碩士學位論文

맞춤형 임베디드 미들웨어 생성 시스템 설계 및 구현

濟州大學校 産業大學院

電子電氣工學科

孫 熙 哲

2010年 8月

맞춤형 임베디드 미들웨어 생성 시스템 설계 및 구현

指導敎授 金 度 縣

이 論文을 工學 碩士學位 論文으로 提出함

2010年 8月

濟州大學校 産業大學院

電子電氣工學科 컴퓨터工學 專攻

孫 熙 哲

孫熙哲의 工學碩士學位 論文을 認准함

2010年 8月

審査委員長 _____ 印

委 員 _____ 印

委 員 _____ 印

감사의 글

2008년 대학원 생활을 시작하여 어느덧 석사 학위과정을 마치고 학위를 받게 되었습니다. 2년 동안 열심히 했던 대학원 생활을 뒤 돌아보니 저에게 도움을 주셨던 모든 분들이 머릿속을 스쳐 갑니다.

먼저 제주한라대학 시절부터 제 옆에 언제나 있으면서 고민하고 있던 저에게 조언과 충고를 해주시던 선배이신 민주, 서주 아빠인 이철식 선배님과 미영 누나에게 너무 고맙다는 말을 하고 싶습니다.

무엇보다 제가 2년간 대학원 생활을 함에 있어 아낌없이 가르침과 지도를 해주신 지도교수님이신 김도현 교수님께 감사합니다. 또한 논문 심사에 애써주신 곽호영 교수님과 이상준 교수님께도 감사합니다.

항상 대학원 생활에 세심한 지도와 가르침을 아끼지 않아주셨던 안기중 교수님, 변영철 교수님, 송왕철 교수님, 김장형 교수님께 감사합니다. 또한 회사일 하느라 살피지 못한 여러 가지 학교 일들을 꼼꼼히 챙겨 주시며 알려주셨던 정은경 선생님과 연구실 생활에 도움을 주웠던 연준이와 연구실 동생들에게 고맙다는 말을 전합니다.

그리고 회사일 하면서 대학원 생활을 문제없이 마무리 할 수 있도록 도움을 주었던 사무실 식구들인 이종원 사장님과 김혁, 전영돈, 석훈, 경석 그리고 이현석 팀장, 하치일 팀장에게 감사하고 논문이 완성되기 까지 많은 도움을 주웠던 민성에게 감사합니다.

끝으로 항상 제 곁에 있으면서 응원 해 주시고 격려해 주시던 부모님과 부족한 저에게 언제나 힘이 되어주며 사랑하는 가족 영인이와 민석이 그리고 아내 정선에게 고마운 마음을 전하고 언제나 나를 이끌어 주시면서 도와주시던 장모님 그리고 형과 형수님께 감사함을 전합니다. 마지막으로 하늘에 계시면서 저희 가족을 지켜봐 주시는 장인어른께 감사합니다.

목 차

그림 목차

표 목차

국문초록

맞춤형 임베디드 미들웨어 생성 방법 설계 및 구현

전자전기공학과 컴퓨터공학전공 손희철
지 도 교 수 김도현

임베디드 시스템(Embedded System)은 일반적으로 미리 정해진 특정 기능을 실행하기 위해 시스템을 동작시키는 소프트웨어를 하드웨어에 내장한 컴퓨팅 장치를 의미한다. 이러한 임베디드 시스템에 내장된 소프트웨어를 임베디드 소프트웨어라 하며 운영체제, 미들웨어 및 응용 프로그램으로 나누어 질 수 있다.

현재까지 일반적인 임베디드 시스템은 특정 분야에 맞도록 소프트웨어가 탑재되어 있어 다른 분야에 적합하도록 개선하기 위해서는 그에 맞는 소프트웨어를 새로 탑재하거나 시스템을 재구축해야 한다. 또한 디바이스의 추가나 제거 또는 구동기 변동 등 다양한 환경 변화에 적응하기 위한 관리 프로그램이 매우 복잡하여 효율적인 자원 운용 및 유지보수를 위한 전담요원 배치 등에 있어 많은 비용과 시간 낭비를 초래하고 있다.

본 논문은 임베디드 시스템의 재사용성을 개선하기 위한 방안으로 디바이스 종류 및 디바이스 처리 로직이 변경 되더라도 쉽게 적응할 수 있는 미들웨어를 연구하였다. 이를 위해 기존의 임베디드 소프트웨어 기술과 임베디드 미들웨어 기술 및 미들웨어의 프레임워크를 고찰하였으며, 이러한 과정에서 가변적 측면에 대한 고려를 위한 컴포넌트의 공유의 필요함을 알 수 있었고, 기존에 제시된 임베디드 미들웨어의 자원 재사용을 위한 방법들이 개념적인 방법만이 제시되어 구체적인 구현방안 및 구현 모델이 미흡한 상태임을 알 수 있었다.

따라서, 이러한 문제를 해결하고자 본 논문에서는 임베디드 시스템의 가변성 및 소프트웨어의 재사용을 위한 컴포넌트 기반의 임베디드 미들웨어의 프레임워크를 설계하고 센서, 영상 등 다양한 디바이스의 변동에 대한 동적 처리를 위한 컴포넌트 처리 알고리즘과 절차를 제시하였다. 또한 가변성 지원을 위한 임베디드

미들웨어를 실제 구현하여 시스템 확장 및 유지보수에 대한 사용자 인터페이스를 개선하였고, 가동 환경 변화에 따른 미들웨어 처리로직의 변경 및 자원 재구성을 통한 미들웨어 생성과 센서, 영상 등 디바이스에 대한 동적 컴포넌트의 동작 상태를 확인할 수 있었다.

이러한 결과로 볼 때 본 논문에서 제시한 임베디드 시스템의 자원 재사용성 개선과 관리를 위한 인터페이스 구축을 통하여 시스템 구축 또는 유지관리를 위한 비용 및 시간을 줄일 수 있을 것으로 기대한다.

ABSTRACT

Design and implementation of the method for generation of custom-made embedded middleware

Hee-Cheol Son

Department of Electrical and Electronic Engineering
Graduate School of Industry
Cheju National University

Supervised by professor Do-Hyeon Kim

Summary

The Embedded system usually, the computing device that has been hardware equipped with software to operate the system in order to perform specific functions that were already used. The software set in such embedded system is called embedded software; and classified into the operation system, middleware, and application programs.

It has been the recent trend embedded system was equipped with the software to meet specific area, for improvement to meet another area and new software was to be built in or the system to be reconstructed such an area. Also, the operation program to meet various environmental changes such as addition or removal of devices and change of the driver was very complicated; lots of expenses and time were required in arrangement of the exclusive manpower for efficient operation and maintenance of resources.

In this thesis, the middleware was studied as the measure to improve the reusability of the embedded system for easy application in spite of change in the kind of devices or the processing logic of devices. For this, existing embedded software and middleware technology, middleware framework were reviewed in such process, it was found that the reuse of components is required to consider the aspect of variables and that previously suggested methods and the embedded middleware resources consisted of conceptual methods only were insufficient in terms of the concrete method for implementation models.

Therefore, the design of the framework of embedded middleware was suggested based on the component variables of embedded systems and the reuse of software component-processing algorithm. This process is suggested for dynamic processing of the change in the various sensor devices and videos, etc; in order to solve these problems. The user interfaces were improved for system extension and maintenance with the implementation of the embedded middleware to support the variables, it was possible to check the working state of dynamic components device's for sensors and videos through the change of middleware processing logic and reorganization of resources to meet the change in the driving environment.

As a result, it is expected that the expenses and time for system construction and maintenance will be reduced through construction of the interfaces to improve and manage the reusability of the embedded system resources as suggested in this thesis.

약어표

RTOS	Real Time Operating System
ROM	Read Only Memory
GUI	Graphic User Interface
VM	Virtual Machine
API	Application Programming Interface
CBD	Component Based Development
CD	Component Development
CBSD	Component Based Software Development
RUP	Rational Unified Process
EJB	Enterprise JavaBeans
CCM	CORBA Component Model
CF	Compact Flash
USB	Universal Serial Bus
SD	Secure Digital
UMPC	Ultra Mobile PC
WiFi	Wireless Fidelity
PIR	Passive InfraRed
Voc	Volatile Organic Compound
UI	User Interface
AP	Access Point

I. 서 론

1. 연구 배경

최근 유비쿼터스 기술의 확산과 다양한 분야에 최신 IT기술이 융합 되면서 디지털화가 가속되면서 특정 목적에 사용되는 컴퓨터 시장이 커지고 있다. 불과 20년 전 마크와이저(Mark Weiser) 가 “복잡한 컴퓨터가 미래에는 소형화 되면서 모든 제품(사물) 속으로 들어가 사람들이 컴퓨터 존재를 전혀 의식하지 못할 것”이라고 예언했 던 내용이 현실화 되고 있는 것이다. 전기모터가 소형화 되면서 자동차와 카세트 플레이어 속에 들어가 사람들이 모터의 존재를 느끼지 않고도 모터의 혜택을 입는 것과 같은 이치다. 임베디드 시스템의 하드웨어 및 소프트웨어는 이러한 Post-PC 시대에 성장 가능성이 매우 높은 핵심 분야이다.[10]

임베디드 시스템 장치는 가정, 자동차, 사무실, 관리 시설물 등 거의 모든 분야에 적용되고 있다. 이렇듯 많은 분야에서의 임베디드 장치가 사용됨에 따라 임베디드 시스템 개발에 있어 빠른 시간 내에 각각의 분야에 맞는 시스템을 개발해야 하는 어려움이 발생하게 된다. 임베디드 시스템은 크게 하드웨어와 소프트웨어 부분으로 나눌 수 있는데, 최근 들어 임베디드 시스템 하드웨어의 발전이 빠른 속도로 진행되고 있고 작아지고 성능은 점점 좋아지고 있다. 그러나 임베디드 시스템의 특성상 소프트웨어는 새로운 디바이스를 지원할 수 있도록 설계되어야 하는데 확장성이 고려되지 않은 임베디드 시스템의 경우 재구축해야 하는 어려움이 있다. 또한 임베디드 시스템은 개발 생산성 및 재사용성이 매우 낮으며, 이런 흐름은 임베디드 시스템의 유지보수 비용 증가, 개발비 증가, 시스템의 신뢰성저하 문제에 직면 하게 될 것이다.

임베디드 시스템의 하드웨어 측면에서는 하나의 시스템을 다양한 분양에 적용할 수 있도록 많은 연구와 개발이 진행되고 있으나 소프트웨어는 기존 임베디드 시스템의 성능문제 등으로 인해 현행 장비에 맞는 소프트웨어만 개발 및 탑재되고 있는 실정이며, 하드웨어의 많은 성능 개선에도 불구하고 자원 재사용 측면에서의 연구 진행이 미흡한 실정이다. 그러나 임베디드 시스템에서 소프트웨어의 가중치는 점점 커지고 있으며 사용 중인 임베디드 시스템에서의 오류의 발생 가능성 증가와 함께 지속적인 기능 및 성능 개선에 대한 요구가 늘어나고 있다.

본 논문은 이러한 임베디드 시스템의 소프트웨어적인 문제점을 극복하기 위한 기반을 제공하고자 임베디드 시스템의 재사용성을 위한 미들웨어를 연구하였다. 이를 위해 기존의 임베디드 소프트웨어 기술과 임베디드 미들웨어 기술 및 미들웨어의 프레임워크를 고찰하였으며, 기존에 제시된 임베디드 미들웨어의 자원 재사용을 위한 방법론 등 기존 문헌을 분석하여 자원 재사용을 위한 방안을 제시하고 임베디드 미들웨어 프레임워크를 제시 및 구현하여 실제로 자원 재사용을 통해 다양한 분야에 쉽게 확장할 수 있는 임베디드 미들웨어에 대한 연구를 수행하였다.

2. 연구 목적 및 방법

임베디드 SW 기술은 전통적인 RTOS(Real Time Operationg System)기반으로 스마트폰, 모바일TV 및 센서 네트워크 및 다양한 형태의 융합 SW 분야에 사용되는 SW기술로 발전하고 있다.

임베디드 운영체제 기술은 다양한 기기별 적용을 위해 경량화 기술, 저 전력화 기술, 빠른 부팅 기술 및 고 신뢰도 지원 기술 제공에 중점을 두고 있으며, 최근에는 센서 네트워크용 초소형 운영체제 기술, 플래시 메모리 지원 소프트웨어 기술, 보안 커널에 대한 연구가 발전해 가고 있다. 임베디드 미들웨어의 경우는 기기 및 서비스의 발전과 사용자 요구사항이 다양화를 충족시키기 위한 임베디드 SW의 기능이 복잡, 다양해 가고 있으며, 이에 따라 유연하고 확장성 있는 구조의 임베디드 시스템 미들웨어 플랫폼이 빠른 응용서비스 개발을 위해 점점 중요한 위치를 차지해 가고 있다. 즉, 세분화 된 공통 기능의 효율적인 공유, 동적 설치, 재구성 및 업그레이드 지원 등이 기능이 요구되고 있는 것이다. 따라서 응용서비스 지원 미들웨어 기술로서 독립된 응용을 지원하는 것이 아닌 응용들이 공통으로 필요한 기능을 제공해 응용 개발을 쉽게 할 수 있게 도와주는 라이브러리 역할을 하는 소프트웨어가 필요하다.

본 논문에서는 라이브러리 역할을 할 컴포넌트를 설계하여 입력 컴포넌트, 처리 컴포넌트, 출력 컴포넌트를 설계하고 각각의 기능의 컴포넌트들이 서로 연동될 수 있도록 일반화 작업을 통한 인터페이스를 정의하고 연동 인터페이스를 사용하여 플러그인 기법을 적용할 수 있도록 알고리즘을 설계하여 컴포넌트는 공유하여 사용 할 수 있도록 한다. 또한 플러그인 기법을 적용하기 위해 정의한 인터페이스 정보를 프로파일로 변환하고, 유지관리를 위한 관리프로그램과 전송 수신하는 모델을 제시하고 구현한다. 그리고 임베디드 시스템 재사용성을 높이기 위해 구성한 컴포넌트를 쉽게 설정하고 재배치 할 수 있도록 관리 프로그램을 설계하고 임베디드 시스템 상황 모니터링 및 저장 되어진 로그를 분석할 수 있도록 로그 View를 설계하고 구현하여 자원 재사용을 위한 컴포넌트의 동적 로드를 통한 임베디드 시스템의 확장에 관한 결과를 도출한다.

3. 논문 구성

본 논문의 구성은 1장 서론에 이어 2장 관련 연구에서는 본 연구와 관련된 임베디드 소프트웨어의 특성 및 응용분야 와 임베디드 미들웨어에 대한 개념과 구조 등을 조사하고, 자원 재사용을 위한 CBD(Component Based Development) 방법론과 이를 이용해 동적으로 커스터마이제이션을 하기 위한 방법에 관한 기술을 분석한다. 3장에서는 본 논문에서 제시하는 맞춤형 임베디드 미들웨어 설계와 컴포넌트 동적 로드를 위한 인터페이스 정의, 동적로드 알고리즘을 제안한다. 4장 에서는 3장에서 기술한 설계 및 알고리즘을 토대로 시스템을 실제 구현하고 고찰하였다. 마지막으로 5장에서는 결론을 맺는다.

II. 관련 연구

1. 임베디드 소프트웨어

임베디드 소프트웨어는 광범위한 임베디드 시스템에 탑재되는 소프트웨어를 통칭한다. 범용 소프트웨어와는 달리 보통 임베디드 소프트웨어는 임베디드 시스템 내의 마이크로프로세스 및 비휘발성 메모리(ROM,플래시 메모리 등)에 내장되어 동작한다.

구분	패키지 소프트웨어	임베디드 소프트웨어
특징	◆ 사용자의 요구사항 및 정보처리를 주목적으로 함 ◆ 개인 및 기업용 범용 SW 포괄 ◆ 미국의 특정 기업이 주로 독점 ◆ 실시간성, 자원 제한성,고신뢰성 등이 중대(critical)하게 요구되지 않음	◆ 종전의 HW제어에서 부가 기능 제공으로 역할 확대 ◆ 특정 제품에서만 동작하는 SW ◆ 제1 강자가 존재하지 않음 ◆ 실시간성, 자원 제한성, 고 신뢰성등을 요구
개발자 측면	◆ SW마을 개발 ◆ 프로그램밍 기술 및 로직만 필요 ◆ 운영되는HW, OS와의 거의 동일 ◆ PC와 같은 네이티브 개발 환경	◆ HW와 함께 개발(HW에 대한 지식 및 경험 필요) ◆ 시스템 소프트웨어 기술 필요 ◆ 같은 기능이라도 다양한 HW에 이식해야 함 ◆ 호스트와 타킷으로 구성된 교차 개발 환경
최종 사용자 측면	◆ 데스크톱PC에서 선택적 운용 ◆ HDD에 저장 ◆ CD 및 플로피 디스켓으로 배포 ◆ 사용자 인터렉션은 GUI 활용 ◆ 고장 발생시 쉽게 유지 보수	◆ 임베디드 시스템 HW상에서 자동으로 운영됨 ◆ ROM에 내장 ◆ HW와 함께 배포 ◆ 사용자 인터렉션이 최종 제품을 통해 발생 ◆ 고장 발생시 제품 사용이 불가

표 1. 패키지 소프트웨어와 임베디드 소프트웨어의 비교

일반 범용 시스템에서와 마찬가지로 임베디드 소프트웨어도 운영체제, 미들웨어 및 응용프로그램으로 나누어 질수 있으며, 또한 임베디드 소프트웨어의 개발을 지원하는 개발 도구 및 개발 방법론도 넓은 의미의 임베디드 소프트웨어 범주로 볼 수 있다. 무인 항공기용 비행제어시스템이나 항법시스템에 내장되는 임베디드 소프트웨어는 다양한 센서로부터 받은 값에 따라 항공기의 제어작업을 항상 주어진 마감시간(Dead Line)내에 처리해야 하는데, 이를 경성 실시간성 이라고 한다. 소프트웨어의 오동작 및 작동 중지가 허용되지 않는 임베디드 시스템에서는 고도의 신뢰성이 요구 된다.[13]

임베디드 소프트웨어는 범용 데스크톱 또는 서버에서 실행되는 패키지 소프트웨어와는 달리 특정 임베디드 시스템에서 실행하는 것을 목적으로 하므로, 임베디드 소프트웨어의 기능은 탑재될 임베디드 시스템의 기능에 따라 결정되며, 임베디드 소프트웨어의 개발에는 풍부한 하드웨어 지식과 시스템 소프트웨어의 개발 경험이 요구 된다.

임베디드 소프트웨어의 주요 기술분야는 기본 응용 분야 미들웨어분야 임베디드 시스템 S/W분야 임베디드 S/W 개발도구 분야 임베디드 S/W 플랫폼 분야로 볼 수 있다. 그림 1은 임베디드 소프트웨어의 분류와 주요 응용 기술 분야을 나타내고 있다.

그림 1. 임베디드 S/W 주요 기술 분야

2. 임베디드 시스템 미들웨어

임베디드 미들웨어는 특정 목적의 하드웨어에 내장되어 요구되는 기능을 만족시키는 하드웨어형 미들웨어를 말하며, 일반 소프트웨어와는 달리 하나의 다른

디바이스 개념이라 볼 수 있다. 임베디드 미들웨어는 유비쿼터스 컴퓨팅 미들웨어, 응용서비스 지원 미들웨어, 가상머신(Virtual Machine), 보안 미들웨어 등으로 분류할 수 있다.

유비쿼터스 컴퓨팅 미들웨어 기술에는 Jini, Upnp, HAVi 등이 있다. Jini는 1998년 Sun 마이크로 시스템즈에서 발표한 분산 환경의 홈 네트워 자원 공유 플랫폼이며, PC와 프린터 등의 사무기기 뿐만 아니라 오디오 TV등의 가전 장비를 인터넷으로 연결, 제어 가능한 기술을 말한다. [13]

응용서비스 지원 미들웨어 기술은 자체로서 독립된 응용을 지원하는 것이 아닌 응용들이 공통으로 필요한 기능을 제공해 응용 개발을 쉽게 할 수 있게 도와주는 라이브러리 역할을 하는 소프트웨어 기술이다.

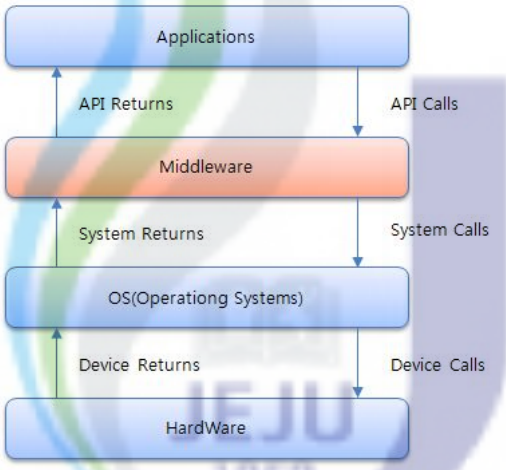

그림 2. 임베디드 미들웨어 구성도

임베디드 미들웨어는 시스템 소프트웨어와 응용 소프트웨어의 중간에서 특정 응용분야에 최적화된 공통 프레임워크이다. 일반적으로 미들웨어는 그림 2에서와 같이 응용프로그램과 운영체제 중간에 위치하고 있다. 응용 프로그램, 미들웨어, 운영체제 및 하드웨어 계층은 각각 인터페이스 및 API 함수를 통해 상호 호출하여 동작하는 소프트웨어다.

임베디드용 미들웨어의 경우 설계자는 여러가지 운영체제에서도 동작할 수 있도록 개발하여야 하므로 다양한 운영체제가 제공하는 시스템 콜 및 리턴 등의 상호 호출에 대해서도 상호 운용이 가능한지를 고려해야 한다. 이러한 설계단계의 복잡성으로 인해, 동일 미들웨어를 여러 임베디드 시스템에 설치 할 경우 일

부 임베디드 시스템은 정상적으로 동작하지 않는 경우가 발생하게 된다. 물론 이러한 문제를 해결하기 위해 하드웨어 종류에 구애받지 않도록 하기위해 임베디드 미들웨어를 자바로 개발할 수도 있으나, 임베디드 시스템의 하드웨어 자원의 한계 등으로 인해 현실적으로는 어려운 실정이다.

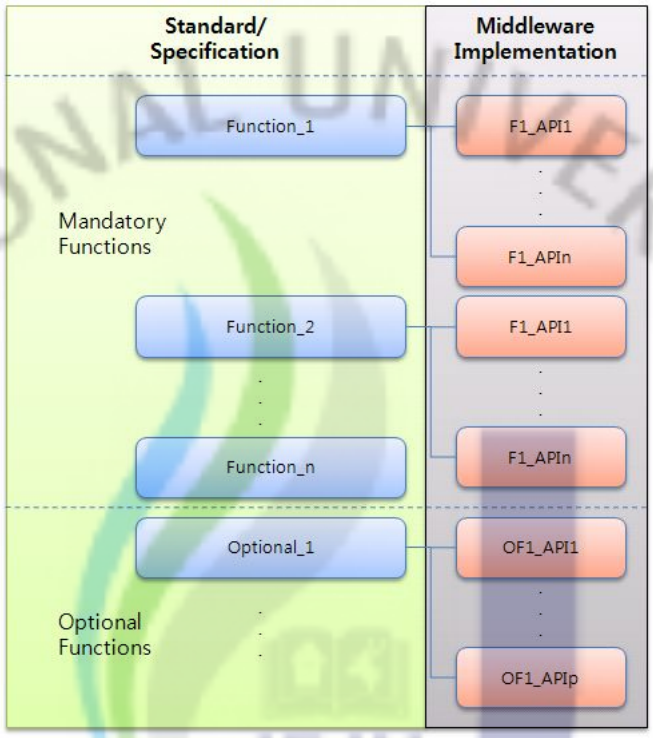

그림 3. 임베디드 미들웨어 API 구성도

일반적으로 미들웨어는 미들웨어 표준 및 기술문서를 기반으로 개발된다. 그림 3에서와 같이 미들웨어 표준은 필수적인 기능과 부가적인 기능으로 구성되어 있으며 개발자는 여러 개의 API 함수로 해당 기능을 구현하게 된다. 즉, 표준에서 요구하는 기능은 여러 개의 API가 조합으로 구현될 수 있기 때문에 미들웨어 표준 적합성은 단일 API의 정상 동작을 확인하는 유닛과 API의 유기적 조합으로 구현된 기능의 정상 동작여부 및 표준 적합성 여부를 살피는 기능이 있어야 한다. 단일 API 함수의 정상동작 여부 확인만으로는 미들웨어 표준이 제안하는 기능의 정상동작 여부는 알 수 없으므로 기능은 표준 준수여부 확인을 위해 필수적으로 있어야 한다.[14]

3. CBD(Component Based Development) 방법론

80년대에 떠오른 소프트웨어 위기(Software Crisis)와 그 해결책으로 생각되었던 객체지향 개념은 소프트웨어 모듈의 독립성은 보장하지만, 재사용성에 대한 소프트웨어 모듈로서 수정이나 재 컴파일 과정 없이 클라이언트 모듈 호출만으로 사용할 수 있도록 하지는 못했다. 이와 같이 본질적인 한계성이 나타남에 따라 CBD(Component Based Development)는 컴포넌트라는 소프트웨어 모듈의 재사용성과 독립성을 보장하게 하여 소프트웨어의 복잡성과 생산성 문제를 해결하고자 하는 일종의 새로운 개발 패러다임을 제시한 것이다. [2]

"CBD란 일반적으로 재사용 가능한 소프트웨어 모듈을 의미하며, 기계 부품과 같이 소프트웨어도 하나의 부품으로 제작한 다음 이를 조립하여, 보다 복잡한 소프트웨어를 만드는 방식이다." 라고 미국 조지메이슨 대학의 브래드 콕수는 정의하고 있다.[5]

일반적인 컴포넌트 기반 개발은 대체로 요구분석 및 컴포넌트 획득단계, 컴포넌트 기반 설계 단계, 컴포넌트 조립 단계, 시험 단계, 유지보수 단계를 거친다. 컴포넌트 기반 개발 방법은 크게 두 가지 형태로 나눌 수 있는데 하나는 컴포넌트 획득 방법 중의 한 방법인 컴포넌트 자체를 개발하는 컴포넌트 개발(CD)이고 다른 하나는 이미 구축되어 있는 컴포넌트를 사용하여 소프트웨어를 개발하는 컴포넌트 기반 소프트웨어 개발(CBSD)이다[4].

CBD 방법론으로는 RUP, Catalysis, CBD96, UNIFACE, Fusion 등과 같은 CBD 방법론들이 소개되고 있을 뿐만 아니라 여러 개발도구인 .Net, EJB, CCM 등과 같은 컴포넌트 기술을 사용하고 있는 기술들이 제시되고 있다. 방법론 중 Catalysis 방법론은 컴포넌트는 개별 단위로 독립적으로 개발되어서 배포 될 수 있는 응집력 있는 소프트웨어 패키지로서 다른 컴포넌트들과 조립할 수 있도록 서비스를 제공하는 인터페이스와 사용하는 인터페이스가 정의 되어 있어야 한다고 제시 하고 있고 또한 여기서 하나의 컴포넌트 패키지에는 제공하는 인터페이스 목록, 사용하는 인터페이스 목록, 컴포넌트 명세서(스펙), 실행 코드, 확인 코드 설계 등이 포함되어 있어야 한다고 제시하고 있다.[2]

한 컴포넌트가 재사용성이 높아지기 위해서는 여러 유사 응용 프로그램들 간의 공통적인 부분들이 구현되어 있어야 할 뿐만 아니라 각각의 응용 프로그램마다 필요에 따라 특화될 수 있도록 하는 메커니즘을 제공해야 한다. 이러한 방법

론이 다양하게 제시되고 있고 계속 적으로 연구가 되어 있다.

플러그인 기술은 CBD 방법론에서 제시 되었던 컴포넌트를 재사용을 하기 위해 제시되는 방법으로, 개발되어 있는 프레임워크에 어떤 컴포넌트를 연결하면 그것이 적용되는 기술이다. 하나의 전기 콘센트에 여러 플러그를 연결 하듯이 끼워질 플러그인의 형식에 맞게 개발되어진 컴포넌트를 프레임워크에 끼우기만 하면 바로 그 프레임워크에 적용되어 하나의 프로그램으로 구성 되어지는 것이다. 이 플러그인 기술이 가장 잘 적용되어진 예로는 자바 개발 도구인 이클립스(Eclipse)가 있다. 이클립스는 오픈 소스이며 자바 기반의 확장 가능한 개발 플랫폼이다. 그 자체로서가 프레임웍이며 플러그인 컴포넌트에서 개발 환경을 구현하는 서비스 세트이다. 그림 4는 플러그인 기술을 적용한 이클립스의 구조도를 나타낸다.

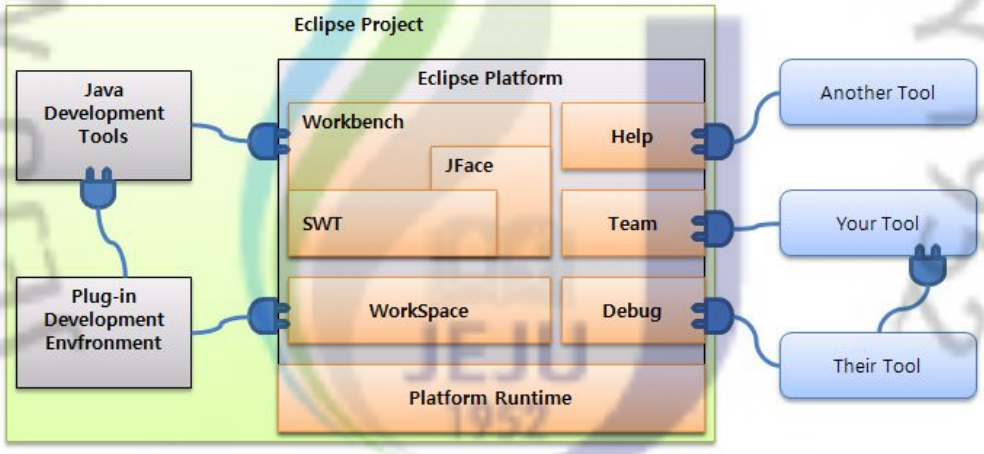

그림 4. 이클립스 플러그인 구조도

각 플러그인은 콘센트에 플러그를 연결하듯 플랫폼에 연결되어 있으면 그 플러그인이 역할을 수행하는 것이고, 그것이 없더라도 전체 플랫폼의 동작에는 아무 문제가 없다. 이 기술을 사용하면 무한한 확장성과 유연성을 가질 수 있게 된다.

4. 커스터마이제이션 기법

CBD(Component Based Development) 방법론들은 커스터마이제이션이 필요성

에 대해 언급하고 있으며 개념적인 수준에서 접근 방법을 제시 하고 있다. 커스터마이제이션에 대해 언급하고 있는 방법론으로 Catalysis와 Componentware가 있는데 컴포넌트의 가변성을 정의 하여 커스터마이제이션에 대해 제시하고 있다.

Catalysis는 모델 간에 일관성 규칙을 잘 정의 하고 있으며 복잡한 시스템을 설계하는데 다양한 뷰(View)를 제공하기 위한 기법을 정의하고 있다. 추가적으로 Catalysis는 다른 방법론과는 다르게 프레임워크를 사용하여 컴포넌트를 개발할 수 있는 개발 프로세스를 정의 한다. Catalysis 의 프로세스는 요구사항 분석, 시스템 스펙, 아키텍처 설계, 그리고 컴포넌트 내부 설계 단계로 구성되며 설계부터 소스 코드에 이르는 전 과정의 추적성(Trace-ability)과 정확성(Precision)을 보장하고 재사용 측면에서는 소스 코드뿐만 아니라 설계 산출물까지 재사용할 수 있도록 한다. 그림 5는 추적성과 정확성을 보장하기 위한 Plug-Points를 나타내고 있다.

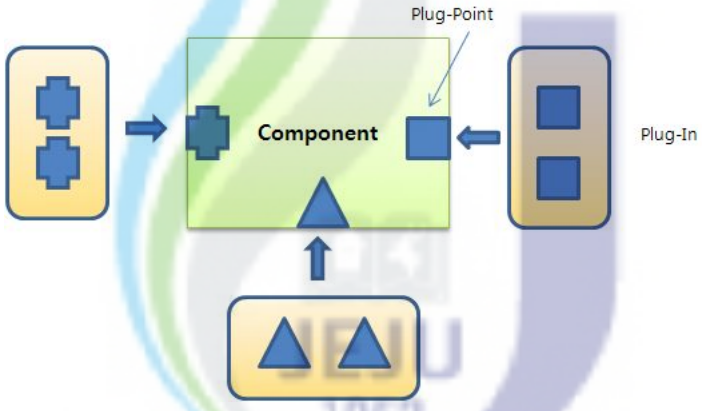

그림 5. Plug-Points

Componentware는 커스터마이제이션을 위해 Catalysis의 "Required Interface"와 유사한 "Import Interface"를 정의한다. 이 인터페이스도 컴포넌트 외부의 다양한 요구사항을 충족시킬 수 있도록 한다. Componentware는 일반적으로 컴포넌트에서 제공되는 인터페이스를 "Export Interface"로 정의하며 외부 컴포넌트에 제공되는 인터페이스를 "Import Interface"로 정의한다. 이와 같이 "Export Interface"와 "Import Interface" 모두는 컴포넌트 사이와 연결을 가능하게 하며 가변성을 제공할 수 있다.[2]

동적으로 커스터마이제이션을 하기 위해 임베디드 시스템의 가변부를 제공하기 위한 기법으로는 선택기법과 플러그인 기법을 이용할 수 있다. 선택 기법은

가변부에 대해 선택 가능한 기능을 설계하여 임베디드 시스템 외부에서 선택하는 기법이다. 선택기법은 매개 변수화로 가능하며 플러그 인 기법은 임베디드 시스템 내부로 변경기능(디바이스 드라이버 등)을 입력하여 임베디드 시스템 내부의 기능을 변경하는 기법[1]을 제시 하고 있다.

커스터마이제이션을 하기 위해 CBD방법론이나 Catalysis기법, Componentware 기법 등 많은 방법론과 기법들을 제시하거나 연구되어지고 있다. 이러한 연구에 컴포넌트 자체의 범용성과 재사용성을 초점을 두고 연구를 하고 있으나 컴포넌트 내부 메시지 흐름의 동적 변경을 위한 설계 기법을 제시 하거나 구현을 하지는 못하고 있다. 기법[1]에서는 동적 커스터마이제션에 대해서 설계를 하고 있으나 실제 구현을 통해 제시된 기법이 어느 정도의 효과를 볼 수 있는지는 연구가 미흡한 실정이다.

III. 맞춤형 임베디드 미들웨어 생성 시스템

임베디드 시스템을 필요로 하는 곳에 적용을 위하여 시스템에 특성상 새로운 디바이스를 지원할 수 있도록 설계 하거나 제작을 해야 하는데, 임베디드 시스템은 확장성을 고려되지 않은 경우 시스템을 재구성 또는 재구축해야 하는 어려움이 가지고 있다. 현재의 일반적인 임베디드 시스템은 개발 생산성 및 재사용성이 매우 낮으며, 이러한 임베디드 시스템은 유지보수 및 관리에 대한 비용 증가와 개발비 증가, 시스템의 품질이 떨어지는 문제점을 가지고 있다.

본 논문에서는 이와 같은 임베디드 시스템의 문제를 극복하고자 임베디드 시스템의 재사용성을 높이고 유지보수 및 관리에 용의 하게 하려는 방법을 제시하고 구현한다.

1. 시스템 개요

임베디드 시스템은 최근 유비쿼터스 기술과 맞물려 작고 고성능화 되고 있으며 이에 따른 사용자들의 요구사항들이 증가하고 있다. 또한 시스템의 업그레이드 및 USN 기술의 발전과 더불어 임베디드 시스템의 확장성이 점차로 중요시되고 있어 이에 따른 임베디드 시스템의 재사용 및 확장성에 관한 요구가 늘어나고 있다. 즉, 임베디드 시스템이 미리 정해진 특정 기능을 수행과 더불어 기술의 발전 및 사용자의 요구를 통한 새로운 기능을 동시에 적용할 수 있는 시스템이 필요로 하게 되었다. 따라서 임베디드 시스템의 소프트웨어를 개발함에 있어 컴포넌트 단위로 개발하고, 이 컴포넌트들을 공유하여 사용하거나 재정의(재배치)하여 임베디드 시스템에 적용할 수 있도록 미들웨어의 개발의 필요하다. 하나의 임베디드 시스템에 사용할 수 있는 컴포넌트들을 공유하여 컴포넌트의 재사용 또는 재배치함으로서 새로이 요구되고 있는 목적에 충족하는 기능을 갖는 새로운 임베디드 시스템을 생성할 수 있게 된다.

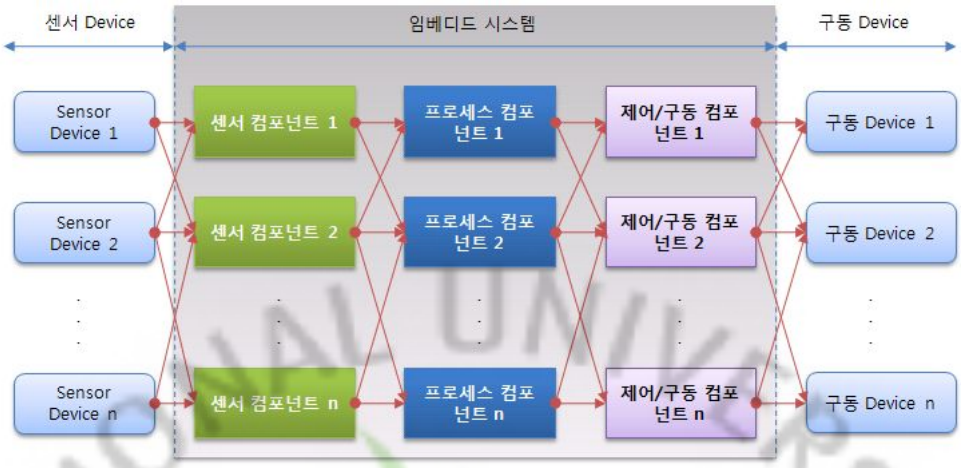

그림 6. 시스템 서비스 구성도

그림 6은 본 논문에서 제안하는 시스템 서비스 구성도로서 센서 Device 에서 센싱된 데이터는 임베디드 시스템의 센서 컴포넌트로 보내어지고, 센서 컴포넌트는 센서 데이터 처리 후 프로세스 컴포넌트로 전송한다. 프로세서 컴포넌트는 센서의 임계값 및 사용자 설정에 따라 구동 Device를 구동 또는 정지 시켜야 할 경우가 발생하게 되므로 이를 수행하기 위하여 제어/구동 컴포넌트로 명령을 보내어 실제 구동 Device를 구동하거나 정지 시킨다. 여기서 센서 Device는 1개에서 n개까지 존재할 수 있으며, 이 센서 정보는 n개의 센서 컴포넌트 중 어느 컴포넌트와도 연결이 가능하게 된다. 센서 컴포넌트 또한 n개의 프로세스 컴포넌트와의 연결이 가능하다. n개의 제어/구동 컴포넌트 역시 n개의 구동 Device에 대한 제어를 할 수 있다.

그림 7. 서비스 구성도

즉, 입력 n개의 컴포넌트에서 n개중 하나 또는 여러개의 프로세스 컴포넌트와 연결을 하고, n개의 프로세스 컴포넌트는 n개중 하나 또는 여러개의 제어/구동 컴포넌트와 연결하여 다양한 서비스를 구성할 수 있게 된다.

그림 7은 본 논문에서 제시하고 있는 미들웨어 서비스 구성으로서 센서 Device로 부터 시작하여 구동 Device 까지 하나의 서비스가 이루어지는 구성도이다. 이러한 서비스 구성으로 컴포넌트를 공유하여 사용하거나 재배치를 통하여 컴포넌트를 다양하게 재사용함으로서 여러 목적에 맞는 서비스를 구성할 수 있도록 하는 것이다.

2. 맞춤형 임베디드 시스템 미들웨어 설계

1) 임베디드 시스템 미들웨어 설계

임베디드 시스템에 탑재할 미들웨어를 설계할 때 고려해야할 사항은 크게 두가지를 들 수 있다. 하나는 임베디드 시스템이 설치된 장소의 환경을 고려하는 것이고, 또 하나는 센서 종류 및 센서 수가 다를 수 있음을 가정하는 것이다. 즉, 각각의 임베디드 시스템이 설치되어진 장소마다 환경이 다르고, 각 환경마다 센서 종류와 센서의 개수가 변경이 되더라도 소프트웨어 변경 없이 사용자 설정만으로 쉽게 적용이 가능하도록 하며, 시스템의 상태 정보를 쉽게 파악할 수 있도록 상태 정보를 로그 기록을 통해 시스템의 동작에 대해 분석을 할 수 있도록 설계한다.

센서의 종류 및 개수와 연결되어진 구동 디바이스가 변경이 되더라도 쉽게 적용할 수 있도록 플러그인 아키텍처 기반을 사용하여 향후 센서나 디바이스가 추가되더라도 추가된 컴포넌트만 업그레이드하여 임베디드 시스템에 탑재하고, 이를 미들웨어가 인식하여 설정된 컴포넌트들을 동적으로 로드 연동할 수 있도록 미들웨어를 설계해야 한다. 플러그인 기반을 사용하기 위해 각 컴포넌트의 인터페이스 구성시 다양한 기능을 고려하여 정보를 표한할 수 있도록 정의한다. 인터페이스 정의는 일반화 정의에 따라 입력 인터페이스, 처리 인터페이스, 출력 인터페이스로 구성하여 컴포넌트를 공유하여 사용할 수 있도록 한다.

그림 8. 임베디드 시스템 미들웨어 구성도

그림 8과 같이 계층별 분류를 보면 제일 하단은 장치 계층으로 각종 하드웨어 및 디바이스 장치가 있고, 그 장치를 관리하기 위한 Device Driver와 임베디드 시스템 OS가 위치하게 된다. 임베디드 미들웨어는 OS 상위에 위치되어 OS를 통해 디바이스와 입출력 명령을 수행하게 되며, 컴포넌트 요소계층, 컴포넌트 연결계층, 관리계층, 응용 인터페이스 계층으로 구성되게 된다.

동적 컴포넌트 연결 모듈은 관리 프로그램에서 설정한 정보를 미들웨어에 전송하면 미들웨어는 센서/구동/프로세스/영상 프로파일 모듈을 통해 동적 컴포넌트로부터 수신된 정보를 통해 프로파일을 생성하여 정보를 저장하고, 이 정보를 가지고 컴포넌트 관리 모듈이 동적으로 로드할 컴포넌트 목록을 작성한 후 컴포넌트를 동적 로드하여 컴포넌트를 관리하게 된다. 로드 되어진 컴포넌트는 동적 컴포넌트 연결 모듈을 통해 프로파일에 설정 되어진 구성으로 컴포넌트 간 연결을 설정하여 새로운 미들웨어를 생성하게 되는 것이다. 주요 모듈별 기능은 표 2와 같다.

모듈	주요기능
센서/구동/프로세스/영상 프로파일 생성 모듈	- 관리 프로그램에서 보내진 정보의 프로파일을 생성하여 관리 - 생성되어진 프로파일을 관리프로그램에 전송
컴포넌트 관리 모듈	- 설정되어진 프로파일 정보를 통해 컴포넌트를 로드하여 컴포넌트를 관리
동적 컴포넌트 연결 모듈	- 입력, 처리, 제어를 담당하는 컴포넌트를 컴포넌트 공통 인터페이스를 통해 연결
장애 관리	- 동작에 필요한 디바이스의 상태를 주기적으로 체크
로그 관리	- 각 컴포넌트들의 로그정보 관리
사건 관리	- 컴포넌트에서 발생하는 이벤트 관리
구성 관리	- 관리 프로그램에 설정한 설정 정보 관리
저장장치 관리	- 임베디드 시스템에서 제공하는 저장장치들에 관한 정보 제공 및 관리
Socket 통신 모듈	- 외부와 Socket 통신 할 수 있는 기능 제공

표 2. 임베디드 시스템 미들웨어 주요 모듈 기능표

임베디드 시스템의 구성은 디바이스 드라이버(Device Driver)와 임베디드 시스템에 탑재한 미들웨어로 구분하여 동작 한다.

그림 9는 임베디드 시스템에 센서, 카메라 등의 디아비스 장치를 장착할 경우 기존 컴포넌트를 공유하여 사용하고, 각 컴포넌트를 재구성하므로 임베디드 시스템을 구축 할 수 있도록 하기 위한 미들웨어 구성도이다. 각 센서부에서 임베디드 시스템에 장착된 물리적 디바이스 장치로부터 센싱된 정보는 디바이스 드라이버를 통해 수신하고 미들웨어 센서 처리부인 센서 컴포넌트로 정보가 들어오게 되며, 센서처리부에서는 센싱된 각각의 정보를 센서 데이터 분석 모듈을 통해 정보를 센서 데이터 구조로 변환 후 데이터 프로세스부인 데이터 조건 처리 모듈로 전송한다. 센싱된 정보는 임계값 체크, 데이터 필터링 또는 스케쥴링을 통하여 이벤트를 생성하게 되고, 구동기인 제어부로 이벤트 명령이 발생하게 된다. 이벤트가 발생하는 정보에 따라 구동시킬 장치를

결정하고 카메라 제어, 사운드 처리, 환기팬 제어 등 구동기 제어를 하게 된다.

이러한 동작을 하기 위해서는 동적 컴포넌트 연결 계층인 컴포넌트 등록, 동적 컴포넌트 연결, 센서/구동/프로세스/영상 프로파일 생성를 통해 컴포넌트를 동적으로 로드하여 구성되어진 프로파일 정보를 기준으로 컴포넌트간 연결 설정을 하고 새로운 임베디드 시스템 미들웨어를 구성하여 동작하게 된다.

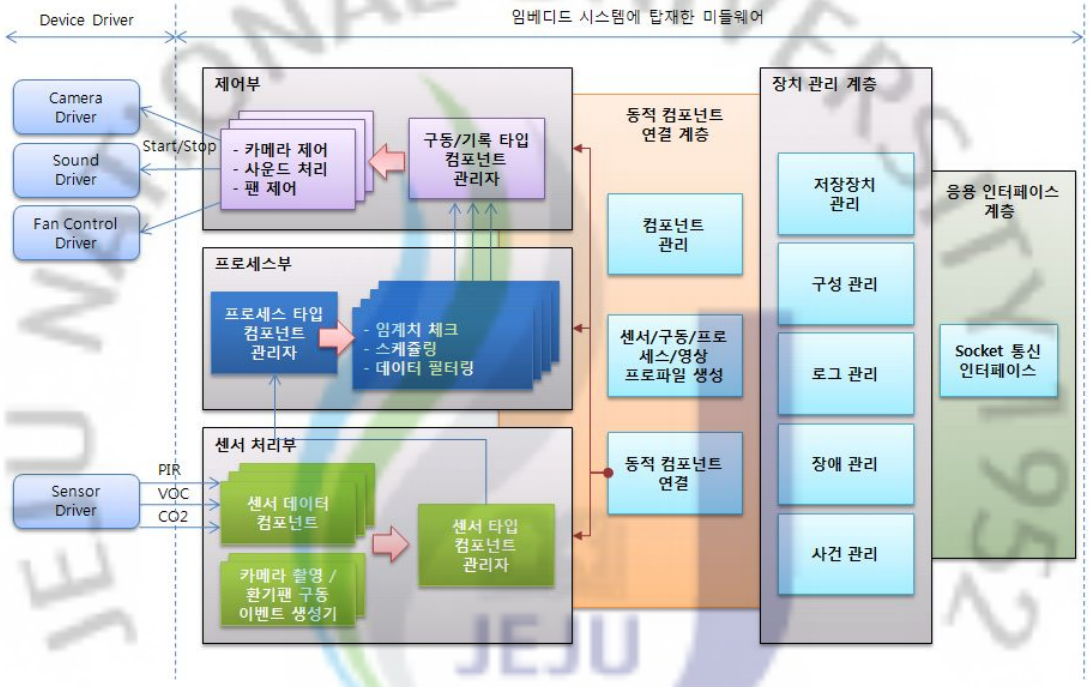

그림 9. 임베디드 시스템 미들웨어 작동 구성도

구성 관리 모듈은 관리 프로그램과 통신을 하기 위해 응용 인터페이스 계층의 Socket 통신 인터페이스 모듈을 사용하여 관리프로그램에 접속하고, 접속 성공 후 임베디드 시스템에 설정되어진 환경 정보들을 센서/구동/프로세스/영상 프로파일 생성 모듈을 통해 관리 프로그램에게 전송하게 된다. 관리프로그램에서는 설정 정보을 새로 작성하거나 갱신하여 미들웨어의 센서/구동/프로세스/영상 프로파일 생성 모듈에게 정보를 전송하고, 관리프로그램이 로그정보 요청시 로그관리 모듈을 통해 임베디드 시스템이 가동 시점에 저장되어진 로그 정보들을 관리프로그램으로 전송함으로서 모니터링 할 수 있게 된

다. 저장장치 관리 모듈은 임베디드 시스템이 저장장치를 결정하는데 여러 가지 저장장치 타입이 있어 하드디스크에 저장할 것인지 CF(Compact Flash)타입 메모리에 저장할 것인지 USB(Universal Serial Bus)타입 메모리에 저장할 것인지 SD(Secure Digital)타입 메모리에 저장할 것인지에 대한 설정 부분과 관리 부분이 필요하기 때문에 이를 관리하기 위한 모듈이다.

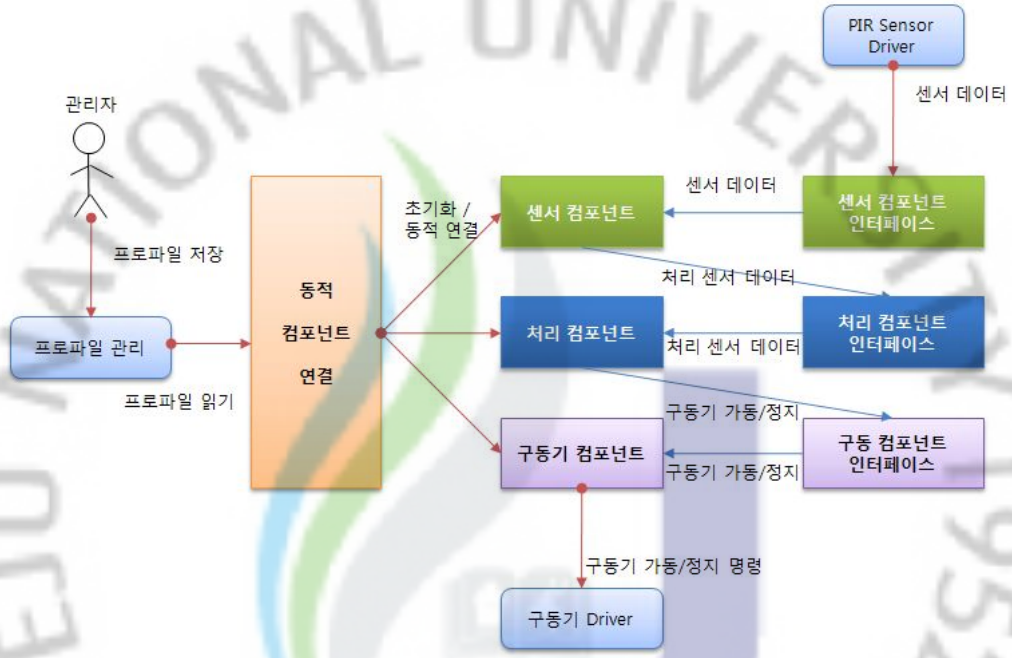

그림 10. 임베디드 미들웨어 데이터 흐름도

그림 10은 임베디드 시스템에서 동적으로 컴포넌트가 로드 되어지는 과정과 센서 디바이스로부터 얻어진 데이터를 가지고 구동기 디바이스를 가동하거나 정지 하는 과정의 컴포넌트 간에 데이터 흐름도 이다. 사용자는 컴포넌트를 조합하여 프로파일을 생성하고 저장하게 된다. 이렇게 저장되어진 프로파일은 임베디드 시스템으로 전송하게 되고 전송되어진 프로파일은 동적 컴포넌트 연결 모듈을 통해 연결하게 된다. 설정되어진 프로파일 정보 기준으로 센서 컴포넌트, 처리 컴포넌트, 구동기 컴포넌트를 동적 로드하여 각 컴포넌트를 초기화를 하게 된다 이 과정은 센서 컴포넌트는 처리 컴포넌트를 연결하게 되고. 처리 컴포넌트는 구동기 컴포넌트와 연결을 하게 되며 연결이 완료 되어진 컴포넌트는 동작을 하게 된다. 센서 디바이스를 통해 들어오는 센서 데이터는 센서 컴포넌트 인터페이스를 통해 센서 컴포넌트로 전달이 되고. 센서

컴포넌트는 센서 데이터를 처리 컴포넌트가 처리할 수 있도록 센서 데이터를 변경하고 처리 되어진 센서 데이터는 처리 컴포넌트 인터페이스를 통해 처리 컴포넌트로 전달하게 된다. 이렇게 전달되어진 데이터는 처리 컴포넌트를 통해 데이터를 임계치 검사, 스케즐링, 데이터 필터링 등을 하여 구동기 가동 여부를 판단하게 되고 판단하여 수행 되어지는 정보는 구동 컴포넌트 인터페이스를 통해 구동기 컴포넌트로 전달이 되며 이 정보에 따라 구동기 디바이스가 동작하게 된다.

2) 관리프로그램 설계

관리 프로그램은 일반 컴퓨터, 노트북, UMPC 등에 많이 사용되고 있는 Windows 기반 운영체제 환경에서 동작 하도록 하며, 하나 이상의 임베디드 시스템 관리를 위한 기능과 임베디드 시스템에 탑재할 미들웨어의 시스템 구성을 변경 하거나 시스템 상태 조회를 위한 기능을 포함하며 간단한 제어 및 시간 동기화와 최신 소프트웨어로 업그레이드 할 수 있는 기능을 제공하도록 한다.

관리 프로그램의 주요 기능은 임베디드 시스템에 사용되는 컴포넌트를 구성하거나 편집 또는 삭제 할 수 있으며, 시스템 상태를 조회할 수 있는 시스템 상태 정보 기능과 임베디드 시스템을 제어 하는 기능으로서 스피커 보륨 조절 및 시스템 소프트웨어를 시작하거나 정지 할 수 있으며 시스템을 재부팅 할 수 있는 장치 제어 기능, 임베디드 시스템에 음원 파일을 업로드 할 수 있는 파일 관리 기능, 임베디드 시스템에 기록되어진 영상을 기간별 조회 하거나 인코딩 할 수 있는 영상 검색 및 인코딩 기능, 임베디드 시스템에 작성되어진 로그 정보를 기간별 조회 하는 로그 검색 기능 갖는 관리프로그램을 설계한다.

관리 프로그램과 임베디드 시스템 사이의 통신은 무선랜(Wifi)을 이용하며 Ad-hoc 모드로 관리 프로그램과 임베디드 시스템 간 연결설정을 통하여 관리 프로그램에서 주요 명령어를 전송하며 임베디드 시스템은 각 명령어에 따른 동작을 수행하고, 수행결과를 관리 프로그램으로 전송할 수 있도록 설계

한다.

그림 11은 시스템 사용자인 관리자(Actor)가 관리 프로그램을 통한 기능조작으로 인한 이벤트 발생으로 임베디드 시스템에 탑재한 미들웨어의 처리 기능을 정의한 유즈케이스(Use Case Diagram) 다이어그램 이다.

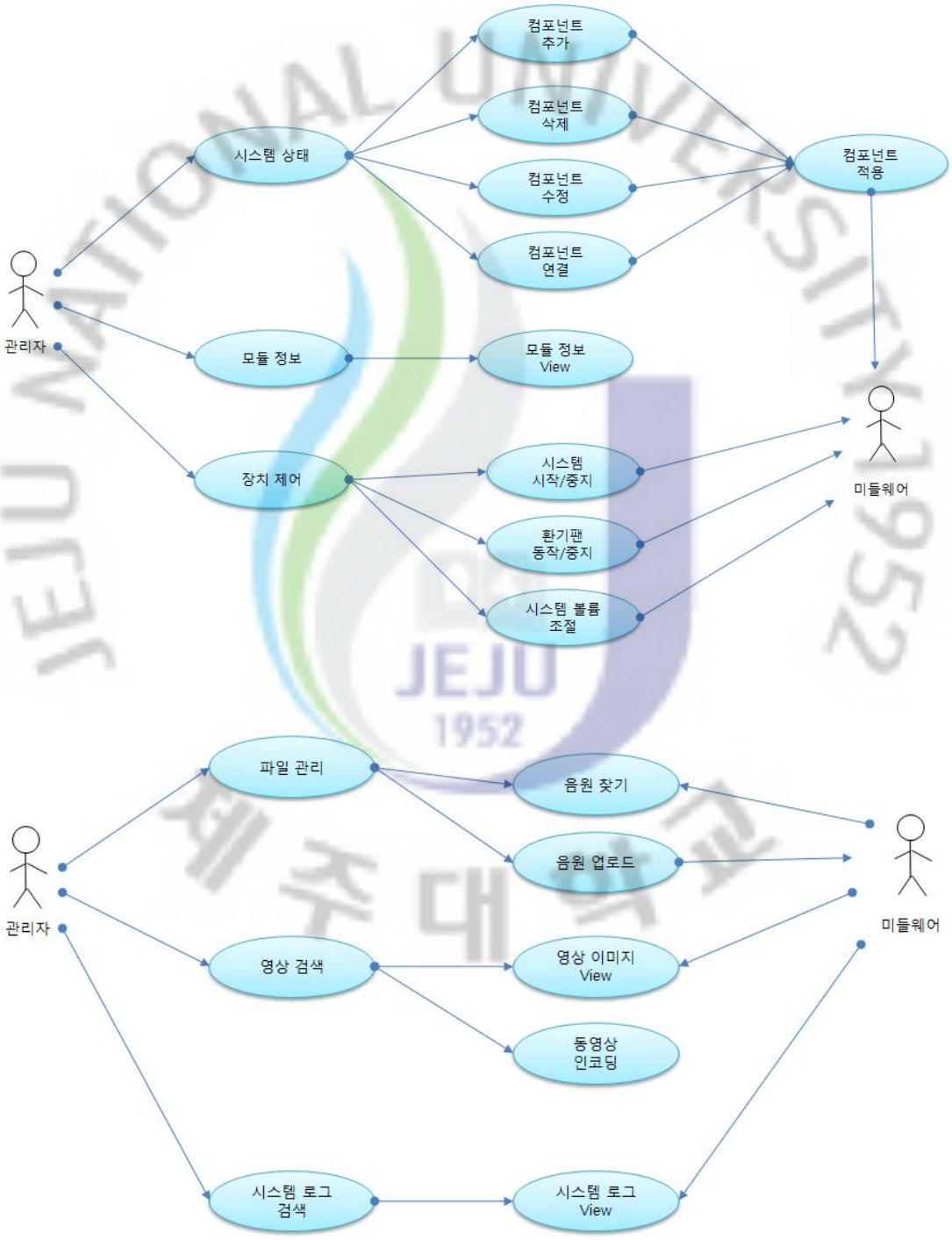

그림 11. 관리프로그램 Use Case 다이어그램

3) 컴포넌트 동적 로드를 위한 인터페이스 정의

2장에 기술한 CBD 방법론에서 컴포넌트 공유를 위해서는 유사 응용 프로그램들 간의 공통적인 부분들이 구현되어 있어야하고, 각각의 응용 프로그램마다 특화된 메커니즘이 필요하다고 하였다. 이런 이유로 임베디드 시스템 적용을 위해 응용 프로그램들 간의 인터페이스 정의가 반드시 필요하다.

임베디드 시스템에 탑재한 미들웨어는 동적으로 컴포넌트 로드를 담당하는 모듈이 있다. 응용 프로그램 간의 인터페이스는 미들웨어가 컴포넌트 동적 로드를 위해 센서 타입 컴포넌트, 처리기 타입 컴포넌트, 구동기 타입 컴포넌트, 기록기 타입 컴포넌트를 서로 연결하기 위한 공통적인 부분들을 찾고 컴포넌트 관리자가 관리 할 수 있도록 정의가 필요하다. 이러한 과정에서 센서 타입 컴포넌트, 처리기 타입 컴포넌트, 구동기 타입 컴포넌트, 기록기 타입 컴포넌트 간에 연결을 위해 일반화 작업이 필요하게 되는데, 이러한 일반화 정의에 따라 센서 인터페이스, 처리 인터페이스, 출력 인터페이스를 정의 하도록 한다. 여기서 출력명령에 디바이스를 제어와 시스템 내부처리(결과 저장)는 인터페이스 분리가 필요하고, 이는 시스템 내부처리에 제한된 저장장치의 효율적인 관리를 위해 리소스를 정의하고 최적화 하는 추가적인 인터페이스가 필요하기 때문이다. 그림 12은 각 컴포넌트들을 서로 연결하고 관리하기 위하여 공통 인터페이스로 연결을 설정하고 컴포넌트 관리 모듈을 통해 관리가 되는 인터페이스 연결 구성도 이다.

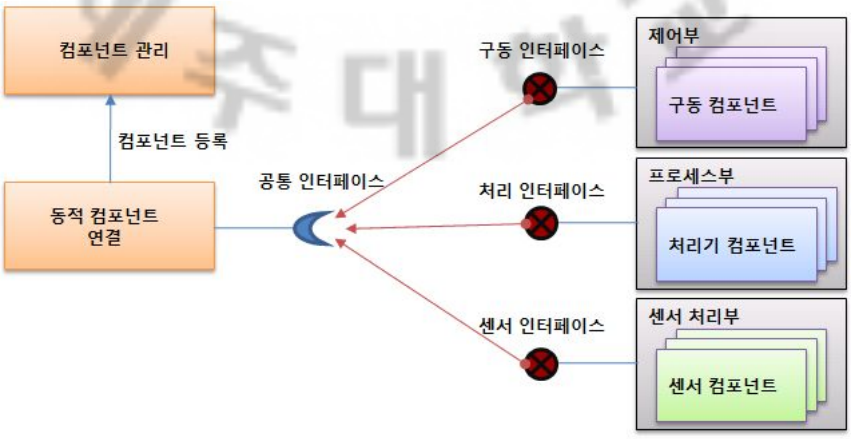

그림 12. 인터페이스 연결 구성도

컴포넌트 관리자가 등록 및 관리를 위한 필요한 정보로는 등록 컴포넌트의 버전 정보를 관리할 수 있는 정보, 현재 지정되어진 컴포넌트가 센서부 컴포넌트인지 처리부 컴포넌트인지 출력부 컴포넌트인지 구분할 수 있는 코드와 컴포넌트 타입정보, 컴포넌트가 연결되어지는 디바이스의 제조사 정보, 컴포넌트를 구분할 수 있는 네이밍 정보, 컴포넌트의 내부 정보를 프로파일로 저장하거나 읽어 올 수 있는 기능, 현재 컴포넌트가 환경 설정을 할 수 있는 컴포넌트인지 아닌지 여부 정보, 컴포넌트에 연결되어진 디바이스 컴포넌트의 환경 설정 가능 여부, 현재 연결되어진 디바이스가 사용하고 있는 채널 정보, 컴포넌트의 동작 상태를 파악할 수 있는 컴포넌트 상태정보가 있다. 또한 현재 컴포넌트의 상태를 알려줄 수 있는 이벤트 설정정보는 저장하고 읽어 올 수 있는 기능이 필요하다. 이러한 정보들을 표현하고 관리하기 위해 다음과 같이 컴포넌트 공통 인터페이스가 필요하다. 공통 인터페이스는 새로운 컴포넌트를 설계 하거나 개발 시 반드시 적용해야 한다. 표 3은 각 컴포넌트의 공통적인 주요 기능 및 내용을 정의한 컴포넌트 인터페이스다.

인터페이스 항목	주요 기능 및 내용
공통 I/F	- 컴포넌트 버전정보, 타입정보, 구분정보, 환경설정 가능 여부, 채널 정보, 상태 정보, 설정 정보 - 컴포넌트 설정정보 읽기 기능, 설정정보 저장 기능
센서 I/F	- 초기화 기능, 시작 기능, 중지 기능, 닫기 기능 - 현재 동작 여부
프로세스 I/F	- 초기화 기능, 구동기 초기화 기능, 데이터 처리 기능
처리기 I/F	- 초기화 기능, 시작 기능, 중지 기능, 닫기 기능 - 현재 동작 여부
기록기 I/F	- 초기화 기능, 시작 기능, 중지 기능, 닫기 기능, 이미지/리소스 파일 정리 기능, 이미지 파일 검색 기능 - 현재 동작 여부

표 3. 컴포넌트 인터페이스 정의

센서타입 컴포넌트는 컴포넌트 초기화 기능, 컴포넌트 가동 기능, 컴포넌트 일시 중지 또는 종료 기능, 현재 컴포넌트의 동작 여부를 알 수 있는 기능의 인터페이스 정의가 필요하다.

처리 타입 컴포넌트는 Actuator 컴포넌트의 초기화 기능과 Recorder 컴포넌트 초기화 기능, Actuator와 Recorder 모두를 초기화 할 수 있는 기능, 센서 컴포넌트로부터 들어오는 데이터를 처리할 수 있는 기능의 인터페이스 정의가 필요하다.

출력(Actuator) 타입 컴포넌트는 컴포넌트 초기화 기능과, 컴포넌트를 가동 기능, 컴포넌트를 일시 중지 또는 종료 기능, 현재 컴포넌트의 동작 여부를 알 수 있는 기능의 인터페이스 정의가 필요하다.

기록(Recorder) 타입 컴포넌트는 초기화 기능과, 컴포넌트 가동 기능, 컴포넌트의 일시중지 또는 종료 기능, 현재 컴포넌트의 동작 여부를 알 수 있는 기능, 이미지 파일과 같은 리소스 파일을 삭제하는 기능, 현재 기록 되어진 파일 개수를 알 수 있는 기능, 기록된 파일 목록을 전체 또는 기간별로 검색할 수 있는 기능의 인터페이스 정의가 필요하다.

이러한 인터페이스 정의는 각각의 컴포넌트를 연결하고 컴포넌트를 관리 하는데 필요하며 동적 컴포넌트 연결 모듈을 통해 각각의 컴포넌트를 연결하고, 연결되어진 컴포넌트는 컴포넌트 관리 모듈을 통해 관리된다. 이처럼 서로 연결되어진 컴포넌트 조합을 통하여 새로운 환경에 적용 할 수 있는 임베디드 미들웨어가 구성된다.

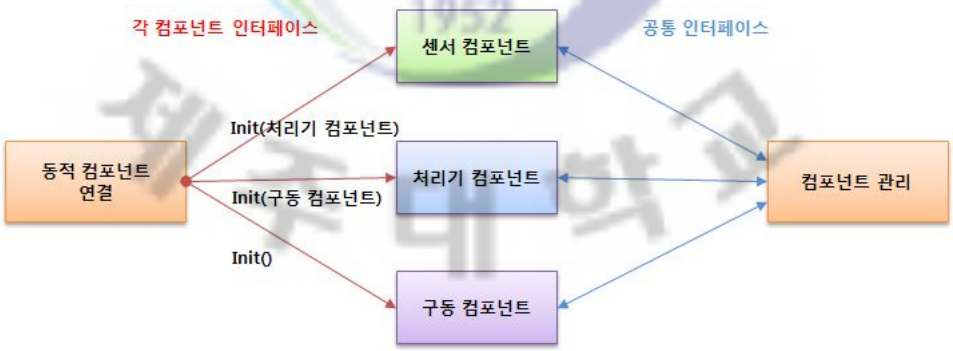

그림 13. 인터페이스 통한 컴포넌트 연결

그림 13와 같이 동적로드를 위한 컴포넌트들간에 연결은 컴포넌트 인터페이스를 통하여 컴포넌트를 연결하고 공통 인터페이스를 통해 컴포넌트를 관리하게 된다. 동적 컴포넌트 연결 모듈은 각각의 컴포넌트인 센서 컴포넌트, 처리기

컴포넌트, 구동 컴포넌트를 로드하기 전 공통 인터페이스와 각 컴포넌트 인터페이스가 구현되어 있는지 확인 후 동적로드하게 되며 로드 되어진 센서 컴포넌트는 Init(처리기 컴포넌트) 초기화 명령을 통해 센서 컴포넌트와 처리기 컴포넌트를 연결하는 초기화 명령을 하게 되며 동적 로드 되어진 처리기 컴포넌트는 Init(구동 컴포넌트) 초기화 명령을 통해 처리기 컴포넌트와 구동 컴포넌트를 연결하는 초기화 명령을 하게된다. 구동 컴포넌트는 Init() 초기화 명령으로 구동 컴포넌트를 초기화 하게된다. 이러한 과정을 통해 센서 컴포넌트와 처리기 컴포넌트를 연결하게 되고, 처리기 컴포넌트와 구동 컴포넌트를 연결하게 된다. 동적 컴포넌트 연결 모듈은 이렇게 연결되어진 컴포넌트들을 컴포넌트 관리 모듈에 등록하여 컴포넌트를 관리하게 되며, 컴포넌트 관리 모듈은 각 컴포넌트들이 가지고 있는 공통 인터페이스를 통해 컴포넌트의 정보와 컴포넌트의 상태 정보, 컴포넌트 설정 정보 등을 관리 할 수 있게 된다.

4) 주요 컴포넌트/ 모듈 설계

(1) 동적 컴포넌트 연결 모듈 설계

동적 컴포넌트 연결 모듈은 센서 컴포넌트, 프로세스 컴포넌트, 구동기 컴포넌트를 동적으로 로드하여 컴포넌트간 연결을 설정하고 관리 하는 부분으로서 컴포넌트를 공유하기 위해 매우 중요한 모듈이다.

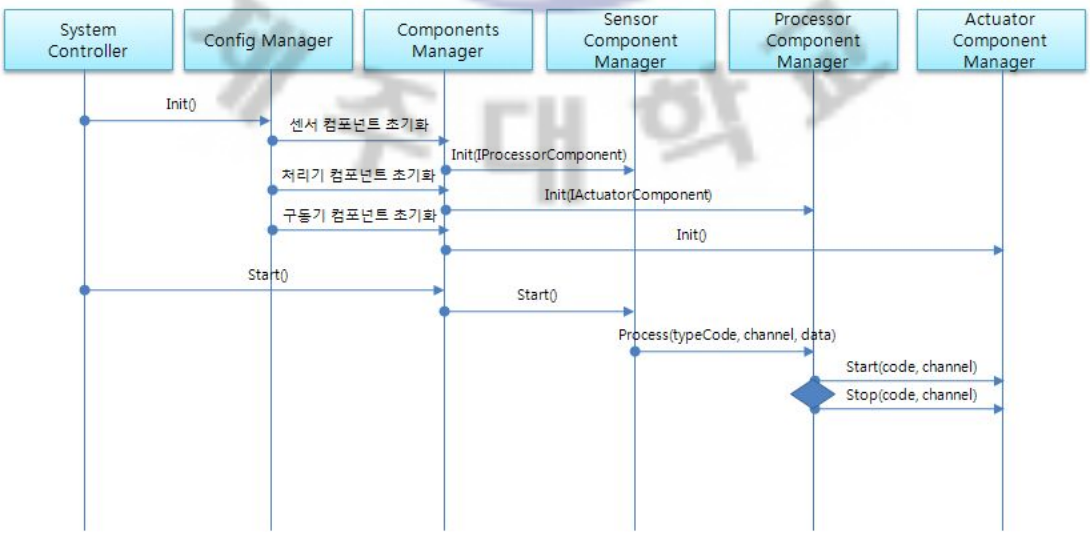

그림 14. 맞춤형 임베디드 시스템 구성 시퀀스 다이어그램

동적 컴포넌트 연결 모듈은 센서/구동/프로세스/영상 프로파일 생성 모듈로부터 로드 되어야 할 컴포넌트 정보를 요청하여 결과를 전송받고, 받은 컴포넌트 정보를 바탕으로 컴포넌트를 로드한다. 로드된 컴포넌트는 앞에서 정의한 인터페이스로 연결하게 되고 연결이 완료된 컴포넌트는 초기화를 진행하게 된다. 초기화가 모두 완료된 컴포넌트는 시스템에서 사용하기 위한 작업으로 컴포넌트 관리 모듈로 등록하여 컴포넌트를 가동하게 된다.

그림 14는 컴포넌트를 동적으로 로드하여 각 컴포넌트의 인터페이스 정보로 초기화 하고 컴포넌트를 연결 설정하여 구동하기까지의 시퀀스 다이어그램이다. 컴포넌트를 동적로드하여 각 Type의 컴포넌트를 초기화 하고 컴포넌트 간 연결설정 후 구동기의 동작시점까지의 시퀀스 다이어그램을 나타낸 것이다. System Controller를 통해 초기화 명령을 내리고 Config Manager 는 센서 컴포넌트와 처리기 컴포넌트, 구동기 컴포넌트를 각각 동적 로드된 정보를 통해 초기화를 수행한다. 초기화가 완료되면 System Controller는 Start() 명령을 Components Manager에게 보내고 Sensor Component Manager에서 센서 데이터를 가공하여 Processor Component Manager로 전송하여 조건에 따라 Actuator Component Manager를 동작 또는 중지 여부를 판단하여 이벤트를 발생하게 된다.

그림 15. 맞춤형 임베디드 시스템 구성 예

그림 15은 실제 컴포넌트 별 시스템을 구성해 본 예 이면 이러한 다양한 연결설정으로 다양한 임베디드 미들웨어를 구성 할 수 있다. 첫번째 PIR Sensor가 인체를 감지한 후 센싱 컴포넌트를 통해 데이터 처리기 컴포넌트로 전송을 하고 음원 구동 컴포넌트는 처리 로직을 수행하여 사운드 처리 컴포넌트를 통해 사운드를 재생을 하게 된다. 두 번째 VoC Sensor와 CO2 Sensor가 감지한 데이터는 센싱 컴포넌트를 통해 임계점 체크 컴포넌트로 전송하고, 지정된 임계치가 초과 하였는지 검사하여 임계값을 초과했을 경우 환기팬을 구동 하게 된다. 세 번째 환기팬 스케줄에 따라 동작하기 위해 환기팬 이벤트 스케줄러 컴포넌트에서 이벤트를 생성하고 생성되어진 이벤트는 환기팬 구동 컴포넌트를 통해 환기팬을 구동하게 된다. 네 번째는 카메라 촬영 이벤트 생성 컴포넌트를 통해 이벤트가 생성되고, 카메라 촬영 컴포넌트로 생성된 이벤트를 보내어 카메라 촬영 컴포넌트를 거쳐 이벤트 마다 영상촬영을 하게 된다.

컴포넌트의 공유를 위한 설계를 위해서는 각 타입별 컴포넌트의 일반화된 정보가 필요하다. 이러한 정보를 표현하기 위해서 인터페이스는 입력 인터페이스, 처리 인터페이스, 출력 인터페이스가 필요하게 된다.

(2) 센서/구동/프로세스/영상 프로파일 생성 모듈 설계

3절에서 정의한 동적로드 컴포넌트의 클래스 정보 및 인터페이스 정보를 저장하거나 관리하기 위해서 반드시 고려해야 하는 요소가 있는데, 각 컴포넌트별 성격이 다양하므로 본 논문에서는 이러한 문제를 해결하기 위해 메타정보인 Xml를 사용하여 데이터 생성 및 저장 방법을 사용하였다. 즉, 각각의 컴포넌트 간 연결정보 및 속성을 메타정보로 변환하여 저장한다.

인터페이스 정의에 있어 컴포넌트의 설정정보 및 연결정보 등을 메타정보로 변환하는 인터페이스만 정의하게 되면 각각의 성격이 다른 컴포넌트의 정보들을 저장하거나 관리 할 수 있다. 즉, 독립적인 데이터를 저장하고 관리하기 때문에 차후 다양한 플렛폼에 적용이 용의하게 된다.

그림 16는 컴포넌트의 메타 정보를 관리 프로그램으로 전송하고, 수정되어진 메타정보를 이용하여 미들웨어의 프로파일을 생성하기까지의 시퀀스 다이어그램을 나타낸 것이다. 미들웨어는 주기적으로 관리 프로그램으로 접속을

시도하여 접속이 성공한 후 설정되어진 프로파일 정보를 프로파일 생성 모듈을 통해 관리프로그램으로 전송하게 된다. 전송된 정보를 통해 관리프로그램은 컴포넌트 설정과 연결 정보를 재설정하거나 재배치하여 임베디드 시스템에 탑재된 미들웨어로 설정 정보를 전송하며 미들웨어는 프로파일 생성 모듈을 통해 프로파일을 생성하고 저장하게 된다.

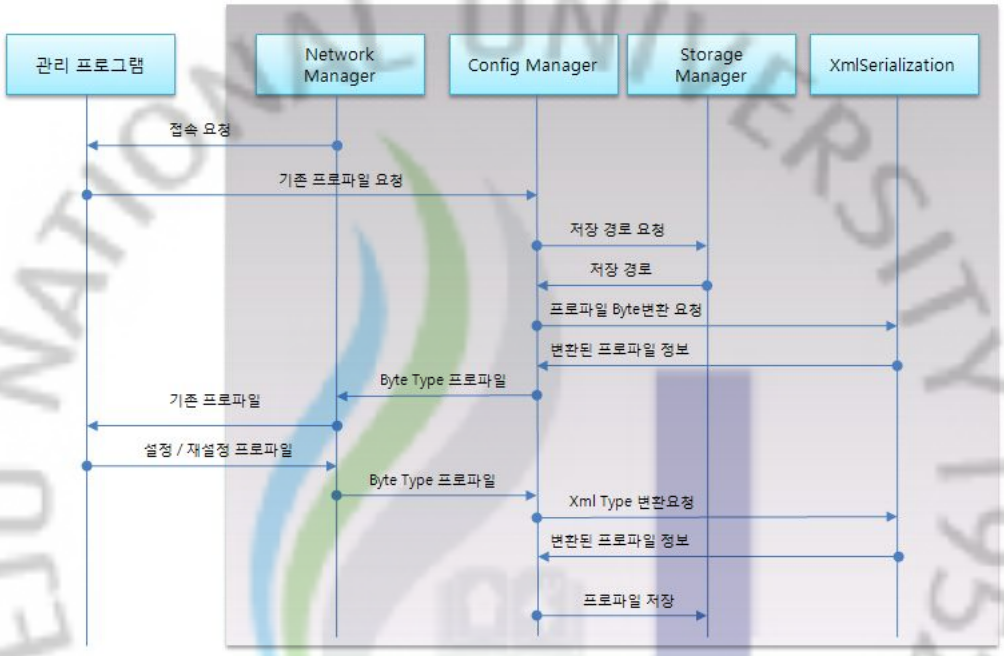

그림 16. 프로파일 생성 모듈 시퀀스 다이어그램

NetworkManager는 주기적으로 관리프로그램에게 접속을 요청하고 접속이 성공적으로 되었을 경우 관리프로그램은 기존 프로파일 정보를 Config Manager에게 요청을 한다. Config Manager은 Storage Manager를 통해 프로파일이 저장되어진 경로 프로파일 정보를 읽어 오게 되며 읽어온 정보를 관리프로그램에게 전송하기 위해 직렬화 작업을 XmlSerialization에게 요청하게 된다.

XmlSerialization는 프로파일 정보를 직렬화하여 Config Manager에게 알려주고, 이 정보는 관리 프로그램에게 NetworkManager를 통해 전송하게 된다. 관리프로그램은 기존 프로파일 정보를 바탕으로 컴포넌트들을 설정하거나 재설정하여 프로파일 정보를 NetworkManager를 통해 ConfigManager에게 전송하게 되고, Byte를 Xml로 변환해주는 XmlSerialization에게 요청하여 역직

렬화 과정을 거친 후 Xml 정보를 Store Manager를 통해 프로파일로 저장하게 된다.

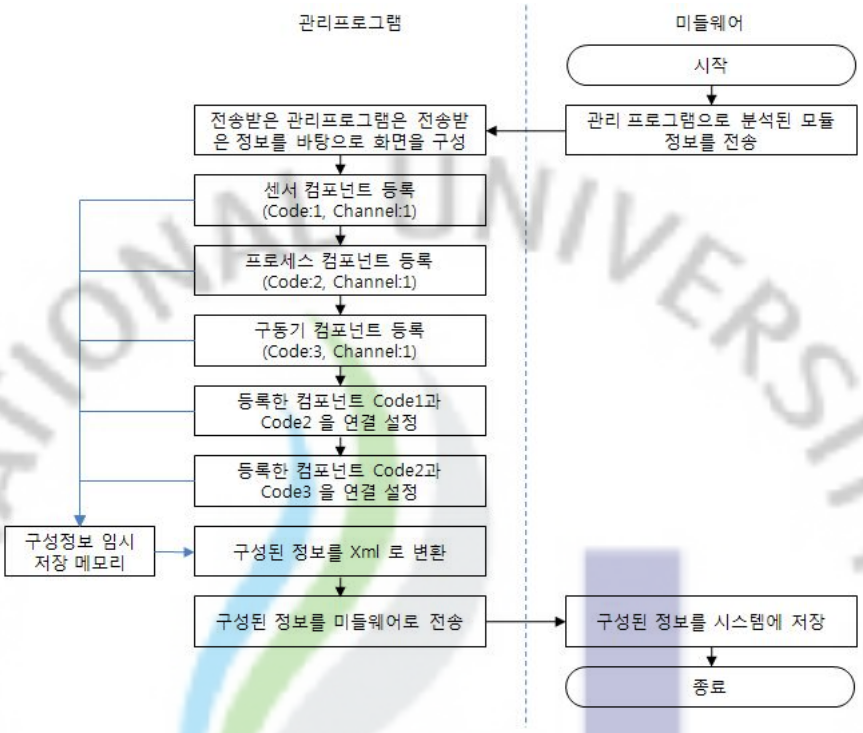

그림 17. 프로파일 생성 순서도

그림 17은 관리프로그램에서 프로파일을 설정하고 생성하는 순서도이다. 미들웨어에서 관리 프로그램으로 접속을 시도 후 접속이 성공하면 미들웨어에서 기존 분석된 컴포넌트 정보를 전송하게 된다. 컴포넌트 정보를 전송 받은 관리프로그램은 받은 정보를 바탕으로 관리 프로그램의 UI를 구성하게 되며 UI 구성 절차는 센서 컴포넌트를 작업창에 등록-> 프로세스 컴포넌트 등록 구동기 컴포넌트 등록 순으로 진행하게 되고 사용자가 모듈 간 드래그를 통한 연결 설정을 한다. 이렇게 설정한 정보는 임시 저장 장소인 메모리로 저장이 되고 최종 구성이 완료가 되어지면 저장되어진 정보를 Xml로 변환을 하여 미들웨어로 전송하게 된다.

(3) 센서 컴포넌트 설계

센서 컴포넌트는 PIR Sensor, VoC Sensor, CO2 Sensor 등 디바이스로 부

터 오는 데이터를 프로세스 컴포넌트가 처리할 수 있도록 데이터를 가공하거나 Log정보를 저장하는 부분을 처리한다. 컴포넌트 공유하여 사용하기 위해서는 일반화 되어진 정보가 필요하므로 앞서 기술한 공통 인터페이스 구현하고 각 센서 데이터 값에 따른 처리 기능이 필요하게 된다.

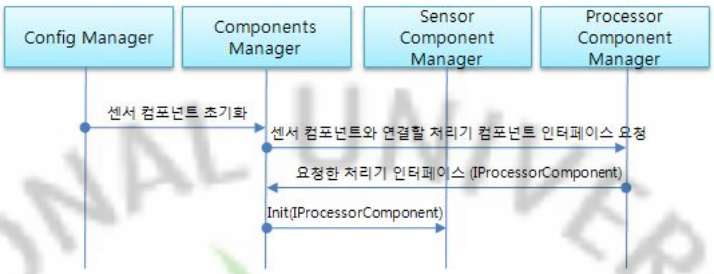

그림 18. 센서 컴포넌트 연결 시퀀스 다이어그램

그림 18는 센서 컴포넌트가 동적 로드 하여 프로세스 컴포넌트와 연결 설정을 하고 센서 컴포넌트가 초기화 하여 사용가능한 시점까지의 시퀀스 다이어그램 이다. 동적 로드된 센서 컴포넌트는 Components Manager에 등록하여 관리가 되고 Config Manager 모듈로부터 컴포넌트 초기화 명령을 통해 컴포넌트 초기화가 시작된다. 컴포넌트 초기화를 위해 Processor Component Manager로 연결 처리할 컴포넌트의 인터페이스를 요청하게 되고, 요청한 결과는 IProcessorComponet 인터페이스를 통해 수신 후 이 인터페이스 정보를 통해 Sensor Component Manager는 Init(IProcessorComponet) 명령으로 센서 컴포넌트를 사용할 수 있는 상태로 초기화 하여 동적 로드를 완료 하게 된다.

(4) 프로세스 컴포넌트 설계

프로세스 컴포넌트는 센서 컴포넌트로부터 가공되어진 정보를 처리하여 구동기를 가동 및 정지 시키는 이벤트를 처리 한다. 이는 센서 컴포넌트 값에 따라 동작 방법이 다를 수 있으며, 주기적으로 처리해야 하는 일들도 발생할 수 있으며 동작 상태 및 데이터 로그 정보를 저장할 수도 있다.

이처럼 프로세스 컴포넌트는 실제 임베디드 시스템에 연결되어진 Deivce 동작 여부를 결정하는 부분으로서 센서 정보의 임계치 체크 또는 스케줄링에 의한 주기적인 이벤트 발생 및 데이터 필터링 기능 등을 처리한다. 프로세스

컴포넌트 역시 센서 컴포넌트와 같이 공유하기 위한 인터페이스 역시 필요하다.

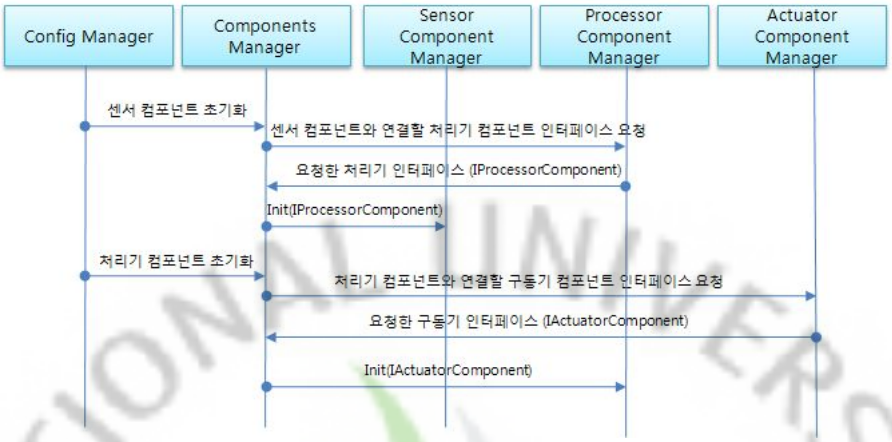

그림 19. 프로세스 컴포넌트 연결 시퀀스 다이어그램

그림 19은 프로세스 컴포넌트를 동적 로드 하여 구동기 컴포넌트와 연결 설정을 하고 프로세스 컴포넌트를 초기화 하여 사용가능한 시점까지의 시퀀스 다이어그램 이다. 동적 로드 되어진 프로세스 컴포넌트는 Components Manager에 등록하여 관리가 되고 Config Manager 모듈로부터 컴포넌트 초기화 명령을 받아 컴포넌트 초기화가 시작된다. 초기화를 위해 Actuator Component Manager로 부터 구동기 컴포넌트의 연결 처리할 컴포넌트의 인터페이스를 요청하게 되고 요청한 결과인 IActuatorComponet 인터페이스를 수신 후 이 인터페이스 정보를 통해Processor Component Manager는 Init(IActuatorComponet) 명령으로 프로세스 컴포넌트를 사용할 수 있는 상태로 초기화를 하여 동적로드를 완료하게 된다.

(5) 구동체 제어/영상 장치 제어 컴포넌트 설계

구동체 제어/영상 장치 제어 컴포넌트의 주요 기능은 임베디드 시스템에 연결되어진 Device를 동작 또는 중지 명령을 처리하는 것이다. 예를 들어 인체 감지 센서를 통해 인체가 감지된 경우 프로세스 컴포넌트를 통해 감지 여부 판단과 안내 방송 재생이 가능한지 판단하여 구동체 제어 컴포넌트로 구동 이벤트를 전달하고, 구동체 제어 컴포넌트는 안내 방송을 가동하거나 중지하

는 명령을 처리하게 된다.

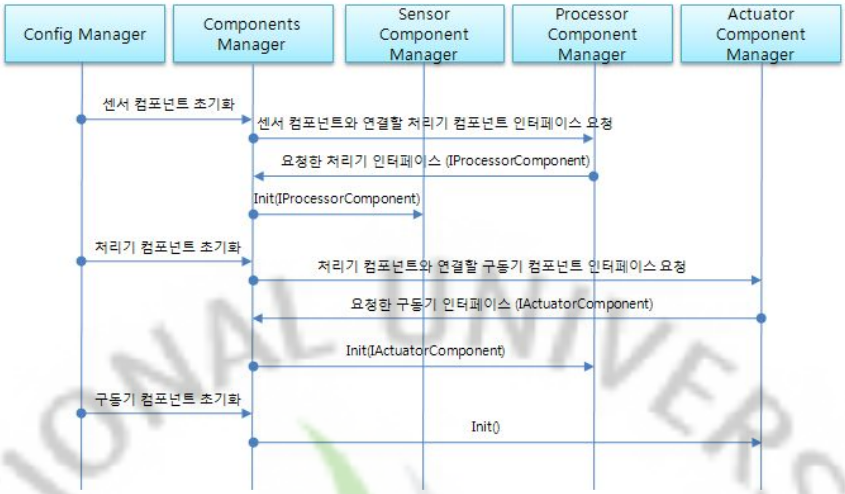

그림 20. 컴포넌트 인터페이스 연결 시퀀스 다이어그램

그림 20은 센서 컴포넌트로부터 시작하여 구동기 컴포넌트의 초기화 까지 모든 컴포넌트의 연결 처리과정을 시퀀스 다이어그램으로 표현한 것이다. 즉, ConfigMagager가 센서 컴포넌트 초기화를 Components Manager에게 요청하면 ComponentsManager는 ProcessorComponentManager에게 연결할 처리기 컴포넌트 인터페이스를 요청한다. ProcessorComponentManager는 처리기 인터페이스인 IProcessor를 결과로 리턴하고, 이 인터페이스 정보로 센서 컴포넌트에 Init(IProcessor) 명령을 통해 초기화를 하면서 센서 컴포넌트와 처리기 컴포넌트 연결 설정이 완료 된다. 처리기 컴포넌트 역시 초기화를 요청하게 되면 ActuatorComponentManager는 구동기 인터페이스인 IActuator를 결과로 리턴하고 Init(IActuator) 명령을 통해 초기화를 하면서 처리기 컴포넌트와 구동기 컴포넌트 역시 연결 설정이 완료된다. 구동기 컴포넌트의 초기화 명령인 (Init())가 수행되면서 모든 컴포넌트의 연결설정이 완료되게 된다.

그림 21은 센서 디바이스에서 수집된 데이터를 처리하여 환풍기 디바이스를 구동하기까지의 컴포넌트 동작 순서도이다. 센서 디바이스에서 수집 되어진 데이터는 선세 컴포넌트 인터페이스를 통해 센서 컴포넌트로 이동을 하게 되고 이 데이터는 프로세스 컴포넌트가 처리할 수 있도록 센서 컴포넌트에서 가공이 되어 프로스세 컴포넌트 인터페이스를 통해 프로세스 컴포넌트로 이동하게 된다. 이 정보를 기준으로 구동기를 가동을 할지 정지 할지 결정을 하

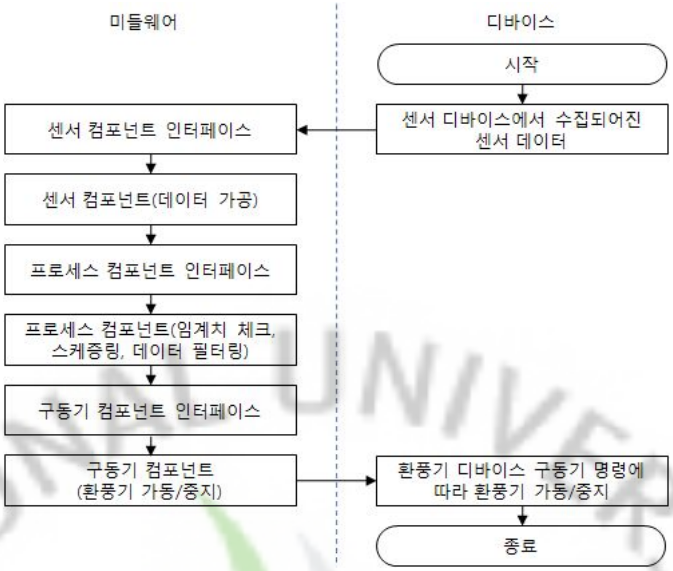

그림 21. 컴포넌트 동작 순서도

여 구동기 컴포넌트 인터페이스를 통해 구동기 컴포넌트로 명령을 전송하게 되며 전송되어진 명령은 구동기 컴포넌트를 통해 환풍기 디바이스를 구동하거나 정지하는 명령을 수행하는 절차로 진행하게 된다.

3. 컴포넌트 동적 로드 알고리즘

임베디드 시스템에 탑재한 미들웨어는 플러그인 기반으로 동적으로 컴포넌트를 로드하고, 로드한 컴포넌트들을 컴포넌트 관리자에게 등록하여 공유하여 사용 또는 재배치 하여 사용할 수 있도록 한다.

그림 22은 컴포넌트를 동적으로 로드하여 임베디드 미들웨어를 구성하는 순서도로 표현 하였다. 임베디드 시스템에 저장되어진 컴포넌트를 검색하여 목록을 만들고 만들어진 목록 중 인터페이스가 구현 되어진 컴포넌트를 확인하여 구현되어진 컴포넌트는 목록으로 저장을 하고 구현이 안 되어진 컴포넌트는 사용 목록으로 저장을 안 한다. 모든 컴포넌트를 조사한 후 프로파일에 저장 되어진 미들웨어 모듈 구성 정보가 있는지를 확인 후 미들웨어를 구성을 해야 할 정보가 없다면 미들웨어를 구성을 종료 처리 하고 있다면 이전에 인터페이스가 구현되어진 컴포넌트들 중 이미 로드 되어진 모듈인지 확인을 한다. 이미 로드 되어진 컴포넌트 이면 다음 컴포넌트 목록으로 확인 하고 해당 컴포넌트가 로드가 안

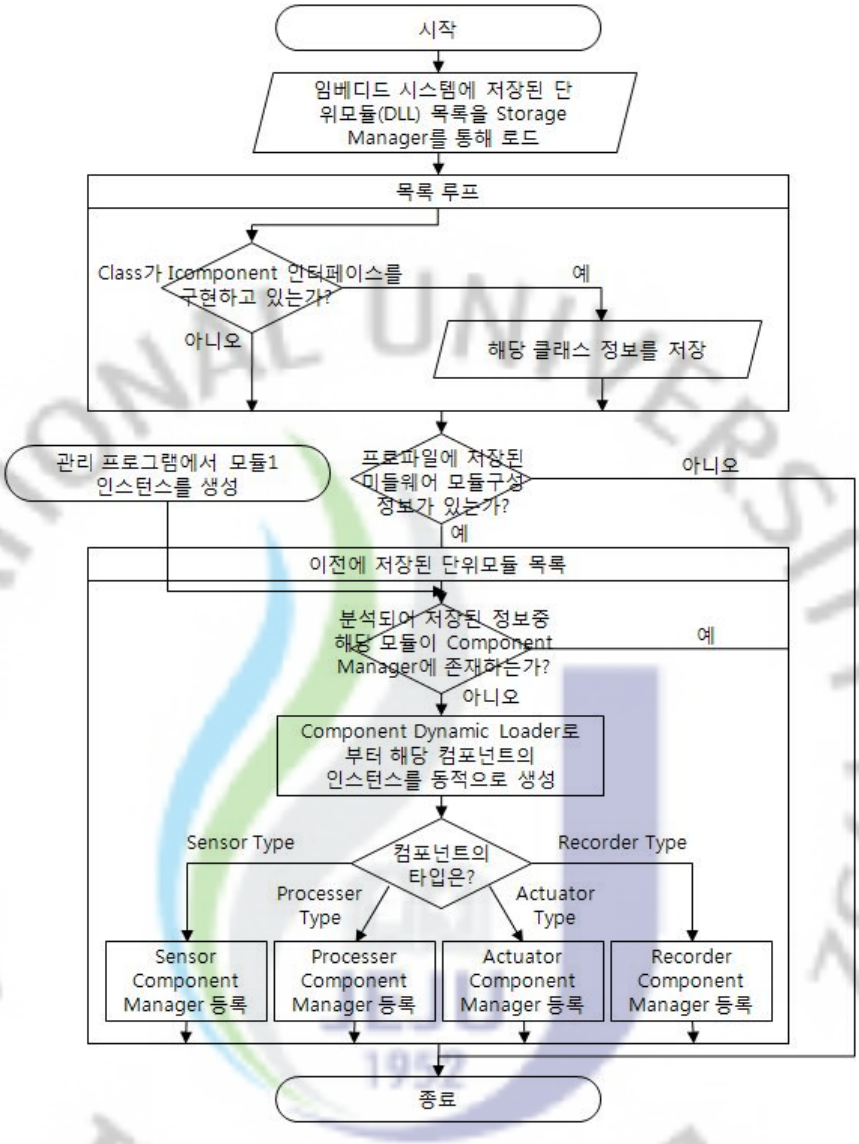

그림 22. 컴포넌트 동적 로드 순서도

되어진 컴포넌트 이면 컴포넌트 동적 로더로 부터 해당 컴포넌트를 동적 로드하고 로드 되어진 컴포넌트은 타입에 따라 센서 컴포넌트 이면 센서 컴포넌트 관리자에 등록하게 되고 프로세스 컴포넌트 이면 프로세스 컴포넌트 관리자에 등록, 구동기 컴포넌트 이면 구동기 컴포넌트 관리자에 등록, 레코딩 컴포넌트 이면 레코딩 컴포넌트 관리자에 등록하게 된다. 이 등록 과정은 프로파일에 저장되어진 컴포넌트 정보 만큼 반복하여 컴포넌트들을 동적 로드를 하게 된다.

```
#include

void 컴포넌트 동적 로드()
{
  for(저장되어진 컴포넌트 목록)
  {
        if(컴포넌트에 인터페이스 구현 되어 있는가?)
        {
          컴포넌트 정보 목록에 저장
    }
  }

  if(컴포넌트 로드하여 처리할 내용이 있는가?)
  {
    for(인터페이스가 구현되어진 컴포넌트 목록)
    {
      if(설정 정보에 로드되어야할 컴포넌트 인가?)
      {
        컴포넌트 Dynamic Loader를 통해 컴포넌트를 로드
        if(센서 타입 컴포넌트 인가?)
        {
          센서 타입 컴포넌트를 초기화
          컴포넌트 관리자에 등록
          공동 인터페이스를 통해 컴포넌트 연결 설정
        }else if(프로세스 타입 컴포넌트 인가?)
        {
          프로세스 타입 컴포넌트를 초기화
          컴포넌트 관리자에 등록
          공동 인터페이스를 통해 컴포넌트 연결 설정
        }else if(구동기 타입 컴포넌트 인가?)
        {
          구동기 타입 컴포넌트 초기화
          컴포넌트 관리자에 등록
          공동 인터페이스를 통해 컴포넌트 연결 설정
        }else
        {
          녹화 타입 컴포넌트 초기화
          컴포넌트 관리자에 등록
          공동 인터페이스를 통해 컴포넌트 연결 설정
        }
      }
    }
  }
}
```

표 4. 컴포넌트 동적 로드 알고리즘 Psuedo Code

표 4는 컴포넌트별로 연결설정을 하기 위한 준비과정으로서 컴포넌트의 인터페이스를 구성하고 구성되어진 정보를 프로파일로 저장을 한 후 프로파일 정보를 가지고 컴포넌트를 로드하여 연결설정하기까지의 동적 로드 알고리즘을 제시

한 것이다. 임베디드 시스템에서 저장 되어진 단위 컴포넌트 목록을 Storage Manager를 통해 목록을 검색하고 미리 설정된 메타정보를 통해 컴포넌트를 로드하게 된다. 로드된 컴포넌트는 공통 인터페이스인 IComponent 인터페이스를 구현하고 있는지를 확인 후 구현되어 있다면 해당 클래스 정보를 저장하고, 미구현 되어 있다면 다음 컴포넌트를 로드하여 다시 IComponet 인터페이스 구현 여부를 확인 한다. 이 작업은 로드할 컴포넌트 목록에 따라 반복적으로 수행하게 되며 모든 컴포넌트 목록의 로드 작업이 끝나면 이전에 저장된 미들웨어 컴포넌트 구성정보가 있는지를 확인한다.

구성정보가 있으면 분석되어 저장된 정보 중 해당 컴포넌트가 컴포넌트 Manager에 등록이 되어 있는지 확인하고 등록이 안 되어 있다면 컴포넌트 Dynamic Loader로부터 해당 컴포넌트의 인스턴스를 동적으로 생성하여 컴포넌트 타입별로 Sensor Type이라면 Sensor 컴포넌트 Manager에 등록 하고, Processer Type이면 Processer 컴포넌트 Manager에 등록, Actuator Type이면 Actuator 컴포넌트 Manager에 등록, Recorder Type이면 Recorder 컴포넌트 Manager 에 등록하여 컴포넌트를 동적로드 하게 된다. 만일 이전에 저장되어진 컴포넌트의 메타정보가 없다면 컴포넌트 로드를 종료하게 되며 컴포넌트 Manager에 이미 등록 된 상태라면 컴포넌트 로드를 종료하게 된다.

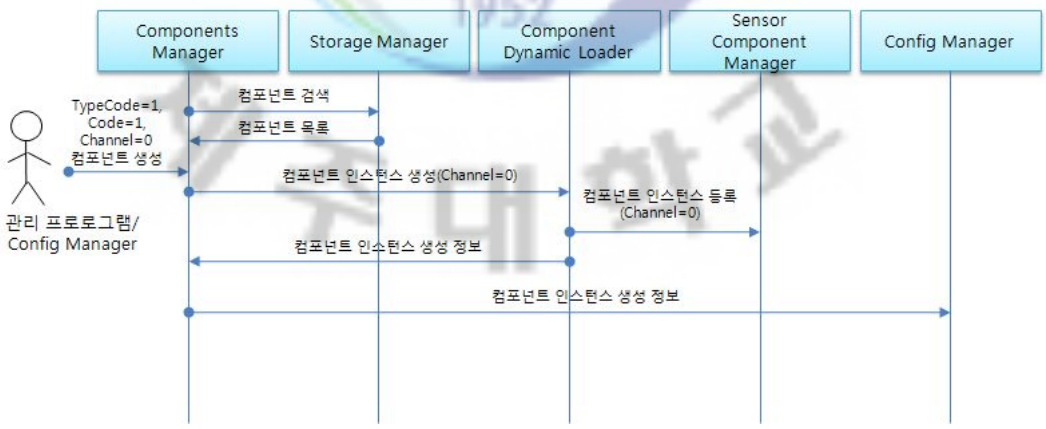

그림 23. 센서 타입 컴포넌트 동적 로드 시퀀스 다이어그램

그림 23은 센서 타입의 컴포넌트를 동적 로드하여 컴포넌트 인스턴스를 컴포넌트 관리자에게 등록 절차를 나타내고 있는 시퀀스 다이어그램이다. 관리프로그

램은 TypeCode=1, Code=1, Channel=0 인 값으로 Components Manager로부터 컴포넌트 생성을 요청하면 Components Manager는 Storage Manager로 컴포넌트 검색을 요청하게 되고, 요청한 결과로 컴포넌트 목록을 반환을 하게 된다. 컴포넌트 목록이 없으면 Component Dynamic Loader로 컴포넌트 인스턴스 생성을 요청하게 되고, Component Dynamic Loader는 Sensor Component Manager로 컴포넌트 인스턴스을 등록한 후 컴포넌트 인스턴스 생성 정보를 Component Manager에게 응답한다. 컴포넌트 인스턴스 생성 정보는 Config Manager로 에게 보내져 설정 정보를 저장하는 과정을 거치게 된다.

IV. 구현 및 고찰

1. 구현환경

맞춤형 임베디드 시스템 미들웨어 생성 시스템은 임베디드 시스템에 탑재한 미들웨어와 임베디드 시스템을 관리하는 관리 프로그램으로 구성되어지며 임베디드 시스템은 Windows Embedded CE 6.0 운영체제 환경에서 개발하였으며 개발환경 및 제반사항은 표 5에 나타나 있다.

구분	임베디드 시스템	관리 프로그램
개발 OS	Windows Embedded CE 6.0 플랫폼 빌더	Windows XP 이상 운영체제
개발 환경	.Net Compact FrameWork	.Net FrameWork 3.5 이상
개발 언어	Visual Studio Pro 2008	Visual Studio Pro 2008
통신 방법	Wifi Client	Wifi Ad-hoc AP

표 5. 프로그램 구현 환경

관리 프로그램은 임베디드 시스템의 환경을 설정하고 제어 할 수 있도록 설계된 .NET 기반의 소프트웨어이다. 임베디드 시스템은 주변에 Ad-hoc AP를 검색하여 자동으로 연결되도록 설정되어 있어 노트북 또는 UMPC 등의 무선네트웍을 통해 임베디드 시스템을 관리할 수 있다. 관리 프로그램이 정상적으로 동작하기 위해서는 PC에 무선AP로 동작 가능한 무선랜(Wi-Fi)이 탑재되어야 하며, Windows XP 이상의 운영체제가 필요하다. 또한 Microsoft .NET FrameWork 3.5 이상의 소프트웨어가 설치되어 있어야 동작이 가능하며 Windows Vista 혹은 Windows 7 버전의 운영체제는 FrameWork 3.5가 기본적으로 설치되어 있으나 Windows XP에서는 추가적으로 설치가 필요하다.

2. 관리 프로그램 인터페이스

관리 프로그램은 하나 이상의 임베디드 시스템을 관리 할 수 있도록 하는 기

능의 프로그램이며 임베디드 시스템의 구성을 변경하거나 시스템 상태를 조회할 수 있으며, 간단한 제어 및 시간 동기화와 최신 소프트웨어로 업그레이드 할 수 있는 기능을 제공 하게 된다. 그림 24은 관리프로그램에 대한 UI를 나타내고 있으며, 관리 프로그램은 접속상태 및 접속목록 표시, 로그인 등록창, 클라이언트 버전 정보, 시스템 로그 및 시스템 상태 정보 창으로 구분되어져 있으며 사용자 인터페이스 항목별 설명은 표 6에 나타나 있다.

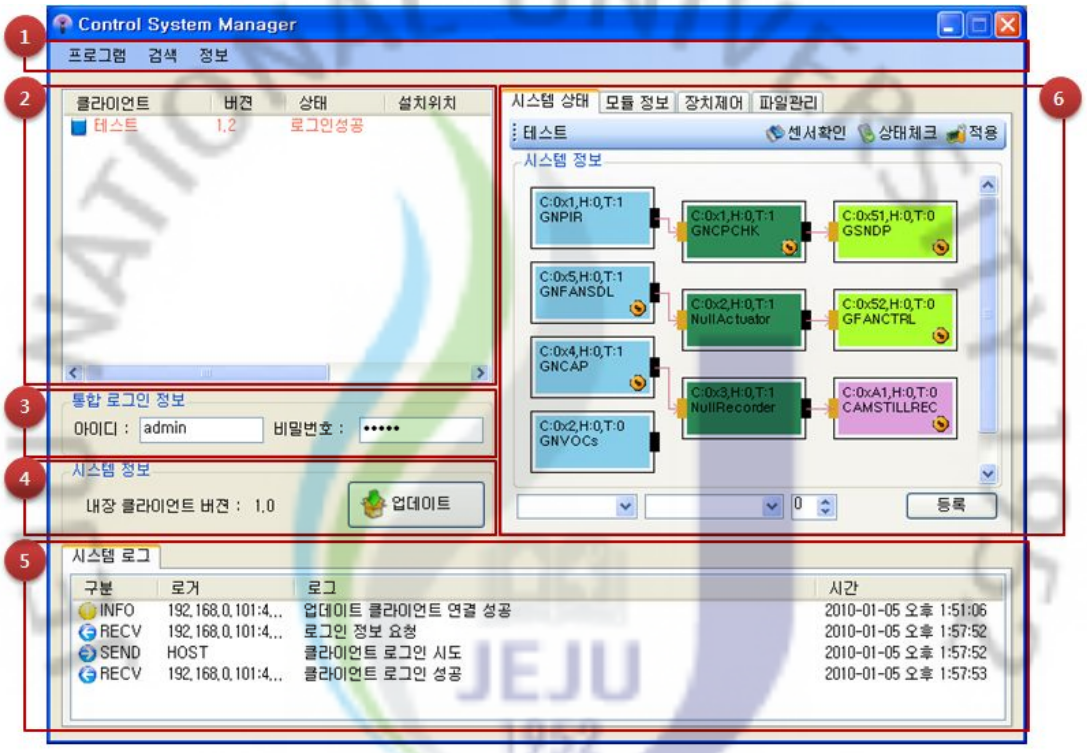

그림 24. 관리 프로그램 GUI 구성

1	프로그램 닫기, 검색, 프로그램 정보를 조회할 수 있는 메뉴
2	접속된 임베디드 시스템 정보 목록을 출력
3	임베디스 시스템에 접속하기 위한 로그인 정보를 입력
4	관리 프로그램 버전 정보를 출력하고 업데이트
5	임베디드 시스템 로그 정보를 출력
6	접속된 임베디드 시스템 세부 상세 내용을 출력

표 6. 관리 프로그램 GUI 구성 설명

표 6의 ⑥번 항목의 시스템 상태정보는 현재 접속된 임베디드 시스템을 사용

자가 드래그를 통하여 새로이 구성하거나 편집/삭제 할 수 있으며 시스템의 상태를 조회 할 수 있는 기능을 포함하며, 관리 프로그램를 통해 접속된 임베디드 시스템의 모듈 정보를 출력할 수 있다.

그림 25. 시스템 상태 GUI 구성

1	접속된 시스템 정보와 센서 및 상태를 조회, 컴포넌트 구성 정보 적용
2	현 시스템 구성 정보를 출력
3	시스템을 구성할 모듈을 등록

표 7. 시스템 상태 GUI 구성 설명

관리 프로그램은 시스템 내 설치된 Sensor, Processer, Actuator, Recoder 모듈을 적절하게 조합하여 시스템을 구성할 수 있다. 시스템정보 창을 통해 모듈을 등록하고 삭제 및 편집할 수 있으며 모듈의 속성을 설정할 수 있다. 모듈은 시스템정보 창을 통해 구분을 위해 색상이 들어 있는 사각형으로 표현하였으며 모듈의 종류에 따라 색상 및 형태를 조금씩 다르게 표현하였다.

Sensor 모듈		Actuator 모듈	
Processer 모듈		Recoder 모듈	

표 8. 모듈별 구분 색상 및 아이콘

- 상자의 왼쪽 주황색 상자는 입력이 있음을 의미하고 오른쪽 검정색 상자는 출력이 있음을 의미
- 오른쪽 하단의 은 환경설정이 가능한 모듈임을 나타내고 모듈의 첫 번째 줄은 모듈의 구성 정보를 의미
- [C]는 모듈 코드번호, [H]는 모듈의 채널 번호 . [T]는 출력 개수

※ 연결하고자 하는 모듈의 입력 부분에 마우스커서를 올리고 클릭을 띄면 모듈이 화살표로 연결이 된다.

표 9. 모듈 세부 아이콘 설명

모듈정보는 모듈에 따라 시스템 구성과 기능 및 역할이 달라질 수 있다. 모듈은 크게 Sensor와 Processer, Actuator, Recoder로 구성된다. Sensor는 외부 입력 데이터를 수신하거나 데이터를 생성 하는 모듈의 집합이며. Processer는 Sensor에서 제공한 데이터를 가공하거나 제어하는 역할을 하는 모듈이고, Actuator는 Processer에서 처리한 데이터를 기반으로 외부 장치를 제어하는 역할을 하는 모듈이다. Recoder는 Processer에서 처리한 데이터를 기반으로 데이터를 저장하거나 외부 장치의 결과를 저장하는 역할을 하는 모듈이다. 모듈정보에 대한 UI는 그림 26의 좌측에 예가 나타나 있으며 장치제어에 관한 UI는 그림 26의 우측에 나타나 있다.

그림 26. 모듈정보와 장치제어 GUI 구성

장치 제어는 관리 프로그램을 통해 접속된 임베디드 시스템을 제어하는 기능으로 임베디드 시스템의 음성출력 볼륨조절 및 미리듣기 기능과 미들웨어를 시작하거나 정지 할 수 있으며, 임베디드 시스템을 재부팅 할 수 있는 기능이 포함

되어 있다. 또한 시스템에 부착된 구동기(환기펜 등)의 동작을 테스트 할 수 있도록 인터페이스를 구성하였다.

그림 27는 음성출력을 위한 음원 파일관리 UI를 나타낸 것으로 파일은 무선 또는 메모리를 통하여 업로드 하거나 시스템 내의 음원을 검색할 수 있는 기능을 포함하고 있다.

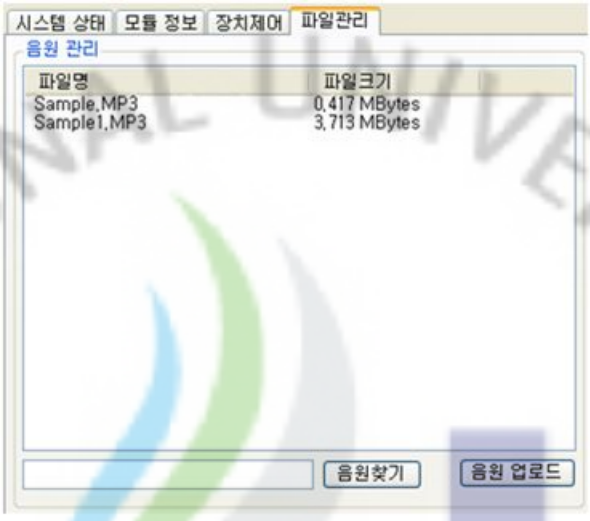

그림 27. 파일 관리 GUI 구성

그림 28는 영상정보에 대한 검색 및 인코딩에 관한 UI로 관리 프로그램을 통해 임베디드 시스템에 기록된 영상을 기록일자별 또는 기간별로 조회 하거나 검색된 영상을 하나의 동영상 파일로 만들 수 있는 인코딩 기능을 제공하여 동영상으로 모니터링을 할 수 있다.

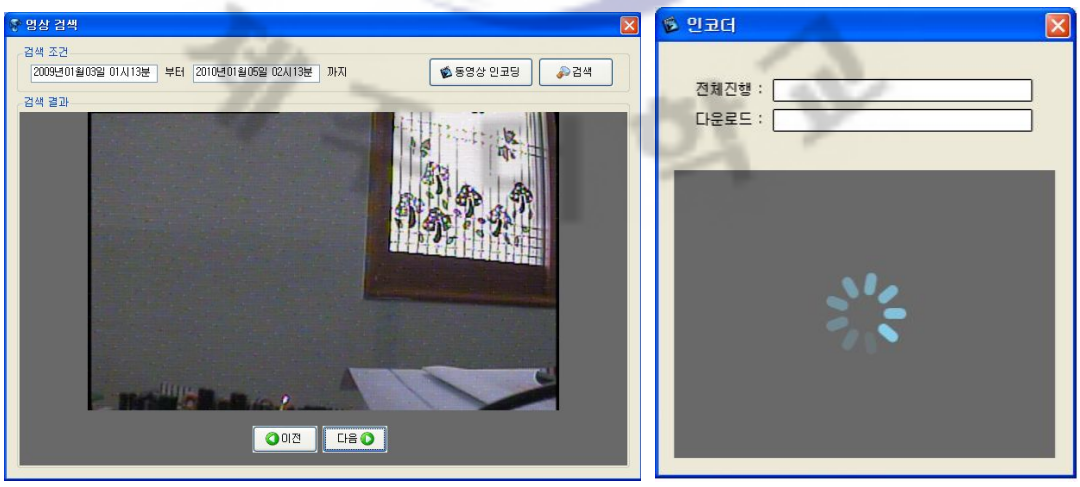

그림 28. 영상 검색 및 인코딩 GUI 구성

그림 29은 시스템 로그에 대한 UI로 환경설정, 센서 데이터 임계치 초과, 시스템 가동 및 중지 등 시스템 관리를 위해 기록되어진 로그 내용을 기간별로 조회할 수 있는 기능을 제공하여 과거 임베디드 시스템의 동작 내용을 분석할 수 있도록 도움을 줄 수 있다.

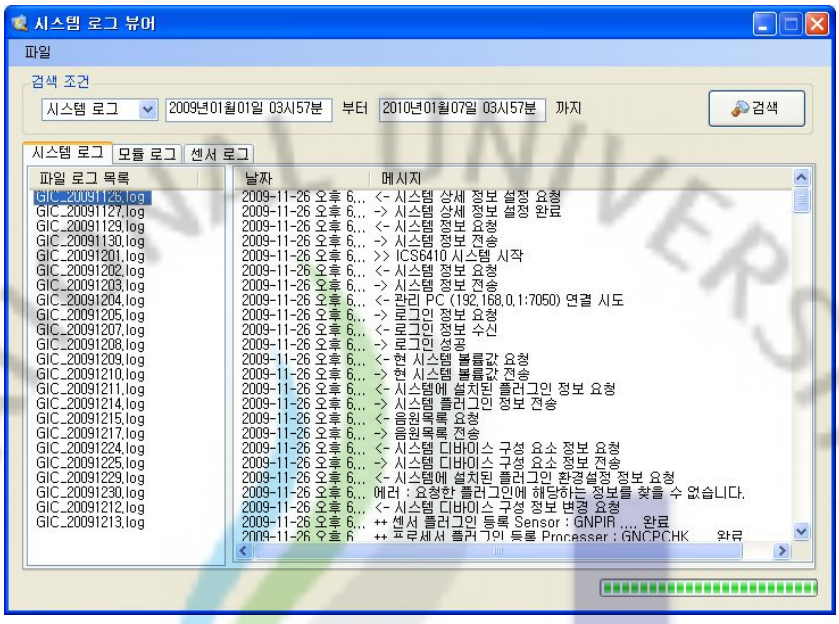

그림 29. 시스템 로그 GUI 구성

그림 30은 센서 데이터에 관한 로그로서 센서 정보 같은 경우 센서를 선택하고 기준치 값을 입력하여 센싱된 데이터를 통해 차트를 출력할 수 있어, 센싱된 데이터를 모니터링 하거나 분석을 할 수 있도록 화면을 구성 하였다.

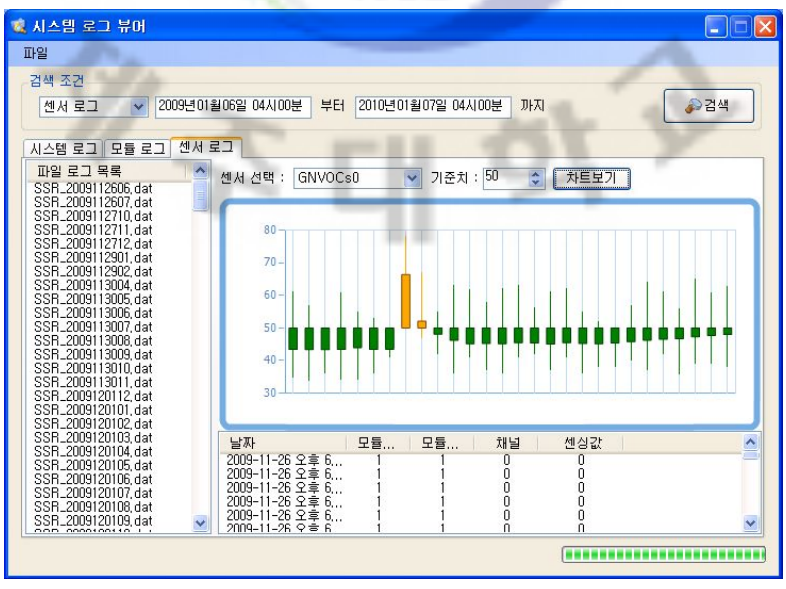

그림 30. 센서 로그 GUI 구성

그림 31은 센서 데이터 로그를 차트로 나타낸 UI이며 차트를 통해 일자별로 센서 데이터의 기준치 초과 여부 및 평균 센서값을 가시화 하였다.

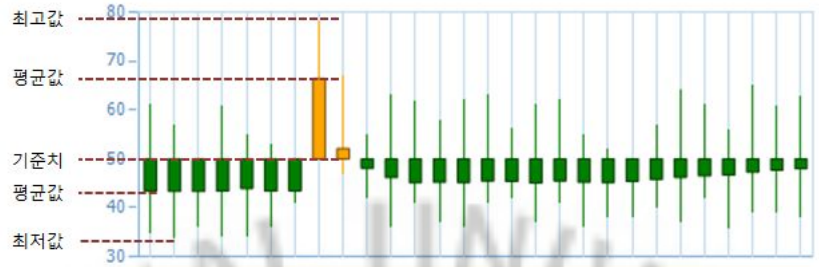

그림 31. 센서 로그 차트

3. 임베디드 시스템 미들웨어

본 논문의 핵심은 임베디드 미들웨어를 구현하기 위하여 3장에서 기술한 고려사항을 충족하기 위해 인터페이스를 설계하기 전에 우선 컴포넌트의 타입의 코드 정의와 컴포넌트의 구분 코드, 컴포넌트의 상태를 알려줄 수 있는 상태 정의가 필요하다. 컴포넌트의 코드정의에 관한 내용은 표 10, 표 11, 표 12에 나타난 내용으로 정의하였다..

0x1 : Sensor	0x2 : Actuator	0x3 : Recorder	0x4 : Processer

표 10. 컴포넌트의 타입 코드 정의

0x1: PIR 센서
0x2: Voc 센서
0x3: CO2 센서
0x4: Camera Capture(카메라를 캡처를 위해 프레임수 만큼 제어데이터 발생)
0x5: Fan Scheduler(스케쥴에 따라 제어데이터 발생)
0x51: 사운드 플레이어
0x52: 펜 컨트롤러
0xA1: 카메라 영상 캡처기(카메라 디바이스로부터 영상을 촬영)

표 11. 컴포넌트의 구분 코드 정의

0: 생성(인스턴스 로드) 1: 초기화 2: 동작중 3: 중지 4: 오류

표 12. 컴포넌트의 상태 코드 정의

인터페이스 구현은 컴포넌트 공통 인터페이스 센서 타입 컴포넌트 인터페이스, 처리 타입 컴포넌트 인터페이스, 출력(Actuator)타입 컴포넌트 인터페이스 , 출력(Recorder) 타입 컴포넌트 인터페이스 정의 하였다.

```
string Version;
string ComponentTypeName
int ComponentTypeCode
string ComponentName
int ComponentCode
string Made;
string Name;
bool Configable
int Channel;
int State;
string StateMessage
bool SetXmlConfig(string);
string GetXmlConfig();
event StateChange StateChanged
```

표 13. 컴포넌트 공통 인터페이스

```
bool Init(IProcessorComponent[] components);
void Start();
void Stop();
void Close();
void IsPlaying();
```

표 14. 센서 타입 컴포넌트 인터페이스

```
bool Init(IActuatorComponent[] components);
bool Init(IRecorderComponent[] components);
bool Init(IActuatorComponent[] components,
        IRecorderComponent[] components);
void Process(int componentTypeCode, int channel,
        byte[] data);
```

표 15. 처리 타입 컴포넌트 인터페이스

```
bool Init();
void Start(int code, int channel);
void Stop(int code, int channel);
void Close();
bool IsPlaying();
```

표 16. 출력(Actuator) 타입 컴포넌트 인터페이스

```
bool Init();
void Start(int code, int channel);
void Stop(int code, int channel);
void Close();
bool IsPlaying();
void ResourceAdjust();
int GetRecordCount();
string[] GetRecordFiles();
string[] GetRecordFiles(DateTime start,DateTime end);
```

표 17. 출력(Recorder) 타입 컴포넌트 인터페이스

정의한 모든 컴포넌트 인터페이스는 반드시 구현해야 할 필요는 없으나 구현되지 않은 인터페이스 메소드는 NotSupportedException 처리를 하여 미구현 되더라도 컴포넌트는 인터페이스를 상속받아 사용해야 한다. 이는 미들웨어에서 컴포넌트를 동적으로 로드하여 컴포넌트 관리자에게 등록하고 각 타입에 맞는 컴포넌트의 연결을 보장하기 위함이다.

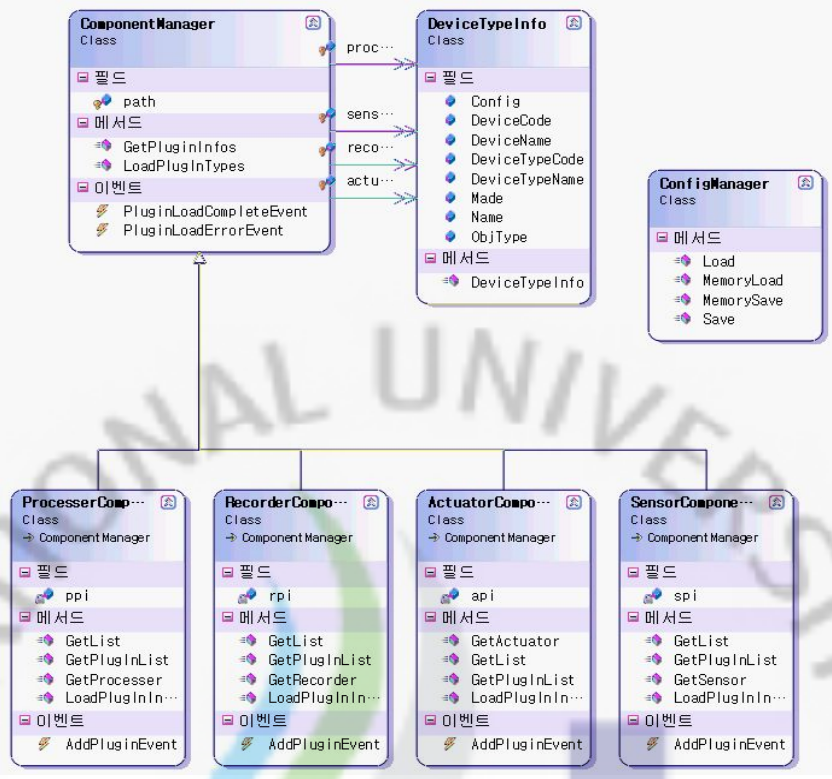

그림 32. 컴포넌트 동적 로드 모듈 클래스 다이어 그램

그림 32와 같이 ComponentManager 클래스는 컴포넌트를 동적으로 로드하기 위해 각 컴포넌트들의 Manager 클래스를 사용하여 동적으로 컴포넌트를 로드하고 로드 되어진 컴포넌트 관리와 컴포넌트는 연결 설정정보에 따라 컴포넌트들을 연결 설정을 한다.

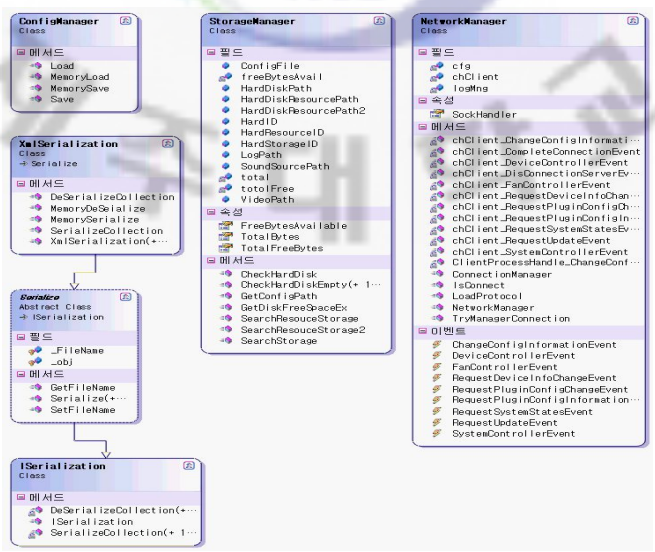

그림 33. 센서/구동/프로세스/영상 프로파일 생성 모듈 클래스 다이어그램

그림 33에서처럼 Config Manager 클래스는 Storage Manager 클래스를 통해 프로파일이 저장되어진 경로 프로파일 정보를 읽어 오게 되며 읽어온 정보를 관리프로그램에게 전송하고 컴포넌트의 동적 로드 정보(프로파일)를 저장 하거나 관리를 담당하게 된다.

```
  <SystemName>Management System</SystemName>
  <Location>Embedded System</Location>
  <Version />
  <Storage />
  <SystemConfig>
    <Devices>
      <DeviceContent>
        <DeviceTypeCode>1</DeviceTypeCode>
        <DeviceCode>1</DeviceCode>
        <DeviceChannel>0</DeviceChannel>
        <Targets>
          <TargetDevice>
            <TargetTypeCode>3</TargetTypeCode>
            <TargetCode>1</TargetCode>
            <TargetChannel>0</TargetChannel>
          </TargetDevice>
        </Targets>
      </DeviceContent>
      <DeviceContent>
        <DeviceTypeCode>3</DeviceTypeCode>
        <DeviceCode>1</DeviceCode>
        <DeviceChannel>0</DeviceChannel>
        <Targets>
          <TargetDevice>
            <TargetTypeCode>2</TargetTypeCode>
            <TargetCode>81</TargetCode>
            <TargetChannel>0</TargetChannel>
          </TargetDevice>
        </Targets>
      <DeviceContent>
        <DeviceTypeCode>2</DeviceTypeCode>
        <DeviceCode>81</DeviceCode>
        <DeviceChannel>0</DeviceChannel>
        <Targets />
      </DeviceContent>
    </Devices>
  </SystemConfig>
  <Log>true</Log>
  <SystemCheckTime>
    <Hours>9</Hours>
    <Minutes>0</Minutes>
    <Seconds>0</Seconds>
  </SystemCheckTime>
</ChConfig>
```

표 18. 프로파일 Xml 저장 결과

표 18은 센서/구동/프로세스/영상 프로파일을 저장한 결과의 Xml 파일에 일부분이다. 컴포넌트의 클래스 정보와 설정 정보를 Xml로 저장하여 보편화 시켰고 이 정보는 차후 다양한 플렛폼에 적용하더라도 쉽고 빠르게 적용할 수 있다.

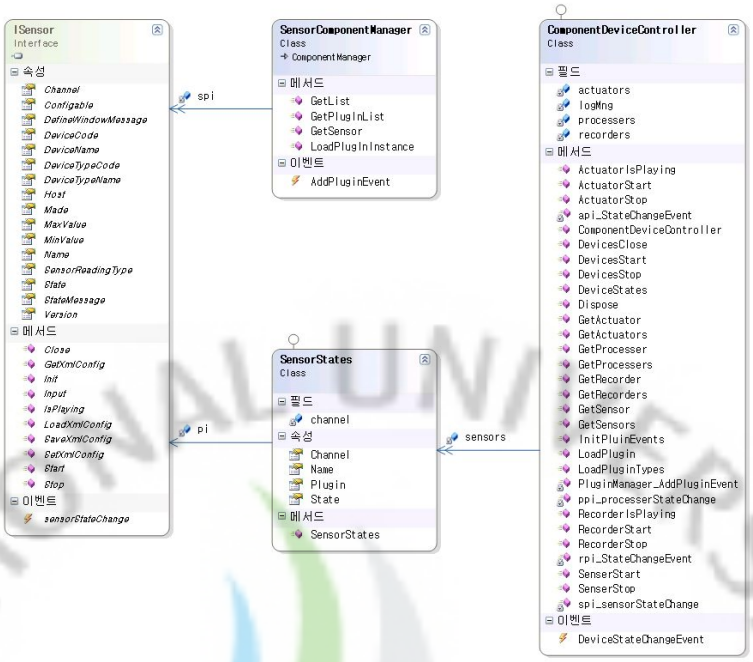

그림 34. 센서 컴포넌트 클래스 다이어그램

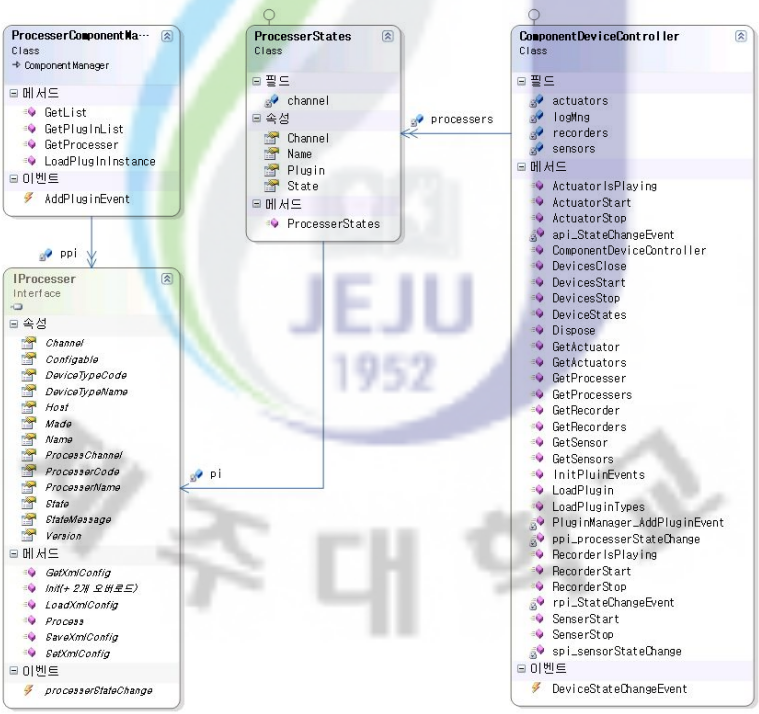

그림 35. 프로세스 컴포넌트 클래스 다이어그램

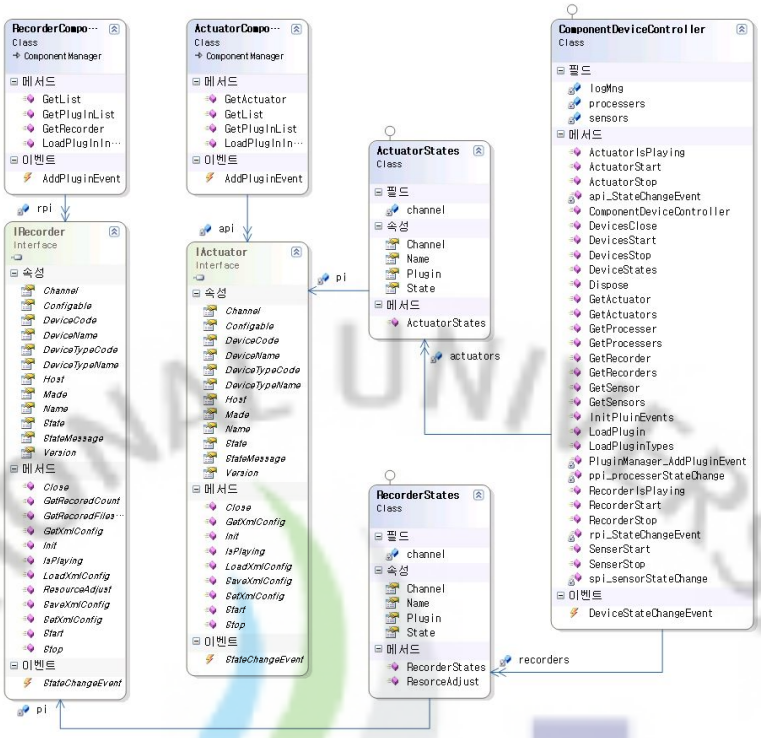

그림 36. 구동체 제어 / 영상 장치 제어 컴포넌트 클래스 다이어그램

각 컴포넌트를 로드하기 위한 Manager 클래스와 각 속성에 맞는 컴포넌트 인터페이스를 컴포넌트의 동작 상태를 알려주는 State 클래스와 Device를 컨트롤하거나 정보를 가져 올 수 있는 ComponentDeviceController 클래스로 구성한 센서 컴포넌트, 프로세스 컴포넌트, 구동/제어 컴포넌트 클래스 다이어그램이다.

```
Assembly ass = null;
ass =  Assembly.LoadFrom(ComponentFiles[i]);
if (ass != null)
{
    Type ObjType = ass.GetType(args + ".SensorComponent");
    if (ObjType != null)
    {
        try
        {
            ISensorComponent ispi = Activator.CreateInstance(ObjType) as
                                                        SensorComponent;
            sensorTypes.Add(ispi.DeviceCode, new DeviceTypeInfo(ObjType,
            ispi.Made, ispi.Name, ispi.DeviceTypeCode, ispi.DeviceTypeName
            , ispi.DeviceCode, ispi.DeviceName, ispi.Configable));
            ispi = null;
        }
        catch
        {
            if (ComponentLoadErrorEvent != null)
                ComponentLoadErrorEvent(args + ".SensorComponent Instance
                                                                Error");
        }
    }
    else
    {
        ObjType = ass.GetType(args + ".ProcesserComponent");
```

```
if (ObjType != null)
{
    try
    {
        IProcesserComponent ippi = Activator.CreateInstance(ObjType)
                                        as IProcesserComponent;
        processerTypes.Add(ippi.ProcesserCode, new DeviceTypeInfo
        (ObjType, ippi.Made, ippi.Name, ippi.DeviceTypeCode,
        ippi.DeviceTypeName, ippi.ProcesserCode, ippi.ProcesserName,
        ippi.Configable));
        ippi = null;
    }
    catch
    {
     if (ComponentLoadErrorEvent != null)
        ComponentLoadErrorEvent(args+ ".ProcesserComponent
                                                Instance Error");
    }
}
                                :
```

표 19. 컴포넌트 동적 로드 소스 코드

표 19은 3장에서 기술한 동적 로드 알고리즘을 각 컴포넌트를 동적 로드하여 정의 되어진 인터페이스 정보를 가지고 동적으로 컴포넌트를 연결 설정하는 소스 코드 이다.

그림 37. 임베디드 시스템에 탑재한 미들웨어 구동화면

그림 37는 실제 임베디드 시스템 에 탑재한 미들웨어 화면이다. 임베디드 시스템에 탑재한 미들웨어는 임베디드 시스템이 부팅과 동시에 로딩 되도록 구성하였으며, 화면에 동작되어지는 로그 정보들을 출력하여 실제 가동되는 정보를 확인할 수 있도록 하였고 이러한 정보들을 File로 저장하여 임베디드 시스템에 문제가 발생하거나 오동작 여부를 UI를 통해 확인할 수 있도록 구성하였다.

4. 고 찰

센서 데이터는 센서 타입과 제조 회사에 따라 전송되어지는 데이터 크기와 데이터 값의 범위가 틀려지는 문제가 발생하였는데 이 문제를 해결하기 위해 센서 데이터를 일반화 하는 작업이 필요하게 되었다.

이에 본 논문에서는 센서 데이터 구조를 정의하기 위해 필요한 정보들을 정리하였는데 어떤 종류의 센서 인지 센서의 타입을 구분할 수 있어야 하면 이 장치가 센서 인지 다른 장치인지 구분할 수 있는 정보와 같은 타입의 센서를 여러개 설치 시 각 센서별로의 데이터를 구분할 수 있는 정보 그리고 실제 센싱된 정보를 저장할 수 있는 구조가 필요하게 되었다.

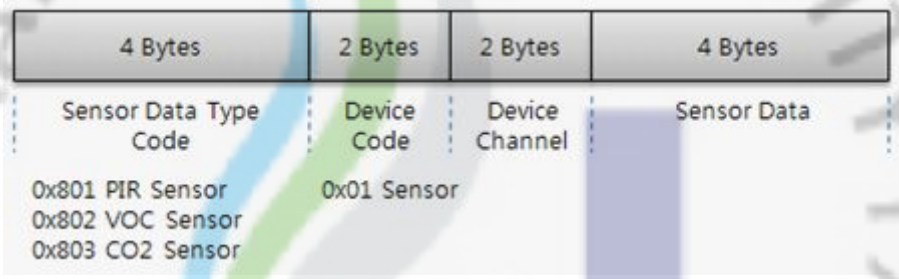

그림 38. 센서데이터 구조도

센서 데이터 구조를 총 12Bytes로 구성하고 센서의 Type을 구분할 수 있는 Sensor Data Type Code의 4Bytes와 연결되어진 디바이스 장치를 구분할 수 있는 Device Code 2Bytes와 같은 타입의 디바이스를 구분할 수 있는 Device Channel 2Bytes로 구성하고 센싱된 데이터 저장하기 위한 Sensor Data 4Bytes로 구성하여 그림 38과 같이 센서데이터 구조를 정의 하였다.

임베디드 시스템에 장착 되어진 센서 Device들의 데이터를 처리 및 기록하기 위해서는 임베디드 시스템 자원이 한정적인 문제가 발생하게 되었다. 이러한 문제는 데이터 처리상에 문제라기보다 데이터의 기록부분에서 문제가 발생되었다. 이 문제는 센싱된 데이터를 분석 후 기록하기 위한 절차를 수행하도록 함으로 인해 이 절차를 수행할 때 임베디드 시스템은 OverFlow가 발생하는 문제점이 발견 되었다.

이러한 문제의 발생 원인은 센싱된 데이터를 저장하기 위해 임베디드 시스템에 장착 되어진 저장 매체인 HDD, CF 메모리, SD 메모리 등에 저장되는 속도보다 더 많은 데이터가 발생하기 때문이였다. 이 문제는 파일의 Open과 Close의 동작으로 오버헤드가 발생하였는데 이를 해결하기 위해서 시스템 메모리에 Queue를 생성하여 기록할 데이터를 저장한 후 일정한 시간단위로 파일을 Open하고 Queue에 쌓여 있는 센싱 데이터를 저장 후 파일 Close를 하여 오버헤드를 줄이는 방법으로 OverFlow 현상을 해결 할 수 있었다.

그림 39. 실행시 컴포넌트 로드 지연 시간 차트

그림 39차트는 컴포넌트 15개를 프로그램 실행 시 컴포넌트가 로드 되어지는 지연 시간의 차트이다. 총 실험 횟수는 총 10회이며 실행시 컴포넌트 로드의 평균 결과는 282 Millisecond 평균 경과 시간을 보이고 있다.

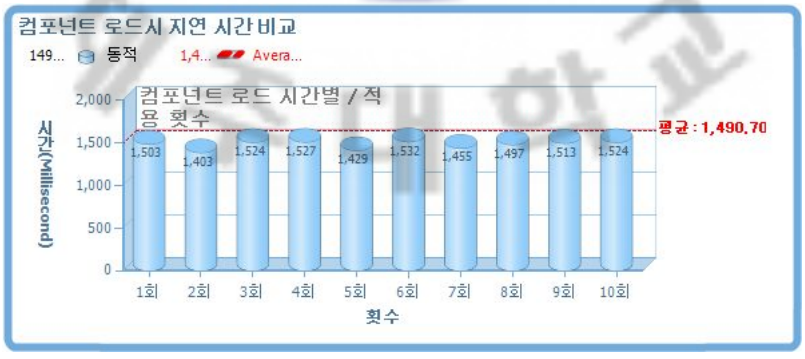

그림 40. 동적 컴포넌트 로드 지연 시간 차트

그림 40 차트는 컴포넌트 15개를 프로그램 로드 시 동적으로 컴포넌트를 로드되어지는 경우의 차트이며 동적 로드시 1490 Millisecond 평균 경과 시간을 보이

고 있으며, 실행시 컴포넌트를 로드 하는 경우와 약 1208 Millisecond 편차를 보이고 있다. 컴포넌트 동적 로드시 시간이 더 걸리는 이유는 저장장치로부터 동적 컴포넌트 저장경로를 검색하는 시간과 컴포넌트 목록을 검색하는 시간, 모듈을 로드 하는 시간, 컴포넌트의 인터페이스가 구현이 되어 있는지 확인하는 시간, 인터페이스가 구현이 안 되어 있을시 Exception이 발생하여 오류 처리하는 시간이 추가 되어 좀 더 높은 경과 시간이 보이고 있음을 알 수 있었다.

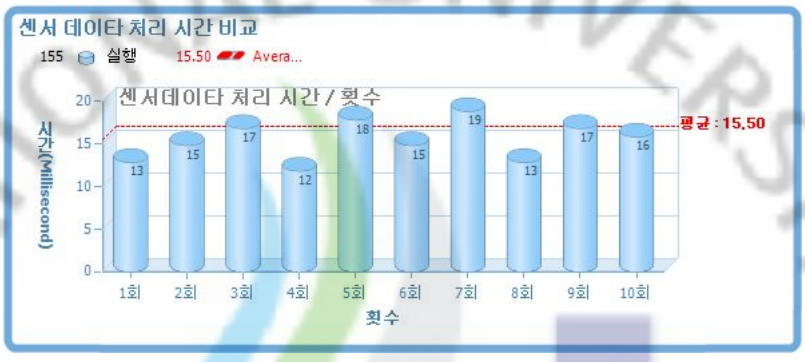

그림 41. 실행시 로드 센서데이타 처리 시간 차트

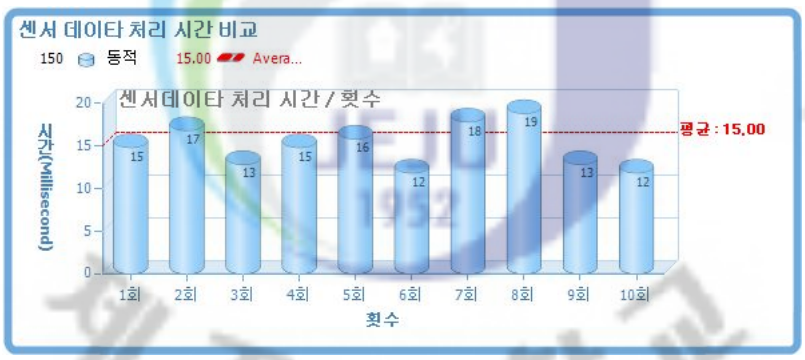

그림 42. 동적 로드시 센서데이타 처리 시간 차트

그러나 실제 로드 되는 시간에 비해 그림 41, 그림 42의 센서 데이터 처리 시간 차트를 보면, 실행 시 로드 되는 경우 15.5 Millisecond 평균 처리 시간과 동적 로드 되는 경우 15 Millisecond의 평균 처리 시간을 보이고 있다. 이는 처음 센서 데이터 처리시 처리시간은 실행시 로드 되어진 컴포넌트와 동적으로 로드 되어진 컴포넌트의 처리 사간이 거의 차이가 없는 결과를 얻게 되었다.

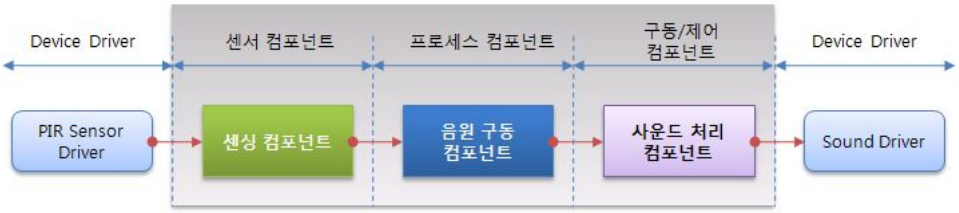

그림 43. 센서가 하나일 경우 미들웨어 서비스 구성도

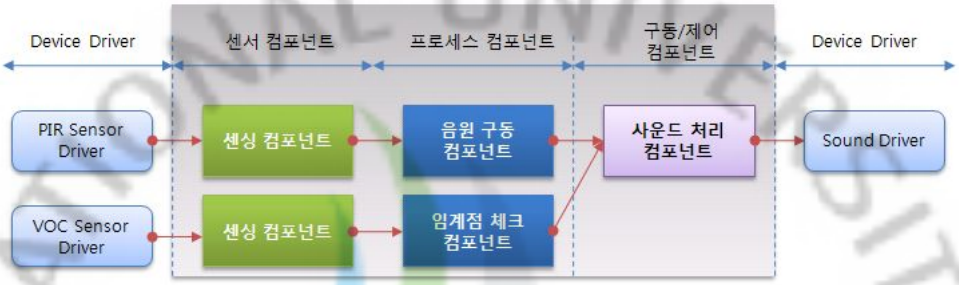

그림 44. 센서 추가시 변경 미들웨어 서비스 구성도

본 논문에서 제시한 임베디드 시스템 미들웨어 설계 및 실제 구현결과를 통해 임베디드 시스템의 특성상 새로운 디바이스를 지원할 수 있도록 설계하기 힘든 부분에 대한 문제점을 어느 정도 개선할 수 있음을 확인할 수 있었다. 즉, 임베디드 소프트웨어의 자원 재사용성을 높이고 임베디드 시스템 구축과 유지관리 및 개발에 투자 하는 비용 및 시간 절감에 도움을 줄 수 있을 것으로 기대한다.

V. 결 론

본 논문은 임베디드 시스템의 재사용성을 개선하기 위한 방안으로 디바이스 종류 및 디바이스 처리 로직이 변경 되더라도 쉽게 적응할 수 있는 미들웨어와 관리자가 임베디드 시스템의 상태 모니터와 미들웨어의 구성을 쉽게 처리 할 수 있도록 관리 프로그램을 구현 하였다.

이를 위해 먼저 임베디드 시스템의 가변성 및 소프트웨어 위한 컴포넌트 구축 방법을 응용하였고, 컴포넌트별 일반화를 위해 타입별 컴포넌트 인터페이스를 정의 하였으며 컴포넌트를 공유하고 재배치하여 하나의 임베디드 시스템에 구성할 수 있도록 하였다. 즉, 컴포넌트 동적 로드 알고리즘을 적용하여 임베디드 시스템의 재개발 없이 다양한 곳에 적용할 수 있도록 시스템 활용과 컴포넌트들을 공유 할 수 있도록 본 논문에서는 구현하고 고찰 하였다.

본 논문에서 구현한 미들웨어를 통해 임베디드 시스템의 특성상 특정 목적에만 사용할 수 있던 일반적인 시스템의 확장을 위해 재구축 하거나, 시스템에 맞는 소프트웨어를 별도로 개발해야 하는 문제로 인한 시스템 개발비 증가와 시스템의 유지 보수 비용 증가에 대한 문제점 개선할 수 있음을 확인하였다. 이로 인해 임베디드 시스템 개발비용 절감 효과와 임베디드 시스템의 활용 범위를 좀 더 다양화 하고, 응용 분양에 좀 더 효과적으로 적용할 수 있을 것으로 기대한다.

본 논문에서 구현한 맞춤형 임베디드 미들웨어 생성 시스템은 입력 컴포넌트, 처리 컴포넌트, 출력 컴포넌트들을 조합하여 하나의 임베디드 시스템 미들웨어를 구성하는 관리프로그램도 구현하였는데, 사용자의 편의성을 위해 직관적인 UI로 일반 사용자들의 접근하기 힘들어하는 임베디드 시스템을 좀 더 쉽고 빠르게 시스템에 적용할 수 있어 좀 더 많은 분야로의 응용을 기대한다.

본 논문에서 구현한 임베디드 미들웨어는 내장된 미들웨어 내에 특정기능 추가 요구에 따른 자원 재사용을 위한 적절한 컴포넌트가 없을시 추가 기능에 따른 별도의 컴포넌트를 작성해야 하며, 이는 새로이 추가·변경을 요하는 디바이스의 컴포넌트가 없을 때에도 해당 디바이스에 대한 추가 컴포넌트 작성이 필요한

문제가 나타났다. 이러한 문제점을 해결하기 위하여 임베디드 시스템 적용을 위한 분야 및 환경별 컴포넌트 카테고리의 일반화 적용과 임베디드 시스템 환경의 표준화에 따른 추가적인 연구 및 컴포넌트 재사용성에 대한 심도 있는 연구를 통하여 문제를 해결할 수 있을 것으로 보이며, 이를 통하여 좀 더 많은 분야에 관리의 용이성을 꾀하는 저비용의 임베디드 시스템을 다양하게 활용할 수 있으리라 기대한다.

참고문헌

[1] 김철진,이숙희,조은숙 『플러그인 기법을 이용한 임베디드 시스템의 재사용 향상 기법』, 한국시뮬레이션학회 학술대회 논문지 pp81-94 , 2009.12

[2] 김철진,정승재,김수동, 『컴포넌트 행위 커스터마이제이션 기법』, 한국정보과학회논문지 제 3,4호 ,2003.04

[3] 조은숙,『CBD상에서 컴포넌트 설계를 위한 통합 컴포넌트 메타 모델에 관한 연구』,한국시뮬레이션학회 학술대회 논문지 pp107-113 , 2003.6

[4] 이종이,윤희병.『컴포넌트 기반의 무기체계 임베디드 소프트웨어 개발 방법론 설계』, Proceedings of KFIS Autumn Conference ,2006

[5] 김준. 『컴포넌트 기반의 e-Business 개발 방법론과 모델링』, e-bizgrou working paper, 2001.06.21

[6] 경주현,이민석 『임베디드 시스템을 위한 동적 업그레이드 프레임워크에 관한 연구』,한국컴퓨터종합학술대회 Vol35 No1(B), 2008

[7] 신원, 김태완, 장천현, 『개선된 모니터링 센서를 이용한 임베디드 모니터링 시스템의 설계 및 구현』, 한국컴퓨터종합학술대회 Vol32 No1(A), p.778-780.

[8] 조은숙, 김철진, 이숙희, 『임베디드 시스템의 재사용 프레임워크를 위한 정적 메타모델 설계』, 멀티미디어학회 논문지 제12권 제2호, 2009.2, pp.231-243

[9] 박충범, 유용덕, 최훈, 『임베디드시스템용 소프트웨어플랫폼을 위한 동적 재구성 관리자 설계』, 한국컴퓨터종합학술대회 2005 논문집 Vol.32, pp.667-669

[10] 김창환, 『임베디드 소프트웨어 개념 및 동향』, 전자부품연구원 보고서, 2008.3.

[11] 김경근,정원수,오영환 『무선랜을 이용한 원격 제어 임베디드 시스템』, 한국통신학회논문지 Vol33 No4 pp105-112, 2008.4

[12] 이형석 『임베디드(Embedded) 소프트웨어』,정보과학회논문지 Vol30 No3 pp185-201

[13] 이상수,신석규 『임베디드 미들웨어 및 플랫폼 시험』, TTA Journal No.111 pp 111-116

[14] UPnp Forum "UPnP Device Architecture 1.1" 15 October 2008

[15] Roman Obermaisser, "Monitoring and Configuration in a Smart Transducer

Network", IEEE Real-Time Embedded System Workshop, 2001

[16] MAX COPPERMAN, "Symbolic Debugging of Optimized Code", ACM Transactions on Programming Languages and Systems, 1993

[17] H. Ben-Abdallah, D.Clarke, I. Lee, and O.Sokolsky, "PARAGON: A Paradigm for the Specification, Verification, and Testing of Real-Time Systems", IEEE Aerospace Conference, Vol. 2, 469-488, Feb 1-8, 1997

[18] Madisetti, Vijay K., Akgul, Tankut Debugging Embedded Systems, 2006.

[19] America P.,Obbink H., and Ommering R.,F.V.D. Linden, "CoPAM: A Component-Orented Platform Architecting Method Family for Product Family Engineering" The First Software Product Line Conference(SPLC), Kluwer International Series in Software Engineering and Computer Science Denver, Colorado, USA, p.15, 2000

[20] Axel J.,Modeling Embedded System and SOCs, Mogan Kaufmann, 2004