The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.8, pp.356~366

eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.8.356

Multi-Agent Monitoring System for Intelligent Service Robots

Haneol Cho† · Insik Yu†† · Jaeho Lee†††

ABSTRACT

Users of intelligent robots require access to the status data of the robots for various reasons. The status data of intelligent robots can be generated by combining the status data of the functional agents that constitute the intelligent robot. However, existing intelligent robot systems do not generate the necessary agent status data for creating the status data of intelligent service robots, or they generate it in different ways, making it impossible to collect this information in a uniform manner. Furthermore, these systems have limitations such as collecting the same information redundantly if multiple users request it and only using a single method of communication to deliver robot information, thereby failing to offer the communication methods desired by users. This paper proposes a multi-agent monitoring system for intelligent service robots designed to overcome these limitations. This monitoring system generates status data in response to the actions performed by functional agents, thereby allowing for the unified generation and collection of agent status data. Additionally, the monitoring system resolves data redundancy issues by collecting the necessary data just once, in accordance with user monitoring demands, and delivers status data through a proxy that supports the preferred communication methods of users, thereby providing compatibility with various communication methods. Through experiments, we have verified that this monitoring system can deliver the status data of intelligent robots to multiple users using various communication methods.

Keywords : Intelligent Robot, Monitoring, Service Robot, Status Data

지능형 서비스 로봇을 위한 멀티 에이전트 모니터링 시스템

조 한 얼† · 유 인 식†† · 이 재 호†††

요 약

지능형 로봇의 사용자는 다양한 이유로 로봇의 상태 정보가 필요하다. 이러한 지능형 로봇의 상태 정보는 지능형 로봇을 구성하는 기능 에이전트의 상태 정보를 조합하여 생성할 수 있다. 하지만 기존 지능형 로봇 시스템은 1) 지능형 서비스 로봇의 상태 정보 생성을 위해 필요한 에이전트 상태 정보를 생성하지 않거나, 각기 다른 방법으로 생성하고 있어 이를 일괄적으로 수집할 수 없다. 또한 2) 복수의 사용자가 같은 정보를 요청하면 이를 여러 번 수집하여 데이터 중복성 문제가 발생하였고, 3) 한 가지 통신 방법만을 지원하여 사용자가 원하는 통신 방법을 제공하지 못한다는 한계점을 가지고 있다. 본 논문에서는 이러한 한계점을 해결하기 위한 지능형 서비스 로봇을 위한 멀티 에이전트 모니터링 시스템을 제안한다. 이 모니터링 시스템은 1) 기능 에이전트의 행위 수행에 따라 상태 정보를 생성하게 함으로써, 에이전트 상태 정보를 단일화된 방법으로 생성 및 수집할 수 있다. 또한 2) 모니터링 시스템이 사용자의 모니터링 수요에 따라 필요한 데이터를 한 번씩만 수집하여 데이터 중복성 문제를 해결하며, 3) 사용자가 원하는 통신 방법을 지원하는 프록시를 통해 상태 정보를 전달하여 통신 방법에 대한 호환성을 제공한다. 우리는 실험을 통해 이 모니터링 시스템이 다양한 통신 방법을 활용하여 여러 사용자에게 지능형 로봇의 상태 정보를 전달하는 것을 확인할 수 있었다.

키워드 : 지능형 로봇, 모니터링, 서비스 로봇, 상태 정보

1. 서 론

지능형 서비스 로봇은 인식, 추론 및 판단, 행위 등 다양한 지능적 요소를 연계하여 사용자가 필요로 하는 기능을 제공할 수 있는 로봇이라고 말할 수 있다[1]. 이러한 로봇은 적용되는 환경에 따라 사용자가 필요로 하는 서비스의 종류가 다양하므로, 처음부터 사용자가 원하는 모든 서비스를 제공할 수 없다. 그러므로 지능형 서비스 로봇은 사용자의 필요에 따라 개발되어야 한다.

이러한 지능형 서비스 로봇을 개발하기 위해선 인식, 추론 및 판단. 행위 등 각 지능적 요소를 개발하고, 이를 효과적으로 통합할 수 있어야 한다. 하지만 각각의 요소를 매번 개발할 경우, 서비스 환경에 의존적인 기능과 범용적으로 활용될 수

※ 이 논문은 2022년도 서울시립대학교 교내 학술연구비에 의하여 지원되었음.
† 비 회 원 : 서울시립대학교 정보기술연구소 연구원
†† 준 회 원 : 서울시립대학교 전자전기컴퓨터공학과 박사과정
††† 종신회원 : 서울시립대학교 전자전기컴퓨터공학부 교수
Manuscript Received : April 1, 2024
First Revision : June 25, 2024
Second Revision : August 6, 2024
Accepted : August 8, 2024
* Corresponding Author : Jaeho Lee(jaeho@uos.ac.kr)

있는 기능이 혼재되어 개발할 때마다 중복되는 영역이 존재한다. 이러한 중복 개발 문제를 해결하고 지능형 서비스 로봇의 개발을 지원하기 위해서는 서비스에 대해 공통적인 기능 및 지식을 분리하여 제공하고, 개발자에게 서비스 도메인에 특화된 기능 및 지식에 집중하여 개발할 수 있는 환경을 제공할 필요가 있다.[2] 국내에서는 이러한 지능형 로봇 서비스 개발을 위한 소프트웨어 프레임워크로 지식기반 로봇 지능체계를 연구한 바 있다.

지식기반 로봇 지능체계는 지능형 서비스 로봇에게 필요한 지능적 요소들의 라이프사이클을 보장하기 위해, 멀티 에이전트 시스템을 적용하여 각 요소를 에이전트로 개발하고 이를 통합할 수 있도록 했다. 또한 서비스 환경에 의존적인 기능을 서비스 패키지로 분리하여, 로봇이 적용되는 환경이 바뀌더라도 전체 프로그램을 새롭게 작성하는 것이 아닌, 새로운 모델의 개발을 통해 해당 도메인에 로봇을 적용할 수 있도록 했다.

이러한 서비스 패키지의 개발 과정에는 지능형 로봇이 정상적으로 동작하는지 확인하는 디버깅 과정이 필수적이며, 이를 위해서는 현재 로봇이 수행 중인 작업이나 로봇이 인지하고 있는 환경 정보와 같은 지능형 로봇의 상태 정보가 필요하다.

이러한 지능형 로봇의 상태 정보는 서비스 개발자뿐만 아니라 다른 로봇 사용자들도 필요하다. 대표적으로 로봇 관리자는 로봇의 상태를 주기적으로 확인하고 정상적인 서비스를 수행하고 있지 않을 경우, 로봇의 서비스 수행을 중지하고 정비하는 등에 활용할 수 있다. 또한 로봇 서비스 이용자는 서비스 진행 중 로봇의 남은 전력이나 현재 상태, 혹은 서비스 진행 상황 등을 확인하며 이후의 로봇이 수행할 작업을 결정하는 데 영향을 줄 수 있으며, 로봇 사용이 끝나면 더 이상 로봇의 데이터가 필요하지 않을 수 있다.

이와 같이, 지능형 서비스 로봇은 사용자에게 현재 로봇의 상태나 서비스 수행 상황 등과 같은 정보를 전달할 수 있어야 한다. 하지만 지능체계는 지능형 서비스 로봇의 상태 정보를 사용자에게 전달하기에 다음과 같은 한계점을 가지고 있다.

1) 지능형 서비스 로봇의 상태 정보 생성을 위해 여러 에이전트의 상태 정보가 필요하지만, 각 에이전트의 상태 정보가 서로 다른 방법으로 생성하거나 생성하고 있지 않다.

사용자가 필요로 하는 지능형 로봇의 상태 정보를 생성하기 위해서는 각 지능적 요소를 담당하는 에이전트의 상태를 조합할 필요가 있다. 하지만 지능체계 내 지능적 요소를 구현하는 에이전트들은 서로 다른 라이프사이클을 가지고 있으며, 상태 정보를 생성하지 않거나, 자신만의 방법으로 에이전트 상태 정보를 생산하고 있다. 이로 인해, 지능형 서비스 로봇의 상태 정보를 생성하는 데 필요한 에이전트 상태 정보를 일괄적으로 수집하고 활용하기 어렵다는 한계를 가지고 있다.

2) 복수의 사용자가 동일 정보를 요청하면 이를 중복하여 수집하게 된다.

지능형 서비스 로봇은 동시에 여러 사용자가 존재할 수 있으며, 각 사용자는 자신의 목적에 따라 로봇에게 상태 정보를 요청할 수 있다. 이때, 복수의 사용자가 로봇에게 동일한 정보를 요청할 경우, 해당 상태 정보 생성에 필요한 에이전트 상태 정보를 중복으로 수집하게 된다. 이러한 중복 수집이 반복적으로 발생할 경우, 에이전트 성능에 악영향을 줄 수 있다.

3) 지능형 서비스 로봇으로부터 상태 정보를 전달받는 로봇 사용자들은 각자 최적화된 통신 방법이 다를 수 있어, 한 가지 통신 방법만 제공하는 경우 호환성이 떨어진다.

위 세 가지 한계점을 해결하기 위하여 본 논문에서는 지능형 서비스 로봇을 위한 모니터링 시스템을 제안한다. 이 시스템은 1) 행위 기반 상태 정보 생성 방법을 제공하여 여러 에이전트의 상태 정보를 단일화된 방법으로 수집할 수 있으며, 2) 모니터링 수요 기반 에이전트 상태 정보 수집을 통해 데이터의 중복성을 제거하며, 3) 프록시를 통해 사용자가 원하는 통신 방법을 활용하여 지능형 로봇 상태 정보를 전달한다.

2. 관련 연구

2.1 개요

본 논문은 멀티 에이전트 시스템을 적용하여 개발된 지능형 서비스 로봇을 위한 멀티 에이전트 모니터링 시스템을 제시하여, 지능형 서비스 로봇 사용자에게 지능형 로봇의 다양한 상태 정보를 전달하는 것을 목표로 한다.

이와 관련하여 본 논문에서는 멀티 에이전트 시스템에서의 모니터링 시스템과 에이전트 기반의 지능형 로봇 아키텍처를 분석 및 비교하고자 한다.

2.2 Monitoring and Organizational level Adaptation of Multi-Agent Systems

Guessoum[3]은 확장 가능한 큰 규모의 멀티 에이전트 시스템에서의 문제 검출 및 복구를 위한 모니터링 시스템을 제안했다. 해당 시스템은 이미 구현되어 있는 멀티 에이전트 시스템을 보조하며, 멀티 에이전트 시스템에서 송수신하는 메시지를 분석하여 문제를 검출한다.

해당 모니터링 시스템은 에이전트 모니터를 통해 각 에이전트가 송수신하는 메시지를 수집한다. 수집된 메시지는 타입, 트래픽, 메시지 전달 순서 등의 정보를 포함하여 호스트 모니터로 전송된다. 만약 멀티 에이전트 시스템에 무언가 문제가 생기게 되면, 메시지 유동량이나 전달 순서 등에 바로 문제가 나타나게 되고 이를 호스트 모니터에서 파악하여 문제가 생긴 에이전트를 바로 발견할 수 있다.

위의 시스템은 추가로 이렇게 수집된 정보들을 통하여 에이전트의 중요도를 측정하고 이 중요도를 기반으로 중요한 에이전트들의 복제를 만들어 만약 해당 에이전트들이 동작을 멈

췄을 경우 대신 행동하도록 할 수 있는 기능도 가지고 있다.

이 모니터링 시스템은 확장 가능한 큰 규모의 멀티 에이전트 시스템 내에서의 문제점 발견에 특화되어 있다. 하지만 지능형 로봇의 모니터링은 사용자들에게 문제점뿐만 아니라 로봇의 다른 상태 정보 또한 전달할 수 있어야 한다. 하지만 해당 모니터링 시스템은 기능이 문제점 발견에 특화되어 있어 일반적인 상태 정보를 전달하는 기능을 적용하기에 적절하지 않다.

2.3 Drums: a Middleware-Aware Distributed Robot Monitoring System

Monajjemi 등[4]은 로봇 미들웨어 시스템을 보완하기 위한 분산 로봇 모니터링 도구인 Drums를 소개했다. Drums는 ROS와 같은 미들웨어 시스템의 추상화된 하부 리소스를 실시간으로 모니터링하여 시스템 수준의 상호작용을 명확하게 드러내며, 로봇 시스템의 품질 모니터링 및 자율 시스템의 상태 분석 도구로 사용될 수 있다. 이 시스템은 미들웨어가 숨기는 네이티브 프로세스와 네트워크 채널을 분석 및 분류하여 로봇 시스템의 상태를 사용자에게 제공한다. Drums는 시스템 수준의 리소스 상태를 실시간으로 모니터링하고, 네트워크를 통해 중앙 데이터베이스에 데이터를 집계한다. 또한, Drums는 이상 탐지를 통해 네트워크 트래픽, CPU 부하, 메모리 사용량 등 시스템 리소스에 대한 문제를 감지할 수 있다.

이와 같이, Drums는 로봇 미들웨어 시스템과 통합되어 분산된 로봇 시스템의 리소스를 모니터링할 수 있는 기능을 제공한다. 하지만, Drums는 주로 시스템 리소스 상태와 네트워크 트래픽과 같은 저수준의 리소스 모니터링에 중점을 두고 있어 고수준의 에이전트 상황 정보와 같은 사용자가 필요로 하는 정보 제공에 최적화되어 있지 않다. 반면, 우리의 시스템은 로봇의 상태 정보를 통합된 방법으로 수집하고, 중복 수집 문제를 해결하며, 다양한 통신 방법을 통해 사용자에게 정보를 전달할 수 있도록 설계되었다.

2.4 An Agent-Based Cognitive Robot Architecture

Changyun Wei[5]는 인지 로봇에 필요한 지식 표현, 추론, 로봇 행위 제어와 같은 능력들을 통합하여 제공하는 것을 목표로 하는 에이전트 기반 인지 로봇 아키텍처를 연구했다.

이 아키텍처는 크게 작업추론과 같은 고수준 추론 기능을 담당하는 인지 레이어, 로봇을 제어하기 위한 로봇 행위 레이어 및 레이어 간 인터페이스로 구성되어 있다. 인지 레이어는 인지 에이전트 개발을 위한 GOAL[6]을 사용하여 추론, 의사결정, 행위 선택과 같은 기능을 제공하고 있다. 로봇 행위 레이어는 로봇 센서로부터 입력된 정보를 처리하여 지식의 형태로 가공하거나, 인지 레이어로부터 전달받은 로봇 행위를 수행하는 역할을 수행한다. 레이어 간 인터페이스는 인지 레이어와 행위 레이어 간 통신 지원 및 환경 관리 등을 수행한다.

에이전트 기반 인지 아키텍처에서는 로봇 센서 및 액추에이터 등을 제어하는 로봇 프로그램 제어를 위한 디버그 모니터를 제공한다. 이는 로봇이 센서를 통해 관측한 환경 정보, 환경 정보를 지식 정보로 과정 등을 모니터링할 수 있으며, 필요한 경우 로봇 액추에이터를 직접 제어하는 등의 기능을 제공하고 있다.

해당 연구에서는 로봇 행위 레이어에 대한 모니터링 기능을 제공하고 있지만, 인지 레이어에 대한 모니터링은 제공하지 않는다. 이로 인해, 로봇 하드웨어의 상태는 알 수 있으나 추론이나 의사결정 과정과 같이 인지 레이어 내부에서 발생하는 일들에 대해 알기 어렵다는 한계점을 가진다. 지능형 로봇의 상태 정보를 생산하기 위해서는 시스템을 구성하는 모든 에이전트의 상태 정보가 필요하므로, 해당 연구에서 사용하고 있는 것과 같이 로봇 행위 레이어만 모니터링을 하는 것은 적절하지 않다.

3. 배경 연구

3.1 지식기반 로봇 지능체계

지능형 서비스 로봇은 실시간으로 주변 상황을 인지하고 사용자의 요구를 만족하기에 적합한 작업을 선택하고 이를 수행할 수 있어야 한다. 더 나아가 사용자에게 서비스를 제공중인 상황에서 새로운 서비스 요청이 발생하여도 이에 대응하여 동시에 복수의 서비스를 수행할 수 있어야 한다. 국내에서는 이러한 요구사항을 만족하기 위해 2016년부터 2020년까지 지능형 서비스 로봇 개발을 위한 소프트웨어 프레임워크로 지식기반 로봇 지능체계를 연구했다. Fig. 1는 지능체계의 아키텍처를 나타낸 것이다.

지능체계는 지능을 구성하기 위해 필요한 상황 정보 관리 기능[7], 지식 정보 관리 기능[8], 학습 관리 기능, 작업 관리 및 추론[9], 외부 서비스용 인터페이스[10-12] 등의 기능을 지원한다.

지능체계의 가장 큰 특징은 로봇이 서비스를 수행하는 데 필요한 지식을 서비스 패키지라는 별도의 모델로 분리했다는 것이다. 서비스 패키지는 서비스에 필요한 도메인 지식, 작업

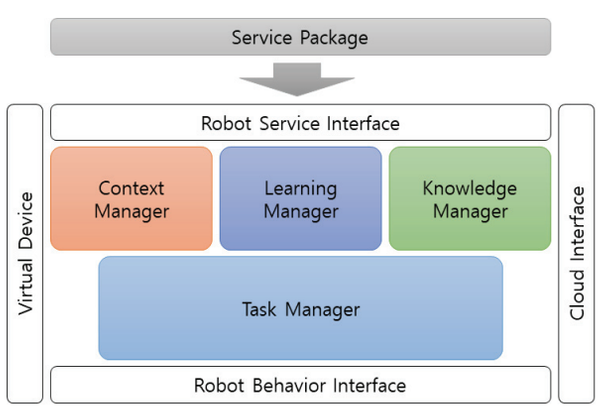

Fig. 1. Architecture of Intelligent Service Robot Framework

계획, 및 추론 규칙 등으로 구성되어 있다. 이를 통해 지능형 서비스 로봇 개발자는 새로운 서비스를 개발할 경우, 전체 기능을 다시 개발하는 것이 아니라, 해당 도메인에서 필요한 지식의 정의를 통해 새로운 서비스를 개발할 수 있다.

지능체계는 멀티 에이전트 시스템을 적용하여 각 지능 요소를 담당하는 여러 에이전트로 구성되어 있다. 이러한 특징으로 인해, 지능형 서비스 로봇의 상태 정보를 생산하기 위해서는 프레임워크 내 각 에이전트의 상태 정보를 수집 및 조합할 필요가 있다. 하지만 에이전트의 라이프사이클이나 동작 방식 등이 서로 달라 에이전트 상태 정보를 수집하기 위해서는 에이전트 별로 다른 방식을 사용해야 하며, 이는 매우 비효율적이다. 따라서 지능형 서비스 로봇의 상태 정보 생산을 위해서 지능형 로봇 내 에이전트들의 상태 정보를 일괄적으로 수집할 방법이 필요하다.

4. 연구 내용

기존 지능체계는 1) 지능형 서비스 로봇의 상태 정보를 생산하기 위해 필요한 에이전트 상태 정보를 일관되게 수집할 방법이 존재하지 않았으며, 2) 복수의 사용자가 지능형 로봇 상태 정보를 요청하면 동일한 상태 정보를 중복하여 수집하는 데이터 중복성 문제가 있었고, 3) 지능형 서비스 로봇 사용자가 원하는 다양한 통신 방법을 제공하지 못해 호환성이 부족하다는 한계점을 가지고 있었다.

본 논문에서는 이러한 한계점을 극복하기 위하여 지능형 서비스 로봇의 상태 정보를 생성 수집하기 위한 지능형 서비스 로봇 모니터링 시스템을 제안한다. Fig. 2는 모니터링 시스템의 구조를 나타낸 것이다.

이 모니터링 시스템은 1) 행위 기반 에이전트 상태 정보 수집 방법을 통해 에이전트의 행위 수행에 따라 생성되는 행위 로그를 기반으로 에이전트의 상태 정보를 일관적으로 생성 및 수집할 수 있다. 다음으로 2) 모니터링 수요 기반 상태 정보 수집을 통해 데이터 중복 문제를 해결한다. 또한 3) 프록시를 통한 상태 정보 전달 방법을 통해 ActiveMQ, ZeroMQ, 소켓 등 사용자가 원하는 통신 방법을 활용하여 지능형 서비스 로봇의 상태 정보를 전달할 수 있다.

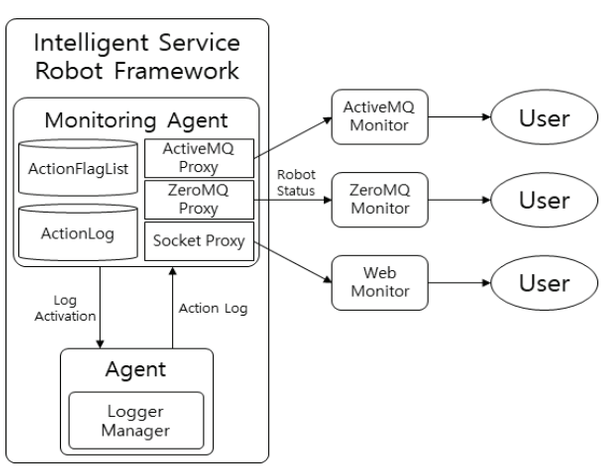

Fig. 2. Intelligent Service Robot Monitoring System

4.1 행위 기반 에이전트 상태 정보 생성 방법

에이전트는 센서를 통해 환경을 인식하고 액츄에이터를 통해 환경을 변화시키는 개체[13]이다. 이러한 에이전트는 능동적으로 센서를 활용하여 환경의 정보를 수집하며, 이러한 행동은 액츄에이터를 통해 환경에 영향을 미치는 것과 더불어 하나의 행위 단위로 정의될 수 있다.

본 연구에서는 이러한 점에 착안하여 에이전트의 행위를 기반으로 에이전트의 상태 정보를 생성하고 수집하는 방법을 제안한다. 각 에이전트는 행위 수행 로그 생성 및 관리를 위한 LoggerManager를 가지며, 이를 활용하여 행위 수행 시 일괄적으로 자신의 상태 정보를 생성한다.

LoggerManager는 registAction, deleteAction, changeActionLogging의 3가지 메소드를 가진다. registAction, deleteAction은 해당 에이전트가 수행하는 행위를 등록 및 삭제하기 위한 메소드이며, changeActionLogging은 행위의 로그 생성 여부를 변경하기 위한 것이다. Table 1은 LoggerManager의 메소드 명세를 나타낸 것이다.

LoggerManager에 에이전트의 Action을 등록 및 삭제하기 위해서는, 먼저 에이전트의 AgentAction의 형태로 구성하여야 한다. AgentAction은 에이전트가 행위 수행 시 동작해야 할 코드를 별도로 분리한 것으로, 행위의 이름과 실행용 코드로 구성된다. registAction 메소드를 통해 행위를 등록하는 경우, 에이전트는 행위 수행 시 로그의 생성 시점을 행위 수행 전, 수행 후, 수행 전과 후 모두 혹은 생성하지 않음의 4가지로 지정할 수 있다.

changeActionLogging 메소드는 행위 수행 로그 생성 시점을 조절하기 위한 것으로 Table 3과 같은 인자를 필요로 한다. 이는 모니터링 에이전트가 로그 수집 과정에서 더 이상 필요로 하지 않거나, 혹은 새롭게 수집이 필요한 행위가 존재할 경우에 사용된다.

위와 같은 메소드들을 사용하여 행위를 등록한 이후에 에이전트가 행위를 수행하면, LoggerManager는 행위 로그를 생성하여 모니터링 에이전트에게 전달한다. Table 4는 행위 로그 메시지의 명세를 나타낸 것이다.

Table 1. LoggerManager Method Specification

Method Name	Description	Arguments
registAction	Regist Agent Action	AgentAction, LogTiming
deleteAction	Delete Agent Action	AgentAction
changeAction Logging	Change action log activation	ActionName, LogTiming

Table 2. Action Log Message Specification

Name	Description	Type
Actor	The agent that performed the action	String
Action Name	Name of action	String
Content	action log content	String
time	The time in milliseconds when the action log was created	String

앞서 제시한 LoggerManager를 활용한 행위 기반 에이전트 상태 정보 수집 방법을 적용하면, 멀티 에이전트 시스템을 적용한 지능형 로봇에서 로봇의 상태 정보 생성을 위해 필요한 에이전트 상태 정보를 일괄적으로 수집 및 활용할 수 있다.

4.2 모니터링 수요 기반 상태 정보 수집 방법

지능형 로봇 사용자들은 다양한 이유로 로봇의 상태 정보를 활용하여 로봇의 상태를 확인하고자 하며, 사용자들이 원하는 상태 정보는 사용자의 요구에 따라 그 범위가 겹칠 수 있다. 이러한 여러 사용자가 요구하는 로봇 상태 정보를 사용자의 요청이 발생한 수대로 수집한다면 동일한 정보를 중복하게 수집하게 된다.

이러한 한계점을 해결하기 위하여 본 연구에서는 모니터링 수요 기반의 상태 정보 수집 방법을 제안한다. 모니터링 에이전트는 사용자의 모니터링 수요를 Action Flag의 형태로 관리하며, Action Flag는 에이전트의 행위 수행 시 로그 생성 여부를 나타내는 것이다. 모니터링 에이전트는 지능형 로봇 내 에이전트가 수행하는 모든 행위의 Flag를 관리한다. 모니터링 에이전트는 Action Flag의 활성화 여부에 따라 에이전트 상태 정보를 수집하고 이를 전달하여 데이터 중복 수집 문제를 해결한다. Fig. 3은 Action Flag를 사용하여 상태 정보를 수집하는 과정을 나타낸 것이다.

모니터링 수요 기반 상태 정보 수집 과정은 다음과 같은 절차로 이루어진다. 1)로봇 사용자가 모니터를 통해 모니터링 에이전트에게 상태 정보를 요청한다. 2)모니터링 에이전트는 사용자가 요청한 행위의 Flag가 활성화되어 있는지 확인한다. 활성화되어 있지 않은 경우, 해당 행위의 Flag를 활성화한다. 3)Action Flag가 새롭게 활성화되었다면, 모니터링 에이전트는 해당 행위를 수행하는 에이전트의 LoggerManager를 통해 로그 생성을 활성화한다. 4)에이전트가 행위를 수행하면 자동적으로 해당 행위에 대한 로그가 생성되고, 모니터링 에이전트에게 전달된다.

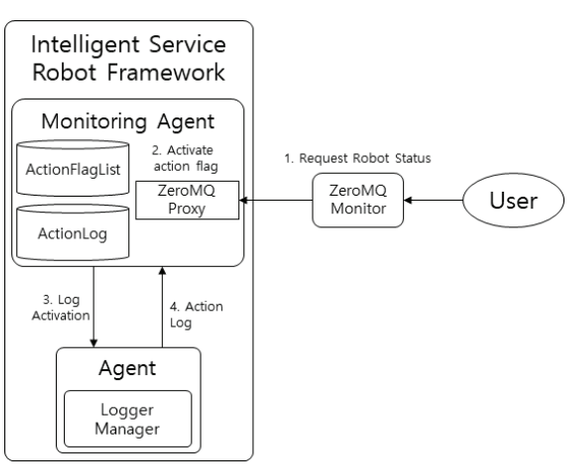

Fig. 3. Robot Status Request Process Using Action Flag

이러한 모니터링 수요 기반 상태 정보 수집 방법을 통해, 멀티 에이전트 시스템에서 지능형 로봇의 상태 정보 생산을 위해 필요한 에이전트 상태 정보를 중복 없이 수집할 수 있다.

4.3 프록시를 통한 상태 정보 전송 방법

각 지능형 로봇 사용자들은 사용 목적에 따라 필요로 하는 지능형 로봇 정보가 다르며, 로봇 정보를 전달받는 데 사용하는 통신 방법이 다를 수 있다.

따라서 지능형 로봇은 각 사용자가 원하는 로봇의 상태 정보를 사용자가 원하는 통신 방법으로 제공할 수 있어야 한다. 이를 위해 본 연구에서는 프록시를 통한 지능형 로봇 상태 정보 전송 방법을 제안한다.

프록시를 통한 지능형 로봇 상태 정보 전송 방법은 다음과 같은 과정으로 이루어진다. 먼저, 모니터링 에이전트는 처음 동작 시, 사용자가 사용하는 통신 방법을 초기화한다. 사용자는 자신이 필요로 하는 로봇 상태 정보를 모니터링 에이전트에게 요청한다. Table 3은 로봇 정보 요청 메시지 형식을 나타낸 것이다. 여기서 ActionList에 들어가는 것은 자신이 필요한 에이전트 행위 목록이며, Flag는 로그 활성화 여부이다. 모니터링 에이전트는 사용자가 접속한 통신 방법 사용하는 프록시를 생성하며, 해당 프록시는 사용자가 요청한 행위 목록을 갖는다.

각 에이전트가 행위 수행을 통해 상태 정보를 생성하면, 모니터링 에이전트는 모니터링 수요에 따라 에이전트 상태 정보를 수집한다. 그 후, 프록시가 가지고 있는 사용자가 필요로 하는 행위 목록을 확인하여, 해당하는 행위 로그를 정보를 사용자에게 전달한다.

일반 지능형 로봇 사용자는 개발자나 관리자와 다르게, 로봇 사용이 끝나면 더 이상 로봇의 상태 정보가 필요하지 않을 수 있다. 이런 경우, Flag를 false로 설정하여 로봇 정보 요청 메시지를 보내는 것을 통해 지능형 로봇 정보 구독을 중단할 수 있다.

이러한 프록시를 통한 상태 정보 전송 방법을 통해, 모니터링 에이전트는 일괄적으로 수집된 상태 정보를 사용자가 원하는 통신 방법을 사용하여 전달할 수 있다.

Table 3. Robot Status Request Message Specification

Name	Description	Type
UserID	Identifier of user	String
ActionList	Robot status that robot user wants	List 〈ActionString〉
Flag	The activation status of the filter	boolean

5. 실 험

5.1 실험 개요

본 논문에서는 지능형 서비스 로봇을 위한 모니터링 시스템을 제안했다. 이 시스템은 1) 행위 기반의 상태 정보 생성을 통해 에이전트의 상태 정보를 일괄적으로 생성하고, 2) 모니터링 수요 기반의 상태 정보 수집을 통해 필요한 정보를 중복 없이 수집하며, 3) 프록시를 통한 상태 정보 전송 방법을 통해 사용자가 원하는 통신 방법으로 로봇의 데이터를 전달할 수 있다.

실험에서는 이 시스템을 검증하기 위하여, 지능형 서비스 로봇의 시나리오 기반 서비스 수행 과정 중 로봇의 상태 정보를 수집하고 이를 외부에서 확인하고자 한다.

또한 관련 연구에서 분석한 다른 모니터링 방법과의 비교를 진행하고자 한다. 이중 Monajjemi[4]와 Changyun Wei[5]이 제시한 방법은 행위 수준의 모니터링을 제공하고 있어 본 연구에서 제시하는 의사결정과 같은 인지 수준의 기능 모니터링과 비교하기 적절하지 않을 것으로 판단된다. 따라서 Guessom[3]이 제시한 메시지 기반의 모니터링 방법을 비교군으로 설정하여, 본 논문에서 제시하는 모니터링 방법과 비교 및 분석하고자 한다.

5.2 실험 설계

지능형 서비스 로봇의 모니터링을 검증하기 위하여 지능형 서비스 로봇이 서비스를 수행하고, 본 논문에서 제시한 시스템을 사용해 수행 과정을 모니터링하고자 한다.

이를 위해 실제 로봇을 활용한 실험을 구성하였다. 실험에 사용한 로봇은 Activemedia사의 PeopleBot이다. 이 로봇은 ROS 통신을 지원하며, 바퀴를 활용한 이동 기능, 외부 TTS 및 STT 서비스를 활용한 음성인식 및 발화 기능을 제공한다. 다음 Fig. 4는 실험 중에 촬영한 사람과 상호작용하는 PeopleBot의 사진이다.

또한 모니터링 시스템의 기능 중 하나인 사용자가 원하는 통신 방법을 활용한 로봇 데이터 전송을 검증하기 위해 3종

Fig. 4. PeopleBot

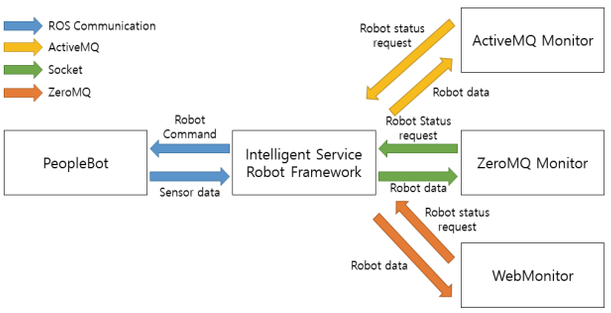

Fig. 5. Configuration of Communication Relay Between Robot, Intelligent Service Robot Framework and Monitors

류의 통신 방법을 사용하는 모니터를 구성하였다. 각 모니터는 ZeroMQ, ActiveMQ 및 소켓의 통신 방법을 사용하여 로봇의 상태 정보를 요청한다. Fig. 5는 실험 내 프로세스 간 통신 관계를 나타낸 것이다. 그림의 화살표는 통신 방법에 따라 색을 다르게 표시하였다.

또한 다음의 Table 4는 본 실험에서 사용되는 행위 목록을 나타낸 것이다.

AssertWorldModel, RetractWorldModel은 로봇이 센서를 통해 관측한 환경 정보를 자신의 세계 모델에 입력하거나 제거하는 Action으로, 주로 외부에서 요청이 들어오거나, 작업이 완료된 경우에 수행된다. 심부름 서비스 내에서 활용되는 환경 정보는 로봇이 인지한 사용자의 요청을 서술하는 RequestPerceived, 로봇의 위치를 서술하는 LocatedIn, 이동 목적지를 서술하는 Destination 등이 있다. 다음 Table 5는 AssertWorldModel 로그의 예시이다.

Table 4. List of Action in Delivery Scenario

Action Name	Description
AssertWorldModel	Assert a new fact to the world model of task management.
RetractWorldModel	Retract a fact from the world model of task management
PostGoal	Add a goal to intention
UnpostGoal	Remove a goal from intention
ExecuteGoal	Execute a perform or achieve goal

Table 5. Example of AssertWorldModel Action Log

```
(SystemLog
  (actor "agent://www.arbi.com/taskManager")
  (action "AssertWorldModel")
  (content
    "LocatedIn
    (test://test.ai/person01,
    test://test.ai/table)“
  )
  (time "1718951686097")
)
```

PostGoal은 자신이 수행할 목표를 해야할 일 목록에 추가하는 것이며, UnpostGoal은 달성이 완료된 목표나 혹은 더 이상 수행하지 않을 목표를 해야할 일 목록에서 제거하는 것이다. 본 실험에서 사용되는 목표는 발화를 위한 SpeakTo, 이동을 위한 MoveTo, 물건을 획득하는 GrabObject, 물건을 건네는 HandOver 등이 있으며, 서비스 진행 상황에 따라 각 목표가 PostGoal을 통해 추가되거나 UnpostGoal을 통해 제거된다. 다음 Table 6은 PostAction의 로그 예시이다.

ExecuteGoal은 해야 할 일 목록에 존재하는 목표 중 가장 우선도가 높은 목표 달성을 위해 작업을 실행하는 것이다. 여기서 목표는 앞서 설명한 PostGoal이나 UnpostGoal과 동일한 것이다. PostGoal 된 목표가 ExecuteGoal을 통해 수행되고, 수행이 완료되면 UnpostGoal을 통해 제거되는 과정을 거친다. 다음 Table 7은 ExecuteGoal 로그의 예시이다.

마지막으로 실험에 사용하고자 하는 시나리오를 설계했다. 본 실험에서 검증하고자 하는 것은 모니터링 시스템을 사용하여 지능형 로봇의 상태 정보를 수집 및 사용자에게 전달하는 것을 확인하는 것이므로, 복잡한 추론이 필요하지 않은 시나리오를 구성하고자 하였다. 따라서 사용자의 요청에 따라 이동, 발화 등을 수행하는 심부름 서비스를 설계하였다. 다음 Fig. 6은 심부름 서비스 시나리오를 나타낸 것이다.

심부름 서비스 시나리오는 다음과 같이 구성된다. 1) 사람이 로봇에게 물을 가져다 달라고 요청한다. 2) 로봇은 이를 인식하고 SpeakTo 작업 수행을 통해 사람에게 요청을 인지했다는 대답을 한다. 3) ActiveMQ 모니터는 ExecuteGoal, AssertWorldModel, RetractWorldModel에 대한 로그를, ZeroMQ 모니터 및 웹모니터는 ExecuteGoal에 대한 로그를 로봇에게 요청한다. 4) 로봇은 MoveTo 작업 수행을 통해 물병이 있는 위치로 이동한다. 5) ZeroMQ 모니터가 추가적으로 PostGoal과 UnpostGoal에 대한 로그를 요청한다. 6) 로봇은 물병을 획득하는 GrabObject 작업을 수행한다. 여기서 실제로는 로봇이 직접 물을 잡아야 하지만, PeopleBot에는 물건을 잡을 수 있는 팔과 같은 하드웨어가 없다. 따라서 사람이 로봇에 물을 올려두고 로봇의 화면에 나타나는 UI의 버튼을 누르는 것으로 물병을 잡는 과정을 대체하였다. 7) 로봇이 MoveTo 작업 수행을 통해 다시 처음 위치로 돌아가 SpeakTo 작업 수행을 통해 사람에게 물병을 가져왔음을 알린다. 그 후 로봇은 HandOver 작업 수행을 통해 물을 건내줄 준비를 한다. 8) 물을 요청한 사용자는 로봇에게 있는 물병을 획득하고, 로봇 화면의 버튼을 눌러 로봇의 서비스를 종료한다.

Table 6. Example of PostGoal Action Log

```
(SystemLog
  (actor "agent://www.arbi.com/taskManager")
  (action "PostGoal")
  (content "GrabObject")
  (time "1718951666087")
)
```

Table 7. Example of ExecuteGoal Action Log

```
(SystemLog
  (actor "agent://www.arbi.com/taskManager")
  (action "ExecuteGoal")
  (content "MoveTo")
  (time "1718951686098")
)
```

지능형 서비스 로봇을 위한 모니터링 시스템을 검증하기 위하여 본 실험에서는 로봇이 위 서비스를 수행하는 과정에서, 1) 작업 관리 기능의 행위 수행에 따라 로그가 정상적으로 생성되는지, 2) 모든 행위 중 사용자가 요청한 행위 로그만 중복 없이 수집되는지, 3) ZeroMQ, ActiveMQ 및 소켓을 사용하는 3개의 모니터로 활용하여 각기 다른 통신 방법을 통해 상태 정보를 전달받을 수 있는지를 검증하고자 한다.

또한 비교군으로 설정한 메시지 기반 모니터링 방법을 적용한 실험은 앞서 설명한 시나리오와 동일한 것을 사용하며,

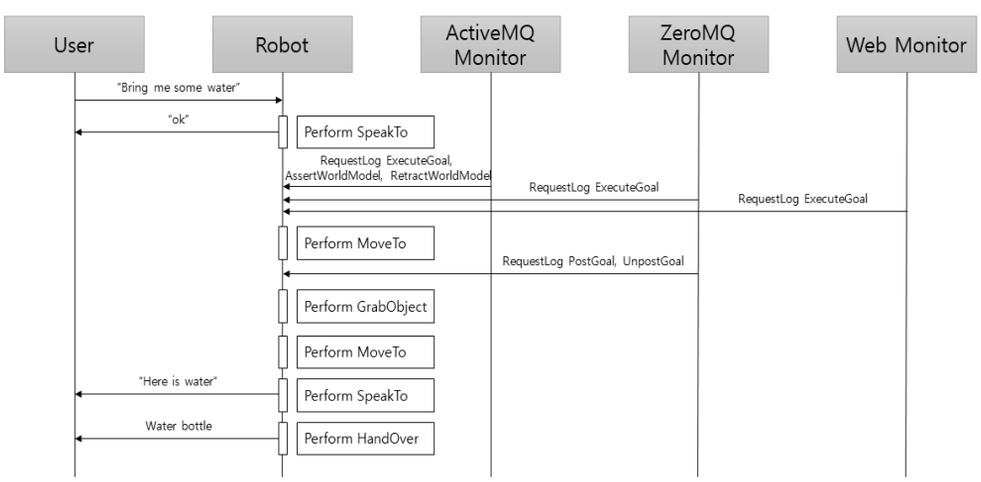

Fig. 6. Delivery Service and Monitoring Flow

사용자의 모니터링 요청이 발생할 때마다 작업 관리 기능이 주고받는 메시지를 수집하는 것으로 설계하였다.

5.3 실험 결과

1) 행위 기반 상태 정보 생성 검증

행위 기반 상태 정보 생성 검증을 위해 앞서 설계한 서비스 수행 과정 중 서비스 로봇의 상태 정보가 행위 수행에 따라 생성되는지 로그를 통해 확인하고자 한다.

Table 8은 서비스를 시작한 직후의 작업 관리 기능의 로그, Table 9는 모니터링 에이전트의 로그이며 각 로그는 서비스 시작 직후부터 이동을 시작하기까지의 부분이다. Table 8의 첫 네 줄을 보면 사람으로부터 요청을 받아 로봇이 발화 작업을 수행하지만 별도로 로그를 생성하지 않고 있는 모습을 볼 수 있다. 그 후 로그 활성화 메시지를 받자 다음 작업인 이동 작업을 시작할 때는 AssertWorldModel, ExecuteGoal 등의 행위 수행에 따라 Action Log Message Sent와 같이 로그를 생성하여 모니터링 에이전트에게 전달하는 것을 확인할 수 있다. 또한 Table 9의 모니터링 에이전트에서도 처음 SpeakTo 작업에 대한 로그는 수집되지 않은 반면, 로그가 활성화 되고 난 뒤인 MoveTo 작업부터 로그가 수집되는 것을 확인할 수 있다.

이를 통해, 에이전트가 행위 수행을 통해 상태 정보를 생성하고 이를 자동적으로 수집하는 것을 확인할 수 있었다.

2) 모니터링 수요 기반 상태 정보 수집 검증

모니터링 수요 기반 상태 정보 수집을 검증하기 위해, 서로 다른 모니터가 동일한 상태 정보를 요청하고 이에 대해 중복없이 수집하는지 확인하고자 한다.

Table 9의 모니터링 에이전트의 로그에서 Create Monitor-Request를 보면 각 모니터로부터 한 번씩, 총 세 번의 상태 정보 수집 요청을 받는 것을 볼 수 있으며, 모든 요청은 동일하게 ExecuteGoal 행위에 대한 정보를 요청하고 있다. 하지만 첫 번째 요청을 받아 ExecuteGoal 행위에 대해 Action Flag가 활성화된 이후의 요청에서는 별도의 Flag 활성화가 이루어지지 않음을 볼 수 있다. 또한 표 내용의 마지막 부분에서 ExecuteGoal 행위를 통해 MoveTo 작업이 수행되어 Received log message와 같이 로그를 한 번만 수집하고, Sending log를 세 번 실행하여 각 모니터에게 전달하는 것을 볼 수 있다.

이를 통해 모니터링 수요 기반 상태 정보 수집 방법을 적용하여 중복없이 상태 정보를 수집하는 것을 확인할 수 있었다.

3) 프록시를 통한 상태 정보 전송 검증

프록시를 통한 상태 정보 전송을 검증하기 위해, ActiveMQ, ZeroMQ, 소켓과 같이 서로 다른 통신 방법을 사용하는 모니터를 사용하여 동시에 로봇 상태 정보를 모니터링하고자 한다.

Table 10은 ActiveMQ를 사용하는 모니터의 로그, Table 11.는 ZeroMQ를 사용하는 모니터의 로그, Table 12.는 소켓을 사용하는 웹모니터의 로그이다. 각 로그를 보면 ExecuteGoal, AssertWorldModel, RetractWorldModel 등의 행위에 대해 모니터링 정보를 요청하였으며, 자신이 요청한 행위의 로그만 받는 것을 확인할 수 있다. 또한 ZeroMQ 모니터는 첫 번째 상태 정보 요청 이후에 추가적으로 PostGoal에 대한 정보를 요청하였고, 요청 이전에 수행된 MoveTo 작업에 대한 PostGoal 행위 로그는 전달받지 못했지만, 이후에 수행된 GrabObject 작업에 대한 PostGoal 로그는 전달받는 것을 확인할 수 있다.

이를 통해, 모니터링 에이전트가 사용자가 원하는 통신 방법을 활용하여 지능형 로봇의 상태 정보를 전달할 수 있음을 확인할 수 있었다.

Table 8. Log of Task Management in Intelligent Service Robot Framework

```
executed : RequestPerceived(test://test.ai/person01,bringWater)
executed : SpeakTo
SpeakTo Plan started : ok
SpeakTo Plan finished : ok
[ Activation Message Received ] {"Action":"ExecuteGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}
[ Activation Message Received ] {"Action":"AssertWorldModel","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}
[ Activation Message Received ] {"Action":"RetractWorldModel","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}
[Action Log Message Sent]  (SystemLog (actor "agent://www.arbi.com/taskManager")(action "AssertWorldModel") (content
"Destination(test://test.ai/table01,0.4,4.2,0)") (time "1718951576080"))
executed : MoveTo
[Action Log Message Sent]  (SystemLog (actor "agent://www.arbi.com/taskManager")(action "ExecuteGoal") (content "MoveTo")
(time "1718951576081"))
MoveTo Plan started : test://test.ai/table01
[ Activation Message Received ] {"Action":"PostGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}
[ Activation Message Received ] {"Action":"UnpostGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}
MoveTo Plan finished : test://test.ai/table01
[Action Log Message Sent]  (SystemLog (actor "agent://www.arbi.com/taskManager")(action "UnpostGoal") (content "MoveTo")
(time "1718951666087"))
...
```

Table 9. Log of Monitoring Agent in Intelligent Service Robot Framework

{"ID":"ZeroMQMonitor/message","Action":"Create Monitor","Filter" : [{"Actor":"agent://www.arbi.com/taskManager", "Action":"ExecuteGoal", "Flag":"true"}]}
[Create MonitorRequest] :[{"Action":"ExecuteGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}]
[Proxy Created] : monitoringID : ZeroMQMonitor/message , protocol : ZeroMQ, actionFlag : [{"Action":"ExecuteGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}]
[Changing ActionFlag] :{"Action":"ExecuteGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}
Socket message received : {"ID":"Socket/message","Action":"Create Monitor","Filter" : [{"Actor":"agent://www.arbi.com/taskManager", "Action":"ExecuteGoal", "Flag":"true"}]}
[Create MonitorRequest] :[{"Action":"ExecuteGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}]
[Proxy Created] : monitoringID : Socket/message , protocol : Socket, actionFlag : [{"Action":"ExecuteGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}]
[Create MonitorRequest] :[{"Action":"ExecuteGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"},{"Action":"AssertWorldModel","Actor":"agent://www.arbi.com/taskManager","Flag":"true"},{"Action":"RetractWorldModel","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}]
[Proxy Created] : monitoringID : monitor1 , protocol : ActiveMQ, actionFlag : [{"Action":"ExecuteGoal","Actor":"agent://www.arbi.com/taskManager","Flag":"true"},{"Action":"AssertWorldModel","Actor":"agent://www.arbi.com/taskManager","Flag":"true"},{"Action":"RetractWorldModel","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}]
[Changing ActionFlag] :{"Action":"AssertWorldModel","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}
[Changing ActionFlag] :{"Action":"RetractWorldModel","Actor":"agent://www.arbi.com/taskManager","Flag":"true"}
[Received system log] :(SystemLog (actor "agent://www.arbi.com/taskManager") (action "AssertWorldModel") (content "Destination(test://test.ai/table01,0.4,4.2,0)") (time "1718951576080"))
[Sending log] : receiver is "monitor1", {"Action":"AssertWorldModel","Content":"Destination(test://test.ai/table01,0.4,4.2,0)","Actor":"agent://www.arbi.com/taskManager","Time":"1718951576080"}
[Received system log] :(SystemLog (actor "agent://www.arbi.com/taskManager") (action "ExecuteGoal") (content "MoveTo") (time "1718951576081"))
[Sending log] : receiver is "ZeroMQMonitor/message", {"Action":"ExecuteGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951576081"}
[Sending log] : receiver is "Socket/message", {"Action":"ExecuteGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951576081"}
[Sending log] : receiver is "monitor1", {"Action":"ExecuteGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951576081"}
...

Table 10. Log of ActiveMQ Monitor

log4j:WARN No appenders could be found for logger (org.apache.activemq.transport.WireFormatNegotiator).
log4j:WARN Please initialize the log4j system properly.
activeMQ broker connected : tcp://127.0.0.1:61611
Message Sent = {"ID":"monitor1","Action":"Create Monitor","Filter" : [{"Actor":"agent://www.arbi.com/taskManager", "Action":"ExecuteGoal", "Flag":"true"},{"Actor":"agent://www.arbi.com/taskManager", "Action":"AssertWorldModel", "Flag":"true"},{"Actor":"agent://www.arbi.com/taskManager", "Action":"RetractWorldModel", "Flag":"true"},]}
Message received = {"Action":"ExecuteGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951576081"}
Message received = {"Action":"AssertWorldModel","Content":"Destination(test://test.ai/table01,0.4,4.2,0)","Actor":"agent://www.arbi.com/taskManager","Time":"1718951576080"}
Message received = {"Action":"ExecuteGoal","Content":"GrabObject","Actor":"agent://www.arbi.com/taskManager","Time":"1718951666088"}
Message received = {"Action":"AssertWorldModel","Content":"LocatedIn(test://test.ai/person01,test://test.ai/table)","Actor":"agent://www.arbi.com/taskManager","Time":"1718951686097"}
Message received = {"Action":"AssertWorldModel","Content":"Destination(test://test.ai/table,3.1,1.7,0)","Actor":"agent://www.arbi.com/taskManager","Time":"1718951686098"}
Message received = {"Action":"ExecuteGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951686098"}
Message received = {"Action":"ExecuteGoal","Content":"SpeakTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951776114"}
...

Table 11. Log of ZeroMQ Monitor

message sent = {"ID":"ZeroMQMonitor/message","Action":"Create Monitor","Filter" : [{"Actor":"agent://www.arbi.com/taskManager", "Action":"ExecuteGoal", "Flag":"true"}]}
Message received = {"Action":"ExecuteGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951576081"}
message sent = {"ID":"ZeroMQMonitor/message","Action":"Change Filter","Filter" : [{"Actor":"agent://www.arbi.com/taskManager", "Action":"PostGoal", "Flag":"true"},{"Actor":"agent://www.arbi.com/taskManager", "Action":"UnpostGoal", "Flag":"true"}]}
Message received = {"Action":"UnpostGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951666087"}
Message received = {"Action":"PostGoal","Content":"GrabObject","Actor":"agent://www.arbi.com/taskManager","Time":"1718951666087"}
Message received = {"Action":"ExecuteGoal","Content":"GrabObject","Actor":"agent://www.arbi.com/taskManager","Time":"1718951666088"}
Message received = {"Action":"UnpostGoal","Content":"GrabObject","Actor":"agent://www.arbi.com/taskManager","Time":"1718951686097"}
Message received = {"Action":"PostGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951686098"}
Message received = {"Action":"ExecuteGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951686098"}
...

Table 12. Log of WebMonitor

Message sent = {"ID":"Socket/message","Action":"Create Monitor","Filter" : [{"Actor":"agent://www.arbi.com/taskManager", "Action":"ExecuteGoal", "Flag":"true"}]}
Server response: {"Action":"ExecuteGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951576081"}
Server response: {"Action":"ExecuteGoal","Content":"GrabObject","Actor":"agent://www.arbi.com/taskManager","Time":"1718951666088"}
Server response: {"Action":"ExecuteGoal","Content":"MoveTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951686098"}
Server response: {"Action":"ExecuteGoal","Content":"SpeakTo","Actor":"agent://www.arbi.com/taskManager","Time":"1718951776114"}
Server response: {"Action":"ExecuteGoal","Content":"HandOver","Actor":"agent://www.arbi.com/taskManager","Time":"1718951786122"}

4) 실험 결과 분석

이 단락에서는 메시지 기반 모니터링 방법과 본 연구에서 제시하는 모니터링 방법을 비교 분석하고자 한다. 다음 Table 13은 본 연구에서 제시한 수요 기반 모니터링 방법을 통해 수집된 로그의 수와 메시지 기반 모니터링 방법을 통해 수집된 로그의 수를 나타낸 것이다.

본 연구에서 제시하는 모니터링 방법은 총 19개의 로그를 수집하였으며, 이 중 중복 수집된 로그는 없었다. 또한 사용자의 수요에 따라 필요한 로그만 수집하였으므로 사용자의 수요와 관련없는 로그는 수집되지 않았음을 볼 수 있다.

하지만 메시지 기반 모니터링 방법을 적용한 실험에서는 총 281개의 로그가 수집되었으며, 이중 181개는 사용자의 요청과 관련이 없는 ack 메시지나 로봇 센서 데이터 등이 수집되었다. 또한 외부에서 요청이 발생할때마다 로그를 수집함으로 인해 중복되어 수집된 로그가 40개가 있었다. 이로 인해, 전체 수집된 로그 중 실제로 사용자가 필요로 하는 로그의 비율이 약 25 퍼센트 정도로 본 연구에서 제시한 방법보다 매우 떨어지는 모습을 보였다.

Table 13. Result of Monitoring Method

	Demand-driven agent action monitoring	Message based monitoring
collected log count (a)	19	281
related log count (b)	19	101
duplicated log count	0	40
ratio (b-c/a)	1	0.253

이와 같은 실험을 통해 본 연구에서 제시하는 모니터링 시스템이 다른 모니터링 방법에 비하여 메시지의 수집량이나 수집된 메시지의 관련성 등에서 효율적임을 볼 수 있었다.

6. 결 론

기존 지능체계는 지능형 로봇 사용자에게 로봇의 상태 정보를 전달하는 데 있어 1) 에이전트의 상태 정보를 생성하지 않거나, 생성하는 방법이 서로 달라 일괄적으로 수집할 수 없었으며, 2) 사용자 요청에 따라 데이터를 수집하므로 데이터를 중복 수집하여 에이전트의 성능 하락이 발생할 가능성이 있었고, 3) 다양한 통신 방법을 지원하지 못해 호환성이 떨어진다는 한계점을 가지고 있었다.

본 연구에서는 이러한 한계점을 극복하기 위하여 지능형 로봇을 위한 멀티 에이전트 모니터링 시스템을 제안했다. 이 시스템은 1) 행위 기반 상태 정보 생성을 통해 에이전트의 상태 정보를 일괄적으로 생성하고, 2) 모니터링 수요 기반 상태 정보 수집을 통해 필요한 데이터를 중복 없이 수집하며, 3) 프록시를 활용한 상태 정보 전달을 통해 사용자가 원하는 통신 방법을

활용하여 로봇 상태 정보를 전달할 수 있어, 기존 지능체계의 한계점을 해결했다. 또한 실험을 통해 모니터링 시스템의 각 기능이 기존의 한계점을 해결하는 것을 확인할 수 있었다.

이처럼 본 연구에서는 지능형 로봇의 상태 정보를 사용자에게 전달하는 방법에 집중했다. 하지만 사용자가 로봇을 모니터링하다 보면, 상태 정보를 활용하기 위해선 해당 상태 정보를 바탕으로 로봇을 다시 제어할 수 있어야 한다. 향후 연구에서는 행위 수행 로그의 수집뿐만이 아닌 사용자가 외부에서 직접 에이전트의 행위를 실행시킬 수 있는 방법을 추가하고자 한다.

또한 본 연구에서 제안한 모니터링 시스템은 보안과 관련된 부분을 크게 고려하지 않았다. 향후 연구에는 이러한 부분 또한 보강하여 통신 중 메시지 암호화 및 사용자 인증 등 보안을 강화할 수 있는 부분 또한 개발 및 검증하려고 한다.

References

[1] 이상형 and 신희선, "로봇역사문화연구 (2): 네트워크 기반 지능형 서비스 로봇 사업," Korea Robotics Society Review, Vol.16, No.4, pp.22-36, 2019.

[2] B. G. Choi, "A specification-based service development for intelligent service robots," Ph.D. dissertation, University of Seoul at Seoul, Korea, 2020.

[3] Z. Guessoum, M. Ziane, and N. Faci, "Monitoring and organizational-level adaptation of multi-agent systems," in *AAMAS*. Vol.4. pp.514-522, 2004.

[4] M. Monajjemi, J. Wawerla, and R. Vaughan, "Drums: A middleware-aware distributed robot monitoring system," in *Proceedings of the Doctoral Symposium of the 16th International Middleware Conference*, pp.1-4, 2015.

[5] C. Wei, and K. V. Hindriks, "An agent-based cognitive robot architecture," in *International Workshop on Programming Multi-Agent Systems*. Berlin, Heidelberg, 2012.

[6] K. V, Hindriks "Programming rational agents in GOAL," Multi-agent programming: Languages, tools and applications, Boston, MA: Springer US, pp.119-157, 2009.

[7] S. J. Lee and I. C. Kim, "A spatio-temporal context query processing framework for service robots," *Journal of Institute of Control, Robotics and Systems*, Vol.25, No.1, pp.37-48, 2019.

[8] D. S. Chang, G. H. Cho, and Y. S. Choi, "Ontology-based knowledge model for human-robot interactive services," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp.2029-2038, 2020.

[9] B. G. Choi, I. S. Yu, and J. H. Lee, "A policy-based meta-planning for general task management for multi-domain services," *The Transactions of the Korea Information Processing Society (KTSDE)*, Vol.8, No.12, pp.499-506, 2019.

[10] B. G. Choi, J. U. Lee, and J. H. Lee, "A model-based interface to cloud services for intelligent service robots" *The Transactions of the Korea Information Processing Society (KTSDE)*, Vol.8, No.1 pp.1-10, 2020.

[11] C. S. Park, B. G. Choi, and J. H. Lee, "Behavior interface for robot intelligence integration framework," in *Korea Computer Congress 2018*, pp.935-937, 2018.

[12] H. G. Lee, J. U. Lee, B. G. Choi, and J. H. Lee, "Model-based IoT Linkage and Abstraction Interface for Intelligent Service Robot: Virtual Device," in *Korea Computer Congress 2019*, pp.331-333, 2019.

[13] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach," 3th ed., New York: Prentice Hall, 2009.

조 한 얼

https://orcid.org/0000-0002-1022-4679
e-mail : dudung123@gmail.ac.kr
2018년 서울시립대학교 컴퓨터과학과(학사)
2022년 서울시립대학교 전자전기컴퓨터공학과(석사)
2022년 ~ 현 재 서울시립대학교 정보기술연구소 연구원
관심분야 : 지능형 로봇, 인공지능

유 인 식

https://orcid.org/0000-0001-8086-1342
e-mail : youin10uosai@gmail.com
2018년 서울시립대학교 환경공학부(학사)
2020년 서울시립대학교 전자전기컴퓨터공학과(석사)
2020년 ~ 현 재 서울시립대학교 전자전기컴퓨터공학과 박사과정
관심분야 : 인공지능, 지능형 로봇, 클라우드 로봇, 멀티 로봇

이 재 호

https://orcid.org/0000-0002-3332-3207
e-mail : jaeho@uos.ac.kr
1985년 서울대학교 계산통계학과(학사)
1987년 서울대학교 계산통계학과(석사)
1997년 University of Muchigan(박사)
1998년 ~ 현 재 서울시립대학교 전자전기컴퓨터공학부 교수
관심분야 : 인공지능, 지능 로봇