JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 26 No. 1, pp. 147-154, January 2021
https://doi.org/10.9708/jksci.2021.26.01.147

A Study of the Standard Interface Architecture of Naval Combat Management System

Chi-Sun Baek*, Jin-Hyang Ahn*

*Engineer, SW Team(Naval), Hanwha Systems Co., Ltd., Gumi, Korea
*Engineer, SW Team(Naval), Hanwha Systems Co., Ltd., Gumi, Korea

[Abstract]

Naval Combat Management System(a.k.a. CMS) is the core combat power of ROK Navy. CMS which has been localized since 1993 has been developed in various categories. However, in the characteristic of defense industry, CMS software has been rarely developed technically and structurally while the environment of computing system has been developed dramatically. A need for a new paradigm of CMS software development was raised. This paper suggests Naval Shield Component Platform(NSCP) as a standard interface architecture of CMS based on SOLID of OOP which is an advanced programming paradigm and introduce its functionality and feature. We expect NSCP's higher reusability, concurrency and maintainability in CMS software development. As a future work, we are going to apply NSCP to the next CMS software development project and evaluate quantitative, qualitative method.

▸ **Key words**: Combat Management System, Object-Oriented Programming, Component Platform, Standard Interface Architecture, Interface Control Unit

[요 약]

해군 전투력의 핵심이라고 할 수 있는 함정전투체계(통칭 CMS)는 1993년 국산화에 성공한 이후 다방면에서 꾸준한 발전을 이루어왔다. 반면 CMS 소프트웨어는 컴퓨팅 시스템 환경이 비약적으로 발전해 왔음에도 불구하고 방위산업의 구조적인 특성 때문에 기술 및 구조적 발전이 미비했다. 따라서 CMS 소프트웨어 개발의 새로운 패러다임의 필요성이 대두되었다. 본 논문에서는 진보된 프로그래밍 패러다임인 객체지향 프로그래밍과 그 설계 원칙인 SOLID를 준수하는 함정전투체계 표준 연동 아키텍처로서 Naval Shield Component Platform(NSCP)를 제시하고 각 컴포넌트들의 구조와 기능 및 특징을 소개한다. 더 나아가 이를 통해 CMS 소프트웨어 개발에 있어 높은 재사용성, 동시성 그리고 유지보수성을 기대한다. 향후연구로 NSCP를 차기 CMS 소프트웨어 개발 프로젝트에 시범 적용하고 정량적, 정성적 평가를 통해 NSCP의 효용성을 평가할 예정이다.

▸ **주제어**: 전투체계, 객체지향 프로그래밍, 컴포넌트 플랫폼, 표준 연동 아키텍처, 연동단

• First Author: Chi-Sun Baek, Co-author: Jin-Hyang Ahn, Corresponding Author: Chi-Sun Baek
*Chi-Sun Baek (cs214.baek@hanwha.com), SW Team(Naval), Hanwha Systems Co., Ltd.
*Jin-Hyang Ahn (jinh.ahn@hanwha.com), SW Team(Naval), Hanwha Systems Co., Ltd.
• Received: 2020. 12. 03, Revised: 2021. 01. 05, Accepted: 2021. 01. 13.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

해군 전투력의 핵심이라고 할 수 있는 함정전투체계는 1993년 첫 100% 국산화에 성공한 이후 올해로 약 27년의 기간 동안 많은 발전을 이루어왔다. 반면 CMS의 두뇌라고 할 수 있는 CMS 소프트웨어는 CPU, OS등 구동환경이 비약적으로 발전했음에도 불구하고 초기의 오래된 구조를 벗어나지 못했다. 방위산업의 구조적인 특성과 가격경쟁력을 높이기 위해 원가절감을 추구해야하는 기업의 목표가 서로 맞물려왔기 때문이다[1,2]. 기업입장에서는 재사용이 가능하고 기술 및 구조적 특징이 겉으로 드러나지 않는 소프트웨어의 투자에는 소극적이었고, 소프트웨어 개발의 핵심인 프로그래밍 패러다임은 과거에서 벗어날 수 없었다[3]. 그로인해 아직도 대부분의 CMS 소프트웨어는 무기체계 소프트웨어 간 형상이 상이하여 관리 애로 및 체계통합 구현에 제한적이고, 공통된 기능에 대한 개별구현으로 재사용/재활용 저하 및 구현 리스크가 클 뿐만 아니라, 유연성 및 확장성이 낮다. 또한 소프트웨어 모듈 내 기능 간 구분이 확실하지 않아 높은 상호의존성으로 기능변경에 따른 코드 수정소요가 크다. 종래의 CMS 소프트웨어는 여러 측면에서 비효율적이라고 할 수 있기 때문에 유지보수 및 구조개선 뿐만 아니라 신규개발에도 높은 효율을 낼 수 있는 새로운 패러다임이 필요하다고 할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 배경지식인 프로그래밍 패러다임과 객체 지향 프로그래밍의 다섯 가지 기본 원칙(SOLID) 및 관련연구에 대해 소개한다. 3장에서는 함정전투체계 표준 연동 아키텍처로서 NSCP를 제시하고 각 컴포넌트들의 상세구조 및 체계동작에 대해 설명한다. 4장에서는 기존 소프트웨어와 NSCP기반 소프트웨어를 비교하고 평가한다. 마지막으로 5장에서 결론 및 향후연구방향을 제시한다.

II. Preliminaries

1. Background

1.1 함정전투체계

함정전투체계는 인간의 두뇌와 같은 역할을 수행한다. 함정에 탑재된 레이더, 소나와 같은 센서, 함포, 유도탄과 같은 무장과 기타 장비들을 통합해 작전에 필요한 모든 정보를 종합적으로 수집, 전술상황평가·지휘결심·위협평가·무기할당·교전 등의 기능을 효율적으로 수행할 수 있도록 지원하는 일련의 자동화된 무기체계를 함정 전투체계라고 한다. 현재 함정 전투체계는 Fig. 1과 같이 지휘무장통제체계(Combat Fire Command System, CFCS), 센서, 무장 및 데이터링크로 구성되며 함정의 임무와 특성에 따라 다양한 구성이 가능하다. 이 가운데에서 다양한 체계를 통합해 최대의 성능을 발휘할 수 있도록 하는 핵심적인 역할은 CFCS가 수행한다. CFCS는 데이터를 처리하는 단일기판컴퓨터(Single Board Computer, SBC)가 내장된 정보처리장치(Information Processing Node, IPN), 운용자의 명령을 입력 받고 전술상황을 전시하는 다기능콘솔(Multi Function Console, MFC), 센서 및 무장을 연동시키는 연동단(Interface Control Unit, ICU), 연동되는 체계 상호 간 데이터를 전송하는 전투체계 통합 네트워크, 레이더비디오 및 TV/IR비디오를 분배·전송하는 영상분배장치(Radar Tv Video Distributor Unit, RTVDU) 그리고 각종 전시기 등으로 구성된다[4].

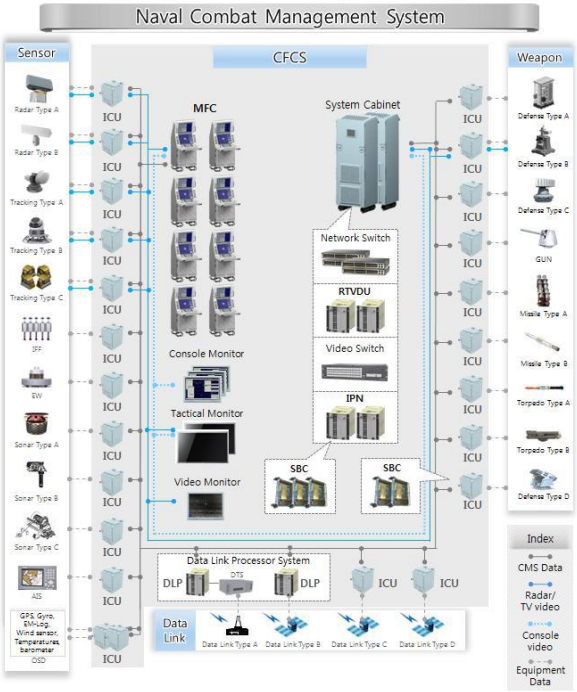


Fig. 1. Naval Combat Management System

1.2 프로그래밍 패러다임

프로그래밍 패러다임에는 여러 종류가 있지만 본 논문에서는 CMS 소프트웨어의 과거 구조인 절차적 프로그래밍과 새로운 구조적 지향점인 객체지향 프로그래밍에 대해 간단히 소개한다.

1.2.1 절차적 프로그래밍

절차적 프로그래밍은 절차지향 프로그래밍 혹은 절차지향적 프로그래밍이라고도 불리는 프로그래밍 패러다임의

일종으로서, 때때로 명령형 프로그래밍과 동의어로 쓰이기도 하지만, 프로시저 호출의 개념을 바탕으로 하고 있는 프로그래밍 패러다임을 의미하기도 한다. 프로시저는 간단히 말하여 수행되어야 할 연속적인 계산 과정을 포함하고 있다. 프로그램의 아무 위치에서나 프로시저를 호출할 수 있는데, 다른 프로시저에서도 호출 가능하고 심지어는 자기 자신에서도 호출 가능하다. 그러나 이 방식은 알고리즘이 복잡해질수록 순서도로 나타내는 것이 불가능할 정도로 꼬인 "스파게티 코드"를 만들게 된다[5,6]. 이렇게 꼬여버린 코드는 다른 사람이 보고 이해하는 것이 거의 불가능할 뿐더러 심지어는 작성한 본인조차도 유지보수에 어려움을 겪게 된다. 명령어의 양이 많아지는 것은 기본이고, 특정 코드 부분은 어디에 사용되는 코드고 해당 코드 부분은 어디까지 이어지는지의 흐름을 파악하기도 힘들어지며, 중복 코드 대처도 매우 어렵다.

1.2.2 객체지향 프로그래밍

객체 지향 프로그래밍은 컴퓨터 프로그래밍의 패러다임 중 하나이다. 객체 지향 프로그래밍은 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하고자 하는 것이다. 각각의 객체는 메시지를 주고받고, 데이터를 처리할 수 있다[7].

객체 지향 프로그래밍은 프로그램을 유연하고 변경이 용이하게 만들기 때문에 대규모 소프트웨어 개발에 많이 사용된다. 또한 프로그래밍을 더 배우기 쉽게 하고 소프트웨어 개발과 보수를 간편하게 하며, 보다 직관적인 코드 분석을 가능하게 하는 장점을 갖고 있다. 하지만 아무리 객체지향 프로그래밍을 적용했다 한들 구조적 설계가 부실하면 코드에서 더 심오한 문제를 일으킬 가능성이 있다. 이를 코드 스멜[8,9]이라고 한다.

1.2.2.1 SOLID

컴퓨터 프로그래밍에서 SOLID란 Robert C. Martin이 2000년대 초반에 명명한 객체 지향 프로그래밍 및 설계의 다섯 가지 기본 원칙[10,11]이다. 프로그래머가 시간이 지나도 유지 보수와 확장이 쉬운 시스템을 만들고자 할 때 이 원칙들을 함께 적용할 수 있다. SOLID 원칙들은 소프트웨어 작업에서 프로그래머가 소스 코드가 읽기 쉽고 확장하기 쉽게 될 때까지 소프트웨어 소스 코드를 리팩터링하여 코드 스멜을 제거하기 위해 적용할 수 있는 지침이다. 이 원칙들은 애자일(Agile) 소프트웨어 개발과 적응적 소프트웨어 개발의 전반적 전략의 일부이다.

3장에서 제시하는 NSCP의 클래스들은 SOLID 원칙하에 설계되었다.

◎ **SRP(Single Responsibility Principle)**

단일책임 원칙 : 작성된 클래스는 하나의 기능만 가지며 클래스가 제공하는 모든 서비스는 그 하나의 책임을 수행하는 데 집중되어 있어야 한다.

◎ **OCP(Open Closed Principle)**

개방폐쇄 원칙 : 소프트웨어의 구성요소(컴포넌트, 클래스, 모듈, 함수)는 확장에는 열려있고, 변경에는 닫혀있어야 한다.

◎ **LSP(Liskov Substitution Principle)**

리스코브 치환 원칙 : 프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다.

◎ **ISP(Interface Segregation Principle)**

인터페이스 분리 원칙 : 한 클래스는 자신이 사용하지 않는 인터페이스는 구현하지 말아야 한다.

◎ **DIP(Dependency Inversion Principle)**

의존관계 역전 원칙 : 추상화에 의존해야지, 구체화에 의존하면 안 된다.

2. Related works

함정전투체계 소프트웨어의 재사용성, 동시성, 유지보수성 개선에 대한 연구는 지금까지 수차례 진행되었지만 체계 소프트웨어 전반에 적용할 수 있는 설계를 제시한 사례는 없었다.

2.1 함기준센서(OSD) 연동단 표준화

함기준센서 연동단 표준화에 대한 연구[12]가 진행되었으나, 표준화 대상이 OSD에 국한되어 있어 동시성을 만족하지 못하고 모듈 내 객체지향 설계가 고려되지 않아 환경 변화에 따른 코드변경 폭이 컸다.

2.2 애자일 개발 방법론 적용

전투체계 분야의 애자일 개발 방법론 적용에 대한 연구[13]가 있었으나 전통적인 개발 방법론인 폭포수 모델을 기반으로 PDR, CDR등의 설계검토회의를 필수로 요구하는 방위산업 구조 상 애자일 방법론을 전투체계 개발 전반에 적용하기 어렵다는 것을 해당 연구에서 언급하고 있으

며, 결과적으로 해당 연구는 이미 개발 완료된 전투체계에 한정적으로 적용을 고려하는 것에 그쳤다.

2.3 디자인패턴을 이용한 함정전투체계 표준화 설계

디자인패턴을 이용한 함정전투체계 소프트웨어 설계에 대한 연구[14,15]가 진행되었다. 객체지향 기반의 설계로 재사용성, 확장성, 유지보수성을 만족하지만 전투체계 기능의 일부분인 교전(Warfare)만을 한정적으로 고려하고 있으며, 전투체계 소프트웨어 전반에 적용가능 한 공용설계를 제시하지 못한다는 점에서 함정전투체계 소프트웨어 표준화 설계라고 보기에는 다소 무리가 따른다.

III. Naval Shield Component Platform

3장에서는 함정전투체계 표준 연동 소프트웨어 아키텍처로서 NSCP를 제시한다.

1. NSCP 컴포넌트 상세구조

Fig. 2는 NSCP의 구성을 개략적으로 보여준다.

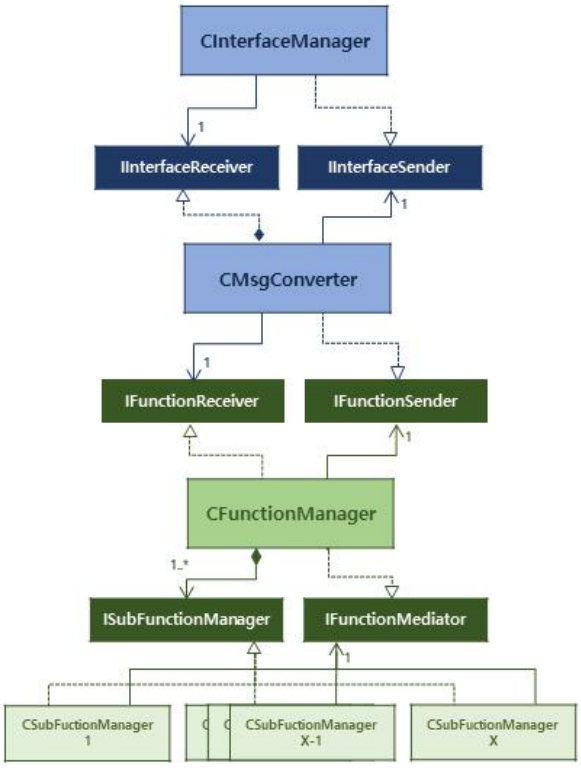


Fig. 2. Naval Shield Component Platform

NSCP는 크게 4개의 컴포넌트로 구성되어 있으며 각 컴포넌트는 인터페이스 클래스를 통해 연결되어 있다. 각 컴포넌트의 명칭과 주요기능은 Table 1에 표현한다.

Table 1. NSCP Component

Name	Major Function
Interface Manager	External Communication
Message Converter	Data Converting
Function Manager	Data Classification & Distribution
Subfunction Manager	Data Processing

1.1. Interface Manager

Fig. 3와 Fig. 4는 Interface Manager 컴포넌트의 추상적 클래스 구성을 나타낸다.

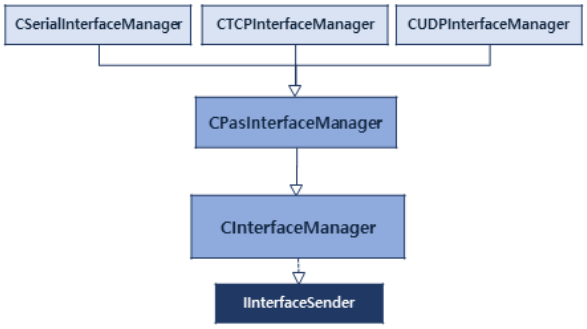


Fig. 3. Interface Manager - GFE

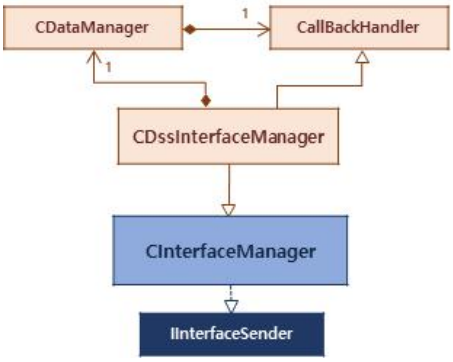


Fig. 4. Interface Manager - CSDB

1.1.1 주요기능

외부통신 어댑터 역할로서 통신 미들웨어 Application Programming Interface(API)를 호출하여 모듈 외부와 통신(송신 및 수신)을 수행한다.

1.1.2 특징

역할 상 NSCP 외부와 통신을 수행하기 때문에 외부와 연결된 통신 방식에 따라 다수의 콘크리트 클래스가 존재할 수 있으며, 또한 동시에 다수의 클래스 인스턴스를 가질 수 있도록 설계되어 있다.

예를 들어 전투체계 연동단은 일반적으로 관급장비(Government Furnished Equipment, 이하 GFE)와의 통신과 전투체계 데이터 버스(Combat System Data Bus, 이하 CSDB)와의 통신을 모두 수행하여야 하므로 Fig. 2와 같은 GFE용 Interface Manager 인스턴스와 Fig. 3와 같은 CSDB용 Interface Manager 인스턴스를 모두 가진다.

1.1.3 세부구조

◎ **컴포넌트 간 통신 인터페이스**

- IInterfaceSender : Message Converter 컴포넌트

◎ **클래스관계**

- CPasInterfaceManager : GFE 외부통신
- CDssInterfaceManager : CSDB 외부통신

1.1.4 연결관계

◎ **상위연결**

- GFE, CSDB 외부 모듈

◎ **하위연결**

- Message Converter 컴포넌트

1.2. Message Converter

Fig. 5는 Message Converter 컴포넌트의 추상적 클래스 구성을 나타낸다.

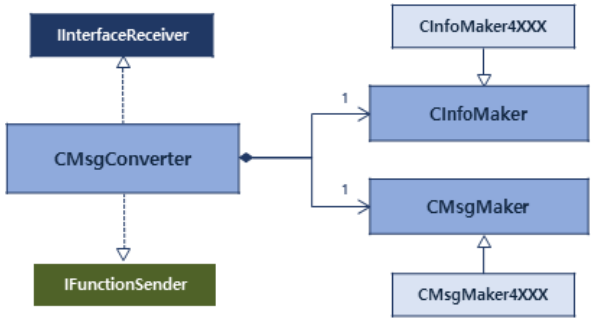


Fig. 5. Message Converter

1.2.1 주요기능

양방향 데이터변환 역할로서 상위로부터 수신한 데이터로부터 의존성을 제거하고 각 컴포넌트에서 관리하는 NSCP 규격의 정보로 변환하거나 하위로부터 송신할 데이터를 외부 규격의 데이터로 변환하는 역할을 수행한다.

1.2.2 특징

양방향 데이터변환을 위해 송신 변환기(Msg Maker) 클래스, 수신 변환기(Info Maker) 클래스 하나씩을 멤버로 가진다. 모듈 역할에 따라 송/수신 메시지가 달라지고 메시지 구성에 따라 변환기의 변환로직이 달라져야 하므로 변환기 클래스의 일부는 모듈 개발자의 개별구현이 필요하다.

1.2.3 세부구조

◎ **컴포넌트 간 통신 인터페이스**

- IInterfaceReceiver : Interface Manager 컴포넌트
- IFunctionSender : Function Manager 컴포넌트

◎ **클래스관계**

- CInfoMaker : 상위 수신 데이터변환
- CMsgMaker : 하위 송신 데이터변환

1.2.4 연결관계

◎ **상위연결**

- Interface Manager 컴포넌트

◎ **하위연결**

- Function Manager 컴포넌트

1.3. Function Manager

Fig. 6는 Function Manager 컴포넌트의 클래스 구성을 나타낸다.

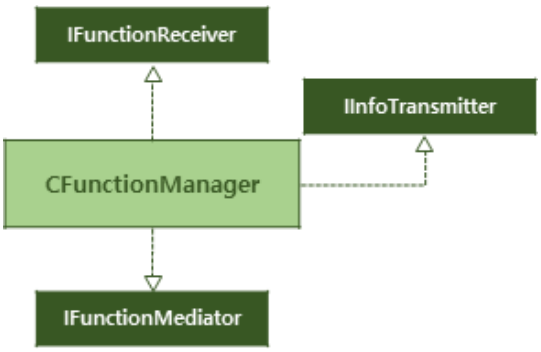


Fig. 6. Function Manager

1.3.1 주요기능

정보중재자 역할로서 상위로부터 수신한 정보를 기능 및 목적에 맞게 분류하여 해당하는 Subfunction Manager 컴포넌트로 분배하는 역할을 수행하거나, 반대로 하위로부터 수신한 정보를 상위로 전달하는 역할을 수행한다. 또한 Subfunction Manager 컴포넌트 간의 정보 교환 역할도 수행한다.

1.3.2 특징

정보중재자 역할 뿐만 아니라 모듈 정보처리를 담당하는 최하위 컴포넌트인 Subfunction Manager를 멤버로 가진다. 모듈의 기능에 따라 Subfunction Manager의 구

성이 달라지기 때문에 모듈의 특징에 따라 클래스 멤버구성에 변화가 생긴다. 설계 상 모듈의 기능처리 부분은 배제되어 있기 때문에 기능 개발 의존도가 낮다.

1.3.3 세부구조

◎ **컴포넌트 간 통신 인터페이스**

- IFunctionReceiver : Message Converter 컴포넌트
- IInfoTransmitter : Message Converter 컴포넌트 내 송/수신 변환기
- IFunctionMediator : Subfunction Manager 컴포넌트

◎ **클래스관계**

- 다수의 Subfunction Manager : 기능 정보처리

1.3.4 연결관계

◎ **상위연결**

- Message Converter 컴포넌트

◎ **하위연결**

- Subfunction Manager 컴포넌트

1.4. Subfunction Manager

Fig. 7은 Subfunction Manager 컴포넌트의 클래스 구성을 나타낸다.

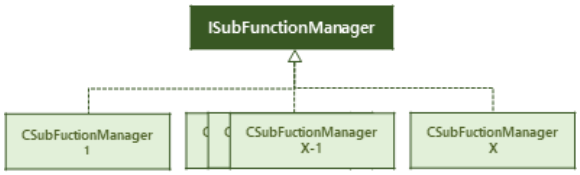


Fig. 7. Subfunction Manager

1.4.1 주요기능

정보처리 역할로서 상위로부터 수신한 정보를 실제로 해석하고 기능을 수행하는 역할을 수행한다.

1.4.2 특징

모듈 개발자는 정의된 모듈의 전체기능을 세부기능으로 나눌 수 있고 각 세부기능에 대응 하는 Subfunction Manager를 구성하고 기능을 할당 할 수 있다. 전체기능을 세부기능을 나누는 것은 모듈 개발자의 주관적인 영역이기 때문에 NSCP는 Subfunction Manager를 개발영역으로서 취급하고 그 기준과 개수를 제한하지 않는다.

Subfunction Manager 간에 직접적인 정보교환은 불가능하지만 필요할 경우 IFunctionMediator 인터페이스를 통해 Function Manager에 요청하여 교환가능하다.

1.4.3 세부구조

◎ **컴포넌트 간 통신 인터페이스**

- ISubFunctionManager : Function Manager 컴포넌트

◎ **클래스관계**

- 클래스관계 없음

1.4.4 연결관계

◎ **상위연결**

- Function Manager 컴포넌트

◎ **하위연결**

- 하위연결 없음

2. NSCP 체계동작

2.1. 플랫폼 외부 -> NSCP

2.1.1 Interface Manager -> Message Converter

Interface Manager 컴포넌트는 외부에서 수신된 데이터를 IInterfaceReceiver를 통해 Message Converter 컴포넌트로 전달 함.

2.1.2 Message Converter -> Function Manager

Message Converter 컴포넌트는 수신한 데이터를 멤버클래스 Info Maker를 이용해 NSCP 규격의 정보로 변환하고 IFunctionReceiver 인터페이스를 통해 Function Manager 컴포넌트로 전달 함.

2.1.3 Function Manager -> Subfunction Manager

Function Manager 컴포넌트는 수신한 정보를 분류하여 세부기능을 식별하고 ISubFunctionManager 인터페이스를 통해 담당 Subfunction Manager 컴포넌트로 전달 함.

2.1.4 Subfunction Manager

Subfunction Manager 컴포넌트는 수신한 정보를 이용해 담당 기능처리를 수행 함.

2.2. NSCP -> 플랫폼 외부

2.2.1 Subfunction Manager -> Function Manager

Subfunction Manager 컴포넌트는 기능처리 수행 중 NSCP 외부로 메시지 송신이 필요할 시 IFunctionMediator 인터페이스를 통해 Function Manager 컴포넌트로 메시지 송신을 요청 함.

2.2.2 Function Manager -> Message Converter

Function Manager 컴포넌트는 IFunctionSender 인터페이스를 통해 Message Converter 컴포넌트로 해당요청을 전달 함.

2.2.3 Message Converter -> Function Manager

Message Converter 컴포넌트는 해당요청을 멤버 클래스 Msg Maker로 전달하고 Msg Maker는 메시지를 구성하는데 필요한 NSCP 정보들을 IInfoTransmitter 인터페이스를 통해 Function Manager 컴포넌트로 요청 함.

2.2.4 Function Manager -> Subfunction Manager

Function Manager 컴포넌트는 요청 받은 NSCP 정보들의 관리 주체를 식별하여 ISubFunctionManager 인터페이스를 통해 담당 Subfunction Manager 컴포넌트로 요청 함.

2.2.5 Subfunction Manager -> Function Manager

Subfunction Manager 컴포넌트는 요청 받은 NSCP 정보를 반환 함.

2.2.6 Function Manager -> Message Converter

Function Manager 컴포넌트는 수신한 NSCP 정보들을 IInfoTransmitter 인터페이스를 통해 Message Converter 컴포넌트로 전달 함.

2.2.7 Message Converter -> Interface Manager

Message Converter 컴포넌트의 멤버 클래스 Msg Maker는 수신한 NSCP 정보들을 외부 규격의 데이터로 변환하고 IInterfaceSender 인터페이스를 통해 Interface Manager 컴포넌트로 전달 함.

2.2.8 Interface Manager

Interface Manager 컴포넌트는 수신한 데이터를 외부로 송신 함.

IV. Evaluation

Table 2와 Table 3은 기존 CMS 연동단과 NSCP 기반의 연동단의 클래스 구성을 비교하고 재사용 및 유지보수 관점의 비율을 측정한 결과를 보여준다.

Table 2. Class Comparison with Conventional ICU

	기존	NSCP
구성 클래스 개수(α)	13	38
공용 클래스 개수(β)	1	25
메시지 변경 시 수정 클래스 개수(γ)	5	5

Table 3. Ratio Comparison with Conventional ICU

	기존	NSCP
공용 클래스 포함 비율(β/α)	8 %	66 %
단일메시지 변경 시 구조변경 비율(γ/α)	38 %	13 %

NSCP는 구성클래스 개수가 기존대비 약 3배 늘어 구조적 복잡도가 상승하였으나 공용 클래스 포함 비율이 기존대비 약 58% 향상되어 높은 유지보수성을 확보했고, 단일메시지 변경 시 구조변경 비율이 기존대비 약 25 % 감소하여 높은 재사용성을 확보했다.

V. Conclusions

사용기간이 긴 방위산업 제품들의 특성 상 방산기업은 제품을 개발함에 있어 높은 재사용성과 동시성, 유지보수성을 확보하는 것이 바람직하다. 개선에 대한 연구가 수차례 진행되었음에도 불구하고 종래의 CMS 소프트웨어는 방위산업의 구조적인 특성과 기업의 이윤추구로 인해 재사용, 비용절감 명분하에 초기의 오래된 구조를 크게 벗어나지 못했다. 절차적 프로그래밍 패러다임으로 설계된 초기의 구조는 무기체계 소프트웨어 간 형상이 상이하여 관리 애로 및 체계통합 구현에 제한적이고, 공통기능에 대한 개별구현으로 재사용/재활용 저하 및 구현 리스크가 클 뿐만 아니라, 유연성 및 확장성이 낮다. 본 논문에서는 객체지향 프로그래밍과 그 설계 원칙인 SOLID를 준수하는 함정전투체계 표준 연동 소프트웨어 아키텍처로서 NSCP를 제시하였다. NSCP가 CMS 전반의 소프트웨어에 적용된다면 신규개발 뿐만 아니라 유지보수, 개선 및 확장에도 높은 효율을 낼 수 있을 것으로 보인다.

향후연구로 NSCP 아키텍처 설계 고도화 및 리팩터링이 필요하다. 또한 NSCP를 차기 CMS 소프트웨어 개발 프로젝트에 적용하여 기존 CMS 소프트웨어와 개발비용에 대한 정량적 평가를 수행할 뿐만 아니라, 프로젝트 참여 개발자들에게 기존대비 개발편의성 등에 대한 설문조사를 통해 정성적 평가를 수행하여 NSCP의 효용성을 평가할 예정이다.

REFERENCES

[1] Dong-Uk Kim, Hyung-Rok Jung, Young-Il Song, "A Study on Cost Behavior of Korean Defense Industry," Korean Journal of Management Accounting Research, Vol. 11, No. 2, pp. 55-82, 2011.

[2] Dong-Uk Kim, Hyung-Rok Jung, "A Study on Cost Structure of Korean Defense Firms," The Quarterly Journal of Defense Policy Studies, Vol. 28, No. 3, pp. 109-143, 2012.

[3] Susan S. Brilliant, Timothy R. Wiseman. "The first programming paradigm and language dilemma," Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, pp. 338-342, 1996.

[4] Sang-Min Kwon, Seung-Mo Jung, "Virtualization based high efficiency naval combat management system design and performance analysis," Journal of the Korea Society of Computer and Information Vol. 23, No. 11, pp. 9-15, 2018.

[5] Richard Conway, David Gries, "An Introduction to Programming (3rd ed.)," Little, Brown, New York City, 1979.

[6] Barry W. Boehm, "A spiral model of software development and enhancement," IEEE Computer, Vol. 21, No. 2, pp. 61-72, 1988.

[7] Peter Wegner, "Concepts and paradigms of object-oriented programming," ACM SIGPLAN OOPS Messenger, Vol. 1, No. 1, pp. 7-87, 1990.

[8] Martin Fowler, "Refactoring : Improving the Design of Existing Code," Addison-Wesley Professional, Boston, pp. 63ff, 1999.

[9] Michele Tufano, Fabio Palomba, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Andrea De Lucia, Denys Poshyvanyk, "When and Why Your Code Starts to Smell Bad," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, pp. 403–414, 2015.

[10] Robert C. Martin, "Agile Software Development, Principles, Patterns, and Practices," Prentice Hall, New Jersey, pp. 95-145, 2002.

[11] Robert C. Martin, "Clean Code: A Handbook of Agile Software Craftsmanship," Prentice Hall, New Jersey, pp. 138-140, 2008.

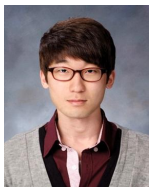
[12] Hun-Yong Shin, Joo-Yong Kim, "Research of OSD Standardization in Naval Combat System," Information and Control Symposium, pp. 354-355, 2012.

[13] Byung-Chul Choi, "A Study on the Agile software development about Combat System," Information and Control Symposium, pp. 323-326, 2014.

[14] Ki-Tae Kwon, "A Study on the Standardization of the combat system software in battleship using Objectoriented Design," Proceedings of the Korean Society of Computer Information Conference, Vol. 25, No. 2, pp. 296-297, 2017.

[15] Ki-Tae Kwon, Ki-Pyo Kim, HwanJun Choi, "Design of the Scalable Naval Combat System Software using Abstraction and Design Pattern," Journal of the Korea Society of Computer and Information, Vol. 24, No. 7, pp. 101-108, 2019.

Authors



Chi-Sun Baek received the B.S. and M.S. degrees in Computer and Information Engineering from INHA Univ., Korea, in 2012 and 2015, respectively. He is specialized in high performance computing. He is currently an Engineer in Hanwha Systems Co., Ltd.

Jin-Hyang Ahn received the B.S degree in Information and Communication Engineering from Yeungnam Univ., Korea, in 2016. She is currently an Engineer in Hanwha Systems Co., Ltd. She is interested in Standard Programming Model with Software Design Pattern.