JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 31 No. 1, pp. 213-226, January 2026
https://doi.org/10.9708/jksci.2026.31.01.213

An Improved Multilingual Support Techniques Applicable to Naval Combat System

Ye-Jun Jang*, Jae-Geun Lee*

*Engineer, Naval R&D Center, Hanwha Systems, Gumi, Korea

[Abstract]

This study proposes a structured approach to implementing multilingual support for naval combat management system software. Existing methods require repeated code modification and rebuilding, leading to high costs and effort that scale linearly as more languages are added. To overcome these limitations, the proposed design applies database-driven mapping data, dynamic binding for run-time switching, feature modeling to manage variability, and the singleton pattern to optimize resource usage. The core mechanism is that, at program initialization, mapping data for the selected language is loaded once, and each class subsequently references this dataset to obtain the required translated information. Evaluation results indicate that while the initial setup requires more time, later additions or modifications are handled through database updates alone, yielding substantial efficiency improvements. The approach thus enhances export competitiveness and ensures interoperability in multinational naval operations.

▸ **Key words**: Naval Combat System, Multilingual Support, Dynamic Binding, Feature Model, Singleton Design Pattern

[요 약]

본 연구는 함정 전투체계 소프트웨어에 다국어 지원 기능을 적용하기 위한 구조적 기법을 제안한다. 기존 방식은 언어별 코드 수정과 리빌드가 필요하여 관리 비용이 많이 들고, 언어 수가 늘어날수록 수정 과정이 선형적으로 증가하는 한계를 가진다. 이를 해결하기 위해 본 연구는 데이터베이스 기반 매핑 구조, 런타임 언어 전환을 위한 동적 바인딩, 변동성 관리를 위한 휘처 모델, 효율적 자원 활용을 위한 싱글톤 패턴을 적용하였다. 핵심은 응용 프로그램 초기 실행 시 설정된 언어에 대응하는 매핑 데이터를 생성하고, 이후 각 클래스가 필요할 때 이를 참조하여 변환된 정보를 반환받는 구조이다. 성능 검증 결과, 초기 도입에는 시간이 더 소요되지만 이후 언어 추가나 용어 수정은 데이터베이스 갱신만으로 처리되어 높은 효율성을 보였다. 본 연구는 다국어 지원을 통한 수출 경쟁력 강화와 연합 운용 환경에서의 상호운용성을 확보할 수 있음을 확인하였다.

▸ **주제어**: 함정 전투체계, 다국어 지원, 동적 바인딩, 휘처 모델, 싱글톤 디자인 패턴

• First Author: Ye-Jun Jang, Corresponding Author: Ye-Jun Jang
*Ye-Jun Jang (yejunjang@hanwha.com), Naval R&D Center, Hanwha Systems
*Jae-Geun Lee (jg5250.lee@hanwha.com), Naval R&D Center, Hanwha Systems
• Received: 2025. 11. 13, Revised: 2025. 12. 17, Accepted: 2025. 12. 24.

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

I. Introduction

한국의 방위산업 수출은 최근 몇 년간 괄목할 만한 성장세를 보이고 있다. 우리나라 방산 수출은 2022년과 2023년 연속으로 100억 달러를 돌파했으며, 이를 바탕으로 글로벌 시장에서 위상이 점차 확대되고 있다는 평가가 있다[1]. 특히, 국내 함정 전투체계는 최근 해외 시장에서도 성과를 내기 시작했다. 예컨대, 한 국내 방산 업체는 2,400톤급 필리핀 연안경비함 6척에 국산 함정 전투체계를 공급하는 계약을 체결했으며, 이 계약은 전술데이터링크를 포함해 총 3,450만 달러 규모이다[2]. 국내 방위산업 관련 동향 보고서에서는 수출 대상국과 체계 종류가 다양해지고 있다는 점을 강조하며 국내 기업의 기술 역량 강화와 수출 경쟁력 제고의 중요성을 함께 제시하고 있다[3].

이러한 수출 확대 흐름 속에서, 단순히 무기 체계 자체의 성능만으로는 경쟁력을 확보하기 어렵다. 특히 함정 전투체계는 소프트웨어 계층의 중요성이 매우 크며, 해외 해군에 수출된다면 현지 운용자들이 쉽게 접근하고 이해할 수 있는 UI 및 언어 지원 능력이 구매 결정 요인으로 작용할 수 있다. 즉, 해외 해군은 기술 사용뿐 아니라 운용자 친화성, 안정성, 유지보수성 그리고 현지화 가능성(Localization) 등을 중요하게 고려할 가능성이 높다.

함정 전투체계에 다국어 지원 기능을 도입해야 하는 이유는 다음과 같다. 첫째로 수출 경쟁력 강화이다. 해외 해군이 시스템 운용 시 UI나 메시지를 현지 언어로 사용할 수 있다면 기술 도입 진입 장벽이 낮아지고, 구매국의 신뢰도를 높일 수 있다. 둘째로 운용 효율성 및 교육 부담 감소를 들 수 있다. 승조원이나 운용자가 모국어 환경에서 체계를 사용하는 경우, 시스템 이해와 조작이 쉬우므로 교육 기간과 오류 발생 가능성이 줄어든다. 다음은 연합 작전 및 상호운용성 확보이다. 다국적 연합 작전 환경에서는 승조원이 각자 선호 언어로 체계를 활용할 수 있어야 하며, 이를 통해 협업과 정보 공유가 원활해진다. 마지막으로 신규 언어 추가나 유지보수 작업이 비교적 덜 복잡해지므로 확장성 및 유지보수성 측면에 강점이 있다. 즉, 다국어 지원 기능은 전투체계의 운용 편의성을 넘어 전술적 효과와 직결되는 핵심 기술 요소로서, 글로벌 방산 시장 경쟁력 확보와 국제 연합 작전이 보편화되는 현대 해군 환경에서 반드시 고려되어야 한다.

본 논문에서 제안하는 다국어 지원 기법은 네 가지 핵심 축을 중심으로 전개된다. 우선, 응용 프로그램 실행 시 언어를 선택하고 이를 전역적으로 관리하여 모든 모듈이 동일한 언어 설정을 공유하도록 한다. 싱글톤 패턴을 활용해 다국어 지원 객체를 단일하게 관리함으로써 불필요한 자원 소모를 줄이고 시스템 전체의 일관성을 유지한다. 이어서 데이터베이스(Database, DB) 기반의 용어 매핑 구조를 도입하여, 언어별 리소스를 독립적으로 관리하고 유지보수의 효율성을 확보한다. 나아가 런타임 시점에서 언어 전환이 가능하도록 설계함으로써, 운용자는 언어 설정의 변경만으로 원하는 언어 환경을 얻을 수 있다. 마지막으로 기능별 휘처 모델(Feature Model)을 적용하여 필수 기능과 선택 기능을 명확히 구분함으로써, 향후 유지보수 및 확장 과정에서 유연성을 보장한다. 또한 본 시스템 구조는 장기 유지보수 가능성과 변화 대응 능력을 고려하여 아키텍처 지속성(Sustainability) 관점에서 평가할 수 있는 구조로 구성하고자 한다. 유사 연구[4]에서는 기업용 소프트웨어에서 기능 모듈별 언어 리소스 파일을 분리하지 않고 공통 리소스를 활용하는 설계를 제안하여 메모리나 저장 공간 낭비를 줄이고 유지보수성을 개선한 바 있다.

본 논문의 구성은 다음과 같다. 2장에서는 함정 전투체계 소프트웨어 구조와 제안하는 기법에 적용되는 기술을 설명한다. 3장에서는 현재 함정 전투체계의 다국어 지원 방식을 설명하고, 4장에서는 기존 방식을 개선한 다국어 지원 기법을 제안한다. 5장에서는 기존 및 제안하는 다국어 지원 기법을 반영했을 때의 성능을 비교 및 분석한다. 마지막으로 6장에서는 결론 및 기대하는 향후 연구 방향성을 기술한다.

II. Preliminaries

1. Naval combat system software

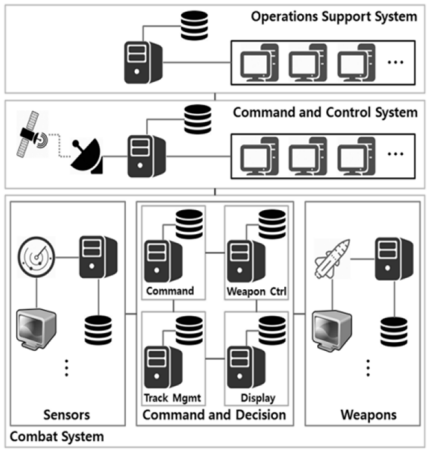

Fig. 1. Structure of legacy naval combat systems

국내에서 개발된 함정 전투체계는 Fig. 1과 같이 다양한 하부체계들이 통합되어 있다[5]. 함정 전투체계 소프트웨어는 함정 내 다양한 센서, 무장, 항해 장치 등을 통합적으로 관리하고, 전술적 상황 인식 및 교전을 지원하는 핵심 체계 소프트웨어이다. 레이더나 소나와 같은 센서에서 수집한 데이터를 처리하고, 이를 기반으로 전술 화면을 제공한다. 이러한 소프트웨어는 실시간성, 고신뢰성, 안정성이 요구되며, 동시에 다수의 장비와 연동되어야 하는 특성 때문에 개발 및 유지보수의 복잡성이 매우 높다.

이러한 복잡성을 완화하고 개발 효율을 높이기 위해 소프트웨어 공학적 접근이 꾸준히 시도 되어 왔다. 특히 소스코드의 재사용률을 높이고, 효율적인 유지보수를 위한 연구가 활발히 진행되고 있으며, 공통 요소와 가변 요소를 구분하여 관리하는 휘처 모델링, 표준 아키텍처 적용, 디자인 패턴 활용 등이 대표적인 방법론으로 제안되고 있다.

선행 연구들에서는 전투체계 소프트웨어의 구성 요소를 분석하여 재사용성을 높이고, 변경에 유연하게 대응할 수 있는 구조를 마련함으로써 개발 비용 절감과 시험 시간 단축 효과를 확인하였다[6-8]. 이러한 성과는 함정전투체계 소프트웨어의 지속적인 진화와 수출 경쟁력 강화에도 이바지할 수 있으며, 본 연구 역시 이와 같은 흐름 속에서 개선된 기법을 제안한다.

2. Related works

2.1 Dynamic Binding

바인딩(Binding)이란 프로그램의 실행 과정에서 변수, 함수 호출 혹은 자원 참조가 실제 메모리상의 주소나 구체적인 동작과 연결되는 과정을 의미한다. 바인딩 시점에 따라 정적(Static) 바인딩과 동적(Dynamic) 바인딩으로 구분할 수 있으며, 구조적 차이는 Fig. 2와 같다.

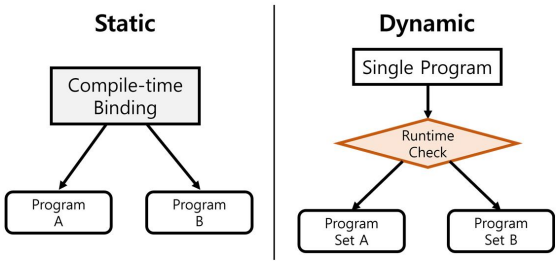

Fig. 2. Conceptual diagrams of binding

정적 바인딩은 컴파일 시점에 호출 대상을 결정하는 방식이다. 이 방식은 실행 속도가 빠르고 구현이 단순하다는 장점이 있으나 프로그램 수정이나 기능 확장이 필요한 경우 전체를 재컴파일해야 하므로 유연성과 재사용성 측면에서 한계가 존재한다.

동적 바인딩은 실행 시점에 호출 대상을 결정하는 방식이다. 프로그램은 런타임에 조건이나 설정값을 확인한 뒤 적절한 동작을 선택한다. 초기 실행 시 다소의 비용이 발생할 수 있으나, 다양한 상황에 대응할 수 있어 확장성과 유연성이 뛰어난 방식이다.

바인딩 기법은 기존에도 다양한 연구에서 논의 되어 왔다. 예를 들어, 정적 바인딩 구조에서 데이터 형식 변경 시 재컴파일이 불가피한 한계를 지적하고, 공유 메모리(shared memory)를 이용하여 동적으로 데이터 형식과 변수명을 바인딩하는 기법의 제안이 있다. 이를 통해 응용 소프트웨어를 중단하지 않고도 데이터 구조를 수정할 수 있음을 보인 바 있다[9]. 이러한 사례는 동적 바인딩이 정적 바인딩에 비해 실행 중 유연성과 확장성 확보에 유리함을 뒷받침한다.

2.2 Feature Model

휘처 모델(Feature model)은 소프트웨어 제품군(Software product line, SPL)에서 공통성과 가변성을 명확히 구분하기 위해 고안된 대표적인 모델링 기법이다. 이를 통해 여러 제품군에서 반복적으로 사용되는 기능은 공통(Common) 휘처로, 상황에 따라 달라질 수 있는 기능은 가변(Variable) 휘처로 정의하여 체계적으로 관리할 수 있다. 이러한 접근은 복잡한 시스템의 구조를 단순화하고, 코드 재사용성과 유지보수 효율을 높이는 데 효과적이다.

휘처 모델은 각 기능을 휘처(Feature) 단위로 표현하며, 휘처 간 관계는 보통 네 가지 유형으로 분류된다. 필수(Mandatory) 휘처는 모든 제품에 반드시 포함되는 기능이고, 선택(Optional) 휘처는 필요에 따라 포함 여부를 결정할 수 있다. 대안(Alternative) 휘처는 여러 후보 중 하나만 선택할 수 있는 경우를 의미하며, Or 휘처는 두 개 이상을 동시에 선택할 수 있으나 최소 하나는 선택되어야 하는 경우를 나타낸다.

Fig. 3은 마우스를 대상으로 작성한 휘처 모델의 예시이다. 좌·우 클릭 버튼은 필수 휘처이며, 뒤로가기와 같은 특수 기능의 버튼은 선택 휘처로 표현된다. 연결 방식(Connectivity)은 유선(Wired)과 무선(Wireless) 중 하나 이상을 선택하는 Or 휘처로, 센서 타입(Sensor type)은 광학(Optical), 레이저(Laser) 중 하나를 선택하는 대안 휘처로 나타낼 수 있다.

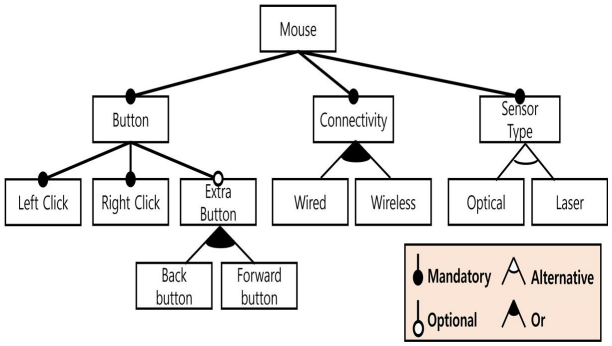

Fig. 3. Example of a mouse feature model

국내 연구에서도 휘처 모델은 SPL의 재사용성 향상과 변동성 관리에 적용되었다. 일부 연구에서는 휘처 모델 기반 분석을 통해 공통 요소와 가변 요소를 구분하여 유지보수 비용을 절감하고, 시스템 확장 시 효율성을 확보할 수 있음을 보였다[10, 11]. 따라서 휘처 모델은 실제 소프트웨어 개발 현장에서 효율성과 재사용성을 높이는 중요한 수단으로 자리 잡고 있으며, 본 연구에서도 제안하는 다국어 지원 기법의 구조적 관리에 이를 적용하였다.

2.3 Singleton Design Pattern

싱글톤 패턴(Singleton pattern)은 객체지향 설계에서 널리 사용되는 디자인 패턴(Design pattern) 중 하나로, 특정 클래스의 인스턴스를 하나만 생성하고 이를 전역적으로 공유할 수 있도록 보장하는 구조를 갖는다. 시스템 전역에서 공통으로 사용되는 자원이나 설정을 관리할 때 유용하며, 로깅, 캐시, 스레드 풀 등 다양한 분야에서 활용된다. 이러한 특성 때문에 싱글톤 패턴은 소프트웨어의 일관성과 자원 관리 효율성을 높이는 대표적인 설계 기법으로 자리 잡았다.

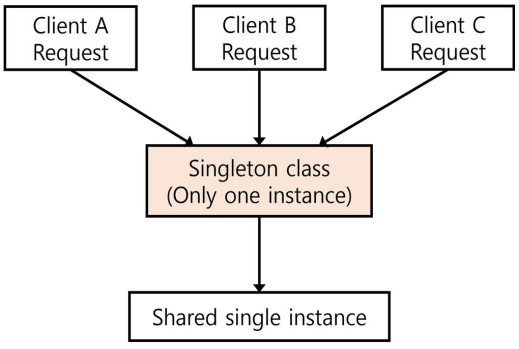

Fig. 4. Concept of the singleton pattern

Fig. 4는 싱글톤 패턴의 개념을 시각적으로 나타낸 것이다. 여러 클라이언트(Client A, B, C)의 요청은 모두 싱글톤 클래스(Singleton class)로 전달되며, 이 클래스는 단 하나의 인스턴스만 생성한다. 결과적으로 모든 요청은 공유된 단일 인스턴스(Shared single instance)를 참조하게 되며, 이를 통해 불필요한 객체 생성을 방지하고 일관된 자원 관리를 보장할 수 있다. 이러한 구조는 특히 시스템 규모가 커질수록 관리 효율성에 중요한 역할을 한다.

관련 연구에서는 싱글톤 패턴을 적용하여 메모리 사용량을 줄이고 관리 효율을 높인 사례가 보고되었다[12]. 이는 싱글톤 패턴이 단순한 설계 개념을 넘어, 실제 응용 소프트웨어에서 성능 최적화와 자원 효율화를 실현할 수 있는 실질적인 기법임을 보여준다.

기존 연구[4]에서는 3-tier 구조의 응용 소프트웨어를 대상으로 다국어 지원을 위한 구조를 제안하고, 언어 리소스를 공통화함으로써 메모리 사용량과 저장 공간 측면에서의 효율성을 개선하는 방안을 제시하였다.

그러나 다수의 응용 프로그램이 통합되어 운용되는 함정 전투체계 환경에서는, 개별 응용 프로그램 수준의 구조 개선뿐만 아니라 시스템 전반에 적용 가능한 다국어 지원 구조와 언어 추가/변경 시 발생하는 유지 보수 작업의 효율성을 고려할 필요가 있다. 기존 연구에서는 이러한 통합 운용 환경을 대상으로 한 구조 적용이나, 언어 확장 과정에서 요구되는 유지보수 작업량에 대한 정량적 비교는 다루지 않았다.

이에 본 연구에서는 함정 전투체계 소프트웨어 환경을 대상으로 다국어 지원 구조를 시스템 전반에 적용하고, 언어 추가 및 용어 수정 과정에서 발생하는 작업 시간을 기준으로 기존 방식과 제안 기법을 비교 분석함으로써, 통합 운용 환경에서의 유지보수 효율성과 확장성을 정량적으로 평가한다는 점에서 기존 연구와의 차별성을 가진다.

III. The Exist Scheme

기존의 다국어 지원 방식은 정적 바인딩에 기반한다. 이 방식에서는 응용 프로그램을 빌드하기 이전에 특정 언어가 코드 내부에 고정되어 포함되며, 하나의 언어를 지원하기 위해서는 별도의 프로젝트를 복제한 뒤 언어 관련 코드를 직접 수정해야 한다. 이렇게 생성된 프로젝트는 각각 독립적으로 빌드되어 언어별 실행 파일을 산출한다.

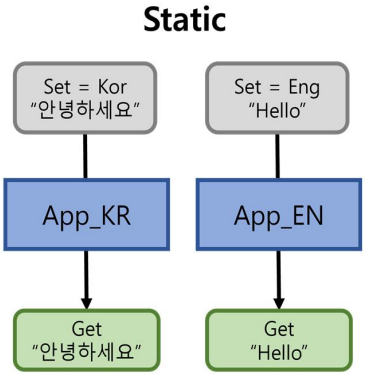

Fig. 5. Static binding in multilingual support

이러한 구조는 Fig. 5에서 확인할 수 있다. 예를 들어, 사용자가 언어 설정을 한국어(Set = Kor)로 지정하면 응용 프로그램 App_KR가 생성되고, 이를 통해 "안녕하세요"가 출력된다. 마찬가지로 영어(Set = Eng)를 지정하면 응용 프로그램 App_EN이 생성되며, 이를 통해 "Hello"가 출력된다. 즉, 언어의 수만큼 독립된 응용 프로그램이 존재해야 하며, 새로운 언어를 지원하기 위해서는 그에 대응되는 응용 프로그램이 추가로 생성되어야 한다.

문제는 이러한 구조가 유지보수 단계에서 심각한 부담을 초래한다는 점이다. 예컨대 일부 기능에 수정이 발생할 경우, 한국어 버전과 영어 버전의 프로젝트에 각각 동일한 소스코드를 반복 적용해야 한다. 즉, 소스코드 수정, 빌드, 테스트 및 신뢰성 시험 과정 또한 언어의 수만큼 중복되므로 관리 효율성은 급격히 떨어진다. 이를 정량적으로 표현하면, 유지보수 비용 $C(n)$은 아래와 같이 나타낼 수 있다.

$$C(n) = k \cdot n \tag{1}$$

여기서 k는 각 프로젝트 유지보수에 필요한 비용을, n은 지원하는 언어 수(=프로젝트 수)를 의미한다. 즉, 새로운 언어가 하나 추가될 때마다 전체 비용은 k만큼 증가하며, 장기적으로는 프로젝트 수가 늘어날수록 관리와 유지보수 부담이 선형적으로 커지게 된다.

결론적으로, 정적 바인딩 기반 접근은 언어가 늘어날수록 프로젝트 수가 선형적으로 증가하여, 유지보수 비용 및 중복적인 테스트의 부담이 누적되는 구조적인 한계를 안고 있다. 따라서 이 방식은 단기적인 구현에는 적합할 수 있으나, 장기적인 확장성과 효율성 측면에서는 한계가 뚜렷하다.

IV. The Proposed Scheme

1. Multilingual support method

1.1 Database structure

함정 전투체계 소프트웨어는 다수의 모듈과 기능을 포함하며, 각 모듈에서 사용되는 용어와 UI 요소가 언어별로 다르게 표현될 수 있다. 따라서 모든 용어와 메시지를 소스코드 내부에 하드 코딩하는 기존의 방식은 유지보수성, 확장성 측면에서 한계를 가진다. 이를 극복하기 위해 본 연구에서는 Table 1과 같은 다국어 관리 DB 구조를 도입하였다. DB 테이블은 기본적으로 Index, Base, 다국어 변환 용어 그리고 SW ID로 구성된다.

Table 1. Database structure for multilingual support

Index	Base	Lang 1	Lang 2	SW ID
1	가로	가로	Width	11
2	세로	세로	Height	11
3	높이	높이	Depth	11

여기서 Index는 각 프로그램 내에서 용어를 식별하기 위한 번호로 SW 단위로, 독립적으로 부여된다. 즉, SW ID가 달라지면 Index는 다시 1부터 시작한다. Base는 소스코드에 삽입되는 원본 용어를 저장한다. Lang1, Lang2와 같은 다국어 열(Column)은 한국어, 영어, 중국어 등 지원 언어별 번역 문자열을 저장한다. 새로운 언어가 필요할 경우, 단순히 새로운 열을 추가하는 방식으로 확장이 가능하다. 마지막으로 SW ID는 각 응용 프로그램을 식별하기 위한 고유 번호이다.

Table 2. Example of multilingual term mapping in the database

Index	Base	Lang 1	Lang 2	SW ID
1	수동1	수동	Manual	11
2	수동2	수동	Passive	11
1	시간	시간	Time	30
1	시간	시간	h	42

Table 2는 Base의 활용과 SW의 독립성을 나타낸 예이다. Base는 소스코드에 작성되는, 외부로 드러나지 않는 용어이다. 이를 활용하여 하나의 응용 프로그램 내에서 특정 언어에서는 동일한 용어가 다른 언어에서는 다르게 표현되도록 할 수 있다. 예를 들어, Base에 "수동1", "수동2"와 같이 구분하여 DB에 저장하면 설정 언어가 한국어라면 모두 동일한 "수동"으로 출력되지만, 설정 언어가 영어

라면 상황에 따라 각각 “Manual”, “Passive”가 출력된다. 반면 3, 4번째 행과 같이 동일한 Base여도 SW ID가 다를 경우, 독립된 정보이므로 Base에서 별도 구분을 주지 않더라도 각 응용 프로그램별로 원하는 변환 결과를 얻을 수 있다.

1.2 Class architecture and configuration

제안하는 기법의 다국어 지원 클래스 구조는 초기화 단계(Initialization Phase)와 운용 단계(Runtime Usage Phase)로 구분된다. 이러한 구분은 시스템의 성능 저하를 방지하고, 소스코드 레벨에서 일관성을 확보하며, 유지보수성의 향상을 목적으로 한다.

초기화 단계는 응용 프로그램이 실행되는 시점에서 단 한 번만 수행된다. 먼저 외부의 언어 설정 파일을 참조하여 운용자가 설정 언어를 확인하고, 이후 응용 프로그램 식별자(SW ID)를 기반으로 해당 응용 프로그램에 속하는 언어 DB를 읽어온다. 불러온 데이터는 Base와 선택 언어의 변환 용어 간 Key-Value 구조의 데이터 맵(Data map)을 생성하며, 이 매핑 정보는 이후 응용 소프트웨어 내에서 전역적으로 활용할 수 있다. 이와 같은 초기화 과정을 통해 다국어 리소스가 일괄적으로 준비되고, 프로그램 전체에서 일관된 참조가 보장된다.

운용 단계에서는 다국어 지원 클래스가 제공하는 기능들을 활용한다. 예를 다국어 지원 클래스의 TransTerm() 함수는 Base 용어(Key)를 입력하면 이에 대응하는 다국어 용어(Value)를 반환한다. 만약 DB에 해당 Key가 존재하지 않을 경우, 빈값이 반환되도록 설계되어 예외 상황에서도 안정적인 동작을 보장한다.

앞서 설명한 초기화 및 운용 절차의 구조는 Fig. 6과 같이 나타낼 수 있다. 초기화 단계가 “설정 파일 읽기 - DB 읽어오기 - 매핑 데이터 생성”의 과정을 포함하며, 초기화가 완료된 이후 운용 단계에서 TransTerm() 함수를 통해 입력된 용어에 대한 변환 결과를 얻을 수 있다. 이러한 구조적 분리는 다국어 지원 기능의 효율적인 구현을 가능하게 하며, 특히 언어 확장성과 시스템 유지보수 측면에서 유리하다.

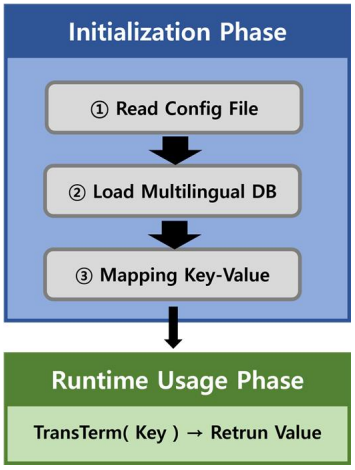

Fig. 6. Initialization and runtime usage phase flow

2. Application of the proposed method

본 절에서는 제안하는 다국어 지원 기법의 세부 기술을 다룬다. 주요 기법은 동적 바인딩, 휘처 모델링, 싱글톤 패턴으로 구분되며, 이를 순차적으로 설명한다.

2.1 Dynamic binding for runtime language switching

동적 바인딩은 응용 프로그램 실행 시점에서 언어 변환을 유동적으로 처리하기 위한 핵심 기법이다. 기존 정적 바인딩 방식은 특정 언어 리소스를 소스코드에 직접 포함하므로 다국어 확장성이 제한되는 단점이 있다. 반면 동적 바인딩은 런타임에서 운용자가 선택한 언어를 판별한 후, 매핑 구조에 저장된 해당 언어의 리소스를 참조하여 결과를 반환하는 방식으로 동작한다.

제안하는 기법에서는 응용 프로그램 초기화 과정에서 외부의 언어 설정 파일을 통해 제공할 언어가 확정된다. 이후 다국어 맵 데이터를 생성하게 되며, 입력된 용어(Key)를 설정된 언어에 대응하는 변환 용어(Value)로 매핑한다. 예를 들어, “안녕하세요”라는 용어가 입력될 경우, 설정된 언어가 한국어라면 동일하게 “안녕하세요”를 반환하고, 영어로 설정되었다면 “Hello”라는 변환 결과를 반환한다.

이와 같은 동작은 아래의 Fig. 7에 제시된 바와 같다. 단일 응용 프로그램에서 언어를 확인하고, 이후 선택된 언어에 따라 서로 다른 번역 결과를 반환하는 과정을 도식화하였다. 이러한 동적 바인딩 구조는 다국어 지원 기능의 유연성을 극대화하며, 사용자 요구에 따라 언어 전환을 가능하게 한다는 점에서 기존 정적 구조와 명확히 구분된다.

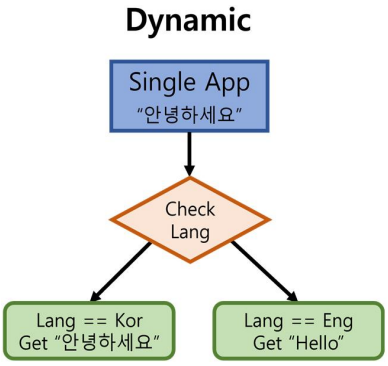

Fig. 7. Dynamic binding in multilingual support

동적 바인딩의 적용은 여러 가지 장점을 제공한다. 첫째, 하나의 실행 파일만으로 다수의 언어를 지원할 수 있으므로, 유지관리 과정이 단순화된다. 둘째, 효율성 측면에서 기존 정적 구조와 달리 언어 추가 시 전체 빌드가 불필요하므로 개발 시간과 비용이 절감된다. 마지막으로 새로운 언어를 추가할 경우, 기준 용어에 대응되는 다국어 용어만 추가하면 되므로 확장성이 보장된다. 따라서 동적 바인딩은 기존의 정적 바인딩 구조에 비해 다국어 지원의 실효성을 크게 높이며, 함정 전투체계와 같이 다양한 환경에서 운용되는 시스템에 특히 적합한 기법이라 할 수 있다.

2.2 Feature modeling for selective language support

휘처 모델링은 소프트웨어 제품 계열에서 공통성과 변동성을 체계적으로 관리하기 위한 대표적 기법으로, 시스템의 확장성과 유지보수성을 보장하는 데 효과적이다. 본 연구에서는 함정 전투체계 소프트웨어에 다국어 지원 기능을 적용하기 위해 휘처 모델을 도입하였다. 이를 통해 계층별 기능을 필수 휘처, 선택 휘처, 그리고 대안 휘처로 구분함으로써, 운용 환경과 요구사항에 따라 유연하게 확장할 수 있는 구조를 마련하였다.

제안하는 모델은 데이터 계층(Data layer), 응용 계층(Application layer), 표현 계층(Presentation layer)의 세 부분으로 구성된다. 데이터 계층은 설정 파일 처리, DB 가져오기, Key-Value 매핑과 같은 핵심 기능을 포함하며, 이 중 설정 파일 쓰기 기능은 운용자의 언어 변경 요청이 있을 때만 필요하므로 선택 휘처로 정의하였다. 응용 계층은 용어 변환 기능을 필수 휘처로 설정하여 모든 프로그램이 공통으로 참조할 수 있도록 하였다. 표현 계층은 UI 조정 및 값 전시 포맷 기능을 선택 휘처로 두어, 특정 언어나 국가권에 따라 다르게 동작할 수 있다.

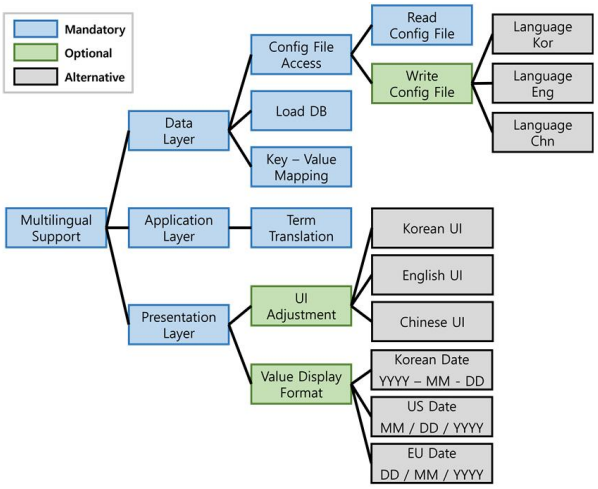

Fig. 8. Proposed multilingual support feature model

Fig. 8은 제안하는 휘처 모델을 시각적으로 나타낸 것이다. 모델은 왼쪽에서 오른쪽으로 확장하는 형태로, 필수·선택·대안 휘처를 색상으로 구분하였다. 예를 들어, 영어가 기본 언어로 선택된 경우, UI 전시 및 날짜 표기 방식이 해당 언어와 문화권에 맞게 전환된다.

제안하는 모델의 장점은 다음과 같다. 첫째, 기존의 언어별 프로젝트 분리 또는 하드코딩 방식과 달리, 단일 구조 내에서 다국어 지원을 통합 관리할 수 있어 중복 구현과 유지보수 부담을 크게 줄인다. 둘째, 신규 언어 추가 시 DB 및 설정 파일 확장만으로 대응이 가능하므로 확장성이 뛰어나다. 셋째, UI 및 데이터 전시 포맷을 선택적 기능으로 분리함으로써, 언어 및 국가권별 운용 요구에 대한 대응력을 확보할 수 있다. 결과적으로 다국어 지원 기법에 휘처 모델을 적용함으로써 구조적 명확성, 유연한 확장성 그리고 운용 환경 적합성을 동시에 달성할 수 있다.

2.3 Singleton pattern for efficient resource usage

소프트웨어 아키텍처 설계에서 싱글톤 패턴은 객체의 인스턴스를 단 하나만 생성하고, 전역적으로 이를 공유할 수 있도록 보장하는 대표적인 설계 기법이다. 이러한 특성은 불필요한 중복 생성을 억제하고, 공통 자원을 중앙에서 관리할 수 있다는 점에서 효율적인 자원 활용을 가능하게 한다. 전통적으로 싱글톤 패턴은 설정 관리, 로그 기록 등과 같이 시스템 전반에서 공용 데이터 접근이 필요한 영역에 널리 사용 되어왔다.

제안하는 다국어 지원 기법에서도 싱글톤 패턴을 적용하였다. 다국어 매핑 데이터를 관리하는 초기화 클래스(Initialize class)는 시스템 구동 시 실행되면서, 외부 설정 파일과 DB를 참조하여 현재 운용 언어와 이에 대응하

는 매핑 정보의 집합을 한 번만 생성한다. 이 집합은 응용 프로그램 전체에서 공통으로 참조되는 단일 인스턴스로 유지되며, 이후 각 응용 클래스는 별도의 초기화 과정 없이 이를 활용할 수 있다.

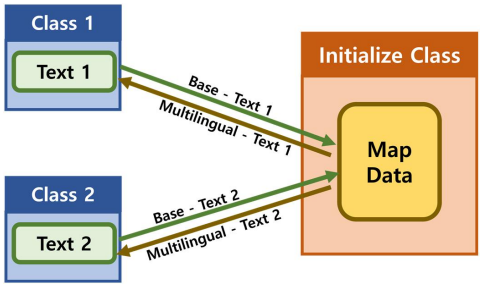

Fig. 9. Singleton design pattern

Fig. 9는 제안된 구조를 나타낸 것이다. 오른쪽의 초기화 클래스는 다국어 맵 데이터를 관리하며, 왼쪽의 각 응용 클래스(Class 1, 2)는 다국어 변환이 필요한 Base 용어를 전달한다. 전달된 값은 맵 데이터로부터 변환 텍스트로 반환된다. 이러한 흐름은 요청과 응답의 관계로 표현되며, 모든 모듈이 동일한 자원을 공유함을 직관적으로 보여준다.

이와 같은 구조는 다음의 효과들을 제공한다. 첫째, 다국어 맵 데이터가 한 번만 생성되어 모든 클래스가 이를 공유하므로 효율적인 자원 활용이 가능하다. 둘째, 전체 시스템이 동일한 맵 데이터를 참조하므로 변환 결과의 일관성이 유지된다. 마지막으로, 언어 변경이나 용어 수정 시 맵 데이터만 갱신하면 전체 응용에 반영되므로 유지보수성이 높다. 결과적으로 싱글톤 기반 구조는 함정 전투체계 소프트웨어의 다국어 지원 기능을 효율적이고 일관되게 관리하는 핵심 메커니즘으로 작동하며, 시스템 운용 및 유지보수 비용 절감에도 효과가 있다.

V. Software Evaluation

1. Evaluation environment

본 연구의 성능 검증은 Table 3에 제시된 시험 환경에서 수행되었다. 하드웨어 및 소프트웨어의 사양은 CPU, 메모리, 운영체제(OS), 개발 및 신뢰성 시험 도구, 그리고 DB 도구로 구성된다.

Table 3. Experimental system specifications

Component	Specification
CPU	Intel(R) Core(TM) Ultra7 155H
Memory	64 GB
Operating System	Windows 11 x64
Development Tool	Visual Studio 2008
Reliability Tool	LDRA
Database Tool	Microsoft Access Database

또한 본 논문에서는 다국어 지원 기법의 성능 차이를 정량적으로 평가하기 위해 함정 전투체계 내 5개의 응용 프로그램 A~E를 선정하였다. 각 응용 프로그램의 규모, 용어 및 UI/포맷의 수는 Table 4와 같다. LOC(Line of code)는 최소 14,578라인에서 최대 약 136,876라인까지 다양하게 분포한다. 이를 통해 소규모에서 대규모까지 다양한 사례를 포괄하도록 하였다. 특히, 다국어 지원 클래스의 LOC는 215라인에 불과하여 전체 응용 대비 작은 비중을 차지했다. 결과적으로 제안 기법 적용 여부와 무관하게 빌드 시간은 극히 적은 차이가 존재했다. 빌드 시간과 신뢰성 시험 시간은 Table 3의 시험 환경에서 실제로 측정한 결과이며, 계산·비교의 일관성을 위해 10초 단위로 정규화(반올림)하여 기록하였다.

Table 4. Test applications for evaluation

App	LOC	No. of term	No. of UI / format	Build time (sec)	Reliability time (sec)
A	136,876	360	6	310	51,840
B	26,152	360	1	310	2,710
C	23,577	519	1	270	2,250
D	14,578	98	0	380	10,590
E	23,042	110	0	370	17,620

본 연구에서는 Table 4에 제안하는 기법 적용 여부와 관계없이 각 응용 프로그램별 빌드 시간을 동일하게 정의하였다. 마찬가지로 신뢰성 시험 또한 기법 적용 여부에 따른 차이가 극히 미미하여 시험 시간을 동일하게 두었다. 즉, 본 연구에서는 빌드 및 신뢰성 시험 시간을 일정 값으로 두고, 언어 확장 및 용어 수정 시 발생하는 소스코드 수정과 다국어 DB 갱신 여부에 따른 성능 차이를 비교한다.

2. Performance evaluation

2.1 Performance at first language introduction

최초 외국어를 도입하는 단계에서 제안하는 기법은 기존 방식과 달리 다양한 초기 작업을 요구한다. 여기에는 초기화 코드 삽입, DB 추가, 용어 치환, 그리고 UI 및 포맷 수정이 포함된다. 초기화 코드는 단순히 다국어 지원 객체를 생

성하는 작업이 아니라, 싱글톤 패턴의 적용을 통해 전체 시스템에서 공유되는 단일 객체를 준비하는 과정이다. 초기화 클래스에서 단 한 번 생성된 객체가 이후 모든 클래스에서 참조되므로, 중복 생성으로 인한 메모리 낭비와 데이터 불일치 가능성이 차단되고 매핑 데이터에 대한 즉시 접근이 가능해진다. 초기화는 최초 한 번만 수행되며, Table 5와 같은 코드 수정 시간은 60초로 산정하였다.

본 연구에서 정의한 단위 작업 시간은 실제 함정 전투체계 소프트웨어 개발 및 유지보수 경험을 보유한 실무 인력 8명이 참여한 협의 과정을 통해 도출된 산정값이다. 참여 인력은 10년 이상의 경력을 보유한 2명, 5년 이상 10년 미만의 경력을 보유한 인원 4명, 5년 미만의 경력을 보유한 인원 2명으로 구성되었으며, 각 작업 단계에 소요되는 시간에 대해 실무 경험을 바탕으로 합의된 값을 적용하였다.

Table 5. Example of initialization code insertion

CInitialApp.cpp

```
BOOL CInitialApp::Instance()
{
   ...
   m_pcMultilingual = new CMultilingual();
   m_pcMultilingual->setLanguageInfo(SW_ID);
   ...
}
```

DB 추가는 각 용어에 대한 매핑 정보를 정의하는 과정으로 Table 2와 같이 인덱스, 기준 용어, 변환 용어, SW ID 항목을 입력해야 한다. 본 성능 검증에서는 하나의 용어 정보 추가 시간을 15초로 산정하였으며, 이는 응용 프로그램의 변환 용어 수에 비례하여 증가한다.

용어 치환의 경우, 기존 방식은 용어를 직접 수정하는 단순 치환 구조로, 언어가 추가될 때마다 소스코드 수정과 빌드를 반복해야 한다. 반면 제안 기법에서는 초기화 클래스에서 단 한 번 생성된 다국어 지원 객체를 통해 모든 변환이 수행된다. 용어 변환 함수 TransTerm()는 이 싱글톤 객체에 저장된 맵 데이터를 참조하여 입력된 기준 용어에 대응하는 다국어 용어를 반환한다. 즉, 제안하는 기법에서도 최초에는 하드코딩 된 "용어"를 TransTerm("용어")로 소스코드를 교체하는 과정이 필요하다. 이는 기존 방식에서 용어를 직접 수정하는 것과 동일한 난이도를 가지므로, 양 방식 모두 용어 당 수정 시간을 10초로 산정하였다.

마지막으로 UI 및 포맷 수정은 언어별, 국가권별 차이를 반영하는 작업으로, 기존 방식은 단순히 값을 교체하면 되기 때문에 항목당 10초를 산정하였다. 제안하는 기법에서는 현재의 언어 정보를 확인하기 위한 함수 호출과 이를 활용하여 UI 및 포맷들을 나누기 위한 조건 분기 작성이 필요하므로 상대적으로 많은 시간이 소모되며, 해당 시간은 20초로 정의하였다. 실제 사례에서의 응용 프로그램별 UI 및 포맷 수정의 수는 Table 4에 정리되어 있다. 이 작업은 전체 작업 내에서 차지하는 소요 비용이 크지 않지만, 다국어 운용의 완성도를 위해 반드시 고려되어야 한다.

Table 6. Defined processing time per task for first language introduction

Task Type	Exist Method (sec)	Proposed Method (sec)
Initialization code t_{init}	-	60
Database (add new term) t_{db_t}	-	15/term
Term translation t_{term}	10/term	10/term
UI / Format modification t_{ui}	10/items	20/item

Table 6은 각 작업(Task type)별로 단위 소요 시간을 정의한 것이다. 이를 기반으로 총 작업 시간을 산출한다. 단위 시간은 소문자 t로, 해당 작업의 총 소요 시간은 대문자 T로 표기하여 구분하였다. 예를 들어, 용어 변환의 단위 시간은 t_{term}으로 정의되며, 변환에 소요되는 총 작업 시간은 T_{term}으로 표기된다. 이를 토대로 각 작업 별 작업 시간은 다음과 같이 표현할 수 있다.

$$T_{db\_t} = t_{db\_t} \times N_{db\_t} \tag{2}$$

$$T_{term} = t_{term} \times N_{term} \tag{3}$$

$$T_{ui} = t_{ui} \times N_{ui} \tag{4}$$

수식 (2)는 DB 생성에 걸리는 총 시간, 수식 (3)은 전체 용어 변환 소스코드 수정의 총 소요 시간을, 그리고 수식 (4)는 UI 및 포맷 시간을 나타낸다. N은 각 항목의 전체 개수를 의미한다. 추가로, 빌드와 신뢰성 시험을 개별 항목으로 정의하였으나 실제 작업에서는 두 과정이 연속적으로 이루어지므로 사실상 단일 작업으로 간주할 수 있다. 이에 따라 본 과정의 총 소요 시간을 수식 (5)와 같이 하나로 묶어 표현하였다.

$$T_{br} = T_{build} + T_{rel} \quad (5)$$

여기서 T_{build}은 빌드 시간을, T_{rel}은 신뢰성 시험 시간을 의미한다. 이를 토대로 응용 프로그램 X에 대한 다국어 지원 최초 반영 시 소요되는 총 작업 시간은 다음과 같이 표현할 수 있다. 기존 방식의 경우,

$$T_{exsit}(X) = T_{term} + T_{ui} + T_{br} \quad (6)$$

이고, 제안하는 기법은 초기화 및 DB를 갱신하는 과정이 추가되므로 다음과 같이 나타낼 수 있다.

$$T^{init}_{proposed}(X) = t_{init} + T_{db\_t} + T_{term} + T_{ui} + T_{br} \quad (7)$$

위의 정의를 실제 응용 프로그램 A~E에 적용한 결과는 Table 7과 같으며, Fig. 10은 이를 시각화한 그래프이다.

Table 7. Comparison of first language introduction time per application

App	Exist Method (sec)	Proposed Method (sec)
A	55,810	61,330
B	6,630	12,100
C	7,720	15,575
D	11,950	13,480
E	19,090	20,800

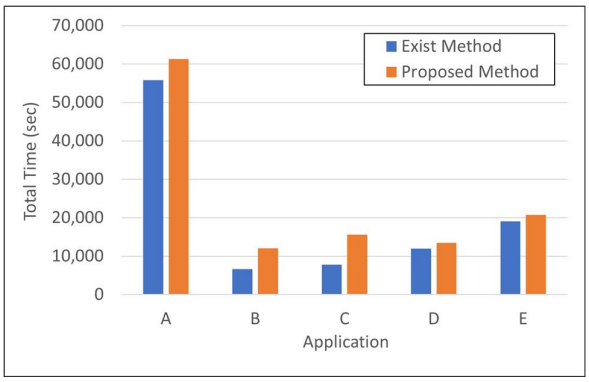

Fig. 10. Comparison of first language introduction time per application

분석 결과, 제안하는 기법은 최초 적용 단계에서 기존의 방식보다 많은 작업 시간이 소요되었다. 그러나 이는 초기화와 DB 갱신에 따른 일회성 비용이며, 필수 휘처인 초기화와 용어 변환은 한 번만 구현되면 이후 별도 수정 없이 재사용된다. 반면 UI 및 포맷 수정은 일부 응용 프로그램에서만 요구되는 선택 휘처로서 전체 비용에 미치는 영향은 제한적이었다. 따라서 제안하는 기법은 초기 도입 시에는 시간 비용 측면에서 불리하나, 장기적인 유지보수와 확장성 측면에서는 뚜렷한 우위를 제공한다.

2.2 Performance at additional language extension

추가 언어가 반영되는 경우를 대상으로 기존 방식과 제안하는 기법을 비교하였다. 본 절에서는 별개의 신규 용어의 추가는 고려하지 않으며, 새로운 지원 언어가 하나 추가된다는 전제를 둔다. 이때 제안하는 기법의 경우, 초기화 코드는 이미 작성되어 있으므로 수정이 필요하지 않고, DB에 새로운 언어 열을 추가하는 작업만 수행된다. 따라서 DB 작성 시간은 초기 도입 시보다 크게 감소하여, 용어당 5초로 정의하였다.

기존의 방식은 모든 용어에 대해 초기 다국어 수정 시와 마찬가지로 직접 소스코드 수정이 필요하며, UI 및 포맷 수정이 발생할 경우에도 동일하게 작업해야 한다. 반면 제안하는 기법은 용어 변환 코드 수정이 불필요하며, UI 및 포맷 변경이 전혀 발생하지 않는다면 빌드와 신뢰성 시험도 불필요하므로 소요 시간이 극적으로 단축된다. 또한 UI 및 포맷 변경이 발생하더라도 이미 작성된 현재 언어 확인과 분기 구조의 소스코드가 존재하므로 단순히 분기 내에 새로운 값을 추가하는 수준에 불과하다. 따라서 UI 및 포맷 항목 하나당 수정 시간을 10초로 산정하여, 초기 도입 대비 현저히 줄어든 비용을 반영하였다. Table 8은 산정한 작업별 소요 시간을 정리한 것이다.

Table 8. Defined processing time per task for additional language extension

Task Type	Exist Method (sec)	Proposed Method (sec)
Initialization code t_{init}	-	-
Database (add language column) t_{db_l}	-	5/term
Term translation t_{term}	10/term	-
UI / Format modification t_{ui}	10/item	10/item

수식 (2)~(4)와 마찬가지로 DB에 새로운 언어에 대한 열을 추가하는데 걸리는 총 작업 시간은 아래와 같이 표현할 수 있다.

$$T_{db\_l} = t_{db\_l} \times N_{db\_l} \quad (8)$$

이를 토대로 제안하는 기법에서 새로운 언어를 추가할 경우 걸리는 총 작업 시간은 수식으로 표현하면 다음과 같다.

$$T^{add}_{proposed}(X) = T_{db\_l} + \delta \cdot (T_{ui} + T_{br}) \quad (9)$$

$T^{add}_{proposed}(X)$는 DB에 신규 열 추가에 따른 기본 작업 시간과 UI 및 포맷 수정 발생 여부에 따른 추가 작업 시간을 구분하여 모델링한 수식이다. 이를 통해 UI 수정이 발생하지 않는 경우, 제안하는 기법의 작업 시간이 DB 갱신 작업에 국한됨을 정량적으로 나타낸다.

여기서 δ는 UI 및 포맷 수정 발생 여부를 의미하며, 수정이 발생한 경우 1, 그렇지 않은 경우 0으로 정의한다. 본 절에서 정의한 내용들을 응용 프로그램 A~E에 적용한 결과는 Table 9와 같다. 보다 명확한 비교를 위해, UI 및 포맷 수정이 발생하는 경우 응용 프로그램 A~E 모두에서 1건의 소스코드 수정이 이루어진다고 가정하였다. 응용 프로그램 A의 경우 실제 수정 건수는 6건이지만, 본 연구에서는 이를 1건으로 환산하였다. Table 8에서 보듯이 UI 및 포맷 수정은 건당 10초로 정의되므로, 6건을 1건으로 단순화하더라도 전체 수정 시간의 차이는 약 50초에 불과하다. Fig. 11은 이를 그래프로 시각화한 것이다.

Table 9. Comparison of additional language extension time per application

App	Exist Method (sec)	Proposed Method (UI change) (sec)	Proposed Method (No UI change) (sec)
A	55,810	53,960	1,800
B	6,630	4,830	1,800
C	7,720	5,125	2,595
D	11,950	11,470	490
E	19,090	18,550	550

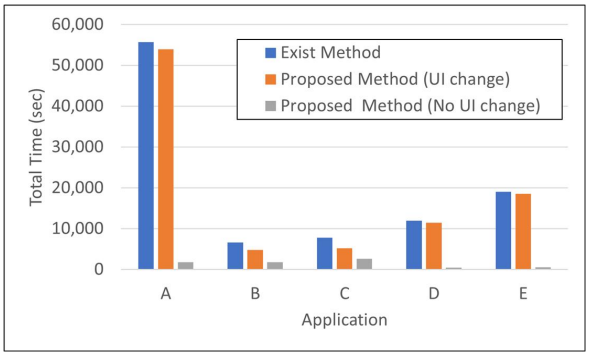

Fig. 11. Comparison of additional language extension time per application

결과적으로 기존의 방식은 언어가 추가될 때마다 선형적으로 증가하는 수정 비용을 감당해야 하며, 어떠한 경우라도 빌드와 신뢰성 시험이 필연적으로 수반된다. 또한, 본 연구에서 제시한 Table 9와 Fig. 11에서는 기존 방식에서의 UI 및 포맷 수정 발생 여부를 별도로 구분하지 않았다. 이는 하드코딩 구조상 1건에 대한 수정을 제외하고 나머지 수정 시간은 동일하므로, 총 수정 시간의 차이가 10초밖에 나지 않아 무의미하다고 판단했기 때문이다. 반면 제안하는 기법은 DB 입력이라는 일정한 비용만 발생하며, UI 및 포맷 수정이 존재하더라도 이미 구축된 분기 구조를 확장하는 형태이므로 비용이 크게 절감된다. 특히 응용 프로그램 D와 E처럼 UI 및 포맷 수정이 존재하지 않으면, 전체 소요 시간이 수백 초 단위로 줄어드는 등 극적인 효과를 보인다. 이는 제안하는 기법이 실제 운용 단계에서 유지보수 효율성을 극대화함을 실험적으로 증명한다.

2.3 Performance in source code modification

본 절에서는 새로운 기능적 요구나 용어의 추가·삭제와 같은 소스코드 수정 발생 상황을 대상으로, 기존 방식과 제안하는 기법의 성능을 비교하였다. 단순히 기존 용어를 다른 언어 표현으로 치환하는 경우는 5.2절에서 다룬 "UI 및 포맷 수정 없음" 시나리오와 중복되므로 간략히 언급만 하고, 소스코드 변경 자체가 필수적으로 요구되는 경우를 중심으로 기술한다.

기존 방식에서는 소스코드 내부에 다국어 표현이 하드코딩 되어 있기 때문에, 새로운 용어를 추가할 경우 모든 지원 언어별 소스코드 라인에서 동일한 수정이 필요하다. 예를 들어 동일한 기능에 대해 3개 언어를 지원한다면, 동일한 소스코드 수정뿐만 아니라 빌드 및 신뢰성 시험과 같은 작업이 세 배로 반복되어야 한다. 이는 언어 수에 비례하여 작업 시간이 선형적으로 증가함을 의미한다.

반면 제안하는 기법에서는 다국어 처리가 코드 내부가 아니라 외부 DB로 분리되어 있기 때문에, 소스코드 수정은 단일 프로젝트 레벨에서만 발생한다. 즉, 새로운 기능이나 용어가 추가되더라도, 언어 수와 무관하게 한 번의 코드 수정만으로 모든 언어 지원이 가능하다. 다국어 표현 자체는 DB의 새로운 행(Row) 또는 항목 추가로 처리되며, 이는 용어당 일정한 시간만 소요된다. 이를 수식으로 표현하면 다음과 같다.

$$T_{mod} = t_{term} \times N_{new} \quad (10)$$

$$T^{mod}_{exsit}(X) = L \cdot (T_{mod} + T_{br}) \quad (11)$$

$$T_{proposed}^{mod}(X) = T_{mod} + T_{br} + (N_{new} \times t_{db}) \quad (12)$$

여기서 T_{mod}는 신규 용어에 대한 소스코드를 수정하는데 걸리는 작업 시간이다. N_{new}는 추가되는 신규 용어의 수, L은 지원 언어 개수를 의미하며, t_{db}는 신규 용어를 DB에 추가하는데 걸리는 단위 시간으로, t_{db_t}와 t_{db_l}을 합한 것과 같다.

$T_{exsit}^{mod}(X)$과 $T_{proposed}^{mod}(X)$는 신규 용어 추가 시 기존 방식과 제안하는 기법 간 유지보수 작업 시간의 차이를 모델링한 수식이다. 기존 방식에서는 L에 비례하여 동일한 소스코드 수정과 빌드 및 신뢰성 시험 과정이 반복되는 반면, 제안하는 기법은 L과는 무관하게 소스코드 수정과 DB 갱신만으로 처리가 가능하다. 이에 따라 지원 언어 수가 증가할수록 두 방식 간의 작업 시간 격차가 확대됨을 수식으로 나타낸다.

기존 방식은 지원 언어 수 L에 비례하여 작업 시간이 증가하는 반면, 제안하는 기법은 전체 작업 시간에 큰 영향을 미치지 않는다. Table 10은 응용 프로그램 A~E가 3개의 언어를 지원할 때, 1개의 신규 용어를 추가하는 상황을 가정하여 총 수정 시간을 비교한 결과이며, Fig. 12는 이를 시각화한 것이다.

Table 10. Comparison of source code modification time per application

App	Exist Method (sec)	Proposed Method (sec)
A	156,480	52,180
B	9,090	3,050
C	7,590	2,550
D	32,940	11,000
E	54,000	18,020

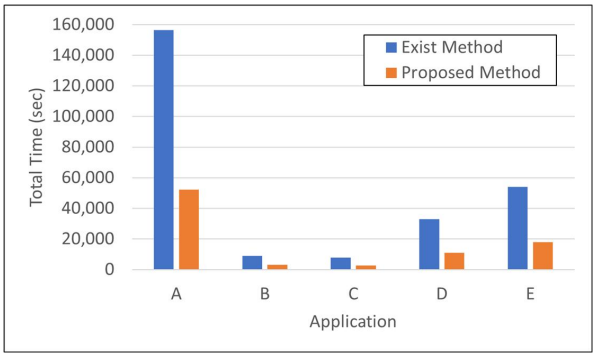

Fig. 12. Comparison of source code modification time per application

앞선 결과들에서 확인할 수 있듯이, 전체 작업 시간에 절대적인 영향을 미치는 것은 빌드 및 신뢰성 시험 시간이다. 기존의 방식은 지원 언어의 수만큼 이러한 고정 시간이 선형적으로 증가하였다. 반면 제안하는 기법은 빌드 및 신뢰성 시험의 과정을 반복하지 않고, 소스코드 수정 및 DB 갱신 작업만으로 관리가 가능하였다. 그 결과 지원 언어 수가 증가할수록 두 방식 간 격차는 크게 벌어졌으며, 특히 대규모 응용 프로그램일수록 절감 효과가 극대화되었다.

3. Results and discussion

앞선 세부 절에서 언어 지원의 최초 적용, 추가 언어 확장, 그리고 다국어 지원 상태에서의 소스코드 수정이라는 세 가지 주요 시나리오를 구분하여 성능 검증을 수행하였다. 그 결과, 기존 방식과 본 논문에서 제안하는 기법 사이에는 구조적인 차이로 인해 일관된 경향이 확인되었다.

우선 초기 도입 시에는 제안하는 기법이 초기화 코드 삽입, DB 구축 등으로 인해 초기 소요 시간이 발생하였지만, 전체 프로젝트 규모에 비해 미미한 수준이었다.

다음으로 언어 확장 시 기존 방식은 언어 수에 비례하여 수정, 빌드, 신뢰성 시험의 전체 절차를 반복해야 했다. 반면, 제안하는 기법은 DB의 언어 열 추가와 같은 단순 작업으로 충분하여, 지원 언어 수가 늘어날수록 두 방식 간의 시간 격차는 기하급수적으로 확대되었다. 특히, 신뢰성 시험이 수천에서 수만 초에 이르는 대규모 응용 프로그램에서는 이 차이가 절대적으로 크게 나타났다.

그리고 소스코드 수정이 발생한 경우에도 기존 방식은 모든 언어별로 동일한 소스코드 수정과 빌드·신뢰성 시험의 과정을 반복하였으나, 제안하는 기법은 단일 수정만으로 전체 언어에 반영할 수 있었다. 신규 용어가 포함되는 경우에도 DB에 용어를 추가하는 정도의 부담만이 발생하였으며, 이는 전체 절차와 비교하면 미미한 수준에 불과했다.

제안하는 다국어 지원 구조는 초기 실행 시 언어 매핑 데이터의 로딩 및 바인딩 과정으로 인해 일정 수준의 오버헤드가 발생할 수 있다. 그러나 해당 과정은 시스템 초기화 단계에서 한 번만 수행되며, 이후 운용 단계에서는 정적 바인딩 방식과 유사한 수준의 런타임 성능을 유지한다.

본 연구에서 사용하는 다국어 매핑 DB는 함정 전투체계의 다기능콘솔(Multi-Function Console, MFC)에 로컬 파일로 탑재되어 운용되는 환경을 전제로 설계되었다. 이에 따라 초기화 이후에는 빈번한 디스크 접근이나 추가 연산을 요구하지 않도록 구성되어, 제한된 자원 환경에서도 런타임 성능과 실시간성 요구를 저해하지 않는다.

이상의 결과를 종합하면, 제안하는 기법은 초기 단계에서 일정한 초기 작업 시간을 감수하더라도, 언어 확장과 유지보수 과정에서 두드러진 시간 절감 효과를 제공함이 확인되었다. 특히 다국어 지원이 필수적인 함정 전투체계와 같은 장기 운용 시스템에서는, 빌드·신뢰성 시험의 과정을 최소화함으로써 개발 비용을 크게 줄일 수 있다. 이는 곧 수출 경쟁력 강화와 유지보수 효율성 제고라는 두 가지 측면에서 실질적인 성과로 이어질 수 있음을 시사한다.

한편, 본 연구에서 제시한 작업 시간 기반 비교 결과는 단위 작업 시간에 대한 가정값을 기반으로 산출된 것이다. 실제 개발 환경이나 조직의 특성에 따라 각 작업 단계에서 소요되는 절대적인 시간 값은 달라질 수 있으며, 이에 따라 정량적 비교 결과의 수치 또한 변동될 가능성이 있다. 다만 동일한 기준 하에서 단위 작업 시간을 적용한 경우, 언어 확장 및 유지보수 과정에서 기존 방식과 제안 기법 간의 상대적인 작업량 차이와 경향은 유지될 것으로 판단된다. 따라서 본 연구의 결과는 절대적인 시간 값보다는 다국어 지원 구조에 따른 유지보수 효율성의 상대적 비교와 경향 분석에 초점을 두고 해석하는 것이 적절하다.

VI. Conclusions

본 연구에서는 함정 전투체계 소프트웨어에 다국어 지원 기능을 효율적으로 적용하기 위한 구조적 기법을 제안하였다. 기존의 언어별 프로젝트 복제나 하드코딩 방식은 관리 부담이 크고, 지원 언어 수가 증가할수록 산출물을 얻는 일련의 과정에 소요되는 시간이 선형적으로 증가하는 한계가 있었다. 이에 제안하는 기법은 DB 기반 구조, 동적 바인딩 기법, 휘처 모델을 통한 재사용성 관리 그리고 싱글톤 패턴을 통한 일관된 언어 관리 체계를 도입하였다. 이러한 구조적 접근은 다국어 지원을 체계적으로 통합하면서도 유지보수성을 크게 향상시킨다.

성능 검증 결과, 제안하는 기법은 초기 구축 시에는 상대적으로 많은 시간이 소요되지만, 일단 구조가 완성되면 언어 확장이나 용어 변경 시 소스코드 수정·빌드 과정 없이 DB 갱신만으로 대응할 수 있음을 확인하였다. 따라서 언어 수나 변경 횟수가 증가할수록 기존 방식과 비교한 효율성의 차이가 더욱 두드러진다. 이는 개발 비용 절감과 운용 효율성 제고라는 두 가지 측면에서 큰 장점을 제공한다.

또한 제안하는 기법은 단순히 개발 효율을 넘어, 해외 해군에 체계를 수출할 때의 현지화 요구 및 연합 훈련·다국적 운용 환경에서의 상호운용성 확보에도 기여할 수 있다. UI 전시와 국가권별 데이터 포맷을 선택적 기능으로 관리할 수 있어, 다양한 운용 환경에 유연하게 대응할 수 있다는 점이 그 근거다.

결론적으로 본 연구는 함정 전투체계의 다국어 지원을 위한 효과적인 소프트웨어 구조를 제시하였으며, 이를 통해 향후 수출 경쟁력 강화와 유지보수 비용 절감에 실질적인 기여가 가능함을 확인하였다. 향후 연구에서는 제안 기법을 실제 운용 환경에 적용하고, AI 기반 자동 언어 학습 및 번역 지원과 같은 확장 기능과의 결합 가능성을 탐색함으로써 다국어 지원의 범위와 활용도를 한층 넓힐 수 있을 것으로 기대된다.

ACKNOWLEDGEMENT

This work was supported by the Government of the Republic of Korea(KRIT) under the project titled “Development of main SW for global submarine combat system”(Project No. F230012).

REFERENCES

[1] SIMTOS.(2026), “Status and Trends of the Defense Industry,” https://simtos.org/kor/media/info_view.do?BIdx=6615

[2] Editorial Board, "Defense & Technology," Korea Defense Industry Association, Vol. 532, p. 20, Jun. 2023.

[3] National Strategic Information Portal(NSP), “Current Status and Future Tasks of Defense Industry Exports,” https://nsp.nanet.go.kr/plan/subject/detail.do?nationalPlanControlNo=PLAN0000043830

[4] Jeong-Gook Koh, “Design and Implementation of Multilingual support method for 3-tiered softwares. Journal of Korea Multimedia Society,” Journal of Korea Multimedia Society, Vol. 15, No. 2, pp. 266-272, Feb. 2012. DOI: 10.9717/kmms.2012.15.2.266

[5] Jin-yong Im, and Dong-seong Kim, "Performance Evaluation of Virtualization Solution for Next Generation Naval Combat Systems," Journal of The Institute of Electronics and Information Engineers, Vol. 56, No. 2, pp.41-49, Feb. 2019. DOI: 10.5573/ieie.2019.56.2.41

[6] Young-Dong Heo, "A Study on the Standardization of System Support Software in the Combat Management System," Journal of the Korea Society of Computer and Information, Vol. 25, No. 11, pp. 147-155, Nov. 2020. DOI: 10.9708/jksci.2020.25.11.147

[7] Jae-Geun Lee, "A Study on the Standard Architecture of Weapon

Control Software on Naval Combat System," Journal of the Korea Society of Computer and Information, Vol. 26, No. 11, pp. 101-110, Nov. 2021. DOI: 10.9708/jksci.2021.26.11.101

[8] Cheol-Hoon Kim, Dong-Han Jung, Young-San Kim, and Hyo-Jo Lee, "A Study on Standardization of IISS Software for Combat Interface Information Analysis of Naval Combat Management System," Journal of the Korea Society of Computer and Information, Vol. 29, No. 2, pp. 119-126, Feb. 2024. DOI: 10.9708/jksci.2024.29.02.119

[9] Kyung-Ho Bae, Tae-Wan Kim, and Chun-Hyon Chang, "Shared Data Structure using Dynamic Binding," Transactions on Programming Languages, Vol. 18, No. 3, pp. 50-58, Nov. 2004.

[10] Se-Hoon Kim, and Jeong-Ah Kim, "Consistency Checking Rules of Variability between Feature Model and Elements in Software Product Lines," Proceedings of KIIT Conference, pp. 519-524, Nov. 2012.

[11] Jun-Seok Park, and Geun-Hyuk Yeom, "View-Driven Architecture for Service-Oriented Software with Reusability," The Journal of Korean Institute of Next Generation Computing, Vol. 6, No. 3, pp. 29-40, Jan. 2010.

[12] Me-So Ko, and Yong-Jun Kim, "A Study on the Optimization of Memory Efficiency in Game Manager," Proceedings of KIIT Conference, pp. 543-546, Nov. 2023.

Authors

Ye-Jun Jang received the B.S. degrees in Information and Communication Engineering from Yeungnam University, Korea, in 2014. He is currently working in Hanwha Systems Co. from 2022.

He is interested in Naval Combat Management System, Design Pattern, Server Virtualization and so on.

Jae-Geun Lee received the B.S. degrees in Computer Engineering from Yeungnam University, Korea, in 2014. He is currently working in Hanwha Systems Co. from 2014. He is interested in Naval Combat Management System, Software Reuse, Design Pattern and so on