工學博士 學位論文

신경망을 이용한 유비쿼터스 컴퓨팅 환경의 통합관제를 위한 시스템 미들웨어 구현

The System Middleware Implementation for Integration Control of Ubiquitous Computing Environment using Neural Network

指導敎授　李　尙　培

2008年　2月

韓國海洋大學校 大學院

電子通信工學科　李 柱 尙

목 차

그 림 목 록

표 목 록

The System Middleware Implementation for Integration Control of Ubiquitous Computing Environment using Neural Network

Joo-Sang Lee

Department of Electronics & Communication Engineering
Graduate School
Korea Maritime University

Abstract

Middleware of ubiquitous computing environment is defined as the support of an application in heterogeneous environment, ubiquitous environment. Middleware-the software layer that lies between the operating system and the application-can be defined as software that helps to give users service regardless of lower hardware, operating system, network. In other words, it means software that supports communication between different protocols, system operating systems, databases and applications, and plays a role in allowing applications to operate in any information system environments.

The purpose of this research is to design and implement the system middleware for real-time integration control in ubiquitous computing

environment. To accomplish this purpose, firstly requirements for integration control in ubiquitous computing environment was analyzed, and then 3 layer structure and functions of middleware core that compose middleware were derived based on the requirements analysis. The 3 layer structure consists of Application Connector Layer which takes charge of communication connection between heterogeneous information gathering terminal equipment and middleware, External Layer which both guarantees external interface through connector and takes charge of link to internal layer through listener, and Internal Layer which both plays an important role for interface between external and middleware core and takes charge of actual business logic implementation laying on middleware core.

Middleware core manages life cycle of all middleware agents from creation to extinction such as resources management, event management, message management, process management and agent management, etc. It also not only maintains but also manages the whole functions including communications among all modules of systems and resources allocation.

Secondly, after each module of middleware was derived, the whole system structure was designed so that shows how middleware operates with derived modules. At this time, neural network model to sort priority of events by situation was designed. Moreover, the functions of each middleware module was defined concretely. Especially the classification methodology of neural network to handle priority of events in real time in the process of situation recognition of middleware for integrated control is proposed. This methodology enabled to shorten time spent on events priority classification and classify events priority more efficiently without

composing rule-database.

This research also implemented the actual middleware system based on the design and offered simple API to enable developer to implement easily even if internal action of system is unknown during developing application program in the environment where this middleware is applied.

Finally, integrated control platform in ubiquitous computing environment was constructed to test performance of the implemented middleware, and actual performance test was executed by composing virtual performance workload examination module and using system workload test tool. The result showed that response time and server resources processing ability are excellent.

제 1 장 서 론

인터넷 시대에 언급되던 사이버 스페이스가 컴퓨터와 네트워크로 구성된 가상 공간상에 사람이 개입하는 방식이었다면 제록스사의 Mark Weiser에 의해 처음 발표된 유비쿼터스 컴퓨팅(Ubiquitous Computing), 혹은 스며드는 컴퓨팅(Pervasive Computing)은 사람이 존재하는 공간에 컴퓨터 군이 개입하는 방식으로 사용자가 언제 어디서나 일상 생활 속에 편재해 있는 컴퓨팅 자원을 이용하여 다양한 서비스를 제공받을 수 있는 기반구조이다.

기존 컴퓨팅 환경에서는 키보드나 마우스와 같은 정형화된 형태의 입력 장치를 통해서 자신의 의도를 전달하고, 프린터나 모니터 혹은 사운드 카드와 같이 사용자의 오감으로 인식할 수 있는 형태의 출력장치로 그 결과를 인지하였다. 이에 비해, 유비쿼터스 컴퓨팅 환경에서는 사용자의 실생활에 편재되어 있는 다양한 센서와 컴퓨팅 자원들이 사용자의 의도와 주변 환경을 인식하고 이를 근거로 사용자에게 최적의 서비스를 제공해야 한다.

유비쿼터스 환경은 컴퓨터가 도처에 편재하여 센싱과 트랭킹을 통해 장소나 시간에 따라 변경된 정보를 서비스 받을 수 있다. 분산화된 많은 양의 컴퓨팅 노드 존재와 동적인 Ad-hoc 네트워크를 이용한 유비쿼터스 컴퓨팅은 장소, 이동여부, 휴대용 컴퓨터 기기, 통신기기 및 통신 대역폭의 다양성과 같은 물리적 요소로부터 독립적인 모델이 되어야 하며 분산화된 환경에서 변화에 따른 이기종간에 객체들이 원활하게 상호작용을 하기 위해서는 환경 및 대역폭에 독립적이고 시스템을 동적으로 재구성할 수 있는 유비쿼터스 환경에 적합한 객체들간에 통신이 가능하도록 소프트웨어 버스를 제공하는 분산객체용 미들웨어 기술이 필요하다.

특히, 유비쿼터스 환경의 통합관제는 도시내 통신망, 교통망, 시설물, 통합단말기 등의 이기종 센서기기로부터 도시정보를 수집하고 이를 통합하여 표준

화하고 분석·가공하여 단위 서비스 및 유관시스템에 정보를 연계하는 통합연동 미들웨어가 요구된다.

유비쿼터스 통합 관제를 위한 미들웨어는 각종 센서 및 디바이스들이 이기종 하드웨어 및 소프트웨어 자원에 종속되지 않고 손쉽고, 자유롭게 외부환경과 이음새 없는 서비스를 지원하기 위한 기술과 향후 개발될 새로운 기술이나 디바이스들을 용이하게 미들웨어에 수용할 수 있고, 사용자나 시장의 요구에 따라 손쉽게 확장과 축소할 수 있는 적응성, 융통성, 재사용성을 만족시킬 수 있어야 한다.

현재 통합관제 기술개발은 관제 기능의 구현을 위한 기술적 융합 및 서비스 제공을 위한 비즈니스 연계 등 업체간 다양한 제휴와 함께 경쟁이 상호 공존하는 구도를 보이고 있다. 또한 유무선 인프라 사업자간의 서비스 공유나 부가서비스(보안, 위성방송 등) 개발을 위한 업무 협력, 컨텐츠나 애플리케이션의 공동 개발을 전개하는 등 다각적인 형태의 사업이 일어나고 있다.

기존의 통합관제를 위한 시스템 구축에 대한 연구는 통합 데이터베이스 기반의 EAI(Enterprise Application Integration)기술을 적용한 시스템 통합 및 연동이 주요내용이다. 현재 대부분의 관제 시스템이 EAI기술에 의해 구축되어 운영되고 있지만, EAI기술에 의한 통합은 이기종 시스템간 인터페이스를 위한 협의가 어렵고, 각 시스템에서 공통핵심기능을 중복해서 구현하고 있으며, 신규 시스템에 대한 확장이 유연하지 못하고, 통합 데이터베이스 기반으로 운영되어 시스템 과부하에 의한 오류가 종종 발생하는 문제점을 갖고 있다.

이와 같은 문제점을 해결하기 위해 많은 연구가 수행되고 있으나, 데이터베이스 기반으로 구현되는 기술을 벗어나지 못하고 있는 현실이다.

데이터베이스 기반으로 구축되는 시스템은 센서로부터 수집한 데이터를 데이터베이스에 저장하고, 저장된 정보를 다시 조회하여 처리하는 구조이므로 실시간 데이터 처리가 어렵다. 통합관제의 핵심은 과거의 이력데이터를 검색

하여 보는 것이 아니라, 센서로부터 실시간 발생하는 데이터에 대한 처리이다.

본 논문에서는 다양한 복수 이기종 센서 데이터를 수집하는 응용 서비스들 간의 독립성을 보장하면서도 유연한 통합을 지원하고, 수집한 데이터를 공유 메모리를 사용하여 실시간으로 처리하고, 공통 핵심 기능을 지원하여 중복개발을 배제하고, 신규 서비스 확장시 유연하게 적용될 수 있는 프레임워크 구조를 지원하는 유비쿼터스 통합관제를 위한 미들웨어를 구현하였다.

또한, 데이터베이스에 종속적이지 않은 미들웨어를 구현하기 위해 다양한 환경에서의 상황이벤트 발생시 우선순위를 분류하는 과정에서 사용되는 이벤트 우선순위 알고리즘을 일반적인 룰-데이터베이스 기반의 우선순위 알고리즘 대신에 신경망을 이용한 이벤트 우선순위 분류 알고리즘으로 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서는 신경회로망 이론에 대해 설명하였다. 3장에서는 유비쿼터스 컴퓨팅 환경에서의 미들웨어에 대한 전반적인 내용을 설명하였다. 4장에서는 유비쿼터스 환경에서 통합관제를 위한 미들웨어가 갖추고 있어야 할 요구사항에 관해 먼저 파악하고, 그러한 요구 조건을 만족시키는 미들웨어의 전체 구조를 설계하였으며, 이벤트 우선순위 분류를 위한 신경망 모형을 설계하였다. 5장에서는 먼저, 이벤트 우선순위 분류를 위한 신경망 모형의 구현 및 실험결과에 대하여 설명하고, 그 다음 4장에서 설계한 내용을 바탕으로 구현된 미들웨어의 하드웨어와 소프트웨어 구성에 관하여 설명하고, 마지막으로 본 연구에서 구현된 미들웨어를 기반으로 유비쿼터스 환경의 통합관제시스템 테스트베드를 구축하여 실험 결과를 고찰하였다. 끝으로 6장에서는 본 연구의 결론과 앞으로 추가 연구되어야 할 사항에 대하여 제시하였다.

제 2 장 신경회로망

본 논문에서 제안된 미들웨어에 적용한 인공지능 이론을 제시하고자 한다. 인공지능은 인간의 지능으로 할 수 있는 학습능력, 의사결정능력 등을 컴퓨터가 할 수 있도록 하는 방법을 의미한다. 이러한 방법들은 컴퓨터가 인간의 지능적인 행동을 모방할 수 있도록 컴퓨터 공학 및 정보기술의 한 분야로서 대두되고 있다. 특히 현대 정보화 사회에서는 정보기술의 여러 분야에서 인공지능적 요소를 도입하여 그 분야의 문제 풀이에 활용하려는 시도가 활발하게 연구되어지고 있다.

인간의 학습능력을 이론적으로 구현한 것이 신경회로망(neural networks) 이론이고 의사결정능력을 구현하는 것이 바로 퍼지 논리(fuzzy logic) 이론이라고 할 수 있다. 본 논문에서는 미들웨어의 경합 알고리즘을 위하여 인공지능 기법 중의 하나인 신경회로망을 사용하였다.[5]

2.1 신경회로망

신경회로망은 인간과 의사소통하고 학습을 통해 지식과 경험을 축척 하여 스스로 상황을 판단할 수 있는 인간과 유사한 것을 만들고자하여 생겨난 인공지능의 한 분야로써, 인간을 비롯한 동물들이 가지고 있는 뇌에 대한 연구결과를 근거하여 인공적으로 지능을 만들어 보고자 하는 연구이다. 다시 말해, 뇌에 존재하는 생물학적 신경세포와 그것들의 연결 관계를 단순화시키고 수학적으로 모델링함으로써 뇌가 나타내는 지능적 형태를 구현해 보고자 하는 것이다.

신경회로망은 개념적으로 매우 단순하며 그러한 단순함에도 불구하고 복잡한 뇌가 나타내는 여러 가지 특성들을 보여주고 있다. 특히, 사람의 뇌가 경험

을 통해 학습하듯이, 주어진 입력에 대해 자신의 내부구조를 스스로 조직화함으로써 학습해 나가는 능력은 신경회로망이 가지는 독특한 특성중의 하나이다.

지금까지의 연구 결과를 통해 신경회로망은 애매하고 불완전한 화상, 음성, 문자 등의 패턴인식과 특징 추출, 로봇이나 플랜트 등의 제어[1]-[4], 각종 센서로부터의 정보 인식, 판단, 계획 등의 처리와 그들 결과를 이용한 조작기의 운동제어와 자기 조직화 기능을 적용하면 실시간 제어가 가능하고 동시에 적응성 있는 로봇의 구현이 기대된다. 원인과 결과의 인과 관계가 애매하고 복잡한 문제 즉, 기후나 지진, 경제문제 등을 신경회로망의 자기 조직화를 이용하여 복잡하고 어려운 규칙을 자동으로 획득하는 문제 등에 응용되고 있다.

2.1.1 신경회로망 배경

신경회로망에 대한 연구는 1943년 McCulloch와 Pitts에 의해 처음으로 가능성이 제시되었다.[5] 그들은 인간의 두뇌를 수많은 신경세포로 이루어진 잘 정의된 계산기라고 생각했다. 단순한 논리적 임무를 수행하는 모델을 보여주었고, 또한 패턴분류 문제가 인간의 지능적인 행위를 규명하는 이론에 매우 중요하다는 것을 인식하였다.

Hebb은 두 뉴런사이의 가중치(weight)를 조정할 수 있는 최초의 학습 규칙에 대한 제안을 하였고, 이 규칙은 학습에 관한 연구를 발전시켰으며 적응적인 신경회로망의 연구에 많은 영향을 끼쳤다. 실질적인 신경회로망에 관한 연구는 1957년 Rosenblatt의 퍼셉트론(perceptron) 모델이 발표되면서부터라 할 수 있다.

여기에서는 학습 과정에 알파 강화 규칙을 사용하고 있으며, 퍼셉트론에 대한 관심의 주된 이유는 어떤 타입의 패턴이 입력층에 주어졌을 때 이 모델이 반응하게 하는 가중치의 집합을 스스로 발견하는 자동적인 절차에 있다. 학습

은 현재 주어진 입력에 대하여 현재의 각 가중치를 조정함으로써 얻어질 수 있었다.

Minsky와 Papert등이 “퍼셉트론즈(perceptrons)”란 저서에서 퍼셉트론 모델을 수학적으로 철저히 분석하여, XOR(eXclusive OR) 함수와 같이 단순한 비선형 분리 문제도 풀 수 없다는 것을 밝혀내고 난 후 신경회로망에 관련된 연구는 약 20년간 침체의 길을 걷게 되었다.[5]

그러나 1970년대 말과 1980년대 초반 Kohonen, Hopfield, Kirkpatrick, Hinton, Grossberg, Rumelhart 등이 역전파(backpropagation) 학습 알고리즘을 이용하여 신경회로망을 다시 활성화시켰다. 여기서 사용되어진 역전파 학습 알고리즘은 오차를 정정하는 규칙으로써, 입력에 대해 원하는 반응과 실제로 얻어진 것들에 대한 차이를 줄여 나가는 것이다. 오차 전파(error propagation)에 의한 내부 학습에서 입력패턴은 충분한 은닉층 유니트들만 있으면 항상 코드화될 수 있다. 이 과정은 신경회로망의 가중치를 반복적으로 조정하여 실제 신경회로망의 벡터와 원하는 출력간의 차이를 줄여 나간다.

2.1.2 생물학적 뉴런과 인공적 뉴런 사이의 고찰

뉴런(neuron)은 생체 속에서 정보 처리를 위해 특별한 분화로 이루어진 세포이다. 그림 2.1에 나타난 것처럼 뉴런은 크게 정보처리의 핵심이 되는 세포체, 또는 소마(soma)와 기억의 핵심이 되는 뉴런의 연결 사이에 존재하는 매우 좁은 간극 즉, 시냅스(synapse)로 이루어져 있다. 생물학적 뉴런의 신호 전달 과정은 시냅스 전 뉴런의 활동 전위 펄스가 시냅스 후 뉴런의 막전위 변화를 일으킴으로써 정보가 전달된다.

뉴런은 자신과 연결된 많은 다른 뉴런들로부터 전기, 화학적 신호들을 시냅스를 통해 받아들여 종합한다.

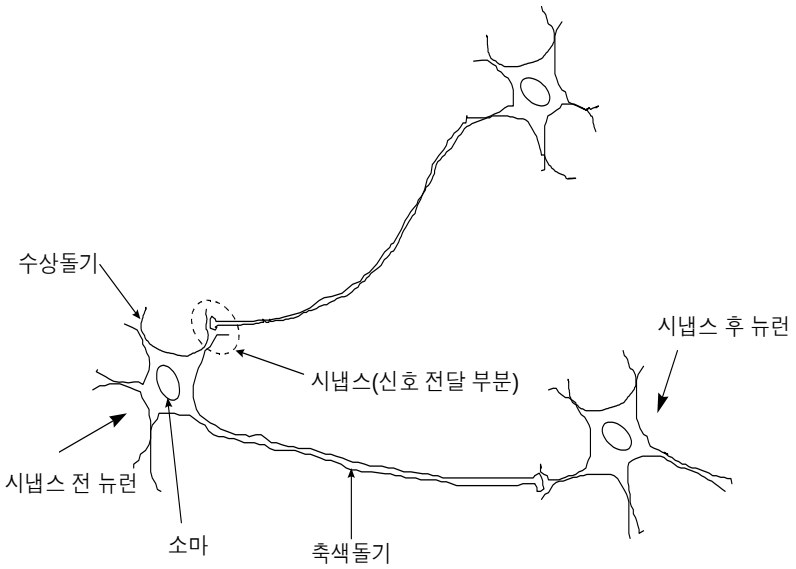


그림 2.1 생물학적 뉴런의 구조

그 값이 임계치라고 부르는 뉴런 고유의 한계 값보다 커지면 뉴런은 발화(fire)되어 다른 신경세포에 자신의 출력을 전달한다. 즉, 다른 뉴런들로부터 받아들인 신호들을 종합한 결과, 자신의 내부 전압이 임계치 이상이 되면 그 뉴런은 발화되고, 그렇지 않은 경우 발화되지 않는다.

뉴런들 간의 정보전달 과정에서 매우 중요한 것은 어떤 뉴런들 간에는 비교적 정보가 잘 전달되는 반면, 어떤 뉴런들 간에는 정보가 잘 전달되지 않는다는 점이다. 그리고 이러한 정보전달 정도에 대한 권한과 책임은 시냅스에게 있다. 뉴런을 수학적으로 모델링하면 그림 2.2와 같이 된다.[7][8]

수상 돌기(dendrite)는 다른 뉴런으로부터 입력을 받아들이는 곳으로써 이 입력을 소마로 운송하는 것으로 그림 2.2에서 입력 x를 뜻한다. 소마는 수상돌기로부터 받은 값들을 처리하는 부분으로 그림 2.2에서는 값이 집결되는 부

분을 말한다. 축색돌기(axon)는 소마로부터 처리된 값을 다른 뉴런으로 전송하는 부분 즉, 출력되는 부분을 말한다. 시냅스에서는 뉴런과 뉴런사이의 신호 전송을 담당하는 부분으로써 펄스가 발생했을 때 이 펄스에 의한 전위가 막전위 보다 높았을 때는 흥분성 시냅스가 되어 신호가 전송이 되고 낮을 때는 억제성 시냅스가 되어 신호 전송을 막는다. 일반적으로 시냅스는 신경회로망에서 조정 가능한 가중치라 한다.

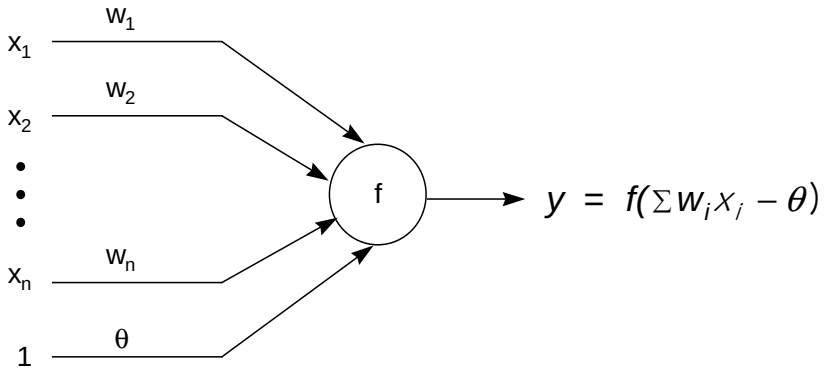


그림 2.2 기본적인 뉴런의 수학적 모델링

요약해서 말하면, 그림 2.2의 뉴런은 입력과 각 입력에 대응하는 가중치를 곱하여 합한 값[$\sum(W_i * X_i - \Theta)$]에 대해서 함수관계를 적용한 값을 출력한다. 여기에서 Θ는 뉴런 자체의 임계치 또는 바이어스(bias)로써 $\sum(W_i * X_i)$이 임계치 보다 작을 때는 뉴런이 활성화되지 않도록 하는 역할을 한다. 또한 함수 f을 신경회로망에서는 활성함수(activation function)라고 하며, 그림 2.3에 나타내었다.

신경회로망에서 많이 사용되는 대표적인 활성함수로는 단극성 함수(unipolar function), 양극성 함수(bipolar function), 그리고 선형 함수(linear function) 등이 있다.

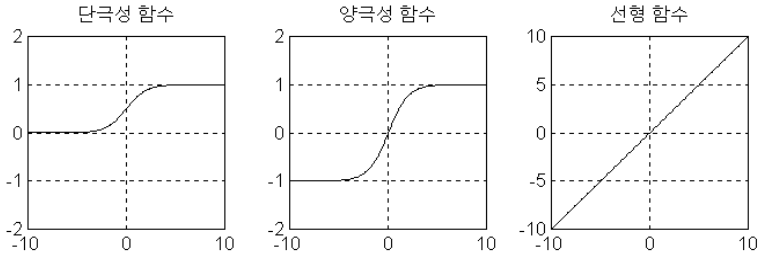


그림 2.3 뉴런의 활성 함수

2.2 다층 신경회로망의 학습과 구조

패턴인식이나 특징추출, 시스템인식과 제어에 대해 가장 일반적으로 사용되는 신경회로망은 다층 신경회로망(multi-layer neural network)이고, 가장 공통적으로 적용되는 학습알고리즘은 오류 역전파 알고리즘(error back propagation algorithm)이다. 전형적인 다층 신경회로망은 그림 2.4에 나타내었다.

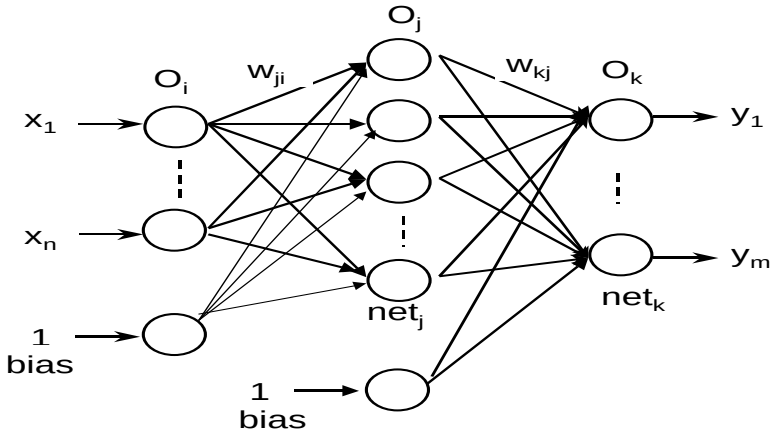


그림 2.4 다층 신경회로망의 구조

그림 2.4에서 각각의 원은 그림 2.2에서 보여준 뉴런이다. 이 신경회로망은 x라는 입력을 갖는 입력층과 y라는 출력을 갖는 출력층으로 이루어지며 입력층과 출력층 사이의 층을 우리는 보통 은닉층이라 한다.

여기서 O_i, O_j, O_k는 입력층, 은닉층, 출력층의 출력을 의미하고, 입력층과 은닉층 사이의 가중치를 W_{ji}, 은닉층과 출력층 사이의 가중치를 W_{kj}로 표기한다. 모든 정보는 신경회로망의 가중치에 저장되며, 학습 과정 동안 가중치 W_{ji}, W_{kj}의 성분은 계속적으로 새로운 정보로 바뀌어 진다. 일반적으로 새로운 정보를 변경하는 신경회로망의 대표적인 알고리즘은 오류 역전파 알고리즘이다. 오류 역전파 알고리즘은 하나의 신경회로망에서 각각 뉴런에 의해 계산된 출력과 바라는 출력사이의 오차를 자승하여 최소화시키는 최소 평균 자승법(least-mean square method)을 사용한다.

오류 역전파 학습 알고리즘의 원리는[1] 먼저 입력층에서는 신경회로망의 입력 x를 은닉층으로 보낸다. 두번째로 은닉층의 뉴런들은 각각의 입력층으로부터 입력된 값과 가중치들의 곱을 합산함과 동시에 활성함수를 통해 연산된 결과인 뉴런의 출력을 출력층으로 보낸다. 출력층은 은닉층과 같은 뉴런 연산을 하여 출력한다. 이때 신경회로망의 출력값이 바라는 목표값과의 차이를 구하며, 이 차를 오차라고 말한다. 이 오차를 최소화 하기위해 각 층에 있는 가중치들을 오차 벡터 항들의 편미분을 계산하여 가중치를 조정한다. 다시 말해 출력층의 출력과 바라는 목표치 사이의 오차를 연산한 후 출력층에서 은닉층으로, 은닉층에서 입력층으로 역전파하여 오차에 따른 각각의 가중치 변화량에 의해 가중치들을 조정하며 이것을 오류 역전파라고 말한다. 그림 2.4를 수식적으로 표현하면 입력층, 은닉층, 그리고 출력층은 다음과 같은 기호로 나타낼 수 있다.

O_i : 입력층의 출력값

O_j : 은닉층의 출력값

O_k : 출력층의 출력값

j번째 은닉층의 뉴런으로부터 k번째 출력층의 뉴런간의 가중치를 W_{kj}로 표기한다. 따라서, 출력층을 계산하면 식 (2.1)과 같다.

$$O_k = f(net_k) \tag{2.1}$$

$$net_k = \sum_j W_{kj} O_j \tag{2.2}$$

이와 같은 방법으로 은닉층을 계산하면 식 (2.3)이 된다.

$$O_j = f(net_j) \tag{2.3}$$

$$net_j = \sum_i W_{ji} O_i \tag{2.4}$$

여기서, 활성함수는 아래와 같은 단극성 함수를 사용하였다.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

신경회로망을 학습시키기 위해 신경회로망의 출력값이 바라는 목표값과의 차인 오차를 구해야 하며, 이 오차를 구하는 수식은 식 (2.6)에 나타내었다. 여기서 D_k는 출력층의 k번째 뉴런의 목표값을 의미한다.

$$E = \frac{1}{2} \sum_k (D_k - O_k)^2 \tag{2.6}$$

학습의 목적은 가중치를 조정하여 E를 최소화하는 것이므로 가중치의 조정에 대하여 살펴보면 오차를 최소화하기 위해 가중치를 음의 경사방향(negative gradient direction)으로 변화시켜야 한다. 따라서 가중치를 음의 경사 방향으로 오차에 대한 가중치의 방향 벡터를 편미분 함으로써 가중치 변화량을 구할 수 있다. 각 층에 있는 가중치 변화량을 구하면 다음과 같다.

$$\triangle W_{kj} = -\eta \frac{\partial E}{\partial W_{kj}}, \qquad \eta > 0 \tag{2.7}$$

여기서 η는 학습 속도를 나타내는 상수이며 이것을 학습률이라 한다. 따라서, 일반화된 오차신호는 다음과 같다.

$$\delta_k = -\frac{\partial E}{\partial net_{kj}} \tag{2.8}$$

식 (2.8)을 연쇄 규칙(chain rule)을 사용하여 아래와 같이 간단하게 쓸 수 있다.

$$\frac{\partial E}{\partial W_{kj}} = \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial W_{kj}} = -\delta_k \frac{\partial net_k}{\partial W_{kj}} \tag{2.9}$$

식 (2.2)에 의해서 다음과 같이 된다.

$$\frac{\partial net_k}{\partial W_{kj}} = \frac{\partial(\sum_j W_{kj}O_j)}{\partial W_{kj}} = O_j \tag{2.10}$$

그러므로, $\triangle W_{kj} = \eta\delta_k O_j$가 되며, δ_k는 아래와 같이 계산된다.

$$\delta_k = -\frac{\partial E}{\partial net_k} = -\frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial net_{kj}} \tag{2.11}$$

여기서, 식 (2.6)으로부터 다음과 같이 계산된다.

$$\frac{\partial E}{\partial O_k} = -(D_k - O_k) \tag{2.12}$$

또한, 식 (2.1)에 의해서 다음과 같이 표현된다.

$$\frac{\partial O_k}{\partial net_k} = f'(net_k) \tag{2.13}$$

식 (2.5)에 의해 활성함수의 미분은 다음과 같이 쓸 수 있다.

$$\begin{aligned} f'(x) &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{(1+e^{-x})}\left(1 - \frac{1}{1+e^{-x}}\right) \\ &= f(x)(1-f(x)) \end{aligned} \tag{2.14}$$

그러므로,

$$\frac{\partial O_k}{\partial net_k} = O_k(1 - O_k) \tag{2.15}$$

이다. 따라서, 가중치의 변화분은 다음과 같다.

$$\begin{aligned} \triangle W_{kj} &= \eta\delta_k O_j \\ \delta_k &= O_k(1 - O_k)(D_k - O_k) \end{aligned} \tag{2.16}$$

지금까지는 출력층에 대한 가중치 변화에 대하여 기술하였고, 다음은 은닉층에 대한 가중치의 변화에 대해서 기술한다. 여기서도 오차를 최소화하기 위해 가중치를 음의 경사 방향으로 변화시켜 준다.

$$\triangle W_{ji} = -\eta\frac{\partial E}{\partial W_{ji}}, \qquad \eta > 0 \tag{2.17}$$

식 (2.17)을 연쇄 규칙을 사용하여 아래와 같이 간단하게 쓸 수 있다.

$$\frac{\partial E}{\partial W_{ji}} = \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial W_{ji}} \tag{2.18}$$

$$\delta_j = -\frac{\partial E}{\partial net_j} \tag{2.19}$$

식 (2.4)로부터

$$\frac{\partial net_j}{\partial W_{ji}} = O_i \tag{2.20}$$

이다. 여기서, 입력층의 출력값 O_i는 신경회로망 입력값 X_i와 같다. 즉, $O_i = X_i$이다. 그러므로, $\triangle W_{kj} = \eta\delta_j O_i$가 되고, δ_j는 아래와 같이 계산된다.

$$\delta_j = \frac{\partial E}{\partial net_j} = -\sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial O_j}\frac{\partial O_j}{\partial net_j} \tag{2.21}$$

여기서, 식 (2.2), (2.3), (2.8)으로부터 다음과 같이 쓸 수 있다.

$$\begin{aligned}\delta_j &= \sum_k \delta_k W_{kj} f'(net_k) \\ &= O_j(1 - O_j)\sum_k \delta_k W_{kj}\end{aligned} \tag{2.22}$$

따라서 가중치의 변화는 다음과 같다.

$$W_{ji}' = W_{ji} + \triangle W_{ji} \tag{2.23}$$

$$W_{kj}' = W_{kj} + \triangle W_{kj} \tag{2.24}$$

지금까지 설명했듯이, 오류 역전파 알고리즘은 오류 신호를 계산하고 신경망의 가중치들을 조정하기 위해서는 바라는 목적값이 필요하다. 이러한 초기의 학습 후에 신경망은 학습에 사용되지 않은 새로운 데이터의 집합을 입력할 수 있다. 학습된 데이터의 집합이 아닌 데이터를 갖는 신경회로망의 정확성은 신경망에 일반화(generalization) 능력을 부여한다. 그리고 이것은 곧 신경망의 신뢰도를 가리킨다. 학습과 검사단계 후에, 신경회로망은 패턴 분류기, 또는 잘모르는 비선형 함수와 복잡한 처리를 모델화 하는데 사용될 수 있다.

2.3 오류 역전파 알고리즘의 학습 요소들

오류 역전파 알고리즘에서 중요한 요소는 가중치에 대한 적절한 평가에 있다. 오류 역전파 알고리즘에서 오차 최소화 과정의 문제점 중의 하나는 오차 함수의 지역 최소점(local minimum)에 빠질 수 있다는 것이다. 그림 2.5에서 학습이 a나 b의 지점에서 시작되면 학습이 멈출 수 있다. 반면에, 학습이 c에서 시작된다면 지역 최소점이 아닌 우리가 허용할 수 있는 오차 E_{min}에 접근할 수 있을 것이다. 지금부터 오류 역전파 알고리즘에 대한 중요한 요소에 대하여 설명을 한다.

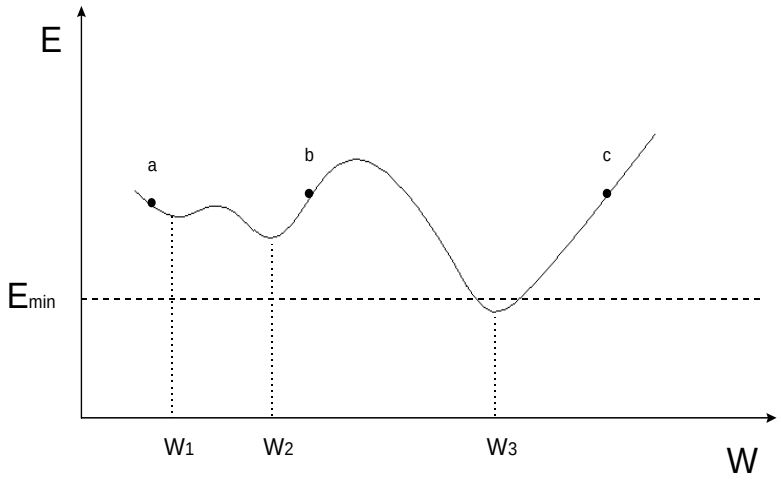


그림 2.5 가중치와 오차 함수의 관계

2.3.1 초기 가중치

초기 가중치는 작은 무작위값(random value)으로 설정하고 이런 초기화는 최종 출력에 영향을 미친다. 초기 가중치가 지역최소점에 빠졌다면 가중치가 고정된 값에서 머물게 되므로 다른 지점에서 학습을 다시 시작해야 한다. 이것은 그림 2.5에서 a, b에서 시작한 경우에 해당되며 c에서 다시 시작하면 원하는 값을 얻을 수 있다. 보통 일반적으로 초기 가중치는 -0.5에서 0.5사이의 값을 주로 사용한다.

2.3.2 누적 가중치 조정과 증분 갱신

식 (2.6)은 각 학습 패턴에 따라 가중치를 조정하는 방식으로 증분 갱신이라 한다. 학습률이 충분히 작고 p는 학습 패턴의 개수라고 한다면, 식 (2.25)는 모든 학습패턴이 종료한 후 가중치를 조정한다.

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{K}(D_{pk} - O_{pk})^2 \qquad (2.25)$$

이런 방법을 누적 가중치 조정이라 한다. 증분 갱신 방법에 의한 가중치 조정은 실제 컴퓨터 계산 과정에서 조정치가 저장될 필요가 없고, 매 학습단계마다 변화한다. 그러나 이런 방법은 가중치의 조정량이 가장 최근의 가중치에 크게 영향을 받는다. 이런 문제 해결을 위해 다음과 같이 누적 가중치 조정법을 사용한다.

$$\Delta w = \sum_{p=1}^{P} \Delta w_p \tag{2.26}$$

증분 갱신법에서 너무 적은 학습률을 사용하면 학습시간이 늦지만, 일반적으로 누적 가중치 조정법 보다는 증분 갱신법을 많이 사용한다.

2.3.3 활성함수의 기울기

뉴런의 연속 활성 함수는 기울기 λ에 의해 특성이 결정되고 미분치 $f'(net)$은 에러 신호 δ를 만드는 구성 요소이므로, 이 두 요소는 신경회로망의 학습 속도에 영향을 미친다.

$$f(net) = \frac{2}{1+e^{(-\lambda net)}} - 1 \tag{2.27}$$

그림 2.6은 식 (2.27)의 수식에 대한 미분치를 나타내고 λ의 변화에 따른 미분치의 결과는 식 (2.28)와 같이 된다.

$$\frac{df}{d\lambda} = f'(net) = \frac{2\lambda e^{(-\lambda net)}}{[1+e^{(-\lambda net)}]^2} \tag{2.28}$$

net = 0일 때 $f'(net) = \frac{2\lambda}{(1+1)^2} = 0.5\lambda$이므로 λ를 그림 2.6에서와 같이 변화시키면서 값을 확인할 수 있다. 그림에서 알 수 있듯이, 미분치는 λ값이 일정할 때 중간 대역의 값이 가장 크고 net값이 일정할 때 λ값이 클수록 커진다. 즉 이러한 때 가중치의 변화폭이 가장 크다. 학습 상수가 고정되어 있을 때 가중치의 조정치는 λ값에 비례한다. 이것은 큰 학습 상수의 값을 사

용하는 것과 같은 효과이다. 학습률과 λ값을 같이 조정하기 보다는 λ는 1로 놓고 학습 상수로 학습속도를 조정하는 것이 바람직하다.

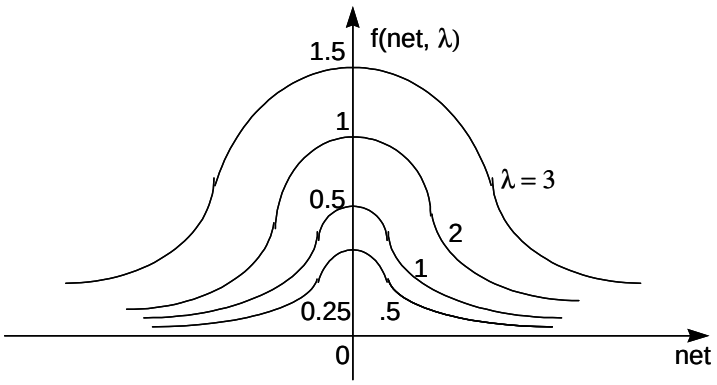


그림 2.6 활성화 함수의 기울기

2.3.4 학습률

오류 역전파 알고리즘의 수렴 정도는 학습률에 의해서도 달라질 수 있다. 학습률은 신경회로망의 구조와 응용 목적에 따라 각각 달리 선택되고 일정한 기준이 없다. 보통 0에서 1사이의 값을 사용한다.[1] 기울기가 작고 폭이 넓은 최소 지점에서는 큰 학습률을 사용하여 빠른 수렴을 이끌어 내야 할 것이고, 기울기가 크고 폭이 좁은 최소 지점에서는 작은 학습률을 사용하여 최소 지점을 지나쳐 버리는 오버슈트가 일어나지 않도록 해야 한다. 큰 학습률을 사용하면 오버슈트가 일어날 수 있고 작은 학습률을 사용하면 학습속도가 느려질 수 있으므로 위에서 제시한 범위 내에서 적절히 선택해야한다.

2.3.5 모멘텀 방법

모멘텀 방법[1]도 오류 역전파 알고리즘에서 수렴 속도를 향상시키는 목적으로 사용한다. 여기서 식 (2.17)에서 가장 최근의 가중치 변화량을 추가시킨 것

이 모멘텀 방법(momentum method)이고 추가시킨 항을 모멘텀 항(momentum term)이라 하고, 아래와 같이 표현되어질 수 있다.

$$\Delta w(t) = -\eta \nabla E(t) + \alpha \Delta w(t-1) \tag{2.29}$$

또, 상수 α 를 모멘텀 상수(momentum constant)라고 한다. 보통 모멘텀 상수는 0.1에서 0.8의 값으로 사용한다. 모멘텀 방법을 사용한 N 스텝 전체의 가중치 변화량은 식 (2.30)과 같다.

$$\Delta w(t) = -\eta \sum_{n=0}^{N} \alpha^n \nabla E(t-n) \tag{2.30}$$

그림 2.7에서 A′에서 경사 하강법이 시작된다고 할 때, 모멘텀을 사용한 다음단계의 A′′은 이전단계의 변화량 $\Delta w(t)$ 을 추가하면서 하강속도가 더 빨라진다. 즉 수렴 속도가 향상된다. B′에서는 (-)방향으로 가중치가 변화하지만 B′′에서는 반대 방향인(+)방향으로 향한다. 이것은 최소지점 M을 지나치는 오버슈트가 발생하게 한다.

여기서 이전의 B′의 (-)가중치 변화량을 모멘텀항으로 추가하면 이러한 오버슈트를 줄이게 되어 학습의 신뢰도를 향상시킨다.

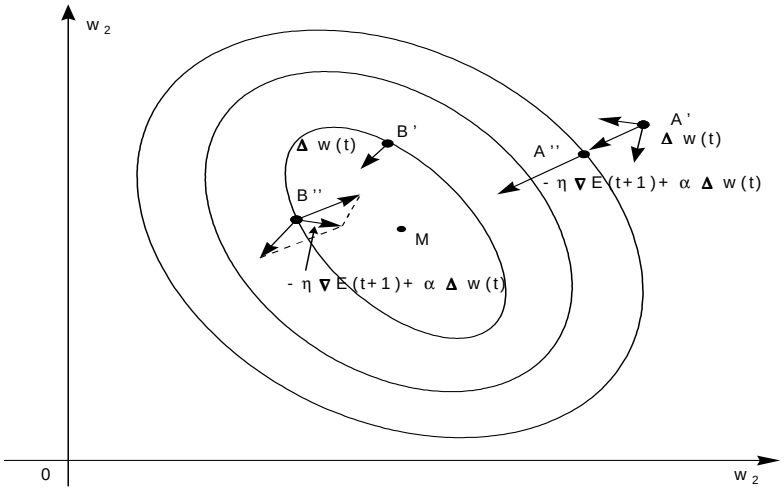


그림 2.7 오류 역전파 알고리즘에 모멘텀항 추가

제 3 장 유비쿼터스 미들웨어

3.1 유비쿼터스 미들웨어 개요

유비쿼터스 컴퓨팅(Ubiquitous Computing)은 다양한 컴퓨터가 사용자를 중심으로 동작하는 것이라고 말할 수 있는데, 유비쿼터스는 라틴어로 “언제어디서나”, “동시에 존재한다”라는 의미를 갖는다. 이를 위하여 상황인지(context awareness)와 통신 기능을 가진 칩, 센서 및 컴퓨터가 주위의 모든 사물 또는 공간에 보이지 않게 내장되어 사람, 사물 및 기계 등 무엇이든지 서로 접속하여 실시간으로 어떠한 정보든지 주고받을 수 있으며, 이러한 방법을 통하여 컴퓨팅(Computing), 커뮤니케이션(Communication), 접속(Connection), 콘텐츠(Contents), 조용함(Calm)등 5C의 5Any화(Anytime, Anywhere, Anynetwork, Anydevice, Anyservice)를 제공함으로써 인간에게 최적의 서비스를 제공할 수 있는 차세대 기술이다.

유비쿼터스 환경은 컴퓨터가 도처에 편재하여 센싱과 트래킹을 통해 장소나 시간에 따라 변경된 정보를 서비스 받을 수 있는 환경을 말한다. 분산화된 많은 양의 컴퓨팅 노드 존재와 동적인 Ad-hoc 네트워크를 이용한 유비쿼터스 컴퓨팅은 장소, 이동여부, 휴대용 컴퓨터 기기, 통신기기 및 통신 대역폭의 다양성과 같은 물리적 요소로부터 독립적인 모델이 되어야 하며 분산화된 환경에서 변화에 따른 이기종간에 객체들이 원활하게 상호작용을 하기 위해서는 환경 및 대역폭에 독립적이고 시스템을 동적으로 재구성할 수 있는 유비쿼터스 환경에 적합한 객체들간에 통신이 가능하도록 소프트웨어 버스를 제공하는 분산객체용 미들웨어 기술이 필요하다.

미들웨어는 클라이언트가 서버로 서비스를 요청하는 일종의 가교 역할을 한다. 유비쿼터스 미들웨어는 이 기종 환경, 유비쿼터스 환경에서의 응용의 지원

으로 정의가 되고 있다. 유비쿼터스 세상을 실현시키기 위해서는 실생활에 있는 모든 장치가 작게나마 컴퓨팅 능력을 가지고 있고, 또한 서로가 서로를 인지하며, 서로 정보를 교환하고 통신을 하여 서비스를 실현시키게 된다. 또한 지금까지의 응용제품이 특정한 서비스를 위해서만 존재하지 않고, 상황과 공간 또는 시간을 인지하여 그에 합당한 서비스를 해주게 된다. 이러한 세상을 구현하기 위해서는 각종 컴퓨터, 센서, 각종 기계 정보, 가전제품, 포스트PC 등의 다양한 플랫폼에서 다양한 서비스를 실현 시킬 수 있어야 한다. 하지만 이기종 하드웨어 및 스프트웨어 자원이 너무 다양하기 때문에 상호 운영성과 호환성을 유지하기란 굉장히 어렵다.[9]-[12]

그러므로 유비쿼터스 산업에서 미들웨어의 중요성은 매우 커지게 된다. 일반적 미들웨어란, 운영체제와 응용 사이에 존재하는 소프트웨어 계층으로 사용자에게 하부의 하드웨어나, 운영체제, 네트워크에 상관없이 서비스를 제공할 수 있도록 도와주는 소프트웨어로 정의할 수 있다. 즉 서로 다른 프로토콜이나 시스템 운영체제, 데이터베이스와 애플리케이션 간에 통신을 지원해 주는 소프트웨어를 의미하며, 애플리케이션이 어떤 정보시스템 환경에서도 작동할 수 있도록 지원해 주는 역할을 한다.

유비쿼터스 미들웨어란 서비스에 따라 동적으로 구성될 수 있는 정형성이 없는 것이 특징이다. 또한, 유비쿼터스 컴퓨팅을 위한 미들웨어는 기존에 개발되어 있는 수많은 미들웨어와는 또 다른 차별성을 가져야만 한다.

유비쿼터스 컴퓨팅 환경에서는 다양한 장치들이 서브 네트워크로 연결되고, 그 장치들이 서로 정보를 교환하고 공유해야 하며, 이러한 네트워크 환경은 단순한 유선 무선뿐만 아닌 광역 모바일 네트워크와 ad-hoc 네트워킹을 지원해야만 한다. 즉, 다양한 네트워크 환경에서 다양한 장치들간의 투명성을 제공해야만 하는 것이다. 또한 다양한 장치들이 서로를 인지하고 통신하며 적합한 서비스를 제공하기 위해서는 물리적 공간, 시간, 네트워크 환경, 각종 장치들

을 인지하여 서비스를 제공하는 상황인식 기술 또한 반드시 필요하다. 또한 매우 다양한 장치들이 통신하며 운영되기 때문에 효율적인 자원 관리는 필수적이다. 이러한 기능을 제공하는 기술이 컨텍스트 관리기술이다. 이 또한 유비쿼터스 지향의 미들웨어에서 제공되어져야할 필수적 요소이며 또한 미들웨어의 역할인 것이다.

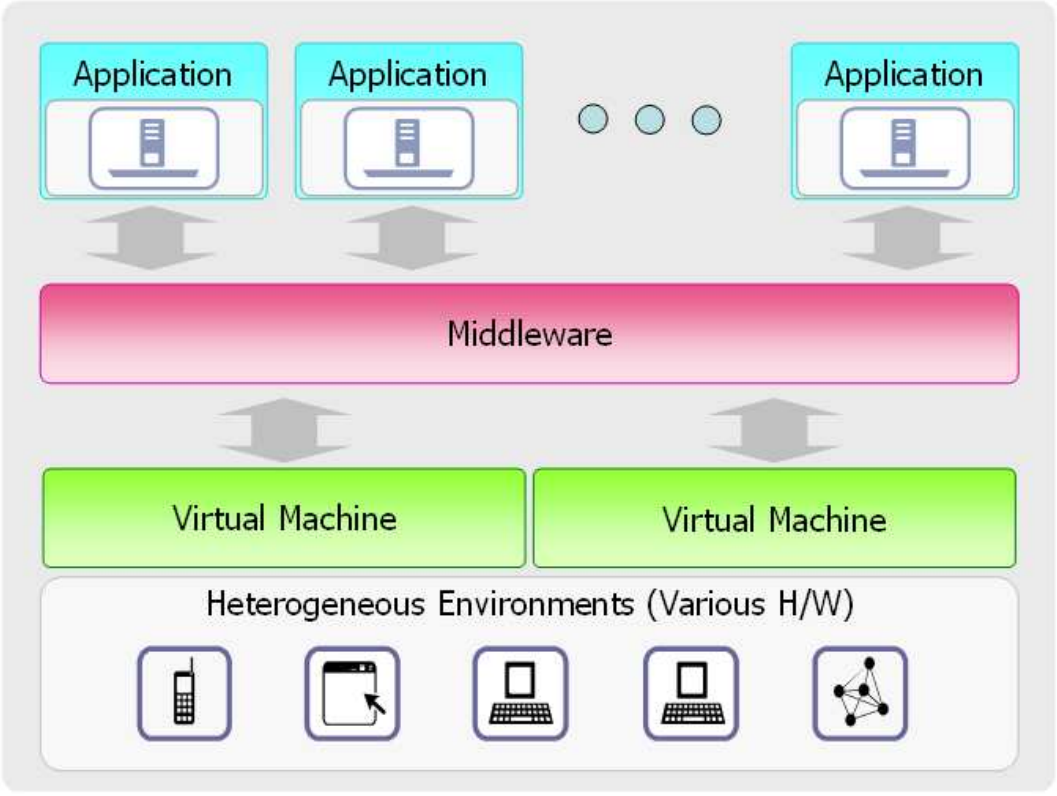


그림 3.1 유비쿼터스 환경에서의 미들웨어 연동

3.2 유비쿼터스 미들웨어 분류

유비쿼터스 환경의 미들웨어란 상황인식 기반의 미들웨어라는 포괄적 의미에 아래와 같이 세분화된 미들웨어 기술로 분류할 수 있다.[10]

첫째, Control 미들웨어란 다양한 네트워크에 오리엔트 되어진 서비스기반의 미들웨어로 유비쿼터스 환경으로 진입을 위해선 통합되어진, 혹은 공용 플랫폼상의 번들 서비스로 이루어져야 한다. 현재는 각각의 표준에 의존적인 중복된 서비스를 구현하는데 초점이 맞추어졌으나 각 표준끼리의 호환을 위해 많은 노력이 이루어지고 있다.

둘째, QoS 미들웨어로 유비쿼터스 서비스의 중심기술인 다양성에 있어서 서비스의 질적 보장이 없인 어떠한 서비스도 사용자의 만족을 가져올 수 없다. 이러한 서비스들을 보장할 수 있는 미들웨어로 네트워크 측면과 응용측면에서 QoS를 보장해 주는 미들웨어 기술들이 개발되어져야 한다.

셋째, 상황인지 미들웨어란 사용자의 환경 변화, 이기종 호스트와 네트워크에 적용될 수 있는 애플리케이션을 위해서, 시스템은 자신이 사용되는 컨텍스트를 인지할 수 있는 모바일 애플리케이션을 제공해야 한다.

사용자 컨텍스트에는 시스템의 위치정보, 근접한 프린터나 데이터베이스 등과 같은 연관된 위치 정보, 프로세싱 파워나 입력 장치와 같은 디바이스의 특성 정보, 잡음수준과 대역폭과 같은 물리적 환경 정보, 사용자의 움직임 정보, 사회적 관계정보 등이 있다.

표 3.1 유비쿼터스 환경의 미들웨어

분류	종류	특징
Control 미들웨어	HAVI	AV기기를 중심으로 미들웨어 표준정의 및 확산, HAVi-Jini, HAVi-UPnP브릿지 개발
	Jini	홈네트워크 환경에 적합한 서비스 확산을 위한 하부구조정의, IP 기반의 분산된 환경에 적합하며, 차세대 정보가 전분야인 유비쿼터스 환경으로 확장가능
	UPnP	홈네트워크 환경에서 디바이스를 연결하고 제어를 가능하게 하는 기술정의, 2000년들어 MS사가 적극

분류	종류	특징
		지원한다는 이유로 가장 빠르게 확산
	LonWork	전력선을 이용하는 전등, 센서, 백색 가전기기를 구성하고 제어하는 표준정의, 기능과 성능면에서 전력선을 이용하는 기술 중 가장 우수하며 빌딩, 공장자동화에 널리활용
QoS Aware 미들웨어	Nahrstedt	유비쿼터스 응용에게 QoS specification, QoS compilation, QoS Setup 그리고 QoS Adaptation로 구성된 QoS 프레임웍 제공
	Campbel	멀티미디어 응용의 터미널 이동성 지원을 위한 QoS 인지 미들웨어 플랫폼(Mobiware)을 제안
	Yamazaki	분산환경 멀티미디어 응용을 위한 Agent 기반 적응적 QoS 관리 프레임웍 MARM 제안, 계층간 Parameter mapping, QoS 현상, QoS 적응 기법 수행.
	ICEBERG	인터넷과 전화망을 통합한 네트웍 환경에서 사용자 이동성 지원 IAP(ICEBERG Access Point)를 네트웍 접속점에 위치시켜 데이터와 시그널을 매핑하는 방식
	MPA	개인마다 Personal Proxy를 두어 사용자 이동과 데이터 변환을 수행
	Cui	유비쿼터스 환경에서 끊김없는(seamless) 멀티미디어 서비스를 위한 사용자 수준 핸드오프 프로토콜을 제안
	NetChaser	이동 Agent 기반 인터넷 서비스 제공
상황인지 미들웨어	Dey	Context: 사람, 장소, 사물의 상황에 대한 특징적인 상태정보 Context Toolkit
	RCSM	응용들의 개발과 운영을 지원하기 위한 CORBA,

분류	종류	특징
		ORB 기반 미들웨어
	Ranganathan	술어(predicate)에 기반을 둔 컨텍스트 모델을 제시하고 이를 이용한 컨텍스트 인식 미들웨어 구조 제시

3.3 유비쿼터스 미들웨어 기술 특성

유비쿼터스 미들웨어를 추구하기 위한 기본 요소들은 확장성, 신뢰성, 이기종, 가용성, 견고성, 이동성, 보안성 그리고 상호운용성 등의 기술적 특징으로 정의한다. 유비쿼터스 미들웨어 기술은 상황인식(Context-Awareness) 처리 기술, 센서 네트워크의 동적 라우팅 기술, 코드 이동성을 지원하는 통신 플랫폼 및 스마트 메시지 기술로 구분된다.

1. 상황인식(Context-Awareness) 처리 기술

응용 프로그램이 사용되는 위치, 주변의 객체, 그리고 객체의 시간에 따른 변화에 적응적(adaptive)으로 서비스를 제공하는 기술이다. 사용자 환경에 대한 탐지(detection), 센싱을 통해 컴퓨터 자원(resource) 및 서비스를 최대한으로 활용한다. 상황 인식 처리를 위한 필수 요소로는 현재 Context 탐지(detection)기능, 객체의 Context 표현 및 저장(storage) 기술, 가장 관련성이 높은 Context 선택 및 검색 기능, Context에 기반한 서비스의 자동적 수행 기능 등이 있다.

1) 자원 발견(Resource discovery)

○ 서비스는 자원을 광고/등록

○ 원격 객체가 위치한 장소가 아닌 서비스에 의한 자원 발견

○ Intentional Naming System(INS)

2) 위치 발견(Location discovery)

○ 위치 기반 상황인식 기술

○ 고정형 Beacon

○ 이동형 Beacon

3) 상황인식 기반 자동 설정 기술

2. 센서 네트워크의 동적 라우팅 기술

센서간의 신뢰성 있는 동적 라우팅 알고리즘과 무선 Ad-hoc 네트워크 기술, 상황인식에 따른 상태변수 자동설정 기술하는 기술이다. 센서는 제한된 계산 능력과 자원을 가지고 스스로 이웃 노드와 네트워크를 구성할 수 있는 장치여야 하고, 센서 네트워크는 저전력 사용, 네트워크의 자동 설정(self-organization), 협동적인 시그널 처리, 질의(query)능력이 요구된다.

1) 산재된 센서 노드의 기능

○ 데이터 수집

○ 멀티홉에 의하여 싱크 노드로 데이터를 라우팅

2) Ad-hoc 라우팅 알고리즘

○ Table-Driven 방식

주기적으로 또는 네트워크 토폴로지가 변화할 때 라우팅 정보를 브로드캐스팅함으로 모든 노드가 항상 최신의 라우팅 정보를 유지하는 방식

○ On-Demand-Driven 방식

트래픽이 발생하는 시점에서 경로를 탐색하는 방식으로 테이블 관리 방식이 가지는 제어 메시지 오버헤드 문제를 해결하는 방식

3) 센서 protocol stack

○ 이웃 센서에서 메시지를 받은 후에 리시버의 전원을 내림

○ 센서노드의 이동을 발견 및 등록

○ 언제나 사용자로의 라우팅을 유지

○ 특정 지역에서의 센싱 테스트를 스케줄

3. 코드 이동성을 지원하는 통신 플랫폼 및 스마트 메시지 기술

사용자의 요구에 실시간으로 대응할 수 있는 네트워크와 새로운 기술과 서비스를 즉각 적용할 수 있는 네트워크 기술이다. 이는 네트워크 장비의 기능을 실시간으로 변경할 수 있고, 예측 불가능한 전송 트래픽의 변화에 능동적으로 대응 가능하다.

1) 센서 네트워크에서 효율적인 정보 전송을 위한 표준 통신 플랫폼

2) 메시지 구조, 수신 모듈, 인터프리터 모듈

3) 이동 코드 보호를 위한 안정성 확보 기술

3.4 유비쿼터스 미들웨어 유형

유비쿼터스 미들웨어는 여러 가지 다양한 서비스들과 디바이스 간의 통합·융합을 담당하고 기반이 되는 공통 기능들을 탑재하고 있다. 유비쿼터스 환경에서 적용되는 미들웨어들의 유형은 다음과 같다.

첫째, 다양한 복수 이기종 센서 처리를 위한 USN 미들웨어를 들 수 있다. USN 미들웨어는 다양한 이기종 USN H/W, 이기종 센서 네트워크, 그리고 다양한 응용서비스들 간의 독립성을 보장하면서도 이에 대한 유연한 통합을 지원한다. 그 중에 가장 핵심이 되는 기능은 응용 서비스로부터 주어지는 다양한 질의들에 대한 응답을 신속히 제공하기 위하여 이기종의 센서 노드들로부터 센싱 정보를 수집하고 이를 가공하는 것이라 할 수 있다.

둘째, 다양한 제조사들의 RFID 리더로부터 수집된 RFID 태그정보를 응용

시스템이 처리 가능한 방식으로 전달해주는 RFID 미들웨어이다. RFID로부터 발생하는 대규모 데이터 처리 및 시스템 부하, 다수의 하드웨어들을 원격으로 제어하기 위한 컴퓨팅 기술, 분산 환경에서는 데이터 무결성을 보장한다.

셋째, 물리적인 공간과 소프트웨어 인프라가 함께 자연스럽게 융화되는 액티브 공간을 보다 용이하게 구현할 수 있는 프레임워크를 제공해주는 상황인식 미들웨어이다. 접속 가능한 망의 상태인지, 사용자의 동작형태를 인지, 그리고 환경인식에 대해 지능적 판단하는 기술로 서비스 상황에 따라 재구성 가능한 객체기반의 상황인식 기술을 제공한다.

이외에, 지도정보 및 위치정보 등을 표출하기 위한 GIS 엔진과의 연계를 위한 GIS 미들웨어와 휴대폰, PDA, UMPC 등 다양한 모바일 디바이스간의 통합 연계 처리를 위한 u-Device Integration 미들웨어, 홈네트워크와의 연동을 위한 홈게이트웨이 미들웨어 등이 있다.

제 4 장 유비쿼터스 컴퓨팅 환경의 통합관제 미들웨어 설계

4.1 요구사항 분석

유비쿼터스 컴퓨팅 환경에서의 통합 관제를 위한 실시간 데이터 처리 미들웨어를 설계하기 위해서는 미들웨어가 가져야 할 기능에 대한 분석이 먼저 이루어져야 한다. 따라서, 본 절에서는 유비쿼터스 환경의 통합관제를 위한 미들웨어가 지니고 있어야 할 기능에는 어떤 것들이 있는지 살펴보고, 그러한 기능들을 만족시키는 미들웨어의 모듈들을 정의한다.

먼저, 실시간 데이터 처리 미들웨어는 통합 관제 환경에서 이기종 어플리케이션 장치 간 통합 인터페이스를 지원하는 Connector 기능이 있어야 한다. 도시에서 다양한 센서로 부터 올라오는 정보 데이터를 수집하는 장치들은 운영환경 및 프로그래밍언어 그리고 통신방식이 다르다. 따라서, 여러 장치들간의 통합된 인터페이스를 위해 하나의 채널을 구성해야 한다. 이를 위해 다양한 애플리케이션 서버의 OS(Windows, Linux, Unix 등)와 프로그래밍언어(C++, VB, PB, Delphi, Java 등)에 적용할 수 있는 다양한 환경의 Connector 기능이 요구된다. 또한, 통합관제를 위한 미들웨어는 실시간으로 데이터를 처리해야 하므로, 오류 또는 장애에 의해 일부 모듈이 정지하거나, DBMS가 Down되어도 미들웨어 Core는 정상적으로 동작되어야 한다. 그리고, 대용량 데이터의 멀티캐스팅을 위한 프로세스 기능과 멀티캐스트를 통해 다양한 데이터의 송수신이 동시에 이루어져야 한다. 즉 여러 종류의 데이터를 공유메모리를 사용하여 동시에 송수신 할 수 있어야 한다. 그리고, 복잡하고 다양하게 발생하는 이벤트에 대한 우선순위를 결정하여 긴급한 것부터 순차적으로 처리해야 한다. 이

외에도 보안을 위한 기능과 신규 서비스 로직이 유연하게 추가될 수 있는 기능이 요구된다.

이와 같이 위에서 살펴본 유비쿼터스 환경의 미들웨어를 위한 여러 요구 사항들을 바탕으로 미들웨어의 모듈들을 정의해보면 다음과 같다. 먼저, 다양한 이기종 정보수집서버와의 데이터 송수신 인터페이스를 담당하는 모듈로서 Connector를 정의할 수 있으며, 미들웨어내부에서 외부와 데이터를 송수신하기 위한 External Agent모듈과 내부에서 각 서비스 업무 로직을 수행하기 위한 Internal Agent모듈을 정의할 수 있으며, 대용량 데이터를 효율적으로 처리하기 위한 멀티 프로세스 처리 모듈을 정의할 수 있으며, 데이터 및 이벤트의 리소스를 관리하기 위한 모듈을 정의할 수 있으며, 메시지 관리를 위한 모듈을 정의할 수 있으며, 이벤트간의 경합을 위한 모듈을 정의할 수 있으며, 보안을 위한 모듈을 정의할 수 있다. 이렇게 정의된 모듈들을 포함하고 있는 실시간 데이터 처리 미들웨어는 다음의 그림 4.1에서 보는 바와 같다.

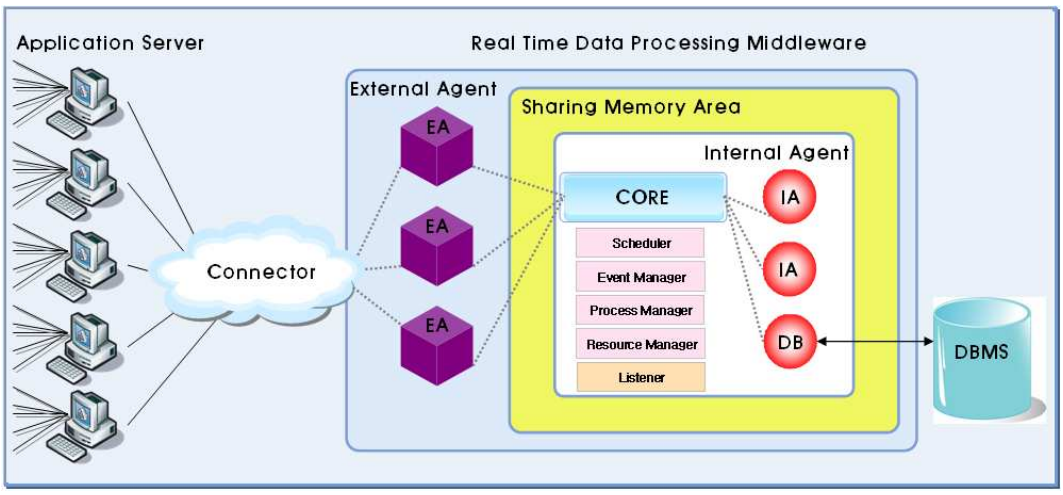


그림 4.1 미들웨어 디자인

4.2 미들웨어 구조 설계

위의 4.1절에서는 미들웨어가 가지고 있어야 할 기본적인 기능들에 관해 분석하였고, 분석된 결과를 바탕으로 미들웨어의 모듈들을 결정하였다. 이번 절에서는 이러한 모듈들을 포함하고 있는 미들웨어의 전체적인 구조를 제안하고, 그것에 대해 자세한 설명을 덧 붙이도록 하겠다.

4.2.1 미들웨어의 전체 구조

본 논문에서 제시한 미들웨어는 유비쿼터스 환경의 통합관제에서 이기종 어플리케이션 장치 간 통합 인터페이스를 지원하고, 관제시 발생하는 대규모 데이터 및 이벤트에 대한 실시간 처리를 수행할 수 있도록 Core를 중심으로 Application Connector Layer, External Layer, Internal Layer 3가지 계층으로 구성된다.

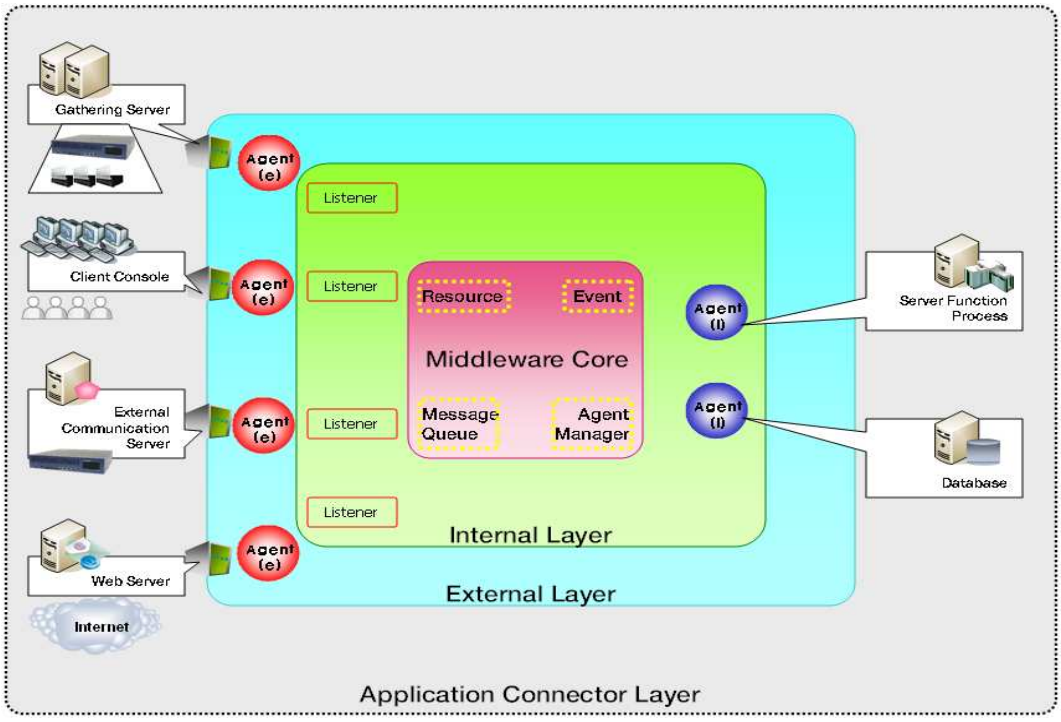


그림 4.2 미들웨어 계층 구조도

Application Connector Layer는 일반적인 응용프로그램을 구현하는 환경을 지원한다. 시스템과의 종속성을 배제한 자유로운 구현과 응용을 보장하면서 응용프로그램에 따라 간단한 절차만으로 미들웨어와의 연계를 보장받을 수 있다.

External Layer는 외부의 요구를 받아서 수행하고 외부와의 통신 안정성과 업무를 명시할 수 있는 작업 Agent 기능을 수행한다. Connector를 통한 외부와의 인터페이스를 보장하며, Listener를 통하여 내부 계층으로의 연계를 수행하는 외부연계 Agent이다.

Internal Layer는 미들웨어 Core를 중심으로 내부의 핵심 모듈로써 Agent의 생성과 소멸 등의 처리가 이루어진다. 또한, 미들웨어 내·외부의 인터페이스를 위한 핵심적인 역할을 수행한다. 구성은 Listener와 Internal Agent와 특수한 목적으로 수행되는 HSA(Half System Agent)와 시스템의 특성을 결정짓는 SA(System Agent)로 구성된다.

미들웨어 Core는 자원관리, 이벤트관리, 메시지관리, 프로세스관리, Agent 관리 등 미들웨어내 모든 Agent의 생성부터 소멸까지의 라이프사이클을 관리하며 시스템의 모든 모듈간의 통신 및 자원분배 등 전체적인 기능을 유지관리한다.

실시간 데이터 처리 미들웨어는 다음과 같은 내용으로 구성되며, 전체 구성도는 그림 4.3에서 보는바와 같다.

- 모든 Application 서버의 OS(Windows, Linux, Unix 등)와 프로그래밍언어(C++, VB, PB, Delphi, Java 등)에 적용할 수 있는 다양한 환경의 Connector 기능
- 대규모 데이터 요청에 대한 자동 부하조절 기능
- 리소스 관리를 통한 실시간 데이터 공유 기능
- 병렬 메시징 구조의 멀티 프로세스 구현으로 대량의 트랜잭션 처리 기능

- 스케줄링, Broadcast, 다중세션관리, 세션간 Load Balancing 기능
- 관제 업무 처리 서비스간의 연동 로직과 입출력 전문처리를 프로그램에서 분리
- Rule & Parameter 기반으로 미들웨어에서 서비스 로직을 구현할 수 있도록 지원
- 프로세스 상태에 대해 실시간으로 모니터링 할 수 있는 기능
- 환경설정파일 및 Debug, 모듈 Maker 지원
- 서비스 모듈에 대해 손쉽게 테스트할 수 있는 시뮬레이션 기능

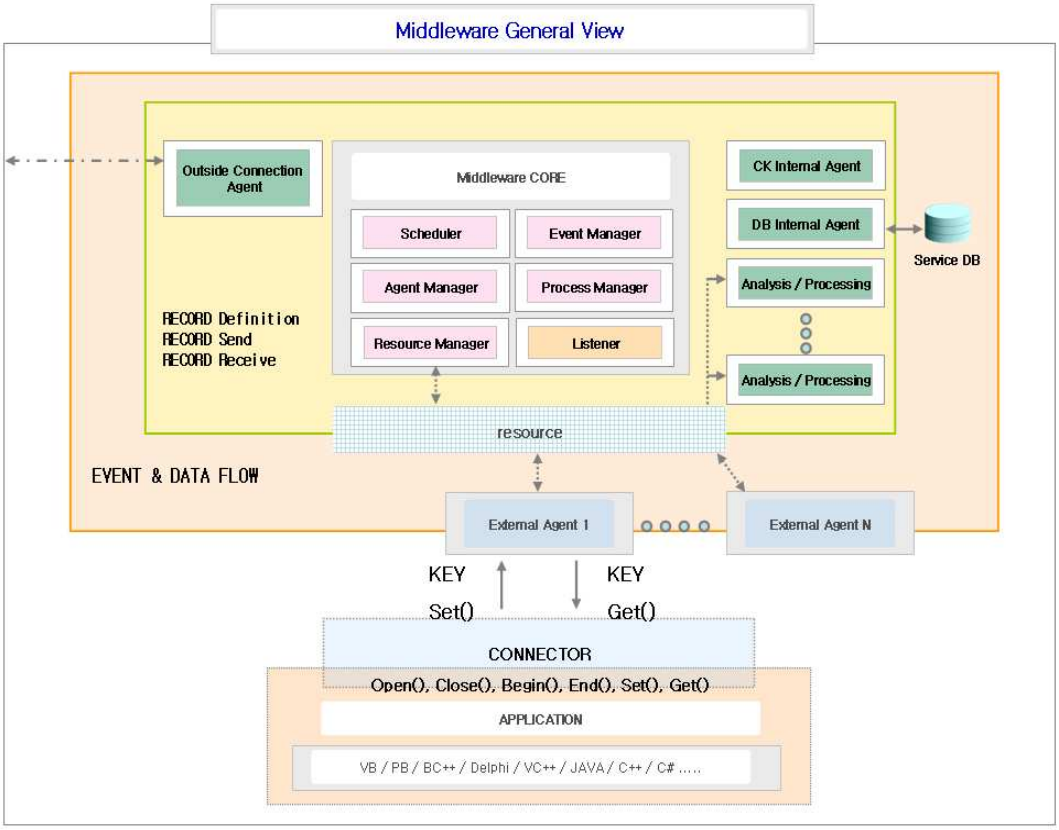


그림 4.3 미들웨어 전체 구조도

4.2.2 Connector

미들웨어와 이기종 어플리케이션 사이의 통신을 위해 어플리케이션의 OS 및 프로그래밍언어 종류에 상관없이 적용될 수 있으며, 기 구현된 어플리케이션들의 통합이 용이하도록 표준화된 인터페이스를 지원한다. 또한, 표준화된 통신방식을 위해 표준함수를 제공하고, 통신상의 오류제어와 흐름제어를 지원하여 보안유지 및 통신안정성을 확보할 수 있다. Connector 구성은 다음 그림 4.4에서 보는 바와 같다.

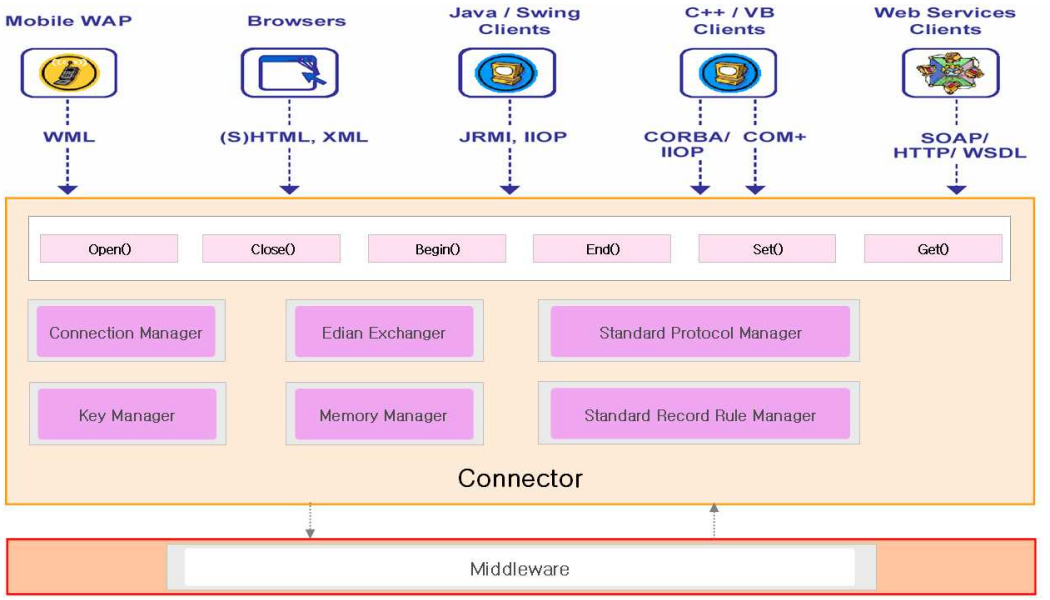


그림 4.4 Connector 구성도

4.2.3 External Agent

어플리케이션 서버와 미들웨어 사이에서 이벤트와 데이터를 연동하는 역할을 수행하며, 이벤트와 데이터를 분석하여 어플리케이션 서버로 리턴하거나 미들웨어 내부로 전송한다. 또한, 어플리케이션 서버의 상태를 체크하여 미들웨어 Core에서 어플리케이션 서버를 관리할 수 있는 기능을 지원하고, 어플리

케이션 서버와의 연결이 끊기면 자동적으로 소멸된다. External Agent 기능 흐름도는 다음 그림 4.5에서 보는 바와 같다.

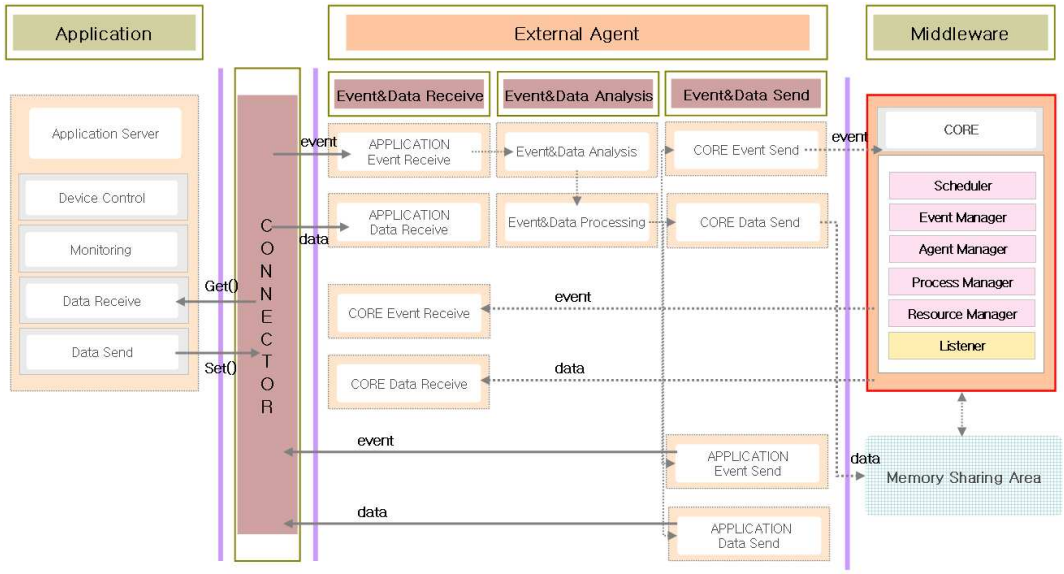


그림 4.5 External Agent 기능 흐름도

4.2.4 Internal Agent

하나의 서비스 로직을 수행할 수 있는 모듈이다. 각각의 Internal Agent는 미들웨어 Core로부터 독립적으로 실행되며, Core로부터 이벤트를 받고, 해당 이벤트를 분석하여 처리한다. Internal Agent는 모든 데이터를 공유메모리에서 읽어서 처리한다. Internal Agent는 비즈니스 서비스 로직을 담당하는 서비스 Agent와 내부 시스템 Clock을 담당하는 CK Agent와 DB와의 연동을 위한 DB Agent와 외부의 다른 미들웨어와의 연계를 위한 외부연계 Agent로 구성된다. 비즈니스 서비스 로직은 업무 목적과 특성에 따라 유연하게 확장할 수 있다. 미들웨어 내부에서 실질적으로 데이터와 이벤트를 분석, 판단, 처리하는 역할을 한다. Internal Agent 기능 흐름도는 다음 그림 4.6에서 보는 바와 같다.

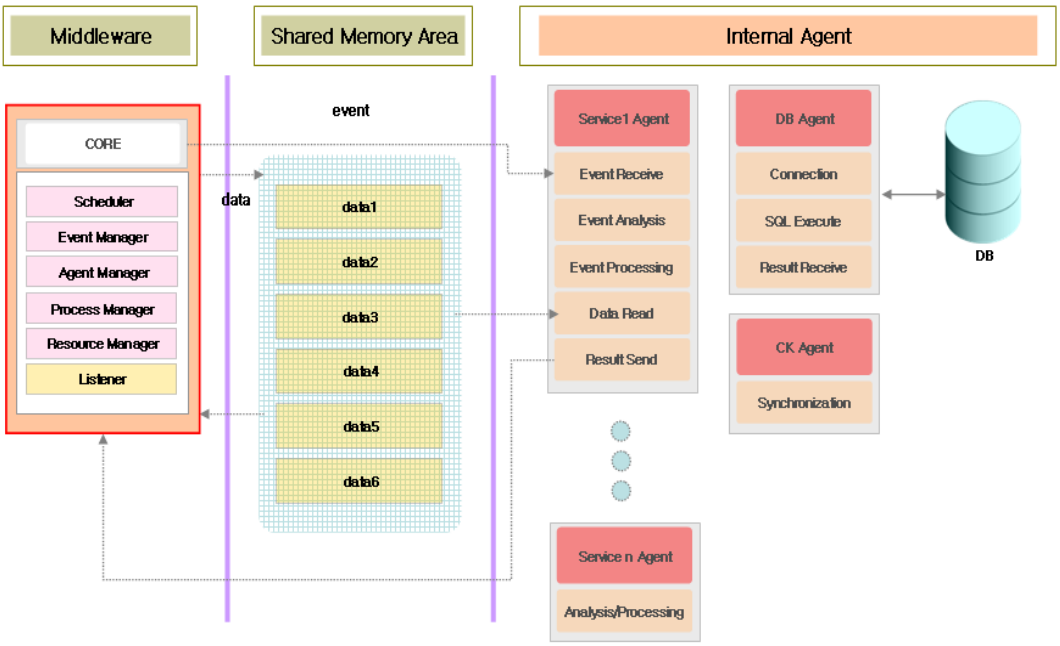


그림 4.6 Internal Agent 기능 흐름도

4.2.5 Listener

어플리케이션 서버에서 미들웨어와 연결할 때 최초로 만나는 모듈로써, 어플리케이션 서버로부터 접속요청을 받는다. Listener는 현재 접속하고자 하는 어플리케이션 서버가 인증된 것인지에 대한 확인절차를 수행하고, 이미 External Agent가 실행되어 연결되었는지 확인하여, 이미 연결되어 있거나, 인증되지 않은 것이라면 연결을 차단한다. 즉, 어플리케이션 서버가 인증된 것이고, 연결되어 있지 않다면 External Agent를 실행시키고 연결하여, 미들웨어와의 연계를 할 수 있도록 지원한다. Listener 기능 흐름도는 다음 그림 4.7에서 보는 바와 같다.

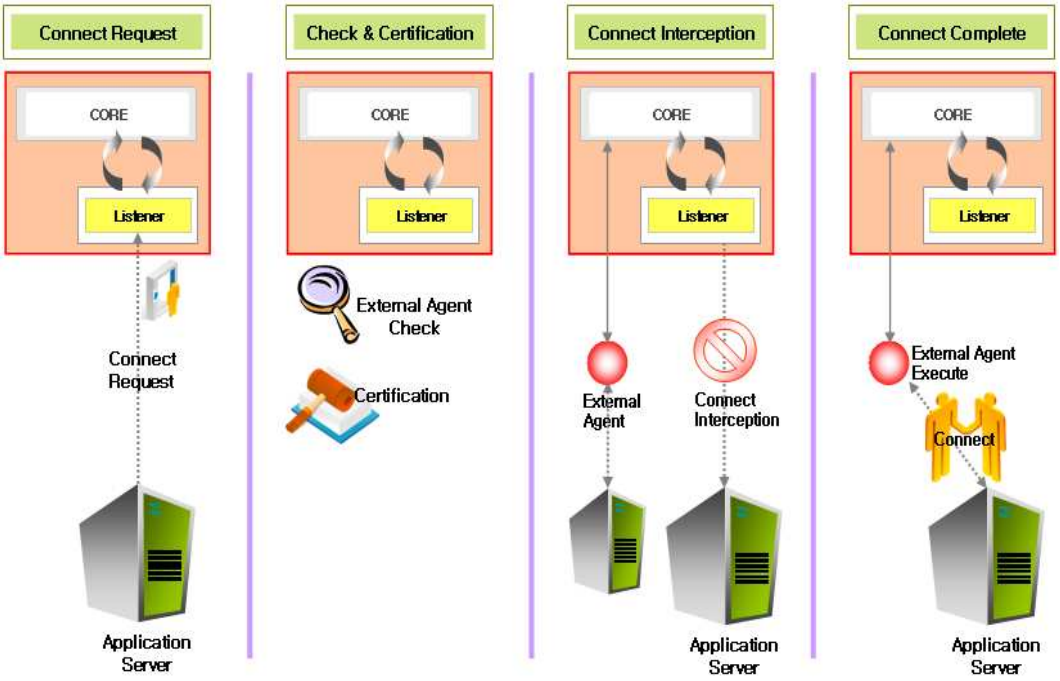


그림 4.7 Listener 기능 흐름도

4.2.6 Core

미들웨어 Core는 미들웨어의 핵심 모듈로써, 미들웨어 내부의 모든 이벤트, 메시지, 데이터, 메모리 자원을 총체적으로 제어 관리하며, Agent의 생성, 생성 후 초기화, 서비스 실행, 소멸에 관한 모든 권한을 가지면서 Agent의 생명주기를 관리한다. 또한 생성된 Agent간에 연동을 주관하고, 모든 프로세스를 관리한다. 미들웨어 Core는 특정 상황에 대한 이벤트를 분석 처리하여 그 결과 지시에 대한 이벤트를 내부의 모든 Agent에게 동시에 전파하고, 미들웨어 내부의 데이터는 공유메모리영역을 통하여 Core와 모든 Agent가 동시에 사용하여, 특정 이벤트가 발생했을 경우, 해당 이벤트와 관련된 데이터를 동시에 읽는다. 미들웨어 Core의 기능 흐름도는 다음 그림 4.8에서 보는 바와 같다.

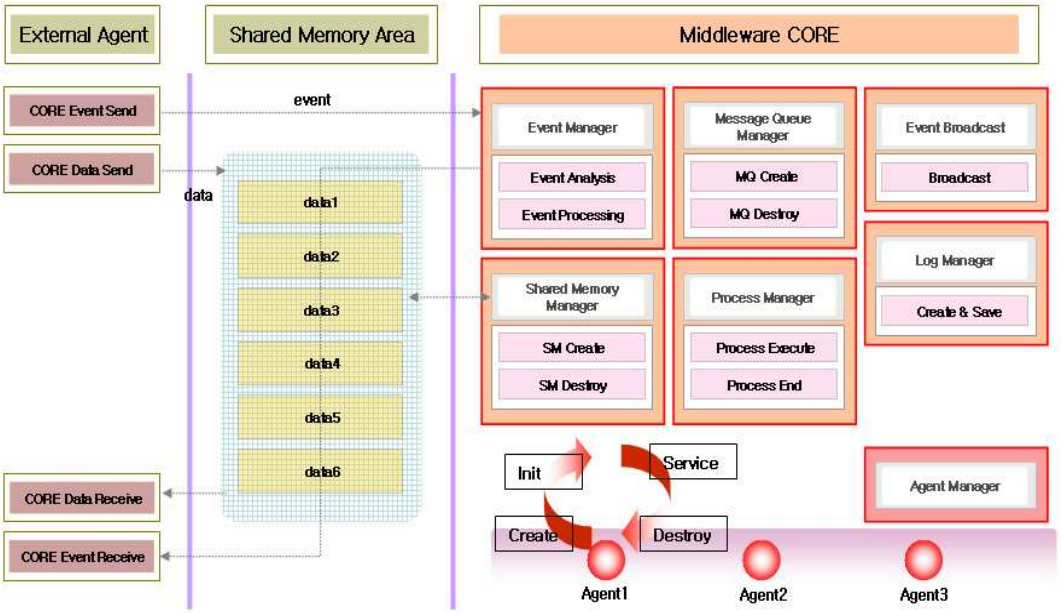


그림 4.8 Core 기능 흐름도

1) 멀티 프로세스 관리 기능

Core 내 모든 프로세스에 대한 관리를 통해 성능을 극대화 한다.

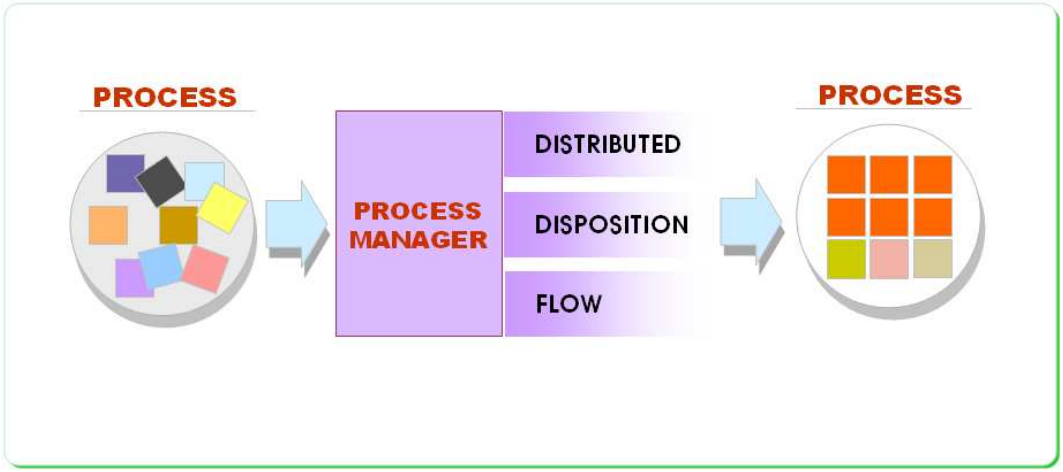


그림 4.9 프로세스 관리 개념도

2) 리소스 관리 기능

미들웨어 내 모든 데이터는 공유 메모리 영역을 통하여 관리하며, 내부의 모든 프로세스가 동시에 데이터를 읽는다.

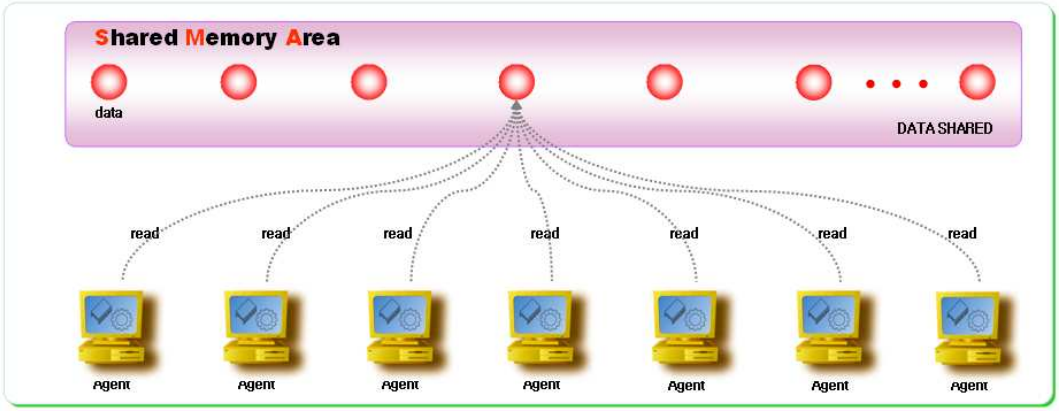


그림 4.10 리소스 관리 개념도

3) 메시지 관리 기능

미들웨어는 메시지를 큐라고 불리는 전달 중계소에 넣어 처리하고 큐에 의한 메시지 관리 기능을 제공한다.

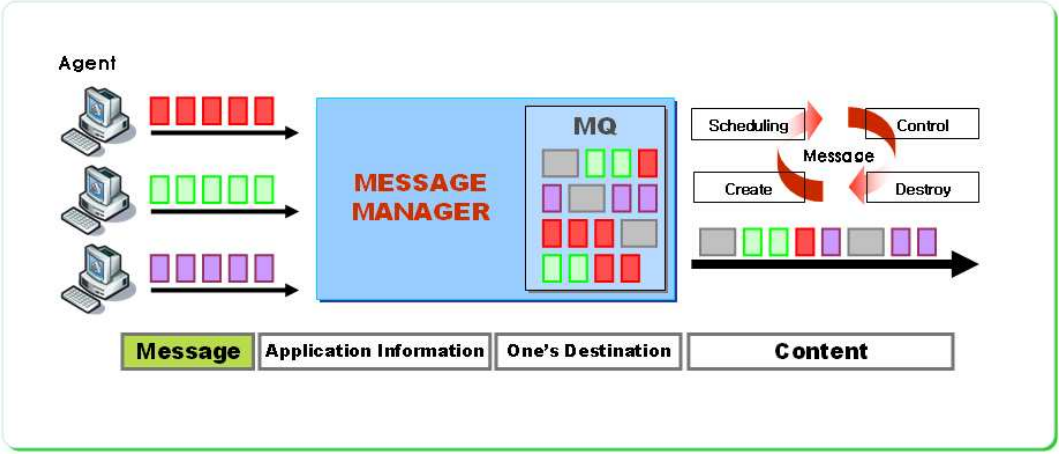


그림 4.11 메시지 관리 개념도

4) 동시 전파 기능

미들웨어는 이벤트에 대한 메시지를 모든 Agent에게 동시에 전파한다.

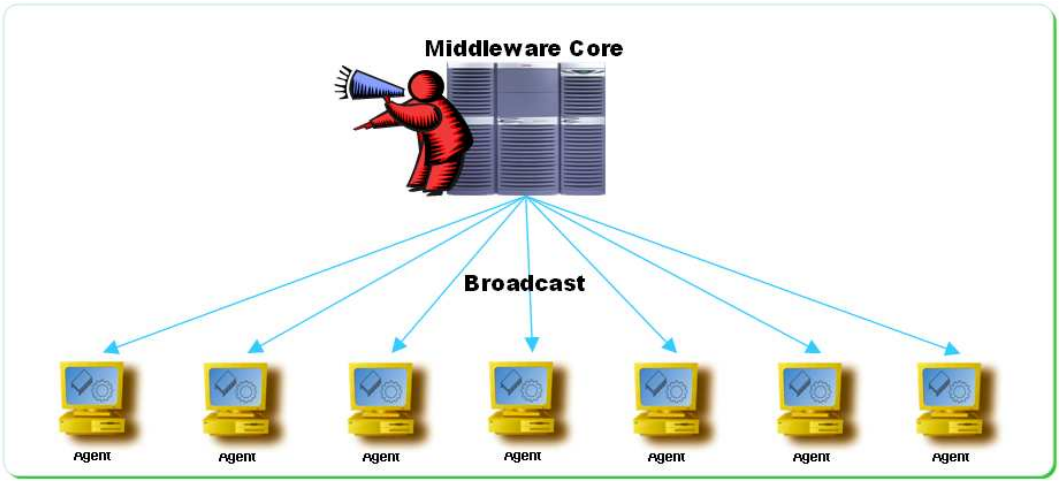


그림 4.12 동시 전파 개념도

4.2.7 Fault Tolerance

실시간 데이터 처리 미들웨어는 Core와 내부 모듈이 완전히 독립적으로 실행되도록 구성한다. 관제를 위한 미들웨어는 실시간으로 이벤트와 데이터를 처리해야 하므로 오류 또는 장애에 의해 일부 모듈이 정지하더라도 미들웨어 Core는 정상적으로 동작되어야 한다. 또한, DBMS에 비종속적으로 구현하여, DBMS가 다운되어도 미들웨어는 정상적으로 동작하며, DB에 저장될 데이터는 임시 저장소에 저장했다가 DB가 정상화되면 임시로 저장했던 데이터를 DB로 이동해야 한다. 이는 미들웨어의 장애 대책을 위한 기능으로 그림 4.13에 개념도를 보여준다.

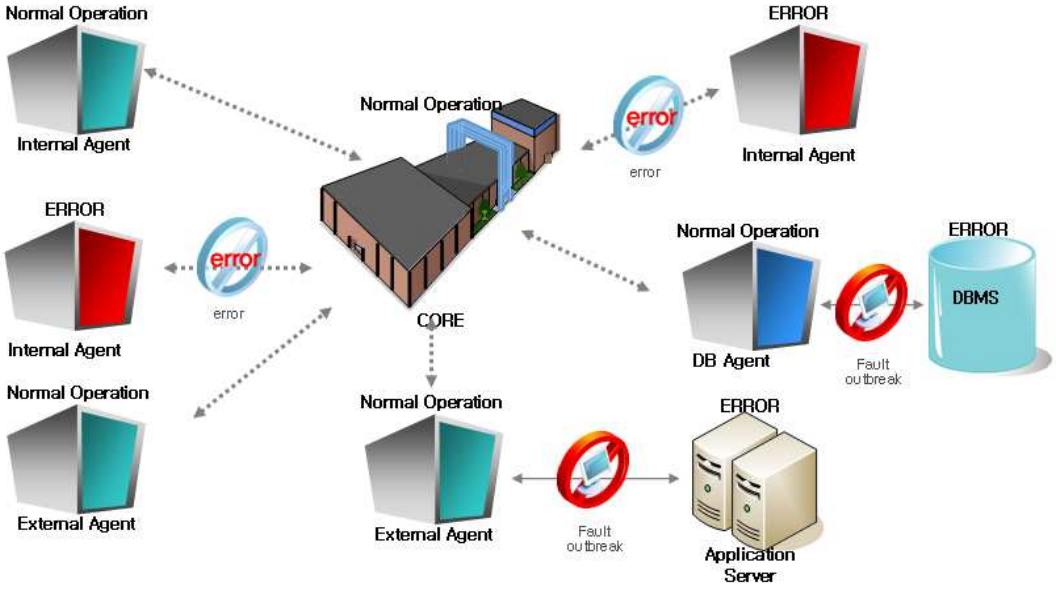


그림 4.13 Fault Tolerance

4.3 이벤트 경합 처리를 위한 신경회로망 모델링

본 절에서는 기존의 이벤트 경합 처리 기술의 현황 및 문제점을 분석하고, 이를 개선하기 위한 방안으로 역전파 신경망을 이용한 이벤트 경합 처리 알고리즘을 제안하였다.

4.3.1 이벤트 우선순위 기준 설정

우리나라의 경우 <자연재해대책법>에서는 자연재해가 대응단계에서의 불가항력적 결과물이라는 측면에서 “재해”라는 용어를, <재난관리법>에서는 인위재난 대응단계에서의 통제가능성에 초점을 맞추어 연속적 진행개념인 “재난”이라는 용어를 사용하고 있다.[17]

재난 관련법상 용어의 정의를 살펴보면 <자연재해대책법> 제2조에서는 재해를 “태풍, 홍수, 호우, 폭풍, 해일, 폭설, 가뭄 또는 지진, 기타 이에 준하는

자연현상으로 인하여 발생하는 피해”로 규정하고 있어 물리적 관점에서 재난을 정의하고 있으며, <재난관리법> 제2조에서는 재난을 “화재, 붕괴, 폭발, 교통사고, 화생방사고, 환경오염사고 등 국민의 생명과 재산에 피해를 주는 사고로서 자연재해가 아닌 것”으로 정의하고 있다. <재난및안전관리기본법> 제3조에서는 재난을 “국민의 생명, 신체 및 재산과 국가에 피해를 주거나 줄 수 있는 것”으로 정의함으로써 “태풍, 홍수, 호우, 폭풍, 해일, 폭설, 가뭄, 지진, 황사, 적조 그밖에 이에 준하는 자연현상으로 인하여 발생하는 재해”, “화재, 붕괴, 폭발, 교통사고, 화생방사고, 환경오영사고 그밖에 이와 유사한 사고로 인한 피해”, “에너지, 통신, 교통, 금융, 의료, 수도 등 국가기반체계의 마비와 전염병 확산 등으로 인한 피해”로 확대하여 재난을 규정하고 있다.[17]

우리나라에서는 소방방재청을 통해 과학적이고 체계적인 국가재난관리체계를 구성하여, 건설교통 관련 시설·수단의 대형화, 복잡화, 고속화 등으로 사고 발생시 많은 인명과 재산피해가 수반됨을 감안 철도, 항공, 지하철, 도로, 댐, 지하공동구, 건설현장 등 분야별 재난대책조직, 사고보고체계, 긴급구조·구급체계와 사고수습복구체계, 사고유형별 대응매뉴얼 등의 재난의 예방·대비·대응·복구체계를 마련, 재난 대비 대응에 쉽게 활용할 수 있도록 하여 인적·물적피해를 최소화하고 있다.

일반적으로 통합관제 시 경합처리를 위한 체계는 데이터베이스에 국가 재난/재해 관리법에 따라 미리 우선순위 정보를 저장하고 정보 수집 인터페이스에서 경합이 일어났을 때 입력된 자료를 근거로 비교 자료로 쓰인다. 또한 도로교통 정보와 일반 정보도 미리 저장하여 우선순위에 의해 어떤 메시지를 보낼지 결정한다. 본 논문에서의 이벤트 우선순위 기준은 국가 재난관리법과 건설교통부 지침에 따라 안정성, 사용자 필요성, 사건 횟수, 피해 중요도에 따라 상대적인 우선권을 주었다. 우선순위는 재난, 재해 긴급정보, 도로 교통정보, 일반 정보의 순서로 데이터베이스에 미리 저장되어, 경합 처리의 기준 데이터

로 사용된다.

본 논문은 통합관제 환경에서 발생하는 상황정보에 대한 우선순위 정의를 위해 국가재난관리법[13]과 건설교통부 지침[14][15] 및 u-City협회[16]의 자료 등을 참조하여 다음 표 4.1 ~ 표 4.4에 정의하였다.

표 4.1 재난, 재해 긴급 정보

우선순위	본 논문	
1 순위	방재정보	대형 화재
2 순위	기상정보	태풍, 폭우, 폭설, 하천 범람, 저수지 범람
3 순위	지하시설물정보	가스 누출
4 순위	공공시설물정보	교량 붕괴, 건물붕괴
5 순위	대기정보	황사, 오존, 자외선, 스모그, 대기오염
6 순위	환경정보	방사능 유출, 위험물 유출

표 4.2 도로 교통 정보

우선순위	건설 교통부 지침		본 논문	
1 순위	돌발상황 발생	교통사고, 공사 구간 기타 돌발상황	교통상태정보	구간속도, 소통상태, 통행시간
2 순위	상시적인 정체	정체구간의 교통 상황 통행시간 우회도로 정보	유고정보	발생지점, 발생내용 지체정도, 공사구간 행사구간, 교통사고
3 순위	소통원활	정체구간의 교통 상황 통행시간	우회정보	우회경로
4 순위			도로기상정보	기상상태(눈, 비, 폭설)
5 순위			도로정보	차로 / 길 어깨, 노면상태
6 순위			동영상정보	차량움직임

표 4.3 일반 정보

우선순위	본 논문	
1 순위	방재정보	화재
2 순위	방범정보	침입
3 순위	지하시설물정보	유량, 유압량, 계측, 열온도
4 순위	공공시설물정보	침하, 균열, 진동, 기울기,
5 순위	기상정보	온도, 습도, 풍속, 풍향, 강우/적설량
6 순위	대기정보	시정, 황사
7 순위	영상 정보	각종 시설물 확인, 불법쓰레기 투척
8 순위	관공서정보	지역공지

표 4.4 등급별 정보

정보	상세 정보	등급
방재 정보	대형화재	1등급
	중형화재	2등급
	소형화재	3등급
기상정보	태풍, 폭설	1등급
	온도, 풍향, 풍속, 강우, 적설, 습도	2등급
지하시설물정보	열 감지	1등급
	유압, 유량, 계측	2등급
공공시설물	침하, 균열	1등급
	기울기, 진동	2등급
대기정보	황사, 오존, 자외선, 스모그	1등급
	미세먼지, 각종 오염물질	2등급
영상정보	화재, 침입, 도로, 하천	1등급
	쓰레기불법투기, 각종시설물	2등급

본 논문에서는 통합관제 환경에서의 상황 발생에 따른 이벤트 우선순위 분류 실험을 위해 위와 같은 자료를 토대로 통합관제에서 발생하는 이벤트 정보를 표 4.5와 같이 코드로 정의하였다.

표 4.5 상황 코드 정의

항목	코드	설명
서비스 코드	1	방범/방재 서비스
	2	도로/교통 서비스
	3	환경 감시 서비스
	4	공공 시설 서비스
이벤트 코드	1	침수 발생
	2	누전 발생
	3	미세먼지경보 발생
	4	돌발상황 발생
	5	비상도움 발생
	6	화재 발생
내용 코드	1	대형 백화점, 고층 빌딩, 산불
	2	사무실, 공장, 교육, 위생 시설
	3	주거시설
	4	반경10m이상
	5	반경5~10m
	6	반경1m이하
	7	미세먼지 100$\mu g/m^3$ 이하
	8	미세먼지 101$\mu g/m^3$ ~ 150$\mu g/m^3$
	9	미세먼지 151$\mu g/m^3$ ~ 200$\mu g/m^3$
	10	미세먼지 201$\mu g/m^3$ ~ 400$\mu g/m^3$
	11	미세먼지 401$\mu g/m^3$ ~ 800$\mu g/m^3$
	12	미세먼지 800$\mu g/m^3$ 이상
	13	오존 농도 0.12ppm/h 이상
	14	오존 농도 0.3ppm/h 이상
	15	오존 농도 0.5ppm/h 이상
	16	경상자 1명
	17	경상자 2명이상
	18	중상자 1명
	19	중상자 2명이상
	20	사망
	21	붕괴
	22	유독성 가스
	23	위협
	24	침입
	25	시위

본 논문에서는 신경회로망의 상황별 이벤트 우선순위 분류 테스트를 위해 임시로 상황별 코드체계를 구성하고, 우선순위를 표 4.6의 예시와 같이 정하였다.

표 4.6 상황 코드 체계 구성 예

항목	순위	상황코드	설명
상황별 이벤트 우선순위	1	1-6-1	방범/방재서비스에서 대규모 화재(대형백화점, 고층빌딩, 산불)발생
	2	4-1-4	공공시설서비스에서 대규모 침수(반경 10m 이상)발생
	3	3-3-12	환경감시서비스에서 매우 강한 황사(미세먼지 농도 800㎍/㎥ ~)발생
	4	3-3-15	환경감시서비스에서 매우 강한 오존(오존의 농도 0.5ppm/h 이상)발생
	5	2-6-20	도로/교통서비스에서 대형 교통 사고(차량파손 3대이상 중상자 2명이상 사망)발생
	6	4-4-21	공공시설서비스에서 건물붕괴
	7	3-3-22	환경감시서비스에서 유독성 가스 누출
	8	1-6-2	방범/방재서비스에서 중간규모 화재(사무실, 공장, 교육, 위생 시설)발생
	9	4-1-5	공공시설서비스에서 중규모 침수(반경5~10m)발생
	10	3-3-11	환경감시서비스에서 강한 황사(미세먼지 농도 400~800㎍/㎥)발생
	11	3-3-14	환경감시서비스에서 강한 오존(오존의 농도 0.3ppm/h 이상)발생
	12	2-6-19	도로/교통서비스에서 중형 교통 사고(차량파손 3대이상 중상자 2명이상)발생
	13	1-6-3	방범/방재서비스에서 소규모 화재(주거 시설)발생
	14	4-1-6	공공시설서비스에서 소규모 침수(반경1m이하)발생
	15	3-3-10	환경감시서비스에서 매우 나쁜 미세먼지(농

			도 201㎍/㎥ ~ 400㎍/㎥)발생
	16	3-3-13	환경감시서비스에서 약한 오존(오존의 농도 0.12ppm/h 이상)발생
	17	2-6-18	도로/교통서비스에서 소형 교통 사고(중상자 1명이상) 발생
	18	1-5-23	방범/방재서비스에서 위협(긴급요청)발생
	19	3-3-9	환경감시서비스에서 약간 나쁜 미세먼지(농도 151 ~ 200㎍/㎥)발생
	20	2-6-17	도로/교통서비스에서 소형 교통 사고(경상자 2명이상) 발생
	21	1-5-24	방범/방재서비스에서 침입에 의한 도움요청 발생
	22	3-3-8	환경감시서비스에서 나쁜 미세먼지(농도 101 ~ 150㎍/㎥)발생
	23	2-4-16	도로/교통서비스에서 소형 교통 사고(접촉사고, 경상자 1명이하)발생
	24	3-3-7	환경감시서비스에서 약간 나쁜 미세먼지(농도 100㎍/㎥ 이하)발생
	25	2-4-25	도로/교통서비스에서 집회 및 시위발생

4.3.2 룰-데이터베이스 기반 우선순위 알고리즘

데이터베이스에 이벤트에 대한 우선순위 정보를 저장하고, 경합이 발생하였을 때 데이터베이스의 자료를 근거로 비교하여 우선순위를 결정하는 방법으로 두 개 이상의 프로세스가 경합할 때 프로세스에 우선순위를 부여하고, 우선순위가 큰 프로세스에게 실행 권한을 주는 알고리즘이다.

우선순위 부여를 위해 Ready Queue에 순차적으로 들어간 프로세스는 다음 두 가지 방법으로 무한정비(Blocking) 또는 기아 상태(Starvation)를 해결한다. 첫째, 메시지 구성안에 따라 정보들을 우선순위로 배정한다. 둘째, 우선순위 값이 낮을수록 높은 우선순위를 배정 받는다.

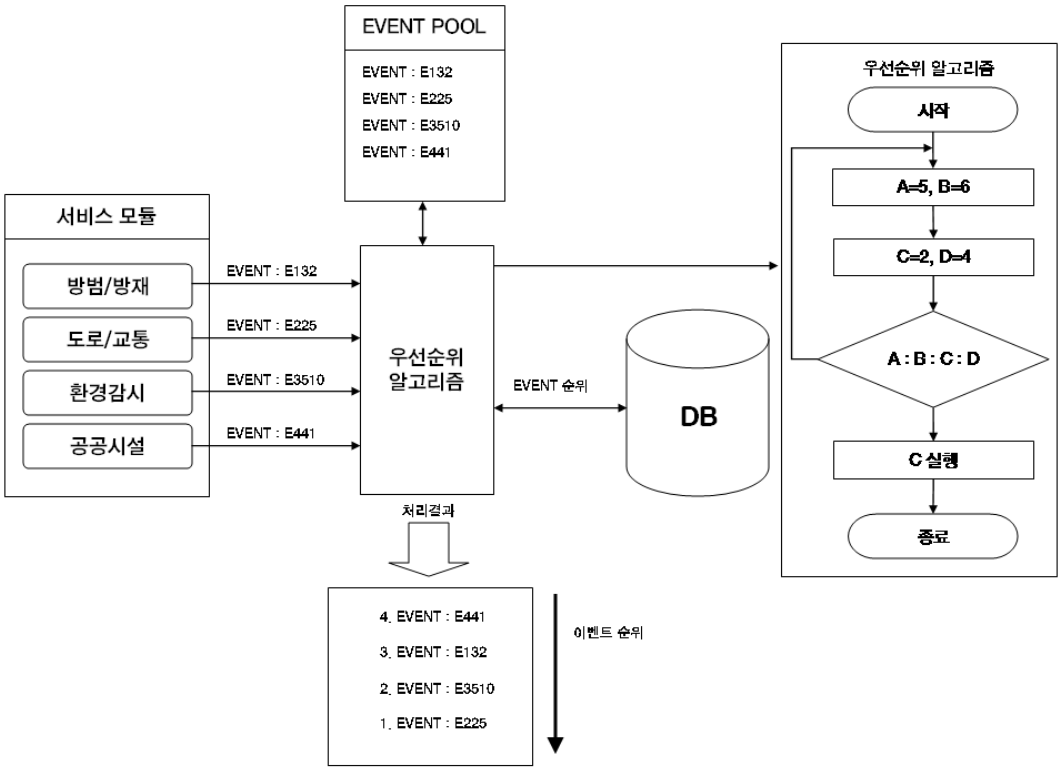


그림 4.14 이벤트 우선순위 경합처리 구조

4.3.3 룰-데이터베이스 기반 우선순위 알고리즘의 문제점

일반적으로 사용중인 룰-데이터베이스 기반 우선순위 알고리즘은 하나의 이벤트에 하나의 조건을 정의하여 룰 테이블을 구성하고, 룰 테이블 데이터를 데이터베이스에 저장하여, 저장된 데이터를 기준으로 분류를 수행한다. 이러한 방식의 우선순위 알고리즘은 실시간 이벤트 처리에 있어서 다음과 같은 문제점을 가지고 있다.

첫째, 많은 이벤트가 동시에 발생했을 경우의 부하문제이다. 우선순위 알고리즘은 입력받은 이벤트의 우선순위 정보를 얻기 위하여 데이터베이스를 조회하여, 조회한 정보의 우선순위를 비교 처리하는 구조로 되어 있다. 이는 데이

터베이스에 대해 많은 트랜잭션을 발생시켜서 부하를 가중시킨다.

둘째, 데이터베이스에 저장되어 있지 않은 이벤트 처리에 대한 문제이다. 우선순위 알고리즘은 미리 데이터베이스에 입력된 정보에 대한 분류만 수행할 수 있다. 저장되어 있지 않은 이벤트에 대한 경우는 분류를 수행할 수 없다.

셋째, 데이터베이스의 오류가 발생했을 경우의 문제이다. 통합관제 환경에서는 수많은 데이터가 데이터베이스에서 조회, 입력, 수정, 삭제 등의 트랜잭션을 통해 사용되는데, 시스템 과부하에 의해 데이터베이스의 오류가 발생하는 경우가 종종 있다. 이러한 경우에 룰-데이터베이스 기반 우선순위 알고리즘은 분류를 수행할 수 없다.

4.3.4 신경망을 이용한 이벤트 우선순위 분류 알고리즘

본 논문에서는 4.3.3절에서 설명한 기존의 이벤트 우선순위 알고리즘의 한계를 극복하고자 역전파 신경망을 이용한 분류 알고리즘을 제안하였다.

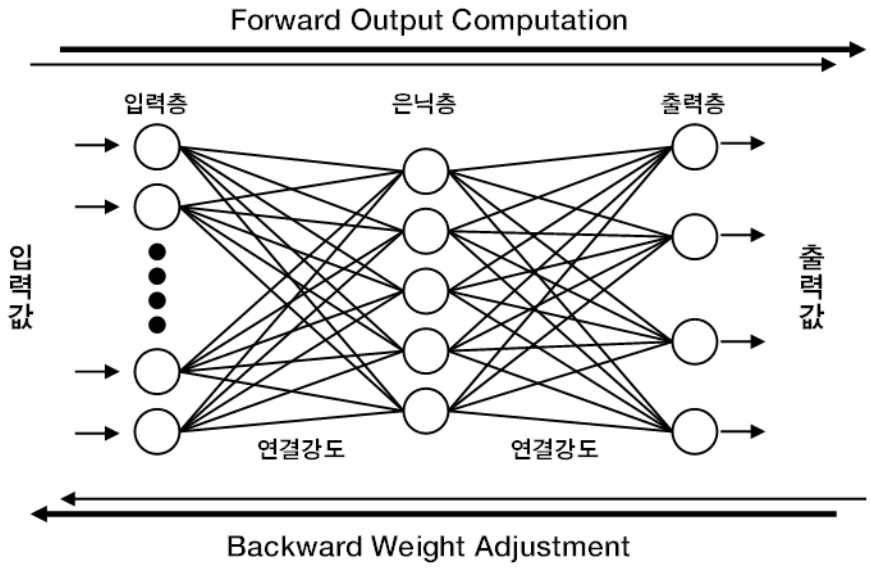


그림 4.15 다층 역전파 신경망 구조

신경망 알고리즘은 경험에 의해 학습하고, 학습을 통해 결과를 도출한다. 결과 도출 과정에서 일반 알고리즘처럼 입력값을 제시하면 파라메터와 변수들과의 출력값이 아닌 신경망 모형안의 유니트, 연결강도와의 상호 작용을 통하여 입력값에 대한 적절한 출력값이 형성된다.

신경망 학습방법으로 추구하고자 하는 값을 제시하는 감독학습 방법과 그렇지 않은 무감독 학습방법이 있다.

본 연구는 통합관제 환경에서 발생하는 이벤트에 대한 우선순위 분류 알고리즘을 개발하기 위해 이벤트에 따른 우선순위 값을 신경망의 목적패턴으로 제시할 수 있는 감독학습 방법이 적절하며, 단층 신경망이 가진 문제점을 고려하여 목적 패턴을 제대로 학습 시킬 수 있는 다층 신경망 모형 적용이 필요하다. 이러한 두 가지 조건을 모두 만족시킬 수 있는 신경망 학습방법으로 다층의 구조와 최급하강법의 원리를 적용하여 출력패턴의 오차를 줄여 목적 패턴이 제시하는 값을 산출할 수 있도록 설계된 역전파 알고리즘(Back propagation Algorithm)을 사용하였다.

본 논문에서의 신경망 모형은 다층 신경망 모형을 이용하여 구현하였다. 신경망의 학습을 위한 입력변수는 표 4.6에서 정의한 상황 코드 체계인 서비스코드, 이벤트코드, 내용코드가 사용되었고, 학습을 위한 목적값은 우선순위가 사용되었다.

다층 신경망모형은 입력자료에 대하여 은닉층 수, 은닉층 유니트 수, 전이함수 형태, 학습반복 횟수, 초기연결강도, 학습계수 등 네트워크 구조를 어떻게 설정하느냐에 따라 달라진다. 은닉층 수와 은닉층에 포함된 유니트 수가 많으면, 신경망 최적화를 위한 수렴시간이 많이 소요되므로, 본 연구에서 추구하는 신경망의 경우 입력 유니트 뉴런이 3개(서비스코드, 이벤트코드, 내용코드)로 비교적 간단한 구조의 신경망인 점을 고려하여 은닉층의 수를 1개로 고정하고 은닉층 유니트 수의 경우 유니트 수의 2배수씩 증가시키면서 학습패턴을 보았

다. 또한, 신경망 수렴과 학습계수와 지역극소점 문제를 고려할 수 있는 모멘텀 계수의 경우 일정하게 정해진 값이 없으므로 0.5 ~ 0.9사이의 값으로 적절하게 적용하고, 학습률은 0.1 ~ 2사이의 값을 적절하게 적용하고, 활성화 함수의 기울기도 학습결과에 따라 변화시켰다. 신경망의 학습에서 학습 반복횟수의 경우 많으면 많을수록 오차가 줄어들지만, 본 개발 모형은 최대 학습횟수를 10,000,000번으로 정하고, 그 안에서 에러오차가 0.0001보다 작을 때까지 수행하여 학습에러와 학습횟수가 가장 작은 경우를 최적 학습대안으로 선정하였다.

표 4.7 신경회로망 학습대안 및 각 대안별 학습횟수

예측 대상	입력층 유니트 수	은닉 층수	반복 횟수	은닉층 유니트 수	α 학습 계수	β 모멘텀 계수	학습Error (학습횟수)
상황 우선 순위	3	1	10,000,000	15	1.2	0.001	0.574427
						0.002	0.151844
						0.003	0.198159
					1.5	0.001	0.477732
						0.002	0.541047
						0.003	0.187895
					1.7	0.001	0.575157
						0.002	0.238596
						0.003	0.280742
				30	1.2	0.001	0.219545
						0.002	0.000603
						0.003	0.030267
					1.5	0.001	0.083093
						0.002	0.003645
						0.003	0.136346
					1.7	0.001	0.234100
						0.002	0.256211
						0.003	0.190721
				60	1.2	0.001	0.156529
						0.002	0.000100(98,427번)
						0.003	0.000100(68,431번)
					1.5	0.001	0.027860

						0.002	0.000100(120,410번)
						0.003	0.000100(72,136번)
					1.7	0.001	0.000100(304,760번)
						0.002	0.000239
						0.003	0.000100(168,499번)

본 연구의 서비스 코드, 이벤트 코드, 내용코드를 효율적으로 학습시키기 위한 신경망 학습대안은 표 4.7과 같이 설정하였다. 각 학습대안에 따른 신경망 학습은 상황 코드 체계로 구성할 있는 서비스코드(4) × 이벤트코드(6) × 내용코드(25)의 총 600가지의 상황패턴을 기준으로 500건의 상황패턴은 신경망 학습을 위하여 이용하고, 나머지 100건의 상황패턴은 신경망모형 검증을 위한 자료로 활용하였다.

본 연구에서 제시된 신경망 학습 대안 중 에러오차 0.0001을 만족하여 학습이 완료되었을 때의 학습횟수가 가장 작은 값으로 학습된 신경망을 최종 모형으로 결정하였다.

분석결과 은닉층 유니트 수, 학습계수, 모멘텀 계수에 따른 최적 학습 대안은 표 4.8와 같고, 신경망 구조는 그림 4.16과 같다.

표 4.8 최적 학습대안

신경망 모형	입력층 유니트 수	은닉 층수	은닉층 유니트 수	α 학습 계수	β 모멘텀 계수	학습횟수
	3	1	60	1.2	0.003	68,431

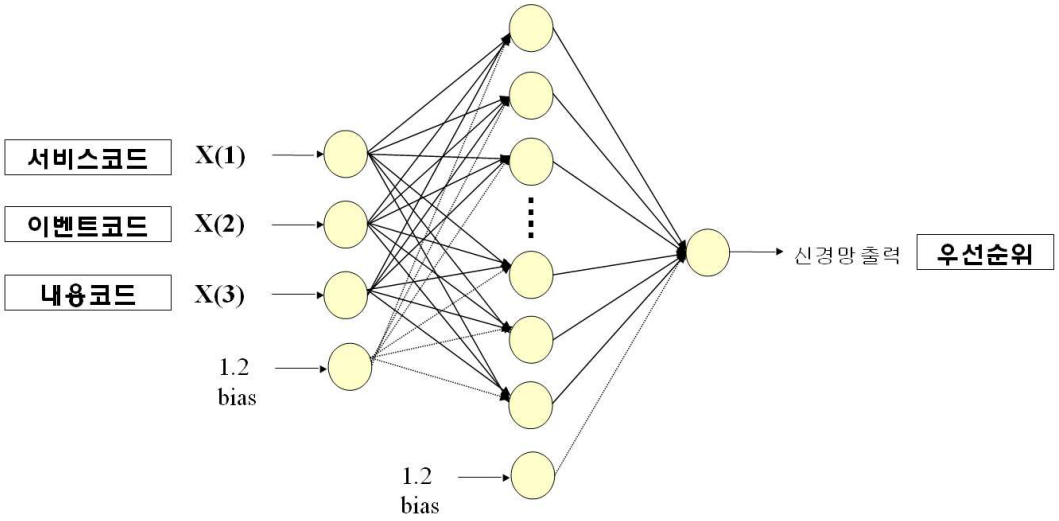


그림 4.16 이벤트 우선순위 분류를 위한 신경망 구조

4.4 미들웨어 처리 흐름도 설계

위의 4.2절에서는 미들웨어의 전체적인 구조에 대해 설명하였고, 각 내부 모듈의 기능 및 역할에 대해 정의하였다. 이번 절에서는 미들웨어의 처리 흐름도에 대해 살펴보겠다.

4.4.1 미들웨어 전체 처리흐름도

미들웨어 전체 처리 흐름도는 다음 그림 4.17에서 보는 바와 같다.

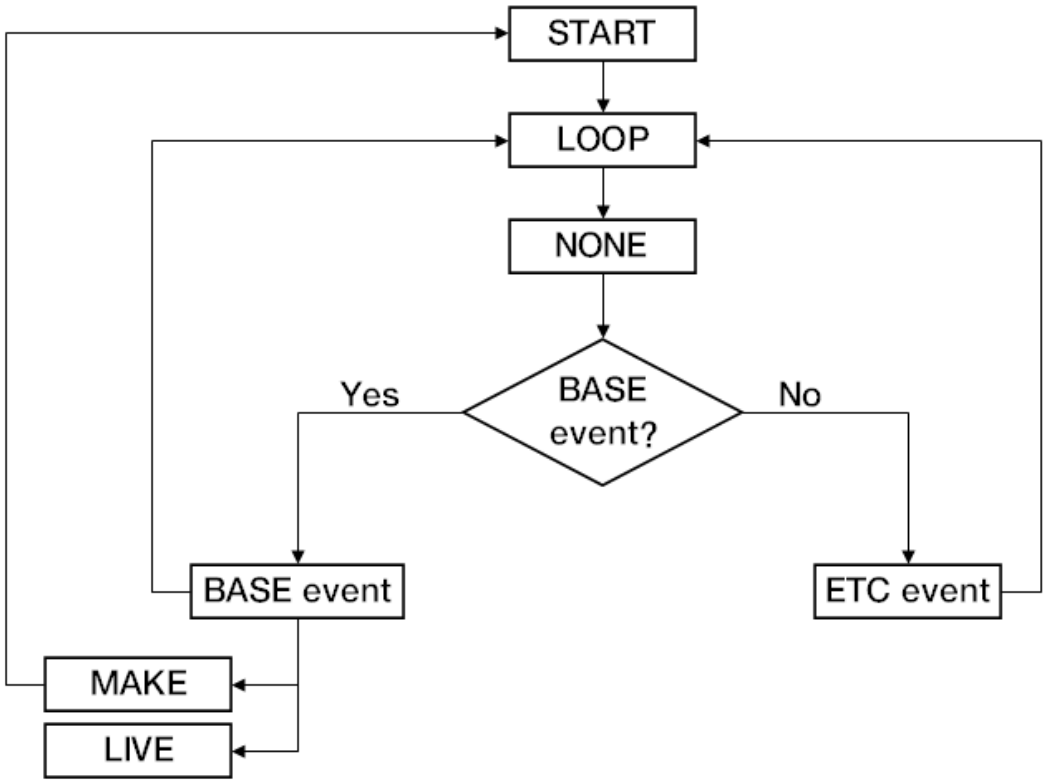


그림 4.17 미들웨어 전체 처리 흐름도

① START : 미들웨어의 초기화를 수행하면서, 메인프로세스, 내부프로세스, 내외부프로세스를 기동한다.

② LOOP : 프로세스의 정상동작 상태 및 반복처리 부분을 체크한다.

③ NONE : 모든 기능이 정상적으로 동작된다.

④ BASE event? : 이벤트가 기본적으로 수행되는 것인지? 특수 목적에 의한 것인지에 대한 여부를 판단한다.

⑤ BASE event : 기본 이벤트에 대한 처리를 수행한다.

⑥ MAKE : 프로세스 또는 Agent를 생성해야 한다면, 생성하고 그에 대한 상태를 체크한다.

⑦ LIVE : 이미 생성되어 있다면, 그에 대한 이벤트를 처리한다.

⑧ ETC event : 기본 이벤트가 아닌 특수 목적을 수행하기 위한 이벤트를 수행한다.

4.4.2 미들웨어 START(초기화) 상세흐름도

미들웨어가 처음 기동되면서 초기화할 때의 흐름도를 보여준다.

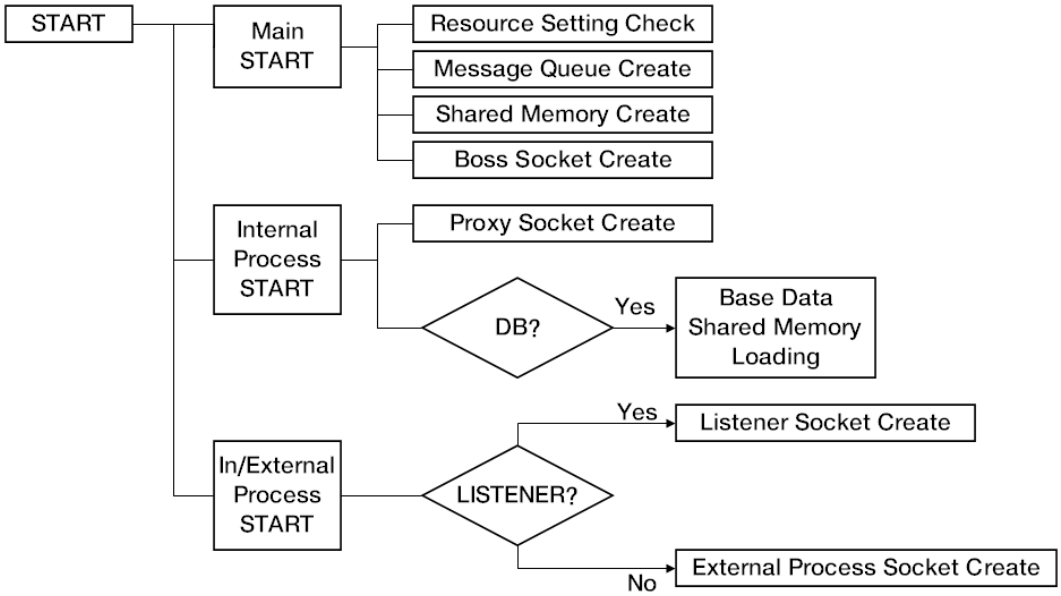


그림 4.18 미들웨어 초기화 상세흐름도

① Main Start : 미들웨어 Core의 메인로직을 시작시킨다. 리소스 설정에 대한 검사와 메시지 큐 생성, 공유 메모리 생성, Core를 위한 보스 소켓을 생성한다.

② Internal Process Start : 내부 프로세스를 위한 프록시 소켓을 생성하고, DB가 연결되면, 기초 데이터를 공유 메모리에 적재한다.

③ In/External Process Start : 내부와 외부 프로세스를 위한 Listener를 체크하여, Listener 소켓을 생성하거나, 외부 프로세스를 위한 소켓을 생성한다.

4.4.3 미들웨어 LOOP 상세흐름도

미들웨어내의 프로세스 정상동작 검사 및 반복처리 부분에 대한 흐름도이다.

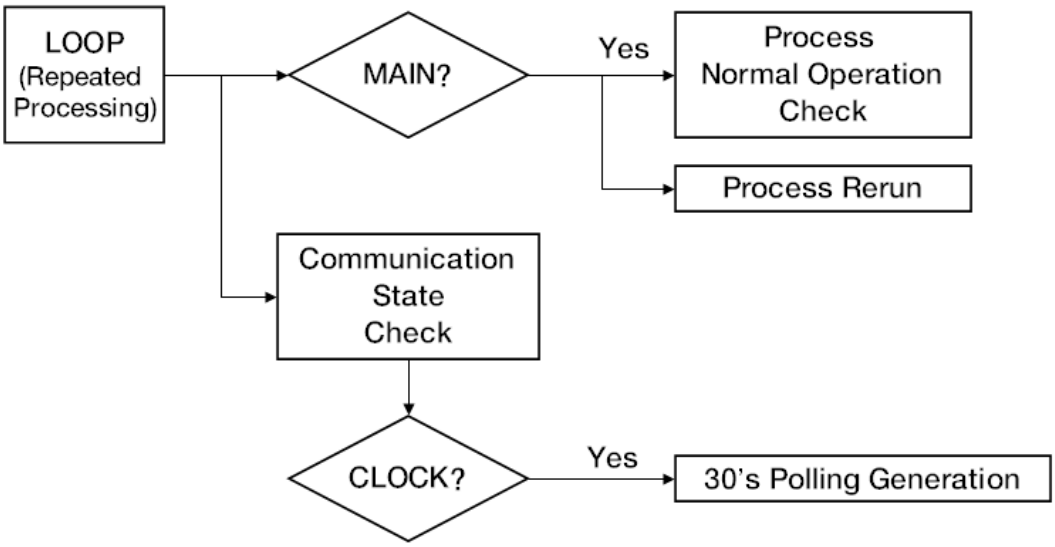


그림 4.19 미들웨어 LOOP 상세흐름도

① MAIN : 미들웨어 메인 동작을 위한 프로세스가 정상적으로 동작하는지 체크하여 프로세스가 죽어 있다면, 프로세스를 재실행한다.

② Communication State Check : 생성된 프로세스에 대해 통신 상태를 체크하고, 클럭을 필요로 하는 프로세스는 30초 폴링한다.

4.4.4 미들웨어 ETC 이벤트 처리 상세흐름도

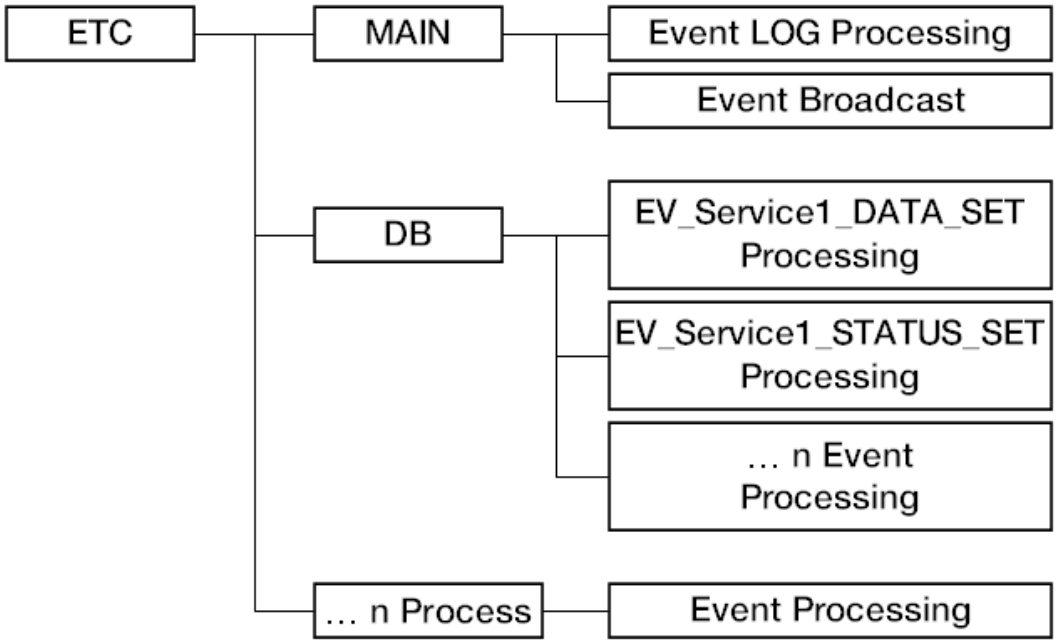


그림 4.20 미들웨어 ETC 이벤트 처리 상세흐름도

① MAIN : 이벤트에 대한 로그 관리를 하고, 이벤트를 중계(broadcast) 한다.
② DB : DB와의 연계를 위한 이벤트를 처리하는데, 이벤트와 관련된 데이터와 상태정보를 처리한다.
③ n Process : 발생한 이벤트를 처리한다.

4.5 미들웨어 프로세스간 이벤트 송수신 설계

위의 4.4절에서 미들웨어의 동작 흐름도에 대해 설명하였다. 이번 절에서는 미들웨어의 프로세스간 이벤트 송수신 처리 흐름도에 대해 살펴보겠다.

4.5.1 이벤트 처리흐름도

미들웨어내 프로세스간 이벤트 처리는 통신상태 체크, 이벤트 검사, 이벤트 처리를 반복한다. 다음 그림 4.21에서는 이벤트 처리 개념도에 대해 보여준다.

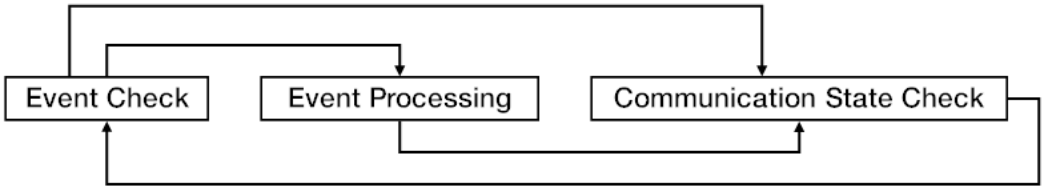


그림 4.21 이벤트 처리 개념도

프로세스간 이벤트 처리는 다음 그림 4.22에서 보는 바와 같이 Process1에서 메시지큐에 이벤트를 전송하고, MAIN에서는 메시지큐의 이벤트를 받는다. MAIN은 이벤트맵을 참조하여 이벤트를 메시지큐로 Broadcast하고, Process2에서는 메시지큐의 이벤트를 수신하여 처리하는 흐름으로 되어 있다.

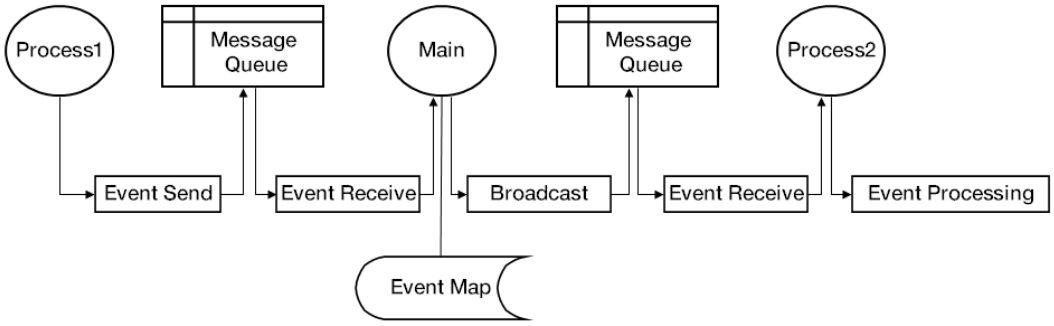


그림 4.22 프로세스간 이벤트 처리 흐름도

4.5.2 이벤트 사용 방법

미들웨어의 이벤트맵 구조는 그림 4.23과 같이 송신권한, 수신권한, 이벤트 타입, 송신 데이터 타입으로 되어 있다.

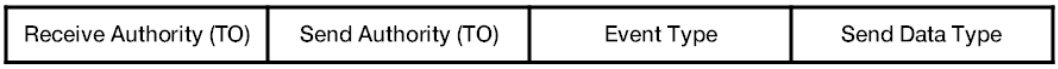


그림 4.23 이벤트맵 구조

이벤트는 이벤트맵에 설정된 이벤트의 순서와 동일하게 헤더파일에 정의한다. 다음 그림 4.24에서는 이벤트 전송 처리에 대해 이벤트 송신, 이벤트 중계, 이벤트 수신 부분으로 구분하여 보여준다.

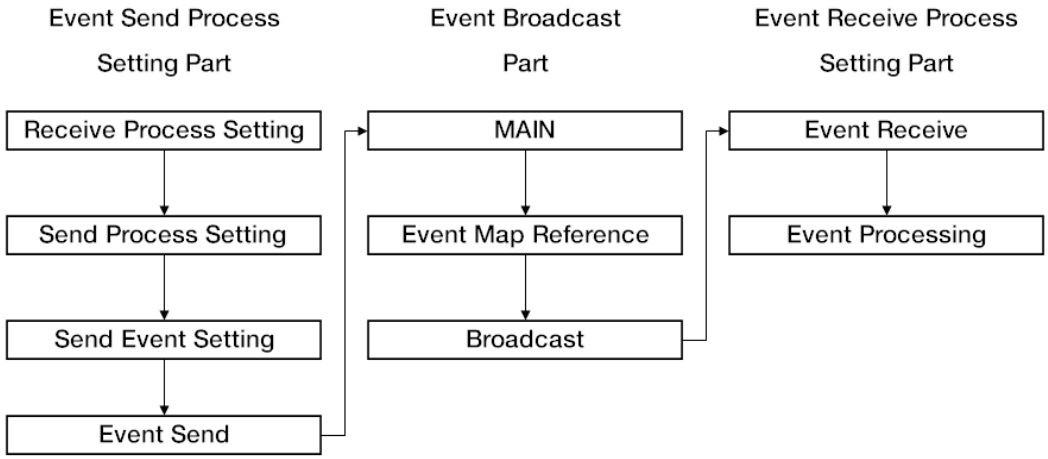


그림 4.24 이벤트 전송 흐름도

4.6 미들웨어 공유메모리 관리 설계

위의 4.5절에서 미들웨어내의 프로세스간 이벤트 송수신 처리에 대해 설명하였다. 이번 절에서는 미들웨어의 공유메모리 관리에 대해 살펴보겠다.

미들웨어에서 공유메모리에 데이터를 쓰고 읽기 위한 구조체를 정의한다. 다음 그림 4.25에서는 공유메모리 구조체를 정의하는 방법에 대해 보여준다.

필드	Structure 명	DATA 형	길이	내 용
장비ID	PHONE_ID	CHAR	12	MASTER에 가져올 장비 ID
호출번호	PHONE_NUM	INT	14	호출 선로 번호
검출번호	CIRCUIT_NUM	INT	14	통화여부 검출 번호
모기/자기여부	MASTER_FLAG	CHAR	1	모기/자기 여부
단말기버전	PHONE_VERSION	CHAR	10	단말기 VERSION
단말기명	PHONE_NAME	CHAR	12	단말기 명칭
자가진단순번	SAFTY_CODE	CHAR	5	단말기 자가진단 순서 번호
기점방향	PHONE_DIRECT	CHAR	7	기점 방향 설정
그룹구분	GROUP_ID	CHAR	10	그룹구분ID, 일반, 터널명

```
struct EMERGENCY_CALL {                    // 기본적인 운영에 필요한 구조체
        char PHONE_ID[12];                 // MASTER 에 가져올 장비 ID
        int  PHONE_NUM;                    // 호출 선로 번호
        int  CIRCUIT_NUM;                  // 통화여부 검출 번호
        char MASTER_FLAG;                  // 모기/ 자기 여부
        char PHONE_TYPE[10];               // 단말기 VERSION
        char PHONE_NAME[12];               // 단말기 명칭
        char SAFTY_CODE[5];                // 단말기 자가진단 순서 번호
        char PHONE_DIRECT[7];              // 기점 방향 설정
        char GROUP_ID[10];                 // 그룹구분 ID ,일반,터널명
};
```

그림 4.25 공유메모리 구조체 정의

공유메모리에서 사용할 구조체를 정의한 후에는 메모리명을 정의하고, 메모리 맵을 설정한다.

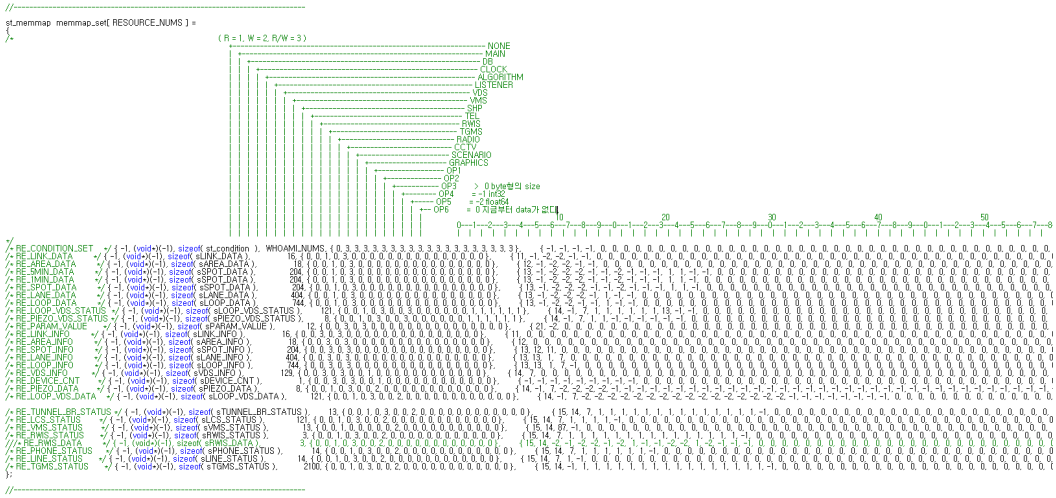


그림 4.26 메모리 맵 설정

공유메모리를 설정할 때는 권한설정과 메모리를 읽고 쓰기 위한 구조체 멤버 type을 지정하고, 메모리 row수를 지정해야 한다. 다음 그림 4.27에서는 공유메모리 구조에 대해 보여준다.

Shared Memory ID	Shared Memory Address	Row Size	Row Count	Read, Write Authority	Structure Member Type & Size Appointment

그림 4.27 공유메모리 구조

제 5 장 미들웨어의 구현 및 실험

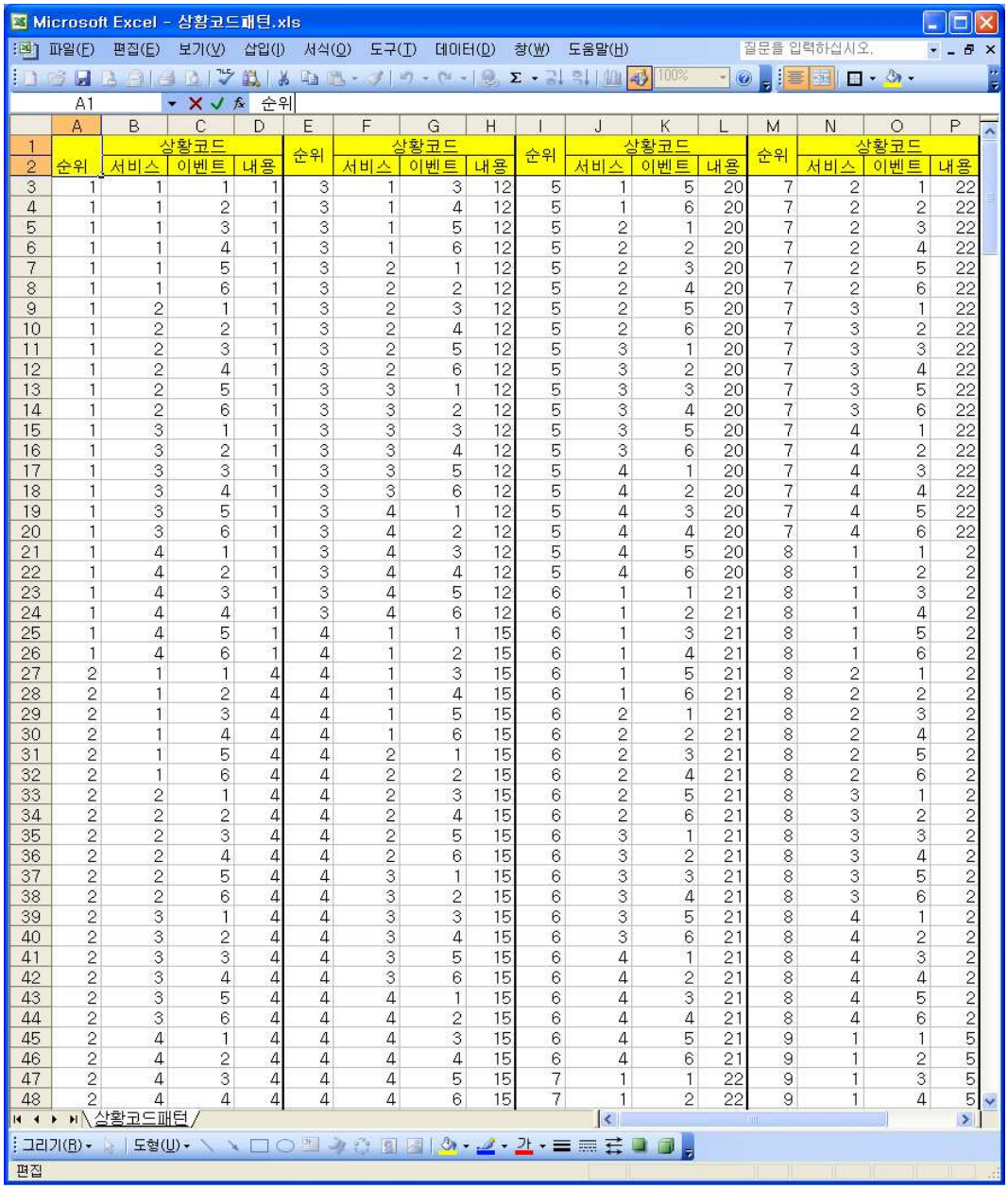
이번 장에서는 먼저 신경망 모형을 검증하고, 이어서 실제 미들웨어가 어떻게 구현되었으며, 구현된 결과는 어떤 식으로 구성되었는지 살펴본다. 마지막으로 유비쿼터스 도시통합운영환경의 테스트베드를 구현하여, 미들웨어에 대한 성능 테스트를 수행하고, 그 결과를 살펴보도록 하겠다.

5.1 신경회로망 구현 및 실험결과

5.1.1 실험 환경

본 연구에서 제안한 신경망을 이용한 이벤트 우선순위 분류 알고리즘의 성능을 평가하기 위하여, 4.3절에서 정의한 상황별 이벤트 코드체계를 이용하여 우선순위 분류 실험을 수행하였으며 실험에 사용된 자료의 수는 서비스코드(4), 이벤트코드(6), 내용코드(25)의 곱인 600개이다. 실험에 사용된 상황코드패턴은 그림 5.1과 같다.

상황별 이벤트 코드체계를 2개 그룹으로 나누어 먼저 500개의 상황 코드 체계 값을 학습 데이터(Training Set)로 사용하였고, 학습하지 않은 나머지 100개와 학습한 500개의 상황 코드 체계 값을 실험 데이터(Test Set)로 사용하였다. PENTIUM IV 3.2 GHz에서 Visual C++를 이용하여 역전파 신경망 알고리즘을 구현하였다. 본 실험에 이용한 역전파 신경망은 입력층 유니트 3개, 은닉층 1개층 유니트 10개, 출력층 1개 유니트로 구성하였다.



Microsoft Excel - 상황코드패턴.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		상황코드			순위	상황코드			순위	상황코드			순위	상황코드		
2	순위	서비스	이벤트	내용		서비스	이벤트	내용		서비스	이벤트	내용		서비스	이벤트	내용
3	1	1	1	1	3	1	3	12	5	1	5	20	7	2	1	22
4	1	1	2	1	3	1	4	12	5	1	6	20	7	2	2	22
5	1	1	3	1	3	1	5	12	5	2	1	20	7	2	3	22
6	1	1	4	1	3	1	6	12	5	2	2	20	7	2	4	22
7	1	1	5	1	3	2	1	12	5	2	3	20	7	2	5	22
8	1	1	6	1	3	2	2	12	5	2	4	20	7	2	6	22
9	1	2	1	1	3	2	3	12	5	2	5	20	7	3	1	22
10	1	2	2	1	3	2	4	12	5	2	6	20	7	3	2	22
11	1	2	3	1	3	2	5	12	5	3	1	20	7	3	3	22
12	1	2	4	1	3	2	6	12	5	3	2	20	7	3	4	22
13	1	2	5	1	3	3	1	12	5	3	3	20	7	3	5	22
14	1	2	6	1	3	3	2	12	5	3	4	20	7	3	6	22
15	1	3	1	1	3	3	3	12	5	3	5	20	7	4	1	22
16	1	3	2	1	3	3	4	12	5	3	6	20	7	4	2	22
17	1	3	3	1	3	3	5	12	5	4	1	20	7	4	3	22
18	1	3	4	1	3	3	6	12	5	4	2	20	7	4	4	22
19	1	3	5	1	3	4	1	12	5	4	3	20	7	4	5	22
20	1	3	6	1	3	4	2	12	5	4	4	20	7	4	6	22
21	1	4	1	1	3	4	3	12	5	4	5	20	8	1	1	2
22	1	4	2	1	3	4	4	12	5	4	6	20	8	1	2	2
23	1	4	3	1	3	4	5	12	6	1	1	21	8	1	3	2
24	1	4	4	1	3	4	6	12	6	1	2	21	8	1	4	2
25	1	4	5	1	4	1	1	15	6	1	3	21	8	1	5	2
26	1	4	6	1	4	1	2	15	6	1	4	21	8	1	6	2
27	2	1	1	4	4	1	3	15	6	1	5	21	8	2	1	2
28	2	1	2	4	4	1	4	15	6	1	6	21	8	2	2	2
29	2	1	3	4	4	1	5	15	6	2	1	21	8	2	3	2
30	2	1	4	4	4	1	6	15	6	2	2	21	8	2	4	2
31	2	1	5	4	4	2	1	15	6	2	3	21	8	2	5	2
32	2	1	6	4	4	2	2	15	6	2	4	21	8	2	6	2
33	2	2	1	4	4	2	3	15	6	2	5	21	8	3	1	2
34	2	2	2	4	4	2	4	15	6	2	6	21	8	3	2	2
35	2	2	3	4	4	2	5	15	6	3	1	21	8	3	3	2
36	2	2	4	4	4	2	6	15	6	3	2	21	8	3	4	2
37	2	2	5	4	4	3	1	15	6	3	3	21	8	3	5	2
38	2	2	6	4	4	3	2	15	6	3	4	21	8	3	6	2
39	2	3	1	4	4	3	3	15	6	3	5	21	8	4	1	2
40	2	3	2	4	4	3	4	15	6	3	6	21	8	4	2	2
41	2	3	3	4	4	3	5	15	6	4	1	21	8	4	3	2
42	2	3	4	4	4	3	6	15	6	4	2	21	8	4	4	2
43	2	3	5	4	4	4	1	15	6	4	3	21	8	4	5	2
44	2	3	6	4	4	4	2	15	6	4	4	21	8	4	6	2
45	2	4	1	4	4	4	3	15	6	4	5	21	9	1	1	5
46	2	4	2	4	4	4	4	15	6	4	6	21	9	1	2	5
47	2	4	3	4	4	4	5	15	7	1	1	22	9	1	3	5
48	2	4	4	4	4	4	6	15	7	1	2	22	9	1	4	5

그림 5.1 신경망 학습을 위한 상황코드패턴

5.1.2 신경망 학습 및 실험 결과

이벤트 우선순위 분류 실험을 수행하기에 앞서 먼저 500개의 상황 코드 체계 학습 데이터를 이용하여 신경망 학습을 수행하였다. 그림 5.2와 5.3은 학습률, 오류 목표값, 모멘텀을 각각 1.2, 0.0001, 0.003으로 하고 학습을 수행한 결과 오류가 특정 값으로 수렴됨으로써 학습이 수행된 결과를 보여준다.

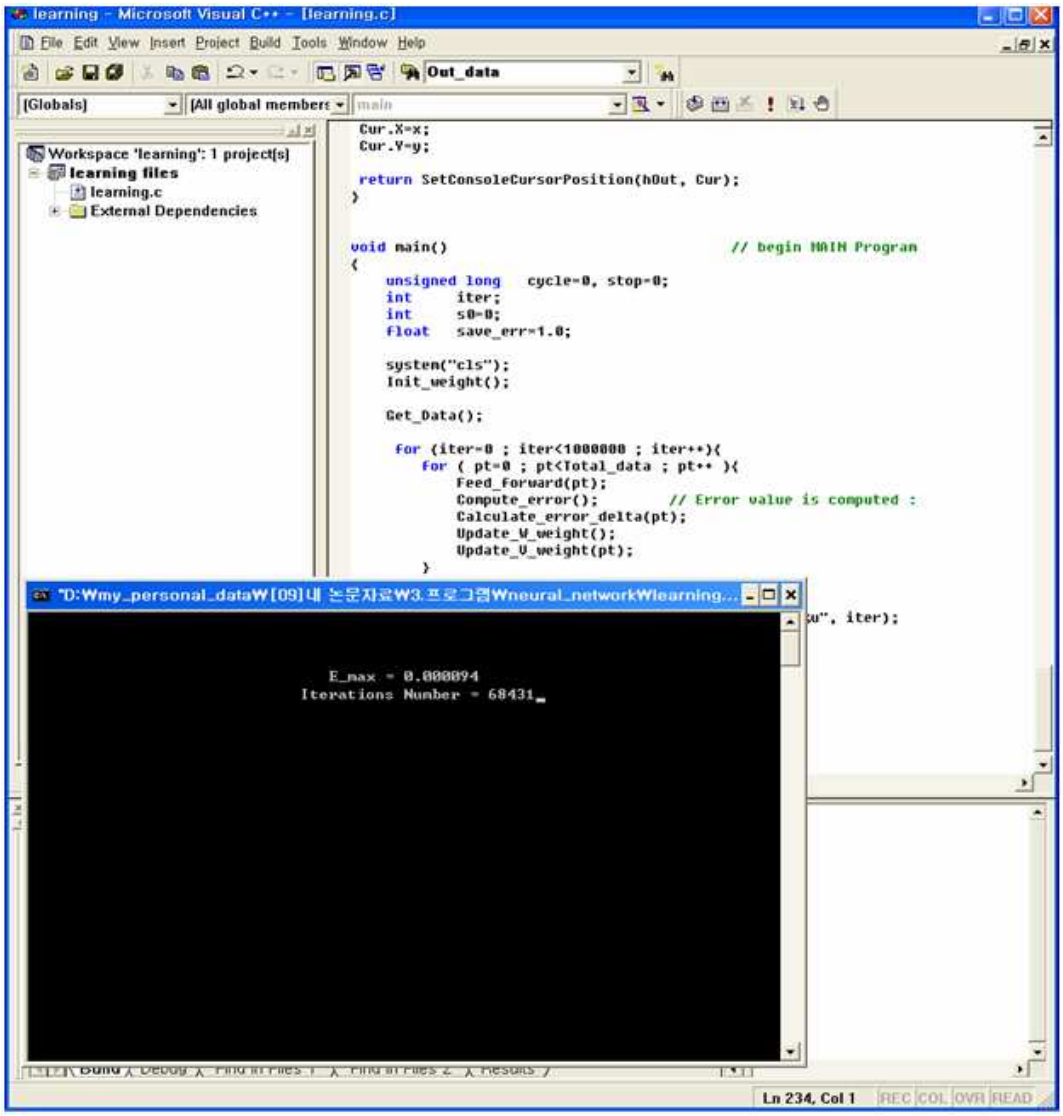


그림 5.2 신경망 학습 실행 결과

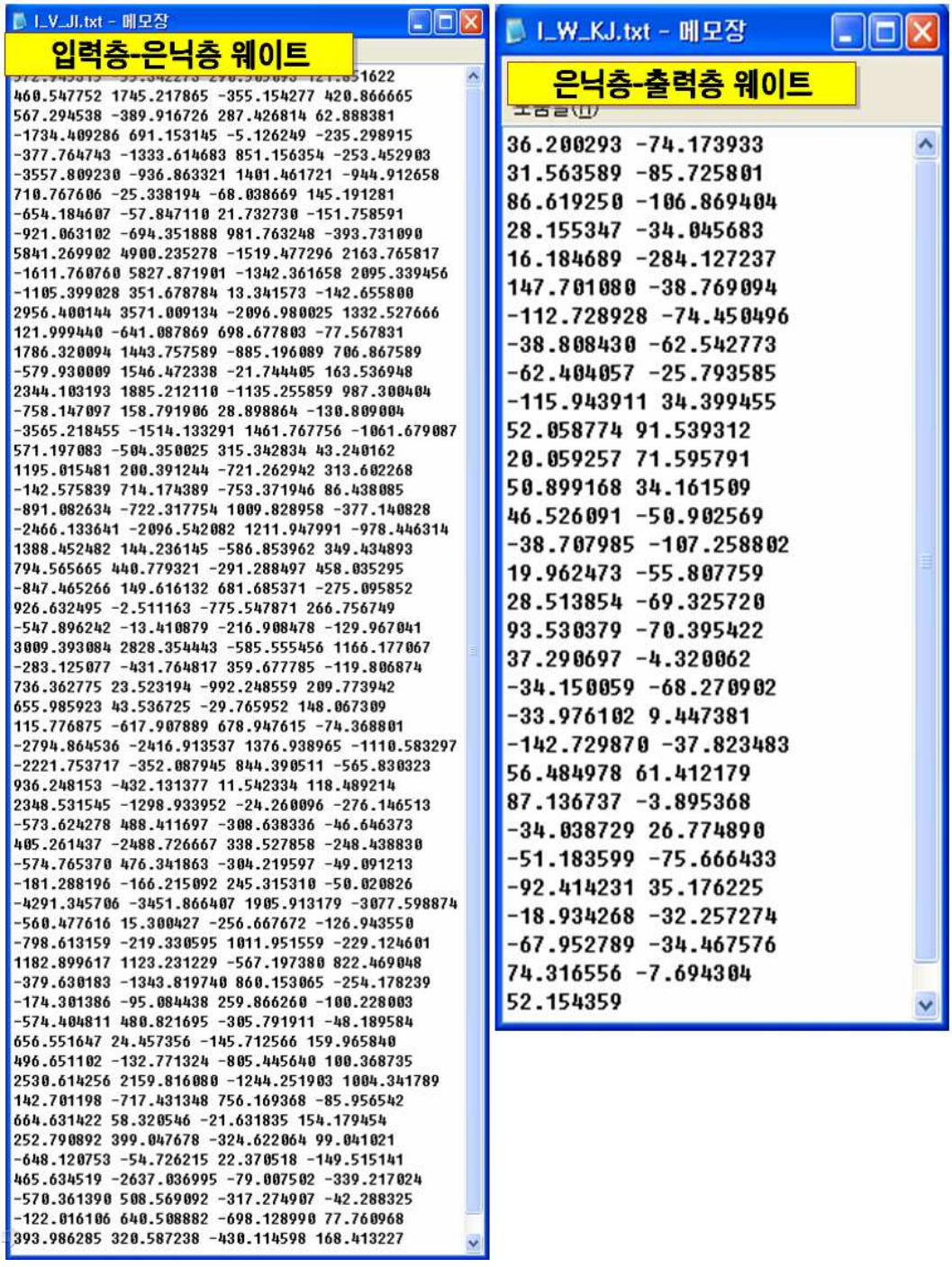


그림 5.3 학습이 끝난 후의 신경망 웨이트

학습을 수행한 후 학습하지 않은 100개의 이벤트와 학습한 500개의 이벤트 데이터를 이용하여 이벤트 우선순위 분류 실험을 수행하였으며, 그 결과는 표 5.1과 같다. 실험 결과의 평가를 위한 평가지표로 인식율과 처리속도를 사용하였으며, 인식율은 총 시험이벤트 중 미분류 이벤트를 제외하고 분류대상 이벤트 중 정확하게 분류된 이벤트수를 말한다. 이를 식으로 나타내면 식 5.1과 같다.

$$\text{인식율} = \frac{\text{분류된 이벤트수}}{\text{실험 이벤트수}} \qquad \text{(식5.1)}$$

실험 결과 전체 실험대상 이벤트 600개 중 학습한 500개의 이벤트 우선순위는 정확하게 분류하고, 학습하지 않은 100개 중 78개의 이벤트 우선순위를 정확하게 분류하여 전체 600개 중 총 578개를 정확하게 분류하였고, 이에 대한 전체 인식률은 96.3%로 나타났고, 평균 처리속도는 0.01초였다.

학습여부에 따른 실험 결과를 살펴보면 학습한 이벤트의 경우 이벤트 500개 중 500개의 이벤트를 분류하여 100%의 인식율을 보였다.

학습하지 않은 이벤트일 경우 이벤트 100개중 78개의 이벤트를 분류하여 78%의 인식율을 보였다. 22개의 이벤트는 정확하게 분류하지는 못했지만, 유사한 분류를 출력함으로써 분류를 하지 못하는 오류는 0%임을 보였다.

표 5.1 신경망 이벤트 우선순위 분류 실험결과

실험결과	학습한 이벤트 (총 500개)	학습하지 않은 이벤트 (총 100개)	합계 (총 600개)
	500개 분류	78개 분류	578개 분류
인식률	100%	78%	96.3%

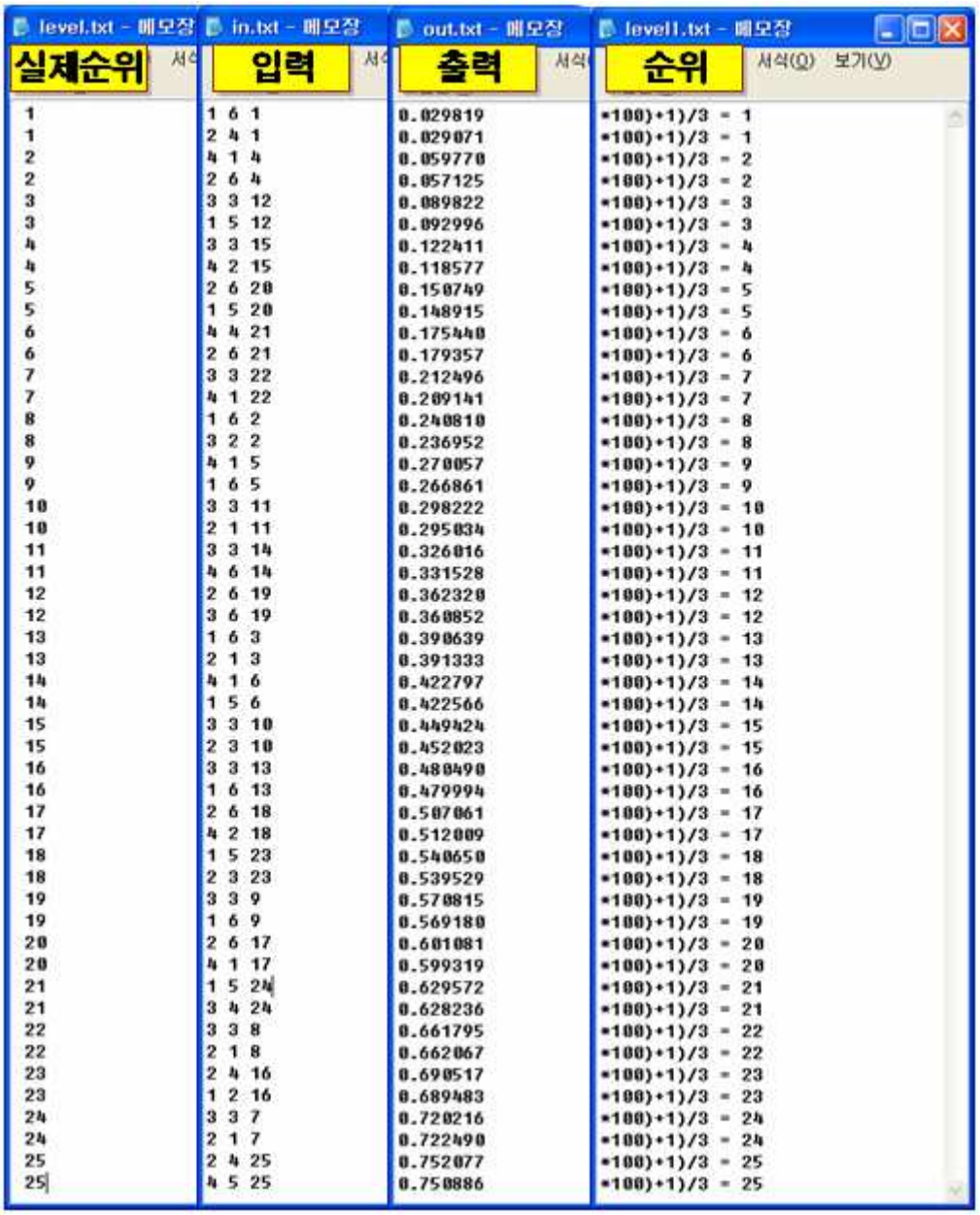


그림 5.4 학습한 패턴에 대한 실험결과

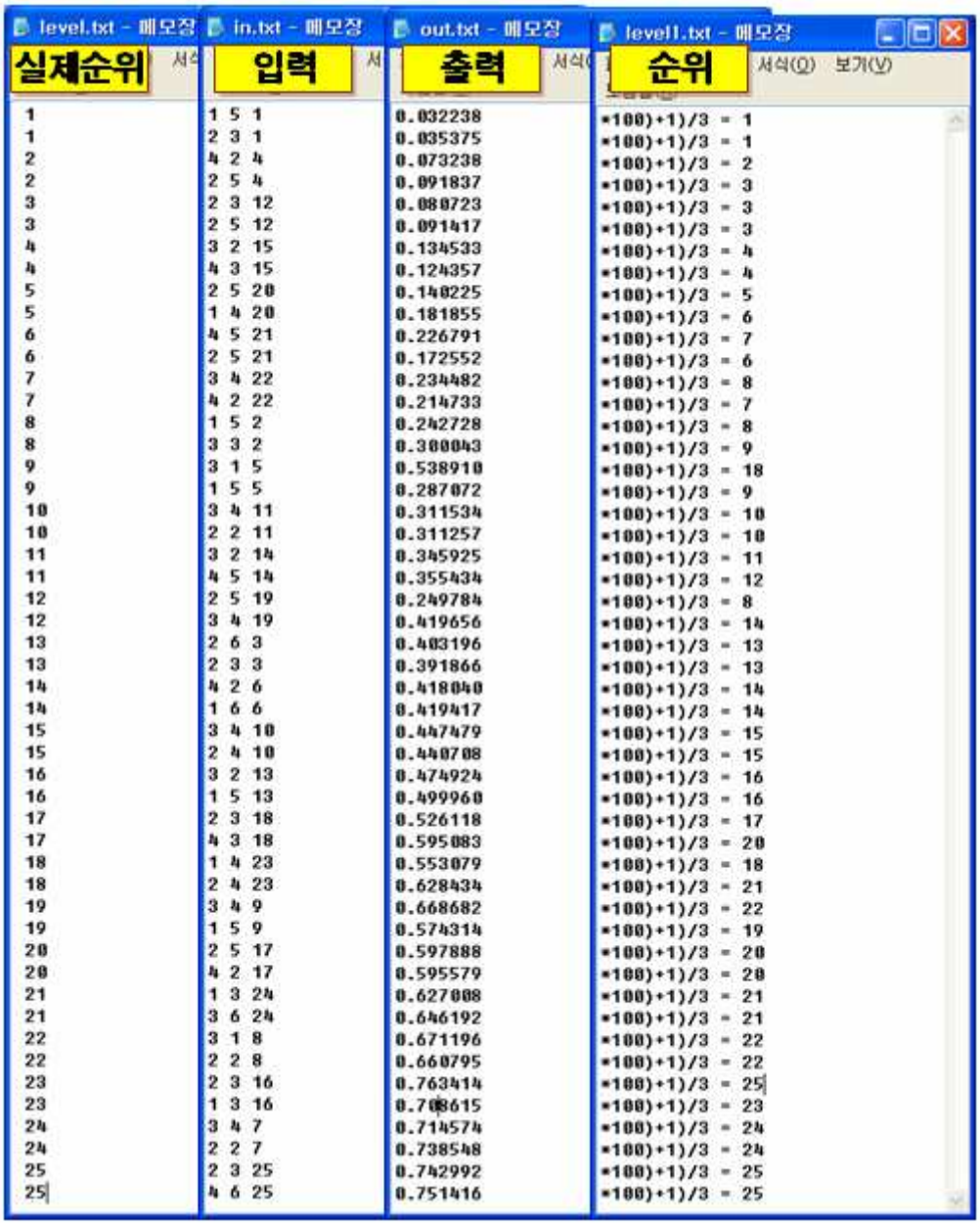


그림 5.5 학습하지 않은 패턴에 대한 실험결과

5.1.3 이벤트 우선순위 분류 알고리즘 평가

본 연구에서 제안한 신경망을 이용한 이벤트 우선순위 분류 알고리즘을 기존의 룰-데이터베이스 기반 우선순위 알고리즘과 비교하여 분석하면 표 5.2와 같다. 기존의 우선순위 알고리즘은 데이터베이스에 저장된 룰 테이블을 참조하여 이벤트의 우선순위 분류를 수행하는 방식이나, 신경망을 이용한 알고리즘은 데이터베이스와 상관없이 신경망 자체로 분류를 수행하는 방식이다.

2가지 이벤트 우선순위 분류 방식의 핵심 알고리즘을 살펴보면, 기존의 우선순위 알고리즘 분류 방식은 이벤트에 따라 구성한 룰-데이터를 데이터베이스에 미리 저장하고, 이벤트가 발생했을 경우에 데이터베이스에 저장된 우선순위 정보를 조회하여 우선순위를 분류하는 것으로, 입력되지 않은 룰-데이터에 대한 이벤트가 발생했을 경우에는 분류할 수 없고, 데이터베이스 오류가 발생할 경우에도 분류할 수 없다. 본 연구에서 제안한 신경망을 이용한 우선순위 분류 방식은 이벤트에 따라 구성한 상황코드패턴으로 신경망 학습을 수행하고, 이벤트가 발생했을 경우에 학습된 신경망이 우선순위를 자동 분류하는 것으로, 학습되지 않은 이벤트가 발생했을 경우에도 분류할 수 있고, 데이터베이스의 오류와 상관없이 분류할 수 있다.

룰-데이터베이스 기반 우선순위 알고리즘의 경우 실험결과 600개의 이벤트 중 미리 저장된 500개의 이벤트만 분류를 수행하였고, 저장되지 않은 100개의 이벤트는 분류하지 못하여 인식률이 83.3%로 나타났다.

본 연구에서 제안한 신경망을 이용한 이벤트 우선순위 분류 알고리즘은 학습 및 실험을 실시한 결과 600개의 이벤트 중 학습된 500개의 이벤트를 분류하고 학습되지 않은 78개의 이벤트를 분류하여 인식률이 96.3%로 나타났다. 이는 본 연구에서 제안한 알고리즘이 기존 우선순위 알고리즘에 비해 통합관제 환경에서 실시간 데이터 처리를 수행하는 성능이 뛰어남을 나타낸다.

표 5.2 이벤트 우선순위 알고리즘 비교

특징 \ 방식	룰-데이터베이스 방식	신경망 방식
입력된 데이터 또는 학습한 데이터 인식개수	500개 중 500개 분류	500개 중 500개 분류
입력되지 않은 데이터 또는 학습하지 않은 데이터 인식개수	100개 중 0개 분류	100개 중 78개 분류
처리속도	평균 0.1초	평균 0.01초
총 인식개수	600개 중 500개 분류	600개 중 578개 분류
인식률	83.3%	96.3%

5.2 미들웨어 구현 방법 및 결과

본 절에서는 4장에서 설계한 내용을 바탕으로 실제 미들웨어를 구현해 본 결과를 제시하도록 하겠다. 먼저, 미들웨어 기본 구조를 구현한 방법에 대해서 설명하고, 이어서 미들웨어를 구성하고 있는 각 부분들(Connector, External Agent, Internal Agent, Core, Listener)에 대해 어떻게 구현되었는지 설명하도록 하겠다. 이때, 전체적인 구현을 위해 C언어를 사용하였고, 구현 툴은 Visual C++ 6.0을 이용하였으며, 구현운영OS는 unix를 사용하였다.

5.2.1 미들웨어의 기본 구조 구현

본 연구에서 구현된 미들웨어는 4.2.1절에서 제안하고 있는 Server-Client구조를 사용하고 있다. 따라서, 미들웨어를 구현할 때 가장 먼저 이러한 기본 구조를 구축해야만 한다. 그리하여, 본 연구에서는 Server의 역할을 하고 있는 미들웨어를 다중 네트워크 연결이 지원되도록 구현하여 여러 클라이언트와 연동할 수 있도록 하였다. 다음의 그림 5.6의 순서도는 이러한 Server측 구현과정을 나타내고 있다. 미들웨어는 리스너와 External Agent를 통해 클라이언트에서 접속 요청을 할 때마다, 먼저 Listener가 접속요청을 접수하고 이미 접속되어 있는지, 아니면 허가된 접속인지를 검사하여 인증이 되면, Core에 알려서 클라이언트와의 통신을 담당하는 External Agent를 생성하여 클라이언트와 통신 연결을 한다. 그리고 여러 개의 클라이언트가 미들웨어와 각각 개별적으로 통신할 수 있도록 클라이언트마다 각기 다른 External Agent를 생성시킨다. 이때, 이러한 External Agent를 통해 주고 받는 메시지는 그림 5.6과 같은 구조를 따르도록 구현하였다.

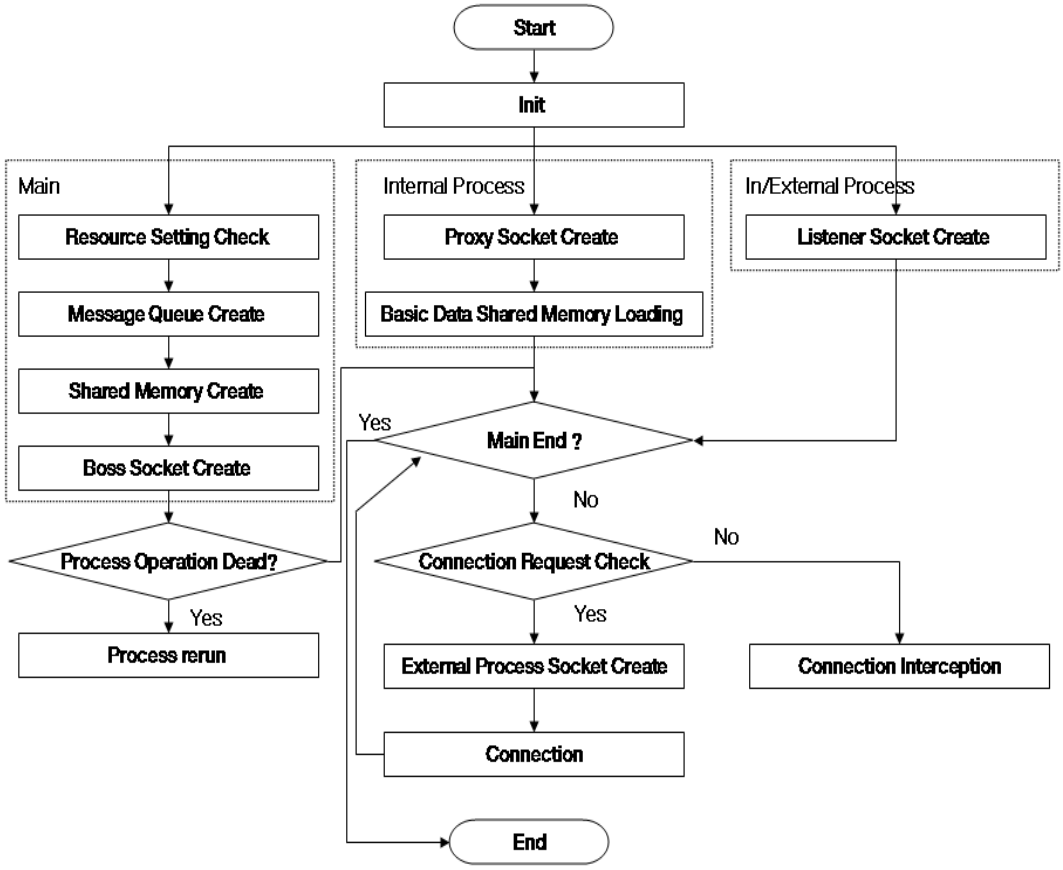


그림 5.6 미들웨어의 Server 역할 수행을 위한 구현 과정

클라이언트에서 미들웨어 Server로 데이터를 보낼 때는 데이터 셋 형태로 보낸다. 이는 그림 5.7에서와 같이 테이블과 같은 구조로 데이터를 담아서 보내는 형태이다. 먼저, 테이블을 생성하고, 정의한 컬럼에 데이터를 입력한다.

ID	TABEL NAME	COLUM NAME	TYPE	SIZE
UUID	TEST	COL1	C	10
		COL2	C	50
		COL3	F	10
		COL4	I	10
		KEY	I	10

```
// 데이터 테이블 생성
mw_tbcreate( UUID, TEST );
mw_addcolum ( UUID,  TEST, "col1", 'c', 50 );
mw_addcolum ( UUID,  TEST, "col2", 'c', 50 );
mw_addcolum ( UUID,  TEST, "col3", 'f',  0 );
mw_addcolum ( UUID,  TEST, "col4", 'i',  0 );
mw_addcolum ( UUID,  TEST, "key",  'i',  0 );
```

ID	TABEL NAME	COL1	COL2	COL3	COL4	KEY
UUID	TEST	첫번째	0001	1212.1	212	1
		두번째	0002	3434.4	234	2

```
// 데이터 테이블 값 입력
mw_insert (  UUID, TEST );
mw_slong (  UUID, TEST, "key" ,  1 );
mw_stext ( UUID,  TEST, "col1", "첫번째", 6 );
mw_stext ( UUID,  TEST, "col2", "0001",   4 );
mw_sdouble ( UUID,  TEST, "col3", 1212.1  );
mw_slong (  UUID, TEST, "col4", 212 );
mw_post ( UUID,TEST );
mw_insert (  UUID, TEST );
mw_slong (  UUID, TEST, "key" ,  2 );
mw_stext ( UUID,  TEST, "col1", "두번째", 6 );
mw_stext ( UUID,  TEST, "col2", "0002",   4 );
mw_sdouble ( UUID,  TEST, "col3", 3434.4  );
mw_slong (  UUID, TEST, "col4", 234 );
mw_post ( UUID,TEST );
```

그림 5.7 미들웨어의 데이터 구조

5.2.2 Agent 구현

1) Agent 정의

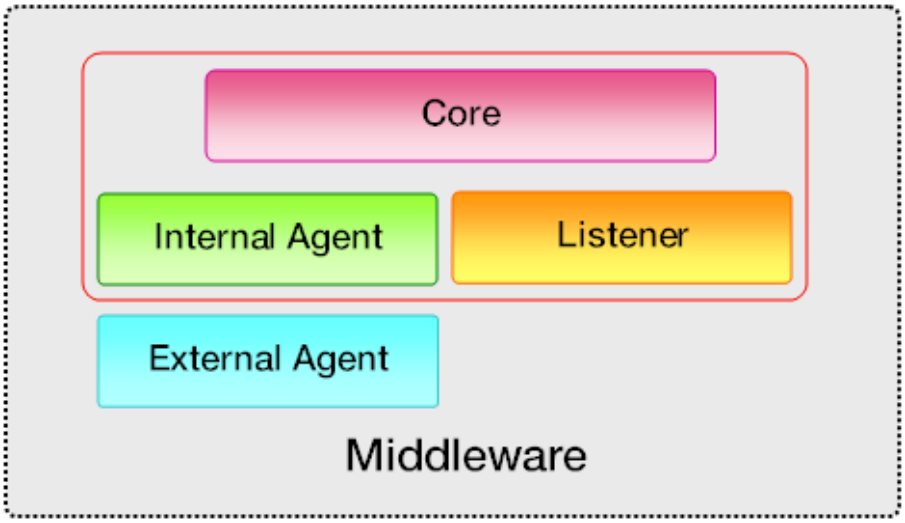


그림 5.8 Agent 구조

① Agent는 역할 분류에 따라 Listener, External Agent, Internal Agent로 구분된다.

- Listener : 외부 통신의 절차에 따라 Core와 통신을 개설한다.
- Internal Agent는 미들웨어의 기동 시 자동으로 생성되며 서비스 로직을 처리한다.
- External Agent는 외부접속 시 생성되며 외부 어플리케이션과 통신한다.

② Agent는 생존유무에 따라 자동 재생성한다.

- LIVE_RETRY
- LIVE_DONE

③ Agent는 재실행 횟수를 정의할 수 있다.

- get_relive_status

2) Listener Agent

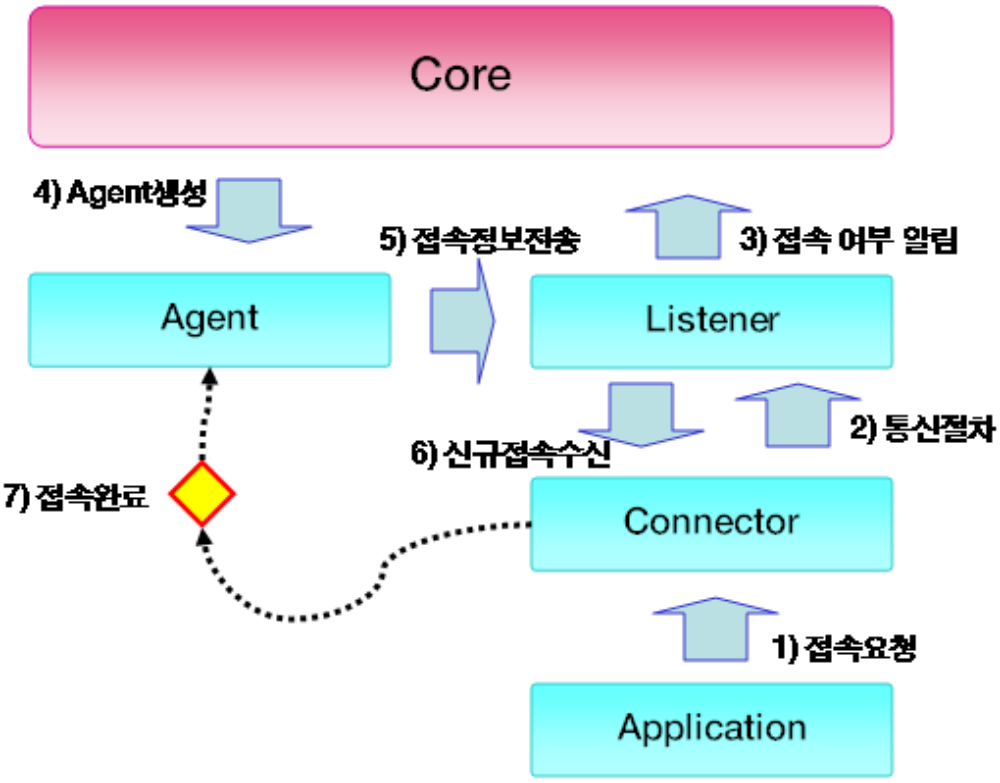


그림 5.9 Listener Agent 구성도

① Listener는 Internal Agent중에서 마지막으로 생성된다.

② Gateway와 같은 공개된 IP와 Port를 가진다.

③ 실제 접속되는 Port정보는 가지고 있지 않고 통신을 통해서만 외부에 이를 전달한다.

④ 외부와의 접속 종료 시 재실행되는 Agent이다.

⑤ 접속 처리가 끝나면 기존 접속을 끊고 새로운 접속을 받아들인다.

3) Internal Agent

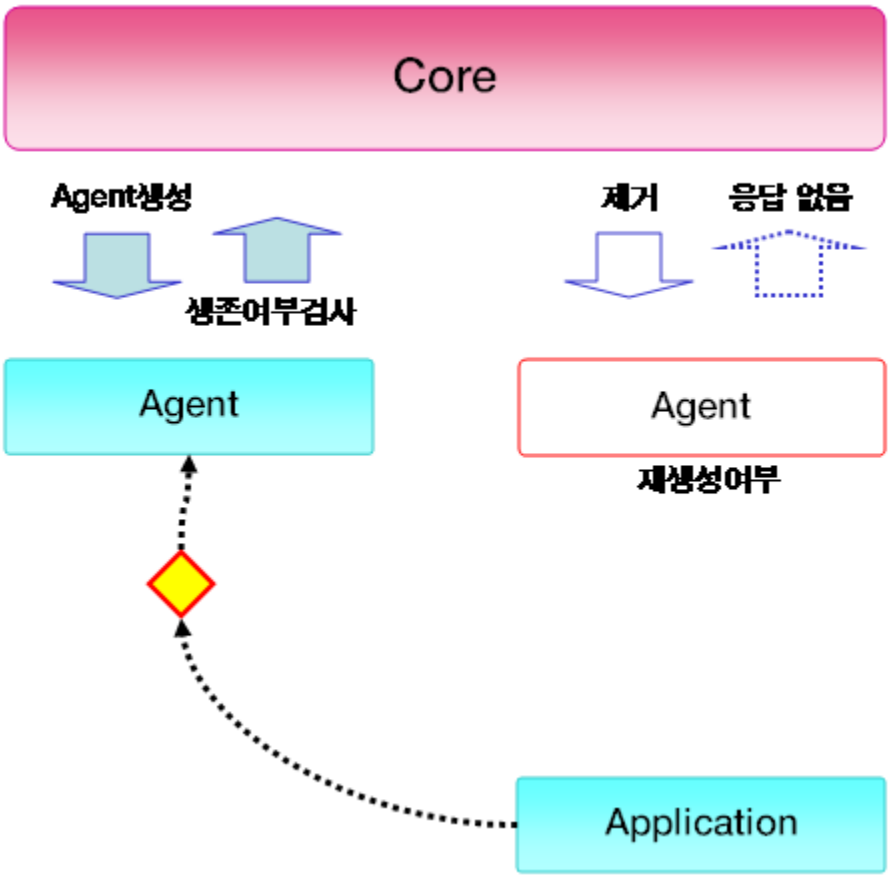


그림 5.10 Internal Agent 구성도

① Internal Agent에서 초기 생성순서를 가진다.

② Listener와 같이 외부와 별도 Socket을 통해 통신을 할 수 있다.

③ 종료 시 재생성 여부를 결정할 수 있다.

④ 특정 시간 동안 응답이 없으면 제거 되며 재 생성여부에 따라 재 생성된다.

4) External Agent

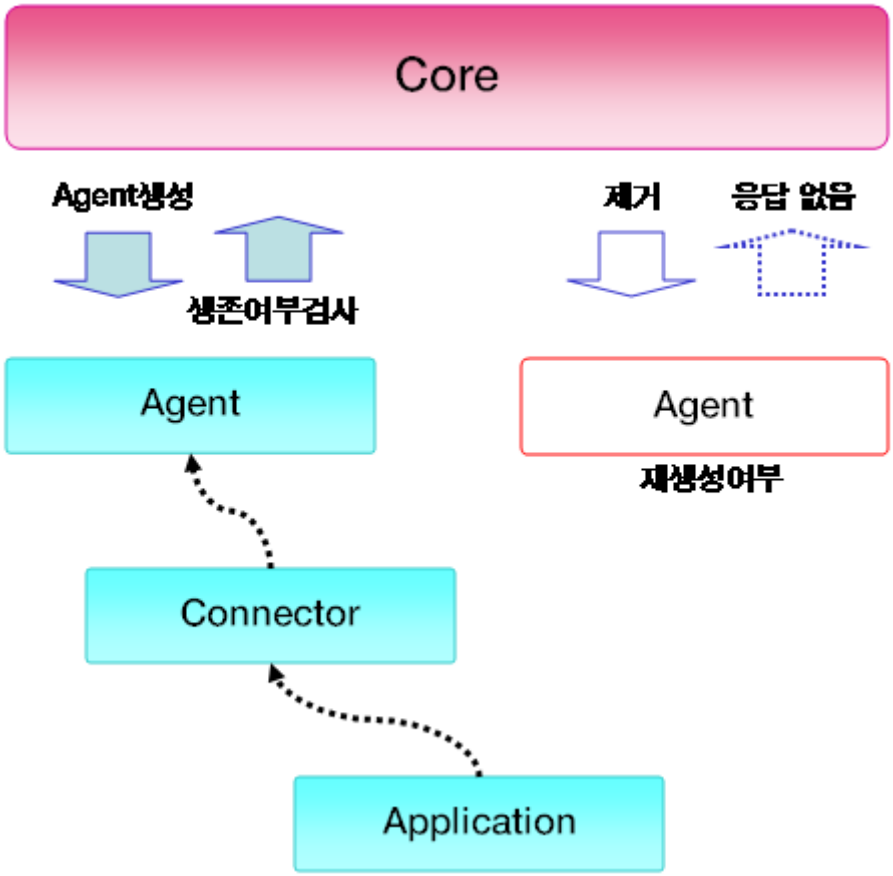


그림 5.11 External Agent 구성도

① 외부접속으로 생성이 처리된다.

② 외부와 접속이 종료되면 제거된다.

③ Connector를 통한 절차적 통신만을 수행한다.

④ 특정 시간 동안 응답이 없으면 제거된다.

5) Agent 설정

Agent 만들기

```
//-------------------------------------------------------------
//
// int addagent( int uuid, char *name, char type)
//-------------------------------------------------------------
int define_agent()
{
        add_agent(TEST,        "test",        'i',       LIFE_AGAINE);
        add_agent(VDS,         "vds",         'i',       LIFE_DONE);
        add_agent(OP,          "op",          'e',       LIFE_DONE);

        return 1;
}
```

Listener는 내부에서 자동으로 설정하도록 구성한다.

'i' : Internal Agent로 설정
'e' : External Agent로 설정

LIFE_AGAINE : 종료 시 재생성
LIFE_DONE : 종료 시 재생성하지 않음

5.2.3 Agent 구성하기

Agent는 미들웨어에서 실제 로직을 수행하는 주요구성이다. Agent는 실제 하나의 단독 프로그램과 같은 역할을 수행하며, 미들웨어 내부의 이벤트교환, 통신처리, 자원사용 등의 실제적인 작업을 담당한다. 또한, 미들웨어는 Agent를 추가하는 과정을 통해 확장과 특화된 미들웨어 구현이 가능하게 된다.

1) Agent 특징

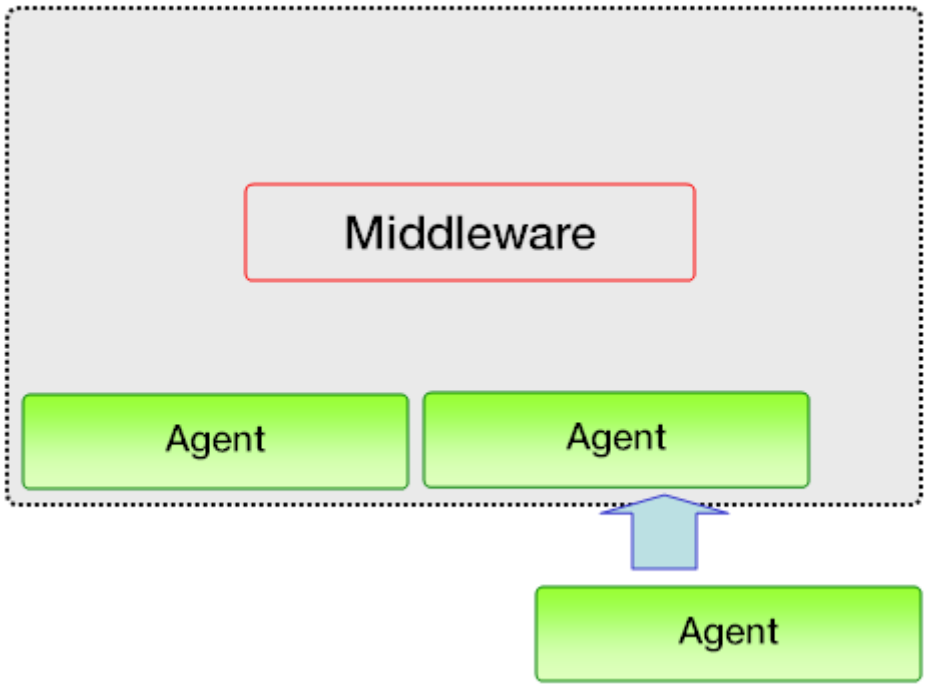


그림 5.12 Agent 특징

Agent는 미들웨어에서 다음과 같은 특징을 가진다.

① Agent는 Plug-in되는 모듈이다.

Agent를 구성하고 사용자 라이브러리를 배포하는 것만으로 미들웨어는 무한히 확장되는 구조를 가진다.

② Agent는 초기화, 반복, 이벤트처리, 통신처리, 종료 등의 사이클(cycle)을 가진다.

미들웨어의 기본적인 이벤트 처리와 통신처리, 자원관리의 기능을 상속받는 다기능 객체이다.

③ Agent는 독립된 객체이다.

미들웨어 환경내에서 Agent는 자원을 공유하지만 객체화 되어 있으므로 객체의 오류는 안정성에 영향을 주지 않도록 설계되어 있다.

2) Agent 기능

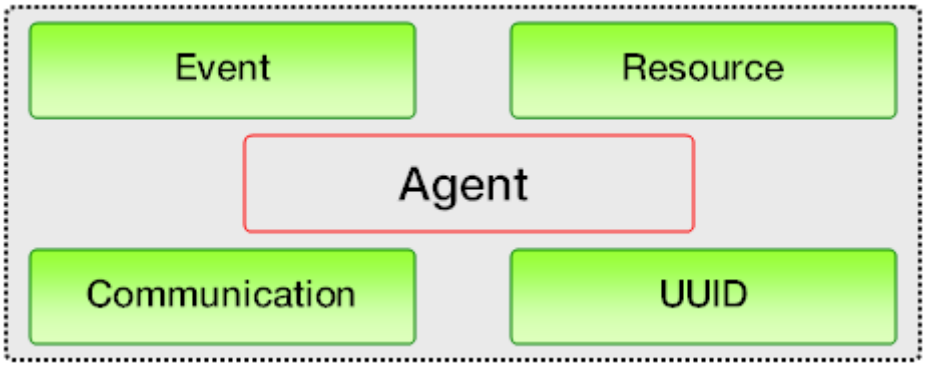


그림 5.13 Agent 기능

Agent는 기본적으로 관리명칭인 UUID를 가지고 다음과 같은 기능을 포함한다.

① 이벤트

자신에게 전달된 메시지를 처리하는 핸들러(msg_event)를 가진다.

또한 자신이 다른 agent와 미들웨어에게 이벤트를 발생시키는 권한을 가진다.

② 리소스

공유 리소스를 사용하고 새로운 리소스를 생성하고 이를 공유하는 기능을

가진다.

add_resource, del_resource, get_resource, put_resource

③ 통신

외부통신이 가능한 Agent(External Agent)는 외부에서 자신에게 전달된 정보를 처리하는 핸들러(comm_event)를 가진다.

3) Agent 만들기

Agent 만들기-1
[inc]$ ls agent.h child.h common.h core.h include.h mem.h pack.h resource.h server.h table.h uuid.h boss.h client.h config.h event.h listener.h misc.h proxy.h seq.h sig.h user.h [lib]$ ls libOFUE.so libENV.so makefile user.cc

Agent를 만들기 위해서는 먼저 미들웨어가 설치되어 있어야 한다.

user.h와 user.cc의 두 개의 파일을 이용하여 미들웨어에 Agent를 추가하고 관리한다.

1) user.h에서는 Agent의 관리명칭인 uuid 명칭을 부여한다.

2) user.cc에서는 Agent를 미들웨어에 추가하고 기본 핸들러 함수를 구성한다.

Agent 만들기-2
typedef enum

```
{
/*************************************************************
* 필요한 Agent를 정의하세요.
* MAIN, LISTENER는 예약(RESERVED)되어 있습니다.
* Internal Agent 정의
* 범위 2~98번까지
*************************************************************/
R_NONE,
R_MAIN,
TEST,
R_LISTENER = 99,
/*************************************************************
* External Agent 정의
* 범위 100~255번까지
*************************************************************/
OP,
R_UUID_NUMS = 256
}
em_useragent;
```

ofue/inc/user.h를 편집기로 열어서, em_useragent의 열거형 자료를 찾는다. 그리고, R_MAIN과 R_LISTENER사이에 TEST라는 새로운 Agent를 등록한다. Agent는 외부와 내부로 나누어지게 된다. R_LISTENER전에 기술된 Agent는 내부로 인식이 되어 미들웨어의 기동과 더불어 자동으로 실행되며, R_LISTENER이후에 등록된 Agent는 외부의 접속이 있을 때만 생성되는 특징을 가진다.

Agent 만들기-3

```
/*************************************************************
*
* 사용자 정의 함수
*
*************************************************************/
int test_after_start ( int uuid );
void test_before_stop ( int uuid );
int test_after_loop ( int uuid );
void test_msg_event ( int uuid, st_event e );
void test_comm_event ( int uuid, int key, char *tablename );
```

test라는 접두사가 붙는 명칭으로 핸들러를 선언한다.
위 함수의 리턴 값과 인수는 미들웨어에서 예약되어 있는 형태이므로 반드시 같은 형태로 선언하여야 한다. 반드시 접두사 외에는 이름이 동일하여야 정상적으로 동작하므로 규칙을 지켜야 한다.

Agent 만들기-4

```
/*************************************************************
*
* int addagent(int uuid, char *name, char type);
*
*************************************************************/
int define_agent()
{
    add_agent( TEST,    "test",    'i',    LIFE_AGAINE );
    add_agent( VDS,     "vds",     'i',    LIFE_DONE );
```

```
    add_agent( OP,      "op",   'e',   LIFE_AGAINE );

    return 1;
}
```

ofue/inc/user.c를 편집기로 열어서, define_agent() 함수에 새로운 Agent를 추가한다.

```
add_agent( TEST,   "test",   'i',   LIFE_AGAINE );
```

Internal Agent의 경우는 Listener의 앞에 작성되어야 하며 이 순서에 따라 Agent가 생성된다.

① TEST : user.h에 선언한 열거형 명칭을 부여한다.

② "test" : 함수 정의 시 사용된 접두어를 기술한다.

③ 'i' : Internal Agent임을 등록한다. 'e'는 External Agent선언시 사용된다.

④ 'LIFE_AGAINE', 'LIFE_DONE'은 종료시 재시작 여부를 설정한다.

Agent 만들기-5

```
/*************************************************************
*
* 사용자 정의 함수
*
*************************************************************/
int test_after_start ( int uuid )
{
    printf("Hello test agent!!\n");
}
void test_before_stop ( int uuid )
```

```
{
}
int test_after_loop ( int uuid )
{
}
void test_msg_event ( int uuid, st_event e )
{
}
void test_comm_event ( int uuid, int key, char *tablename )
{
}
```

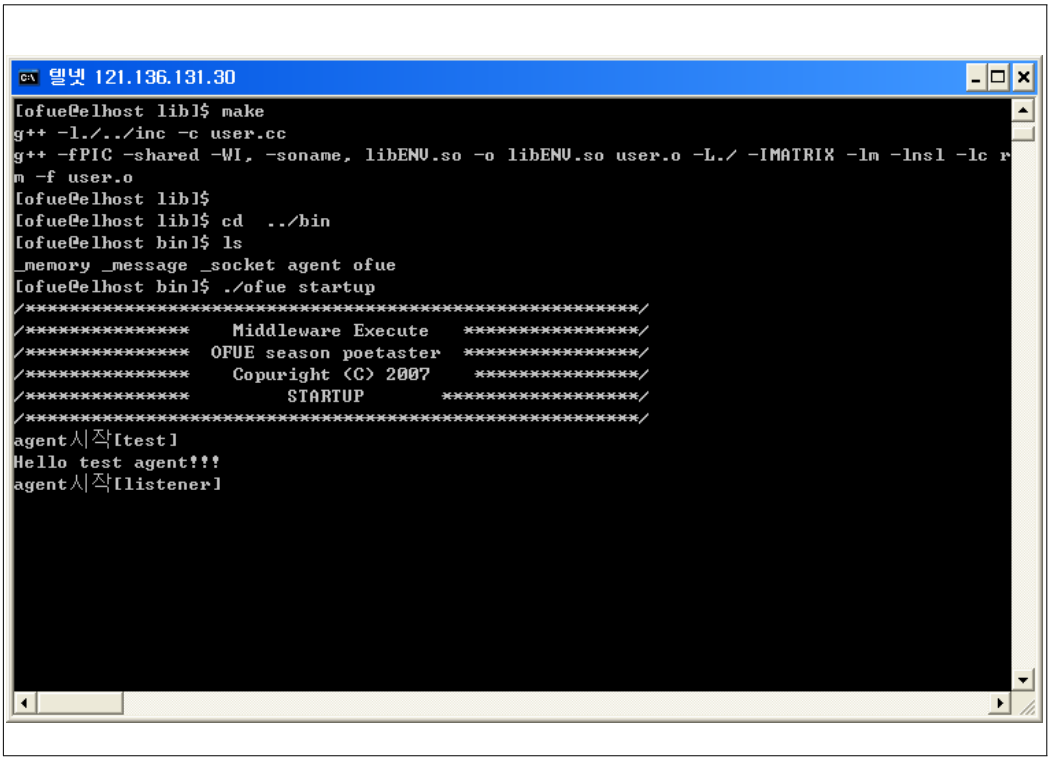
신규로 추가한 test에 대한 핸들러를 기술한다.

앞에서 user.h에 선언했던 함수에 대한 실제 루틴을 작성한다.

agent가 생성되면 Hello test agent!! 를 출력한다.

int test_after_start (int uuid)
- 정상적으로 agent가 실행되면 처리되는 핸들러
void test_before_stop (int uuid)
- 정상적으로 agent가 종료되면 처리되는 핸들러
int test_after_loop (int uuid)
- 미들웨어가 agent의 상태를 검사할 때 처리되는 핸들러
void test_msg_event (int uuid, st_event e)
- agent간에 메시지를 수신받을 때 처리되는 핸들러
void test_comm_event (int uuid, int key, char *tablename)
- 외부 통신을 통해 정보가 전달될 때 처리되는 핸들러

Agent 실행


텔넷 121.136.131.30
[ofue@elhost lib]$ make
g++ -l./../inc -c user.cc
g++ -fPIC -shared -WI, -soname, libENV.so -o libENV.so user.o -L./ -IMATRIX -lm -lnsl -lc r
m -f user.o
[ofue@elhost lib]$
[ofue@elhost lib]$ cd ../bin
[ofue@elhost bin]$ ls
_memory _message _socket agent ofue
[ofue@elhost bin]$ ./ofue startup
/**********   Middleware Execute   **********/
/********** OFUE season poetaster **********/
/**********   Copuright (C) 2007   **********/
/**********   STARTUP   **********/
agent시작[test]
Hello test agent!!!
agent시작[listener]

5.2.4 Event 구성하기

Event는 Agent간의 내부통신을 처리하는 구성이다. Agent들은 상태나 처리 내용을 다른 Agent에게 통보하거나 통보 받는 과정에 따라 업무를 진행하도록 설계될 수 있다. 이러한 내부 통신을 처리하기 위한 Event는 전혀 다른 Agent간에 자료를 공유하거나 특정기능을 호출하도록 지원한다.

1) Event 특징

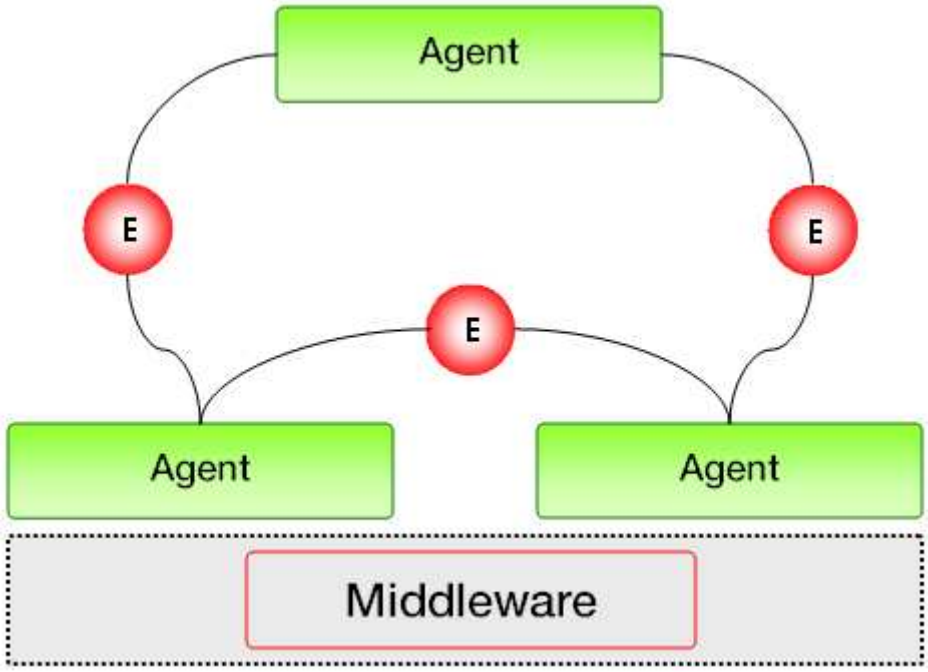


그림 5.14 Event 특징

Event는 미들웨어에서 다음과 같은 특징을 가진다.

① Event는 Agent간의 신호체계이다.

Event는 개별 Agent간에 상태나 정보를 전송하는 신호체계이다. Agent간에 분담되는 작업을 서로에게 통보하고 이를 송수신한다.

② Event는 수신자, 발신자, 송수신권한, 정보 등을 가진다.

Event는 수신과 발신의 Agent가 정해져 있으며 이를 여러 Agent가 수신하거나 발신할 수 있는 설정이 가능하다. 또한 여러 형태의 자료를 이벤트를 통해 전달 할 수 있다.

2) Event 기능

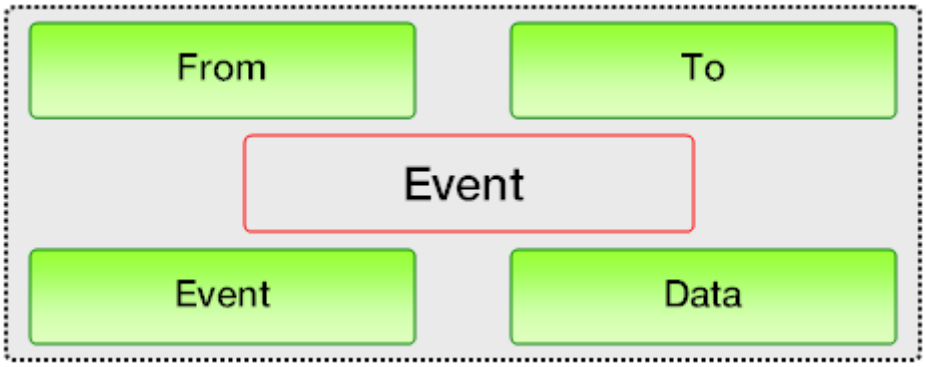


그림 5.15 Event 기능

Event는 관리 명칭을 가지며 권한에 따라 송수신을 지정할 수 있다.

① from

송신자에 대한 uuid를 가진다.

② to

수신자에 대한 uuid를 가진다.

③ event

명칭을 가지며 수신 시 이벤트 핸들러(comm_event)를 통해 통보된다.

④ data

일정크기의 정수형이나 문자형을 실어 보낼 수 있다.

3) Event 만들기

Event 만들기-1

```
typedef enum
{
    EV_USEREVENT = 100;
/*************************************************************
* 필요한 이벤트를 정의하세요.
* 범위 101~65535번까지
*************************************************************/
    EV_TEST
}
em_userevent;
```

ofue/inc/user.h를 편집기로 열어서, 내용 중에 em_userevent의 열거형 자료를 찾아 EV_USEREVENT 다음에 EV_TEST라는 새로운 Event를 등록한다.
100번 이내의 Event는 시스템 이벤트로 예약이 되어 있으므로 주의를 한다.

Event 만들기-2

```
/*************************************************************
*
* int addevent(int eventno, int inout, int uuid);
* RECV_GRANT, SEND_GRANT
*************************************************************/
int define_event()
{
     add_event( EV_TEST,    TEST,    SEND_GRANT );
     add_event( EV_TEST,    TEST,    RECV_GRANT );
```

```
    return 1;
}
```

ofue/inc/user.c를 편집기로 열어서, define_event() 함수에 새로운 Event를 추가한다.

```
    add_event( EV_TEST,   TEST,   SEND_GRANT );
    add_event( EV_TEST,   TEST,   RECV_GRANT );
```

EV_TEST는 user.h에 선언한 열거형 명칭을 부여한다.
TEST는 Agent 명칭으로 define_agent에 추가된 Agent명을 기술한다.
SEND_GRANT는 송신권한, RECV_GRANT 수신권한을 부여한다. 만약, 송신권한과 수신권한이 없으면 Event는 전송되지 않는다.

Event 만들기-3

```
/*****************************************************************
*
* 사용자 정의 함수
*
*****************************************************************/
int test_after_start ( int uuid )
{
    st_event e;
    printf("Hello test agent!!\n");
    e.data.i = 100;
    e.to = TEST; e.from = uuid; e.type = EV_TEST; tell(e);
}
void test_before_stop ( int uuid )
{
```

```
}
int test_after_loop ( int uuid )
{
}
void test_msg_event ( int uuid, st_event e )
{
      printf("%s [%d] data [%d] \n", who(uuid), e.type, e.data.i);
}
void test_comm_event ( int uuid, int key, char *tablename )
{
}
```

TEST에 Event를 발생시킨다.

```
st_event e;      // 전송구조체를 선언한다.
e.data.i = 100; // 전송값을 정수형으로 100을 대입한다.
e.to = TEST; e.from = uuid; e.type = EV_TEST; tell(e);
```

먼저 test_after_start에서 생성 시 자신에게 EV_TEST이벤트를 보내고 여기에 100값을 실어 전송하도록 기술한다.

TEST에서 test_msg_event(int uuid, st_event e) 핸들러를 작성한다.

```
printf("%s [%d] data [%d] \n", who(uuid), e.type, e.data.i);
```

수신 받은 Event 번호와 값을 화면에 출력한다.

Event 만들기-4

```
typedef struct
{
    int to;
    int from;
    int type;
```

```
    union
    {
        int i;
        char s[ DATA_SIZE ]
    }
    data;
}
st_event;
```

Event 구조체는 위의 사용명령어는 tell(st_event e)이라는 미들웨어 제공함수를 사용한다.

st_event e;

e.to : 전송되는 수신 Agent uuid를 기술한다.

e.from : 전송하는 발신 Agent uuid를 기술한다.

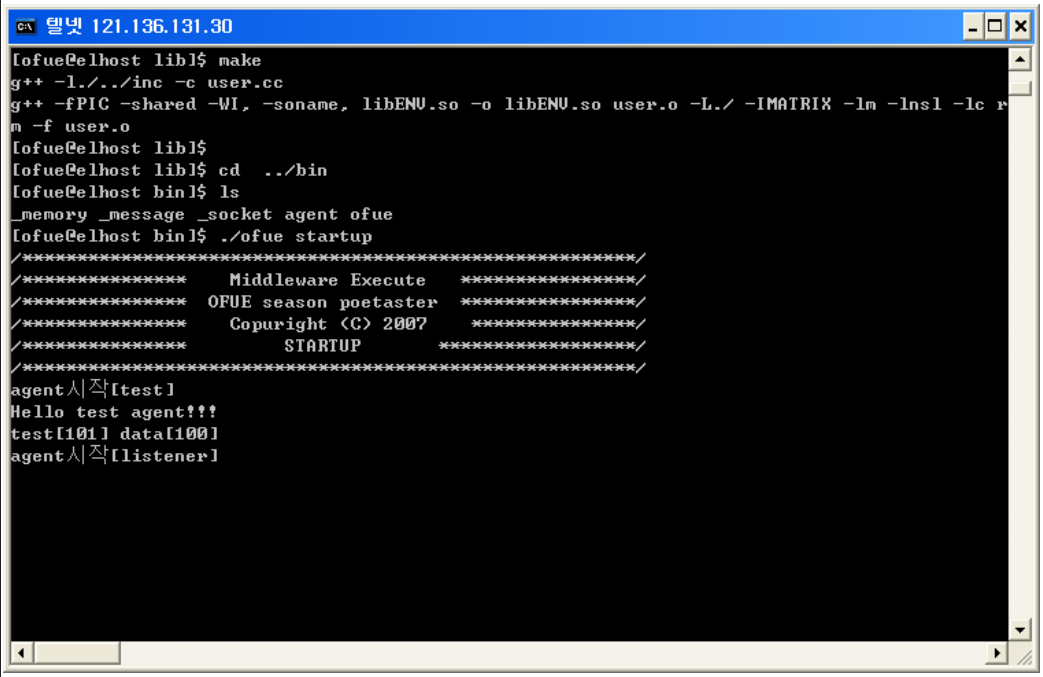
e.type : 전송하고자 하는 Event 명칭을 기술한다.

e.data.i : 정수형 자료

e.data.s : 실수형 자료

tell(e) : 전송구조체에 해당하는 목적지로 이벤트를 전송한다.

Event 실행


텔넷 121.136.131.30
[ofue@elhost lib]$ make
g++ -l./../inc -c user.cc
g++ -fPIC -shared -WI, -soname, libENV.so -o libENV.so user.o -L./ -IMATRIX -lm -lnsl -lc r
m -f user.o
[ofue@elhost lib]$
[ofue@elhost lib]$ cd ../bin
[ofue@elhost bin]$ ls
_memory _message _socket agent ofue
[ofue@elhost bin]$ ./ofue startup
/***/
/***************** Middleware Execute ****************/
/***************** OFUE season poetaster ****************/
/***************** Copuright (C) 2007 ****************/
/***************** STARTUP ****************/
/***/
agent시작[test]
Hello test agent!!!
test[101] data[100]
agent시작[listener]

5.2.5 Resource 구성하기

Resource는 Agent간의 공유자원을 설정하는 구성이다. 미들웨어 내부에서 처리될 자원공간을 처리하는 메모리 영역이다. 특정 처리된 내용을 DB에 기록하거나 다른 Agent로 하여금 공유할 수 있도록 설계할 수 있다.

1) Resource 특징

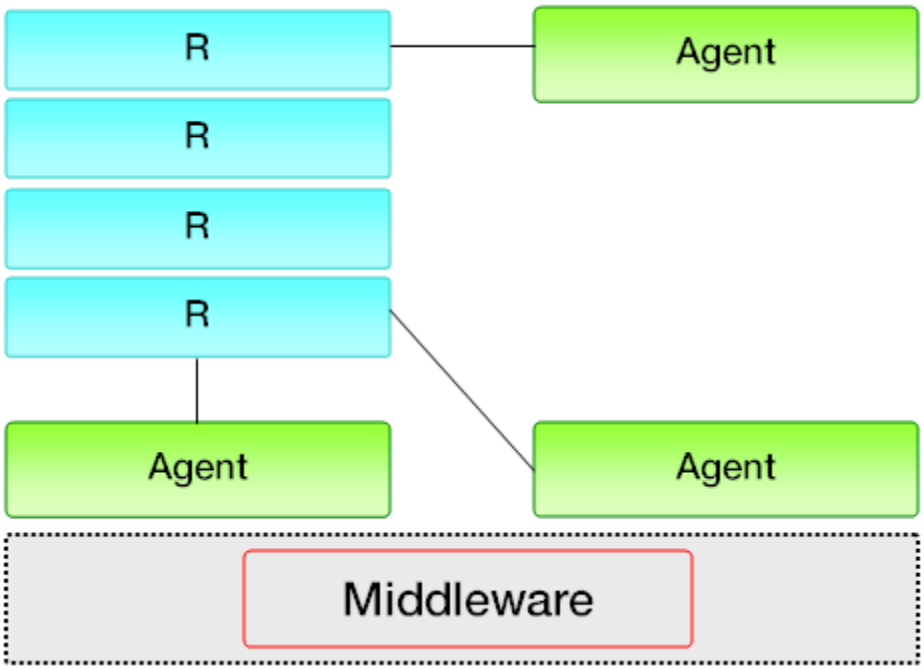


그림 5.16 Resource 특징

Resource는 미들웨어의 메모리 관리로 다음과 같은 특징을 가진다.

① Resource는 Agent에서 공유되는 자원이다.

Resource는 공유되어야 하는 자원의 집합이다. Agent간의 지정된 구조체 형태로 자원을 정의하고, Agent간에 자원을 공유할 수 있다.

② Resource는 초기 설정 시 생성되거나 동적으로 생성이 가능하다.

Resource는 초기 설정에 의해서 생성된다. 메모리 관리와 자원의 효율을 위해 일시적인 생성과 전달·처리를 구성할 수 있다.

2) Resource 기능

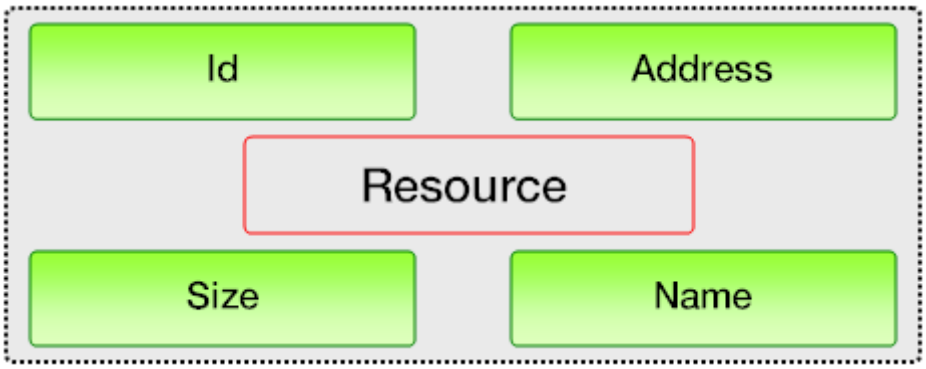


그림 5.17 Resource 기능

Resource는 관리 명칭에 의해서 읽거나 쓰기를 처리할 수 있다.

① id

메모리 관리 번호를 가진다.

② address

메모리의 참조 가능한 주소를 가진다.

③ size

실제 메모리의 크기를 가진다.

④ name

리소스 관리 명칭으로 사용자에 의해 부여된다.

3) Resource 만들기

Resource 만들기-1

```
typedef enum
{
/*************************************************************
* 필요한 리소스를 정의하세요.
* 범위 101~65535번까지
*************************************************************/
    RE_USERRESOURCE = 100,
    RE_TEST
}
em_userresource;
/*************************************************************
* 리소스 자료구조정의
*************************************************************/
typedef struct
{
    int test;
}
st_test;
```

ofue/inc/user.h를 편집기로 열어서 내용중에 em_userresource의 열거형 자료를 찾아 RE_USERRESOURCE 다음에 RE_TEST라는 새로운 Resource를 등록한다. 100번 이내의 Resource는 시스템 Resource로 예약이 되어 있으므로 주의를 한다. 그리고, Resource로 등록할 구조체(st_test)를 정의한다.

Resource 만들기-2

```
/*************************************************************
*
* int addresource(int resourceno, int size, int rows);
*************************************************************/
int define_resource()
{
     add_resource( RE_TEST,    sizeof(st_test),    100 );

     return 1;
}
```

ofue/inc/user.c를 편집기로 열어서, define_resource() 함수에 새로운 Resource를 추가한다.

```
     add_resource( RE_TEST,    sizeof(st_test),    100 );
```

RE_TEST는 user.h에 선언한 열거형 명칭을 부여한다.
sizeof(st_test)는 구성할 공유자원의 단위 크기이다. 100은 공유자원의 개수이다. 자원의 단위크기와 개수에 따라 공유될 메모리의 크기가 결정되게 된다.

Resource 만들기-3

```
/*************************************************************
*
* 사용자 정의 함수
*
*************************************************************/
int test_after_start ( int uuid )
{
     st_event e;
     st_test test[100];
```

```
        for(int i=0; i<100; i++)
        {
            test[i].test = i;
        }
        put_resource(RE_TEST, (void *)&test, 100);
        e.to = TEST; e.from = uuid; e.type = EV_TEST; tell(e);
    }
    void test_msg_event ( int uuid, st_event e )
    {
        st_test test[100];
        get_resource(RE_TEST, (void *)&test, 100);
        for(int i=0; i<100; i++)
        {
            printf("RE_TEST [%d] \n", test[i].test);
        }
    }
```

TEST에서 공유자원에 값을 넣고 이벤트를 발생시킨다.

```
    st_test test[100];
    for(int i=0; i<100; i++)
        {
            test[i].test = i;
        }
        put_resource(RE_TEST, (void *)&test, 100);
```

먼저 test_aftet_start에서 구조체를 정의하고 0~99까지의 정수값을 넣고 이를 리소스에 put_resource 함수를 이용하여 기록한다.

TEST에서 test_msg_event(int uuid, st_event e)핸들러를 작성한다.

```
        st_test test[100];
        get_resource(RE_TEST, (void *)&test, 100);
        for(int i=0; i<100; i++)
```

```
        {
            printf("RE_TEST [%d] \n", test[i].test);
        }
```

이벤트를 수신받으면 Resource를 인출하여 화면에 값을 출력한다.

get_resource와 put_resource는 자료형 크기로 자원을 읽거나 쓰는 함수이다.

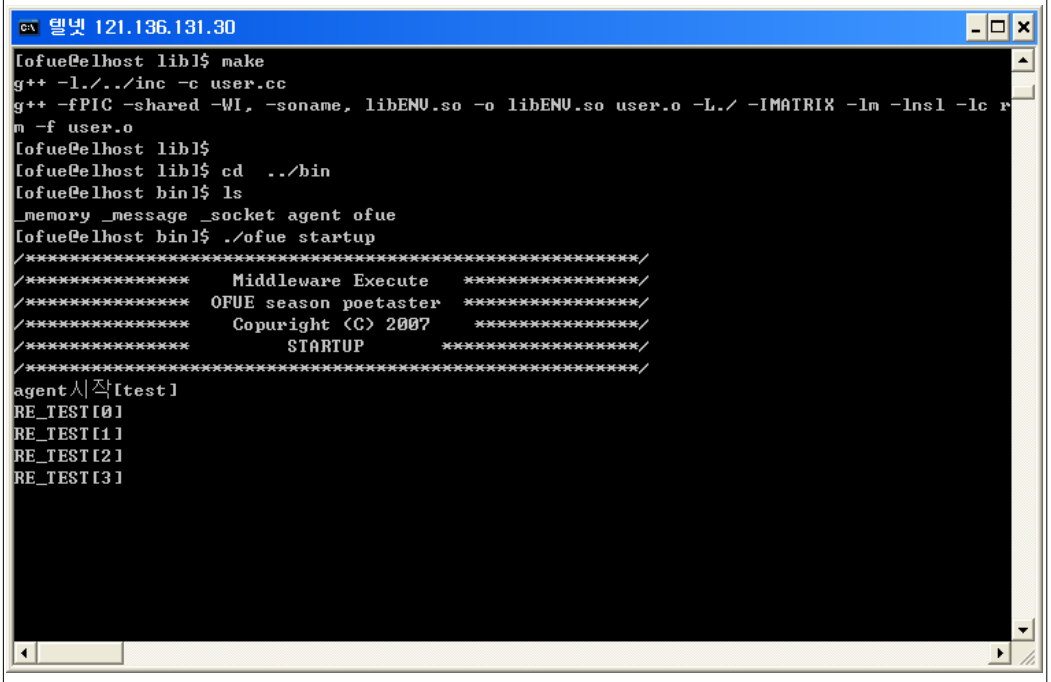
자료를 공유메모리에 적재할 때

put_resource(자원명칭, (void *)자료형 변수 주소, 자료의 개수);

자료를 공유메모리에서 읽을 때

get_resource(자원명칭, (void *)자료형 변수 주소, 자료의 개수);

Resource 실행



5.2.6 Connector 구성하기

Connector는 어플리케이션과 미들웨어의 통신을 설정하는 구성이다. 어플리케이션에서 미들웨어에 데이터셋 형태로 데이터를 전송한다. 어플리케이션에서 SQL을 사용하여 가상의 DB에 데이터를 입력하여 미들웨어로 전송할 수 있다.

1) Connector 특징

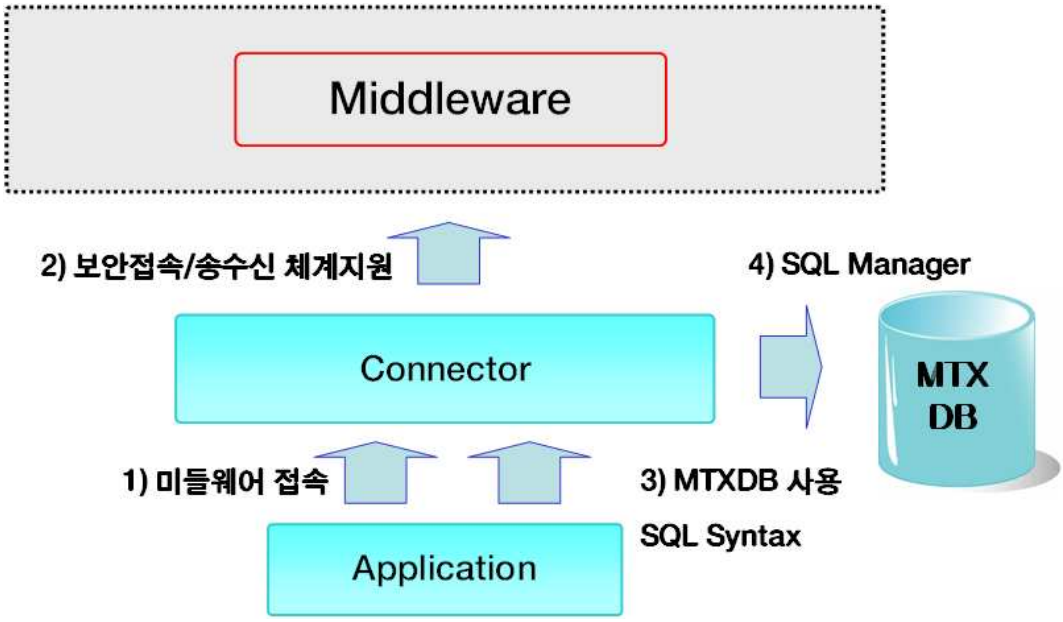


그림 5.18 Connector 특징

Connector는 다음과 같은 특징을 가진다.

① Connector는 미들웨어와 연결되어야만 서비스를 지원한다.

② 미들웨어에서 공개된 IP와 Port를 통해 접근하며, 보안절차에 따라 접속을 허가 받는다.

③ 송수신 Data는 Table명으로 접근이 가능하며 접근 함수를 제공한다.

2) Connector 기능

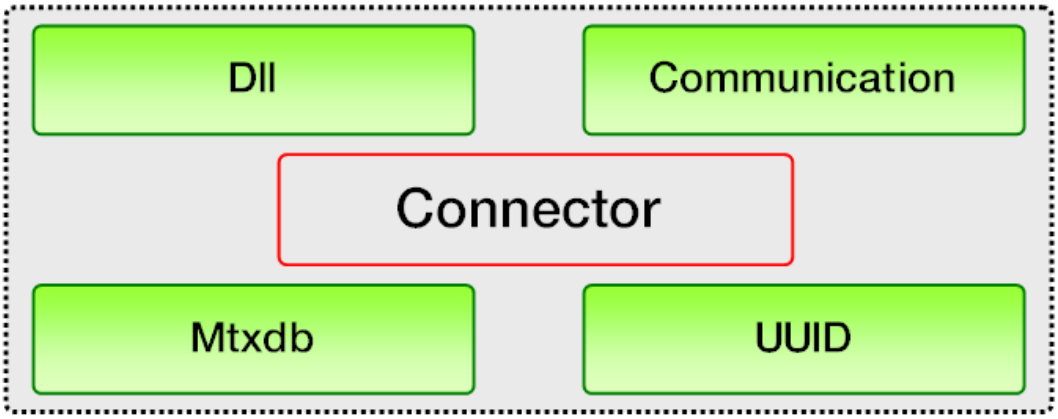


그림 5.19 Connector 기능

① Connector는 미들웨어와 통신하는 외부연계 기능과 Local DB인 Mtxdb를 포함하는 미들웨어이다.

- DLL : Connector는 dll형태로 제공된다.
- Communication : 외부연계를 위한 절차, 보안을 담당하며 통신유지 및 송수신을 처리한다.
- Mtxdb : SQL기반의 Local DB 기능을 제공한다. 별도의 DB없이도 DB 기능을 사용할 수 있다.
- UUID : 미들웨어와 통신이 개설된 후 운영되도록 설계되어 있다.

② Connector의 송수신 정보는 내부 테이블 형태의 체계를 가진다.

- Connector 송수신 정보는 조회, 순서접근, 삭제 등의 자료체계를 가진다.

③ 미들웨어의 지원

- open, exec, close의 3개의 기본 함수를 통해 다양한 SQL문의 처리를 지원한다.

3) Connector 사용하기

Connector 접속하기

```
#define uuid 101
WindowProc = ConnectorProc;
pipe_open(this->Handle, uuid, "211.222.345.123", 50000);
pipe_close(uuid);
```

pipe_open 명령을 통하여 미들웨어와 접속을 수행한다.

uuid는 미들웨어에서 External Agent에 대한 고유번호이다.

IP는 Ipv4형태로 문자열 형태로 지정한다.

PORT는 미들웨어에서 외부접속을 허가한 포트번호이다.

pipe_close함수는 uuid를 지정하고 Connector 통신을 종료한다.

Connector 전송자료 구성 - 테이블생성

```
long ret;
long opcode = 1234;
ret = mw_tbcreate( uuid, opcode);
ret = mw_addcolum ( uuid,  opcode, "col1", 'c', 50 );
ret = mw_addcolum ( uuid,  opcode, "col2", 'c', 50 );
ret = mw_addcolum ( uuid,  opcode, "col3", 'f',  0 );
ret = mw_addcolum ( uuid,  opcode, "col4", 'i',  0 );
ret = mw_addcolum ( uuid,  opcode, "key", 'i',  0 );
```

① mw_tbcreate

전송하고자 하는 자료의 이름을 구성하고 생성한다.

opcode는 전송되는 테이블의 번호이다.

② mw_addcolum

전송할 자료의 구조를 생성한다.

- opcode : mw_tbcreate로 생성된 테이블번호
- col1 : 각 컬럼의 이름지정
- 'c', 'f', 'i' : 컬럼의 타입을 지정(c : 문자열, f : 실수, i : 정수)
- 50 : 컬럼의 길이를 지정(c 타입일 경우만 유효하다.)

Connector 전송자료 구성 - 자료입력

```
long ret, key;
ret = mw_insert ( uuid, opcode);
ret = mw_slong ( uuid, opcode, "key" ,  1 );
ret = mw_stext ( uuid, opcode, "col1", "첫번째", 6 );
ret = mw_stext ( uuid, opcode, "col2", "0001",   4 );
ret = mw_sdouble ( uuid, opcode, "col3", 1212.1  );
ret = mw_slong (  uuid, opcode, "col4", 212 );
key = mw_post ( uuid, opcode );

ret = mw_insert ( uuid, opcode );
ret = mw_slong ( uuid, opcode, "key" ,  2 );
ret = mw_stext ( uuid, opcode, "col1", "두번째", 6 );
ret = mw_stext ( uuid, opcode, "col2", "0002",   4 );
ret = mw_sdouble ( uuid, opcode, "col3", 3434.4  );
ret = mw_slong ( uuid, opcode, "col4", 234 );
key = mw_post ( uuid, opcode );
```

① mw_insert

새로운 행을 추가하기 위한 명령어이다.

② mw_slong

"key"라는 컬럼에 정수형 값 2를 입력한다.

③ mw_stext

"col1"이라는 컬럼에 문자열 값 “첫번째”를 입력한다.

④ mw_post

채운 값을 테이블에 저장한다. key는 저장된 행의 위치값이다.

Connector 전송자료 구성 - 자료확인

```
long key, a, s;
double b;
char c[50];

key = mw_begin(uuid, opcode);
mw_glong ( uuid, opcode, "key" , &a );
mw_gtext ( uuid, opcode, "col2", c, &s);
mw_gdouble ( uuid, opcode, "col3", &b);
.
.
key = mw_end(uuid, opcode);
key = mw_prev(uuid, opcode, key);
key = mw_next(uuid, opcode, key);
```

① mw_begin

구성된 테이블의 값을 확인하기 위한 행 위치를 구하는 명령이다. 여러행 중 맨 처음 행의 값을 가져온다.

② mw_glong

"key"라는 컬럼에 정수형 값을 a라는 변수에 넣는다.

③ mw_gtext

"col1"이라는 컬럼에 문자열 값을 c라는 변수에 넣고, s라는 변수에 c변수의 길이를 넣는다.

④ mw_end, mw_prev, mw_next

각각 값을 확인하기 위한 행 위치 이동명령이다.

Connector 전송자료 구성 - 전송하기

long stat;
stat = mw_tx_table(uuid, opcode);
mw_tbdrop(uuid, opcode);

① mw_tx_table

구성된 테이블형식의 자료를 전송한다. stat는 성공 1, 실패 0을 리턴한다.

② mw_tbdrop

전송 후 생성했던 테이블을 제거한다.

MTXDB 사용하기

mtxdb *db;
char *ErrMsg = 0;
int rc;
rc = mtxdb_open("test.db", &db);
rc = mtcdb_exec(db, "select * from test", callback, 0, &ErrMsg);
mtxdb_close(db);

① mtxdb *db

mtxdb를 사용하기 위한 자료형을 선언한다. 해당 핸들러를 이용하여 자료를 처리할 수 있다.

② mtxdb_open

"test.db"는 Local db파일명을 지정한다. db는 mtxdb *db로 선언한 핸들러

변수명이다.

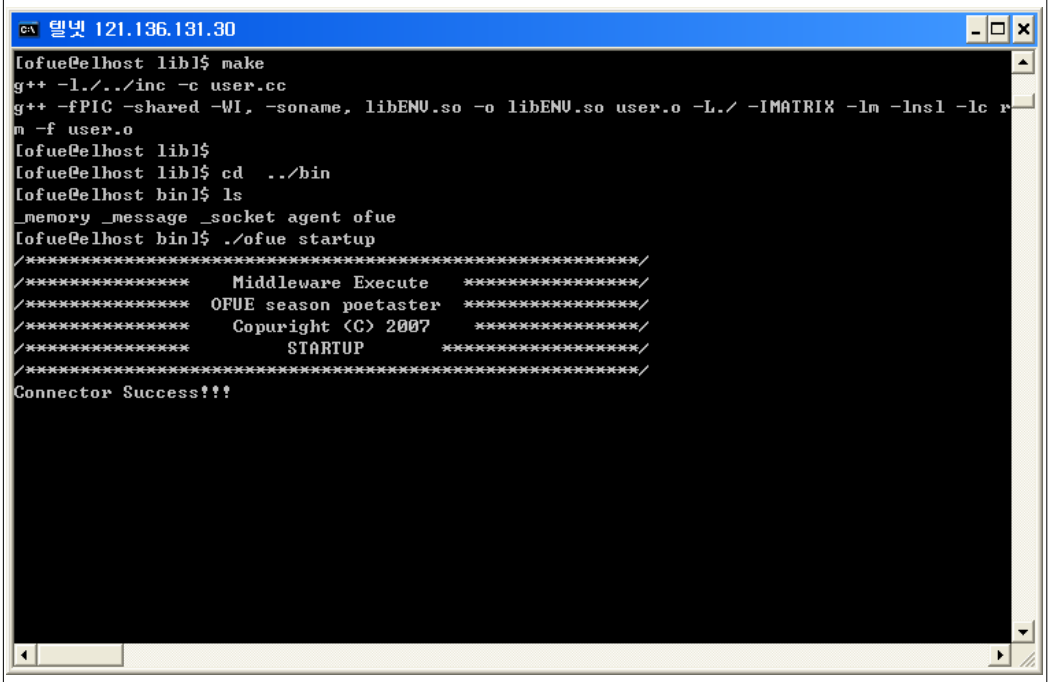
③ mtxdb_exec

열려진 mtxdb에 SQL명령을 전송하는 명령어이다.

④ mtxdb_close

열려진 mtxdb를 닫는다. db는 mtxdb_open으로 열려진 핸들러 변수이다.

Connector 실행



5.2.7 미들웨어 구성 내역

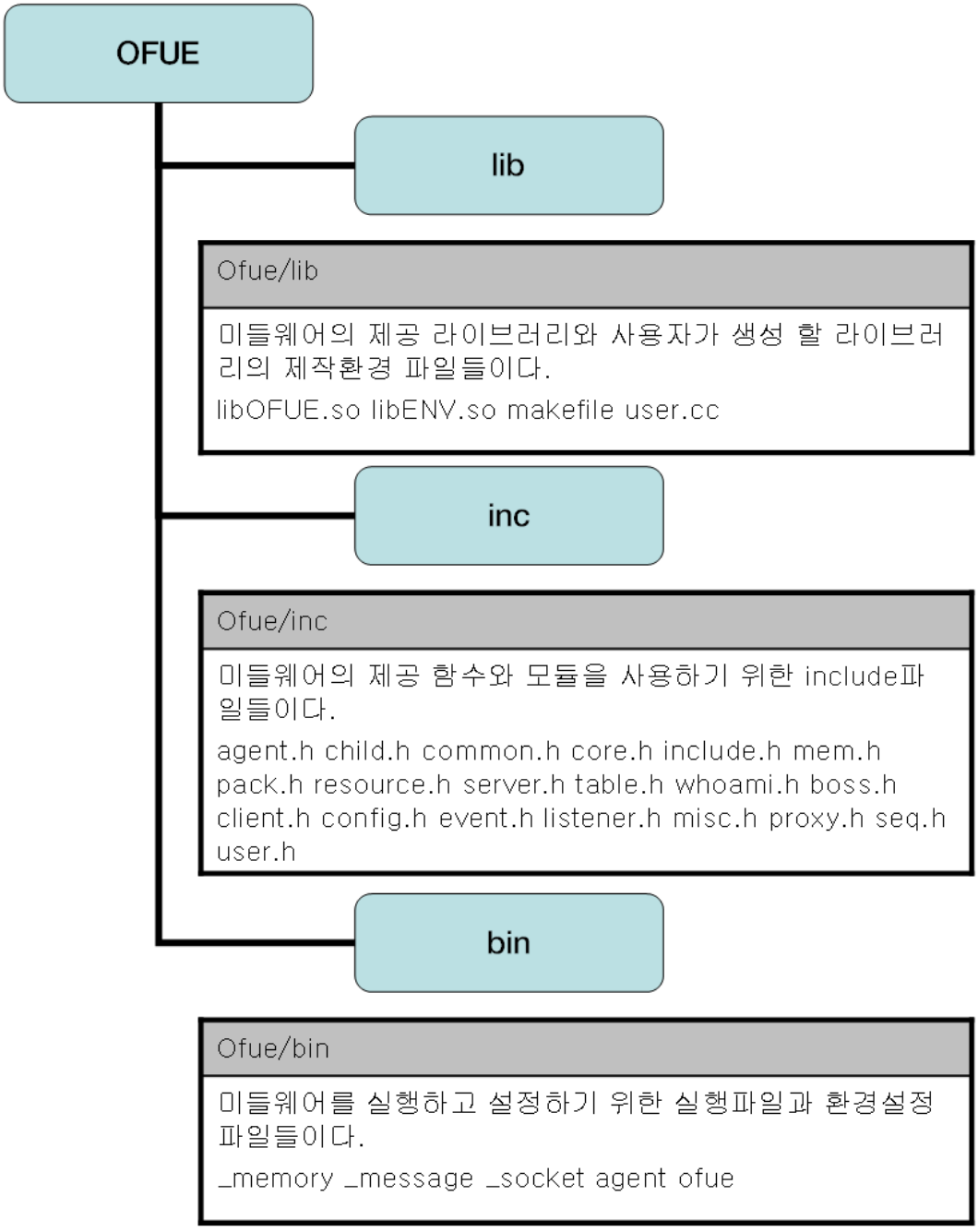


그림 5.20 미들웨어 구성 내역

5.3 미들웨어 적용 및 실행결과

본 절에서는 5.2절에서 구현한 미들웨어의 성능테스트를 위해 도시의 공간정보를 기반으로 공공정보서비스와의 인터페이스를 통해 도시내에서 발생하는 다양한 재난/재해, 기상, 환경, 교통 등의 다양한 이벤트와 u-City 시설물들의 상태 등을 통합 운영자 접속환경을 통해 관제, 운영 및 관리할 수 있는 통합관제플랫폼을 구축하고, 그 성능을 테스트한 결과를 제시하도록 하겠다.

5.3.1 통합관제플랫폼 구축

본 연구에서 통합관제플랫폼을 구현하기 위해 사용한 구현PC의 운영체제는 WindowXP환경이고, 통합UI 구축을 위해 HTML, JAVASCRIPT, JSP, JAVA, AJAX 등의 프로그래밍언어를 사용하였고, 구현툴은 이클립스3.1, WAS는 웹로직8.1.6, DBMS는 오라클10g, 구현서버 OS는 unix, 구현서버 장비는 Sun fire V480, Dell powerredge 2950을 사용하였다.

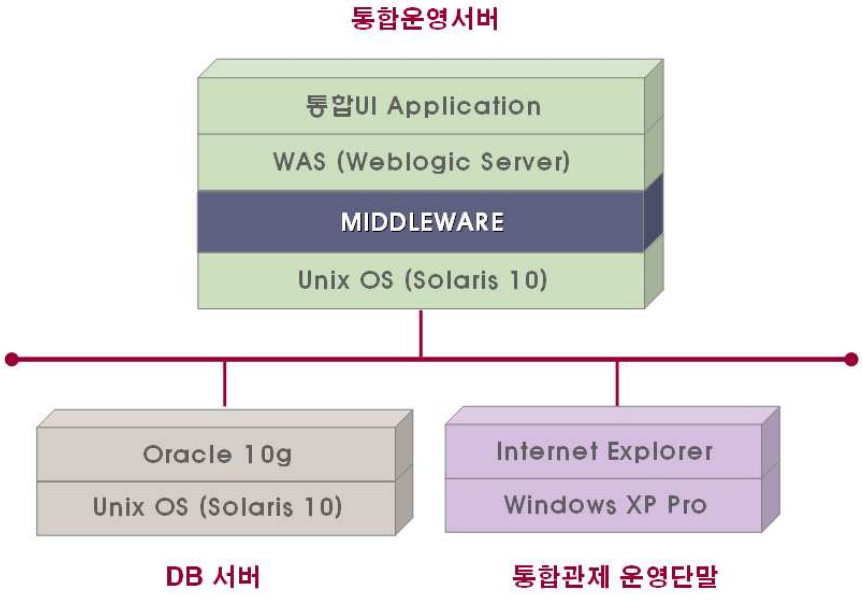


그림 5.21 통합관제플랫폼 S/W구성도

통합관제플랫폼의 시스템 구성은 정보수집파트, 통합연동파트, 통합UI파트의 3개 파트로 구성하였다.

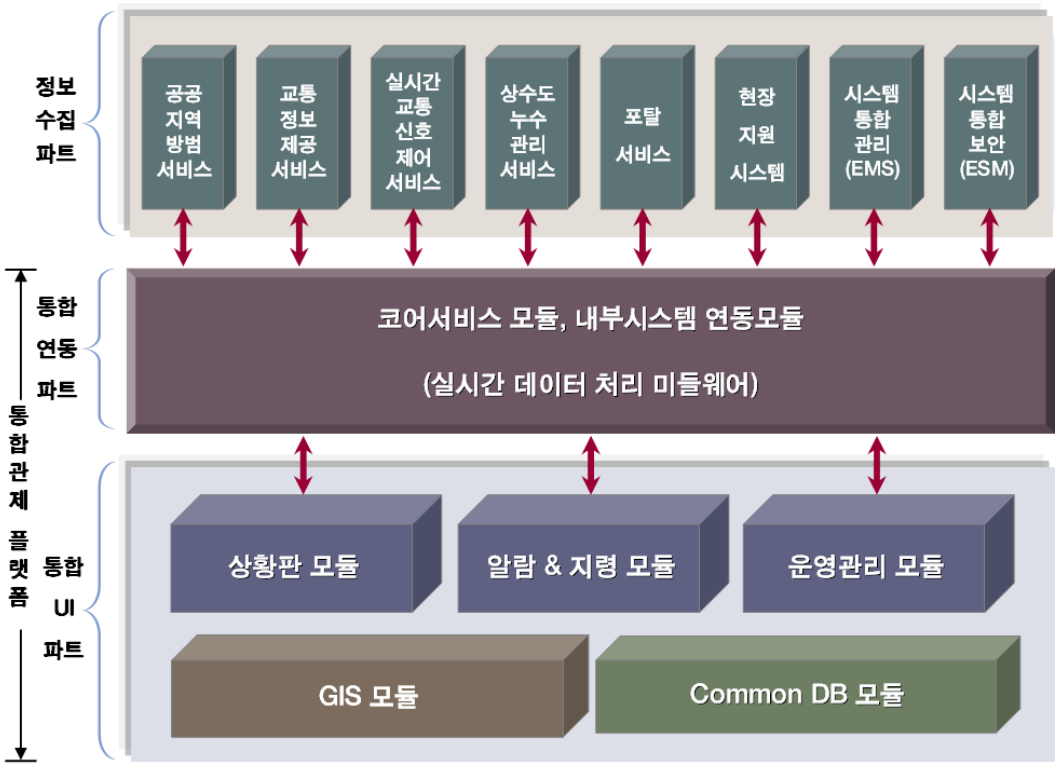


그림 5.22 통합관제플랫폼 시스템 개요도

① 정보수집

도시 내 발생하는 이벤트와 데이터를 통합관제플랫폼으로 전송하며, 플랫폼과 Connector를 이용한 데이터 생성 및 송/수신 역할을 수행하는 정보수집서버이다. 본 논문에서는 시뮬레이션 환경을 위해 도시 내 5대 공공서비스(방범, 교통정보, 신호제어, 상수도, 포탈)를 기반으로 정부수집부분을 구성하였으며, 시설물관리를 위한 현장지원시스템과 시스템 통합관리를 위한 EMS와 시스템 통합 보안을 위한 ESM 등 총 8개의 정보수집서버를 구성하였다.

② 통합연동

정보수집서버 및 통합UI와 인터페이스에 기반한 메시지 송/수신 및 프로세스 기능을 수행하며, 통합관제를 위한 데이터 분석 및 가공 로직을 수행하는 실시간 데이터 처리 미들웨어이다.

③ 통합UI

통합연동과 인터페이스를 통해 도시 내 상황 및 시설물 정보를 통합 사용자 접속환경을 통해 관제 및 운영할 수 있는 환경을 제공할 수 있도록 웹기반으로 구성하였고, GIS를 기반으로 실시간 정보 표출 기능 구현으로 효율적인 관제를 할 수 있도록 구성하였다.

통합관제플랫폼의 데이터 흐름은 정보수집서버로부터 데이터를 수신하는 INBOUND와 통합UI로부터 데이터를 조회하는 OUTBOUND로 나누어져 있다.

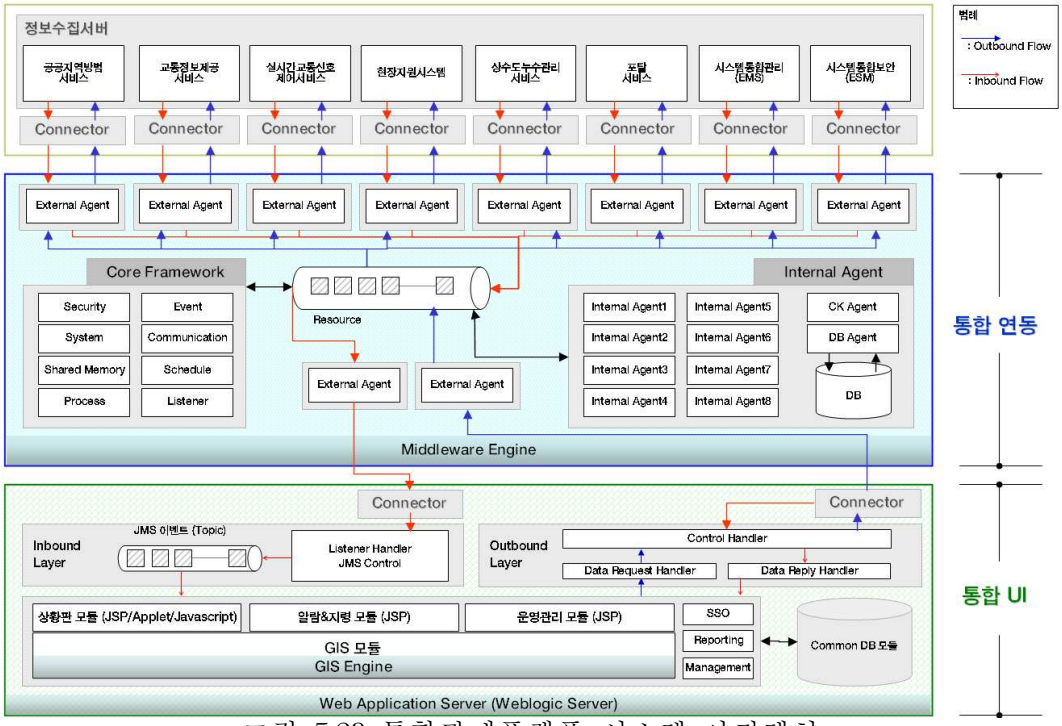


그림 5.23 통합관제플랫폼 시스템 아키텍처

본 논문에서는 도시 내에서 발생하는 이벤트와 데이터의 시뮬레이션을 위해 8개의 정보수집서버를 구성하였고, 다음 표 5.3에서 설명하였다.

표 5.3 정보수집서버 시스템 설명

정보수집서버		시스템 설명
공공정보서비스	공공지역방범서비스	▪ u-City 전역에 방범용 CCTV를 설치하여 도시의 사건, 사고를 사전에 예방 및 감시하여 도시 입주민들이 안심하고 거주할 수 있는 환경을 제공하는 서비스이다.
	교통정보제공 서비스	▪ 도로상의 교통상황을 수집, 가공/처리한 정보와 외부기간(교통센터)과 연계, 취득한 교통정보(돌발상황, 혼잡상황, 소통정보, 주변도로 교통상황 등)를 도로전광표지(VMS), WEB등의 정보제공 매체를 통해 제공하는 서비스이다.
	실시간 교통신호 제어 서비스	▪ 교차로를 통과하는 차량의 흐름을 실시간으로 감지하여 신호주기 및 녹색등 시간조절등 신호체계를 관리하고, CCTV영상을 통한 교차로 소통상황 모니터링 및 링크 소통 정보를 수집하여 능동적으로 교통상황에 대처하는 서비스이다.
	상수도 누수관리 서비스	▪ 도시 내 공급되는 상수관로상의 유수율 향상과 누수발생지점의 관리를 통해 상시 풍부한 수돗물을 안정적으로 공급하고자 하는 서비스이다.
	포탈 서비스	▪ u-City의 교통, 기상 등 공공정보, u-City체험정보, 지역정보, 커뮤니티 등 지역주민들에게 다양하고 유익한 정보를 웹서비스 방식을 활용하여 제공하며, u-City의 관문역할을 하는 서비스이다.
현장지원시스템		▪ u-City에서 설치 및 운영되는 현장시설물(CCTV, VMS, VDS, 신호제어기, 유량계, 압력계 등)에 대한 관리 및 유지보수 업무를 지원하는 시스템이다.
시스템		▪ 통합관제센터 상황실 내부 시스템, 내부 네트워크 및

통합관리(EMS)	자가망에 대한 관리 환경을 제공하는 시스템이다.
시스템 통합보안(ESM)	▪ 통합관제센터 상황실 내부의 시스템 보안에 대한 보안 관제 기능을 수행하는 시스템이다.

표 5.4 통합관제플랫폼 인터페이스 내역

연동대상시스템		연동내역	송/수신 구분
공공정보서비스	공공지역 방범 서비스	CCTV동작/통신상태	수신
	교통정보제공 서비스	돌발상황 요약정보	수신
		시설물 고장정보	수신
		교통소통정보	수신
		돌발상황정보	수신
		혼잡상황정보	수신
		VMS표출정보	조회
		VMS상태정보	수신
		VMS메시지전송	송신
	실시간 교통신호 제어 서비스	VDS제어기 상태정보	수신
		신호제어기 상태정보	수신
		신호검지기 상태정보	수신
	상수도 누수관리 서비스	측정지점 유량 계측 데이터	조회
		측정지점 유압 계측 데이터	조회
		통신이상여부	수신
		누수예상 측정지점 알람 발생	수신
	포탈 서비스	사용자 상세정보	조회

		민원정보	수신
현장지원시스템		시설물 관리 담당자	조회
		시설물 현장지원 신규작업지령	송신
		시설물 현장지원 작업진행내역	조회
		시설물 현장지원 작업결과	수신
시스템 통합관리(EMS)		장애발생정보	수신
시스템 통합보안(ESM)		장애발생정보	수신

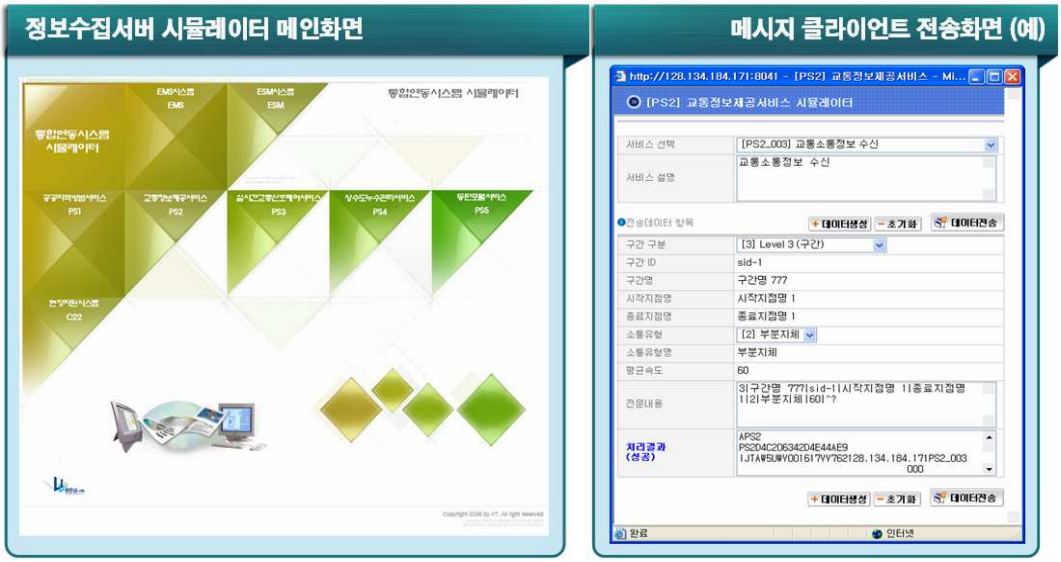


그림 5.24 정보수집서버 시뮬레이터 화면

통합UI시스템은 인터페이스 부분과 사용자UI 부분으로 나누어 구성하였으며, 인터페이스 부분은 통합연동과 데이터를 인터페이스하여 송/수신하는 부분이고, 사용자UI 부분은 운영자가 접속하여 상황정보를 모니터링하고, 이에 대해 조치할 수 있는 부분이다. 다음 그림 5.25에서 보는 바와 같다.

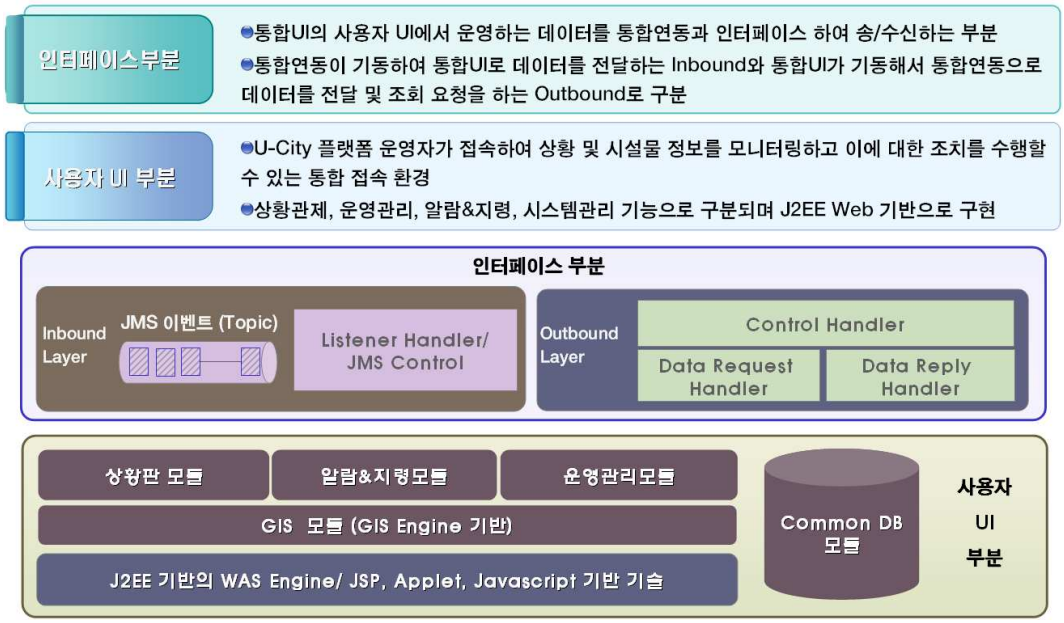


그림 5.25 통합UI시스템 구성내역

표 5.5 통합UI시스템 인터페이스 기능

구 분	기 능	비 고
Inbound Layer	▪ 통합연동으로부터 수신한 데이터에 대하여 통합UI화면에 데이터를 표출하기 위한 모듈 ▪ 수신한 데이터를 JMS Topic에 저장 기능 제공 ▪ APPLET에서는 Topic의 Subscribe로 연결되어 메시지를 받고 Hidden Control모듈로 전송하는 기능 제공 ▪ Hidden Control 모듈에서는 통합UI화면으로 메시지 전송 기능 제공 ▪ 통합UI화면에서 최종 메시지 표출 기	JMS/APPLET

	능 제공	
Outbound Layer	▪ 통합UI화면에서 통합연동으로 데이터를 요청하고 그 결과 데이터를 받아서 통합UI화면에 데이터를 표출하기 위한 모듈 ▪ 통합UI화면에서 AJAX를 통해 통신하는 기능 제공	AJAX

표 5.6 통합UI시스템 사용자 기능

구 분		기 능
상황관제	GIS화면	▪ 도시지리정보를 보여주며, 각 시설물에 대한 정보를 확인
	통합상황관리	▪ 방범, 교통정보, 상수도, 포털 등 서비스 정보에 대한 통합 상황을 모니터링하고, 알람지령전송 작업 수행
	통합시설물정보관리	▪ CCTV, VMS, VDS제어기, VDS검지기, 신호제어기, 신호검지기 등 모든 시설물에 대한 고장 정보를 모니터링 할 수 있고, 그에 대해 시설물 현장지원 전송, 작업결과 상세보기 등의 작업을 수행
	CCTV영상화면	▪ 방범, 교통에서 관리하는 CCTV의 영상을 선택하여 표출
	서비스정보화면(정보제공창)	▪ 연동대상시스템과 정의한 서비스에 대한 정보를 선택하여 표출 ▪ 각 정보에 대한 상세보기, 전체보기 기능을 제공 ▪ 서비스 그룹관리와 연계하여 사용자별 맞춤 화면 설정 및 서비스 이벤트 관리와 연계한 자동 전환이 가능

운영 관리	서비스 그룹관리	▪ 사용자의 개인에게 최적화된 상황관제화면을 구성할 수 있도록 상황관제화면에 표출될 서비스 그룹을 설정하고 관리하는 기능을 제공
	EMS상태정보	▪ EMS와 연계를 통하여 시스템에 대한 주요정보를 모니터링하고 SSO를 통해 이를 직접 접속하여 관리하는 기능을 제공
	ESM상태정보	▪ ESM과 연계를 통하여 보안에 대한 주요정보를 모니터링하고 SSO를 통해 이를 직접 접속하여 관리하는 기능을 제공
알람 지령	알람지령	▪ ACS, FAX, SMS등을 통한 동보전송 기능 및 이에 대한 통계 등의 관리기능을 제공
	E-FAX관리	▪ Fax를 수신하여 온라인상으로 이를 확인하고 관리할 수 있는 기능을 제공
	현장시설송신관리	▪ 현장지원시스템에 지령한 작업 내용을 확인하고 이를 관리할 수 있는 기능을 제공
	VMS메시지관리	▪ 상황관제에서 VMS에 표출되는 문구를 확인할 수 있으며, 통합UI에서 교통정보제공서비스에 요청한 VMS 메시지를 관리
	신고접수관리	▪ 유/무선 전화를 통하여 신고되는 다양한 상황을 접수하고 이를 관리하는 기능을 제공
	인터넷 민원접수관리	▪ 포탈 서비스를 통해 접수되는 민원내용을 확인하고 이를 관리하는 기능을 제공
시스템관리	서비스분류관리	▪ 통합UI에서 정의된 정보제공서비스에 대한 등록, 수정, 삭제, 조회 등의 서비스분류관리 기능을 제공
	서비스이벤트관리	▪ 서비스의 이벤트에 맞도록 상황관제 정보제공창을 변경할 수 있도록 이벤트를 설정 및 관리하는 기능을 제공
	사용자접속통계	▪ 사용자별 접속현황을 조회하는 기능을 제공

	시스템로그정보	▪ 시스템의 로그를 관리하는 기능을 제공
	유형코드관리	▪ 공통코드로 사용되는 코드의 그룹코드를 관리하는 기능을 제공
	시스템코드관리	▪ 시스템에서 사용하는 코드를 관리하는 기능을 제공
	시스템사용자관리	▪ 시스템 사용자를 등록/수정/삭제/조회하는 기능을 제공
	비상연락망관리	▪ 유관기관 등의 비상연락망을 관리하는 기능을 제공

□ 통합관제플랫폼 메인화면

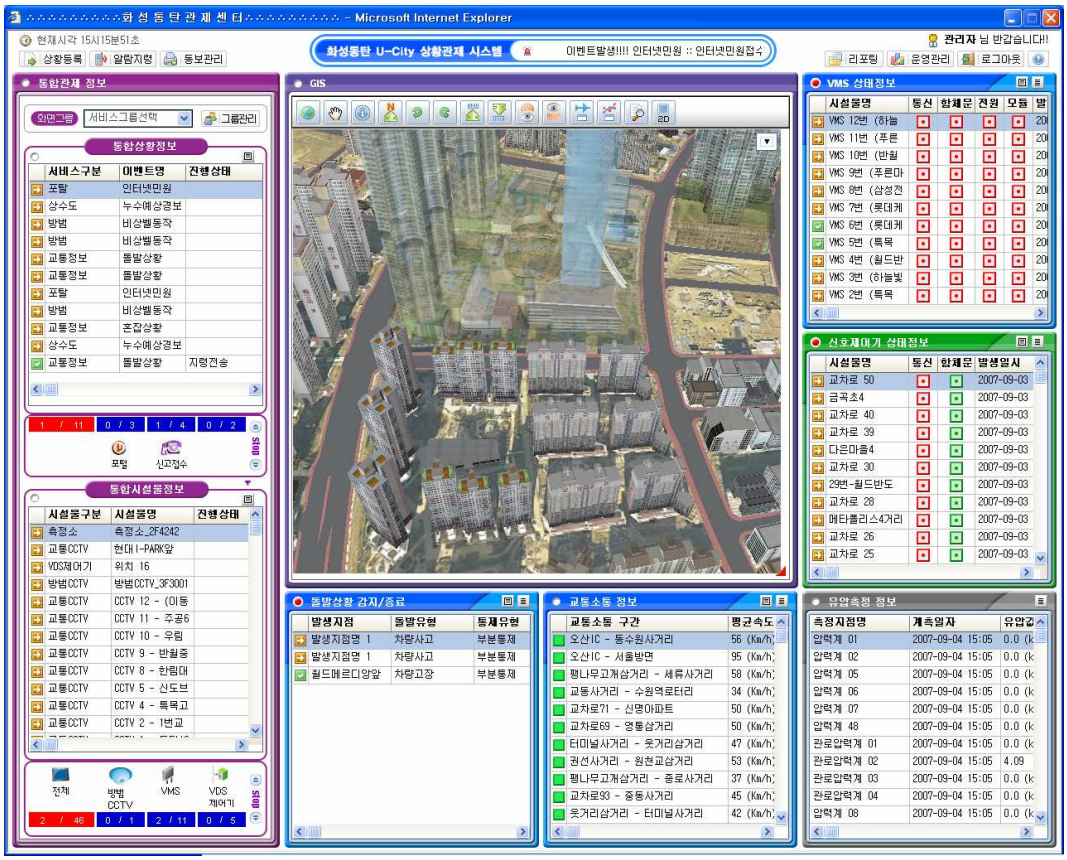


그림 5.26 통합관제플랫폼 메인화면

□ 관제센터에서의 통합관제플랫폼 실행화면

그림 5.27 관제센터에서의 통합관제플랫폼 실행화면

5.3.2 통합관제플랫폼 성능테스트

본 연구에서 통합관제플랫폼의 성능테스트를 위해 가상의 성능부하 시험모듈을 구성하여 테스트를 실시하였다. 시험은 1시간 동안 트랜잭션을 발생시켜 응답시간 및 서버 자원을 모니터링 하였다. 테스트 툴의 그림은 5.28과 같다. 시험결과는 그림 5.29, 5.30, 5.31, 5.32에 나타내었다.

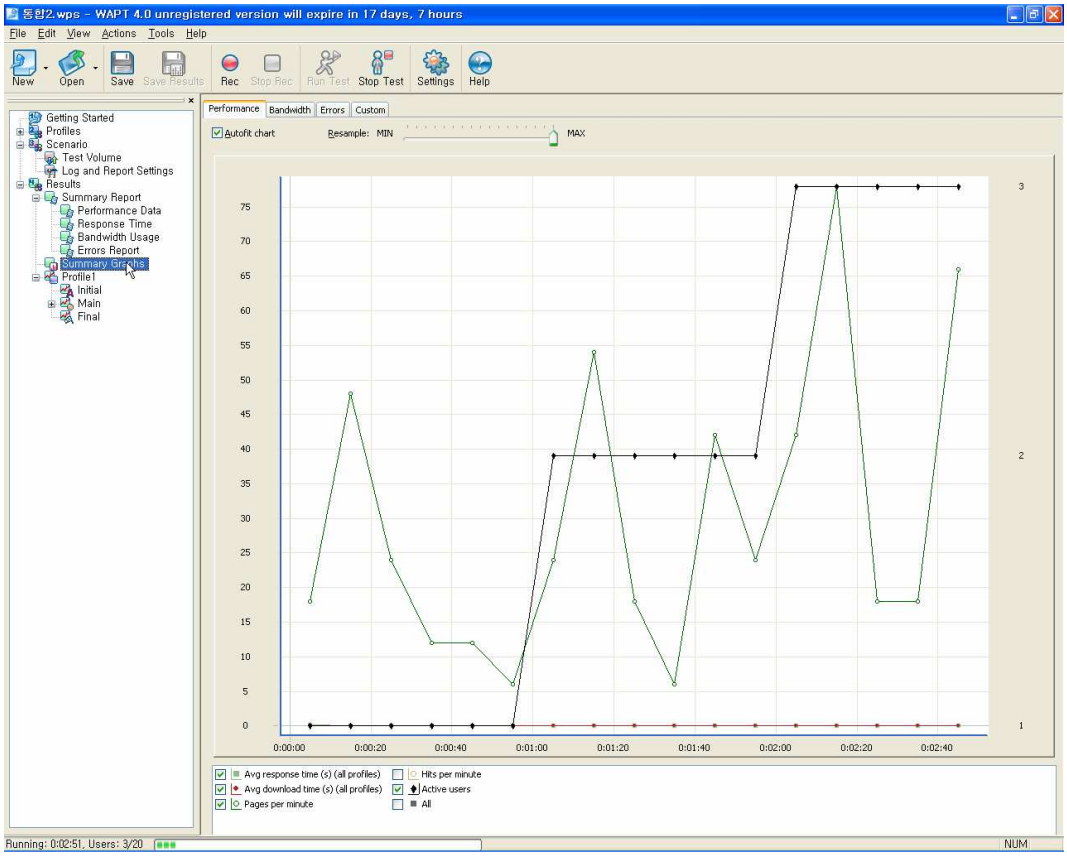


그림 5.28 성능테스트 툴 WAPT4.0 화면

1) 시험내용 및 항목

① 성능테스트 툴

► 제품명 : WAPT4.0

► 제조사 : SoftLogica LLC

② 시험내용 및 항목

► 시험항목수 : 31개

► 시험데이터 : 1set

► 사용자 수 : 1명부터 20명까지 1명씩 증가

► 부하발생방법 : 부하발생을 위한 성능부하시험 모듈 사용

► 시험방법 : 60분간 지속적으로 트랜잭션 발생

③ 성능시험 판정기준

► 기준 : 목표값(3초 이내)을 만족하고 Fail수가 1%이내

2) 시험결과

① 시험결과 내용

► 수행된 프로파일 총 개수 : 366개

► 수행된 페이지 총 개수 : 13,179개

► Fail 수 : 0%

► 평균응답시간 : 0.001초

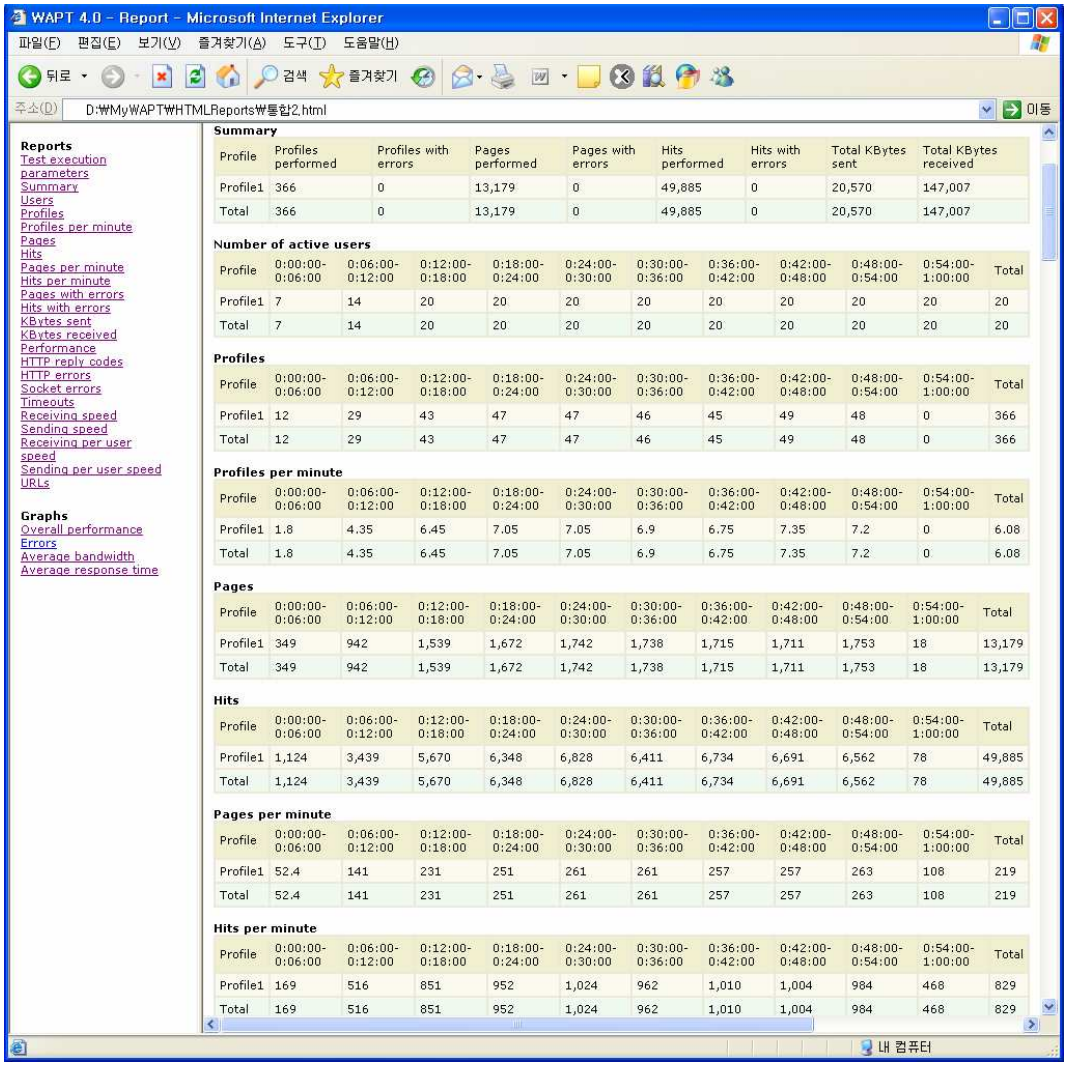
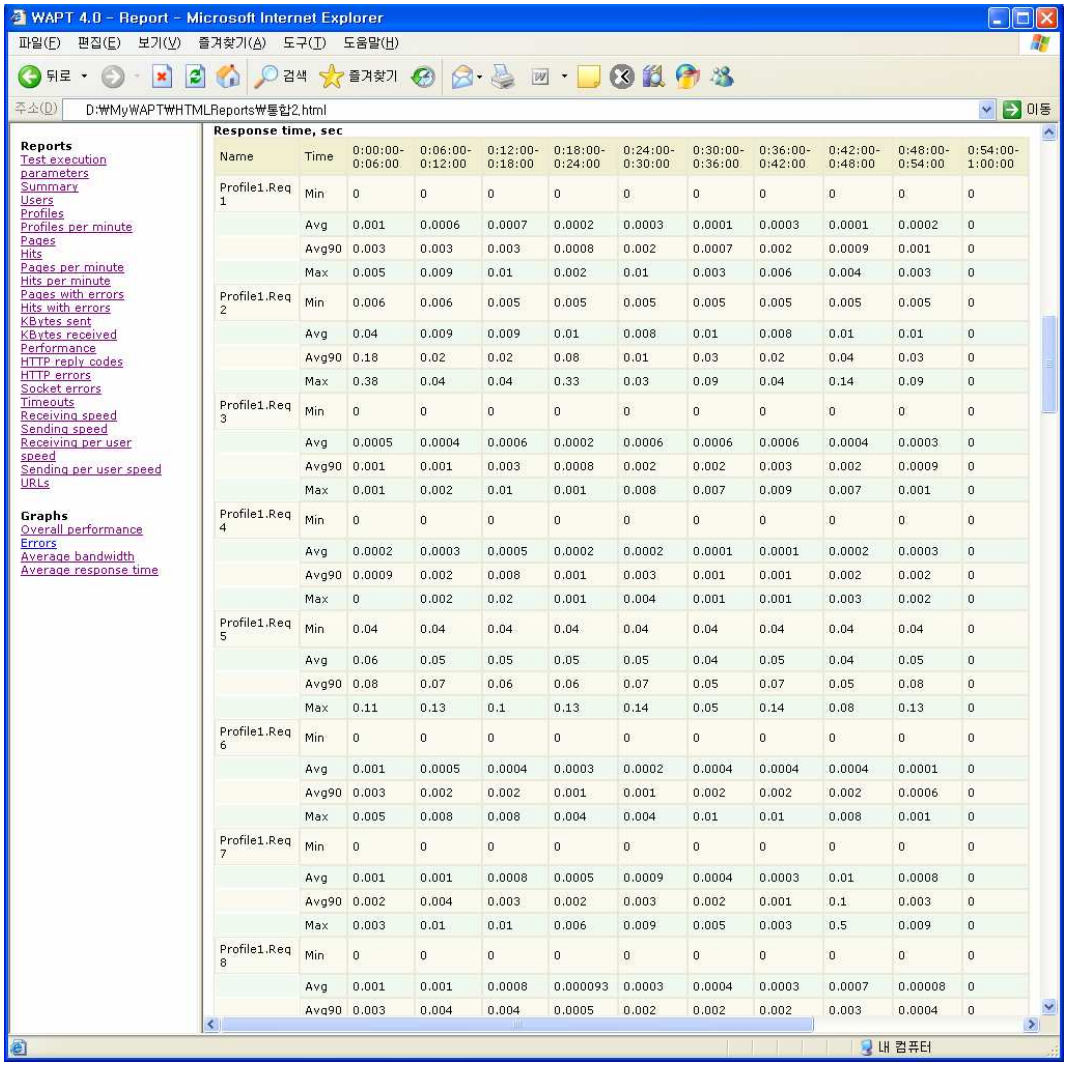


그림 5.29 테스트 결과 리포트 전체 수행 내역



Name	Time	0:00:00-0:06:00	0:06:00-0:12:00	0:12:00-0:18:00	0:18:00-0:24:00	0:24:00-0:30:00	0:30:00-0:36:00	0:36:00-0:42:00	0:42:00-0:48:00	0:48:00-0:54:00	0:54:00-1:00:00
Profile1.Req 1	Min	0	0	0	0	0	0	0	0	0	0
	Avg	0.001	0.0006	0.0007	0.0002	0.0003	0.0001	0.0003	0.0001	0.0002	0
	Avg90	0.003	0.003	0.003	0.0008	0.002	0.0007	0.002	0.0009	0.001	0
	Max	0.005	0.009	0.01	0.002	0.01	0.003	0.006	0.004	0.003	0
Profile1.Req 2	Min	0.006	0.006	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0
	Avg	0.04	0.009	0.009	0.01	0.008	0.01	0.008	0.01	0.01	0
	Avg90	0.18	0.02	0.02	0.08	0.01	0.03	0.02	0.04	0.03	0
	Max	0.38	0.04	0.04	0.33	0.03	0.09	0.04	0.14	0.09	0
Profile1.Req 3	Min	0	0	0	0	0	0	0	0	0	0
	Avg	0.0005	0.0004	0.0006	0.0002	0.0006	0.0006	0.0006	0.0004	0.0003	0
	Avg90	0.001	0.001	0.003	0.0008	0.002	0.002	0.003	0.002	0.0009	0
	Max	0.001	0.002	0.01	0.001	0.008	0.007	0.009	0.007	0.001	0
Profile1.Req 4	Min	0	0	0	0	0	0	0	0	0	0
	Avg	0.0002	0.0003	0.0005	0.0002	0.0002	0.0001	0.0001	0.0002	0.0003	0
	Avg90	0.0009	0.002	0.008	0.001	0.003	0.001	0.001	0.002	0.002	0
	Max	0	0.002	0.02	0.001	0.004	0.001	0.001	0.003	0.002	0
Profile1.Req 5	Min	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0
	Avg	0.06	0.05	0.05	0.05	0.05	0.04	0.05	0.04	0.05	0
	Avg90	0.08	0.07	0.06	0.06	0.07	0.05	0.07	0.05	0.08	0
	Max	0.11	0.13	0.1	0.13	0.14	0.05	0.14	0.08	0.13	0
Profile1.Req 6	Min	0	0	0	0	0	0	0	0	0	0
	Avg	0.001	0.0005	0.0004	0.0003	0.0002	0.0004	0.0004	0.0004	0.0001	0
	Avg90	0.003	0.002	0.002	0.001	0.001	0.002	0.002	0.002	0.0006	0
	Max	0.005	0.008	0.008	0.004	0.004	0.01	0.01	0.008	0.001	0
Profile1.Req 7	Min	0	0	0	0	0	0	0	0	0	0
	Avg	0.001	0.001	0.0008	0.0005	0.0009	0.0004	0.0003	0.01	0.0008	0
	Avg90	0.002	0.004	0.003	0.002	0.003	0.002	0.001	0.1	0.003	0
	Max	0.003	0.01	0.01	0.006	0.009	0.005	0.003	0.5	0.009	0
Profile1.Req 8	Min	0	0	0	0	0	0	0	0	0	0
	Avg	0.001	0.001	0.0008	0.000093	0.0003	0.0004	0.0003	0.0007	0.00008	0
	Avg90	0.003	0.004	0.004	0.0005	0.002	0.002	0.002	0.003	0.0004	0

그림 5.30 테스트 결과 리포트 초당 응답 시간

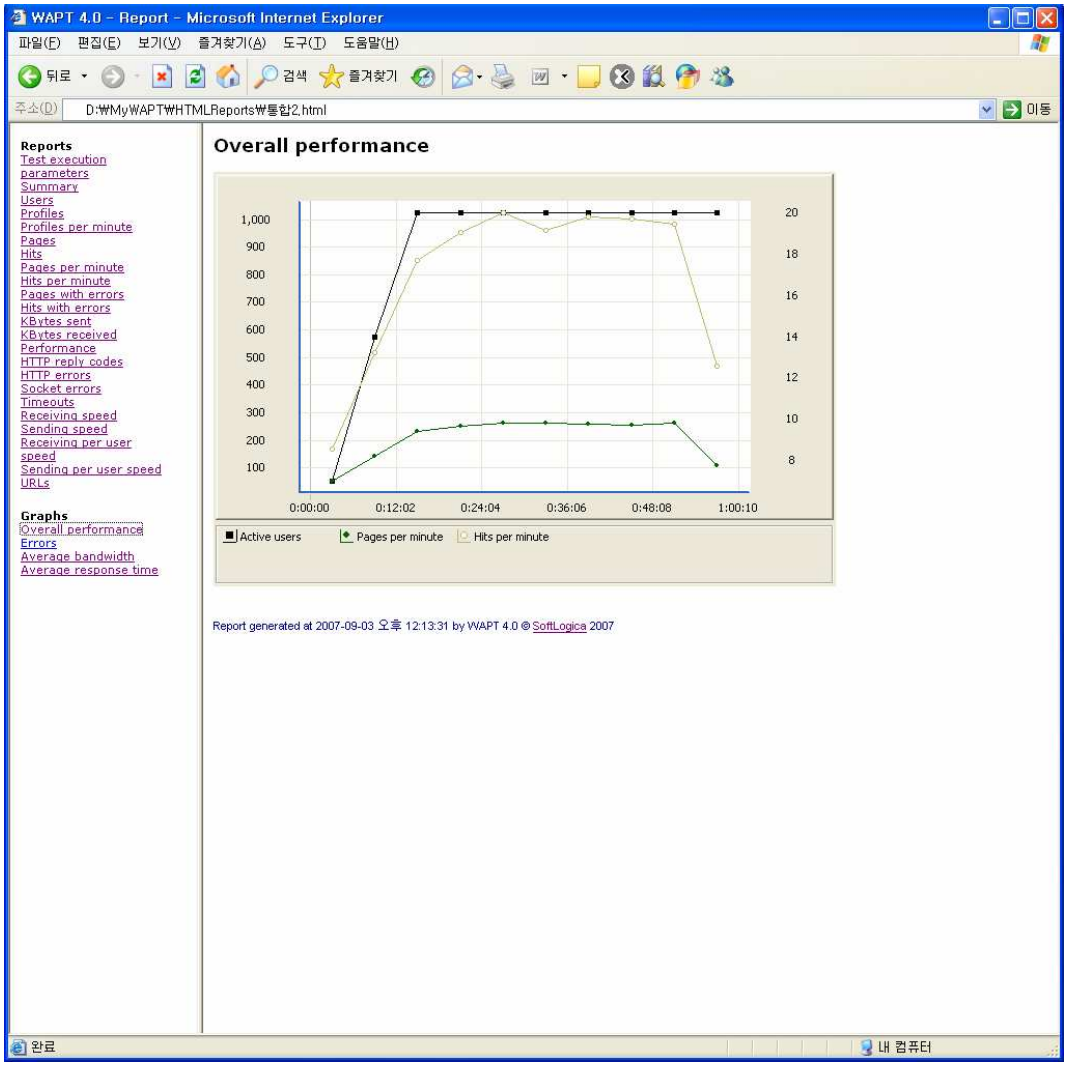


그림 5.31 테스트 결과 리포트 전체 성능 수행 그래프

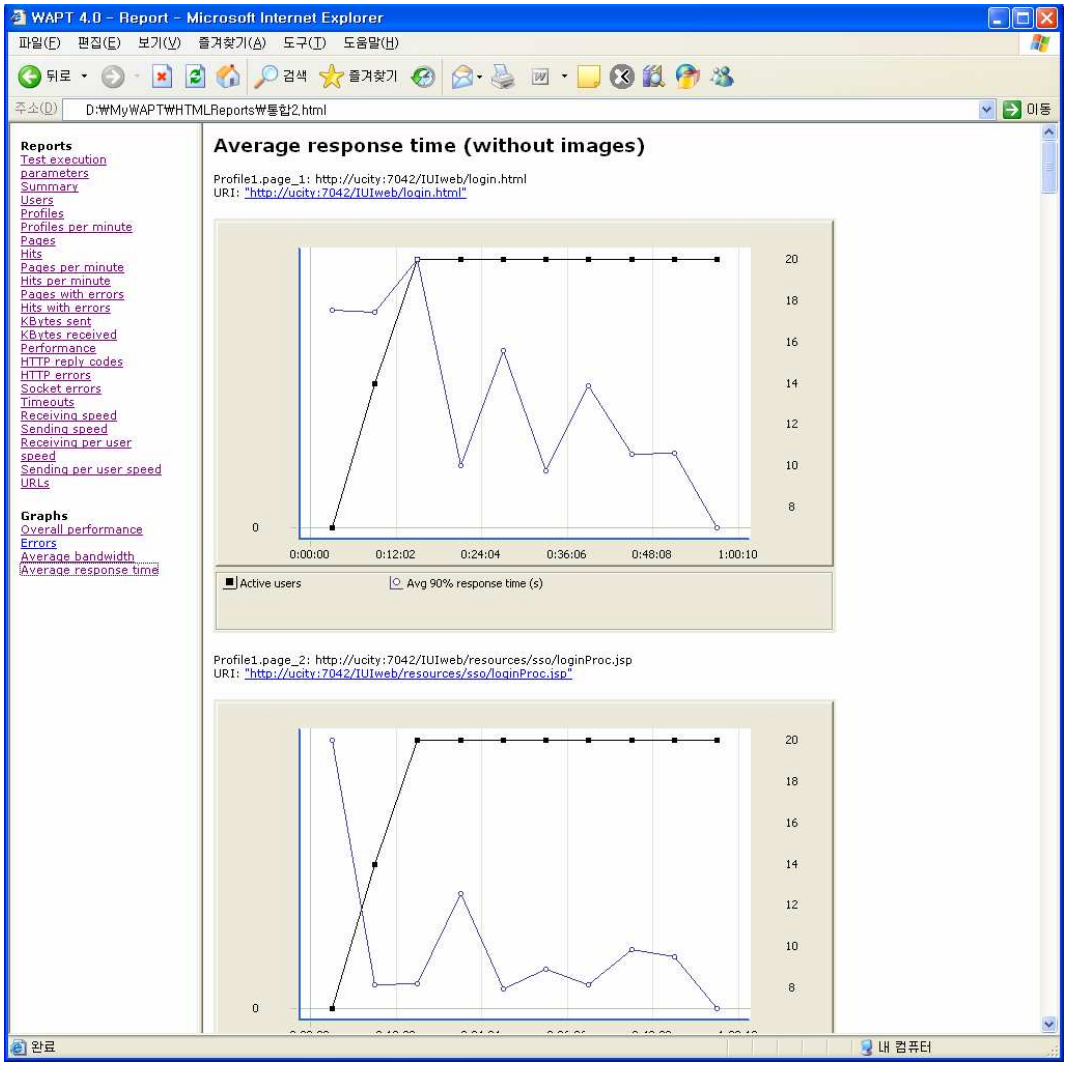


그림 5.32 테스트 결과 리포트 업무별 성능 수행 그래프

제 6 장 결 론

본 연구에서는 유비쿼터스 컴퓨팅 환경에서의 실시간 통합관제를 위한 시스템 미들웨어의 설계 및 구현을 목적으로 하고 있다. 이와 같은 목적을 이루기 위해, 먼저 유비쿼터스 컴퓨팅 환경에서 통합관제를 위해 제공해야 할 여러 요구 사항들에 대해서 분석하였고, 이렇게 분석된 결과를 토대로 미들웨어를 구성하는 3가지 계층구조와 미들웨어 Core의 기능을 도출하였다.

미들웨어의 3가지 계층으로는 이기종 정보수집 단말 장치와 미들웨어와의 통신연결을 담당하는 Application Connector Layer가 있고, Connector를 통한 외부와의 인터페이스를 보장하며, Listener를 통하여 내부 계층으로의 연계를 담당하는 External Layer가 있고, 미들웨어 내부와 외부와의 인터페이스를 위한 핵심적인 역할과 미들웨어 Core를 중심으로 실제 업무 로직 구현을 담당하는 Internal Layer가 있다.

미들웨어 Core는 자원관리, 이벤트관리, 메시지관리, 프로세스관리, Agent관리 등 미들웨어의 모든 Agent의 생성부터 소멸까지의 라이프사이클을 관리하며 시스템의 모든 모듈간의 통신 및 자원분배 등 전체적인 기능을 유지관리한다.

이렇게 미들웨어의 각 모듈들을 도출한 다음, 도출된 모듈들을 가지고 있는 미들웨어가 어떤 방식으로 동작하는지 보여주는 시스템의 전체적인 구조를 설계하였다. 이때, 상황에 따른 이벤트의 우선순위를 분류하기 위해 신경회로망 모델을 설계하였다. 그리고, 각각의 미들웨어 모듈들이 가지고 있는 세부적인 기능들에 대해 자세히 정의하였다. 특징적으로 통합관제를 위한 미들웨어의 상황인지 과정에서 이벤트의 우선순위 분류를 실시간으로 처리하기 위해 신경회로망을 사용한 분류 기법을 제안하였다.

제안된 기법은 기존의 룰-데이터베이스 기반 우선순위 알고리즘의 인식률

(83.3%)과 처리속도(0.1초)에 비해 우수한 인식률(96.3%)과 처리속도(0.01초)로 이벤트 우선순위 분류에서 소요되는 시간을 단축하고, 좀 더 효율적으로 분류하였다.

이와 같이 설계된 내용을 바탕으로 본 연구에서는 실제 미들웨어를 구현하였다. 그리고, 이렇게 구현된 미들웨어가 적용된 환경에서 프로그래머가 응용프로그램을 구현할 때, 시스템의 내부 동작을 몰라도 쉽게 구현할 수 있도록 간단한 API를 제공하였다. 마지막으로, 구현된 미들웨어의 성능을 테스트하기 위해 유비쿼터스 컴퓨팅 환경에서의 통합관제플랫폼을 구축하였고, 시스템 부하 테스트 툴을 사용하여 가상의 성능부하 시험모듈을 구성하고 실제 성능을 테스트하였다.

본 연구에서 고려되지 않아서 제한적인 결과를 도출하게 된 원인을 감소시키고, 더욱 신뢰성 있는 결과를 도출하기 위하여 향후 연구해야할 과제는 우선, u-City에서 발생할 수 있는 상황의 경우에 대해 좀 더 확충하는 것이 필요하다. 본 연구의 범위는 4가지의 u-서비스와 6가지 이벤트와 25가지의 내용에 한하였으나, 향후 좀 더 다양하고 복합적인 경우에 대한 연구가 추가적으로 필요할 것으로 생각되고, 본 연구에서 고려한 신경망 학습방법인 역전파 알고리즘(Backpropagation Algorithm)이외에도 RBF(Radial-basis Function), GrossbergNetwork 등 다른 신경망 학습방법과 퍼지(Fuzzy), 유전알고리즘(Genetic Algorithm) 등에 대한 고려가 필요하다.

참 고 문 헌

[1] Jacek M. Zurada, "Introduction to Artificial Neural System," West Publishing Coimpany, 1992.

[2] 이주상, "A Study on the Fingerprint Recognition Method Directional Feature Detection using Neural Networks", 한국해양대학교 석사학위논문, 2001.

[3] Jae Hyun Lee, Sung Joo Kim, Sang Bae Lee, "The study on the intelligent control of robot system using neural network," Preceedings of Asian Control Conference, vol.Ⅲ, No.Ⅲ, pp.67-70, 1997.

[4] Sung Joo Kim, Jae Hyun Lee, Sang Bae Lee, "The Study on the Optimal control of Linear Track Cart Double Inverted Pendulum Using Neural Network," Preceedings of Asian Control Conference, vol. Ⅱ, no.Ⅱ, pp.15-18, 1997.

[5] 이재현, 강성인, 이상배, "신경회로망을 이용한 동적 시스템의 상태 공간 인식 모델에 관한 연구," 한국퍼지 및 지능시스템학회 논문집 vol.7, no.2, pp.115-120, 1997.

[6] M. Minsky and S. Paper, Perceptrons, MIT Press, 1969.

[7] Y. Chen and F. Bastani, "ANN with Two Dendrite Neurons and Weig -ht Initialization," Proc, IJCNN, Baltimore, vol.Ⅲ, pp.139-146, 1992.

[8] C. Koch and T. Poggio, "Multiplying with Synapses and Neurons ," in Single Neuron Computation, T. Mckenna, J. Davis, and S. F. Zon-netzer, pp.3165-3455, 1992.

[9] 정부만, 이상훈, 이계원, 박용철, 김재원, 박진석, 이창훈, "한국형 u-City모델 제안," 한국전산원, 2005.

[10] 박용민, "U-City 기반기술 상세 내역", RFID-Tech, 3월호, pp.112-119,

2006.

[11] 한국과학기술정보연구원, "유비쿼터스 미들웨어 기술 동향", 한국과학기술정보 연구원 보고서, 2004.

[12] 전자신문사, "2005 유비쿼터스백서", 2005.

[13] 행정자치부, "재난 및 안전관리 기본법 시행령", 대통령령 제20402호, 2007.

[14] 건설교통부, "건설교통재난재해대책편람", 2006.

[15] 건설교통부, "건설교통재난재해 위기대응 메뉴얼", 2006.

[16] u-City 포럼, "www.ubicity.org"

[17] 장태현, "한국 재난통합관리체제에 관한 연구", 인하대학교 석사학위논문, 2004.

[18] Demetri Psaltis et al., "A Multilayered Neural Networks Controller", IEEE Control System Magazine, Vol. 8, pp. 17-21, April 1988.

[19] K. Hornik, M. Stinchcombe, H. White, "Multilayer feedforward network are universal approximators", Dept. Economics, University of California, San Diego, CA, Discussion Pap., June 1988.

[20] Simon Haykin, Neural Networks, Macmillan Company, 1994.

[21] S. Omatu, M. Khalid, R. Yusof, Neuro-Control and its Applications, springer, 1995.

[22] J. S. R. Jang, C. T. Sun, E. Mizutani, Neuro-Fuzzy and Soft Computing, Prentice Hall, 1997.

[23] K. S. Narendra, K. Parthasarathy, "Neural Networks and Dynamical Systems. Part I: A Gradient Approach to Hopfield Networks", Center Syst. Sci., Dept. Electrical Eng., Yale University, New Haver, CT, Tech. rep. 8820, October 1988.

[24] L. Ljung and J. Sjoberg, "A System Identification Perspective on Neural Nets", in Neural Networks for Signal Processing II, Proceedings of the 1992 IEEE-SP Workshop, pp.423-435, 1992.

[25] Eric S. Raymond, 김희석, "Art of Unix Programming", 정보문화사,

2004.

[26] Kay A. Robbins, 권상호, "UNIX Systems Programming", 정보문화사, 2006.

[27] 데이비드 R. 부트노프, 강철민, "POXIS(포직스) 쓰레드를 이용한 프로그래밍", 인포북. 2005.

[28] W.Richard Stevens, Bill Fenner, Andrew M. Rudoff, "Unix Network Programming1(Third Edition)", 교보문고, 2005.

[29] 태봉섭, "유비쿼터스 컴퓨팅을 위한 상황인식 미들웨어의 설계 및 구현", 전북대학교 석사학위논문, 2004.

[30] 짜웨이, 어상훈, 김경배, 조숙경, 배해영, "상황 인지 시스템에서 개선된 역전파 알고리즘을 사용하는 진보된 학습 매커니즘을 위한 프레임워크", 정보처리학회지, 제14-D권, 제1호, pp139-144.

[31] A. Ranganathan and R. H. Compbell, "A Middleware for Context Aware Agents in Ubiquitous Computing Environments.", In ACM/IFIP/USENIX International Middleware Conference 2003.

[32] S. S. Yau and F. Karim. "An adaptive middleware for context-sensitive communications for real-time applications in ubiquitous computing environments", Journal of RealTime Systems, Vol26, pp29-61, 2004.

[33] 정헌만, "유비쿼터스 컴퓨팅 환경에서의 온톨로지 기반 상황 인식 미들웨어", 한국컴퓨터정보학회지, 제14권, 1호, pp.165-173, 2006.

[34] 심춘보, 신용원, "유비쿼터스 컴퓨팅 환경에서 상황인식을 지원하는 컨텍스트 미들웨어 개발", 한국지능정보시스템 학회논문지, 제11권, 1호 pp.53.63, 2005.

[35] 이응주, 윤희용, 조위덕, "유비쿼터스 환경을 위한 코바 기반 미들웨어 비교 연구", 한국지능정보시스템학회 2004 추계학술대회, pp162-168, 2004.

[36] 김은영, 오동열, "분산 유비쿼터스 환경을 위한 상황 인식 미들웨어의 설

계 및 구현", 한국컴퓨터정보학회지, 제11권, 5,43호 pp.105-114. 2006.

[37] 이선명, "테이블탑 컴퓨팅 환경을 위한 미들웨어 설계 및 구현", 포항공과대학교 석사학위논문, 2005.

[38] Klara Nahrstedt Dongyan Xu, Daungdao Wichadakul Baochun Li, "QoS-aware Middleware for ubiquitous and Heterogeneous Environments," 2001, Department of Computer Science Department of Electrical and Computer Engineering University of Illinois at Urbana-Champaign.

[39] Couto Antunes da Rocha. R, Endler. M, "Context Management in Heterogeneous, Evolving Ubiquitous Environments", IEEE distributed systems online, 2006, 7(4), pp. 1-1.

[40] Pham, K.A.N, Lee. Y. K, Lee, S. Y, "Middleware Architecture for Context Knowledge Discovery in Ubiquitous Computing", Lecture notes in computer science, 2005, 3824, pp1177-1188.

부 록

본 연구에서 제공하고 있는 API를 가지고 Agent를 구현하는 방법에 대해 살펴보도록 하겠다.

1. API 사용법

본 연구에서 정의한 API를 사용하기 위해서는 다음의 두 절차를 수행해야 한다. 먼저, 소스 프로그램 안에 본 연구에서 제공하고 있는 API의 헤더 파일을 포함시킨다. 그런 다음, 컴파일을 수행할 때 API에서 제공하고 있는 라이브러리 파일을 링크시켜주면 된다.

2. Agent 프로그램 구조

본 연구에서 정의한 API를 사용하여 Agent를 구현할 때, 다음의 소스코드에서 보여주는 것과 같은 기본적인 구조를 사용할 수 있다. 이 코드는 Agent를 구현할 때 프로그램 각각의 부분에서 API의 함수와 메시지들이 기본적으로 어떻게 사용되고 있는지를 보여주고 있다.

미들웨어 헤더파일 <*include.h*>

```
//#define __deubgrte__

#ifndef _INCLUDE_
#define _INCLUDE_
//-------------------------------------------------------------
------
// 미들웨어 기본설정
//-------------------------------------------------------------
```

```
------
//#define   _debug_
#define _RTESOCKET_   0
#define _LIVE_TIME_   30
//-------------------------------------------------------------
------
// 표준함수/시스템라이브러리
//-------------------------------------------------------------
------
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <stddef.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/shm.h>
#include <sys/select.h>
#include <time.h>
#include <unistd.h>
#include <curses.h>
#include <errno.h>
//동적  라이브러리
```

```
#include <dlfcn.h>
#include <unistd.h>
#include <sys/stat.h>

#ifndef __USE_GNU
#define __USE_GNU
#endif

#include <fcntl.h>

//-------------------------------------------------------------
------
// type 정의
//-------------------------------------------------------------
------
typedef unsigned char  byte;
typedef unsigned long    ulong;

typedef void (*mw_cfn)         ( int, int );
typedef void (*mw_dfn)         ( int, int );
typedef void (*mw_xfn)         ( int, int, long );
typedef void (*mw_rfn)         ( int, int, int, void (*x_fn)(int, int, long));
//-------------------------------------------------------------
------
// 미들웨어 라이브러리
//-------------------------------------------------------------
------
#include "mtxdb.h"
#include "config.h"
#include "seq.h"
#include "mem.h"
```

```
#include "table.h"
#include "pack.h"
#include "resource.h"
#include "event.h"
#include "boss.h"
#include "proxy.h"
#include "child.h"
#include "sig.h"
#include "client.h"
#include "server.h"
#include "common.h"
#include "misc.h"
#include "core.h"
#include "listener.h"

extern int mw_message_key;
extern int mw_memory_key;
extern int mw_socket_key;
//-------------------------------------------------------------
------
// 미들웨어 사용자정의 라이브러리
//-------------------------------------------------------------
------
#include "user.h"

#endif
```

미들웨어 소스파일 <*user.cc*>

```
/* SOURCE FILE
 *
```

```
 * 일반적인 agent를 기술하기 위한 구조를 설명한다.
 *
 * 실제 외부 application으로 접속요청을 받았을 경우
 * 이를 미들웨어Core에게 통보하고 해당하는
 * agent를 생성하도록 한다.
 *
 */

#include "include.h"
//-------------------------------------------------------------
// 신규로 생성할 Agent를 정의한다.
// int addagent( int uuid, char *name, char type, int relive);
// uuid : Agent 식별자
// name : Agent 이름
// type : 내부Agent인지 외부Agent인지 결정
// relive : Agent가 죽었을 경우 다시 살릴 것인가 결정
//-------------------------------------------------------------
int mw_define_agent()
{
        mw_add_agent( TEST,     "test",     'i',     LIFE_AGAINE );

        return 1;
}

//-------------------------------------------------------------
// 신규 Agent에서 사용할 이벤트를 정의한다.
// int addevent( int eventno, int uuid , int inout);
// eventno : 이벤트식별자
// uuid : Agent 식별자
// inout : 전송할 이벤트인지 수신할 이벤트인지 결정
//-------------------------------------------------------------
```

```
int mw_define_event()
{
        mw_add_event( EV_TEST,     TEST,      SEND_GRANT );
        mw_add_event( EV_TEST,     TEST,      RECV_GRANT );

        return 1;
}

//-------------------------------------------------------------
// 사용할 리소스 영역을 정의한다.
// int addresource ( int resourceno, int rowsize, int rows );
// resourceno : 리소스식별자
// rowsize : row size 지정
// rows : row count 지정
//-------------------------------------------------------------
int mw_define_resource()
{
        mw_add_resource( RE_TEST,     100,      10 );

        return 1;
}

//-------------------------------------------------------------
// Agent의 비즈니스 로직을 작성한다.
// user define function
// int test_after_start : Agent 시작 후에 수행할 로직을 작성한다.
// void test_before_stop : Agent 종료 직전에 수행할 로직을 작성한다.
// int test_after_loop : 반복 실행 후에 수행할 로직을 작성한다.
// void test_msg_event : 다른 Agent나 Core로 부터 메시지를 받을때
//                                  처리하는 로직을 작성한다.
// void test_comm_event : 이벤트와 관련된 데이터를 읽고 쓰는 영역이다.
```

```
//-------------------------------------------------------------
int test_after_start( int uuid )
{
        st_event e;             // 이벤트 구조체를 생성한다.
        e.data.i = 100;         // 이벤트 데이터 크기를 100로 한다.
        e.to = DIYCAD;          // 이벤트를 받을 대상을 지정한다.
        e.from = uuid;          // 이벤트를 보내는 자신의 정보이다.
        e.type = EV_TEST;       // 이벤트 타입을 설정한다.
        mw_tell(e);             // 이벤트를 전송한다.
}
void test_before_stop( int uuid )
{
        // Agent 종료 직전에 수행할 로직을 작성한다.
}
int test_after_loop( int uuid )
{
        // 반복 실행 후에 수행할 로직을 작성한다.
}
void test_msg_event( int uuid, st_event e )
{
        // 다른 Agent나 Core로 부터 메시지를 받을 때 처리하는 로직을
           작성한다.
}
void test_comm_event( int uuid, int len, long opcode )
{
        // 이벤트와 관련된 데이터를 읽거나 쓸때 사용하는 함수이다.
        long a, s, key;
        double b;
        char c[50];
        printf("table %d %d\n", len, opcode);
```

```
printf("begin key [%d]\n", (key = mw_begin( opcode )) );

while ( (key = mw_next( opcode, key )) )
{
    mw_glong ( opcode, "key", &a); printf("%d\n", a);
    mw_gtext ( opcode, "col2", c, &s); printf("%s[%d]\n", c, s);
    mw_gdouble ( opcode, "col3", &b); printf("%f\n", b);
};

printf("begin key [%d]\n", (key = mw_begin( opcode )) );

while ( (key = mw_next( opcode, key )) )
{
    mw_glong ( opcode, "key", &a); printf("%d\n", a);
    mw_gtext ( opcode, "col2", c, &s); printf("%s[%d]\n", c, s);
    mw_gdouble ( opcode, "col3", &b); printf("%f\n", b);
};

mw_tx_table ( uuid, opcode);

printf("next key [%d]\n", (key = mw_next( opcode, key )) );

mw_glong ( opcode, "key", &a); printf("%d\n", a);
mw_gtext ( opcode, "col2", c, &s); printf("%s[%d]\n", c, s);
mw_gdouble ( opcode, "col3", &b); printf("%f\n", b);
mw_slong    ( opcode, "col4" ,  213 );
printf("find key [%d]\n", (key = mw_find( opcode, "col4")) );

mw_glong ( opcode, "key", &a); printf("%d\n", a);
mw_gtext ( opcode, "col2", c, &s); printf("%s[%d]\n", c, s);
mw_gdouble ( opcode, "col3", &b); printf("%f\n", b);
```

```
        mw_stext ( opcode, "col2" ,  "0001", 4 );
        printf("find key [%d]\n", (key = mw_find( opcode, "col2")) );
        mw_glong ( opcode, "key", &a); printf("%d\n", a);
        mw_gtext ( opcode, "col2", c, &s); printf("%s[%d]\n", c, s);
        mw_gdouble ( opcode, "col3", &b); printf("%f\n", b);
}
```

3. 파라미터 정의

미들웨어 Core와 각각의 Agent의 기능 수행을 위해 API에서 사용될 파라미터 형태를 정의하였다. 미들웨어에서 사용되는 파라미터는 미들웨어의 기본 동작을 지정하는 파라미터, 이벤트의 역할을 지정하는 파라미터, Agent의 라이프 형태를 지정하는 파라미터, Agent 자신의 정보를 정의하는 파라미터, 리소스를 사용하기 위한 파라미터, 공통으로 사용되는 시스템 내부 이벤트 파라미터로 구분하였다. 이들 파라미터들을 간단히 표로 정리하면 다음의 표 1과 같다.

표 1 파라미터 정의

구분	파라미터 이름	설명
em_status	S_STOP	프로세스를 정지한다.
	S_INIT	프로세스를 초기화한다.
	S_ACCEPTED	프로세스가 값을 받아들일 수 있도록 허락한다.
	S_ERROR	프로세스 에러가 발생했다.
	S_SYSTEM	시스템 변수를 지정한다.
em_eventstatus	SEND_GRANT	이벤트를 전송할 권한을 준다.
	RECV_GRANT	이벤트를 수신할 권한을 준다.
em_agentlife	LIFE_DONE	Agent의 상태를 현상태 그대로 유지한다.

구분	파라미터 이름	설명
	LIFE_AGAINE	Agent가 죽으면 다시 살린다.
em_uuid	NONE	Agent의 일반적인 정보를 지정한다.
	MAIN	Agent를 구분하는 주요 명칭을 지정한다.
	LISTENER	Listener의 번호를 지정한다.
	UUID_NUMS	Agent의 번호를 지정한다.
em_resource	RE_CONDITION_SET	리소스의 사용조건을 설정한다.
	RESOURCE_NUMS	리소스에서 사용할 영역의 번호를 설정한다.
em_event	EV_NONE	일반 이벤트임을 나타낸다.
	EV_START	시작 이벤트임을 나타낸다.
	EV_EXIT	탈출 이벤트임을 나타낸다.
	EV_SIGNAL	신호 이벤트임을 나타낸다.
	EV_SERVERSOCK	서버 소켓 이벤트임을 나타낸다.
	EV_CLIENTSOCK	클라이언트 소켓 이벤트임을 나타낸다.
	EV_LOOP	반복 이벤트임을 나타낸다.
	EV_STOP	정지 이벤트임을 나타낸다.
	EV_STOP_BEFORE	정지전 이벤트임을 나타낸다.
	EV_STOP_CHILD	자식 정지 이벤트임을 나타낸다.
	EV_STOP_PROXY	프록시 정지 이벤트임을 나타낸다.
	EV_STOP_BOSS	보스 정지 이벤트임을 나타낸다.
	EV_STOP_RESOURCE	리소스 정지 이벤트임을 나타낸다.
	EV_MAKE_AGENT	Agent 생성 이벤트임을 나타낸다.
	EV_LIVE_AGENT	Agent를 살리는 이벤트임을 나타낸다.
	EV_DEAD_AGENT	Agent를 죽이는 이벤트임을 나타낸다.

구분	파라미터 이름	설명
	EV_TABLE	테이블 이벤트임을 나타낸다.
	EV_PORTNUMBER	포트번호 이벤트임을 나타낸다.
	EV_ECHO_AGENT	에코 Agent 이벤트임을 나타낸다.
	EVENT_NUMS	이벤트 번호를 설정한다.

4. API 함수 정의

미들웨어 기능을 수행하는 API를 구성하고 있는 함수와 메시지들에 대해 정의하였다. API는 보스 프로세스를 관리하는 모듈, 프록시를 관리하는 모듈, 자식 프로세스를 관리하는 모듈, Listener를 관리하는 모듈, 클라이언트 소켓 자원을 관리하는 모듈, 서버 소켓 자원을 관리하는 모듈, 메인이 되는 미들웨어의 시작과 종료를 수행하는 모듈, Agent와 이벤트를 관리하는 모듈, 미들웨어의 리소스를 관리하는 모듈, 전송을 목적으로 테이블을 생성하고 관리하는 모듈 등으로 구성되어 있다. 다음의 표 2 ~ 표11 에서 함수들에 대해 보여주고 있다.

표 2 보스 프로세스를 관리하는 모듈

함수원형	설명
int mw_boss_init(int uuid, void (*x_fn)(int, int, char *));	보스 프로세스를 초기화하는 함수
int mw_boss_loop(int uuid);	보스 프로세스의 반복 루틴 함수
void mw_boss_stop(int uuid);	보스 프로세스를 해제하고 닫는 함수

표 3 프록시를 관리하는 모듈

함수원형	설명

int mw_proxy_init(int uuid);	프록시를 초기화하는 함수
int mw_proxy_loop(int uuid);	프록시의 반복 루틴 함수
void mw_proxy_stop(int uuid);	프록시를 해제하고 닫는 함수

표 4 자식 프로세스를 관리하는 모듈

함수원형	설명
int mw_child_init (int uuid, void (*x_fn)(int, int, char *));	자식 프로세스를 초기화하는 함수
int mw_child_init_listener (int uuid, void (*x_fn)(int, int, char *));	자식 Listener를 초기화하는 함수
int mw_child_init_other (int uuid, void (*x_fn)(int, int, char *));	다른 자식 프로세스를 초기화하는 함수
int mw_child_loop (int uuid);	자식 프로세스의 반복 루틴 함수
int mw_child_loop_listener (int uuid);	자식 Listener의 반복 루틴 함수
int mw_child_loop_other (int uuid);	다른 프로세스의 반복 루틴 함수
void mw_child_stop (int uuid);	자식 프로세스를 해제하고 닫는 함수

표 5 Listener를 관리하는 모듈

함수원형	설명
int mw_listener_after_start (int uuid);	Listener가 시작된 후의 로직을 처리하는 함수
void mw_listener_before_stop (int uuid);	Listener가 종료되기 전에 로직을 처리하는 함수
int mw_listener_after_loop (int uuid);	Listener가 반복 후 로직을 처리하는 함수
void mw_listener_msg_event (int uuid, st_event e);	Listener에서 메시지 이벤트를 처리하는 함수
void mw_listener_comm_event (int uuid,	Listener에서 공통 이벤트를 처리

int key, char *tablename);	하는 함수

표 6 클라이언트 소켓 자원을 관리하는 모듈

함수원형	설명
int mw_tcp_client_init ();	클라이언트 소켓자원을 사용하기 위한 초기 함수
int mw_tcp_client_exit ();	클라이언트 소켓을 사용하는 모든 자원을 해제하는 함수
int mw_clopen (int uuid, char *ip, int port, void (*c_fn)(int, int), void (*d_fn)(int, int), void (*r_fn)(int, int, int), int sig);	클라이언트 소켓을 새롭게 생성하는 함수
void mw_clloop (int uuid, int c, struct timeval out);	클라이언트 소켓이 살아있는 동안 실행하는 함수
void mw_clclose (int);	클라이언트 소켓을 해제하고 닫는 함수

표 7 서버 소켓 자원을 관리하는 모듈

함수원형	설명
int mw_tcp_server_init ();	서버 소켓자원을 사용하기 위한 초기 함수
int mw_tcp_server_exit ();	서버 소켓을 사용하는 모든 자원을 해제하는 함수
int mw_svopen (int uuid, int port, int max, mtx_cfn, mtx_dfn, mtx_rfn, mtx_xfn, int sig);	서버 소켓을 새롭게 생성하는 함수
void mw_svloop (int uuid, int p, struct timeval tout);	서버 소켓이 살아있는 동안 실행하는 함수
void mw_svclose (int s);	서버 소켓을 해제하고 닫는 함수
int mw_tcp_server_status ();	서버 소켓의 상태를 검사하는 함수

표 8 메인이 되는 미들웨어의 시작과 종료를 수행하는 모듈

함수원형	설명
int mw_startup (int uuid);	미들웨어의 시작을 수행하는 함수
int mw_stop (int uuid);	미들웨어의 종료를 수행하는 함수

표 9 Agent와 이벤트를 관리하는 모듈

함수원형	설명
int mw_define_agent ();	추가로 생성할 Agent를 정의하는 함수
int mw_define_event ();	Agent가 사용할 이벤트를 정의하는 함수
int mw_define_resource ();	Agent가 사용할 리소스를 정의하는 함수
int mw_agent_init ();	Agent를 초기화 하는 함수
int mw_agent_exit ();	Agent를 해제하고 닫는 함수
int mw_add_agent (int uuid, char *name, char type, int relive);	신규 Agent를 추가하는 함수
int mw_event_init ();	이벤트를 초기화 하는 함수
int mw_event_exit ();	이벤트를 해제하고 닫는 함수
int mw_add_event(int eventno, int uuid, int inout);	신규 이벤트를 추가하는 함수
int mw_event_grant(int eventno, int uuid, int inout);	이벤트의 역할을 지정하는 함수
int mw_get_iagent (int uuid, char *argment1, char* argment2);	Internal Agent로부터 값을 가져오는 함수
int mw_get_eagent (int uuid, char *argment1, char *argment2);	External Agent로부터 값을 가져오는 함수
int mw_get_uuid (int uuid);	Agent 자신이 가지고 있는 값을

	가져오는 함수
int mw_type_agent (int uuid);	Agent의 타입을 결정하는 함수
int mw_relive_agent (int uuid);	죽은 Agent를 다시 살리는 함수

표 10 리소스를 관리하는 모듈

함수원형	설명
int mw_resource_create (int uuid);	리소스를 생성하는 함수
void mw_resource_destroy (int uuid);	리소스를 파괴하는 함수
void *mw_get_address (int uuid, int which);	리소스의 주소를 가져오는 함수
int mw_resource_init ();	리소스를 초기화 하는 함수
int mw_resource_exit ();	리소스를 해제하는 함수
int mw_add_resource (int resourceno, int rowsize, int rows);	리소스를 추가하는 함수
int mw_del_resource (int resourceno);	리소스를 삭제하는 함수
int mw_put_resource (int resourceno, void *struct_bff, int row);	리소스에 값을 입력하는 함수
int mw_get_resource (int resourceno, void *struct_bff, int row);	리소스의 값을 가져오는 함수

표 11 전송을 목적으로 테이블을 생성하고 관리하는 모듈

함수원형	설명
long mw_table_init ();	테이블을 초기화하는 함수
long mw_table_exit ();	테이블을 해제하고 닫는 함수
long mw_tbcreate (char *name);	테이블을 생성하는 함수
long mw_tbdrop(char *name);	테이블을 드롭하는 함수

long mw_find (char *tablename, char *name);	테이블의 컬럼을 찾는 함수
long mw_begin (char *name);	테이블의 시작 컬럼으로 이동하는 함수
long mw_end (char *name);	테이블의 끝 컬럼으로 이동하는 함수
long mw_prev (char *tablename, long key);	테이블에서 현재 컬럼의 바로 앞의 컬럼으로 이동하는 함수
long mw_next (char *tablename, long key);	테이블에서 현재 컬럼의 바로 다음 컬럼으로 이동하는 함수
long mw_delete (char *tablename, long key);	테이블에서 컬럼을 삭제하는 함수
long mw_clear (char * tablename);	테이블의 내용을 클리어하는 함수
long mw_insert (char *tablename);	테이블에 내용을 입력하는 함수
long mw_post (char *tablename);	테이블에 수행했던 일의 바로 앞으로 돌아가는 함수
long mw_cancel (char *tablename);	테이블에 수행했던 일을 취소하는 함수
long mw_addcolum (char *tablename, char *name, char type, long size);	테이블에 컬럼을 추가하는 함수
long mw_keycount (char *tablename);	테이블의 컬럼의 수를 계산하는 함수
long mw_fillfield (char *tablename, byte *rows);	테이블 필드전체의 값을 채우는 함수
long mw_slong (char *tablename, char *name, long value);	테이블 컬럼의 값을 정수형태로 입력하는 함수
long mw_sdouble (char *tablename, char *name, double value);	테이블 컬럼의 값을 실수형태로 입력하는 함수
long mw_stext (char *tablename, char *name, byte *text, long size);	테이블 컬럼의 값을 문자형태로 입력하는 함수
long mw_glong (char *tablename, char *name, long *value);	테이블 컬럼의 값을 정수형태로 가져오는 함수
long mw_gdouble (char *tablename,	테이블 컬럼의 값을 실수형태로

char *name, double *value);	가져오는 함수
long mw_gtext (char *tablename, char *name, byte *text, long *size);	테이블 컬럼의 값을 문자형태로 가져오는 함수

감사의 글

내 삶의 여정에서 언제나 함께 하시는 임마누엘의 하나님을 찬양합니다.

논문을 완성하기까지 세심한 배려와 관심으로 지도해주셨던 이상배 지도교수님께 진심으로 감사드리고, 바쁘신 와중에도 성심껏 심사해주신 양규식 교수님과 심준환 교수님, 고려대학교 김민기 교수님, 서울대학교 문병로 교수님께 감사드립니다.

멀리서 지켜봐 주신 경상대학교의 박연식 교수님, 이상욱 교수님, 김청 교수님, 이종룡 교수님, 전성근 교수님, 성길영 교수님, 이우재 교수님께도 감사의 말씀을 올립니다.

물심양면으로 지원해 주신 김일, 탁한호 교수님을 비롯한 졸업하신 여러 선·후배님들께도 감사드립니다. 논문을 작성하는 동안 아낌없는 조언과 격려로 힘을 주었던 성인선배와 은오선배, 그리고 연구실 올 때마다 함께 마음을 나누었던 창규씨와 나의 손과발이 되어 주었던 연구실 막내 혜정이에게도 깊은 감사를 드립니다.

지금까지 나를 이끌어주신 존경하는 박경식 사장님께 깊은 감사를 드리고, 가장 어렵고 힘들던 시기에 사랑과 격려로 함께 했던 구상도 이사님과 김종민 부장님에게도 깊은 사랑과 감사를 드립니다. 언제나 나에 대한 깊은 신뢰와 변함없는 우정을 주는 정종원 대리와 차근수 대리에게도 감사를 드립니다. 또한, 1년 365일 가족보다 더 많은 시간동안 얼굴을 맞대고 지내는 나의 삶의 비젼인 이서비스엔지니어링(주) 가족 모두에게 감사를 드립니다.

회사의 프로젝트를 진행하면서도 논문을 온전하게 작성할 수 있도록 물심양면으로 지원해주신 KT 박정진 과장님, IT-PLUS 이경구 과장님, 지영희 과장님께 감사드리고, 나에 대한 깊은 신뢰와 우정과 사랑으로 도와주신 KTN 남일우 과장님께 감사드립니다.

마음 따스한 밥 한끼 먹고 싶을 때 같이 먹어준 오래된 내 친구들에게 감사드립니다. 이들이 없었으면 내 인생은 훨씬 심심했을 것입니다.

길고 긴 힘들고 어려운 시간동안 언제나 존경과 사랑으로 나와 함께 해준 세상에서 가장 소중한 나의 사랑하는 아내 미라씨와 내가 살아가는 삶의 이유와 내 생에 가장 큰 기쁨을 선물한 아들 태환이와 내 마음속 깊이 가슴 뭉클한 감동과 행복을 선물한 딸 서연이에게 내 영혼의 진실한 사랑과 감사를 드립니다.

마지막으로 언제나 자식의 행복과 발전을 위해 어려움을 감추시고 평생 사랑으로 헌신하신 세상에서 내가 가장 사랑하고 존경하는 아버님, 어머님의 크신 은혜와 사랑에 머리 숙여 깊이 감사드리고, 사랑하는 형님 두분과 형수님 두분, 그리고 우리 이쁜이 조카들을 비롯한 모든 가족들에게 이 작은 결실을 바칩니다.