碩士學位論文

다양한 유형의 명령 처리를 지원하는 구동체 제어 미들웨어 연구

濟州大學校 大學院

컴퓨터工學科

李 昌 榮

2010年 02月

다양한 유형의 명령 처리를 지원하는 구동체 제어 미들웨어 연구

指導敎授 안 기 중

李 昌 榮

이 論文을 工學 碩士學位 論文으로 提出함

2010年 02月

李昌榮의 工學 碩士學位 論文을 認准함

審査委員長 _____ 印

委 員 _____ 印

委 員 _____ 印

濟州大學校 大學院

2010年 02月

A study of middleware for actuator control supports various command processing

Chang-Young Lee

(Supervised by professor Ki-Jung Ahn)

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Computer Engineering

2010. 02.

This thesis has been examined and approved.

Thesis director, _____

Thesis director, _____

Thesis director, _____

February 2010

Department of Computer Engineering

Graduate School

Cheju National University

감사의 글

지나고 나면 모든 것이 아쉽고 뒤돌아보면 훌쩍 지나버린 것 같이 느껴집니다. 사실 순간순간은 그렇게 흘러가지 않았건만, 시간이라는 것이 참으로 야속하게 느껴집니다. 돌아오지 않을 2년이라는 시간을 함께했던 분들께 고마움을 표현 하는 것으로 아쉬움을 달래야 하겠습니다.

학부를 졸업할 즈음에, 자신감이 부족해서 할 줄 아는 것이 별로 없다고 생각했습니다. 그런 부족한 저를 받아주시고 많은 가르침을 주신 김도현 교수님 감사드립니다. 돌아보면 그런 교수님의 은혜를 생각하기 보다는 제가 하고 싶은 데로 했던 일이 많았다는 생각이 듭니다. 머나먼 이국 땅, 호주에 계시면서도 바쁘신 가운데 항상 저희를 챙겨주시고 격려해 주심을 항상 감사드립니다. 조만간에 귀국하시고 건강한 모습으로 다시 뵙기를 기원합니다. 그리고 항상 관심을 가져주시고 학업에 정진할 수 있도록 많은 가르침을 주신 김장형 교수님, 안기중 교수님, 곽호영 교수님, 변상용 교수님, 이상준 교수님, 송왕철 교수님, 변영철 교수님 정말 감사드립니다. 그리고 교수님이 안 계시는 동안 저에게 상담을 해주시고 고충을 들어주신 학과사무실 이정하 선생님, 정은경 선생님 감사드립니다. 누나들의 충고가 저에게는 많은 도움이 되었습니다.

참 많은 시간을 연구실에서 교수님과 연구실 식구들과 함께 지냈습니다. 그래서 정도 많이 들었는데, 이제 얼마 안 남았다고 생각하니 힘들었던 순간, 기분 좋았던 순간 함께 지낸 이들이 많이 그리워질 것 같습니다. 연구실 생활을 함께 꾸려나가는 후배들 김경탁, 김영작, 윤치호, 강창봉, 황인철, 박종훈, 고문철, 바야르마그네, 윤인찬 너희들과 함께 공부하고 웃고 떠들던 시간들 많이 기억이 날꺼야. 모두 각자의 위치에서 최선을 다해서 성공하길 바랄게. 그리고 다음 학기에 졸업할 라지니 인도에서 와서 힘들게 대학원 생활 하는데 많이 도와주지 못해서 미안해. 무사히 학업 마치고 인도에서 잘 지내길 바랄게. 그리고 연구실 졸업생 여러분, 동기 김소현, 이영수, 박창홍 그리고 김용우, 김규리, 강민성, 이철식, 강경옥, 강문석, 강성철, 박영윤, 현진규 선배님들 부준필, 최익준, 강미영, 양현규, 문숙희, 고민정 후배들 모두 감사하고 돌아오는 설에 교수님과 한 자리에서 볼 수 있었으면 좋겠습니다. 모두들 항상 건강하고 하는 일마다 축복이 있기를 빌겠습니다.

대학원에 같이 입학한 둘도 없는 동기, 어려운 시간을 '야(Ya)'와 함께 해쳐왔던 지윤이형, 항상 도움을 주시는 김남식, 노영식 선배님 주경야독(晝耕夜讀)하시는 연준이형, 현탁이형, 도훈이형, 새신랑이 된 문수형, 학부 동기 대오 등 모두 감사드립니다.

마지막으로 정년을 앞두시고 나보다 더 일이 많으신 우리 아버지, 행복한 밥상으로 가족의 건강을 챙겨주시는 우리 어머니, 등산, 여행을 좋아하고 요즘엔 사진에 취미가 생긴 우리 형, 가족이라는 이름으로 편안한 보금자리가 있기에 나는 오늘도 든든합니다. 모두모두 감사합니다.

목 차

그림목차 ····· iii
국문초록 ····· v
영문초록 ····· vi
약어표 ····· vii

Ⅰ. 서 론 ····· 1
1. 연구배경 ····· 1
2. 연구 목적 및 방법 ····· 1
3. 논문 구성 ····· 2

Ⅱ. 관련 기술 연구 ····· 3
1. USN 미들웨어 ····· 3
2. 기존 구동체 제어 미들웨어 ····· 5

Ⅲ. 구동체 제어 미들웨어 설계 ····· 7
1. 전체 시스템 구성 ····· 7
2. 상황 인식 컴포넌트 설계 ····· 9
3. 구동체 제어 컴포넌트 설계 ····· 10

Ⅳ. 다양한 유형의 명령 처리 방안 ····· 12
1. 구동체 제어 명령의 유형 및 정의 ····· 12
2. 통합 제어 Open API 인터페이스를 통한 구동체 제어 처리 설계 ····· 14
1) Snapshot 제어 처리 설계 ····· 14
2) Continuous 제어 처리 설계 ····· 16
3) Temporal 제어 처리 설계 ····· 18
4) Spatial 제어 처리 설계 ····· 21
3. 상황 인지 기반의 구동체 제어 처리 설계 ····· 23
1) 이벤트 기반의 Snapshot 제어 처리 설계 ····· 23

2) 이벤트 기반의 Spatial 제어 처리 설계 ····· 26

Ⅴ. USN 기반의 구동체 제어를 위한 미들웨어 구현 ····· 29

1. 구현 환경 ····· 29
2. 구동체 정보 관리 응용 구현 ····· 30
3. 통합 제어 Open API 인터페이스 ····· 31
4. 구동체 제어 관리 응용 구현 ····· 33
5. 가상 구동체 응용 구현 ····· 35

Ⅵ. 다양한 구동체 제어 명령의 실험 및 성능 평가 ····· 37

1. 통합 제어 Open API 인터페이스를 통한 구동체 제어 명령 실험 ····· 37
   1) Snapshot 제어 명령 테스트 ····· 37
   2) Continuous 제어 명령 테스트 ····· 39
   3) Temporal 제어 명령 테스트 ····· 40
   4) Spatial 제어 명령 테스트 ····· 41
2. 컨텍스트 데이터 Interface를 통한 구동체 제어 명령 실험 ····· 43
   1) 이벤트 기반의 Snapshot 제어 명령 테스트 ····· 44
   2) 이벤트 기반의 Spatial 제어 명령 테스트 ····· 46
3. 성능 분석 ····· 47
2. 평가 및 고찰 ····· 49

Ⅶ. 결론 ····· 50

참고문헌 ····· 51

그 림 목 차

그림 1. USN 미들웨어 개념 모델 ····· 3
그림 2. USN 미들웨어의 기본적인 시스템 구성 ····· 4
그림 3. SANET 시스템 구조 ····· 5
그림 4. Sentire 프레임워크 데이터 흐름도 ····· 6
그림 5. 구동체 제어 미들웨어 및 전체 시스템 구성도 ····· 7
그림 6. 구동체 제어 미들웨어의 컴포넌트 구조도 ····· 8
그림 7. 상황 인식 컴포넌트 구조 ····· 9
그림 8. 구동체 제어 관리 컴포넌트 구조 ····· 10
그림 9. 구동체 제어 명령의 유형별 클래스 다이어그램 ····· 12
그림 10. 구동체 제어 명령의 유형별 특징 ····· 13
그림 11. Snapshot 제어 명령 시퀀스 다이어그램 ····· 1
그림 12. Snapshot 제어 명령 처리 순서도 ····· 1
그림 13. Continuous 제어 명령 시퀀스 다이어그램 ····· 1
그림 14. Continuous 제어 명령 처리 순서도 ····· 1
그림 15. Temporal 제어 명령 시퀀스 다이어그램 ····· 1
그림 16. Temporal 제어 명령 처리 순서도 ····· 2
그림 17. Spatial 제어 명령 시퀀스 다이어그램 ····· 2
그림 18. Spatial 제어 명령 처리 순서도 ····· 2
그림 19. 이벤트 기반의 Snapshot 제어 명령 시퀀스 다이어그램 ····· 2
그림 20. 이벤트 기반의 Snapshot 제어 명령 처리 순서도 ····· 2
그림 21. 이벤트 기반의 Spatial 제어 명령 시퀀스 다이어그램 ····· 2
그림 22. 이벤트 기반의 Spatial 제어 명령 처리 순서도 ····· 2
그림 23. 센서 하드웨어 장비 ····· 32
그림 24. 구동체 정보 관리 응용 실행화면 ····· 33
그림 25. 구동체 정보 XML ····· 35
그림 26. 통합 제어 Open API 인터페이스를 이용한 제어 명령 화면 ····· 3

그림 27. 웹 서비스를 이용한 통합 제어 Open API 인터페이스 ····· 3
그림 28. 구동체 제어 관리 응용 실행 화면 ····· 3
그림 29. 제어 메시지 모니터링 응용 실행화면 ····· 3
그림 30. 가상 구동체 응용 실행화면 ····· 3
그림 31. Snapshot 제어 명령 발생 ····· 3
그림 32. Snapshot 제어 명령을 통한 가상 구동체 제어 ····· 3
그림 33. Snapshot 제어 명령에 따른 결과 확인 ····· 3
그림 34. Continuous 제어 명령 발생 ····· 3
그림 35. Continuous 제어 명령을 통한 가상 구동체 제어 ····· 3
그림 36. Continuous 제어 명령에 따른 결과 확인 ····· 4
그림 37. Temporal 제어 명령 발생 ····· 4
그림 38. Temporal 제어 명령을 통한 가상 구동체 제어 ····· 4
그림 39. Spatial 제어 명령 발생 ····· 4
그림 40. Spatial 제어 명령을 통한 가상 구동체 제어 ····· 4
그림 41. Spatial 제어 명령에 따른 결과 확인 ····· 4
그림 42. 데이터 처리 및 룰 스펙 연산 응용 ····· 4
그림 43. 이벤트 기반의 Snapshot 제어 명령 발생 ····· 4
그림 44. 이벤트 기반의 Snapshot 제어 명령을 통한 가상 구동체 제어 ····· 5
그림 45. 이벤트 기반의 Snapshot 제어 명령에 따른 결과 확인 ····· 5
그림 46. 이벤트 기반의 Spatial 제어 명령을 통한 가상 구동체 제어 ····· 4
그림 47. 이벤트 기반의 Spatial 제어 명령에 따른 결과 확인 ····· 4
그림 48. 구동체 제어 명령의 처리 시간 ····· 4
그림 49. 구동체 제어 명령의 평균 처리 시간 ····· 4

국문초록

다양한 유형의 명령 처리를 지원하는 구동체 제어 미들웨어 연구

컴퓨터공학과 이창영
지 도 교 수 안기중

유비쿼터스 기술이 발전함에 따라 국방, 의료, 물류, 환경, 텔레매틱스 등 다양한 분야에 관한 연구가 도처에서 진행되고 있다. USN 미들웨어는 이들 시스템에 중추적인 역할을 한다. 아직까지는 센서 네트워크에서 센서들 간의 통신에 관한 범주의 연구가 대부분이지만 이기종의 하드웨어 장치 또는 구동체가 혼재한 상황을 고려한 유비쿼터스 서비스에 관한 연구가 점차적으로 증가하고 있다. 해외에서는 센서와 구동체를 네트워크로 연결한 SANET 시스템에서 구동체를 제어하기 위한 Sentire 미들웨어를 제시한 바가 있다. 본 논문에서는 구동체 제어를 위한 다양한 명령을 처리하는 제어 방법을 제시한다. 이를 바탕으로 USN 미들웨어의 범위에서 구동체 제어를 관리하는 구동체 제어 컴포넌트와 룰 기반 상황인식을 통해 미리 저장된 룰에 따라 상황을 발생하고 센서와 구동체 네트워크를 연계하여 구동체를 제어하는 상황인식 컴포넌트를 설계 및 구현한다. 이에 따라 구상한 USN 기반의 구동체 제어 미들웨어는 센싱데이터를 처리하는 동시에 상황을 인식하고 구동체를 제어하고, 관리자가 원하는 구동체를 제어 할 경우 이를 처리하는 다양한 형태의 제어에 대한 처리 방안을 제시할 수 있을 것으로 사료된다.

ABSTRACT

A study of middleware for actuator control supports various type of command processing

Lee, Chang-Young

Department of Computer Engineering

Graduate School

Jeju National University

Recently, Ubiquitous technology was researched various fields such as national defense, medical treatment, distribution, telematics. USN middleware is very important part of Ubiquitous services. Ubiquitous sensor networks contain various hardware devices or actuators with complicated circumstance. We expected to increase gradually even if the research on the category of communication between different sensors in sensor networks generally discussed now-a-days. For instance, several researches suggest that the Sentire middleware which is one of the USN middleware for actuator control on SANET system. This thesis is aims to show methods of various types of actuator control. Based on those proposed methods, we design and implement of actuator control management component and rule based context-aware component. Therefore, actuator control middleware which is based on USN supports actuator control by service application and context-aware. If the actuator controls order what administrator expects to do this, there is many types of feedback control service applications.

약어표

USN	Ubiquitous Sensor Network
GPS	Global Positioning System
SANET	Sensor and Actuator NETworks
QoI	Quality of Information management
XML	eXtensible Markup Language
WSDL	Web Services Description Language
UDDI	Universal Description, Discovery, and Integration
SOAP	Simple Object Access Protocol
HTTP	HyperText Transfer Protocol
IIS	Internet Information Server
IP	Internet Protocol
SQL	Structured Query Language

Ⅰ. 서 론

1. 연구 배경

유비쿼터스 기술의 발전과 더불어 USN 기반의 다양한 유비쿼터스 서비스 개발 및 적용 방안에 대한 연구가 활발하다. 그 중 핵심인 USN 미들웨어는 하드웨어와 서비스 응용의 중간에서 데이터의 오류 처리, 중복 제거, 필터링 등 복잡한 처리를 담당하여 유비쿼터스 서비스의 중추적인 역할을 담당한다. 지금까지 USN 미들웨어 기술은 데이터 처리에 중점을 두어 데이터 처리의 신속성과 정확성을 강조 하였으나, 앞으로 USN 미들웨어 기술은 수집되는 USN 데이터를 이용하여 상황 인지 및 행위제안, 나아가 상황에 따라 구동체 스스로 행위를 할 수 있는 지능형 유비쿼터스 서비스를 위한 미들웨어 기술 연구의 필요성이 증대될 것으로 예상된다. 또한 기존의 USN 미들웨어는 하나의 하드웨어를 대상으로 하였지만, 향후 센서 네트워크에 다양한 하드웨어 및 구동체가 공존하고 상황에 따라 서로 연계하여 동작하는 것을 고려해야 한다. 따라서, 본 논문에서는 수집한 USN 데이터로 룰 기반의 상황처리를 하여 구동체를 제어하거나 서비스 응용을 통해 사용자의 의도대로 다양한 구동체 제어 명령을 처리할 수 있는 구동체 제어 미들웨어를 설계하고 구현한다.

2. 연구 목적 및 방법

본 논문에서는 기존 USN 기반의 데이터 처리 중심의 미들웨어 기능보다는 미리 정의한 룰 데이터와 센싱 데이터를 비교하여 상황인식을 통해서 또는 서비스 응용에서 제어 명령을 처리하는 구동체 제어 처리 기능이 추가된 미들웨어 모델을 제시한다. 이를 위해 다양한 제어 명령을 효과적으로 처리하기 위하여 제어 명령의 형태를 설계하고 이를 적용한 구동체 제어 처리 컴포넌트와 상황인식 컴

포넌트를 설계하고 구현한다. 추가적으로, 구동체 정보를 XML 파일로 관리하기 위한 구동체 정보 관리 응용 및 서비스 응용에서 구동체 제어 처리 컴포넌트로 제어 명령을 전달하기 위한 통합 제어 Open API 인터페이스를 구현한다.

3. 논문 구성

서론에 이어 2장 관련 연구에서는 USN 미들웨어의 주요 기능과 관련 요구 기술을 살펴보고, 구동체 제어 미들웨어와 관련된 기존 연구에 대해 알아본다. 그리고 미들웨어의 표준 인터페이스인 웹서비스 기술에 대해 알아본다. 3장에서는 본 논문에서 제시하는 USN 기반의 구동체 제어를 위한 미들웨어의 전체적인 시스템 구조 및 상황인식 컴포넌트, 구동체 제어 컴포넌트를 설계하고, 4장에서는 다양한 구동체 제어를 위한 구동체 제어 명령을 설계한다. 5장에서 USN 기반의 구동체 제어 미들웨어를 구현한다. 6장에서는 5장에서 구현한 구동체 제어 미들웨어의 환경에서 4장에서 설계한 구동체 제어 명령을 테스트하고 성능평가를 한다. 마지막으로 7장에서 결론을 맺는다.

Ⅱ. 관련 기술 연구

본장에서는 기존의 데이터 처리 중심의 USN 미들웨어의 종류와 기술에 대해 알아보고, 기존의 구동체 제어 미들웨어인 SANET 시스템의 Sentire 미들웨어와 구동체 제어 미들웨어에서 필요한 기능에 대해 알아본다. 또한 본 논문에서 구동체를 제어하기 위한 표준 인터페이스인 웹서비스 기술에 대해 고찰한다.

1. USN 미들웨어

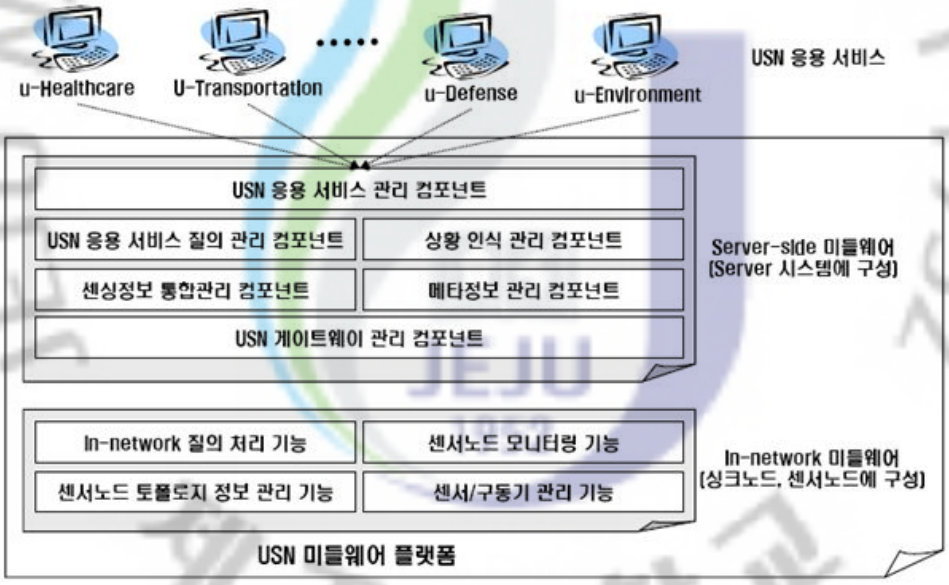

그림 1 USN 미들웨어 개념 모델

USN 미들웨어는 물리적으로 응용 서비스와 센서 네트워크 인프라 중간에 위치하며, 응용 서비스를 효율적으로 지원하기 위하여 서버 시스템에 위치하는 Server-side 미들웨어와 센서 네트워크 내부에서 센싱 정보를 효과적으로 관리하기 위하여 센서 노드 및 싱크 노드들에 위치하는 In-network 미들웨어로 구성된

다. 일반적으로 협의적 의미의 USN 미들웨어는 Server-side 미들웨어를 의미하며, 광의적 의미의 USN 미들웨어는 In-network 미들웨어를 포함하는 것을 의미한다.

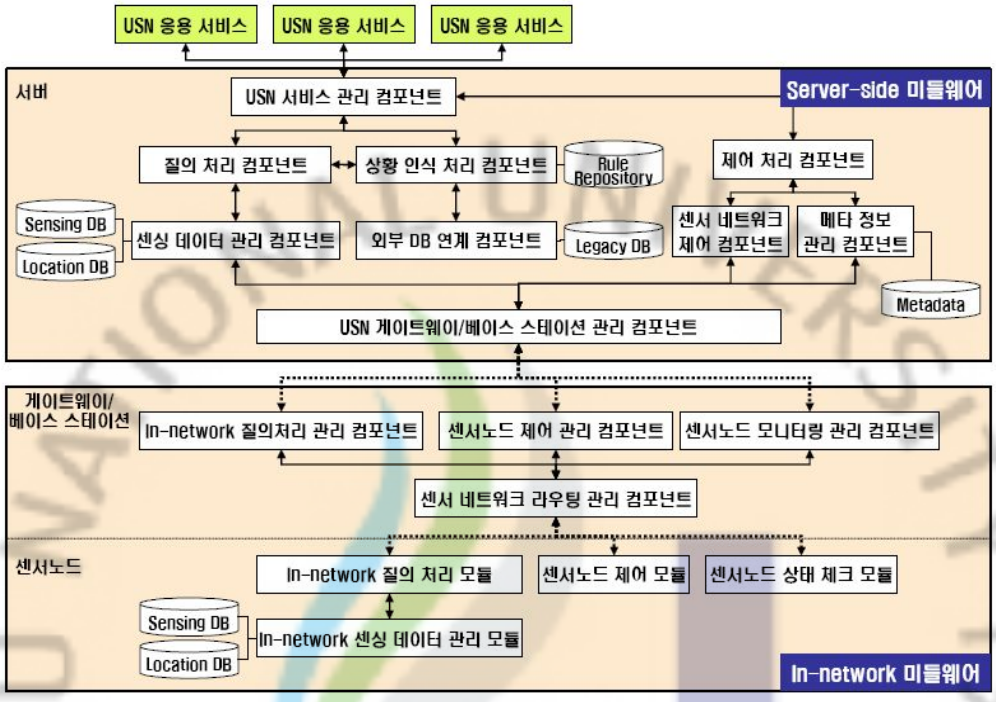

그림 2 USN 미들웨어의 기본적인 시스템 구성

본 논문에서는 광의적 의미의 USN 미들웨어를 기반으로 하고 있으며, 이러한 광의적 의미의 USN 미들웨어의 핵심 기술은 다음과 같이 분류 될 수 있다. 첫째는 USN 응용 서비스에서 제공되는 다양한 형태의 다중 질의들에 대한 분석 및 최적화 기능이고 둘째는 센서 네트워크로부터 끊임없이 주어지는 센싱 데이터의 획득, 가공, 저장, 분석 및 관리 기능, 셋째는 센싱 데이터 마이닝을 통한 상황 정보 생성 및 관리 기능, 마지막 넷째는 센서 네트워크에 관한 실시간 모니터링 및 동적/정적 메타 정보 관리 기능이다.[7] 아래 그림 2는 위에서 제시된 기능을 대부분 수용하여 Server-side 미들웨어와 In-network 미들웨어를 포함한 형태로서 서버, 게이트웨이, 센서 노드에 걸쳐 광범위하고 복잡하게 구성되는 USN 미들웨어 시스템의 예제를 보여준다. 그림 2에서 제시된 모든 컴포넌트들과 모듈들의 기능을 모두 지원할 수 있는 USN 미들웨어 시스템을 개발하는 것

이 가장 바람직하지만, 지금까지 개발 되어 왔던 대부분의 USN 미들웨어들은 주어지는 USN 응용 서비스에 따라서 일부 컴포넌트, 모듈의 기능만을 구현하고 있다.[8]

2. 기존 구동체 제어 미들웨어

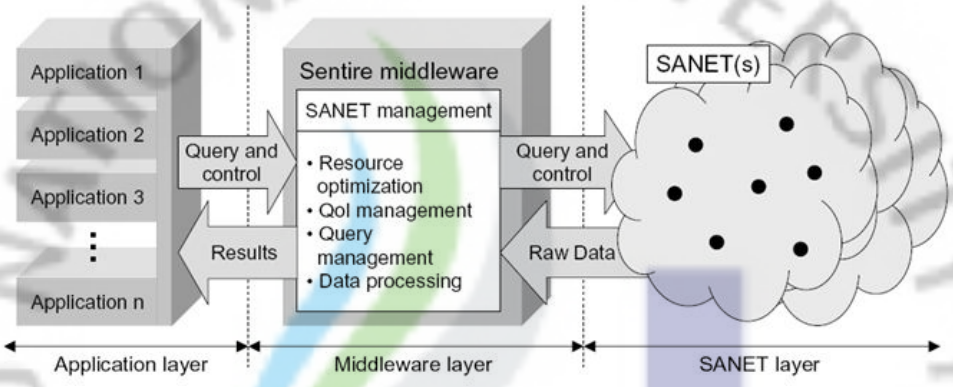

그림 3 SANET 시스템 구조

위의 그림 3은 센서와 구동체간의 네트워크인 SANET과 센싱 데이터 수집과 구동체 제어를 담당하는 미들웨어인 Sentire 미들웨어의 시스템 구조를 나타낸다. Sentire 미들웨어는 응용의 질의와 제어 메시지를 받아 SANET의 구동체를 제어하거나 센서로부터 센싱 데이터를 가져오게 된다. 여기서 SANET 시스템의 미들웨어는 장치의 전력량과 채널의 대역폭을 관리하는 자원관리(Resource management), 센서를 이용한 시스템에 대한 효율적인 질의 처리(Query management), 센싱된 데이터로부터 중요한 특징을 알아내는 처리 과정인 센서 데이터 처리(Sensor data processing), 규정된 수준 이상의 데이터 질을 제공하기 위해 센서를 구성하는 것을 관리하는 정보의 질 관리(Quality of information management, QoI), 환경의 규정이나 외부의 응답을 제공하는 구동체의 관리를 담당하는 구동체 관리(Actuator management) 기능들을 고려해야한다.[4] 이와 더불어 SANET 시스템의 미들웨어는 일반적인 컴퓨팅 패러다임인 상호 작동 가

능성(Interoperability), 재활용성(Reusability), 확장성(Extensibility), 적응성(Adaptability)을 모두 가져야 한다.[5]

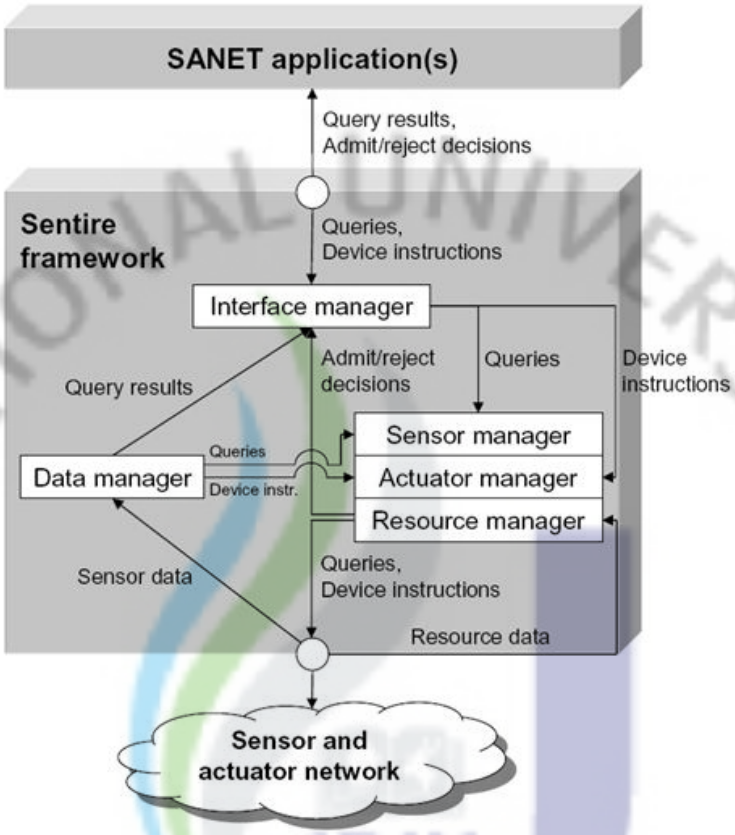

그림 4 Sentire 프레임워크 데이터 흐름도

위 그림 4는 센서와 구동체 네트워크와 SANET 응용 사이에서 Sentire 프레임워크의 주요 관리 기능과 데이터의 흐름을 나타낸다. 인터페이스 관리, 데이터 관리, 자원 관리, 센서 관리, 구동체 관리 컴포넌트는 Sentire 프레임워크에서 핵심적인 역할을 한다. 이중 구동체 관리는 SANET 응용이나 데이터의 관리자로부터 보내지는 구동체 동작 요구를 받아 들여 구동체를 제어한다.

Ⅲ. 다양한 유형의 명령 처리를 지원하는 구동체 제어 미들웨어 구조 설계

본 장에서는 다양한 유형의 명령 처리를 지원하는 구동체 제어를 위한 미들웨어의 전체적인 시스템 구성과 핵심 모듈인 룰 기반의 상황 인식 컴포넌트와 구동체 제어 관리 컴포넌트에 대해 설명한다.

1. 전체 시스템 구성

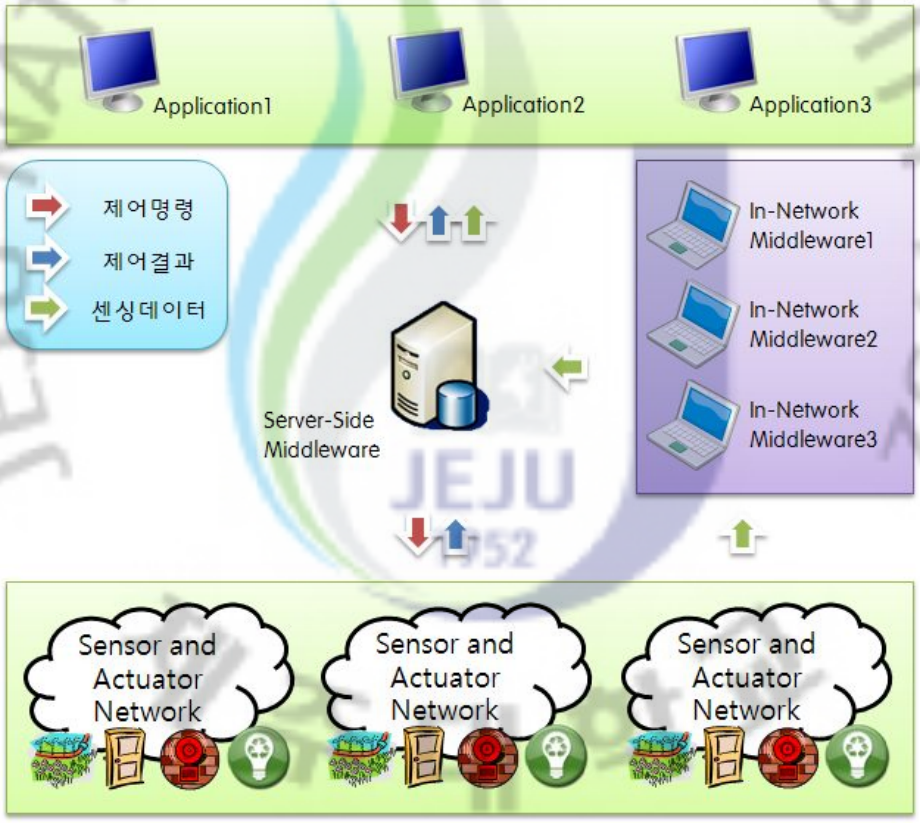

그림 5 구동체 제어 미들웨어 및 전체 시스템 구성도

본 논문에서 제안하는 구동체 제어 미들웨어는 그림 5의 구동체 제어 미들웨어 및 전체 시스템 구성도와 같이 구동체를 제어하는 시스템의 중추적인 역할을

한다. 구동체 제어 미들웨어는 등록된 구동체 정보를 기반으로 구동체를 관리하고 제어메시지를 전송한다. 이를 위해 사용자는 제어 메시지 생성에 필요한 규칙 데이터를 구동체 제어 미들웨어에 설정해야 한다. 구동체 제어 미들웨어는 수집되는 센싱 데이터를 등록된 규칙 데이터 중에서 조건 정보와 비교하고, 조건 정보간의 논리 연산인 규칙 연산 과정을 거쳐 연결된 구동체 목록을 확인한 다음 룰에서 정의한 것을 따라 구동체 제어 메시지를 생성한다. 제안된 구동체 제어 미들웨어의 핵심 컴포넌트는 컨텍스트 데이터를 수신하여 구동체 제어메시지를 생성하고 제어결과 메시지를 분석하여 적용하는 구동체 제어 관리 컴포넌트(Actuator Control Management Component), 미리 저장된 룰 데이터와 센싱 데이터를 비교하여 컨텍스트 데이터를 발생하는 상황 인식 및 처리 컴포넌트(Context-Aware Component), 센싱데이터를 수집하여 데이터 처리를 하여 서비스 응용에 제공하는 데이터 처리 컴포넌트(Data Process Component) 3가지이다.

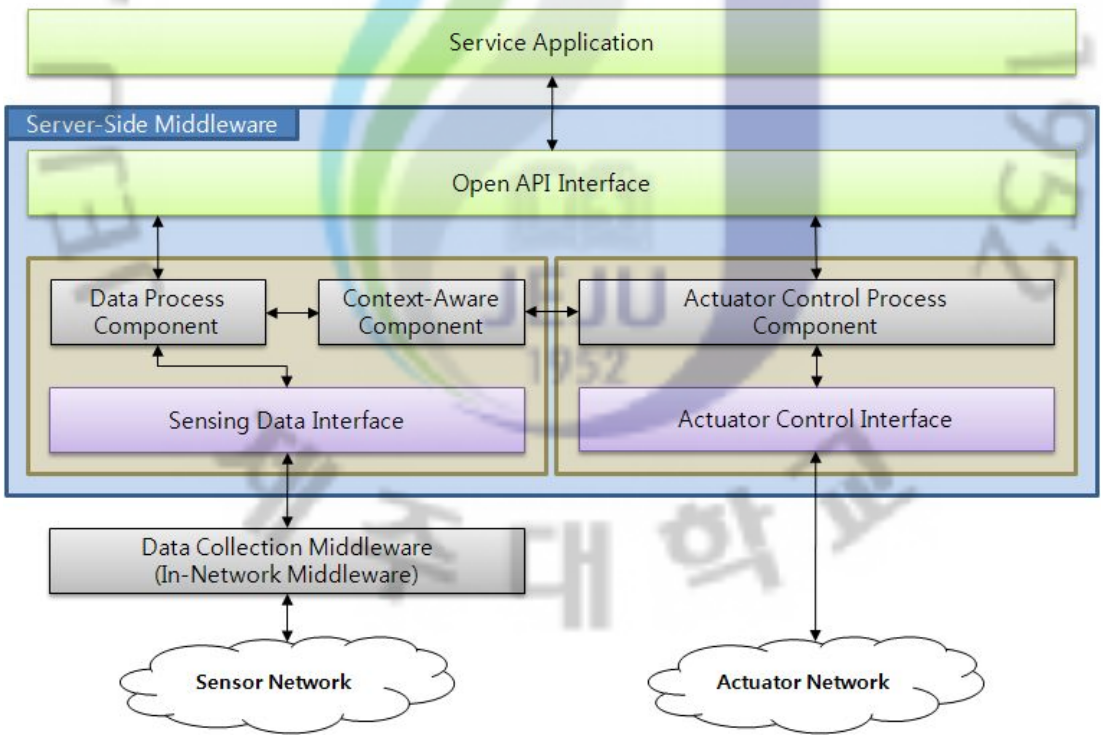

그림 6 구동체 제어 미들웨어의 컴포넌트 구조도

그 외에 구동체로 제어 메시지를 전달하고 결과를 수신하는 구동체 제어 인터

페이스(Actuator Control Interface)와 데이터 수집 미들웨어를 통하여 센싱 데이터를 수신하는 센싱 데이터 인터페이스(Sensing Data Interface), 서비스 응용으로 처리된 센싱데이터를 전달하고 구동체 제어명령을 수신하고 구동체 제어 결과를 나타내주는 닷넷 웹 서비스 기반의 Open API 인터페이스(Open API 인터페이스)로 구성된다. 그림 6은 구동체 제어 미들웨어의 컴포넌트 구조도이다.

2. 상황 인식 및 처리 컴포넌트 설계

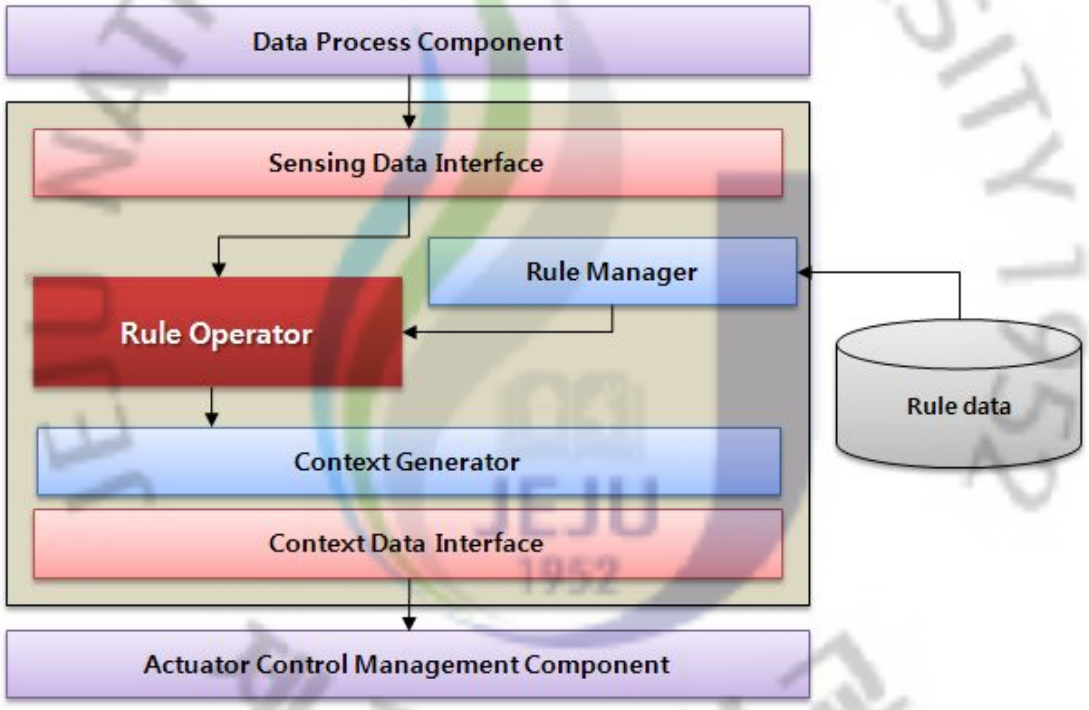

그림 7 상황 인식 컴포넌트 구조

구동체 제어 미들웨어의 상황 인식 컴포넌트의 구조는 그림 7과 같다. 상황 인식 컴포넌트는 데이터 처리 모듈에서 처리한 센싱 데이터의 특정 요소의 값과 룰 데이터의 스펙을 비교하여서 범위가 일치할 경우 룰에 정의된 상황을 발생한다. 룰 연산기(Rule Operator)에서 센싱데이터와 룰 데이터의 스펙의 값을 비교

하고 범위가 일치하면 해당하는 액션을 컨텍스트 생성기(Context Generator)로 전달한다. 컨텍스트 생성기는 액션 정보와 구동체 정보(Actuator Information)를 참고하여 특정 또는 복수의 구동체에 컨텍스트 정보를 생성하여 구동체 제어 컴포넌트로 전달한다.

3. 구동체 제어 컴포넌트 설계

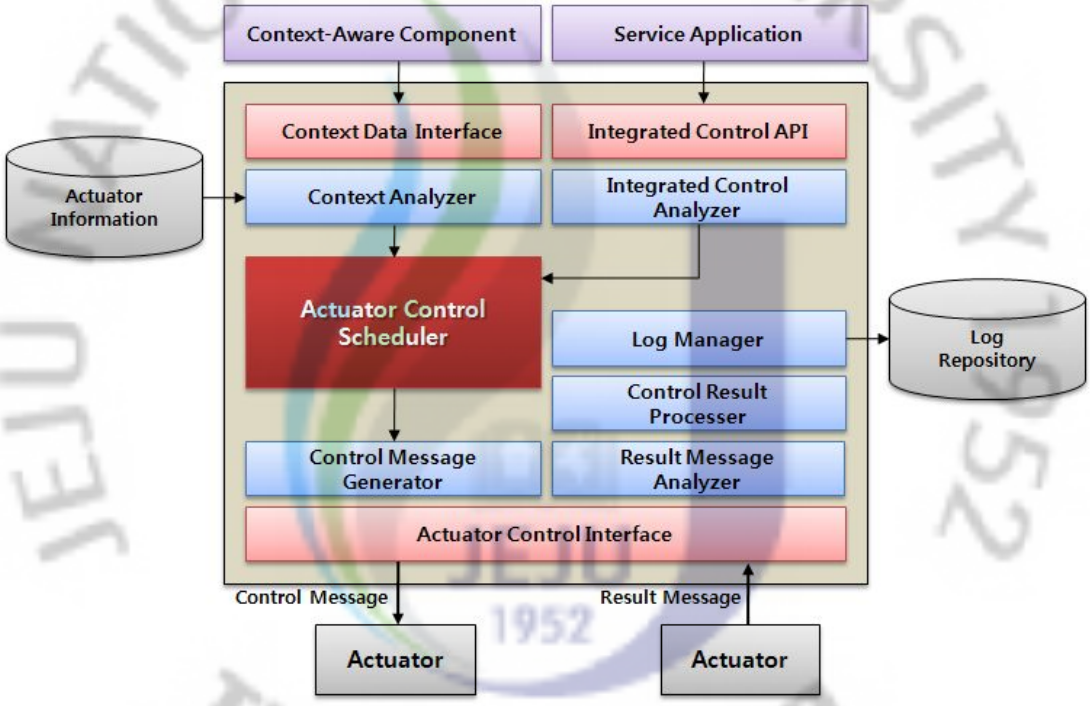

그림 8 구동체 제어 관리 컴포넌트 구조

구동체 제어 컴포넌트의 구조는 그림 8과 같다. 상황 인식 컴포넌트에서 전송된 컨텍스트 데이터(Context Data)를 컨텍스트 분석기(Context Analyzer)에서 분석하거나 서비스 응용에서 전송된 제어 명령을 통합 제어 분석기(Integrated Control Analyzer)에서 분석을 하여서 구동체 제어 스케쥴러(Actuator Control Scheduler)에 정렬한다. 구동체 제어 스케쥴러는 정렬된 구동체 제어 목록을 주기적으로 확인하여 제어할 시간에 방출한다. 제어 메시지 생성기(Control

Message Generator)에서는 스케쥴러에서 방출한 구동체 제어를 실제 구동체 제어 메시지로 변환하는 역할을 한다. 구동체 제어 메시지는 구동체 제어 인터페이스(Actuator Control Interface)를 통해 구동체에 전달된다.

구동체는 구동체 제어메시지에 따른 동작을 하고 나서 결과 메시지를 구동체 제어 인터페이스를 통해 전달한다. 결과 메시지 분석기(Result Message Analyzer)는 구동체 제어 결과를 분석하고 제어 결과 처리기(Control Result Processer)에서 제어의 성공 또는 실패 여부를 확인하고 그에 따라 미들웨어에서 관리하는 구동체 상태를 변경하여 준다. 로그 관리자는(Log Manager)는 이러한 일련의 과정에 대한 정보를 로그 저장소(Log Repository)에 저장한다.

Ⅳ. 다양한 구동체 제어 명령 처리 방안

본 장에서는 구동체 제어 미들웨어에서 사용할 다양한 구동체 제어 명령을 설계 한다. 서비스 응용에서 통합 제어 API를 통한 구동체 제어 명령과 상황 인식 모듈에서 발생한 컨텍스트 데이터를 분석한 제어명령의 유형 및 각 명령의 유형별 처리하는 과정을 설명한다.

1. 구동체 제어 명령의 유형 및 정의

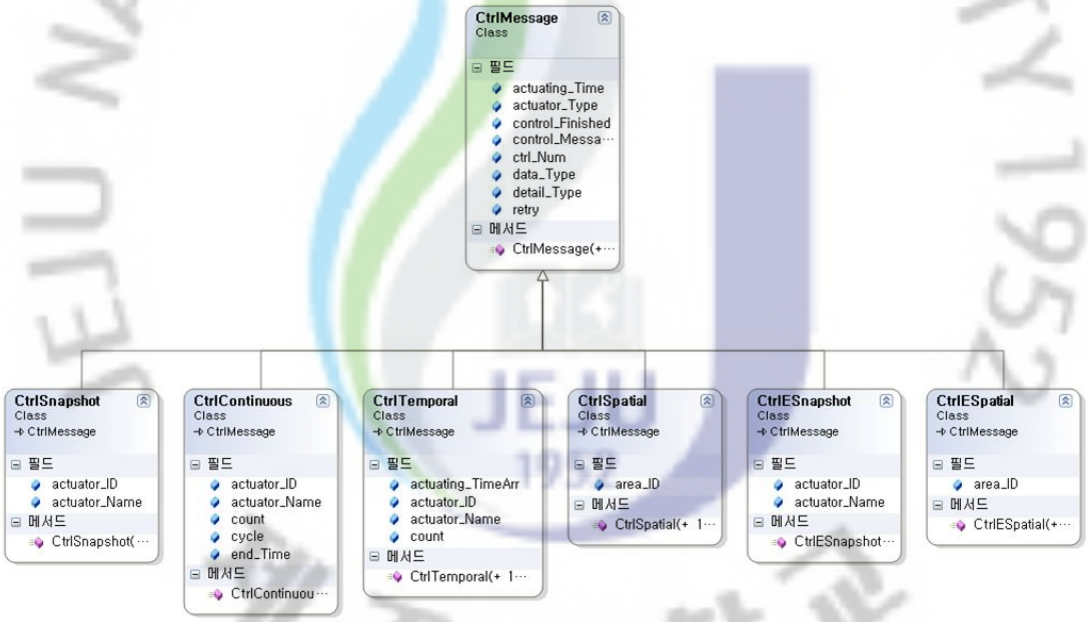

그림 9 구동체 제어 명령의 유형별 클래스 다이어그램

구동체 제어 명령의 유형은 그림 9의 구동체 제어 명령의 유형별 클래스 다이어그램에서처럼 총 6개의 형태로 정의한다. CtrlMessage에 6가지 제어 명령의 기본 공통 항목을 정의하고 각 명령을 나타내는 클래스에서 상속을 받아서 다른 부분을 추가한다. 기본적으로 서비스 응용에서 발생하는 Snapshot, Continuous,

Temporal, Spatial의 4가지 유형의 제어 명령이 있다. 추가적으로 상황인지 컴포넌트에서 발생하는 이벤트 기반의 Snapshot, Spatial 명령 2가지를 합쳐 총 6가지 명령을 정의 하였다.

그림 10 구동체 제어 명령의 유형별 특징

구동체 제어 명령의 종류 및 종류별 특징은 그림 10과 같다. (a) Snapshot 제어 명령은 특정 구동체를 대상으로 한 번의 동작 명령을 전달한다. 사용자가 원하는 하나의 구동체에 동작 명령을 한다. (b) Continous 제어 명령은 특정 구동체를 대상으로 일정한 주기로 복수의 동작 명령을 전달한다. 하나의 구동체의 ON, OFF와 같은 동작을 일정 주기로 반복할 때 사용한다. (c) Temporal 제어 명령은 특정 구동체를 대상으로 지정한 특정 시간에 동작 명령을 전달한다. 하나의 구동체를 사용자가 원하는 시간에 동작할 것을 예약할 수 있다. 마지막으로 (d) Spatial 제어 명령은 위치정보를 이용하여 특정 지역서비스 응용에서 발생하는 다수의 구동체를 대상으로 한 번의 동작 명령을 전달한다. 지역코드를 기준으

로, 한 지역에 있는 다수의 구동체에 동일한 동작을 시킨다. 상황 인식 컴포넌트에서 발생하는 (e) 이벤트 기반의 Snapshot 제어 명령은 센싱 데이터의 특정 요소가 룰에 정의한 스펙의 값과 일치하여 상황이 발생하였을 때에만 특정 구동체를 대상으로 한 번의 동작 명령을 전달한다. 예를 들면, 실내에서 수집한 센싱 데이터의 온도가 40도 이상이면 자동으로 창문을 여는 것과 같은 동작 명령을 한다. (f) 이벤트 기반의 Spatial 제어 명령은 센싱 데이터의 특정 요소가 데이터의 특정 요소가 룰에 정의한 스펙의 값과 일치하여 상황이 발생하였을 때 위치 정보를 이용하여 특정 지역 다수의 구동체를 대상으로 한 번의 동작 명령을 전달한다. 예를 들면, 건물에 화재가 발생하였을 때, 건물 내의 경보기, 스프링클러 등을 지역 ID를 이용하여 동시에 제어한다.

2. 통합 제어 Open API 인터페이스를 통한 구동체 제어 처리 설계

1) Snapshot 제어 처리 설계

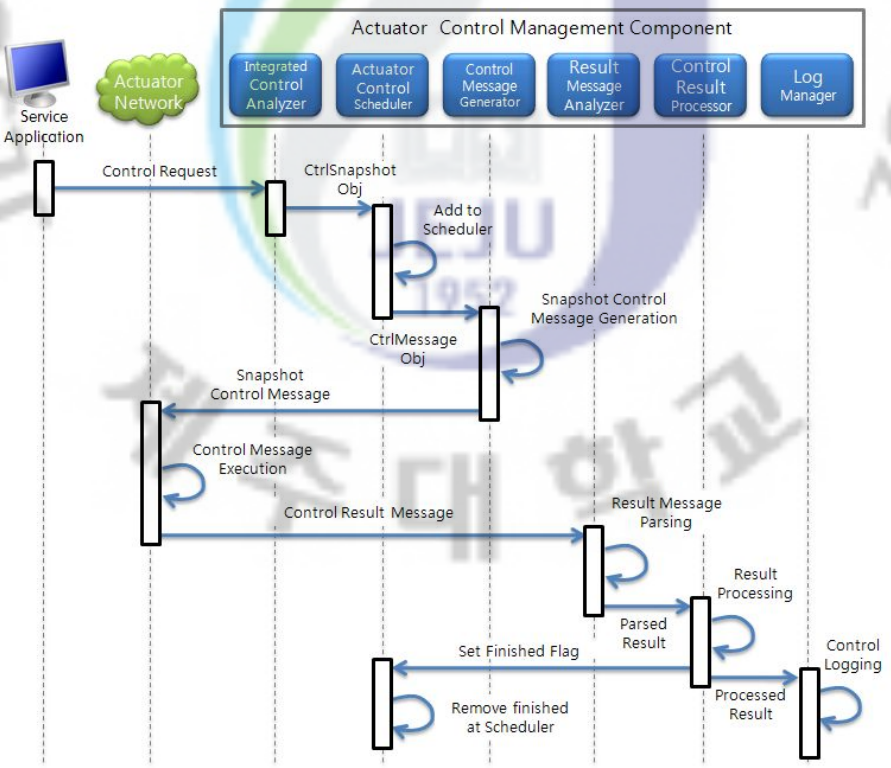

그림 11 Snapshot 제어 명령 시퀀스 다이어그램

그림 11은 Snapshot 제어 명령 시퀀스 다이어그램을 나타낸다. 서비스 응용에서 제어 명령을 발생하면 통합 제어 API 인터페이스를 통해 구동체 제어 관리 컴포넌트로 전달된다. 수신한 제어 명령은 통합 제어 분석기에서 분석하여 CtrlSnapshot 객체로 변환한다. CtrlSnapshot 객체는 리스트 형태의 구동체 제어 스케쥴러에서 상위 클래스인 CtrlMessage형태로 추가된다. 스케쥴러에서 제어할 순서가 되면 제어 메시지 생성기에서 Snapshot 제어 메시지로 변환된다. Snapshot 제어 메시지는 구동체 네트워크의 특정 구동체로 전송된다. 구동체는 제어 메시지를 실행하고 제어의 성패에 따라 제어결과 메시지를 반환한다. 결과 메시지 분석기에서는 제어결과 메시지를 분석하고 제어 결과 처리기로 전달한다. 제어 결과 처리기는 제어가 성공한 경우 구동체 제어 미들웨어에서 관리하는 구동체 목록의 상태를 변화시키고, 구동체 제어 스케쥴러에 해당 Snapshot 메시지의 Finished 플래그를 True로 변경하여 스케쥴러에서 자체적으로 삭제한다. 마지막으로 로그 관리자에 결과를 전달하여 로그 저장소에 저장한다.

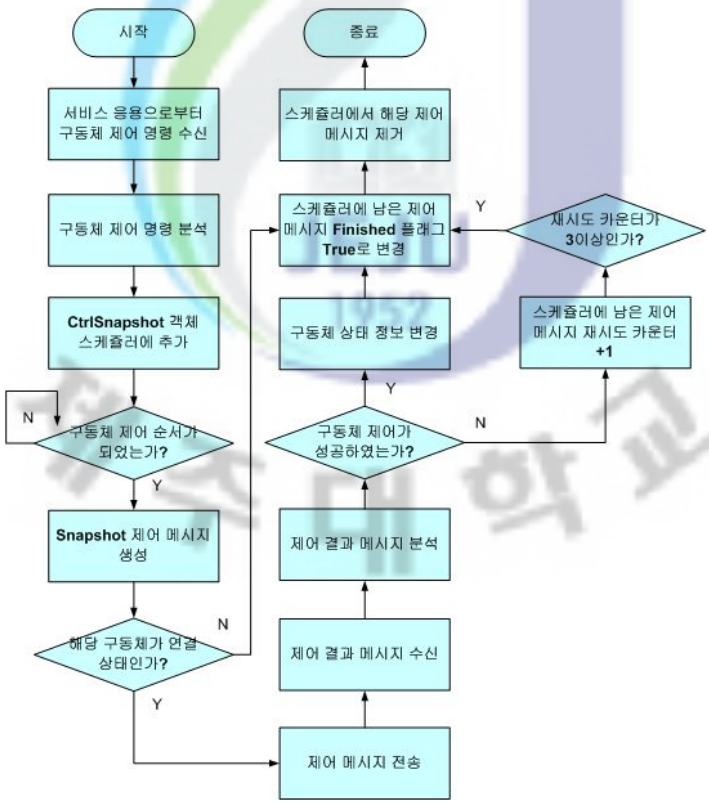

그림 12 Snapshot 제어 명령 처리 순서도

그림 12는 Snapshot 제어 명령 처리 순서도를 나타낸다. 먼저 서비스 응용에서의 제어 명령을 수신하고 구동체 제어명령을 분석하여 CtrlSnapshot 객체로 변환하여 구동체 제어 스케줄러에 추가한다. 스케줄러에서 제어 순서인지 확인이 되면 Snapshot 제어 메시지를 생성하고, 제어 대상인 구동체가 연결 상태인지 확인하고 제어 메시지를 전송한다. 대상 구동체로부터 제어 결과 메시지를 수신하고 제어결과 메시지를 분석 후 구동체 제어가 성공하였다면, 구동체의 상태정보를 변경하고, 스케줄러에 남은 해당 제어 메시지의 Finished 플래그를 True로 변경하여서 스케줄러에서 해당 제어 메시지를 제거 하도록 한다. 구동체 제어가 실패한다면 스케줄러에 남은 해당 제어 메시지의 재시도 카운터에 1을 더한다. 재시도 카운터가 3미만이면 구동체 제어 명령을 다시 재전송한다.

2) Continuous 제어 처리 설계

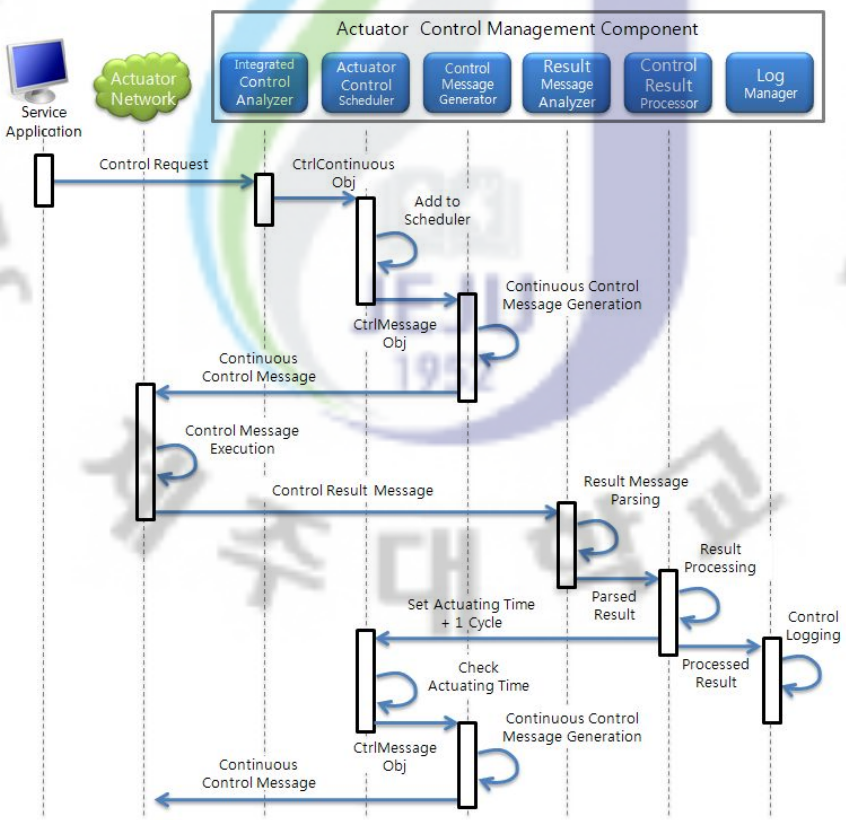

그림 13 Continuous 제어 명령 시퀀스 다이어그램

그림 13은 Continuous 명령 시퀀스 다이어그램을 나타낸다. 서비스 응용에서 제어 명령이 발생하면 통합 제어 API 인터페이스를 통해 구동체 제어 관리 컴포넌트로 전달된다. 수신한 제어 명령은 통합 제어 분석기에서 분석하여 CtrlContinuous 객체로 변환한다. CtrlContinuous 객체는 리스트 형태의 구동체 제어 스케줄러에서 상위 클래스인 CtrlMessage형태로 추가된다. 스케줄러에서 제어할 순서가 되면 제어 메시지 생성기에서 Continuous 제어 메시지로 변환된다. Continuous 제어 메시지는 구동체 네트워크(Actuator Notwork)의 특정 구동체로 전송된다. 구동체는 제어 메시지를 실행하고 제어의 성패에 따라 제어결과 메시지를 반환한다. 결과 메시지 분석기에서는 제어결과 메시지를 분석하고 제어결과 처리기로 전달한다. 제어 결과 처리기는 제어가 성공한 경우 구동체 제어 미들웨어에서 관리하는 구동체 목록의 상태를 변화시키고 구동체 제어 스케줄러에 해당 Contiuous메시지의 Actuating Time에 한 주기 만큼의 시간을 더하여 스케줄러에서 다음 주기에 제어가 발생한다. 마지막으로 로그 관리자에 결과를 전달하여 로그 저장소에 저장한다.

그림 14는 Continuous 제어 명령 처리 순서도를 나타낸다. 먼저 서비스 응용에서 제어 명령을 수신하고 구동체 제어명령을 분석하여 CtrlContinuous 객체로 변환하여 구동체 제어 스케줄러에 추가한다. 스케줄러에서 Actuating Time이 되었는지 확인이 되면 Continuous 제어 메시지를 생성하고, 제어 대상인 구동체가 연결 상태인지 확인하고 제어 메시지를 전송한다. 대상 구동체로부터 제어 결과 메시지를 수신하고 제어결과 메시지를 분석 후 구동체 제어가 성공한다면, 구동체의 상태정보를 변경하고 스케줄러에 남은 CtrlContinuous 객체의 Actuating Time에 한 동작 주기만큼의 시간을 추가하고, 그 시간이 CtrlContinuous 객체의 End Time을 초과 한다면 스케줄러에 남은 해당 제어 메시지의 Finished 플래그를 True로 변경하여서 스케줄러에서 해당 제어 메시지를 제거한다. Actuating Time이 End Time 이하라면 해당 구동체의 연결 상태를 확인하고 다시 제어 메시지를 전송한다.

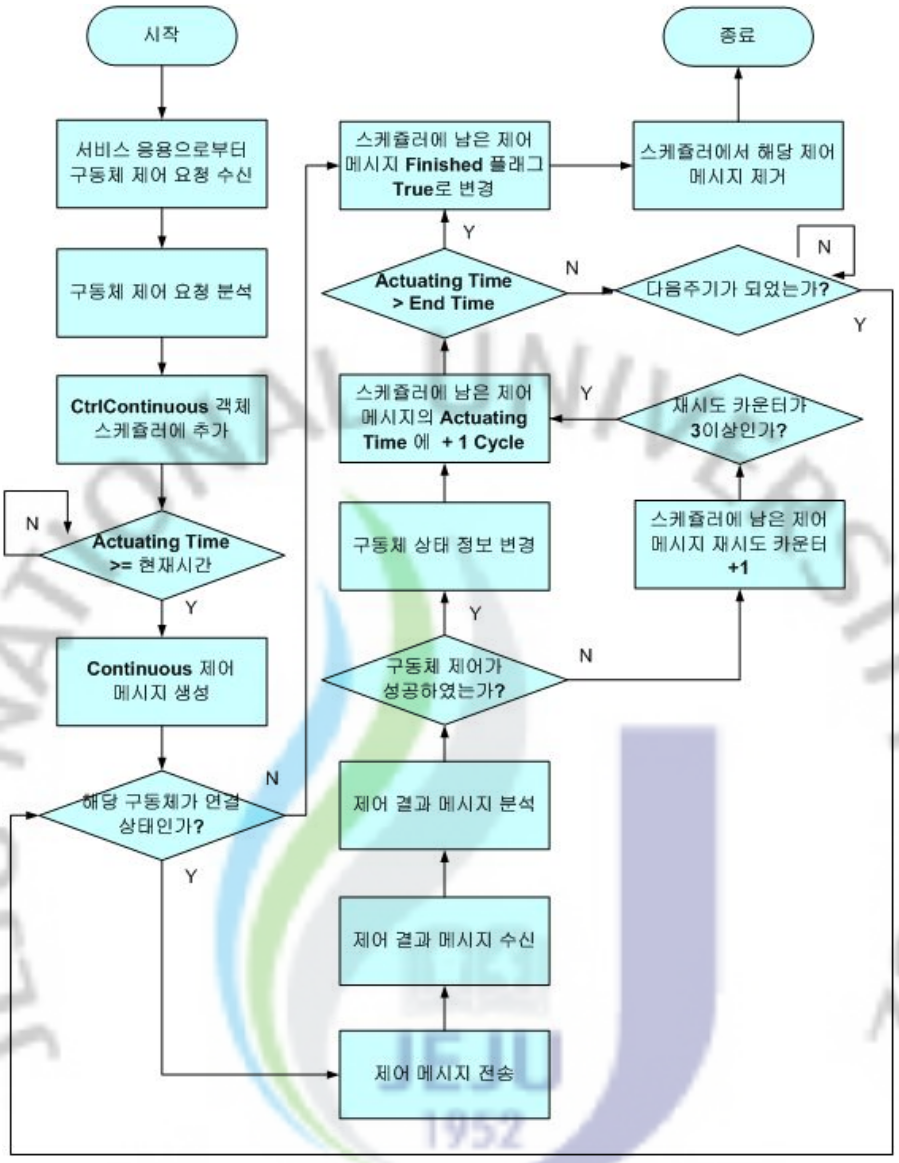

그림 14 Continuous 제어 명령 처리 순서도

3) Temporal 제어 처리 설계

그림 15는 Temporal 제어 명령 시퀀스 다이어그램을 나타낸다. 서비스 응용에서 제어 명령이 발생하면 통합 제어 API 인터페이스를 통해 구동체 제어 관리 컴포넌트로 전달된다. 수신한 제어 명령은 통합 제어 분석기에서 분석하여 CtrlTemporal 객체로 변환한다. CtrlTemporal 객체는 리스트 형태의 구동체 제어 스케쥴러에서 상위 클래스인 CtrlMessage 형태로 추가된다. 스케쥴러에서 제

어할 순서가 되면 제어 메시지 생성기에서 Temporal 제어 메시지로 변환된다. Temporal 제어 메시지는 구동체 네트워크의 특정 구동체로 전송된다. 구동체는 제어 메시지를 실행하고 제어의 성패에 따라 제어결과 메시지를 반환한다. 결과 메시지 분석기에서는 제어결과 메시지를 분석하고 제어 결과 처리기로 전달한다. 제어 결과 처리기는 제어가 성공한 경우 구동체 제어 미들웨어에서 관리하는 구동체 목록의 상태를 변화시키고, 구동체 제어 스케줄러에 해당 Temporal 메시지의 Actuating Time Array의 참조 인덱스를 1 더하여 다음 제어 시간에 맞추어 제어가 발생한다. 마지막으로 로그 관리자에 결과를 전달하여 로그 저장소에 저장한다.

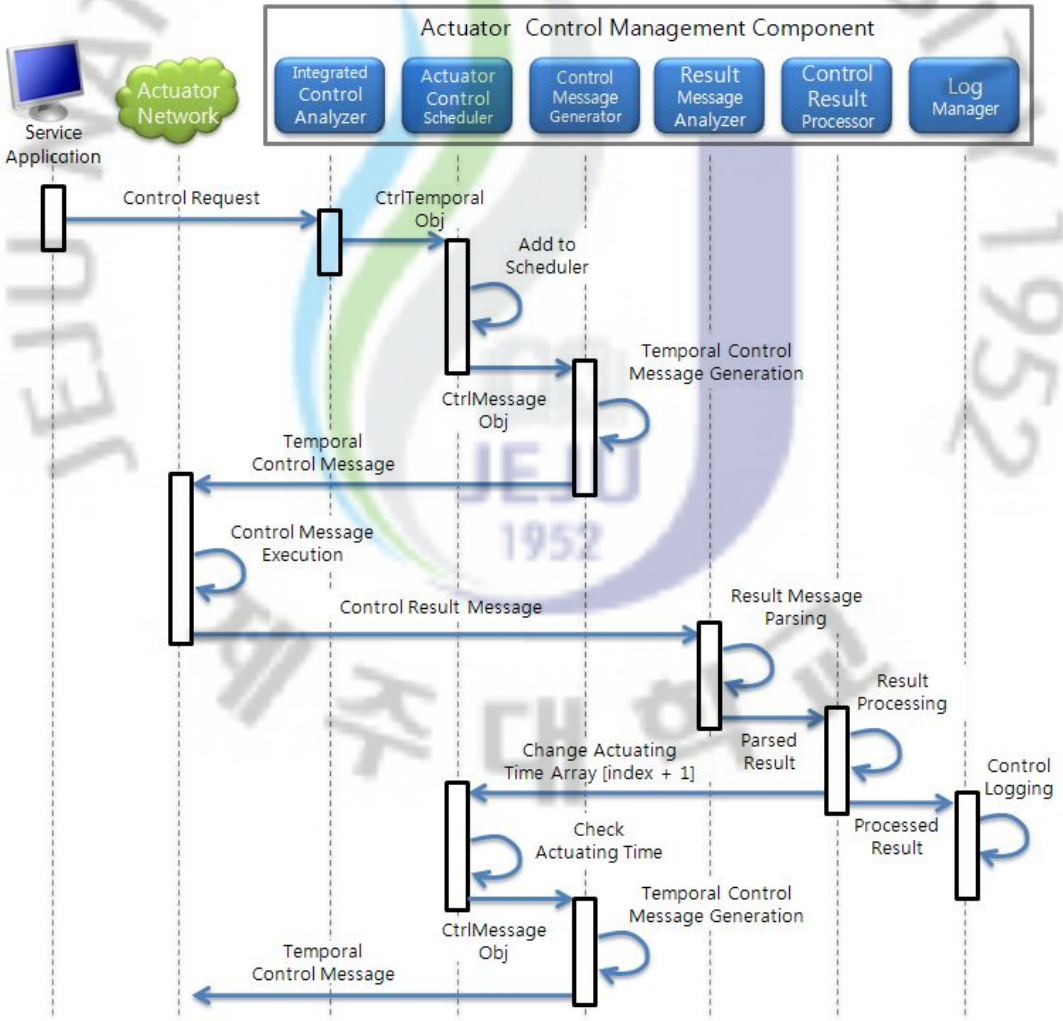

그림 15 Temporal 제어 명령 시퀀스 다이어그램

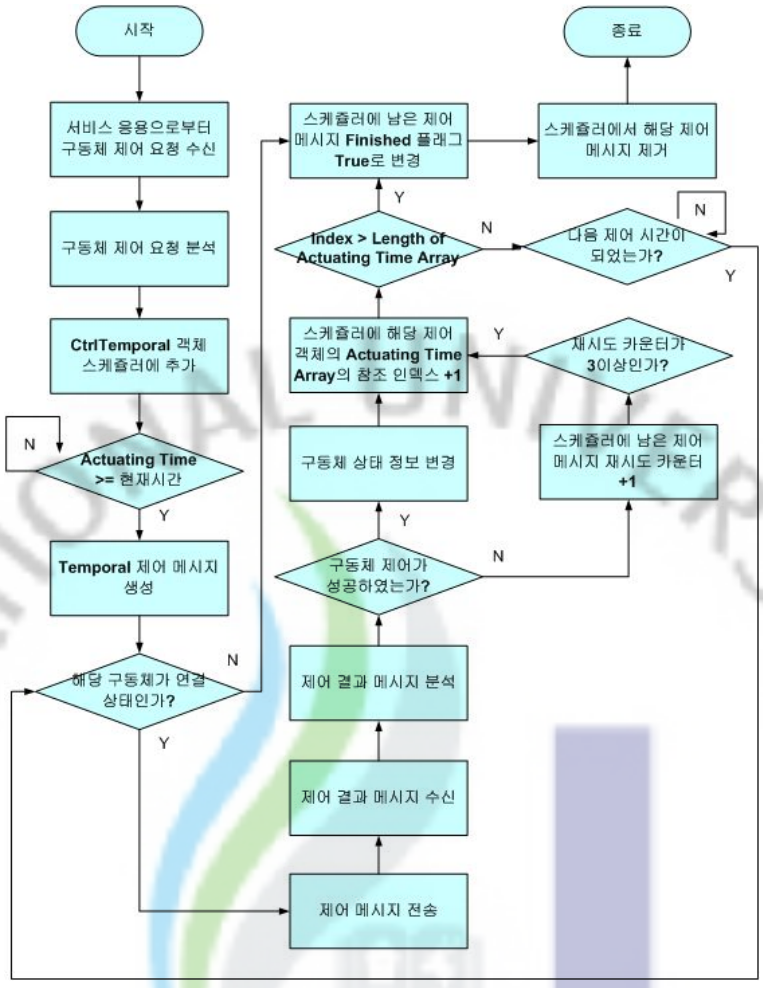

그림 16 Temporal 제어 명령 처리 순서도

그림 16은 Temporal 제어 명령 처리 순서도를 나타낸다. 먼저 서비스 응용에서 제어 명령을 수신하고 구동체 제어명령을 분석하여 CtrlTemporal 객체로 변환하여 구동체 제어 스케쥴러에 추가한다. 스케쥴러에서 Actuating Time이 되었는지 확인이 되면 Temporal 제어 메시지를 생성하고, 제어 대상인 구동체가 연결 상태인지 확인하고 제어 메시지를 전송한다. 대상 구동체로부터 제어 결과 메시지를 수신하고 제어결과 메시지를 분석 후 구동체 제어가 성공한다면, 구동체의 상태정보를 변경하고 스케쥴러에 남은 CtrlTemporal 객체의 Actuating Time Array의 참조 인덱스를 1 추가하고 참조 인덱스가 Actuating Time Array의 크기 이상이면, 스케쥴러에 남은 해당 제어 메시지의 Finished 플래그를 True로 변경하여서 스케쥴러에서 해당 제어 메시지를 제거 하도록 한다. Actuating Time

Array의 참조 인덱스가 Actuating Time Array의 크기보다 작다면 해당 구동체의 연결 상태를 확인하고 다시 제어 메시지를 전송한다.

4) Spatial 제어 처리 설계

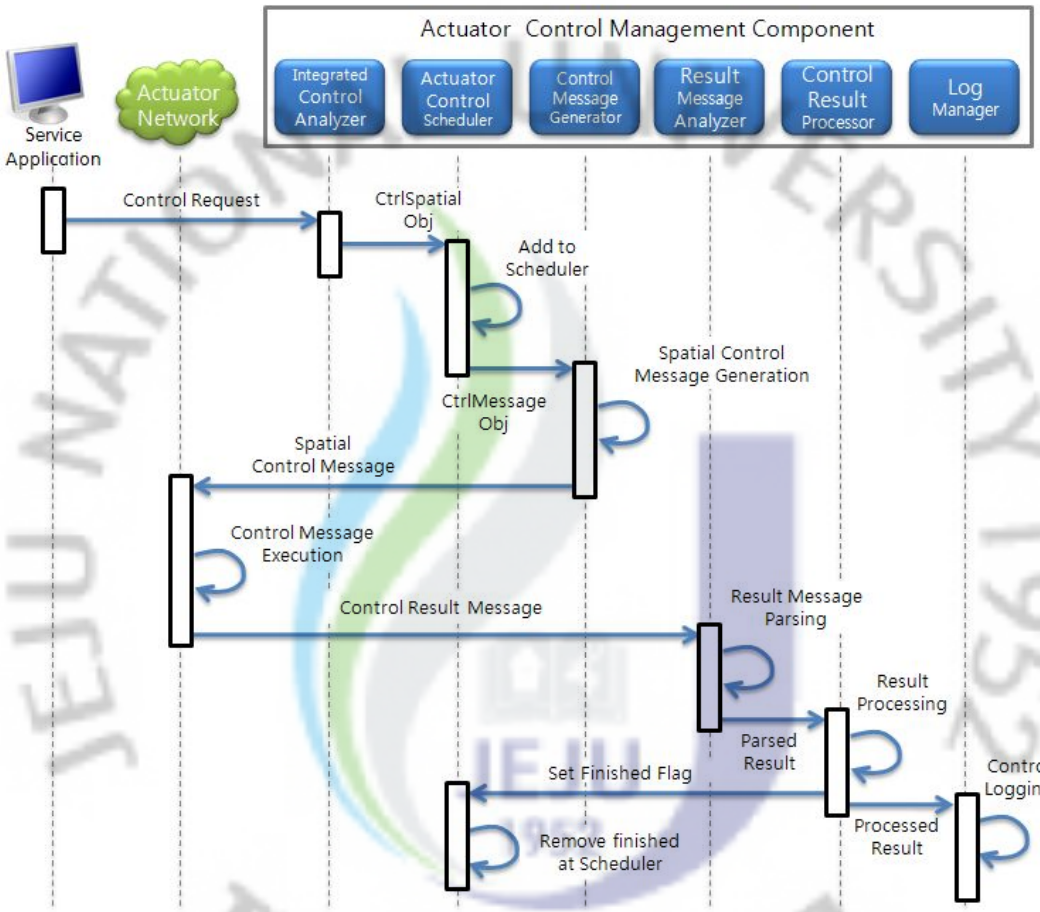

그림 17 Spatial 제어 명령 시퀀스 다이어그램

그림 17은 Spatial 제어 명령 시퀀스 다이어그램을 나타낸다. 서비스 응용에서 제어 명령이 발생하면 통합 제어 API 인터페이스를 통해 구동체 제어 관리 컴포넌트로 전달된다. 수신한 제어 명령은 통합 제어 분석기에서 분석하여 CtrlSpatial 객체로 변환한다. CtrlSpatial 객체는 리스트 형태의 구동체 제어 스케줄러에서 상위 클래스인 CtrlMessage형태로 추가된다. 스케줄러에서 제어할 순서가 되면 제어 메시지 생성기에서 Spatial 제어 메시지로 변환된다. Spatial

제어 메시지는 구동체 네트워크의 특정 지역의 복수의 구동체로 전송된다. 구동체는 제어 메시지를 실행하고 제어의 성패에 따라 제어결과 메시지를 반환한다. 결과 메시지 분석기에서는 제어결과 메시지를 분석하고 제어 결과 처리기로 전달한다. 제어 결과 처리기는 제어가 성공한 경우 구동체 제어 미들웨어에서 관리하는 구동체 목록의 상태를 변화시키고 구동체 제어 스케줄러에 해당 Spatial 메시지의 Finished 플래그를 True로 변경하여 스케줄러에서 자체적으로 삭제 하도록 한다. 마지막으로 로그 관리자에 결과를 전달하여 로그 저장소에 저장하도록 한다.

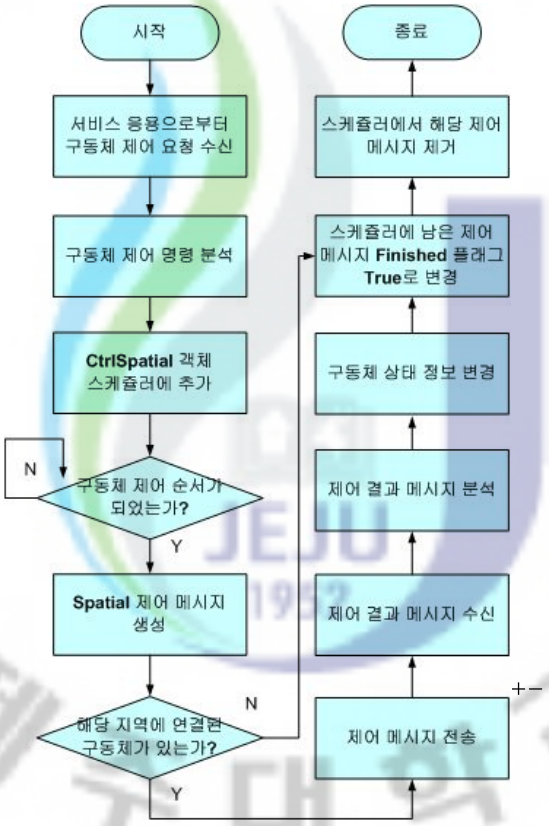

그림 18 Spatial 제어 명령 처리 순서도

그림 18은 Spatial 제어 명령 처리 순서도를 나타낸다. 먼저 서비스 응용에서 제어 명령을 수신하고 구동체 제어명령을 분석하여 CtrlSpatial 객체로 변환하여 구동체 제어 스케줄러에 추가한다. 스케줄러에서 제어 순서인지 확인이 되면

Spatial 제어 메시지를 생성하고 제어 대상인 지역에 연결 상태인 구동체가 있는지 확인하고 연결된 구동체에 제어 메시지를 전송한다. 대상 지역의 구동체로부터 제어 결과 메시지를 수신하고 제어결과 메시지를 분석 후 구동체 제어가 성공한 구동체들은 상태정보를 변경하고 스케쥴러에 남은 CtrlSpatial 객체의 Finished 플래그를 True로 변경하여서 스케쥴러에서 해당 제어 메시지를 제거하도록 한다.

3. 상황 인지 기반의 구동체 제어 처리 설계

1) 이벤트 기반의 Snapshot 제어 처리 설계

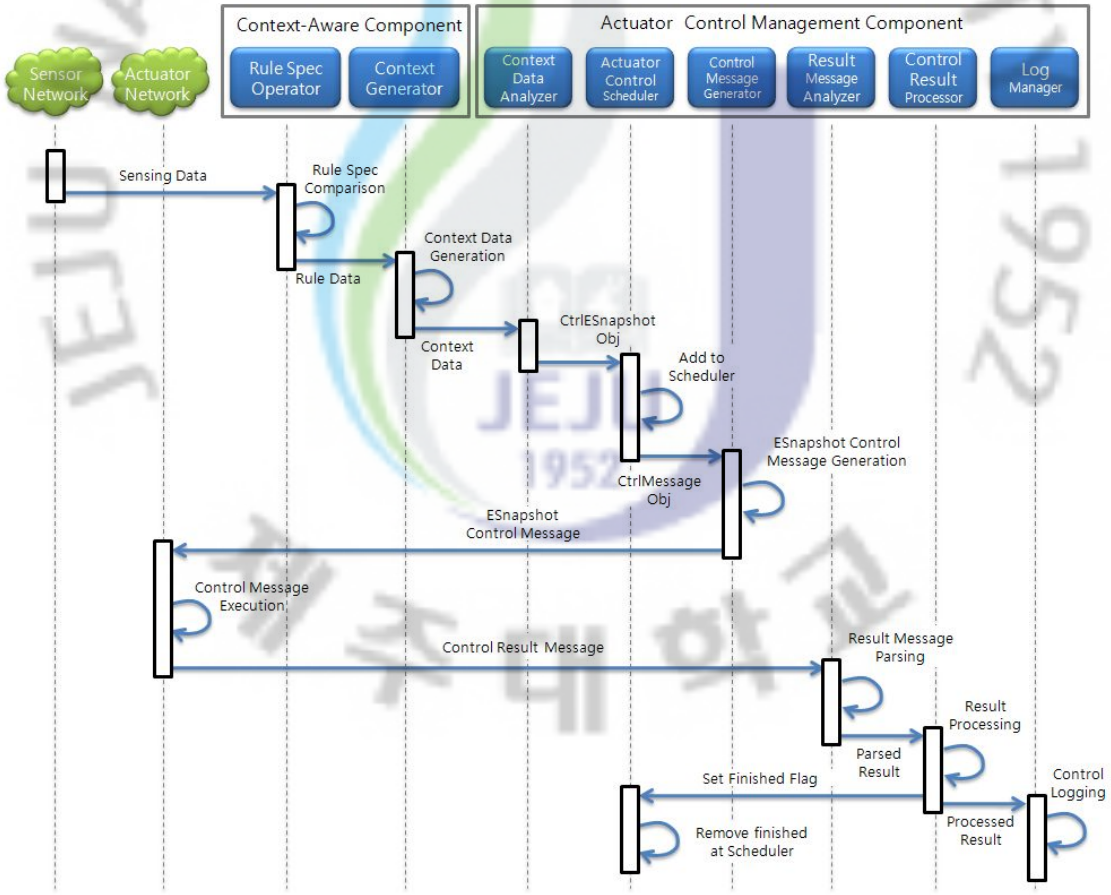

그림 19 이벤트 기반의 Snapshot 제어 명령 시퀀스 다이어그램

그림 19는 이벤트 기반 Snapshot 제어 명령 시퀀스 다이어그램을 나타낸다. 센서 네트워크에서 수집한 센싱데이터를 룰에 정의된 스펙을 룰 스펙 연산기(Rule Spec Operator)에서 비교한다. 스펙의 값과 센싱데이터의 특정요소 값이 일치 한다면 해당 규칙과 관련된 데이터를 컨텍스트 생성기(Context Generator)로 전달한다. 컨텍스트 생성기는 룰 데이터를 기반으로 컨텍스트 데이터를 생성하여 구동체 제어 관리 컴포넌트로 전달한다. 수신한 컨텍스트 데이터는 컨텍스트 데이터 분석기(Context Data Analyzer)에서 분석하여 CtrlESnapshot 객체로 변환한다. CtrlESnapshot 객체는 리스트 형태의 구동체 제어 스케줄러에서 상위 클래스인 CtrlMessage 형태로 추가된다. 스케줄러에서 제어할 순서가 되면 제어 메시지 생성기에서 ESnapshot 제어 메시지로 변환된다. ESnapshot 제어 메시지는 구동체 네트워크의 특정 구동체로 전송된다. 구동체는 제어 메시지를 실행하고 제어의 성패에 따라 제어결과 메시지를 반환한다. 결과 메시지 분석기에서는 제어결과 메시지를 분석하고 제어 결과 처리기로 전달한다. 제어 결과 처리기는 제어가 성공한 경우 구동체 제어 미들웨어에서 관리하는 구동체 목록의 상태를 변화시키고 구동체 제어 스케줄러에 해당 ESnapshot 메시지의 Finished 플래그를 True로 변경하여 스케줄러에서 자체적으로 삭제 하도록 한다. 마지막으로 로그 관리자에 결과를 전달하여 로그 저장소에 저장하도록 한다.

그림 20은 이벤트 기반의 Snapshot 제어 명령 처리 순서도를 나타낸다. 우선 미들웨어가 구동을 시작할 때 룰 저장소에 저장된 룰 데이터를 로드한다. 데이터 처리 컴포넌트를 통해 처리한 센싱데이터와 룰 데이터를 비교하여 센싱데이터의 특정 요소와 룰 데이터가 일치한다면 룰 데이터를 바탕으로 상황데이터를 생성한다. 구동체 제어 관리 모듈에서는 상황 데이터를 분석하고 구동체 제어를 요구하는지 확인한다. 구동체 제어를 요구한다면 구동체 제어 목록과 상황 데이터의 정보를 바탕으로 CtrlESnapshot 객체를 생성하여 구동체 제어 스케줄러에 추가한다. 스케줄러에서 제어 순서인지 확인이 되면 ESnapshot 제어 메시지를 생성하고 제어 대상인 구동체가 연결 상태인지 확인하고 제어 메시지를 전송한다. 대상 구동체로부터 제어 결과 메시지를 수신하고 제어결과 메시지를 분석 후 구동체 제어가 성공하였다면, 구동체의 상태정보를 변경하고 스케줄러에 남은 해당 제어 메시지의 Finished 플래그를 True로 변경하여서 스케줄러에서 해당 제어

메시지를 제거 하도록 한다. 구동체 제어가 실패하였다면 스케쥴러에 남은 해당 제어 메시지의 재시도 카운터에 1을 더한다. 재시도 카운터가 3미만이면 구동체 제어 명령을 다시 재전송한다.

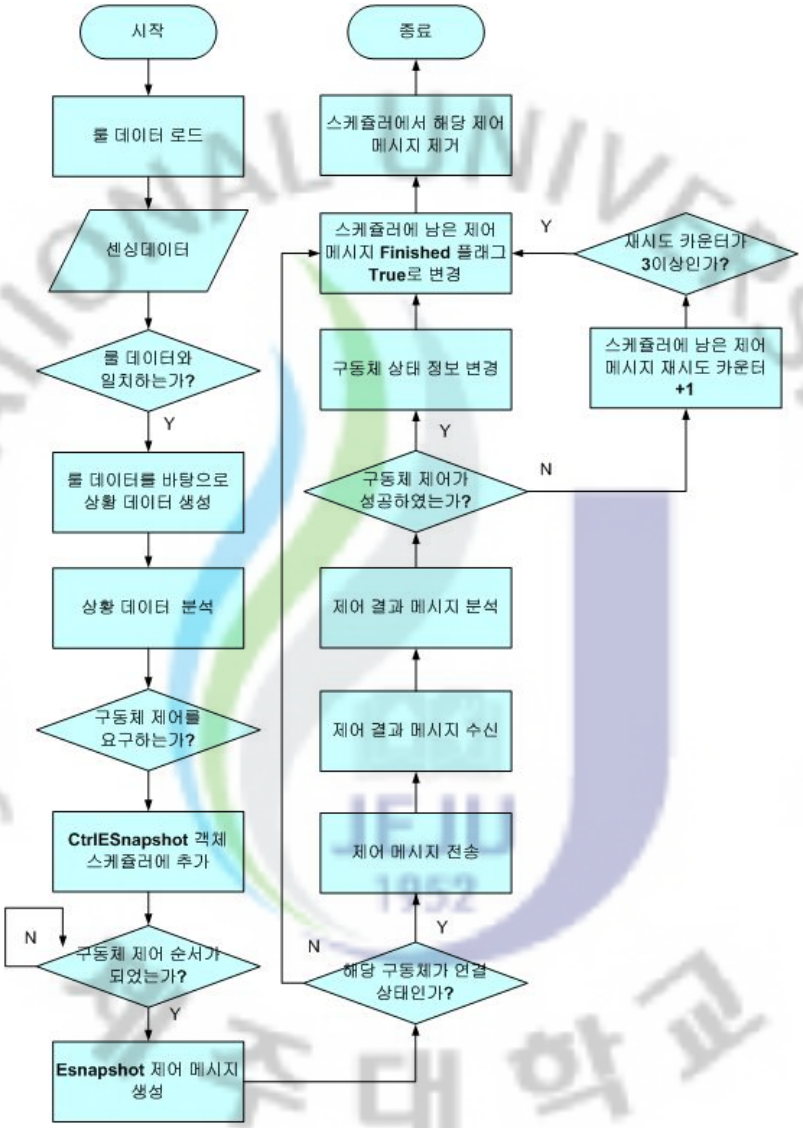

그림 20 이벤트 기반의 Snapshot 제어 명령 처리 순서도

2) 이벤트 기반의 Spatial 제어 처리 설계

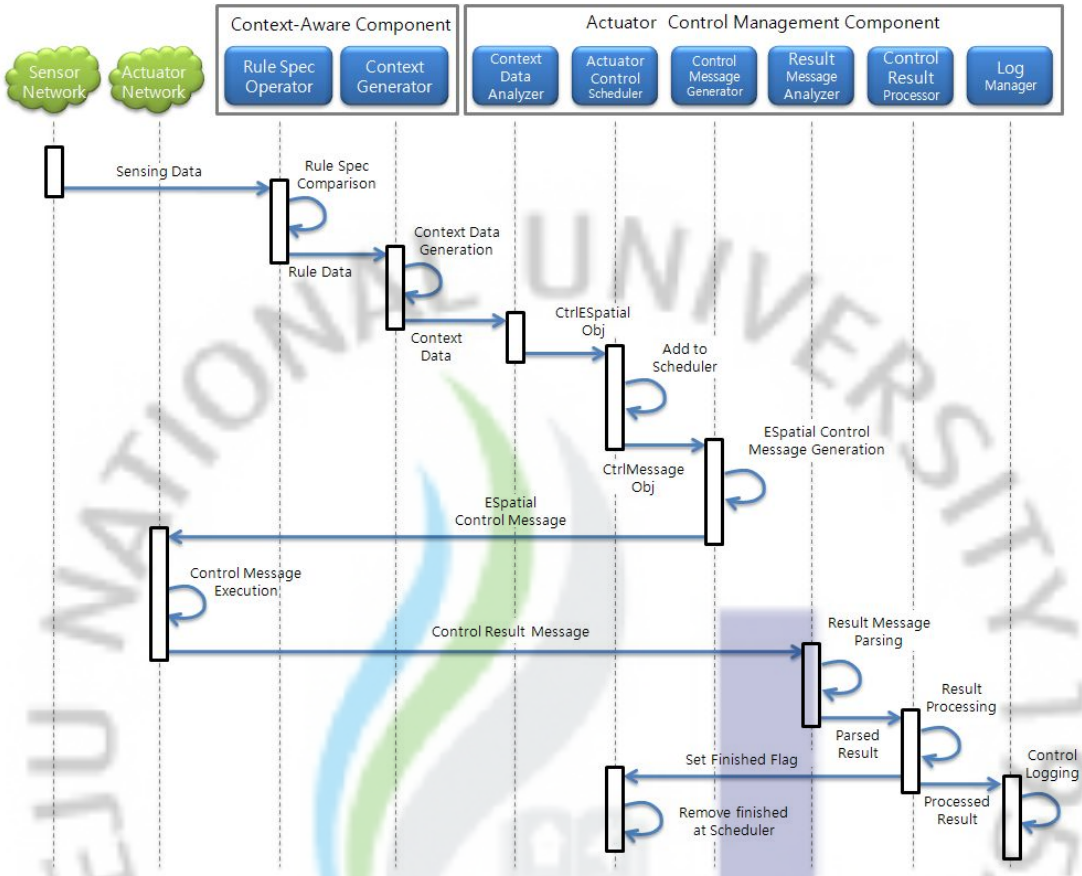

그림 21 이벤트 기반의 Spatial 제어 명령 시퀀스 다이어그램

그림 21은 이벤트 기반 Spatial 제어 명령 시퀀스 다이어그램을 나타낸다. 센서 네트워크에서 수집한 센싱데이터를 룰에 정의된 스펙을 룰 스펙 연산기(Rule Spec Operator)에서 비교한다. 스펙의 값과 센싱데이터의 특정요소 값이 일치 한다면 해당 규칙과 관련된 데이터를 컨텍스트 생성기(Context Generator)로 전달한다. 컨텍스트 생성기는 룰 데이터를 기반으로 컨텍스트 데이터를 생성하여 구동체 제어 관리 컴포넌트로 전달한다. 수신한 컨텍스트 데이터는 컨텍스트 데이터 분석기(Context Data Analyzer)에서 분석하여 CtrlESpatial 객체로 변환한다. CtrlESpatial 객체는 리스트 형태의 구동체 제어 스케줄러에서 상위 클래스인 CtrlMessage형태로 추가된다. 스케줄러에서 제어할 순서가 되면 제어 메시지 생성기에서 ESpatial 제어 메시지로 변환된다. ESpatial 제어 메시지는 구동체 네트

워크(Actuator Notwork)의 특정지역의 복수 구동체로 전송된다. 구동체는 제어 메시지를 실행하고 제어의 성패에 따라 제어결과 메시지를 반환한다. 결과 메시지 분석기에서는 제어결과 메시지를 분석하고 제어 결과 처리기로 전달한다. 제어 결과 처리기는 제어가 성공한 경우 구동체 제어 미들웨어에서 관리하는 구동체 목록의 상태를 변화시키고 구동체 제어 스케쥴러에 해당 ESpatial 메시지의 Finished 플래그를 True로 변경하여 스케쥴러에서 자체적으로 삭제 하도록 한다. 마지막으로 로그 관리자에 결과를 전달하여 로그 저장소에 저장하도록 한다.

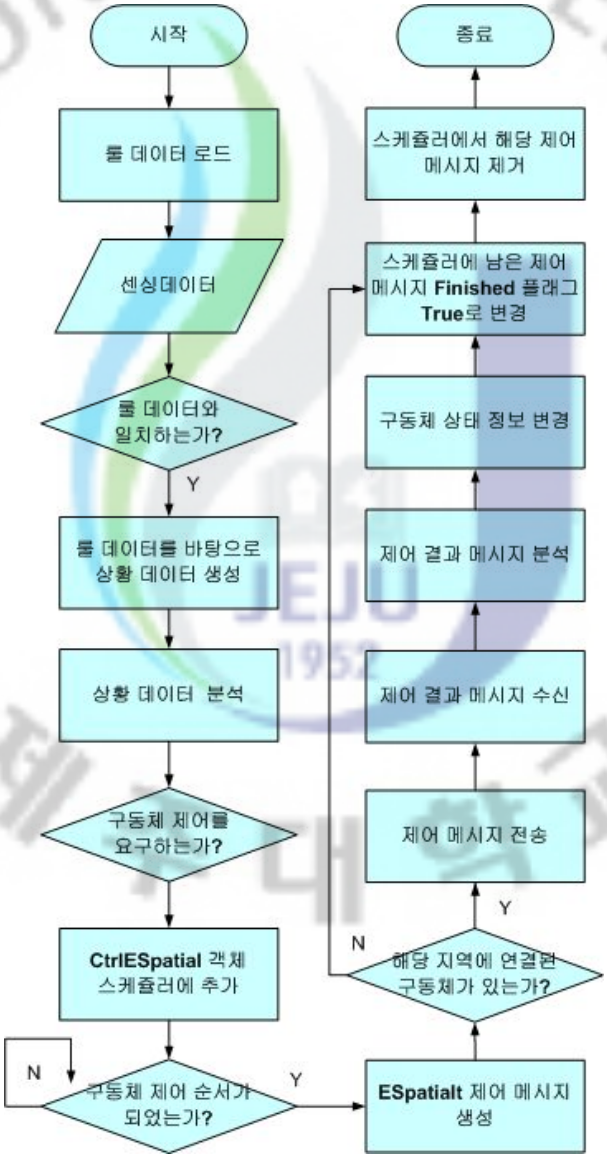

그림 22 이벤트 기반의 Spatial 제어 명령 처리 순서도

그림 22는 이벤트 기반의 Spatial 제어 명령 처리 순서도를 나타낸다. 우선 미들웨어가 구동을 시작할 때 룰 저장소에 저장된 룰 데이터를 로드한다. 데이터 처리 컴포넌트를 통해 처리한 센싱데이터와 룰 데이터를 비교하여 센싱데이터의 특정 요소와 룰 데이터가 일치한다면 룰 데이터를 바탕으로 상황데이터를 생성한다. 구동체 제어 관리 모듈에서는 상황 데이터를 분석하고 구동체 제어를 요구하는지 확인한다. 구동체 제어를 요구한다면 구동체 제어 목록과 상황 데이터의 정보를 바탕으로 CtrlESpatial 객체를 생성하여 구동체 제어 스케줄러에 추가한다. 스케줄러에서 제어 순서인지 확인이 되면 Spatial 제어 메시지를 생성하고 제어 대상인 지역에 연결 상태인 구동체가 있는지 확인하고 연결된 구동체에 제어 메시지를 전송한다. 대상 지역의 구동체로부터 제어 결과 메시지를 수신하고 제어결과 메시지를 분석 후 구동체 제어가 성공한 구동체들은 상태정보를 변경하고 스케줄러에 남은 CtrlSpatial 객체의 Finished 플래그를 True로 변경하여서 스케줄러에서 해당 제어 메시지를 제거 하도록 한다.

Ⅴ. USN 기반의 구동체 제어를 위한 미들웨어 구현

본 장에서는 위에서 설계한 구동체 제어를 위한 구동체 미들웨어의 구동체 제어 관리 컴포넌트, 통합 제어 Open API 인터페이스, 가상 구동체, 구동체 정보 관리 응용을 구현하고 테스트 환경을 구축하여 가상 구동체를 제어한다.

1. 구현 환경

그림 23 센서 하드웨어 장비

본 논문에서는 그림 23의 센서 하드웨어 장비인 Maxfor 사의 TIP700시리즈를 사용하고 GPS 수신기는 ASSEN GPS 610 모델을 사용하였다. 닷넷 프레임워크 기반의 C#을 이용하여 구동체 제어 관리 컴포넌트와 상황인식 컴포넌트를 구현한다. 또한 통합 제어 Open API 인터페이스는 ASP.NET 웹 서비스를 이용하여 구현하고 웹 서버로는 IIS를 사용한다. 데이터 백업을 위한 데이터베이스로는 MS-SQL 2005를 사용하였다.

2. 구동체 정보 관리 응용 구현

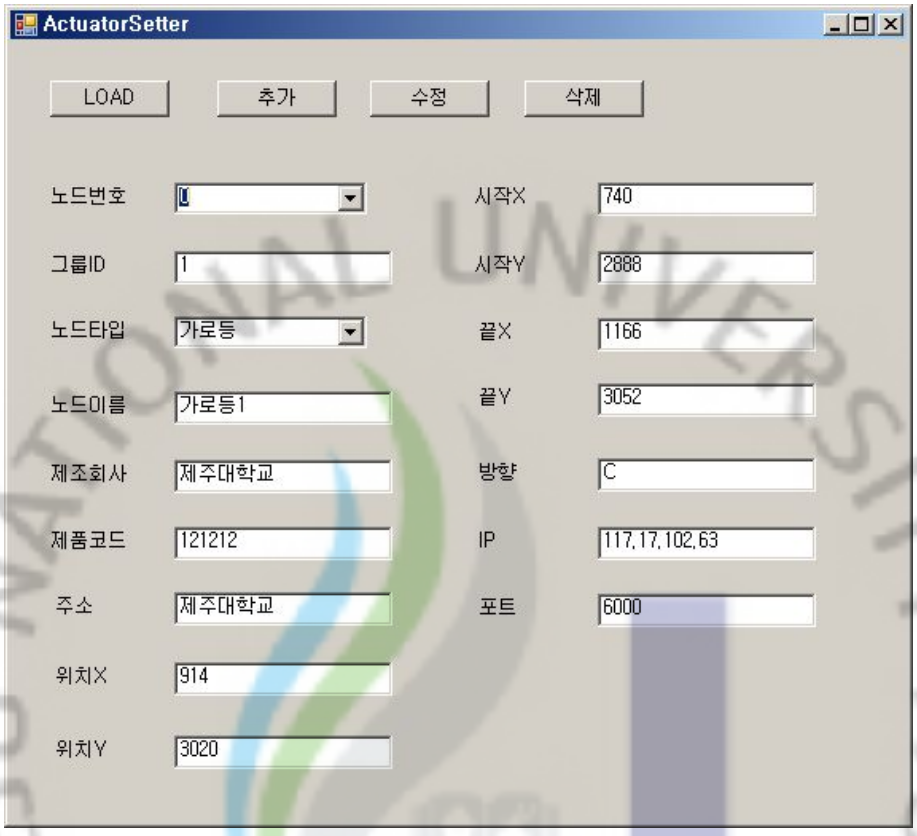

그림 24 구동체 정보 관리 응용 실행화면

그림 24는 구동체 정보 관리 응용의 실행화면이다. 그림은 가상의 도로시설물인 가로등을 수정하는 모습이다. 노드번호, 그룹ID, 노드타입, 노드이름, 제조회사, 제품코드, 주소, 위치 값, 반경에 해당하는 값들을 입력할 수 있다. 입력한 정보들은 XML파일로 저장되어 구동체 제어 관리 응용에서 사용할 구동체 목록을 구성한다. 그림 25는 이러한 형식으로 저장된 구동체의 정보를 나타내는 XML 파일의 내용이다.

```
- <Node>
    <GroupID>1</GroupID>
    <NodeType>가로등</NodeType>
    <NodeName>가로등1</NodeName>
    <Company>제주대학교</Company>
    <ProductCode>121212</ProductCode>
    <Address>제주대학교</Address>
    <PositionX>914</PositionX>
    <PositionY>3020</PositionY>
    <AreaStartX>740</AreaStartX>
    <AreaStartY>2888</AreaStartY>
    <AreaEndX>1166</AreaEndX>
    <AreaEndY>3052</AreaEndY>
    <Direction>C</Direction>
    <IP>117.17.102.63</IP>
    <Port>6000</Port>
  </Node>
```

그림 25 구동체 정보 XML

3. 통합 제어 Open API 인터페이스

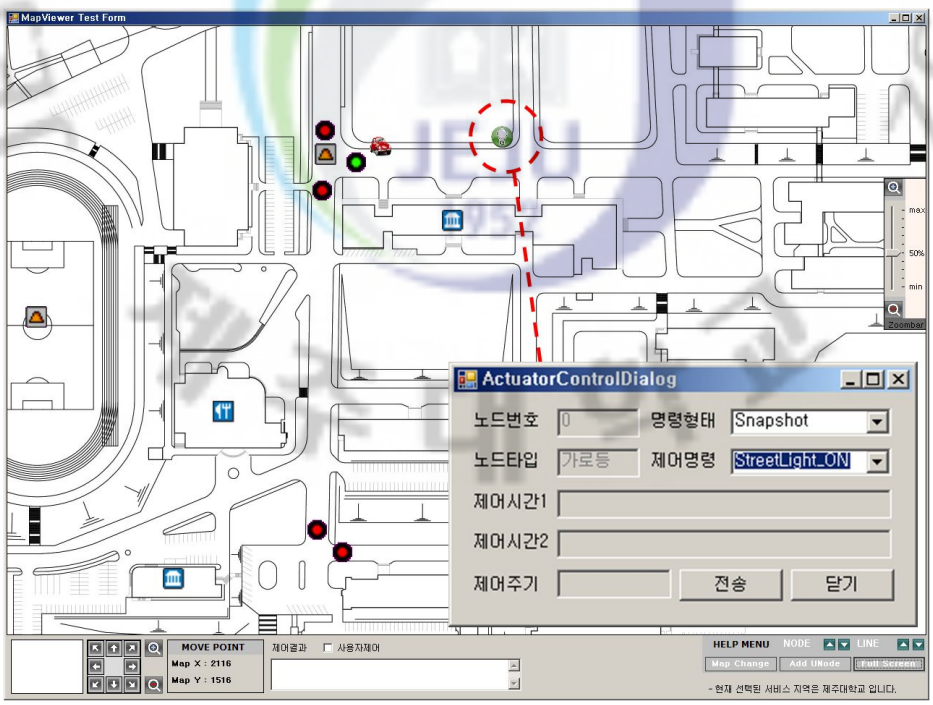

그림 26 통합 제어 Open API 인터페이스를 이용한 제어 명령 화면

그림 26은 서비스 응용에서 통합 제어 Open API 인터페이스를 이용한 제어 명령을 나타낸다. 서비스 응용에서 노드번호가 0인 가로등을 우클릭 하였을 경우 ActuatorControlDialog창이 생겨서 해당 구동체에 제어 명령을 할 수 있다. 그림에서는 Snapshot 제어의 형태로 StreetLight_ON의 제어명령으로 '0번 가로등을 켜라'는 제어 명령을 발생하고 있다.

그림 27은 서비스 응용에서의 제어명령을 수용할 웹 서비스로 구현된 통합 제어 Open API 인터페이스이다. 통합 제어 Open API 인터페이스는 서비스 응용에서 명령할 수 있는 Snapshot, Continuous, Temporal, Spatial 제어 명령을 수용할 수 있다. 그림의 오른쪽은 Snapshot 제어 명령에 필요한 인자들을 나타내고 있다. Snapshot 제어 명령에 필요한 인자로 제어명령 형태, 구동체 이름, 구동체 형태, 제어 시간, 실질적인 제어 내용, 구동체 ID의 값을 전달한다.

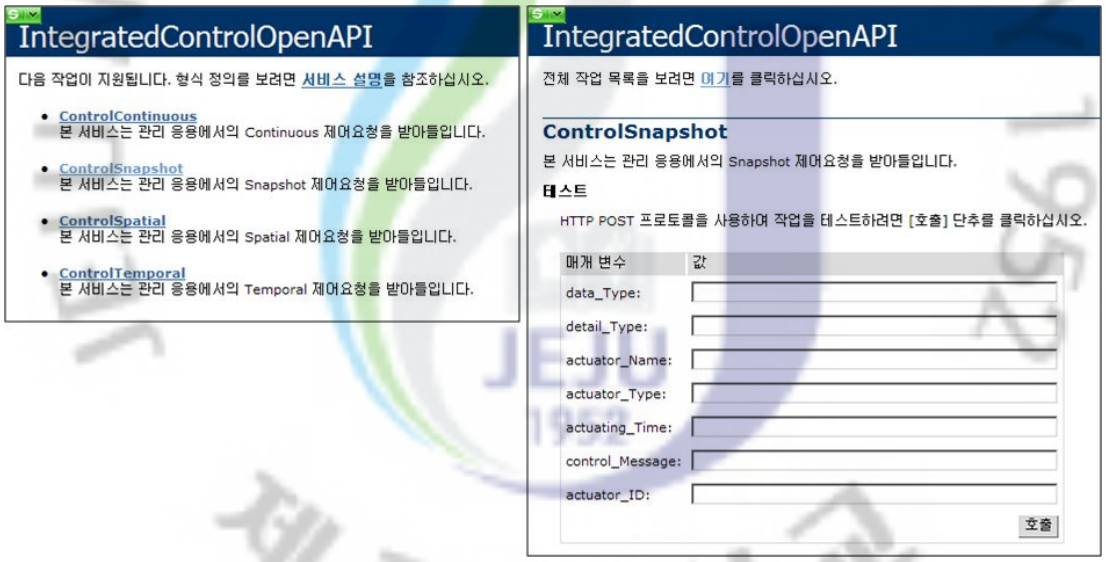

그림 27 웹 서비스를 이용한 통합 제어 Open API 인터페이스

4. 구동체 제어 관리 응용 구현

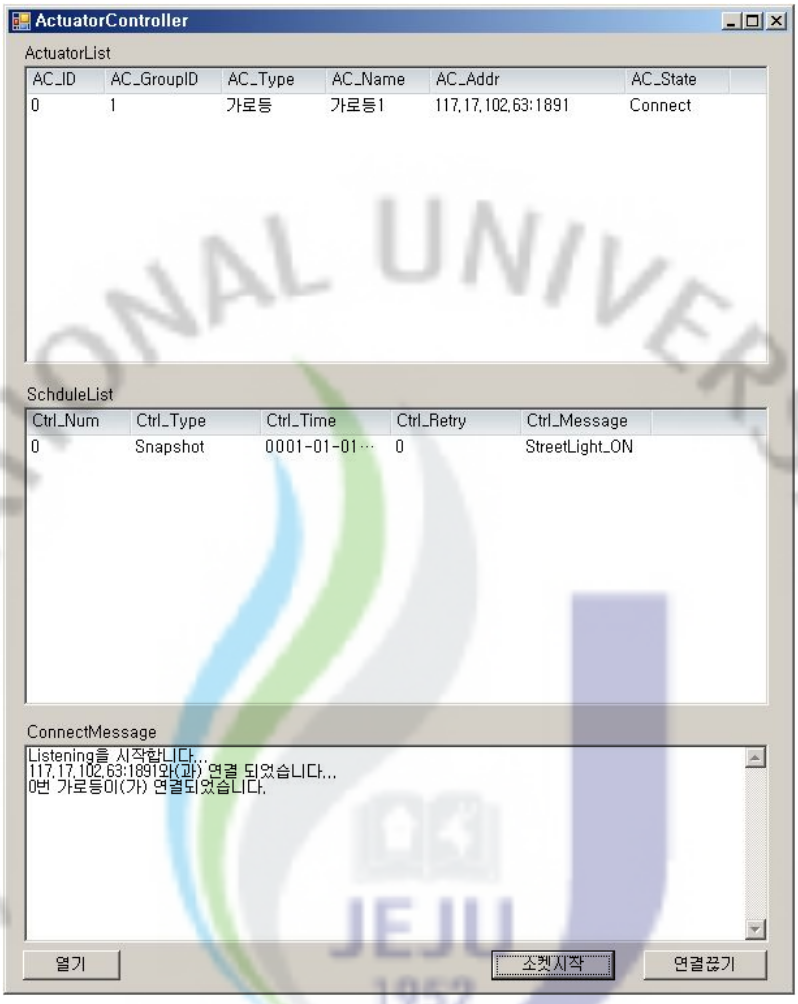

그림 28 구동체 제어 관리 응용 실행 화면

그림 28은 구동체 제어 관리 응용의 실행화면이다. 구동체 제어 관리 응용은 관리 응용에서 발생하여 통합 제어 Open API 인터페이스를 통해 전달된 제어명령을 연결된 가상 구동체에 전달하는 역할을 한다. 본 논문 3장과 4장에서 설명한 구동체 제어 관리 컴포넌트의 동작을 사용자가 확인할 수 있다. 응용의 상단의 구동체 리스트에는 연결된 가상 구동체의 목록을 나타내고, 중단의 스케줄 리스트에는 통합 제어 분석기를 통해 CtrlMessage 클래스로 변환된 제어 명령이 스케줄러에 추가된 모습을 나타낸다. 하단의 연결메시지 텍스트박스에는 가상 구

동체와 구동체 제어 관리 응용이 연결되었을 때, 연결된 가상 구동체의 원격 연결포인트 및 구동체의 간략한 정보를 나타낸다.

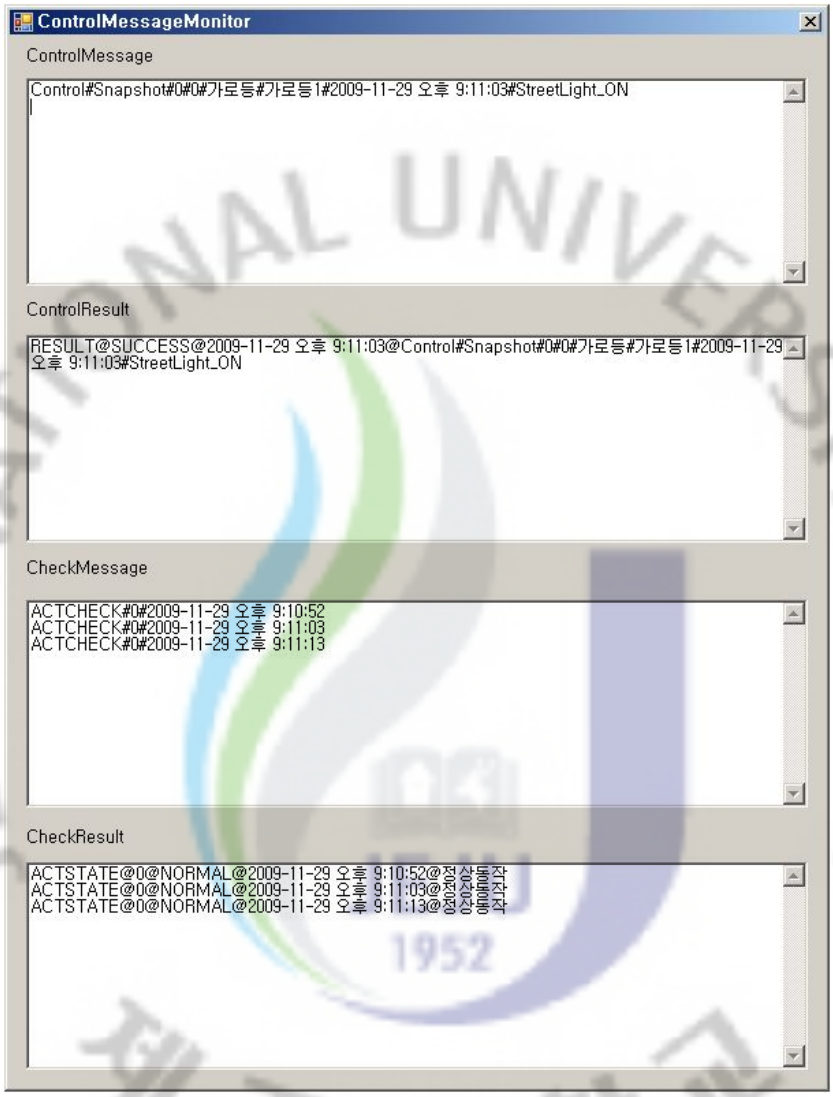

그림 29 제어 메시지 모니터링 응용 실행화면

그림 29는 제어 메시지 모니터링 응용이다. 제어 메시지 모니터링 응용은 구동체 제어 관리 응용과 가상 구동체 사이에 제어 메시지 및 결과 메시지를 표시하는 역할을 한다. 응용의 상단 제어 메시지 텍스트 박스는 가상 구동체로 전송하는 제어메시지의 내용을 표시한다. '#'을 구분자로 하여, 메시지 타입, 세부 메시지 타입, 제어 메시지 번호, 대상 구동체 번호, 구동체 타입, 구동체 이름, 제어

시간, 제어 내용의 순으로 정렬된 제어 메시지가 전송된다. 두 번째 텍스트 박스는 제어 결과 메시지가 표시된다. '@'를 구분자로 하여 메시지 타입, 세부 메시지 타입, 실제 제어 시간, 제어 메시지 내용을 정렬한 제어 결과 메시지를 수신하여 표시한다. 그리고 세 번째, 네 번째 텍스트 박스는 각각 연결된 구동체의 동작 상태를 주기적으로 확인하는 상태 확인 메시지와 응답 메시지를 표시한다. 여기서는 10초를 주기로 상태 확인을 하였다.

5. 가상 구동체 응용 구현

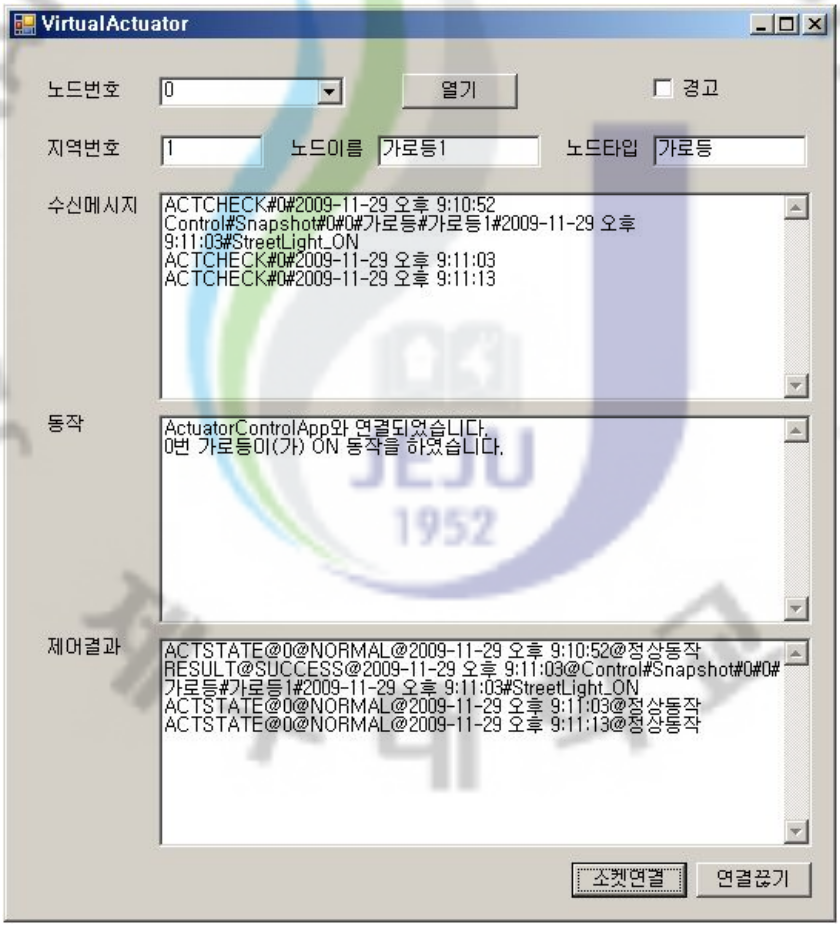

그림 30 가상 구동체 응용 실행화면

그림 30은 가상 구동체 응용 실행화면이다. 가상 구동체 응용은 실제의 구동체 대신해 구동체 제어 컴포넌트와 연결하여, 구동체 제어 Interface를 통해 구동체 제어 관리 컴포넌트로부터 제어 메시지를 수신하여 정상적인 제어 메시지 및 제어 결과 메시지가 발생하는지 확인할 수 있는 역할을 한다. 화면의 상단에는 가상의 0번 가로등에 대한 간략한 정보를 설정한다. 설정한 가상 노드는 실제 노드는 아니지만 해당 구동체로 향하는 제어 메시지를 수신하고 결과를 전송한다. 수신메시지 텍스트 박스는 구동체 제어 Interface를 거쳐 온 구동체 제어 메시지와 상태 확인 메시지의 내용을 표시하고, 화면 중간의 동작 텍스트 박스는 구동체 제어 컴포넌트와의 연결 상태 및 제어 메시지를 처리한 동작을 나타낸다. 하단의 제어 결과 텍스트 박스는 제어 메시지 처리가 끝난 이후에 생성한 제어 결과 메시지 및 상태 확인 메시지에 대한 응답인 상태 응답 메시지의 내용을 표시한다.

Ⅵ. 다양한 구동체 제어 명령의 실험 및 성능평가

본 장에서는 4장에서 설계한 다양한 구동체 제어 명령을 발생하여 5장에서 구현한 구동체 제어 미들웨어로 가상 구동체 제어를 테스트하고 처리시간을 측정하여 성능평가를 한다.

1. 통합 제어 Open API 인터페이스를 통한 구동체 제어 명령 실험

통합 제어 Open API 인터페이스를 통한 구동체 제어는 사용자가 서비스 응용에서 구동체를 오른쪽 마우스 클릭하여 4장에서 설계한 Snapshot, Continuous, Temporal, Spatial 제어 명령을 전송할 수 있다.

1) Snapshot 제어 명령 테스트

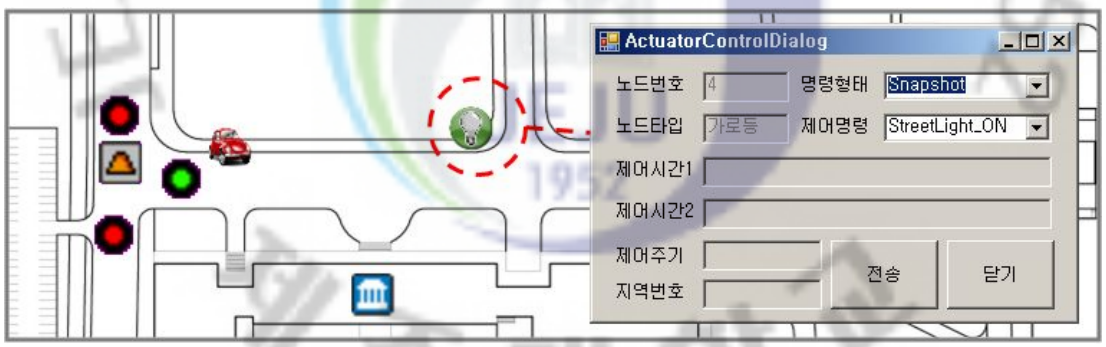

그림 31 Snapshot 제어 명령 발생

그림 31은 Snapshot 제어 명령을 발생하는 모습이다. 중앙의 가로등 노드를 우 클릭하면 우측의 다이얼로그가 나타나는데 명령형태를 Snapshot로 설정하고 전송버튼을 누르면 제어 명령이 발생하여 통합 제어 Open API 인터페이스로 전송된다.

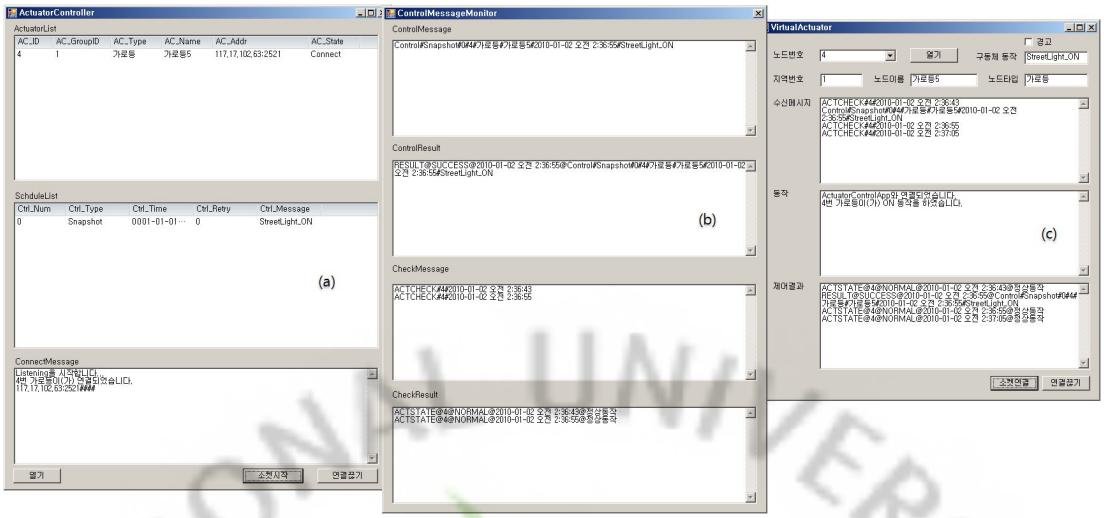

그림 32 Snapshot 제어 명령을 통한 가상 구동체 제어

그림 33 Snapshot 제어 명령에 따른 결과 확인

그림 32는 전송된 Snapshot 제어 명령을 통한 가상 구동체 제어를 나타낸다. (a) 구동체 제어 관리 응용에서 상단의 ActuatorList를 보면 4번 구동체가 연결이 되있고, SchduleList에는 Snapshot 제어 명령이 준비된 상태다. (b) 제어 메시지 모니터링 응용의 상단에 ContrlMessage를 보면 Snapshot 제어 명령으로 'StreetLight_ON' 제어메시지를 4번 노드인 가로등 5에 전송함을 알 수 있다. (c) 가상 구동체 응용에서는 제어 명령을 수신하고 가로등이 켜지고 제어결과를 반환한다. 그림 33은 Snapshot 제어 명령에 따른 결과로 제어한 가로등이 켜졌음을 서비스 응용에서 확인하는 것을 나타낸다.

2) Continuous 제어 명령 테스트

그림 34 Continuous 제어 명령 발생

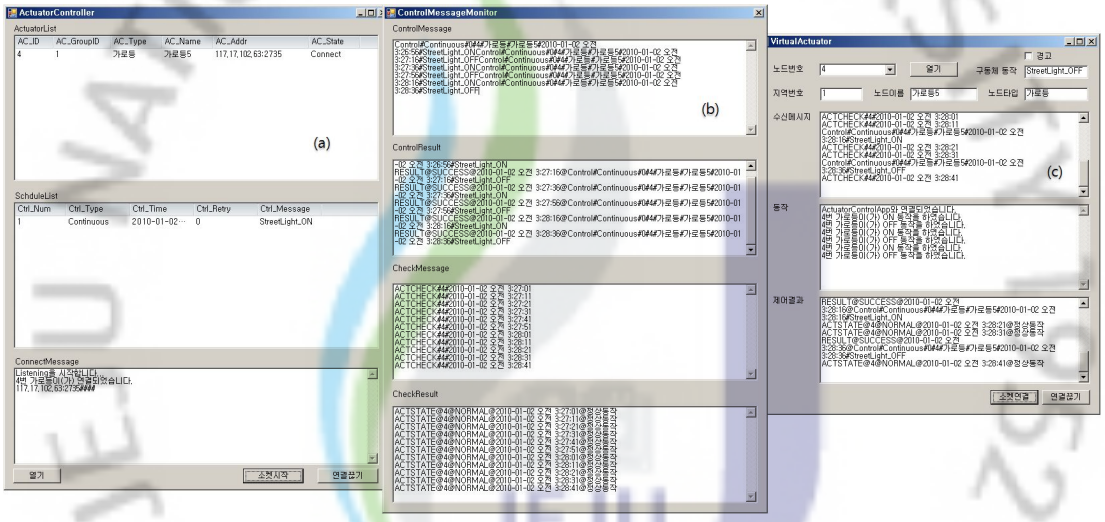

그림 35 Continuous 제어 명령을 통한 가상 구동체 제어

그림 34는 Continuous 제어 명령을 발생하는 모습이다. 중앙의 가로등 노드를 우 클릭하면 우측의 다이얼로그가 나타나는데 명령형태를 Continuous로 설정하고 전송버튼을 누르면 제어 명령이 발생하여 통합 제어 Open API 인터페이스로 전송된다. StreetLight_ON 제어 명령을 시작으로 제어시간1 부터 제어시간2 까지 20초의 주기로 가로등을 켜고 끄기를 반복한다. 그림 35는 전송된 Continuous 제어 명령을 통한 가상 구동체 제어를 나타낸다. (a) 구동체 제어 관리 응용에서 상단의 ActuatorList를 보면 4번 구동체가 연결이 되있고, SchduleList에는 Continuous 제어 명령이 준비된 상태다. (b) 제어 메시지 모니터링 응용의 상단

에 ContrlMessage를 보면 Continuous 제어 명령으로 'StreetLight_ON', 'StreetLight_OFF' 주기마다 반복하여 4번 노드인 가로등5에 전송함을 알 수 있다. (c) 가상 구동체 응용에서는 제어 명령을 수신하고 가로등을 제어할 때 마다 제어결과를 반환한다. 그림 36은 Continuous 제어 명령에 따른 결과로 제어한 가로등이 켜지고 꺼짐을 서비스 응용에서 확인하는 것을 나타낸다.

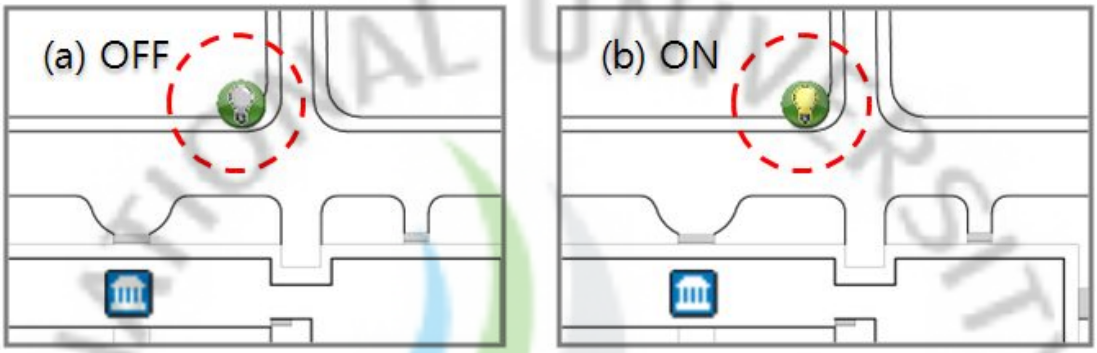

그림 36 Continuous 제어 명령에 따른 결과 확인

3) Temporal 제어 명령 테스트

그림 37은 Temporal 제어 명령을 발생하는 모습이다. 중앙의 가로등 노드를 우 클릭하면 우측의 다이얼로그가 나타나는데 명령 형태를 Temporal 설정하고 전송버튼을 누르면 제어 명령이 발생하여 통합 제어 Open API 인터페이스로 전송된다. 제어시간1과 제어시간2 두 번의 제어 명령을 가상 구동체로 전송한다.

그림 37 Temporal 제어 명령 발생

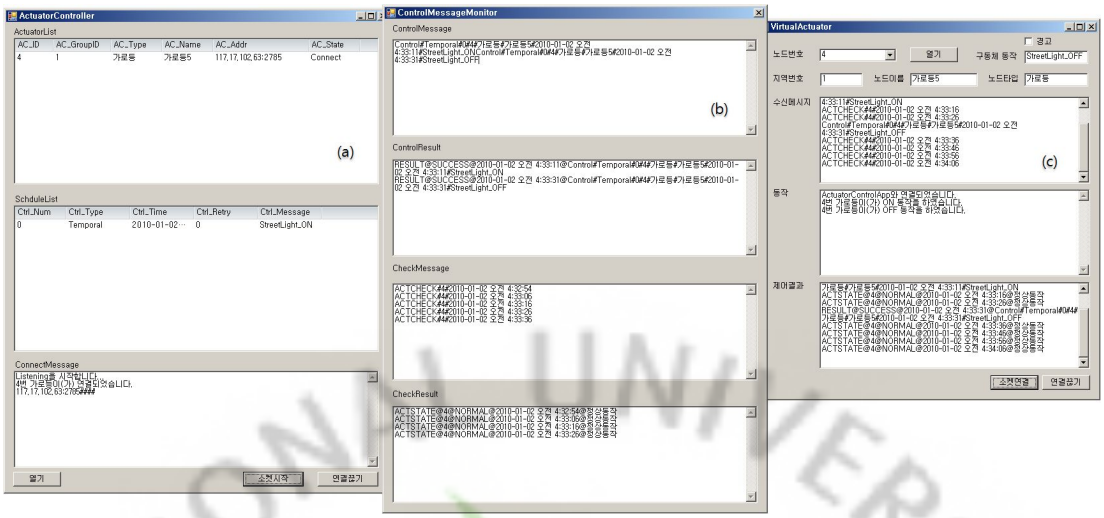

그림 38 Temporal 제어 명령을 통한 가상 구동체 제어

그림 38은 전송된 Temporal 제어 명령을 통한 가상 구동체 제어를 나타낸다. (a) 구동체 제어 관리 응용에서 상단의 ActuatorList를 보면 4번 구동체가 연결이 되있고, SchduleList에는 Temporal 제어 명령이 준비된 상태다. (b) 제어 메시지 모니터링 응용의 상단에 ContrlMessage를 보면 Temporal 제어 명령으로 'StreetLight_ON', 'StreetLight_OFF' 제어 메시지를 앞서 설정한 제어시간1, 제어시간2에 맞추어 4번 노드인 가로등5에 전송함을 알 수 있다. (c) 가상 구동체 응용에서는 제어 명령을 수신하고 가로등을 제어할 때 마다 제어결과를 반환한다.

4) Spatial 제어 명령 테스트

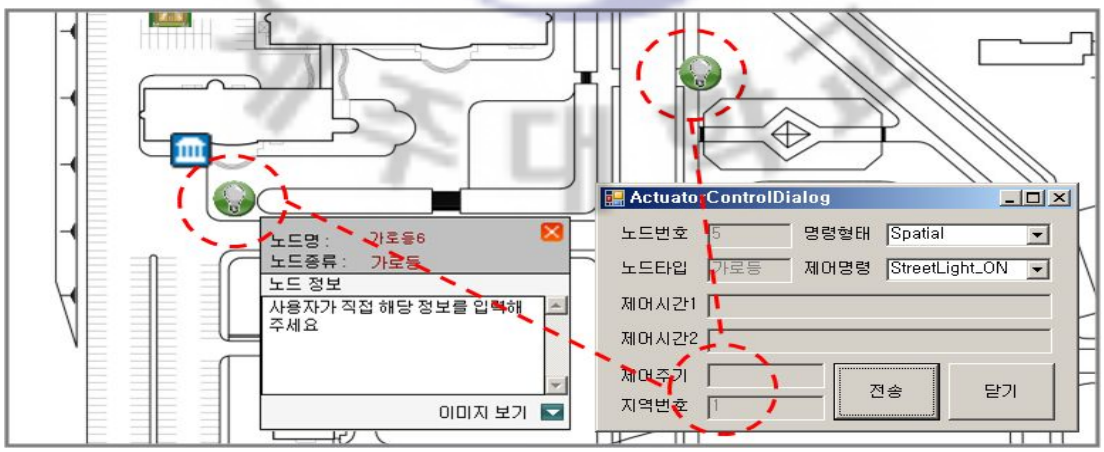

그림 39 Spatial 제어 명령 발생

그림 39는 Spatial 제어 명령을 발생하는 모습이다. 중앙의 가로등 노드를 우클릭하면 우측의 다이얼로그가 나타나는데 명령 형태를 Spatial 설정하고 전송버튼을 누르면 1번 지역의 구동체들을 대상으로 제어 명령이 발생하여 통합 제어 Open API 인터페이스로 전송된다. 그림 40은 전송된 Spatial 제어 명령을 통한 가상 구동체 제어를 나타낸다. (a) 구동체 제어 관리 응용에서 상단의 ActuatorList를 보면 5번, 6번 2개의 구동체가 연결이 되있고, SchduleList에는 Spatial 제어 명령이 준비된 상태다. (b) 제어 메시지 모니터링 응용의 상단에 ContrlMessage를 보면 Spatial 제어 명령으로 'StreetLight_ON' 제어메시지를 5번, 6번 2개의 가상 구동체에 전송함을 알 수 있다. (c), (d) 가상 구동체 응용에서는 제어 명령을 수신하고 가로등이 켜지고 제어결과를 반환하였다. 그림 41은 Spatial 제어 명령에 따른 결과로 제어한 가로등이 켜졌음을 서비스 응용에서 확인하는 것을 나타낸다.

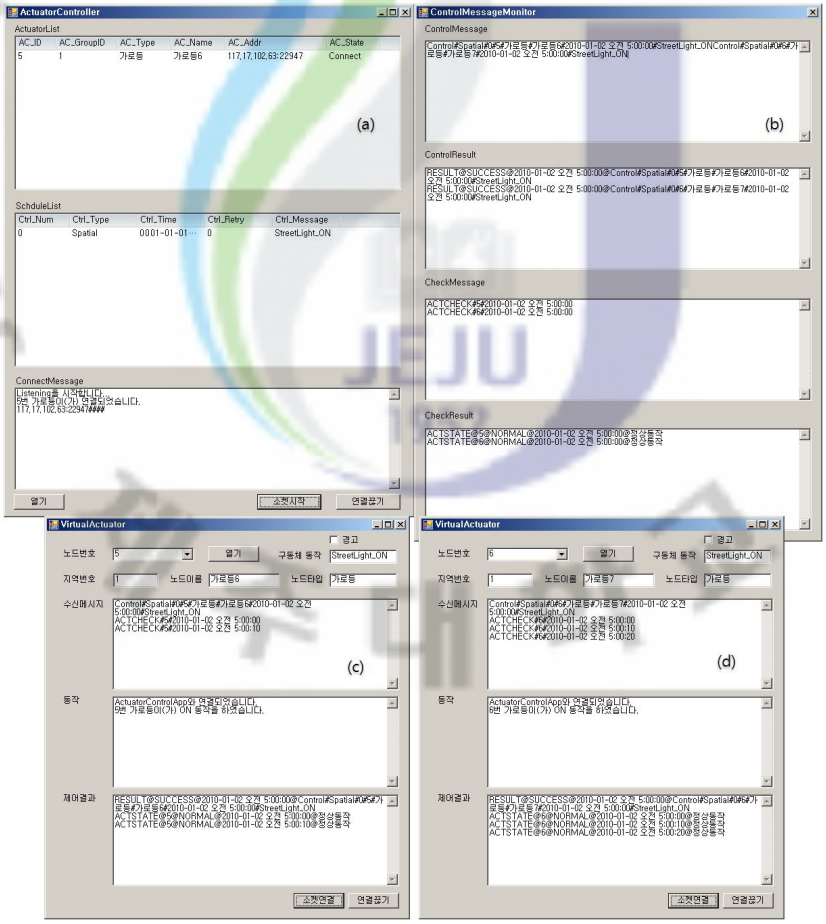

그림 40 Spatial 제어 명령을 통한 가상 구동체 제어

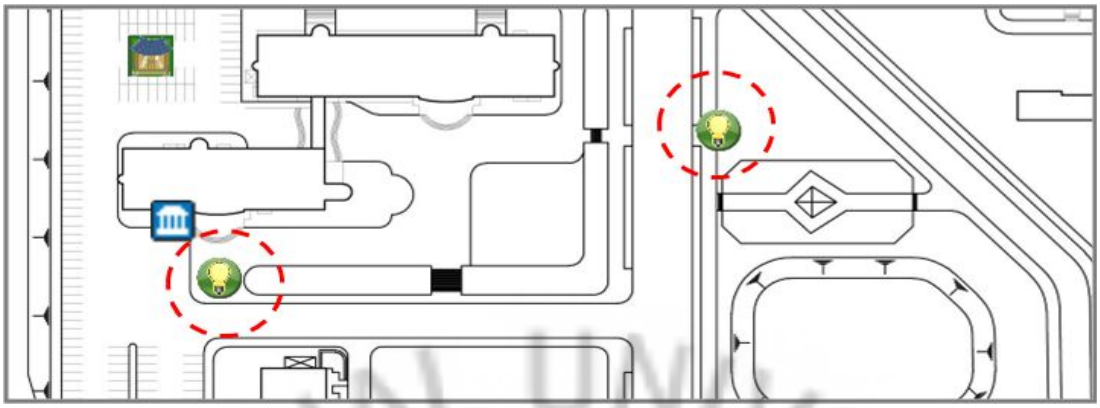

그림 41 Spatial 제어 명령에 따른 결과 확인

2. 컨텍스트 데이터 Interface를 통한 구동체 제어 명령 실험

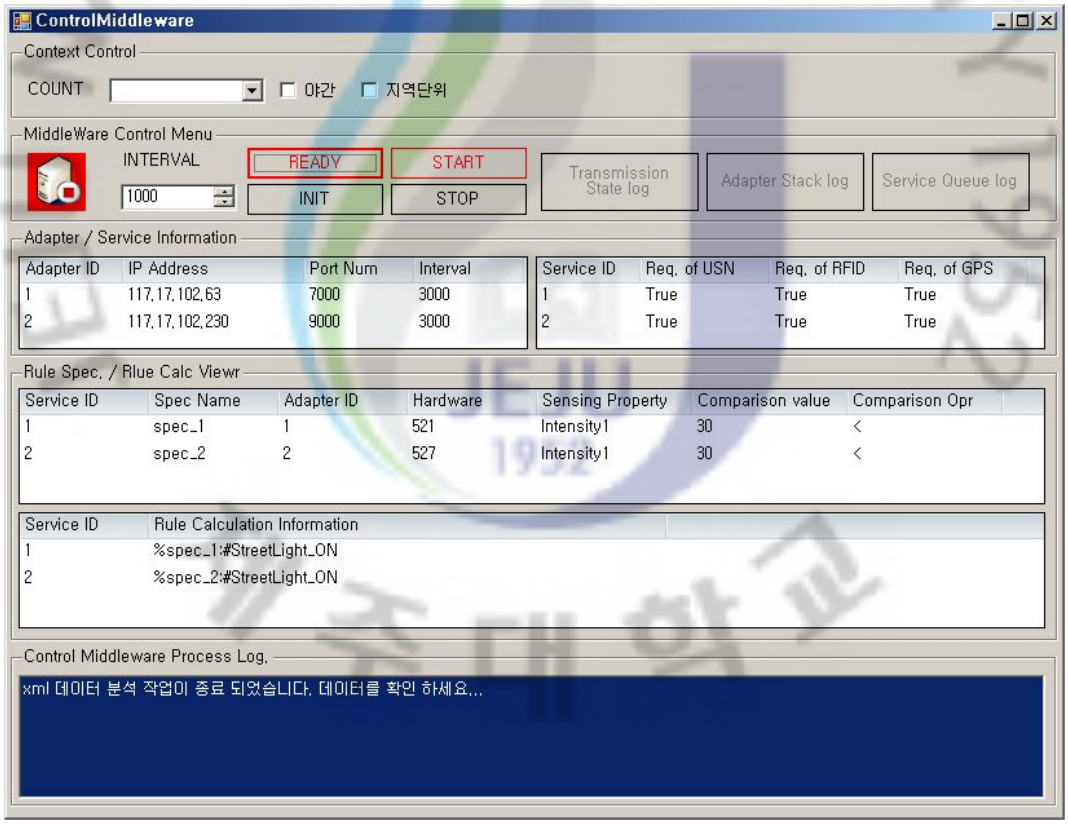

그림 42 데이터 처리 및 룰 스펙 연산 응용

컨텍스트 데이터 Interface를 통한 구동체 제어는 구동체 제어 미들웨어의 상

황인식 컴포넌트에서 전달되는 컨텍스트 데이터를 바탕으로 4장에서 설계한 ESnapshot, ESpatial 제어 명령을 전송한다. 그림 42는 데이터 처리 및 룰 스펙 연산 응용을 나타낸다. 화면 중단의 룰 스펙과 수집한 센싱데이터를 비교하여 제어 명령을 발생한다. 여기서 룰에 따르면 조도(Intensity1) 값이 30 미만이면 StreetLight_ON이라는 상황을 발생하여 컨텍스트 데이터를 구동체 제어 컴포넌트로 전달한다.

1) 이벤트 기반의 Snapshot 제어 명령 테스트

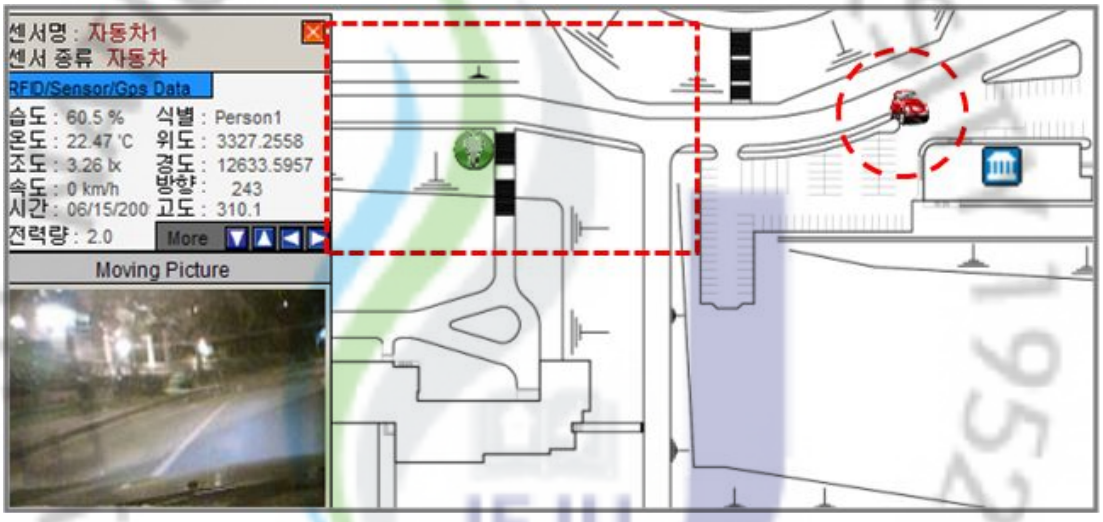

그림 43 이벤트 기반의 Snapshot 제어 명령 발생

그림 43은 이벤트 기반의 Snapshot 제어 명령을 발생하는 모습이다. 구동체 제어 미들웨어는 사람 또는 차량 등 센서, GPS를 탑재한 이동체가 수집한 값이 조도 값이 30 미만이고 중앙의 가로등 노드를 구동체로 등록 시 설정한 둘레의 사각형 영역과 GPS 위치 값이 일치할 경우, Snapshot 제어 명령을 가상 구동체로 전송한다. 그림 44는 이벤트 기반의 Snapshot 제어 명령을 통한 가상 구동체 제어를 나타낸다. (a) 구동체 제어 관리 응용에서 상단의 ActuatorList를 보면 0번 구동체가 연결이 되있고, SchduleList에는 Snapshot 제어 명령이 준비된 상태다. (b) 제어 메시지 모니터링 응용의 상단에 ContrlMessage를 보면 Snapshot

제어 명령으로 'StreetLight_ON' 제어메시지를 0번 노드인 가로등1에 전송함을 알 수 있다. (c) 가상 구동체 응용에서는 제어 명령을 수신하고 가로등이 켜지고 제어결과를 반환하였다. 그림 45는 이벤트 기반의 Snapshot 제어 명령에 따른 결과 확인을 나타낸다. 자동차가 가로등의 영역 안에 있고, 조도 값이 1.63lx를 나타내므로 30 미만으로 가로등이 켜졌음을 확인할 수 있다.

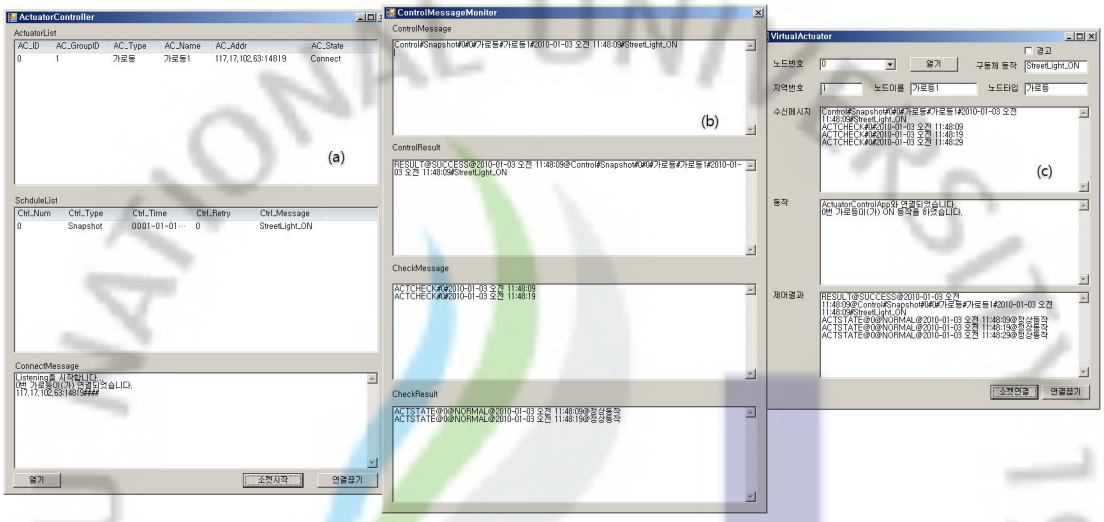

그림 44 이벤트 기반의 Snapshot 제어 명령을 통한 가상 구동체 제어

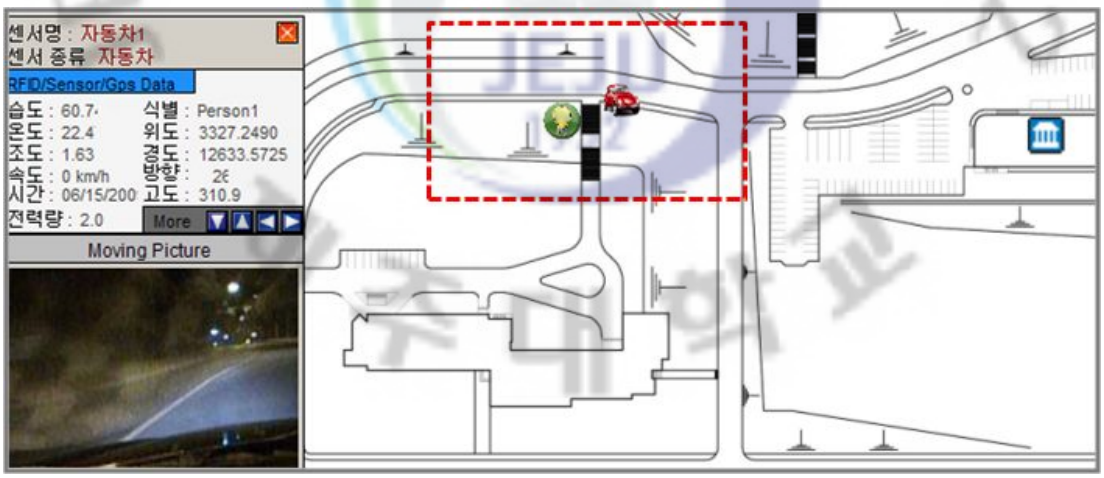

그림 45 이벤트 기반의 Snapshot 제어 명령에 따른 결과 확인

2) 이벤트 기반의 Spatial 제어 명령 테스트

그림 46 이벤트 기반의 Spatial 제어 명령을 통한 가상 구동체 제어

이벤트 기반의 Spatial 제어 명령은 그림 42 데이터 처리 및 룰 스펙 연산 응용에서 지역 단위가 체크 되어 있고 사람 또는 차량 등 센서, GPS를 탑재한 이동체가 수집한 값이 조도 값이 30 미만일 경우, GPS 위치 값을 확인하여 해당지역의 가상 구동체에 제어명령을 전송한다. 그림 46은 이벤트 기반의 Spatial 제어

명령을 통한 가상 구동체 제어를 나타낸다. (a) 구동체 제어 관리 응용에서 상단의 ActuatorList를 보면 0번, 1번 2개의 구동체가 연결이 되있고, SchduleList에는 Spatial 제어 명령이 준비된 상태다. (b) 제어 메시지 모니터링 응용의 상단에 ContrlMessage를 보면 Spatial 제어 명령으로 'StreetLight_ON' 제어메시지를 0번, 1번 2개의 가상 구동체에 전송함을 알 수 있다. (c), (d) 가상 구동체 응용에서는 제어 명령을 수신하고 가로등이 켜지고 제어결과를 반환한다. 그림 47은 이벤트 기반의 Spatial 제어 명령에 따른 결과 확인을 나타낸다. 센서에서 수집한 조도 값이 3.26lx를 나타내어 해당 지역의 가로등에 불이 켜졌다.

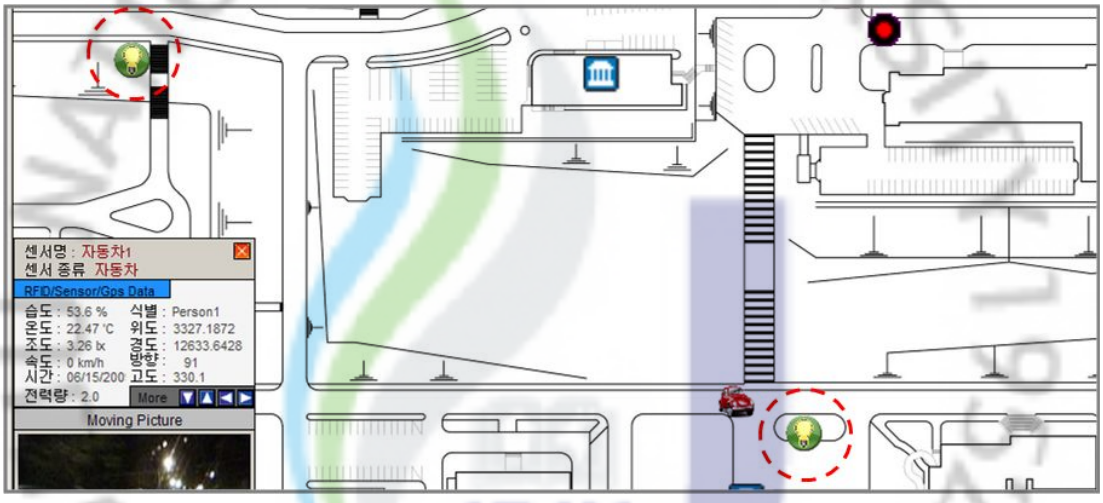

그림 47 이벤트 기반의 Spatial 제어 명령에 따른 결과 확인

3. 성능 분석

설계한 제어 명령 중 Snapshot, Continuous, Temporal 제어 명령 1회 1개의 구동체에 대한 명령의 처리시간은 차이가 없으며, Continuous, Temporal 제어 명령은 시간 및 횟수를 지정하기에 따라 천차만별이기 때문에 다수의 구동체를 동시에 제어하며, 트래픽이 예상되는 Spatial 제어 명령을 사용하였다. 로컬 네트워크 환경에서 서비스 응용을 통하여 연결된 가상 구동체의 개수를 달리하여 각각 100회의 제어 메시지를 전송하였다. 그림 48은 측정한 결과를 바탕으로 구동체 제어 명령의 처리 시간을 나타낸 그래프이다. 처음에 전송한 제어 명령이 빠

르게 처리된 이유는 처리 프로세스가 시작하는 시점이었기 때문에 빠르게 처리된 것으로 추정된다. 그 이후에는 약간의 기복은 있지만 거의 일정한 속도를 유지한다고 할 수 있다.

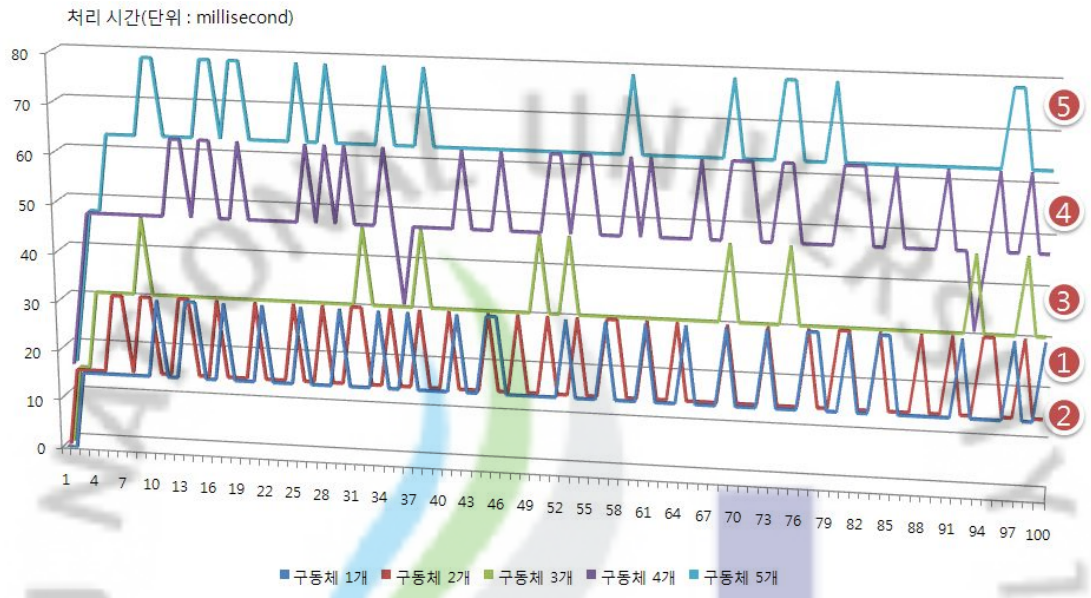

그림 48 구동체 제어 명령의 처리 시간

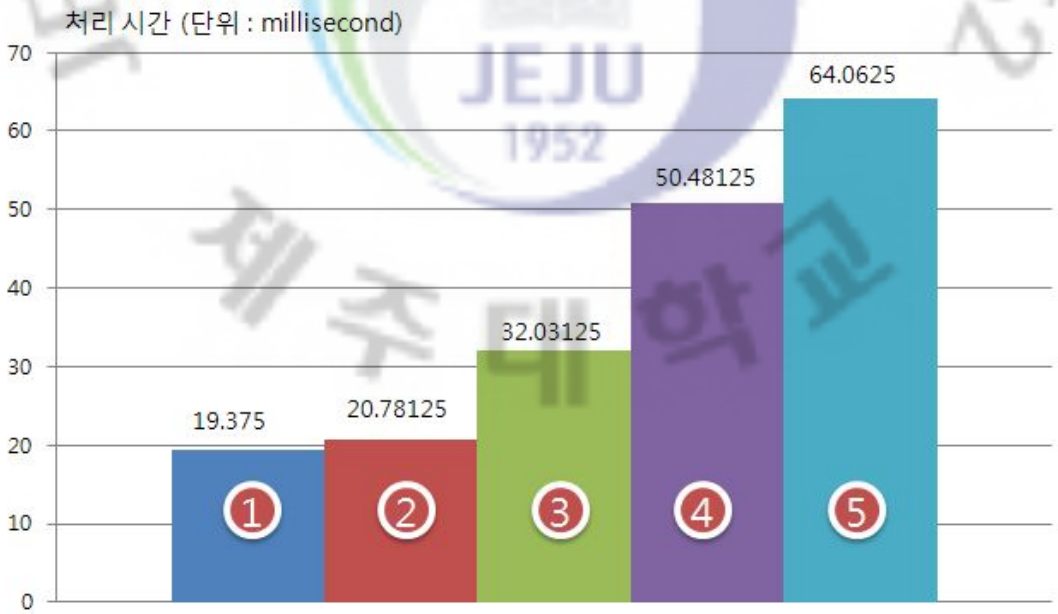

그림 49 구동체 제어 명령의 평균 처리 시간

그림 49는 앞의 그림 48 그래프에서의 구동체 제어 명령 처리 시간의 평균을 나타낸 그래프이다. 구동체 1개와 2개일 경우에는 비슷하지만, 2개 이상일 경우 제어 명령의 처리 시간이 일정하게 증가하였다. 이는 제어메시지를 생성하고 복수의 구동체에 제어 메시지를 전송하는데 걸리는 시간이 처리 시간을 좌우한다고 볼 수 있다. 구동체 5개를 사용했을 경우, 0.06초 정도의 처리 시간을 나타내었다. 그래프에서 구동체의 개당 처리 시간의 증가추세로 보면, 한 지역에 수십 개의 구동체에 제어 명령을 동시 전송을 한다 하더라도 1초 이내에 처리가 가능할 것으로 보인다. 따라서, 위와 같은 구동체 제어 미들웨어의 제어 명령 처리 시간에 따르면, 처리 속도가 매우 빠르다고 판단된다.

4. 평가 및 고찰

기존의 구동체 제어 미들웨어인 Sentire 미들웨어에서는 제어 명령의 형태에 관한 고려가 없으며, 구동체 제어의 구체적인 실례는 찾아볼 수가 없다. 이에 본 논문에서는 구동체 제어 미들웨어에서 사용하는 제어 명령의 다양한 형태를 정의하고 설계 및 구현하였다. 별도로, 구동체 정보 관리 응용을 구현하여 구동체 정보를 XML파일 형태로 저장하여 새로운 구동체를 등록하고 관리 할 수 있는 기능을 제공한다. 구동체 미들웨어, 가상 구동체, 서비스 응용을 이용하여 테스트 환경을 구축하고 처리 시간을 측정하여 성능 평가를 하였다. 이를 통해 여러 개의 구동체를 제어할 경우, TCP/IP 통신을 기반으로 한 네트워크 스트림을 이용한 통신에서 하나 또는 두 개의 연결일 경우 비슷한 처리 시간을 나타내었지만, 그 이상일 경우에는 추가되는 개수에 비례해서 처리 시간이 늘어났다. 그리고 스레드가 동작하는 과정에서 각 개수별로 거의 일정한 처리 시간을 유지하였다. 실제 구동체를 제어하기 위해서는 구동체의 특성에 따라 통신 방법이 달라져야 하는데, 차후의 연구에서는 통신방법에 따른 성능을 고려해야 할 것이다. 현재 구동체 제어 미들웨어는 룰 기반으로 조건 값에 따라 제어가 되지만, 앞으로는 인공지능과 같은 복잡한 상황인식을 통하여 실제 생활에 적용될 수 있는 유용한 형태로 발전해야 할 것이다.

Ⅶ. 결 론

앞으로의 유비쿼터스 관련 시스템은 도처에 설치된 센서와 다양한 하드웨어 장치들이 공존할 것으로 예상된다. 공존 한다는 것은 서로 연결되어 상호작용을 하기도 하고 독자적으로 사용자의 제어 명령에 의해서 동작하는 것도 가능할 것이다. 구동체 제어 미들웨어에서 이러한 장치들의 상호작용과 동시에 사용자측에서의 제어 명령을 처리할 수 있어야 한다. 이에 본 논문에서는 이러한 개념이 혼재하는 유비쿼터스 인프라 환경을 생각하여 하드웨어를 제어를 요구하는 여러 가지 상황을 정의 하였다. 그리고 기존에 SANET 시스템을 기반으로 한 Sentire 미들웨어와 관련된 연구를 바탕으로 다양한 구동체 제어명령을 처리하는 구동체 제어 관리 컴포넌트와 수집한 센싱데이터를 이용해 룰 기반 상황인지를 하여 컨텍스트 데이터를 전달하고 구동체 제어 명령을 하는 상황인지 컴포넌트를 설계하였다. 이에 따라 구동체 제어 관리 컴포넌트와 가상 구동체가 정의한 구동체 제어의 형태로 동작하도록 구현하였다. 다양한 구동체의 형태에 관한 고려가 부족하지만 유비쿼터스 인프라 환경에서 센서 및 사용자와 구동체의 통신에 관한 몇 가지 방법을 제안하였다고 사료된다. 향후 구동체 제어 미들웨어의 연구가 진척되어 유비쿼터스 서비스의 표준으로 자리 잡는 날이 올 것을 기대한다.

참고문헌

[1] Mandar Chitre, “DSAAV - A DISTRIBUTED SOFTWARE ARCHITECTURE FOR AUTONOMOUS VEHICLES”, AUV3 Undersea Vehicles: Enabling Technologies 3 on Oceans, Poles and Climate : Technological Challenges, 2008

[2] Sebastian Schuster and Uwe Brinkschulte, “Model-Driven Development of Ubiquitous Applications for Sensor-Actuator-Networks with Abstract State Machines”, Session 7: Environment Interaction on SEUS 2007, Santorini Island, Greece, May, 2007

[3] Kasper Hallenborg, “Contextual Interfacing: A Sensor and Actuator Framework”, Embedded and ubiquitous computing, International conference, Nagasaki, Japan, December 6-9, 2005

[4] Joel W. Branch, John S. Davis II, Daby M. Sow, Chatschik Bisdikian, “A Framework for Building Middleware for Sensor and Actuator Networks”, Proceedings of the 3rd Int'l Conf. on Pervasive Computing and Communications Workshops, 2005

[5] E. Niemelä and T. Vaskivuo, “"Agile Middleware of Pervasive Computing Environments"", Proc. 2nd IEEE Annual Conf. on Pervasive Comp. and Comm. Workshops, Orlando, Florida, Mar. 2004, p. 192.

[6] Joel W. Branch, Boleslaw Szymanski, Chatschik Bisdikian, Norman Cohen, John S. Davis, Maria R. Ebling, and Daby M. Sow, “"Towards Middleware Components for Distributed Actuator Coordination"", Proc. Third Workshop on Embedded Networked Sensors(EmNets 2006), Bostor, MA, May, 2006

[7] 김대영, 김재언, 성종우, 이강우, “센서 네트워크 운영체제/미들웨어 기술동향,” IITA 주간기술동향 통권 1221 호, 200,11

[8] 김민수, 김광수, 이용준 "USN 미들웨어의 특징 및 기술개발 동향", 주간기술동향 통권 1284호 2007, 2, 21

[9] 김부림, "센서 네트워크 기반의 Pull/Push 서비스를 위한 개방형 응용 인터페이스

설계 및 구현", 석사 졸업 논문, 2007. 12, p 18-20

[10] 김용우, "USN 기반의 구동체 제어 미들웨어 설계 및 구현", 석사 졸업 논문, 2008. 12, p. 1-25

[11] Tim Wark, Chris Crossman, Wen Hu, Wing Guo, Philip Valencia, Pavan Sikka, Peter Corke, Caroline Lee, John Henshall, Kishore Prayaga, Julian O'Grady, Matt Reed, Andrew Fisher "The Design and Evaluation of a Mobile Sensor/Actuator Network for Autonomous Animal Control", Proceedings of the 6th international conference on Information processing in sensor networks, 2007, p 206-215