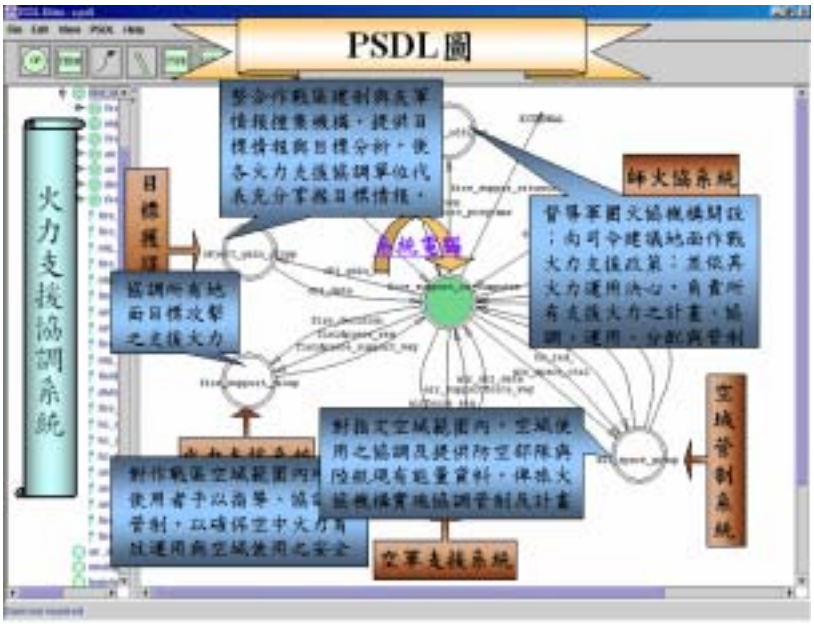

Figure 4: Top-level of FSCS

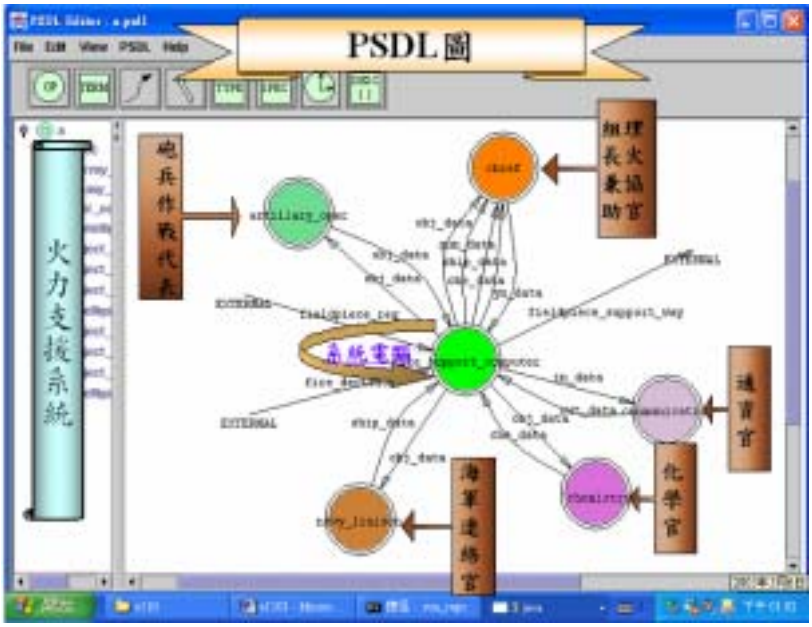

Figure 5: Lower-level of FSCS

The software object in HIS is represented as an operator with properties and connected to another software object by a data stream with properties. From the huge/large-grain view of designing a C4ISR system, the operator can be regarded as a super-system because the operator is an accumulation of its subsystems and the data stream can be specified as a set of data with an array type because the amount of transmission data is numerous. From the small-grain view of designing a C4ISR system, the operator is an atomic component that can be implemented and executed but cannot be decomposed, and the data stream can be specified as simple data with an abstract dada type.

Finishing the requirements analysis on the graphic user interface panel of HIS, we saved the whole picture to the disk; at the same time, the PSDL codes are generated automatically. Through the software evolution steps, the software object of the C4ISR systems evolves by CAPS. Without using CAPS, it is very hard to manage the software object with its related evolutionary steps and components. The CAPS has a very powerful mechanism of configuration management and version control to handle software objects in different system layer vertically and trace software objects in different versions horizontally. This mechanism was conducted in the *Relational Hypergraph model* [Harn, 1999a].

5. Lessons Learned

We have already studied several building methodologies and tools for integrating battlefield objects of the C4ISR system, such as *IDEF*, *UML*, *C4ISR Architecture Framework 2.0*, *System Architect V9.1*, *ARIS*, *Rhapsody*, and so on. We found that only CAPS could match our requirement goals of selecting a suitable tool because we considered the following key reasons to integrate the battlefield objects:

- We specified the development methodology – rapid prototyping;
- We wanted to develop a C4ISR system that was real-time embedded;
- We needed a building tool that had the function of project management, risk assessment, software automation, software component reuse, version control and configuration management;
- We had to quickly obtain the executable code that could simulate all kinds of battlefield objects;
- We had to enhance the software productivity and quality, and reduce the development and maintenance cost;
- This tool had to be learned and handled easily;
- This tool had to be manipulated on the PCs; and
- Our development environment was heterogeneous.

Before we started our project, we were afraid that our officer students were lacks of backgrounds and experiences about integrating the battlefield objects of the C4ISR system. We were supposed to introduce the Petri-Net for them quickly to understand the process of describing the requirements of the C4ISR system; however, we were very surprised that they easily accepted the graphic user interface of the HIS. They elicited

their ideas via a simple knowledge representation that was a combination graph of operators and data streams so they quit to learning the Petri-Net. We proved that the prototyping was the best method to integrate the C4ISR system.

As mentioned earlier, several battlefield objects could evolve to a new version based on the object's *swap_into, swap_out_of, modified* and *unchanged* mechanism. Because we considered the battlefield objects as simulators that were atomic software components generated by CAPS, after the simulators were executed in the CAPS, we would evaluate the system performance and change the battlefield objects of the C4ISR system. The process of the system evaluation and the change of a battlefield object would be executed several times until the system performance reached the users' requirements.

For example, the *Maximum Execution Time (MET)* of a weapon system object had to adjust to 2 seconds instead of 3 seconds for the reason that the capacity of this kind of the weapon system was enhanced. If we would like to improve the efficiency of the C4ISR system but the capacity of the battlefield object of the C4ISR system could not match user's current needs, the changed battlefield objects of C4ISR system could be simulated as a software component with its specific parameters by using CAPS. Therefore, the battlefield objects could be carried out in the physical domain, and the related software objects could be validated in the laboratory.

We were anxious about project failure, so we requested each officer student to integrate a small number of objects of a nonmilitary C4ISR system in advance. The officer student specified the system domain for practice only. The application of nonmilitary C4ISR system was selected by the principle that they had to be very familiar with their knowledge domain. Finally, we obtained the 230 small and independent nonmilitary C4ISR systems and integrated different domain objects, for examples, ID Checking System, Smart Cell Phone System, Intelligent Refrigerator Management System, Water Quality Monitor System, Pet Feed System, Automated Night Stool System, Automatic Slowdown System for Vehicles, Sound Control TV System, and so on. These nonmilitary systems cover the characteristics of the C4ISR system partially or completely.

References

[Conklin and Begeman, 1988] "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *ACM Transactions on Office Information System,* Vol. 6, October 1988, pp. 303-331.

[Denning, 1999] D. E. Denning, *Information Warfare and Security*, Addison-Wesley, July, 1999.

[Harn, 1999a] M. Harn, "Computer-Aided Software Evolution Based on Inferred Dependencies," *Proceedings of Conference on Advanced Information Systems Engineering: 6th Doctoral Consortium,* Heidelberg, Germany, June 14-15, 1999, pp. 116-127.

[Harn *et al*., 1999b] M. Harn, V. Berzins, Luqi, and W. Kemple, "Evolution of C4I Systems," *Proceedings of 1999 Command and Control Research and Technology Symposium*, United States Naval War College, Newport, Rhode Island, June 29 - July 1, 1999, pp.1361-1380.

[Harn *et al*., 1999c] M. Harn, V. Berzins, and Luqi, "Software Evolution Process via a Relational Hypergraph Model," *Proceedings of IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems*, Tokyo, Japan, October 5-8, 1999, pp. 599-604.

[Harn *et al*., 2002] M. Harn, S. Hsu, V. Berzins, and Luqi, "Battlefield Object Control via Internet Architecture," *Proceedings of 2002 Command and Control Research and Technology Symposium*, United States Naval Postgraduate School, Monterey, California, June 11 - June 13, 2002.

[Luqi, 1989] Luqi, "Software Evolution Through Rapid Prototyping," *IEEE Computer*, May 1989, pp. 13-25.

[Luqi,1992] Luqi, "Computer-Aided Prototyping for a Command-And-Control System Using CAPS," *IEEE Software*, January 1992, pp. 56-67.

[Luqi,1993] Luqi, "How to Use Prototyping for Requirements Engineering," *Proceedings of IEEE/ACM Symposium on Requirements Engineering*, San Diego, CA, January 1993, p.229.

[Luqi, 1997] Luqi, "Formal Methods Promises and Problems," *IEEE Software*, January 1997, pp. 73-85.

Appendix A

Yswap_into(i) = *Cswap_into(i)* ∪ *Iswap_into(i)* ∪ *Rswap_into(i)* ∪ *Sswap_into(i)* ∪ *Mswap_into(i)* ∪ *Pswap_into(i)* ∪ *Oswap_into(i)* where

Cswap_into(i): The set of the new criticism components of the C4ISR system to be added to *C(i)*.

Iswap_into(i): The set of the new issue components of the C4ISR system to be added to *I(i)*.

Rswap_into(i): The set of the new requirements components of the C4ISR system to be added to *R(i)*.

Sswap_into(i): The set of the new specification components of the C4ISR system to be added to *S(i)*.

Mswap_into(i): The set of the new module components of the C4ISR system to be added to *M(i)*.

Pswap_into(i): The set of the new program components of the C4ISR system to be added to *P(i)*.

Oswap_into(i): The set of the new optimization components of the C4ISR system to be added to *O(i)*.

Yswap_out_of(i) = *Cswap_out_of(i)* ∪ *Iswap_out_of(i)* ∪ *Rswap_out_of(i)* ∪ *Sswap_ out_of(i)* ∪ *Mswap_out_of(i)* ∪ *Pswap_out_of(i)* ∪*Oswap_out_of(i)* where

Cswap_out_of(i): The set of the original criticism components of the C4ISR system to be removed from *C(i)*.

Iswap_out_of(i): The set of the original issue components of the C4ISR system to be removed from *I(i)*.

Rswap_out_of(i): The set of the original requirements components of the C4ISR system to be removed from *R(i)*.

Sswap_out_of(i): The set of the original specification components of the C4ISR system to be removed from *S(i)*.

Mswap_out_of(i): The set of the original module components of the C4ISR system to be removed from *M(i)*.

Pswap_out_of(i): The set of the original program components of the C4ISR system to be removed from *P(i)*.

Oswap_out_of(i): The set of the original optimization components of the C4ISR system to be removed from *O(i)*.

Yswap_modified(i) = *Cswap_modified(i)* ∪ *Iswap_modified(i)* ∪ *Rswap_modified(i)* ∪ *Sswap_ modified(i)* ∪ *Mswap_modified(i)* ∪*Pswap_modified(i)* ∪*Oswap_modified(i)* where

Cswap_modified(i): The set of the removed criticism components of the C4ISR system to be modified and added to *C(i)*.

Iswap_modified(i): The set of the removed issue components of the C4ISR system to be modified and added to *I(i)*.

Rswap_modified(i): The set of the removed requirements components of the C4ISR system to be modified and added to *R(i)*.

Sswap_modified(i): The set of the removed specification components of the C4ISR system to be modified and added to *S(i)*.

Mswap_modified(i): The set of the removed module components of the C4ISR system to be modified and added to *M(i)*.

Pswap_modified(i): The set of the removed program components of the C4ISR system to be modified and added to *P(i)*.

Oswap_modified(i): The set of the removed optimization components of the C4ISR system to be modified and added to *O(i)*.

Appendix B

Zswap_into(i) = *C′swap_into(i)* ∪ *I′swap_into(i)* ∪ *R′swap_into(i)* ∪ *S′swap_into(i)* ∪ *M′swap_into(i)* ∪ *P′swap_into(i)* ∪ *O′swap_into(i)* ∪ *Q′swap_into(i)* where

C′swap_into(i): The set of the new software prototype demo step of the C4ISR system to be added to *C′(i)*.

I′swap_into(i): The set of the new issue analysis step of the C4ISR system to be added to *I′(i)*.

R′swap_into(i): The set of the new requirements analysis step of the C4ISR system to be added to *R′(i)*.

S′swap_into(i): The set of the new specification design step of the C4ISR system to be added to *S′(i)*.

M′swap_into(i): The set of the new module implementation step of the C4ISR system to be added to *M′(i)*.

P′swap_into(i): The set of the new program integration step of the C4ISR system to be added to *P′(i)*.

O′swap_into(i): The set of the new software product demo step of the C4ISR system to be added to *O′(i)*.

Q′swap_into(i): The set of the new software product implementaton step of the C4ISR system to be added to *Q′(i)*.

Zswap_out_of(i) = *C′swap_out_of(i)* ∪ *I′swap_out_of(i)* ∪ *R′swap_out_of(i)* ∪ *S′swap_ out_of(i)* ∪ *M′swap_out_of(i)* ∪ *P′swap_out_of(i)* ∪ *O′swap_out_of(i)* ∪ *Q′swap_out_of(i)* where

C′swap_out_of(i): The set of the original software prototype demo step of the C4ISR system to be removed from *C′(i)*.

I′swap_out_of(i): The set of the original issue analysis step of the C4ISR system to be removed from *I′(i)*.

R′swap_out_of(i): The set of the original requirements analysis step of the C4ISR system to be removed from *R′(i)*.

S′swap_out_of(i): The set of the original specification design step of the C4ISR system to be removed from *S′(i)*.

M′swap_out_of(i): The set of the original module implementation step of the C4ISR system to be removed from *M′(i)*.

P′swap_out_of(i): The set of the original program integration step of the C4ISR system to be removed from *P′(i)*.

O′swap_out_of(i): The set of the original software product demo step of the C4ISR system to be removed from *O′(i)*.

Q′swap_out_of(i): The set of the original software product implementation step of the C4ISR system to be removed from *Q′(i)*.

Zswap_modified(i) = *C′swap_modified(i)* ∪ *I′swap_modified(i)* ∪ *R′swap_modified(i)* ∪ *S′swap_ modified(i)* ∪ *M′swap_modified(i)* ∪ *P′swap_modified(i)* ∪ *O′swap_modified(i)* ∪ *Q′swap_modified(i)* where

C′swap_modified(i): The set of the removed software prototype demo step of the C4ISR system to be modified and added to *C′(i)*.

I′swap_modified(i): The set of the removed issue analysis step of the C4ISR system to be modified and added to *I′(i)*.

R′swap_modified(i): The set of the removed requirements analysis step of the C4ISR system to be modified and added to *R′(i)*.

S′swap_modified(i): The set of the removed specification design step of the C4ISR system to be modified and added to *S′(i)*.

M′swap_modified(i): The set of the removed module implementation step of the C4ISR system to be modified and added to *M′(i)*.

P′swap_modified(i): The set of the removed program integration step of the C4ISR system to be modified and added to *P′(i)*.

O′swap_modified(i): The set of the removed software product demo step of the C4ISR system to be modified and added to *O′(i)*.

Q′swap_modified(i): The set of the removed software product implementation step of the C4ISR system to be modified and added to *Q′(i)*.