IDAP INSTITUTE OF DEFENSE ACQUISITION PROGRAM

JAMS JOURNAL OF ADVANCES IN MILITARY STUDIES

2021, Vol. 4, No. 3, 153-176. https://doi.org/10.37944/jams.v4i3.116

Real-time service integration of defense information system: defense REST API server reference model design analysis

Yongtak Park* · Doyoung Kim**

ABSTRACT

This study designs a reference model of the Defense REST API server based on the representational state transfer (REST) architecture style to present the most efficient, stable, and sustainable technical criteria for real-time service integration of defense information systems in Korea. The purpose of this component is to provide evidence to be stipulated as part of the Korean Defense Ministry's instructions and regulations, such as the Defense Interoperability Management Directive and the Interoperability Guide, and to support the development of the National Defense Interworking Technology and Interoperability. As the defense information system was subdivided and developed by the army, navy, air force, or business functions, interworking between information systems has become one of the most important factors. However, despite the need for advanced service integration and interworking, various interconnection service modules based on enterprise application integration (EAI), EAI hubs, and spokes were developed at a level that met local requirements (simple data transmission) without specific criteria for each network or information system. As a result, most of the interconnection modules currently in operation suffer from the absence of a technical spectrum, such as not meeting the military's demands for real-time interconnection and service integration, which increases with time. Therefore, this study seeks to identify the above problems by integrating the defense information system into one service and presenting a reference model of the defense REST API server to meet various real-time interworking requirements, analyze the technical basis, and pursue a model that fits military reality.

Keywords : defense information system, defense REST API, real-time service integration, defense interoperability management directive

* (First Author) Kwangwoon University, Department of Defense Acquisition Program, Ph.D. Candidate and Korea Institute for Defense Analyses, senior researcher, ytpark85@gmail.com, https://orcid.org/0000-0003-0801-9200

** (Corresponding Author) Kwangwoon University, Department of Defense Acquisition Program and Electrical & Biological Physics, Professor, kdoyoungnid@gmail.com, https://orcid.org/0000-0003-3798-6152

Ⅰ. 서론

대한민국 국군의 네트워크 중심 정보공유 환경(NCE : network centric environment) 중, 국방정보체계는 전장관리정보체계(지휘통제, 전투지휘, 군사정보체계), 자원관리정보체계(기획・재정, 인사・동원, 군수・시설, 전자행정, 군사정보지원, 상호운용성), 국방 M&S 체계(연습・훈련용, 분석용, 획득용) 등 분야별로 수십 가지의 크고 작은 정보체계들로 이루어져 있다. 또한, 필요에 따라 다양한 형태로 개발된 정보체계들은 각각 의존적인 관계에 있으며 그 수가 많아질수록 상호운용성 관점에서 유기적인 연동에 대한 중요성은 점점 커질 수밖에 없다(Yoo, Lee, & Shin, 2012).

국방정보체계의 연동개념은 크게 두 가지로 구분된다. 첫 번째는 업무적인 연관성보다는 순수하게 데이터 또는 정보의 전달만을 목적으로 하는 방식이며, 국방정보체계의 일반적인 연동 형태로 알려져 있다. 이 방식은 보통 enterprise application integration(EAI)[1] 및 EAI Hub & Spoke[2] 방식으로 구현되며 기술적 작동 메커니즘이 실시간 서비스 대응이 불가하다. 현재 군에서 운용 중인 대부분의 연동 모듈이 이에 해당한다. 두 번째는 각각 End-Node 사이에서 발생하는 단순한 형태의 데이터 송・수신뿐만 아니라 분산된 시스템을 하나의 업무 프로세스로 묶어 사용자는 마치 통합된 하나의 시스템을 실시간으로 사용하는 듯 느낄 수 있도록 하는 개념으로 representational state transfer(REST)[3] application programming interface(API)[4] 또는 simple object access protocol(SOAP)[5] 아키텍처 기반으로 실시간 서비스를 통합한다. 이러한 형태는 enterprise service bus(ESB) 아키텍처와 간혹 혼동되기도 하는데, 실제로 ESB보다 더 진보된 형태의 연동이며 전 세계의 수많은 IT 기업들이 REST를 활용하여 자사 서비스를 통합하고 있다.

이처럼 급변하는 대외적 상황에도 불구하고 아직도 대한민국 국방 분야의 연동체계는 단순히 데이터의 송수신을 목적으로 한 형태에서 벗어나지 못하고 있으며 실시간 서비스 통합 개념에 관한 의미 있는 연구는 진행되고 있지 않다. 또한, 민간영역의 빠른 변화로 인해 눈높이가 높아진 군 사용자들의 다양한 요구사항을 현 국방 연동시스템으로는 충족시키기 힘든 상황이며, 이제는 국방정보체계 연동 또한 서비스 통합 개념으로 패러다임을 확장해야 할 필요성에 직면해 있다. 이는 기존 시스템을 모두 폐기하는 방식의 변화는 아니며 국방연동체계를 이원화하여 대량의 데이터 레코드를 주기적으로 송수신해야 하는 배치(Batch) 성 작업의 경우 기존 연동시스템을 사용하고 여러 가지의 시스템을 하나로 묶어 업무를 통합해야 하는 상황에서는 REST를 도입하여 대응해야 한다.

---

1) EAI(Enterprise Application Integration) : 전사적 응용프로그램 통합, Point to Point 방식의 mesh topology를 사용

2) EAI Hub & Spoke : 전사적 응용프로그램 통합의 한 유형으로, star topology의 형태로 구현

3) REST(Representational State Transfer) : 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식

4) API(Application Programming Interface) : 애플리케이션 소프트웨어 및 서비스를 통합하는 Tool로 새로운 연결 인프라를 지속해서 구축할 필요 없이 서비스가 서로 커뮤니케이션할 수 있도록 도와주는 기능

5) SOAP(Simple Object Access Protocol) : HTTP, HTTPS, SMTP 등을 통해 XML 기반의 메시지를 컴퓨터 네트워크상에서 교환하는 프로토콜

국방 정보체계에 전력화된 연동시스템의 실례로, 전장망과 자원망에서 각각 표준으로 사용 중인 Korean message text format(KMTF) 2.0[6] 기반 integrated interoperability module(IIM),[7] 국방연동관리체계(DIMS : defense interconnection management system)와 자원망의 군수 연동을 담당하고 있는[8]국방군수통합(DELIIS : defense logistics integrated information system) 연동 모듈도 결국 EAI, EAI Hub & Spoke를 토대로 개발되어 약속된 대량의 데이터의 송·수신에는 강점이 있으나, 실시간으로 여러 서비스를 통합하는 데는 아키텍처의 태생적인 제약사항이 따른다.

본 연구는 앞서 언급한 국방연동의 문제점을 해결하고자 지난 2000년 Fielding(2000)의 Architectural Styles and the Design of Network-based Software Architectures에서 처음으로 언급된 후, 지금까지 범용 되어 온 REST API를 국방에 적용하여 실시간으로 서비스를 통합하기 위한 기술적 근거 및 타당성에 대해 논하고자 한다. 그 이유는, EAI와 같은 배치성 데이터 중심 아키텍처로는 기술의 작동 메커니즘 차이로 인해 현업에서 원하는 실시간 서비스 통합을 달성할 수 없으며 이러한 문제점을 해결하는 가장 효율적인 방법이 REST API이기 때문이다. 또한, 이러한 관점에서 현실에 맞는 국방 REST API Server 참조모델을 제안하여 수많은 국방 정보체계를 하나로 연결하고 자유롭게 서비스를 확장시켜 실시간 연동 및 서비스 관점의 통합을 충족시키는 기술적 근거를 분석하고, 논리적으로 그 타당성을 증명하는 과정을 통해 국방정보체계 연동 분야를 더욱 발전시킬 것이다.

Ⅱ. 국방연동 사례 및 REST API의 타당성 분석

본 장은 국방정보체계 실시간 서비스 통합을 위한 참조모델 설계에 앞서, 몇 가지 선행된 연동 전력화 사례, 선행연구, 상호운용성 관련 훈령 및 지침서 등 근거를 통해 국방정보체계 연동의 REST API 적용에 대한 타당성을 확인하고 가능성을 분석하고자 한다. 첫째, 국방정보체계 연동사업의 추진 경과(①~⑧)[9]를 살펴보면 현존하는 모든 연구와 사업이 EAI 또는 EAI-Hub & Spoke 형태로 실시간 서비스 통합이 아닌 단순 연동 모듈 위주로 발전해 왔고 비슷한 형태의 사업들이 점진적으로 영역을 확장하는 개념으로 개발이 진행되었음을 알 수 있다.

① 01년~08년 : 전장체계 연동 구축(KJCCS 등 5대 전장체계) / EAI

6) KMTF(Korean Message Text Format) : 한국형 메시지 텍스트 포맷

7) IIM(Integrated Interoperability Module) : KMTF 메시지 기반으로 개발된 통합연동모듈(전장망 사용)

8) DELIIS(defense logistics integrated information system) : 국방군수통합정보체계 Socket 방식, Webservice 방식, 망전환 방식(XML 형태의 파일 교환) 등의 자체 연동 모듈을 운용함

9) 국방정보체계 ○○○○서버 구축 1단계 1차(표준수립) 사업 산출물, KMTF 기반 국방정보체계 연동 고도화 연구

② 04년~09년 : 자원체계 연동 구축(군수 등 8대 자원체계) / EAI
③ 09년~11년 : 전장-자원체계 연동 구축(4대 전장 - 6대 자원체계) / EAI
- KJCCS[10] - 자원체계 : 09. 10.~10. 03.
- ATCIS[11] - 자원체계 : 11. 06~11. 12.
- KNCCS[12] - 자원체계 : 08. 05.~10. 12.
- AFCCS[13] - 자원체계 : 10. 08.~10. 12.

④ 11년~12년 : KMTF 2.0 통합연동모듈/평가도구 개발사업(통신사) / EAI
⑤ 13. 04.~10. : 국방정보체계 연동통합서버 구축방안 연구(국방부) / EAI-Hub&Spoke
⑥ 14. 01.~11. : 국방정보체계 연동통합서버 구축 개념연구 사업 / EAI-Hub&Spoke
⑦ 16. 01. 12. : 국방정보체계 연동통합서버 구축 1단계 1차 사업 / EAI-Hub&Spoke
⑧ 17년 : KMTF 기반 국방정보체계 연동 고도화 연구 / EAI

이 밖에도 대한민국 국방부는 Indigo EAI(메타빌드), Tmax(티맥스소프트), NCross EAI(모노시스) 등 정보체계 구축사업 때마다 필요에 따라서 연동과 관련된 다양한 민간 상용소프트웨어를 도입하였으나 이마저도 EAI 아키텍처를 벗어나지 못하였다. 둘째, 국방분야 선행연구를 살펴보면 2017년 국군지휘통신사령부 합동상호운용성기술센터에서 의뢰하여 ㈜도전하는사람들이 작성한 연구보고서 KMTF 기반 국방정보체계 연동 고도화 연구(DOSA, 2017)에서 국방연동의 장기과제로 서비스 통합 환경지원과 RESTful(REST의 구현체) 웹서비스 구축을 언급하는 내용이 일부 존재하지만, 그 후 현실적인 추가연구나 사업은 진행되지 않았다. 셋째, 민간의 REST에 관한 연구는 SOAP 기반 웹서비스와 RESTful 웹서비스 기술 비교(Park, Moon, Yoo, Jung, & Kim, 2010), REST API 기반의 통합 아키텍처 연구 사례(An, Park, Kim, & Lee, 2019)와 같이 REST 아키텍처 자체의 기술적 비교연구나 RESTful 웹서비스를 이용한 경량의 통합 커뮤니케이션 및 협업 서비스(Lim, 2011), 웹 애플리케이션 성능 분석을 위한 REST API 기반의 서버 구축(Kim, Park, Choi, Moon, 2018), REST-API 방식의 3자 전송을 이용한 컨테이너 기반의 클라우드 데이터 접근성 향상 기법(Lee, Seok, Moon, Hong, Kim, Kim, & Goo, 2021)처럼 각각 분야별 목적에 따른 구현체를 위한 수단으로써 연구가 진행되었다. 이처럼 민간분야 선행사례에도 본 연구의 주제인 '정보시스템 간 상호운용성 차원의 실시간 서비스 통합'을 중점으로 다룬 연구사례가 존재하지 않았다. 이는 REST라는 아키텍처 스타일의 원초적 존재 이유가 정보시스템 간 실시간 상호운용성을 위해 개발된 기술로 민간 분야에서 이미 과거부터 여러 방면에서 상용화되었기 때문이라고 볼 수 있다. 반면,

10) KJCCS(Korean Joint Command and Control System) : 합동지휘통제체계
11) ATCIS(Army Tactical Command Information System) : 육군전술지휘정보체계
12) KNCCS(Korea Naval Command Control System) : 해군지휘통제체계
13) AFCCS(Air Force Command Control System) : 공군지휘통제체계

국방정보체계의 발전상황은 민간과 상이하므로 이러한 문헌고찰을 통한 표준화의 기틀을 마련해야 할 필요성이 제기한다.

넷째, 현재 전장 및 자원정보체계에서 전력화되어 표준으로 운용 중인 대표적인 국방연동체계를 살펴보면 <Table 1>과 같다. 하지만, 본 연구에서 REST API를 통해 구현하고자 하는 실시간 웹서비스 및 개방형 서비스 통합과 거리가 있다는 것을 알 수 있다. 다만, 군수통합정보체계 연동 모듈의 웹서비스 라우팅처럼 부분적으로 REST API를 채용하여 기능을 구현하는 등의 사례는 존재하였으나 그 또한 REST API Server 구축을 토대로 서비스 통합을 제공한다고 볼 수는 없다. 다섯째, 현재 국방부가 판단하는 연동체계의 척도가 되는 국방 상호운용성 관리지시-연동지침서를 살펴보면, 국방정보화업무훈령의 국방정보시스템 분류 중 KMTF 메시지를 사용하는 전장관리체계 및 전력지원체계를 대상으로 그 문서가 작성되었기 때문에 IIM 연동모듈(KMTF 2.0 기반, EAI 방식) 이외에는 표준으로 거론된 연동방식이 존재하지 않는다. 상기한 사례 및 선행연구를 토대로 국방연동체계의 전반적인 발전 추이와 배경을 고찰하였을 때, 대한민국 국방부는 연동을 아직도 단순히 약속된 데이터를 주고받는 수단 정도로만 규정하고 있는 것으로 판단된다.

<Table 1> Status of defense interconnection modules

Division	IIM	DELIIS Interconnection Module		DIMS
	EAI	EAI	WebService Routing	EAI Hub & Spoke
system	KJCCS, ATCIS, KNCCS, AFCCS, AKJCCS, MIMS,[14] JFOS-K,[15] etc.	DELIIS, DMOBIS,[16] DRIS,[17] DEMIS, NDIFIS,[18] DRIMS,[19] etc.	CIMMS,[20] WMS, etc.	DMOBIS, NDIFIS, NDSTAT,[21] HUMINS,[22] etc.
merits	· standardization module registered in DCMS[23] · high availability	· fast interconnection speed · atypical structure · high availability	· real-time interconnection · system flexibility · neutral/Open Standard · high scalability	· large amount of data interconnection · flexible scheduling
demerits	· batch interconnection	· no flexible scheduling · batch interconnection	· not REST · not service integration	· low availability · batch interconnection

14) MIMS(Military Intelligence Management System) : 군사정보통합처리체계

15) JFOS-K(Joint Fire Operating System-Korea) : 한국형 합동화력운용체계

16) DMOBIS(Defense Mobilization Information System) : 국방동원정보체계

17) DRIS(Defense Requirement Information System) : 국방군수소요획득정보체계

18) NDIFIS(National Defense Integration Finance Information System) : 국방통합재정정보체계

앞서 확인한 바와 같이, 국방과 민간의 선행연구 및 전력화 사례에서는 본 연구에서 원하는 REST API를 통한 국방정보체계 서비스 통합에 대한 충분한 선례를 얻지 못하였다. 따라서, 본 절에서는 Roy Fielding의 박사학위 논문(Architectural Styles and the Design of Network-based Software Architectures)을 기초로 학문적인 REST API의 특성을 분석하고, 그 제약조건을 국방정보체계 연동시스템에서 적용 가능한지를 검증하고자 한다.

REST는 과거 HTTP/1.0 이후 어떻게 하면 HTTP 규격을 망가트리지 않고 웹(WWW)을 진보시킬 수 있을지에 대한 관련 전문가들의 고민에서 시작되었다. 대표적으로 REST의 창시자인 Roy Fielding은 본인의 박사 논문에 REST를 "분산 하이퍼미디어 시스템(WEB)을 위한 아키텍처 스타일(제약조건의 집합)" 또는 "인터넷상의 컴퓨터 시스템 간 상호운용성을 제공하는 방법"이라고 정

<Table 2> REST Characteristics Applicability of Defense Information System

Characteristics	Explanation	Possibility
Client-Server	· Separating the user interface concerns from the data storage concerns	Defense interconnection system is a web-service system that can be satisfied.
Stateless	· Communication must be stateless in nature, as in the client-stateless-server (CSS) style	
Cache	· In order to improve network efficiency, we add cache constraints to form the client-cache-stateless-server style	
Layerd System	· Client calls REST API Server only · Server consists of multiple layers	
Code-On-Demand	· Server sends and executes code to Client (optional)	
Uniform Interface	· Identification of resources : Identify resources through URI · Manipulation of resources through representations : Generation, update, and deletion of resources are manipulated through presentations of HTTP messages · Self-descriptive messages : Specifications for the message to interpret yourself · Hypermedia as the engine of application state (HATEOAS) : Application status is transferred using Hyperlink	Defense interconnection system maintains and synchronizes META information based on interconnection control documents (ICD), thereby satisfying self-descriptive messages.

Note. Adapted from Fielding (2000).

19) DRIMS(Defense IT Resource Information Management System) : 국방정보자원관리체계

20) CIMMS(Computer Integrated Maintenance Management System) : 통합정비관리체계

21) NDSTAT(National Defense Statistics System) : 국방통계포털

22) HUMINS(Human Resouroes Management Information System) : 국방인사정보체계

23) DCMS(Defense Common component Management System) : 국방공통컴포넌트 관리시스템

의하였고 이런 REST 아키텍처 스타일을 따르는 API를 REST API라고 하였다. 즉, 국방정보체계가 REST API를 통해 실시간 서비스 통합을 구축하기 위해서는 구조적으로 REST 아키텍처 스타일을 충족해야 하는데 <Table 2>를 통해 이것이 가능하다는 것을 정리하였다.

또한, 이러한 제약조건을 모두 만족시키며 시스템을 개발하는 것은 매우 어려운 일이며 조건을 만족시키지 못하였더라도 실시간 서비스 통합을 달성하지 못하는 것은 아니다. 하지만, Roy Fielding이 2008년 자신의 개인 블로그를[24] 통해 본인이 제시한 모든 조건을 충족하지 못하면 REST라 부르지 말 것을 권고한 점, 제시된 제약조건을 모두 충족해야 REST의 완성도가 높아지는 점을 고려할 때 국방부는 실시간 서비스 통합을 위해 REST를 활용한다면, 구축 시 최대한 많은 제약조건을 충족하도록 노력해야 한다.

특히, 그중에서도 <Table 3>은 Uniform Interface의 Self-descriptive messages로 REST의 가장 핵심적인 특성이며 대부분의 민간 REST 서비스들이 충족시키지 못하는 조건임에도 불구하고 국방연동은 이미 구축된 ICD META를 통해 이를 만족시킬 수 있다.

<Table 3> Example of Self-descriptive messages

No	Http Message	Comment
1	HTTP/1.1 200 OK [{ “op” : “remove”, “path” : “/a/b/c” }]	· There is no information to know the form of the message and no specification to interpret the body · It’s not self-descriptive messages
2	HTTP/1.1 200 OK Content-Type: application/json [{ “op” : “remove”, “path” : “/a/b/c” }]	· Content-Type is specified, but there is no specification to interpret the body · It’s not self-descriptive messages
3	HTTP/1.1 200 OK Content-Type: application/json-patch+json [{ “op” : “remove”, “path” : “/a/b/c” }]	· Content-type and specifications all exist · It’s self-descriptive messages

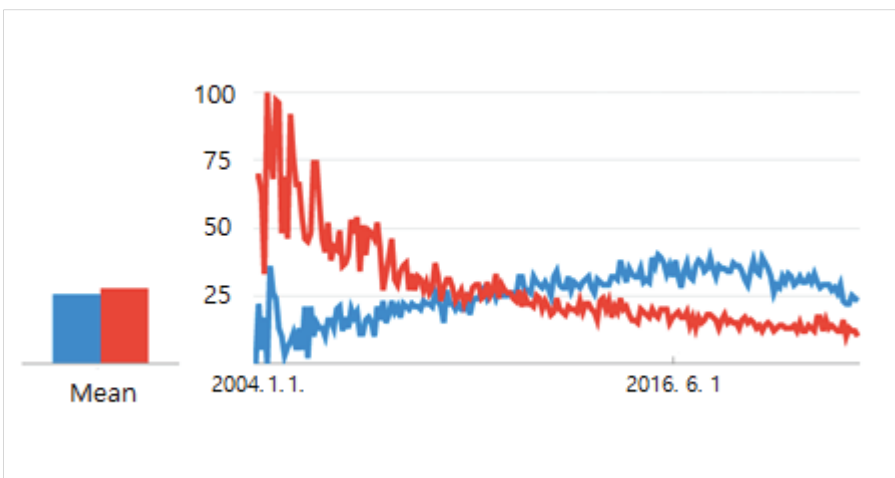
이처럼 국방정보체계의 상호운용성 환경은 REST API Server를 구축하기 위한 REST 아키텍처 스타일의 요구 조건을 모두 만족시킬 수 있으므로 실시간 연동 구축에 REST를 적용하는 것은 매우 적합하다 할 수 있다. 하지만, 방법의 다양성 측면을 고려해보면, 웹상에서 상호운용성을 지원하기 위해 꼭 REST를 사용하여야만 하는 것은 아니다. 그 외에도 HTTP 규격을 만족하면서 상호운용성을 구현하는 방법으로 SOAP(Simple Object Access Protocol)라는 대표적인 아키텍처가 존재한다(Khan & Abbasi, 2015). 그런데도 본 연구에서 REST만을 논하는 이유는 <Table 4>와 같이,

24) 출처 : https://roy.gbiv.com/untangled/2008/no-rest-in-cmis

REST는 SOAP에 비해 구조가 단순하고 규칙이 적으며 구현이 편리하고 메시지에 불필요한 over-head가 적어 상대적으로 데이터 parsing 및 송수신 효율이 높아 속도에도 장점이 있다(Lee, Lee, & Lee, 2009). 따라서, 현시점에 차세대 연동을 구축한다면, REST API가 가장 현명한 선택지가 될 수 있다. 또한, <Figure 1>[25]과 같이 구글 트랜드를 활용하여 전 세계 검색 빈도를 통한 점유율을 예측할 수 있는데, REST는 2000년 이후 현재까지 꾸준히 점유율을 늘려 해가 지날수록 SOAP보다 성장하고 있다는 것을 유추할 수 있다.

<Table 4> Compare characteristics of REST and SOAP

Div.	REST	SOAP
Structure	Simple	Complex
Rule	Less	Many
Difficulty	Easy	Difficult



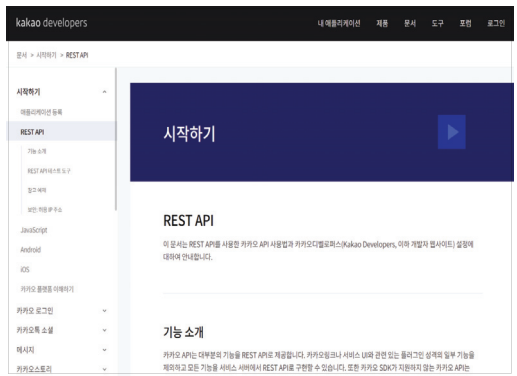
<Figure 1> Compare REST and SOAP interests (Blue=REST, Red=SOAP)

그리고 <Figure 2>,[26] <Figure 3>,[27] <Figure 4>,[28] <Figure 5>[29]처럼 네이버, 카카오 및 다국적 빅테크 기업 구글, 아마존 등도 모두 자사 서비스를 위한 개방형 REST API를 운용하여 플랫폼 서비스의 외연 확장에 심혈을 기울이고 있다. 결론적으로, 앞서 언급한 내용 들을 정리하면

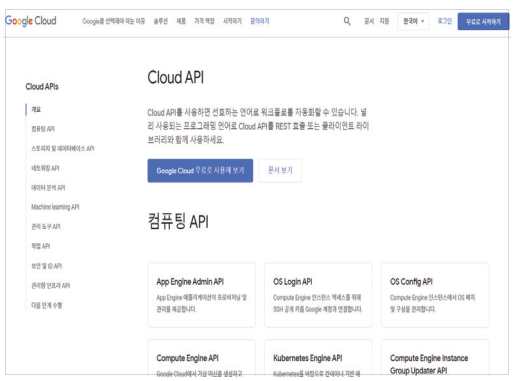
25) 출처 : Google Trends
26) 출처 : https://developers.naver.com/docs/common/openapiguide
27) 출처 : https://developers.kakao.com/docs/latest/ko/getting-started/rest-api
28) 출처 : https://cloud.google.com/apis
29) 출처 : https://docs.aws.amazon.com/ko_kr/apigateway/latest/developerguide

REST를 통한 국방정보체계 실시간 서비스 통합 구현에는 몇 가지 제약조건이 존재하지만, 국방연동 환경은 이를 충분히 만족시킬 수 있는 구조를 갖고 있다. 또한, REST는 같은 조건이라면 SOAP 등의 타 웹서비스 아키텍처 대비 성능과 효율성 측면에서 유리하며 세계적 점유율도 앞서있어 적용사례 등이 다양하므로 사용성이 이미 검증되었다. 따라서, 이러한 모든 상황을 고려했을 때 REST API가 국방연동 분야의 차세대 서비스 통합방안으로 가장 적합하다고 볼 수 있다.

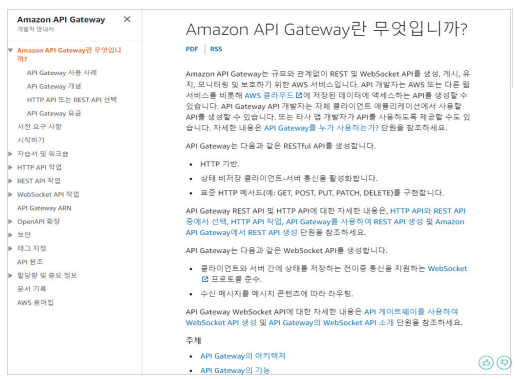
<Figure 2> Naver REST API



<Figure 3> Kakao REST API



<Figure 4> Google REST API



<Figure 5> Amazon REST API

Ⅲ. 연구방법

3.1 참조모델 설계 개요

지금까지 언급한 내용을 토대로 국방 REST API Server의 참조모델을 설계하고, 테스트 모듈(테스트가 가능한 최소한의 유닛)을 구현한다. 또한, bench marking test(BMT)를 실시하여 REST

API Server를 통한 실시간 서비스 통합에 대한 가능성을 증명한다. 참조모델 설계 방법은 ① 국방 연동시스템의 대표적 요구 조건 확인, ② 참조모델 설계 및 상세 설명, ③ 표준 메시지 포맷 설계 순서로 작성된다.

3.2 국방 연동시스템 요구 조건

우선 시스템의 참조모델 설계에 앞서 해당 모델에서 요구하는 조건에 대해서 알아볼 필요가 있다. 지난 국방 연동시스템 구축사업에서 작성한 KMTF 기반 국방정보체계 연동 고도화 연구(합동상호운용성기술센터 의뢰, 2017), 국방정보체계 ○○○○서버 구축 1단계 2차 사업 제안요청서(국방전산정보원, 2017), 국방정보체계 ○○○○서버 구축 1단계 2차 사업 요구사항 정의서(국방부, 2018), 국방상호운용성관리지시-연동지침서(국방부) 등 사업수행 자료들에 포함된 각 군 상호운용성 이해관계자들의 의견수렴(인터뷰 등) 과정을 참조하면 국방 연동시스템에서 만족해야 하는 조건들을 추출할 수 있다.[30]

위와 같은 상호운용성 이해관계자의 요구 조건에 Fielding(2000)이 제안한 REST API의 특성을 더하면 본 연구에서 추구하는 실시간 서비스 통합을 위한 최종 요구 조건을 아래와 같이 도출할 수 있다.

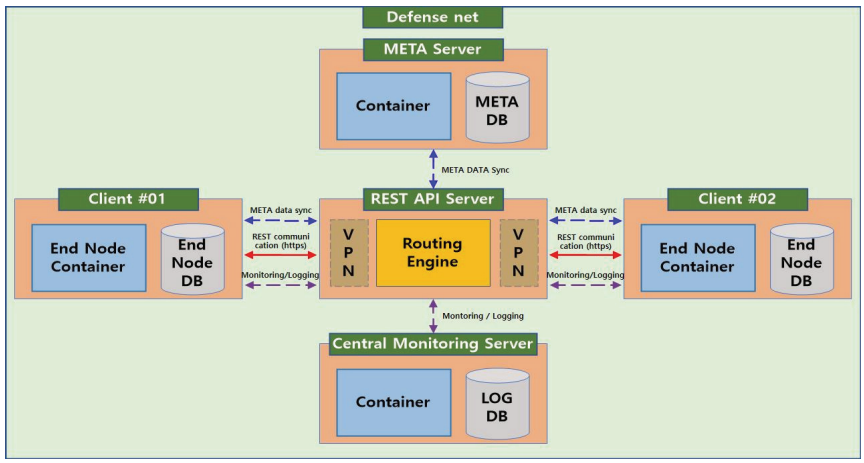
---

① REST API Server를 통한 정보체계들의 상호조건 없는 실시간 통합 및 확장
② Client 정보체계에 별도의 Adapter 설치 없이 모든 기능 지원
③ REST API의 다양한 프로그래밍 언어 지원(Java, C, C# 등)
④ 연동통제문서(ICD, Interface Control Description) 기반 연동자료 명세 META DATA 구축
⑤ End-Node to End-Node 간 구간별 모든 데이터 흐름을 Log 정보로 증명
⑥ 데이터 송수신 간 실제 메시지를 연동 서버에 저장하지 않음
⑦ 국방보안업무훈령 제34조(정보통신망 연동) 내용의 모든 조건 충족
⑧ 이 밖의 REST가 만족해야 하는 모든 조건을 충족

3.3 REST API 기반 실시간 서비스 통합 모델 설계

본 절은 3.2 절에서 정리한 요구 조건들을 토대로 참조모델을 설계하였다.

30) 군 사업수행 자료 특성으로 외부공개가 제한적인 보안 내용은 마스킹 처리함.



<Figure 6> Defense REST API Server Reference Model

<Figure 6>은 참조모델이고 <Table 5>는 상세 설명이다. 위 과정의 연동 대상이 되는 Client가 REST API Server를 통해 통합 서비스를 연계하는 순서는 다음과 같다.

① Source Server에서 REST API Server를 call

② META 정보를 통해서 메시를 해석하고 Validation Check

③ META 정보를 통해서 목적지 정보를 획득 후 Destination Server를 Call

④ Destination Server에서 메시지 수신 후 통합된 서비스 처리(Get or Put)

⑤ Destination Server에서 서비스 처리 후 약속된 결괏값을 실시간으로 Response

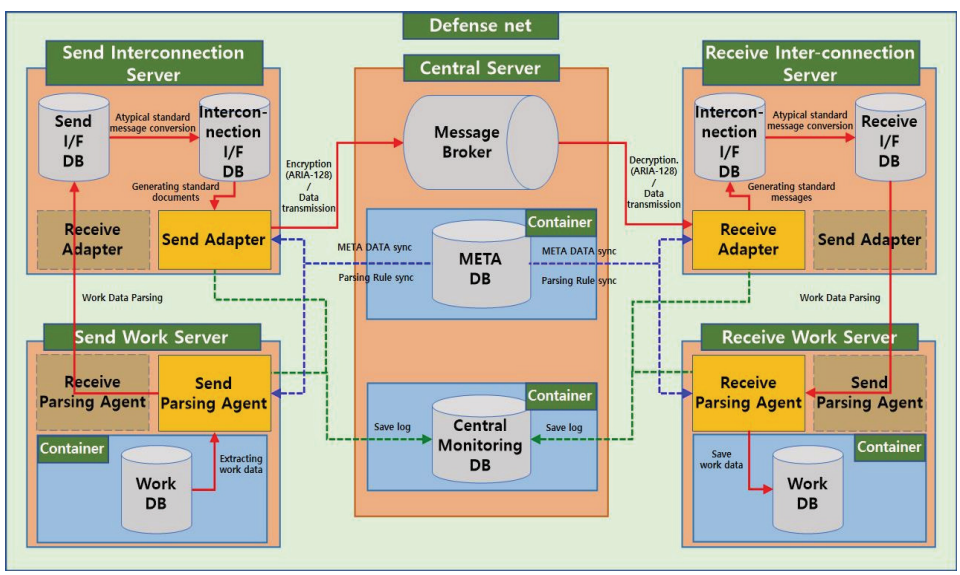
⑥ REST API Servers는 Destination Server로부터 받은 결과를 Source Server로 Routing

⑦ 최종적으로, Source Server가 결괏값을 받아 실시간으로 업무 처리

<Table 5> REST API Server details

Name	Explanation
META Server	Synchronize META information necessary for Interconnection to each node
REST API Server	When receiving a JSON Message from the Client, interpret and validate the message by referring to the WSDL specification of META Server and then route to the destination
Central Monitoring Server	Record the REST API Server and all logs performed in each End Node to the central control server and perform various controls, inquiries, and commands necessary for interworking
Client #01	Target system (End Node) integrated in real time through REST API Server
Client #02	

이 참조모델은 별도의 Adapter 없이 API를 통해 HTTPS 통신 서비스가 제공되므로 REST API가 지원하는 프로그래밍 언어에 대해서는 얼마든지 제약 없이 서비스를 확장할 수 있다. 또한, ICD를 기반으로 META 정보를 구축하여 Node 상호 간 전달되는 메시지에 대한 명세로 활용하여 REST의 난제인 Self-descriptive messages 특성을 충족한다. 그리고 Source Server에서 REST API Server, Destination Server까지의 흐름을 Central Monitoring Server를 통해 로깅 및 관제할 수 있어 무결한 연동 서비스가 가능하다.



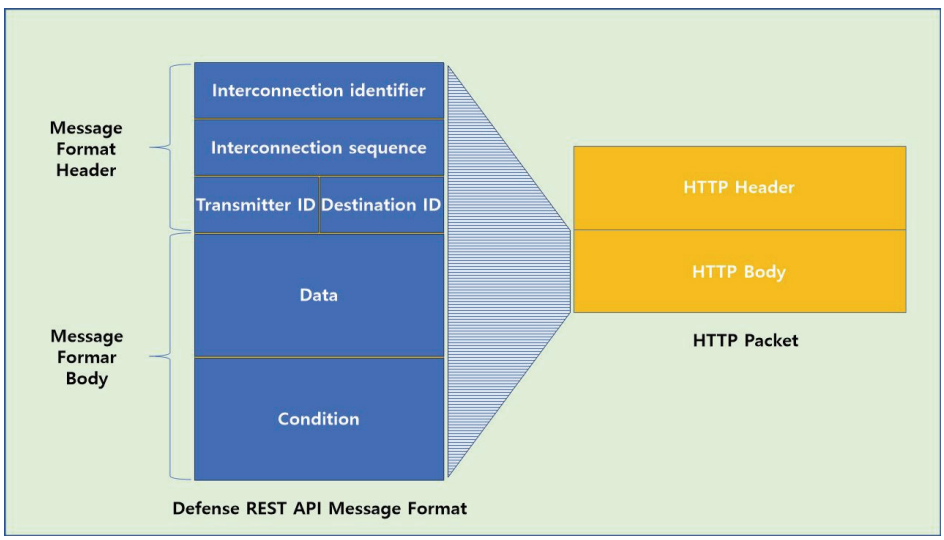
<Figure 7> Defense Interconnection System Structure in the form of EAI Hub & Spoke

위 모델을 보안 측면에서 보면, REST API Server는 META 정보를 통한 Validation Check 및 Routing 역할만 수행하므로 중앙서버에서 XML 형태로 송수신된 데이터를 보관하는 일부 타 연동 시스템 대비 보안에 유리하며, 국방보안업무훈령 제34조(정보통신망 연동)에서 요구하는 간접연동 구조를 만족하는 등 국방연동시스템으로서 조건을 대부분 충족하였다고 할 수 있다. 이것은 <Figure 7>을 통해 일반적인 국방 EAI Hub & Spoke 방식의 연동시스템과 비교해보면 확실히 그 차이를 알 수 있는데, 특히 상기한 요구 조건 중 ①, ②, ③, ⑥은 REST API Server이기 때문에 충족 가능한 조건이라 할 수 있다.

3.4 국방 REST API Message Format 설계

서로 다른 정보체계를 하나의 서비스로 통합하기 위해서는 연동을 위한 Message Format을 표

준화할 필요가 있다. 미리 약속하여 표준화된 메시지를 전달하지 않으면, 상대방이 아무리 메시지에 대한 명세서(META 정보)를 갖고 있다 하더라도 그것을 해석할 수 없기 때문이다. 국방 REST API는 HTTPS 기반의 POST 방식의 통신을 하며 HTTP Body에 JSON 형태의 메시지를 실어서 전송한다. 이때 HTTP Body에 들어가는 JSON 형태의 국방 REST API Message Format은 다음과 같다(Figure 8).



<Figure 8> Defense REST API Message Format

이 메시지는 Header(연동 식별자+연동 순번+송신처 ID+수신처 ID)와 Body(Data+Condition)로 구분되는데 연동 식별자는 ICD에서 정의된 유일한 연동문서를 의미하고, 연동 순번은 그 연동 식별자의 연동 순서로 이 두 가지 속성을 통해서 유니크한 연동 메시지를 식별할 수 있다. 그리고 송신처 ID, 수신처 ID는 메시지의 Routing 및 Logging 등에 활용되며, Body의 Data는 메시지에서 송수신하고자 하는 실제 내용이고 Condition은 대상 체계의 Method에서 사용될 parameter 값이다.

3.5 국방 REST API Server 유닛 테스트 환경

국방 REST API Server의 참조모델을 테스트할 수 있는 테스트 모듈을 구현하여 유닛 테스트를 하기에 앞서 테스트를 진행할 환경을 다음과 같이 구성하였다. 테스트 환경 및 조건에는 하드웨어 부분과 소프트웨어 부분이 있다(Table 6, Table 7). 하드웨어 환경은 서버급이 아닌 PC급의 환경을 구현하였으나, 소규모 유닛 테스트를 하기에는 충분한 환경이라고 판단된다. 또한, 소프트웨어 환경은 일반적으로 REST API Server를 구현하기 위해 가장 대중적인 환경을 구성하였다.

<Table 6> REST API Server & Client H/W Environment

Div.	Defense REST API Server	Defense REST API Client
CPU	Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz	Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz
MEMORY	DDR3 32.0GB	DDR4 32.0GB
NIC	Intel(R) 82579V Gigabit Network Connection	Realtek PCIe GbE Family Controller
Router	ipTIME T5008 1000Mbps LAN Interface	
Network	1000Mbps LAN	

<Table 7> REST API Server & Client S/W Environment

Div.	Defense REST API Server	Defense REST API Client
Language	Java	Java
Development ToolKit	JDK 1.7	JDK 1.7
FrameWork	Spring Boot Framework	Spring Boot Framework
Tool	Spring-tool-suite-4, 4.10.0.RELEASE	Spring-tool-suite-4, 4.10.0.RELEASE

3.6 국방 REST API Server 테스트 모듈 구현 및 테스트

본 절에서는 국방 REST API Server 참조모델의 구현체인 경량 테스트 모듈을 독립된 환경에서 구현하여 유닛 테스트(이하, 실험)를 진행한다. 이 실험은 Client와 REST API Server 모듈을 각각 구현한 뒤 Client가 REST API Server로 송신할 데이터를 REST 아키텍처에서 사용하는 JSON과 SOAP에서 사용하는 XML 형식으로 실 업무에서 발생할 수 있는 크기의 데이터를 임의로 발생시켜 송신했을 때, 응답까지 걸리는 시간을 비교하여 상대적으로 REST API가 타 웹서비스 방식보다 비교우위에 있다는 것을 증명한다. 또한, 기존 legacy 연동 시스템(EAI, Hub & Spoke)과 웹서비스(REST API, SOAP)는 서로 지향하는 바가 다른 시스템이므로 이 둘의 비교실험은 별도로 실행하지 않고, 본 연구의 주제에 부합하는 실시간 서비스 통합을 위한 아키텍처인 REST와 SOAP 중에서 더 효율적인 방법을 찾는 실험으로 국한한다.

실험을 위해 구현된 테스트 모듈의 핵심 코드는 <Figure 9>, <Figure 10>, <Figure 11>과 같다. 각 코드는 순서대로 ① REST API Server로 전송할 임의의 데이터를 REST 방식에서 사용하는 JSON 그리고 SOAP 방식에서 사용하는 XML의 형태로 원하는 크기만큼 자동생성, ② 준비된 데이터를 REST 또는 SOAP 방식으로 Client에서 REST API Server로 송신 후 response time 측정, ③ JSON 또는 SOAP 방식으로 전달된 데이터를 각 형태에 맞게 파싱 후 성공 결과를 반환하는 기능을 한다. 다만, 여기서 REST API의 비교군(比較群)인 SOAP를 실제로 구현하기 위해서는

UDDI(universal description, discovery, and integration)와 WSDL(Web Services Description Language)을 전부 웹서비스 방식으로 구현해야 하지만, 그렇게 되면 모듈 테스트의 범위를 벗어남으로 본 실험에선 REST와 동일한 환경에서 데이터의 형태만 SOAP에서 사용되는 XML의 형태를 사용하여 데이터 크기 및 응답속도를 비교하는 방식으로 대체하였다.

```
List<HashMap<String, Object>> jsonList = new ArrayList<HashMap<String, Object>>();
HashMap<String, Object> valueMap = new HashMap<String, Object>();

String dataType = "JSON";          // JSON or XML
int sendJsonObjectyDataNum = 100; // 데이터 컬럼 갯수
int sendJsonArrayDataNum = 1850;  // 데이터 로우 갯수
int loopCount = 10;               // RESP API Server에 Request 요청 반복 횟수
int responseCode = 0;             // 응답수신코드

ObjectMapper mapper = new ObjectMapper();
String sendJsonStringData = null;

// JSON DATA 생성
for(int i=1; i<=sendJsonArrayDataNum; i++) {
    for(int j=1; j<=sendJsonObjectyDataNum; j++) {
        valueMap.put("data_key"+j, "data_value"+j);
    }
    jsonList.add(valueMap);
}

sendJsonStringData = mapper.writeValueAsString(jsonList);
logger.debug("sendJsonStringData : " + sendJsonStringData);

// XML Converting(JSON to XML, 정확한 비교를 위한 동일한 문자열 사용)
XMLSerializer xmlSerializer = new XMLSerializer();

xmlSerializer.setTypeHintsEnabled(false);

JSON json = JSONSerializer.toJSON(sendJsonStringData);
String sendXmlStringData = xmlSerializer.write(json);

int jsonByte = sendJsonStringData.getBytes().length; // getBytes("euc-kr") or getBytes("UTF-8")
int xmlByte = sendXmlStringData.getBytes().length;   // getBytes("euc-kr") or getBytes("UTF-8")

logger.debug("[JSON TYPE DATA]");
logger.debug(sendJsonStringData);

logger.debug("[XML TYPE DATA]");
logger.debug(sendXmlStringData);

logger.info("JSON Byte Size : " + jsonByte + " Byte");
logger.info("XML Byte Size : " + xmlByte + " Byte");
```

<Figure 9> JSON/XML data generation code (Client Code)

```
if("JSON".equals(dataType)) {
    logger.info("JSON TYPE");
    REST_SRV_PARAM.put("type", "JSON");
    REST_SRV_PARAM.put("data", sendJsonStringData);
} else {
    logger.info("XML TYPE");
    REST_SRV_PARAM.put("type", "XML");
    REST_SRV_PARAM.put("data", sendXmlStringData);
}

HashMap<String, Object> resultMap = new HashMap<String, Object>();
//List<HashMap<String, Object>> resultList = new ArrayList<HashMap<String, Object>>();

for(int i=0; loopCount>i; i++) {
    Date startDate = new Date(System.currentTimeMillis()); // 현재시간을 가져와 Date형으로 저장한다
    SimpleDateFormat dateformat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
    String startDateString = dateformat.format(startDate);
    logger.debug("REST API Server call time : " + startDateString);

    conn = (HttpURLConnection) url.openConnection();

    conn.setRequestMethod("POST");                          // Request 형식 설정(POST, GET, PUT, DELETE 가능)
    conn.setRequestProperty("User-Agent", USER_AGENT);      // User-Agent
    conn.setDoOutput(true);                                 // POST 파라미터 전달을 위한 설정
    conn.setRequestProperty("Content-Type", "application/json");  // Content-Type
    conn.setRequestProperty("x-auth-token", "ABCDE");       // REST API 사용권한 토큰

    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(conn.getOutputStream()));
    bw.write(REST_SRV_PARAM.toString());
    bw.flush();
    bw.close();

    responseCode = conn.getResponseCode(); // 송신 후 결과값 수신

    Date endDate = new Date(System.currentTimeMillis()); // 현재시간을 가져와 Date형으로 저장한다
    String endDateString = dateformat.format(endDate);
    logger.debug("REST API Server response time : " + endDateString);

    long sndRcvResponseTime = endDate.getTime() - startDate.getTime(); // millisecond
    sndRcvResponseTime = Math.abs(sndRcvResponseTime); // 절대값
    //sndRcvResponseTime = sndRcvResponseTime / 1000; // 기본값(밀리세컨드) 나누기 1000 = 초, 기본값(밀리세컨드) 나누기 (24(시)*60(분)*60(초)*1000) = 일
    logger.info("sndRcvResponseTime_"+(i+1) + " : " + sndRcvResponseTime + " ms");

    resultMap.put("sndRcvResponseTime_"+i, sndRcvResponseTime);
}

long sndRcvResponseTimeSum = 0;

for(int i=0; resultMap.size()>i; i++) {
    sndRcvResponseTimeSum = sndRcvResponseTimeSum + Integer.parseInt(resultMap.get("sndRcvResponseTime_"+i).toString());
}

logger.info("sndRcvResponseTimeSum : " + sndRcvResponseTimeSum + " ms");

if (responseCode == 400) {
    System.out.println("400:: 해당 명령을 실행할 수 없음");
} else if (responseCode == 401) {
    System.out.println("401:: X-Auth-Token Header가 잘못됨");
} else if (responseCode == 500) {
    System.out.println("500:: 서버 에러, 문의 필요");
} else { // 성공 후 응답 JSON 데이터받기
    BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    StringBuilder sb = new StringBuilder();
    String line = "";

    while ((line = br.readLine()) != null) {
        sb.append(line);
    }

    responseJson = new JSONObject(sb.toString());
}
```

<Figure 10> Data transmission/Response Time measurement (Client Code)

```
if("JSON".equals(type)) {
    logger.debug("JSON Type");

    dataSting = dataObject.toString();
    dataSting = dataSting.replaceAll("\\\\","");
    //jsonDataSting = jsonDataSting.replace("\\","");

    logger.debug("request type : " + type);
    logger.debug("request dataSting : " + dataSting);

    try {

        // JSON to String
        ObjectMapper mapper = new ObjectMapper();
        List<Map<String, Object>> readList = null;

        readList = mapper.readValue(dataSting,
                new TypeReference<List<Map<String, Object>>>() {
        });

        for(int i=0; readList.size() > i; i++) {
            for(int j=1; readList.get(i).size() >= j; j++) {
                logger.debug("data_key"+j+" : " + readList.get(i).get("data_key"+j).toString());
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
} else {
    logger.debug("XML Type");

    dataSting = dataObject.toString();
    dataSting = dataSting.replace("\\r","");
    dataSting = dataSting.replace("\\n","");
    dataSting = dataSting.replaceAll("\\\\","");

    XMLSerializer xmlSerializer = new XMLSerializer();

    // XML to JSON
    JSON jsonData = xmlSerializer.read(dataSting);
    dataSting = jsonData.toString();

    logger.debug("request type : " + type);
    logger.debug("request dataSting : " + dataSting);

    try {

        // JSON to String
        ObjectMapper mapper = new ObjectMapper();
        List<Map<String, Object>> readList = null;

        readList = mapper.readValue(dataSting,
                new TypeReference<List<Map<String, Object>>>() {
        });

        for(int i=0; readList.size() > i; i++) {
            for(int j=1; readList.get(i).size() >= j; j++) {
                logger.debug("data_key"+j+" : " + readList.get(i).get("data_key"+j).toString());
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

<Figure 11> Data Parsing (REST API Server Code)

Ⅳ. 분석결과

본 장에서는 Ⅲ장에서 구현한 테스트 모듈을 통해 실험을 진행한 결과를 분석한다. 실험방법은, 서로 다른 독립변인을 가진 두 집단을 설정하고(REST API, SOAP) 독립변인의 변화에 따라 결정되는 두 집단의 종속변인을 비교하여 어떤 집단의 방법이 데이터 전송에 얼마나 더 효율적인지를

판단하는 방법으로 진행하였다. 간단하게 설명하면, 같은 데이터를 각각 REST API와 SOAP 방식으로 전송하여 수집된 응답속도를 비교한 뒤 어떤 방식이 더 나은 연동방식인지 알아보는 실험이다. 또한, 본 실험에 사용된 변인을 정의하면 다음과 같다.

- 독립변인 : 데이터의 크기(독립변인이 다른 두 집단 : REST API, SOAP)
- 종속변인 : 응답속도
- 조절변인 : 테스트 환경(H/W 및 네트워크)

REST API와 SOAP에서 발생하는 데이터는 같은 데이터를 발생시키더라도 <Table 8>에서 확인할 수 있듯이 두 아키텍처의 형식 차이로 실제 데이터 크기는 서로 다르게 생성된다. 또한, 같은 이유로 종속변인인 응답속도에 가장 큰 영향을 주는 독립변인으로 데이터 크기를 설정하였다. H/W 및 네트워크 환경은 독립변인과 함께 종속변인의 결과에 영향을 주게 되지만 독립변인인 데이터 크기에 비해 그 영향도가 미미한 수준이기 때문에 본 실험에서는 조절변인으로 설정하였다.

<Table 8> REST API Server & Client S/W Environment

JSON	XML
[{"data_key1":"data_value1","data_key2":"data_value2","data_key3":"data_value3"},{"data_key1":"data_value1","data_key2":"data_value2","data_key3":"data_value3"}]	<?xml version="1.0" encoding="UTF-8"?> <dataList> <dataMap> <data_key1>data_value1</data_key1> <data_key2>data_value2</data_key2> <data_key3>data_value3</data_key3> </dataMap> <dataMap> <data_key1>data_value1</data_key1> <data_key2>data_value2</data_key2> <data_key3>data_value3</data_key3> </dataMap> </dataList>

본 실험은 5MB의 데이터를 임의로 생성하여 REST와 SOAP 방식으로 각각 서버에 데이터를 전송하는 테스트를 하였다. 데이터 크기를 5MB로 결정한 이유는 일상 업무 영역에서 주로 빈번하게 발생하는 크기이며 비교가 가능한 수준의 데이터가 필요하기 때문이다. 만약, 현업에서 발생하지 않는 아주 작은 크기의 데이터를 독립변인으로 활용하면 조절변인에서 발생하는 변수보다 독립변인의 영향이 작아질 수 있어 정확한 실험을 할 수 없으므로 가장 적당한 데이터 크기는 5MB 수준이라고 판단하였다. 본 연구의 참조모델을 구현한 테스트 모듈을 통해 획득한 종속변인 결과로 실험의 개발도구이자 실험 도구인 Eclipse의 console 창에서 결과를 추출하였다(Figure 12-13).

같은 문자열을 같은 조건에서 각각의 방식으로 10번 테스트 결과, Figure 12-13과 같은 실험 결과를 얻을 수 있었다. REST API 방식으로 약 5MB의 데이터를 전송했을 때 평균 응답속도는 65,301ms이고 SOAP 방식으로 약 6.65MB의 데이터를 전송했을 때 평균 응답속도는 96,532ms이다.

```
GET "/callRestApiClient?secretKey=ABCDE", parameters={masked}
Mapped to public java.lang.String com.restapi.home.RestApiClientController
Pre Interceptor
Start @RequestMapping-callRestApiServer
JSON Byte Size : 5154101 Byte
XML Byte Size : 6813599 Byte
JSON TYPE
sndRcvResponseTime_1 : 61615 ms
sndRcvResponseTime_2 : 62766 ms
sndRcvResponseTime_3 : 63398 ms
sndRcvResponseTime_4 : 66213 ms
sndRcvResponseTime_5 : 66151 ms
sndRcvResponseTime_6 : 66163 ms
sndRcvResponseTime_7 : 65908 ms
sndRcvResponseTime_8 : 66469 ms
sndRcvResponseTime_9 : 67509 ms
sndRcvResponseTime_10 : 66816 ms
sndRcvResponseTimeSum : 653008 ms
Using 'text/html', given [text/html, application/xhtml+xml, image/avif, ima
Writing ["{"success":true}"]
Post Interceptor
After Interceptor
Completed 200 OK
```

<Figure 12> REST API method (JSON DATA) transmission result

```
GET "/callRestApiClient?secretKey=ABCDE", parameters={masked}
Mapped to public java.lang.String com.restapi.home.RestApiClientController
Pre Interceptor
Start @RequestMapping-callRestApiServer
JSON Byte Size : 5154101 Byte
XML Byte Size : 6813599 Byte
XML TYPE
sndRcvResponseTime_1 : 92936 ms
sndRcvResponseTime_2 : 94965 ms
sndRcvResponseTime_3 : 94053 ms
sndRcvResponseTime_4 : 92620 ms
sndRcvResponseTime_5 : 93309 ms
sndRcvResponseTime_6 : 100579 ms
sndRcvResponseTime_7 : 99859 ms
sndRcvResponseTime_8 : 98198 ms
sndRcvResponseTime_9 : 100811 ms
sndRcvResponseTime_10 : 97987 ms
sndRcvResponseTimeSum : 965317 ms
Using 'text/html', given [text/html, application/xhtml+xml, image/avif, ima
Writing ["{"success":true}"]
Post Interceptor
After Interceptor
Completed 200 OK
```

<Figure 13> SOAP method (XML DATA) transmission result

여기서 두 방식의 데이터 용량 차이가 발생하는 이유는 위에서 여러 번 언급한 것처럼 같은 문자열을 사용하더라도 데이터의 형식 차이에서 발생하는 오버헤드로 인해 REST API 대비 SOAP의 데이터 크기가 클 수밖에 없기 때문이다. <Table 9>의 실험 결과를 정리하면, REST API가 SOAP 대비 용량은 75.64% 수준으로 24.36% 작고 시간은 67.68% 수준으로 32.32% 적게 소모한다는 것을 알 수 있다. 그 결과, SOAP에 비해 REST API가 각각의 수치만큼 효율적이라는 실험 결과를 도출할 수 있다.

<Table 9> Results of REST API Server & Client S/W Environment

No.	REST API		SOAP		REST compared to SOAP	
	Volume (byte)	Time (ms)	Volume (byte)	Time (ms)	Volume (%)	Time (%)
1	5,154,101	61,615	6,813,599	92,936	75.64	66.30
2	5,154,101	62,766	6,813,599	94,965	75.64	66.09
3	5,154,101	63,398	6,813,599	94,053	75.64	67.41
4	5,154,101	66,213	6,813,599	92,620	75.64	71.49
5	5,154,101	66,151	6,813,599	93,309	75.64	70.89
6	5,154,101	66,163	6,813,599	100,579	75.64	65.78
7	5,154,101	65,908	6,813,599	99,859	75.64	66.00
8	5,154,101	66,469	6,813,599	98,198	75.64	67.69
9	5,154,101	67,509	6,813,599	100,811	75.64	66.97
10	5,154,101	66,816	6,813,599	97,987	75.64	68.19
avg.	5,154,101	65,301	6,813,599	96,532	75.64	67.68

V. 결론 및 논의

2000년대 초반 이후 대한민국 국군은 국방정보체계의 상호운용성 확보와 이러한 기술의 표준화를 위해 수많은 노력을 해왔다. 하지만 그러한 노력에도 불구하고 단순 데이터 연동에 초점을 둔 비슷한 형태의 연동 모듈 개발에만 몰두한 결과 궁극적인 실시간 서비스 통합을 위한 자체 기술확보와 표준화는 이루지 못하였다. 중요한 것은 이러한 상황에서 지속된 연동사업은 변화한 시대 흐름을 반영하지 못하였고 국방부는 선진화된 표준이 부재한 상황 속에 정보체계별로 별도의 연동사업을 중복 투자 및 전력화하여 불필요한 시간과 비용의 소모를 반복해 왔다. 본 연구는 이러한 문제점들을 극복하고 국방정보체계를 하나의 실시간 서비스처럼 유연하게 통합하는 방안으로 REST API Server 구축과 관련 기술의 표준화를 제시하였고 참조모델 설계 및 실제 구축을 통한 유닛테스트(실험)를 수행하여 그 효율성 및 가능성을 증명하였다. 실험은 동일 환경에서 앞서 제안한 국

방 REST API와 일반적인 SOAP 연동 서비스를 각각 구축한 뒤 같은 데이터를 전송하여 발생 되는 데이터양과 소요 시간을 반복 측정하여 평균값을 비교한 결과 국방 REST API가 통신량은 24.36% 그리고 속도는 32.32% 효율적이라는 결과를 도출하였다. 특히, 용량과 소요 시간 측면의 효율성은 네트워크 상황이 민간 대비 상대적으로 느린 국방정보체계에는 매우 결정적인 포인트라 할 수 있다. 따라서, 만약 국방정보계에서 서비스 통합에 대한 소요가 발생한다면 본 연구에서 제안한 국방 REST API Server 참조모델을 적용해야 가장 효율적이라는 것이 증명되었다. 다만, 본 실험은 국방 REST API Server 참조모델을 활용한 서비스 통합의 가능성과 효율성을 증명하기 위한 여러 가지 요소(성능, 보안, 정책 등) 중 성능 부분에 국한된 실험이지만, 기술적 가능 여부를 증명한 뒤 다음 단계의 연구로 확장하기 위해 충분히 가치 있는 성과였다.

또한, 본 연구성과가 국방상호운용성관리지시, 연동지침서 등의 국방부 지시와 규정에 명문화되어 기술적 표준을 정립하게 된다면, 군의 차기 서비스 통합 및 실시간 연동이 요구되는 정보화사업 수행 시 명확한 기준안 제공을 통해 기술의 주도권을 가진 상태에서 개발사업이 수행될 수 있으므로 불필요한 물적·인적자원의 낭비를 막을 수 있을 것이다. 본 연구는 그동안 대한민국 국방 정보체계 분야의 실시간 서비스 통합을 위한 학술적·실무적 노력이 부족했던 실정에서 국방연동기술의 개선 발전을 위한 다양한 기술적 스펙트럼이 적용된 표준 확보의 필요성을 제기하였다는 점에서 연구의 의의가 있다. 끝으로, 해당 연구는 단계적으로 계속하여 보안기술(양방향 암호화 및 키 관리, VPN 및 HTTPS 적용), 메타정보 관리, 로깅 및 데이터 추적, 중앙관제, 웹서비스 라우팅 등 실무에 적용 가능한 수준의 연구·설계 및 구현을 통해 기술적 근거를 제시할 것이며 더 나아가 블록체인, 빅데이터, 클라우드 등 4차 산업혁명 시대에 도달하기 이전 단계의 교두보로서 국방정보체계 발전에 큰 의미가 있는 연구이다.

Acknowledgements

We would like to thank Editage (www.editage.co.kr) for English language editing.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Reference

An, H. S., Park, B. K., Kim, Y. C., & Lee, J. H. (2019). Research practices on Integrated Architecture based on REST API. *Journal of Platform Technology, 7*(1), 3-9. https://www.kci.go.kr/kciportal/ci/sereArticleSearch/ciSereArtiView.kci?sereArticleSearchBean.artiId=ART002522052

DOSA (2017). A Study on the Advanced Interconnection of Defense Information System based on KMTF, 15-17, 99.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* [Doctoral dissertation]. University of California, Irvine.

Khan, M. W., & Abbasi, E. (2015). Differentiating Parameters for Selecting Simple Object Access Protocol (SOAP) vs. Representational State Transfer (REST) Based Architecture. *Journal of Advances in Computer Networks, 3*(1), 63-6. https://doi.org/10.23011/10.7763/jacn.2015.v3.143

Kim, H., Park, J., Choi, M. H., & Moon, I. (2018). REST API based Server Construction for Web Application Performance Analysis. *Journal of Advanced Navigation Technology, 22*(5), 456 - 461. https://doi.org/10.12673/JANT.2018.22.5.456

Lee, E. D., Lee, C. M., & Lee, W. S. (2009). Web Service based on SOAP and REST Integrated Platform Implementation using Ruby on Rails. Proceedings of Symposium of the Korean Institute of communications and Information Sciences, 2049-2050. http://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE02089164

Lee, S. W., Seok, W. J., Moon, J. H., Hong, W. T., Kim, K. H., Kim, C. Y., Goo, Y. H. (2021). Container Based Cloud Data Accessibility Improvement Method Using REST-API Based Third Party Transmission. *The Journal of Korean Institute of Communications and Information Sciences, 46*(10), 1709-1718. https://doi.org/10.7840/kics.2021.46.10.1709

Lim, S. H. (2011). *Light weight unified communications and collaboration service using RESTful web services* [Doctoral dissertation]. Chungnam University, Chungnam. http://www.riss.kr/search/detail/DetailView.do?p_mat_type=be54d9b8bc7cdb09&control_no=b70bb686214397b6ffe0bdc3ef48d419&outLink=K

Park, Y. M., Moon, A. K., Yoo, H. K., Jung, Y. C., & Kim, S. K. (2010). SOAP-based Web Services vs. RESTful Web Services. *Electronics and Telecommunications Trends, 25*(2), 112-120. https://doi.org/10.22648/ETRI.2010.J.250211 https://ettrends.etri.re.kr/ettrends/

122/0905001533/25-2_112_120.pdf

Yoo, J. C., Lee, W. W., & Shin, K. Y. (2012). A Study on the Implementation of NetOps Suited for Korea Military. *Journal of National Defense Security, 55*(2), 47-75. https://doi.org/10.23011/jnds.2012.55.2.003

원 고 접 수 일 2021년 07월 30일
원 고 수 정 일 2021년 12월 15일
게 재 확 정 일 2021년 12월 20일

IDAP 광운대학교 방위사업연구소 INSTITUTE OF DEFENSE ACQUISITION PROGRAM

JAMS JOURNAL OF ADVANCES IN MILITARY STUDIES

2021, Vol. 4, No. 3, 153-176.

https://doi.org/10.37944/jams.v4i3.116

국방정보체계 실시간 서비스 통합 방안 연구: 국방 REST API Server 참조모델 설계 분석

박용탁* · 김도영**

국문초록

본 연구는 representational state transfer(REST) Architecture Style을 기반으로 국방 REST API Server의 참조모델을 설계하여 국방정보체계의 실시간 서비스 통합을 위한 가장 효율적이고 안정적이며 지속 가능한 기술적 기준을 제시한다. 또한, 그 구성요소가 국방상호운용성관리지시, 연동지침서 등의 국방부 지시 및 규정의 한 부분으로 명문화될 수 있도록 근거를 제시하여 추후 국방 연동기술 및 상호운용성의 발전에 보탬이 되는데 그 목적을 두고 있다.

국방 정보체계의 육・해・공군별 그리고 업무기능별 세분된 발전은 물리적으로 떨어진 정보체계 간의 연동을 시스템의 완성도를 높이는 매우 중요한 요소가 되게 하였다. 하지만, 대한민국 국방의 연동체계는 이러한 환경적 요구에도 불구하고 enterprise application integration(EAI), EAI Hub & Spoke 등을 기반으로 한 연동 서비스 모듈이 망(Network, 전장망/자원망 등) 또는 정보체계별로 특별한 기준 없이 지엽적인 요구사항(단순 데이터 전송)을 충족시키는 수준에서 개발되었을 뿐 궁극적인 서비스 통합에는 이르지 못하였다. 그 결과, 현재 운용 중인 대부분의 연동 모듈은 시대 변화에 따라 높아지는 실시간 연동 및 서비스 통합에 대한 군의 요구사항을 만족시키지 못하는 등 기술적 스펙트럼의 부재를 겪고 있다. 따라서 본 연구는 국방 정보체계를 하나의 서비스로 통합하고 다양한 실시간 연동의 요구사항을 충족하기 위해 국방 REST API Server의 참조모델을 제시하고 기술적 근거를 분석하여 군의 현실에 맞는 모델을 통해 위의 문제점들을 해결하는 방향을 찾고자 한다.

주제어 : 국방정보시스템, 국방 레스트 애플리케이션 프로그래밍 인터페이스, 실시간 서비스 통합, 국방 상호운용성 관리지시

* (제1저자) 광운대학교 방위사업학과, 박사과정(한국국방연구원 선임연구원), ytpark85@gmail.com

** (교신저자) 광운대학교 방위사업학과/전자바이오물리학과, 교수, kdoyoungnid@gmail.com