JANT 한국항행학회논문지
Journal of Advanced Navigation Technology

응용 및 융합 기술

J. Adv. Navig. Technol. 19(4): 330-337, Aug. 2015

IoT 서비스 확장성을 고려한 컴포넌트 기반의 미들웨어에 관한 연구

Study on Component-Based Middleware for IoT Service Scalability

신 승 혁
구미대학교 사이버보안과

Seung-Hyeok Shin
Department of Cyber Security, Gumi University, Gyeongsangbuk-do 730-711, Korea

[요 약]

사물인터넷(IoT; internet of things) 환경에서의 서비스는 장비 및 센서들에 의하여 다양한 형태의 서비스로 형성된다. 자동화 설비는 센서에 의하여 동작하는 대표적인 IoT 환경의 서비스로서 센서의 종류에 의하여 다양한 통신 및 프로토콜로 구성이 가능하며, 이러한 다양한 형태의 서비스를 통일된 형태의 IoT 어플리케이션을 구성하기 위해서는 네트워크 및 서비스 구성 효율적인 미들웨어가 필요하다. 본 논문에서는 센서들과 연동하고 제어 및 감시를 목적으로 하는 애플리케이션 개발에 적합한 미들웨어를 설계하였다. 제안된 미들웨어는 소프트웨어 공학의 설계 디자인 패턴중 하나인 어댑터 패턴를 적용하였다. 적용된 디자인 패턴으로 각 센서별 통신 및 서비스 확장성을 확보하였다. 마지막으로 제안된 컴포넌트 미들웨어는 IoT 환경에서의 다양한 센서와 서비스를 쉽게 구성할 수 있음을 확인 할 수 있었다.

[Abstract]

A service in the environments of internet of things (IoT) exist various types with automation facilities and sensors. There can configure so many communication protocols to networking facilities and sensors. To provide efficient various kind of service, a middleware platform, is based on the internet protocol network, is needed a unified access with devices, controlling and monitoring huge kind of facilities and sensors, to provide a efficient IoT service and application configurations. In this paper, we propose a middleware that an application and service interact with automation facilities and monitor sensors. The proposed middleware is designed with adapter pattern that one of the software engineering design pattern. The adapter pattern is to ensure communication with each sensor and to make sure of service scalability. Finally, the proposed component middleware shows that variety sensors can be easily configure the service in the IoT environment.

Key words : Internet of things, Component, Middleware, Adapter design pattern, Service scalability.

http://dx.doi.org/10.12673/jant.2015.19.4.330

Received 24 June 2015; **Revised** 24 July 2015
Accepted (Publication) 24 August 2015 (30 August 2015)

***Corresponding Author; Seung-Hyeok Shin**

Tel: +82-54-440-1347
E-mail: shinbaad@gmail.com

Ⅰ. 서 론

사물인터넷 (IoT; internet of things)은 internet을 통신망으로 사용하는 다양한 장치 (device)들이 상호 간 정보를 교환 및 생성하여 유용한 정보 및 새로운 형태의 정보를 생성한다. 다양한 device에 의하여 생성된 정보는 새로운 정보 및 가치를 창출한다. 이러한 IoT의 급격한 발달은 다양한 산업 분야에서 활용된다[1]. 다양한 형태의 IoT 서비스는 deivce 영역, network 영역 그리고 application 영역의 세 가지 요소로 나눌 수 있다. device 영역은 사물과 주위 환경으로 부터 정보를 획득하기 위한 센싱 기술을 요구하며, network 영역은 사물이 인터넷에 연결되기 위한 유무선 통신 및 network 인프라 기술이 요구된다. 마지막으로 application 영역은 각종 서비스 분야의 정보를 가공, 처리 및 분석 기술이 요구된다. 다양한 산업 분야에 적용되는 IoT 기술을 효율적으로 적용하기 위해서는 device 영역의 여러 형태의 device와 연결이 용이한 통신용 middleware의 효율적인 구축이 요구된다. 이를 위하여 다양한 산업체 및 연구 기관에서 다양한 형태의 연구가 진행되고 있다 [2].

IoT의 접목 분야 중에서 자동화 시스템 산업의 형태는 그림 1에 도시되었다. 자동화 시스템 산업의 IoT 환경에 적합한 middleware는 여러 형태의 device와 연결이 용이한 component 형태의 middleware와 이를 기반으로 운용되는 자동화 소프트웨어(A-SW; automation software)가 요구된다. 또한 IoT 환경의 device들은 인터넷 환경에 분포되는 특성상 원격에서의 제어 (control) 와 감시(monitoring) 기능이 필수적이다. 자동화 시스템의 주요 핵심인 A-SW는 IoT 환경에서 지속적으로 생성되는 device에 대한 정보를 수용되어야 한다. 따라서 A-SW에 대한 지속적인 관리 필요성이 요구된다. 이러한 요구를 충족하기 위하여 A-SW의 기능성, 신뢰성, 사용성, 효율성, 유지보수성 그리고 이식성 등 국제 표준으로 인정된 ISO/IEC 9126의 소프트웨어 품질 특성[3]에 대한 요구사항(requirement)이 급격히 증가한다. A-SW에 대한 requirement의 증가는 시스템의 복잡도 (complexity)를 증가시키는 원인이 된다. A-SW의 complexity는 위에 기술한 requirement의 complexity와 연관된다[4].

자동화 시스템을 최적화하여 구현하기 위해서는 network 기반의 운용 효율적이고 이식성이 높은 통신용 middleware가 필요하다. 기존 산업에 적용되고 있는 주요 통신용 middleware로는 open source 기반의 ACE (adaptive communication environment) 라이브러리와 OMG(object management group)의 CORBA(common object request broker architecture)가 대표적이다. ACE 라이브러리는 운영체제가 제공하는 system call을 C언어로 wrapping 한 후, 상위 application과의 interface를 위하여 객체지향 프로그램 언어인 C++로 작성되어있다. CORBA는 분산 환경에서 RMI (remote method invocation) 서비스를 제공하는 분산 컴포넌트 기반의 middleware이다. 초기 CORBA의 경우 RMI 서비스 중심으로 구성되어 있으며, client/server 방식의 통신 및 thread 관리가 추가된 RT-CORBA (real time-CORBA)가 component middleware의 대표적인 연구 사례이다[5].

A-SW는 각 기능별 message 및 contents 등을 어떻게 교환 할 것인가에 대한 정의가 필요하다. 또한 유사한 application 기능들에 대하여 일관성 있는 API(application program interface)를 고려해야 한다. ACE와 RT-CORBA는 통신을 위한 시스템 레벨의 API 를 기반으로 작성되어 있다. 시스템 레벨의 API의 경우 application 개발자에게 연결 채널 관리에서부터 상호작용에 필요한 이슈를 고려하게 만드는 경향이 있다. 이러한 경향은 application 개발자에게 시스템 레벨의 API를 제공함으로서 A-SW의 전반적인 complexity가 증가 하게 되는 원인을 제공한다. 이러한 원인으로 A-SW를 구성하는 각 소프트웨어 모듈에 대한 재사용성이 감소하게 되며, A-SW의 효율적인 서비스 확장이 불가능하게 된다. 결과적으로 A-SW를 구성하는 통신용 middleware의 재사용성 감소로 인한 A-SW의 유지보수성이 감소하게 된다[6].

본 논문에서 제안하는 middleware는 system call을 기반으로 하는 통신 기능과 application 확장성을 보장하는 component middleware를 제안한다. 제안된 component middleware는 동일한 통신 서비스를 제공하는 기능위에 유사한 application 서비스를 제공하기 위한 기능이 추가된다. 이를 위하여 소프트웨어 공학의 디자인 패턴 중 adapter design pattern을 적용한다.

전체 논문의 구조는 다음과 같다. 2장에서는 기존 산업 및 연구에서 주로 사용되는 open source기반의 middleware를 연구한다. 3장에서는 제안하는 middleware의 requirement 정리 및 middleware 설계 방안 대하여 논의 후 구현을 진행한다. 4장에서는 제시된 방안에 대하여 평가 후, 5장에서는 component middleware의 연구 결과와 향후 연구계획에 대하여 논의 후 결론을 맺는다.

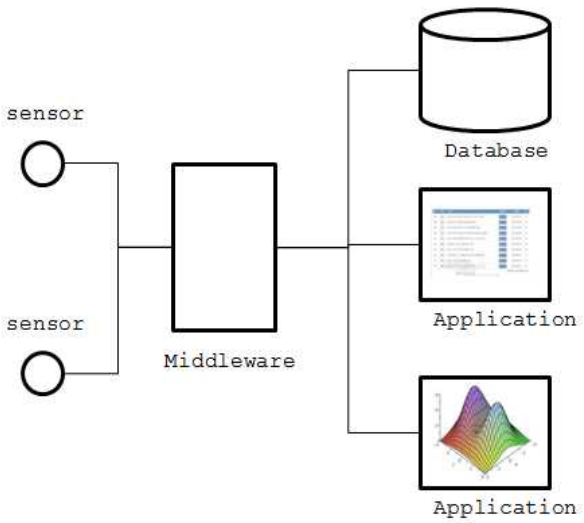

그림 1. IoT 자동화 시스템 개요도
Fig. 1. Overview of IoT automation system.

Ⅱ. 관련 연구

본 장에서는 기존 분산 시스템을 구현하기 위한 통신용 middleware와 adapter design pattern에 대하여 기술 한다.

2-1 ACE

ACE는 객체지향 언어로 구현된 open source 기반의 객체지향 프레임워크이며, network 통신을 위한 여러 종류의 코어 패턴들을 구현하고 있다. System call 레벨의 thread, process, signal, IPC(inter process communication) 그리고 socket의 효율적인 관리를 요구하는 기능단위의 software module로 구성되어 있다. 따라서 A-SW에서 요구되는 유사 기능의 공통 통신 모듈 등은 별도로 개발되어야 한다.

그림 2는 ACE의 계층 구조를 도시한다. C APIs 계층은 운영체제에서 제공하는 가장 일반적인 시스템 수준의 system call에 대한 wrapper 함수를 제공한다. 이는 ACE가 여러 운영체제 에 대해 시스템 함수에 대한 범용성을 제공함을 의미한다. C++ wrappers 계층은 함수와 데이터를 캡슐화 한 여러 class 들로 구성된다. 이러한 계층구조의 ACE는 운영체제에 독립적인 API를 제공 할 수 있도록 구성되어 있다. 결과적으로 system call을 사용하기 위하여 ACE에서 제공하는 C++구성된 API를 이용해야 한다[7].

Application 개발자 입장에서는 ACE에서 제공하는 추상화된 API를 이용하여 시스템에 독립적인 application을 개발 할 수 있다는 장점이 있다. 그러나 모든 운영체제 API를 wrapping 함으로써 발생하는 system call overhead는 전체 시스템에 대한 전반적인 기능 저하로 나타난다. 또한 필요 이상의 수많은 class를 제공함으로써 C#이나 Java와 같은 최상위 GUI (graphical user interface) application 과 연동하기 위하여 CLI (common language infrastructure), JNI (java native interface) 등 별도의 연동 부분을 개발하여야 하는 문제점이 존재하게 된다[8].

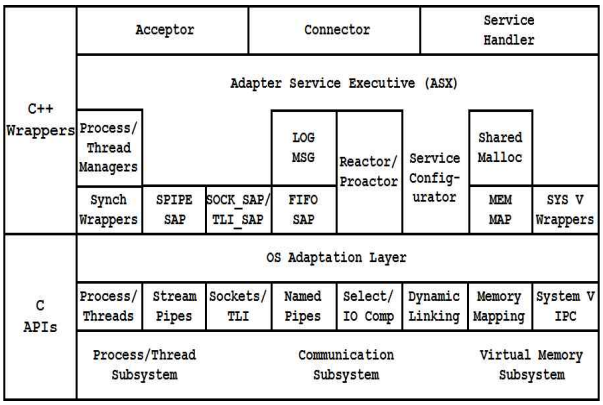

그림 2. ACE 계층 구조
Fig. 2. Layered architecture of ACE.

2-2 CORBA

그림 3은 CORBA의 구성을 도시한다. ORB는 각 object 간 client/server관계를 맺어 주는 middleware의 일종이다. Client stub은 해당 client에 대해 static interface를 제공한다. IDL (interface definition language) 컴파일러를 통해서 미리 생성되는 client stub은 특정 server의 object에게 서비스를 요청하는 기능을 담당한다. DII (dynamic invocation interface)는 compile 시 interface 정보가 알려지지 않은 object에 대한 request를 run-time 시 client 로부터 server로 전달시키기 위한 기능을 제공한다. interface repository는 모든 등록된 object에 대한 interface, method 및 필요한 parameter 들에 관한 정보들을 획득하여 저장하거나 수정하는 역할을 수행한다. Interface repository는 run-time database로서 IDL의하여 정의된 interface를 내부 형태로 변환하여 적재한다. ORB interface는 client와 server에게 ORB에 대한 직접적인 인터페이스를 제공하며, client와 server의 중간적 위치에서 object reference를 생성하고 관리하며, object reference에 대한 서비스를 client와 server 모두에게 제공한다. Object adapter는 server side의 중심 구성요소로서 다른 구성요소들과의 상호 작용을 통해서 client side의 요구에 대한 서비스를 제공한다. 또한 Persistent storage인 구현 저장소를 관리, server의 class들을 등록시키고 run-time에 새로운 object들을 활성화한다. Object reference를 생성, 관리하고 object reference와 상응하는 server side object를 연결하여 해당 object를 관리할 수 있도록 하며, 구현 저장소의 관리를 통해서 등록된 server와 object를 활성화 또는 비활성화 하고 client Side request의 권한을 확인하는 인증과정을 통해 유효한 request만 처리한다. Skeleton은 client side의 stub과 같은 역할을 하는 요소로써 IDL 컴파일러에 의해 생성되고 ORB와 server간의 매개 역할을 한다. Compile 시 결정되는 object에 대한 request만을 전달하고 처리 결과도 역순으로 client에게 전달한다[9].

CORBA는 OMG에서 표준화한 architecture이며, 동일한 application 혹은 network 상의 software component 들 간 상호작용이 가능하도록 한다. CORBA는 분산 object들이 통신 할 수 있는 쉽고 표준화된 수단을 제공한다. 그럼에도 불구하고 분산 application을 개발하려는 개발자는 내부 CORBA 구조, component 그리고 protocol 등 CORBA architecture에 대한 지식

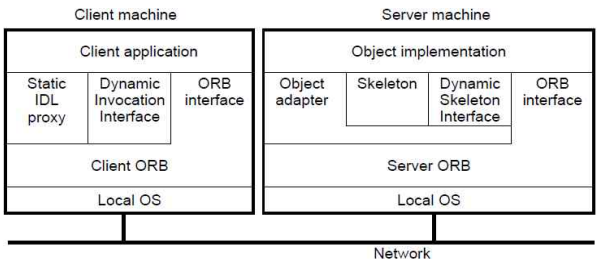

그림 3. CORBA 구조
Fig. 3. Architecture of CORBA.

이 필요하다. 방대하고 복잡한 CORBA의 스펙은 application 개발 시 개발자의 overhead로 작용된다. 이러한 CORBA의 low 레벨의 API 기능으로 A-SW를 구성 할 경우 시스템에 대한 complexity가 증가하게 되며, 결과적으로 component의 재사용성과 서비스에 대한 확장성이 감소하게 된다[10].

2-3 Adapter Design Pattern

소프트웨어 디자인 패턴은 1977년 Christopher Alexander가 정립하여 발표한 개념이다. 1987년 Ward Cunningham과 Kent Beck이 프로그래밍에 패턴을 적용하였고 그 결과는 OOPSLA (object oriented programming, system, languages & applications) 컨퍼런스에 발표하였다. 이후 1994년 Erich Gamma, Richard Helm, John Vlissides 그리고 Ralph Johnson 에 의하여 GoF (gang of four) 디자인 패턴이 체계적으로 발표 되었다. GoF 디자인 패턴은 생성 패턴 (creational patterns), 구조 패턴 (structural patterns) 그리고 행위 패턴(behavioral patterns)으로 구분된다. 생성패턴은 object의 생성과정을 정의하고, 구조 패턴은 class나 object의 합성을 정의한다. 마지막 행위 패턴은 class나 object들이 상호작용 하는 방법과 책임을 분산하는 방법을 정의한다.

본 연구에서는 구조 패턴 중 class의 interface에 대한 호환성이 지원되는 adapter design pattern을 사용한다. Adapter design pattern은 소프트웨어 구현 시 class의 interface를 client에서 사용하고자 하는 다른 interface로 변환하는 패턴이다. adapter design pattern을 이용하면 interface 호환성 문제로 같이 사용할 수 없는 class를 사용할 수 있다. 그림 4는 adappter design pattern을 도시한다.

Adapter design pattern의 target은 client가 사용할 도메인에 종속적인 interface를 정의한다. client는 target interface를 만족하는 object와 동작할 대상이며, adaptee는 interface 변환이 필요한 기존의 interface를 정의한다. 마지막으로 adapter는 target interface에 adaptee의 interface를 맞춰주는 class다 [11].

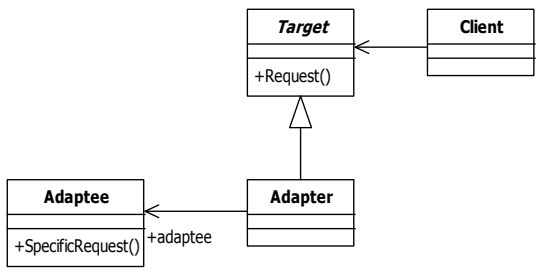

그림 4. 어댑터 디자인 패턴
Fig. 4. Adapter design pattern.

III. Component Middleware

본 장의 1절에서는 서비스 확장성을 고려한 component middleware의 기본적인 요구사항을 기술하고, 2절에서는 1절에서 제시된 요구사항을 기반으로 시스템을 설계한다. 마지막 3절에서는 설계된 시스템을 이용하여 component middleware를 구현한다.

3-1 시스템 요구사항

IoT 서비스 환경에서 network를 주요 통신 기반으로 하는 A-SW는 시스템을 제어하기 위한 control부와 실시간 감시를 위한 monitoring부로 구분된다. control부는 시스템의 network를 기반으로 하는 통신부와 관련이 있으며, monitoring부는 상위 application과 application을 도시하기 위한 GUI와 관련이 있다. Control과 monitoring을 위한 middleware는 그림 5와 같이 상위 application과 하위 통신부를 연결이 용이해야 한다 [12].

그림 6은 IoT 서비스 환경에서 A-SW로 구축하는 대표적인 자동화 시스템을 도시한다. 설비를 원격에서 제어하고 관리하기 위한 각 시스템은 다양한 형태의 제어 센서들로 구성 된다. 서로 다른 형태의 설비를 관리하기 위해서는 외부 장비와 연동하기 위한 연동 규격들이 필요하다. 또한 각 설비별로 구성되어야 할 application의 구성이 필요하다. 서로 다른 형태의 센서와 이를 연결하여 동작되어야 할 설비들을 하나의 A-SW로 구성하기 위해서는 공통적으로 사용되는 부분을 middleware의 중심 기능으로 작성하고, 다양한 연결 부분을 최소한의 변경으로 사용이 가능한 기능으로 작성한다 [13].

IoT 환경의 자동화 설비를 위한 A-SW는 network 운영체제에서 제공하는 기본적인 TCP/IP 통신 기능 외에 추가적으로 설비 제어를 위한 외부 장치와의 연결도 요구된다. 대표적인 외부 장치는 공장 자동화에 대표적으로 사용되는 PLC (programmable logic controller)가 있다. PLC는 mod-bus 표준 통신 프로토콜을 이용하여 통신이 가능하다. 그러나 설비의 운영 신뢰성을 위하여 PLC 공급 업체에서 제공하는 통신용 드라이

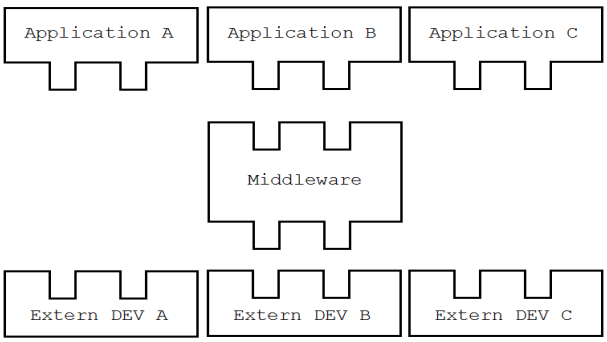

그림 5. 서비스 확장성을 위한 컴포넌트 미들웨어 설계
Fig. 5. Component middleware design for service scalability.

버를 사용하고 있다. 그 외 serial 통신으로 제어가 가능한 설비들도 PLC 설비와 함께 대표적으로 사용되는 통신 기능들이다 [14]. IoT 환경의 설비를 제어하는 A-SW는 설비들에 대한 실시간 monitoring용 GUI 기능도 제공하여야 한다. GUI에 도시되는 내용들은 A-SW에서 설비의 정보를 송/수신 하여 조합한 정보들로 구성되어야 한다.

3-2 시스템 설계

그림 7은 시스템 요구사항에서 도출된 requirements를 기반으로 구성한 component middleware의 기본 구조를 도시한다. component middleware는 상위 application을 위한 application adapter와 하부의 network 운영체제 interface를 위한 MTIF (middle-tier interface)로 구성한다. MTIF는 system call block 과 external adapter로 구성한다. MTIF의 system call block은 TCP/IP 통신을 위한 socket 및 thread, signal, RS232c serial 통신, IPC, XML parser 그리고 database connection interface 등을 위한 내부 API로 구성하고, external adpater는 외부 장치를 연결하기 위한 device API로 구성한다. System call block은 MTIF의 핵심 부분으로 network 운영체제를 interface 하기 위한 system call interface와 상위 application에서 재사용이 가능한 component interface 부분으로 구성한다. 상위 application을 연결하기 위한 application adapter와 external adapter는 서비스 확장성을 고려하여 소프트웨어 공학의 디자인 패턴인 adapter design pattern을 적용한다.

3-3 시스템 구현

그림 8은 component middleware의 주요 class diagram을 도시한다. 가장 최상위 object인 cObject는 운영체제에서 제공하는 file, thread, timer 그리고 signal등 시스템 전반에 걸쳐 공통적으로 사용될 기능들을 명시한다. TCP/IP 를 위한 cSocket

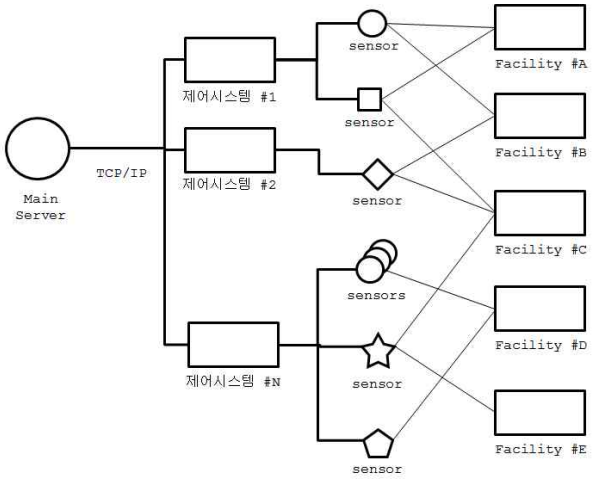

그림 6. IoT 설비 제어 시스템 구성도
Fig. 6. Configuration of IoT facilities controlling systems.

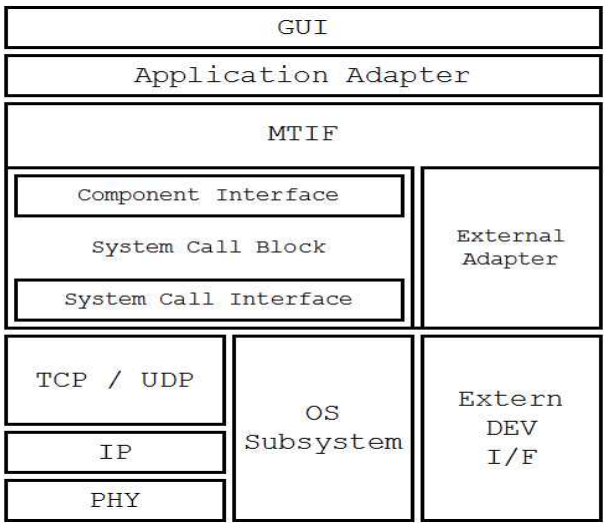

그림 7. 컴포넌트 미들웨어 구조도
Fig. 7. Architecture of component middleware.

class는 BSD socket을 이용하여 통신의 기본적인 기능들을 제공하고, cSocketPack class는 cSocket의 기능들을 이용하여 TCP, UDP, ICMP, unicast, broadcast 그리고 multicast를 결정하는 기능을 제공한다. cSocketManager는 ISocket을 interface로 하는 component로 구성하여 재사용이 가능하도록 한다. Message 기반 통신 middleware (MOM; message oriented middleware)의 특성상 주로 사용되는 database connection, RS232c serial 통신, XML parser 그리고 IPC의 한 종류인 shared memory를 위한 class를 cSocket class와 유사하게 설계 하였다.

그림 9는 external adapter와 application adapter에 적용된 adapter design pattern의 class diagram을 도시한다. 이들 adapter design pattern의 주요 설계 관점은 서비스 확장을 위한 유연한 연결성 제공에 있다. 유연한 연결은 시스템의 각 구성요소와 연

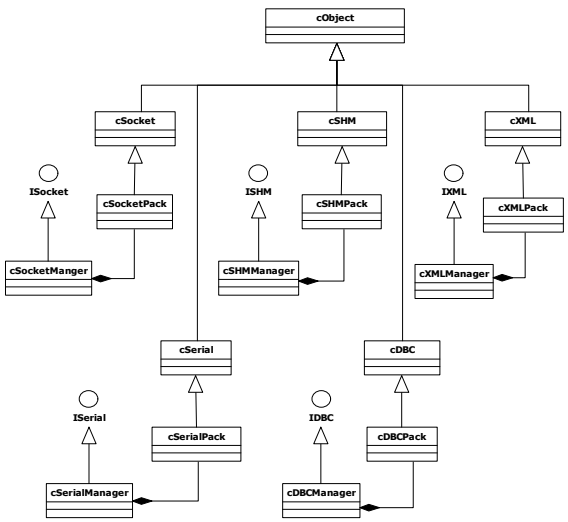

그림 8. 컴포넌트 미들웨어 주요 클래스 다이어그램
Fig. 8. Main class diagram of component middleware.

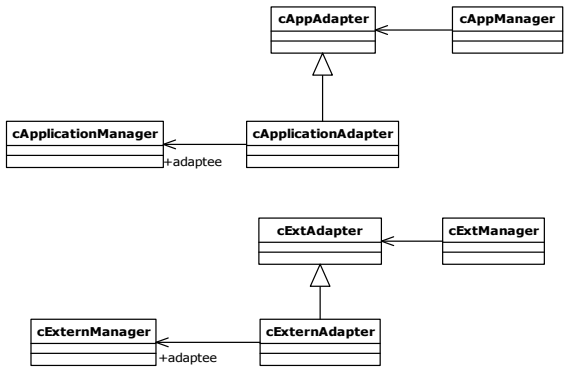

그림 9. 어댑터 디자인 패턴 적용 결과
Fig. 9. Applied result of adapter design pattern.

결할 때 별도의 부가적인 구성없이 연결 및 해제가 용이 함을 의미한다[15].

다음은 통신과 관련된 소켓 클래스 선언 코드의 일부이다.

```
class cSocketManager : public ISocket
{
  ....
};
class cSocketPack : public cSocket
{
 /// 컴포넌트 생성을 위한 인터페이스
 virtual HRESULT __stdcall
                    QueryInterface(const IID&, void**);
 virtual ULONG   __stdcall AddRef();
 virtual ULONG   __stdcall Release();
 ....
 cSocketManager* p_sm;
 ....
};
```

다음은 application 서비스 확장을 위하여 adapter design pattern을 적용한 코드의 일부이다.

```
class cAppAdapter
{
};
class cApplicationManager
{
};
class cApplicationAdapter : public cAppAdapter
{
 cApplicationManager* p_am;
};
```

Ⅳ. 성능 평가

디자인 패턴을 적용한 실시간 분산 시뮬레이션을 위한 데이터 전달처리 체계의 설계에 관한 기존 연구가 있었다[16]. 기존 연구에서는 통신을 위한 서로 다른 서비스 체계를 위하여 adapter, bridge 그리고 facade design patten을 이용하여 설계하였다. 기존 연구의 주된 관점은 network 기반의 통신이 주요 연구 대상이었으며, 이를 위하여 다양한 디자인 패턴을 복합적으로 적용하여 설계하였다. 기존 연구 성과로는 ISO/IEC 9126의 소프트웨어 품질 특성을 기준으로 설계의 우수성을 평가하였다.

본 연구에서는 기존 연구의 주된 연구 결과인 공통된 통신 기능을 component middleware의 통신 component로 구성하여 시스템의 기능 향상을 보였다. 또한 기존 연구에서 사용된 다양한 디자인 패턴의 복잡한 구성을 단순화하여 adapter design pattern만을 적용함으로써 공통적으로 사용되는 통신 기능 외에 추가적으로 A-SW에서 사용해야 할 외부 통신 기능과 GUI의 서비스 확장성에 대한 기능 개선을 보였다. 표 1은 component middleware를 구성하기 위한 시스템 구성을 나타낸다. 그림 10과 그림 11은 본 논문에서 제안하는 component middleware를 이용하여 같은 종류의 원재료에 의한 서로 다른 모델을 생산하는 차량용 2차 전지 셀 조립 자동화 시스템에 구축한 결과이다. 최종 제품을 생산하기 위한 서로 다른 형태의 생산 시스템에서 제품 조립 및 원재료 공급을 위한 유사한 기능의 설비들을 동일한 형태의 통신 기능을 이용하여 제어하며, 최

표 1. 컴포넌트 미들웨어 사양
Table 1. Specification of component middleware.

항 목	사 양
Processor	Intel Core i5 Quad
Main Memory	4GB
Network	100 Mbps
OS	Windows 7 pro
Language	C/C++

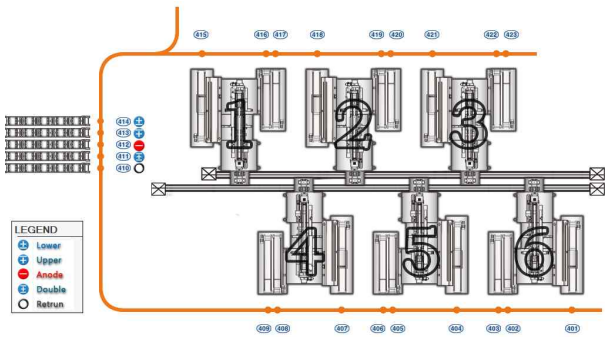

그림 10. 셀 조립 라인 1
Fig. 10. Cell assembly line 1.

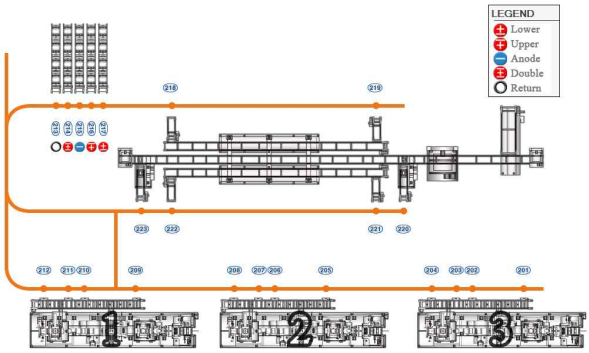

그림 11. 셀 조립 라인 2
Fig. 11. Cell assembly line 2.

종 application 에 대한 서비스를 다양한 형태로 구성할 수 있음을 확인 할 수 있었다. 다음은 GUI 서비스 확장을 위하여 구현된 코드의 일부이다. Line01과 Line02는 각각 그림 10과 그림 11을 위한 기본 클래스를 나타낸다. 셀 조립 라인을 운영하기 위한 각 클래스는 서비스 확장을 위하여 그림 9에 도시된 adapter design pattern을 상속 받아 구성하도록 하였다.

```
class Line01 : public cApplicationAdapter, public ILine
{
  Line01()
  {
    this->type = LINE_TYPE.LINE_01;
  };
  ....
  cSocketPack* p_sp;
};

class Line02 : public cApplicationAdapter, public ILine
{
  Line02()
  {
    this->type = LINE_TYPE.LINE_02;
  };
  ...
  cSocketPack* p_sp;
};
```

다음 표 2는 기존 연구와의 비교한 개선된 기능을 나타낸다.

Ⅴ. 결 론

본 논문에서는 component 기반의 통신 기능과 adapter design pattern을 적용한 시스템 구현으로 서비스 확장성을 고려한 component middleware를 제안하였다. 기존 연구에서 사용된 통

표 2. 시스템 비교 (ISO/IEC 9126)
Table 2. System comparison. (ISO/IEC 9126)

비교항목			기존 연구		제안 시스템	
품질 특성	부품질 특성	품질 속성	통신 확장	Application 서비스 확장	통신 확장	Application 서비스 확장
기능성	적절성	완전성	△	×	○	○
		일관성	△	×	○	○
	상호 운용성	자료 일반성	○	△	○	○
		통신 일반성	○	×	○	○
신뢰성	고장 허용성	일관성	○	△	○	○
		무결성	△	△	△	△
사용성	운용성	계층성	△	×	○	○
유지 보수성	변경성	일관성	○	△	○	○
		단순성	○	×	○	○
		간결성	○	×	○	○
		확장성	△	×	○	○
		독립성	△	×	○	○
이식성	설치성	자료 일반성	△	△	○	○
		통신 일반성	○	×	○	○
		모듈성	○	×	○	○
		간결성	△	×	○	○
	적합성	기계 독립성	△	△	○	○
		자료 독립성	△	△	○	○

신 기능 단위의 모듈화를 상위 application 서비스와 내부 통신 서비스로 확장하여 표 2에서처럼 개선된 기능을 제공함을 보였다. IoT 환경 중 자동화 시스템 분야에서는 공통된 통신 기능과 이를 유연하게 연결하여 다양한 application 서비스가 가능한 middleware 가 필요하다. 이러한 middleware는 개발 시간 단축 및 유지 보수 비용을 절감할 수 있는 필수 요소로서 제안하는 시스템을 이용하여 비용을 효과 적으로 줄일 수 있음을 확인 하였다.

제안하는 component middleware는 application 개발 시 자체 기능에 집중하고 하부 통신 기능 및 유사한 서비스 확장을 수월하도록 지원하는 것이 목적이다. 향후 통신 기능이 주요 핵심인 임베디드 시스템, 웹 소켓과 같은 최상위 통신용 application을 적용하는 방안을 고려하고, 최적의 코드로 고성능의 통신 기능을 구현한다면 IoT 서비스 환경의 자동화 시스템에 다양한 시너지 효과를 낼 수 있을 것으로 기대된다.

참고 문헌

[1] K. Ashton, “Internet of things,” *RFiD Journal*, Vol. 22, No. 7, pp.97-114, Jun, 2009.

[2] H. J. La and S. D. Kim, “Unconventional issues and solutions in developing IoT applications,” *Journal of the Korea Information Processing Society Transactions on Computer and Communication Systems*, Vol. 3, No. 10, pp.337-350, Mar. 2014.

[3] ISO, ISO/IEC 9216, Information Technology − Software Quality Characteristics and Metrics, 1998.

[4] J. K. Hong, “Framework of security development method based on component,” *Journal of the Korea Academia-Industrial Cooperation Society*, Vol. 11, No. 3, pp.926-930, 2010.

[5] H. J. Kim, Y. G. Seo, S. B. Kim, K. J. Kang and B. K. Lee, “Design and implementation of component-based configuration and data management system for weapon system R&D processes,” *Journal of the Korea Society of Computer and Information*, Vol. 13, No. 7, pp.127-138, Dec, 2008.

[6] J. W. Jeong and M. J. Lee, “An energy control model of smart video devices for the internet of things,” *Journal of Advanced Navigation Technology*, Vol. 19, No. 1, pp.66-73, Feb, 2015.

[7] S. J. Park, K. S. Park and C. H. Park, “High efficient game server using ACE network framework,” *Journal of Korea Game Society*, Vol. 9, No. 1, pp.75-84, 2009.

[8] H. Seong, J. Y. Kwak and J. S. Kim, “Design and implementation of communication framework for multi-agents,” in *Proceedings of the 28th Korea Information Science Society Fall Conference*, Seoul: Korea, Vol.28, No.2, pp.568-570, Oct. 2001.

[9] G. H. Lee, J. S. Heo and M. K. Kim, “Design and implementation of a CORBA/SNMP gateway,” *Journal of the Korea Information Processing Society*, Vol. 7, No. 8, pp.2505-2513, 2000.

[10] K. H. An, D. S. Cho and B. H. Hong, “Implementaion of OpenGIS-based middleware using CORBA,” *Journal of the Korea Open Geographic Information Systems Research Society*, Vol. 1, No. 1, pp.19-28, 1999.

[11] Y. J. Seo and Y. J. Song, “A study on component modeling tool based on design pattern,” in *Proceedings of the Korea Information Processing Society Conference*, Gwangju: Korea, pp.437-440, Oct, 2001.

[12] W. G. Jun and M. H. Lee, “Implementation of e-Meters system middleware base on RFID/USN,” *Journal of Advanced Navigation Technology*, Vol. 15, No. 5, pp.729-734, Oct, 2011.

[13] H. J. Kang, “Study on the next disaster safety communication network in M2M communication,” *Journal of Advanced Navigation Technology*, Vol. 15, No. 4, pp.585-590, Aug, 2011.

[14] M. H. Moon and K. W. Koo, “Development of real time distributed object remote monitoring system,” *Journal of Advanced Navigation Technology*, Vol. 13, No. 1, pp.79-86, Feb, 2009.

[15] K. B. Heo Y. G. Kim, and D. I. Yang, “The study of framework model for software productivity enhancement in object-oriented environment,” *Journal of Advanced Navigation Technology*, Vol. 14, No. 6, pp.900-908, Dec, 2012.

[16] J. W. Suk and I. T. Ryoo, “Data transmission processing system design for real-time distributed simulation by using software design patterns,” *Journal of Digital Contents Society*, Vol. 10, No. 4, pp.649-657, Dec, 2009.

신 승 혁 (Seung-Hyeok Shin)

2011년 3월 ~ 현재 : 금오공과대학교 대학원 컴퓨터공학과 박사과정

2014년 4월 ~ 현재 : 구미대학교 사이버보안과 교수

※ 관심분야 : 네트워크 프로토콜, 임베디드 시스템, 빅 데이터