JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 26, No. 12, pp. 1-4, Dec. 2025 (수정 및 삭제 금지)

설계 중심의 Multi AI Agent Design Methodology(MADM) 제안: RACID 역할 모델 중심으로

천지영[1] · 윤석용[2]*
[1]명지대학교 AI정보과학전공 석사과정 · [2]2명지대학교 AI정보과학전공 교수

Multi Agent Design Methodology (MADM) for AI System: A Design-Centric Approach Based on the RACID Role Model

Ji-Yeong Cheon[1] · Seok-Yong Yun[2]

[1]Master's Department of AI Information Science, MyongJi Graduate School of Records, Archives & Information Science Seoul 03674, Korea
[2]Professor, Department of AI Information Science, MyongJi Graduate School of Records, Archives & Information Science Seoul 03674, Korea

[요 약]

다중 에이전트 시스템(MAS)은 복잡한 문제 해결에 활용할 수 있는 기술로 최근 인공지능 분야에서 주목받고 있다. 그러나 설계 단계의 복잡성과 불확실성으로 인해 구현 과정에서 높은 실패율이 보고되고 있다. 기존 소프트웨어 개발 방식은 MAS의 복잡한 상호작용 구조나 LLM 기반 시스템의 비결정성을 다루기 어렵고, 현행 구현 중심 프레임워크 또한 체계적인 설계 지침이 부족하다. 이러한 한계를 극복하기 위해 본 연구는 설계 중심 방법론인 MADM을 제안한다. MADM은 환경 분석, AI 에이전트 설계, MAS 구현의 세 단계로 구성되며, 분석과 설계의 체계화를 중점적으로 다룬다. 또한 RACI 매트릭스를 확장한 RACID 모델을 적용하여 역할과 책임을 명확히 정의하였다. 식사 메뉴 추천 시스템을 구현해 적용 가능성을 검토했다. MADM이 MAS 개발의 설계 결함을 줄이고 일관성과 신뢰성을 높일 것으로 기대된다.

[Abstract]

The Multi-Agent System (MAS) has recently attracted attention in artificial intelligence as a promising technology for solving complex problems. However, the complexity and uncertainty of the design phase often cause high failure rates. Existing software paradigms are inadequate for managing the interactions of MAS and the non-deterministic behavior of LLM-based systems. To address these issues, this study proposes a design-oriented methodology, MADM (Multi-Agent Design Methodology), consisting of three stages: environment analysis, AI agent design, and MAS implementation. The RACID (Responsible, Accountable, Consulted, Informed, Dynamic Coordinator) model, extended from the RACI matrix, is applied to clarify agent roles and hierarchy. A meal recommendation system demonstrates its applicability. In conclusion, MADM aims to reduce design-stage defects and improve the consistency and reliability of MAS development.

색인어 : AI 에이전트, 다중 에이전트 시스템, 인공지능, 거대언어모델, 개발 방법론

Keyword : AI Agent, Multi Agent System, AI, LLM, Design Methodology

http://dx.doi.org/10.9728/dcs.2025.26.12.1(수정 및 삭제 금지)

Received 30 November 2025; **Revised** 01 December 2025
Accepted 02 December 2025 (수정 및 삭제 금지)

***Corresponding Author; Seok-Yong Yoon**

Tel: +82-2-2668-8080
E-mail: icanibe@mju.ac.kr

Ⅰ. 서 론

대규모 언어 모델(LLM; Large Language Model)의 발전으로 AI는 단순한 추론을 넘어 자율적으로 목표를 설정하고 실행하는 AI 에이전트(AI Agent)로 진화하고 있다. 단일 에이전트로는 해결하기 어려운 복합적인 문제에 대응하기 위해, 여러 에이전트가 유기적으로 협업하는 다중 에이전트 시스템(MAS; Multi Agent System)이 활용되고 있다. 그러나, MAS는 복잡한 상호작용과 동적인 컨텍스트 정보를 다루어야 하므로, 전체 시스템 아키텍처와 역할 조정, 워크플로 설계 전반에서 높은 복잡도를 동반한다[1].

UC Berkeley 연구팀의 MAS Failure Taxonomy(MAST)에 따르면, MAS의 초기 설계의 결함에서 비롯된 사양 설계 실패가 전체의 41.77%이며, 이는 초기 설계 단계에서 상당한 어려움이 존재한다는 것을 보여준다[2]. 이러한 실패는 LLM 자체의 한계보다는 에이전트 시스템의 설계나 에이전트 간 조직구조의 문제에서 비롯되는 경우가 많다[13].

MAS 개발이 높은 실패율을 보이는 이유는 크게 세 가지로 정리할 수 있다. 첫째, 기존 소프트웨어 개발 패러다임의 한계이다. 시스템의 명확한 기능을 세분화하고 설계하는 정적인 설계 방식은[3], MAS의 복잡한 상호작용을 충분히 설명하지 못한다. 각 에이전트가 독립적으로 작동하면서도 집단적 목표를 공유하는 MAS의 특성상, 기존 패러다임은 기능 간 사일로(Silo) 현상을 유발하고 협업의 효율을 어렵게 만든다. 결국 MAS 설계는 단순한 기능의 합산이 아니라, 역할 및 협업 중심의 패러다임으로 접근해야 한다. 둘째, LLM이 가지는 비결정성의 문제이다. 에이전트는 LLM을 기반으로 작동하는 이상, 동일한 입력에도 서로 다른 결과가 나올 수 있는 확률적 성격이 존재한다[4]. 에이전트의 행동은 프롬프트, 프롬프트, 도구, 메모리 구조 등 설계자가 정의한 요소에 전적으로 의존하는데[5], 이러한 구성 요소가 체계적으로 설계되지 않는다면 협업 효과는 급격히 떨어지고 시스템 또한 불안정해진다[6]. 셋째, 구현 중심 프레임워크의 방법론적 공백이다. AutoGen, LangGraph, Google ADK 등은 구현 도구를 제공하지만, '무엇을, 왜, 어떻게 설계해야 하는가'에 대한 가이드라인을 제공하지 못한다. 그 결과, 개발자는 경험이나 직관에 의존해 개발할 수밖에 없으며, 이에 따라 반복적인 시행착오를 거치며 앞서 언급한 높은 실패율로 이어진다.

이러한 근본적인 문제는 MAS 개발 과정에서 체계적이고 설계 중심적인 방법론의 필요성을 시사한다. 이에 본 연구에서는 MAS의 높은 실패율을 완화하고, 신뢰할 수 있는 에이전트 시스템을 구축하기 위한 새로운 접근으로 MADM(Multi Agent Design Methodology)을 제안한다.

MADM은 분석, 설계, 구현의 세 단계를 중심으로 이루어지며, 설계 단계에서의 불확실성과 구조적 혼선을 줄이는 데 초점을 둔다. 본 연구의 범위는 3~10개의 에이전트로 구성된 중소규모의 MAS로 한정한다. 대상 시스템은 LLM 기반 에이전트들이 외부 도구를 활용해 다단계의 복합 작업을 수행하는 경우에 적용한다.

그림 2. MADM 방법론의 3단계
Fig. 2. Three Stages of the MADM Methodology

Ⅱ. 관련 연구 및 문헌 고찰

2-1 AI Agent

에이전트(Agent)는 언어학에서 '행동의 주체'를 의미하며, 능동성, 의도성, 자율적 행동이라는 공통된 속성을 가진다. AI 에이전트는 이러한 개념을 바탕으로, 독립적인 의사결정 능력과 능동적인 행동을 통해 목표를 달성하는 완성형 프로그램이라고 정의할 수 있다[13]. 이러한 AI 에이전트는 구조는 일반적으로 LLM, Memory, Tool, Autonomy로 구성된다. LLM은 자연어 이해와 생성, 추론과 계획 수립을 담당하는 인지 엔진이다. Memory는 대화 맥락 유지와 과거 경험 축적을 통한 학습을 가능하게 하는 기억 체계이다. Tool은 외부 환경과 상호작용하고 실시간 정보를 획득하거나 특정 작업을 실행하는 수단이다. Autonomy는 주어진 목표 달성을 위해 스스로 계획을 수립하고 도구를 선택하는 의사결정 메커니즘이다[13]. 이러한 에이전트들이 유기적으로 상호작용을 하며 협력하는 구조가 MAS이다. MAS는 단일 에이전트가 수행하기 어려운 복잡하고 대규모의 문제를 분산적으로 처리할 수 있다는 점에서 의미가 크다. 즉, 각 에이전트가 독립적으로 판단하되, 전체 시스템의 목표 달성을 위해 협업하는 지능적 집단 행위의 틀이라고 할 수 있다.

2-2 MAS 개발의 과제와 실패 분석

이론적으로 MAS는 복잡하고 대규모 작업에 대하여 작업 분해, 특화된 역할 분담 등의 특징을 가지고 있기 때문에 단일 에이전트보다 효율적이고 안정적인 성능을 보일 것으로 기대된다. 그러나 UC Berkeley 연구팀에 따르면, 최신 오픈소스 MAS 프레임워크 기반의 프로젝트에서 각 프레임워크별로 41%에서 86.7%에 이르는 실패율이 보고되었다[2]. 이러한 높은 실패율은 단순한 코드 오류나 모델의 한계 때문이 아니라, 시스템 설계와 협업 구조 자체에 내재된 복합적인 문제에서 비롯된 것으로 분석된다.

이를 규명하기 위해 UC Berkeley 연구팀은 1,600개 이상의 MAS 실행 로그를 근거 이론 방법론을 통해 분석하고, 그 결과 다중 에이전트 시스템 실패 분류 체계(MAST; Multi Agent System Failure Taxonomy)를 제안했다. MAST는 MAS의 실패 원인을 설계 오류(System Design Issues), 역

할 오류(Inter-Agent Misalignment), 산출물 오류(Task Verification Failure)의 세 가지 주요 범주로 구분한다. 분석 결과, 설계 오류가 44.2%, 역할 오류가 32.3%, 산출물 오류가 23.5%를 차지하는 것으로 나타났다. 이러한 결과는 MAS 개발 실패의 근본적인 원인이 어디에 집중되어 있는지 구체적으로 보여준다.

2-3 전통적 소프트웨어 개발 방법론

전통적인 소프트웨어 개발 방법론은 Waterfall, Agile, Spiral, Prototype 모델 등이 대표적으로 있다. 이들은 세부적인 절차에는 차이가 있지만, 기본적으로 요구사항 분석, 설계, 구현, 테스트, 배포의 단계로 진행된다. 이러한 프로세스는 규칙 기반 구조와 고정된 UI/UX 설계를 진행하는 결정론적 시스템에 적합하다. 하지만 MAS는 근본적으로 다른 특성을 갖는다[13]. MAS의 핵심은 자율성과 비결정성이다. 사용자의 자연어 명령을 해석해 스스로 계획을 세우고, 미리 정의되지 않은 방식으로 문제를 해결한다. 이 때문에 목표와 실행 방식이 유동적으로 변화하며, 기존 개발 방법론이 전제하는 '고정된 요구사항'이나 '예측 가능한 결과물'의 개념이 적용되기 어렵다. 결국, 기존의 정적 설계 중심 방법론은 동적으로 변화하는 목표와 상호작용을 다루기에는 한계가 있다.

2-4 기존 MAS 개발 프레임워크

MAS 개발에는 다양한 프레임워크가 활용되고 있다. 일반적으로 MAS 프레임워크는 'LLM 기반 에이전트를 설계, 개발, 배포, 관리할 수 있도록 돕는 SDK(Software Development Kit) 또는 라이브러리'를 의미한다[13]. 이러한 프레임워크는 설계 철학에 따라 네 가지 유형으로 구분할 수 있다. 오케스트레이션 중심(LangGraph, Semantic Kernel 등)은 그래프 기반의 워크플로와 상태 관리에 강점을 보인다. 역할 기반 협업(AutoGen, CrewAI, MetaGPT)은 명확한 역할 분담과 대화형 협업을 강조한다. 경량성과 투명성 중심(SmolAgents, PydanticAI, Agno)은 단순성과 투명성을 우선시한다. 마지막으로, 데이터 및 엔터프라이즈 중심(LlamaIndex, Google ADK)은 대규모 데이터 처리와 분산 시스템에 적합하다. 철학과 기능은 다르지만, 대부분의 MAS 프레임워크는 공통적으로 LLM, Tool, Memory, Guardrails라는 핵심 구성요소를 가진다[6]. 그러나 이러한 프레임워크들은 구현과 배포에서는 유용하지만, 설계 단계의 체계적인 지원은 부족하다. 역할 정의, 자율성 수준 설정, 협업 메커니즘 선택 등 핵심 의사결정이 개발자의 재량에 의존할 수밖에 없으며, 아키텍처의 투명성과 표준화 또한 미흡한 수준이다[13].

2-5 에이전트 자율성(Autonomy) 수준

AI 에이전트의 자율성은 사용자의 개입 없이 작동하도록 설계된 정도를 의미한다. 비슷한 개념으로 종종 혼동되는 에이전시(Agency)는, 특정 목적 달성을 위해 에이전트가 스스로 도구를 선택하고 행동을 수행하는 능력을 의미한다. 두 개념은 밀접하지만 독립적이다. 예를 들어, 많은 기능을 수행하고 도구를 호출할 수 있는 에이전트라도 단계마다 사용자의 피드백을 요청하도록 설계하면 낮은 자율성을 가진다.

에이전트의 자율성은 사용자와 에이전트 간의 상호작용 패턴을 기준으로 다섯 가지 단계로 구분된다[7]. 1단계부터 5단계까지 L1 ~ L5로 표기할 수 있고, L1은 가장 낮은 수준의 자율성을 나타내며, 아래 단계로 내려갈수록 자율성의 수준이 높아진다. 1단계(운영자, Operator)는 사용자가 모든 결정을 내리고, 에이전트는 단순히 지시에 따라 행동한다. 2단계(협력자, Collaborator)는 사용자와 에이전트가 공동으로 계획하고 실행하지만, 주도권은 사용자에게 있다. 3단계(컨설턴트, Consultant)는 에이전트가 주도적으로 작업을 수행하고, 사용자는 전문지식이나 선호도를 입력하는 역할을 맡는다. 4단계(승인자, Approver)는 에이전트가 대부분의 업무를 스스로 처리하지만, 중요한 결정은 사용자에게 승인을 요청한다. 5단계(관찰자, Observer)는 에이전트가 모든 계획과 실행을 주도하고, 사용자는 단순히 결과를 모니터링하거나 필요시 개입하는 수준이다[7]. 자율성 수준을 어떻게 설정하느냐는 단순한 기술적 선택이 아니라 유용성, 효율성, 책임성 등 다양한 요소 간의 균형을 결정짓는 문제다. 따라서 개발자는 시스템의 사용 목적과 대상 사용자 경험에 맞춰 적절한 자율성 수준을 신중히 설계할 필요가 있다.

2-6 MAS의 역할 조정 및 계획

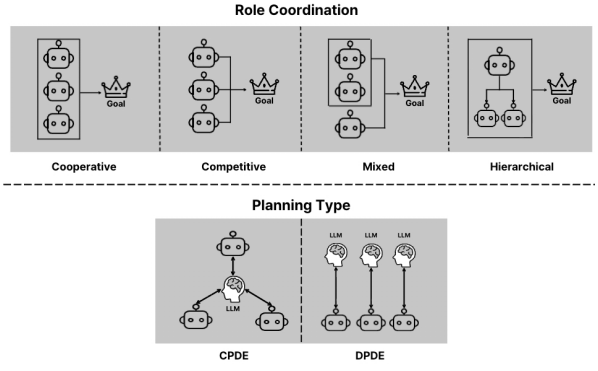

그림 2. MAS에서 역할 조정 및 계획 유형

Fig. 2. Role Coordination Types and Planning Structures in Multi-Agent Systems

MAS의 조직구조는 Fig 2와 같이 에이전트 간 관계 유형과 계획 수립 방식에 따라 다양하게 구분된다[8].

에이전트 간 관계 유형은 협력형, 경쟁형, 혼합형, 계층형으로 나눌 수 있다. 협력형(Cooperative)은 모든 에이전트가 동일한 목표를 공유하며, 문제를 세분화하여 병렬로 처리한

다. 경쟁형(Competitive)은 상반된 목표나 제한된 자원을 두고 경쟁하며 빠른 의사결정을 유도한다. 혼합형(Mixed)은 협력과 경쟁이 동시에 존재하는 형태로, 실제 사회적 상호작용을 반영한다는 점에서 현실성이 높지만, 설계의 난도가 높다. 계층형(Hierarchical)은 상위 에이전트가 문제를 분할하고 하위 에이전트에 작업을 분배하는 구조로, 명확한 통제 체계를 유지할 수 있다. 계획 수립 방식에 따라서도 구분된다. CPDE(Centralized Planning, Decentralized Execution)은 중앙 노드가 전체 계획을 수립하고 각 에이전트가 이를 개별적으로 실행하는 구조로, 시스템 성능 향상에는 유리하지만, 중앙 노드의 과부하 위험이 존재한다. DPDE(Decentralized Planning, Decentralized Execution)는 각 에이전트가 자체적으로 계획을 수립하고 협상을 통해 조율하는 구조로, 유연성이 뛰어나지는 통신 비용과 복잡도가 증가한다[8].

2-7 책임할당 매트릭스(RACI)

책임할당 매트릭스(RACI; Responsible, Accountable, Consulted, Informed)는 프로젝트 내에 각 구성원이 맡을 역할과 책임, 권한 수준을 명확히 정의하기 위한 관리 도구이다. 이 매트릭스는 프로젝트 관리 영역에서 팀 간 의사소통을 용이하게 하고 역할 분담을 명확히 하는 데 널리 활용된다[9].

RACI의 네 가지 역할은 다음과 같다. Responsible(R)은 작업을 실제로 수행하는 실행 책임자 역할을 수행한다. Accountable(A)은 최종 의사결정권자로, 할당된 모든 작업에 대한 궁극적인 책임을 지는 책임자이다. 각 활동에 대해서 R은 다수일 수 있지만, A는 반드시 한 명만 지정되어야 한다. Consulted(C)는 전문가 역할로, 해당 분야에 대한 전문 지식을 바탕으로 정보를 제공하는 자문 역할을 수행한다. Informed(I)는 정보 공유의 대상자로, 진행 중인 프로젝트에 대한 상황과 결정 사항을 통보받는 이해관계자이다.

2-7 AI 에이전트 메모리

메모리는 LLM과 AI 에이전트를 구별하는 핵심 요소로, 대화 맥락을 유지하고 사용자 선호나 과거 상호작용을 기억함으로써 개인화된 응답과 일관된 행동을 가능하게 한다[16]. 최근 연구에서는 LLM이 멀티 턴 대화에서 초기 가정을 수정하지 못하고 대화 흐름을 놓치는 'Lost in Conversation' 현상을 보이며, 단일 턴 대비 정확도가 39% 하락하고 신뢰성이 112% 저하된다고 보고하였다[10]. 이는 누적된 정보와 맥락을 통합하거나 관리하지 못하는 구조적 한계에서 비롯된다.

① 메모리 구조: 메모리는 정보의 지속성과 범위에 따라 단기 메모리(STM; Short Term Memory)와 장기 메모리(LTM; Long Term Memory)로 구분된다[13]. STM은 현재 세션의 대화 히스토리와 임시 작업 상태를, LTM은 사용자의 선호도나 과거 경험 등을 저장한다.

② 메모리 활용 전략: 선제적 메모리 활용은 사용자 입력 전에 메모리 검색을 수행하여 일관된 사용자 경험이 중요한 시스템에 적합하다. 선택적 메모리 활용은 필요한 경우에만 관련 메모리를 검색하여 성능 최적화가 중요한 시스템에 적합하다. 도구로서의 메모리 활용은 메모리 검색기를 에이전트가 호출할 수 있는 도구로 사용하며, 에이전트 자율성이 높은 시스템에 적합하다[15].

③ 메모리 접근 권한: 접근 가능한 데이터의 범위, 보안 수준, 정보 민감도를 고려해 정의된다. 이는 특히 프라이버시 보호와 데이터 윤리 측면에서 중요하다.

④ 메모리 프레임워크: 에이전트의 메모리를 구현하기 위한 기술적 도구인 메모리 프레임워크는 VectorDB, RAG, MemoryBank 등이 있다. 메모리 프레임워크를 선정할 때는 메모리 용량, 보존 기간, 정확도, 속도, 관리 용이성을 고려하고, 도메인 특성과 보안 요구사항에 따라 선택해야 한다.

Ⅲ. Muti Agent Design 방법론

MADM(Multi-Agent Design Methodology)은 AI 에이전트 시스템을 위한 설계 중심 방법론이다. 기존 연구들이 구현 기술이나 운용 도구에 초점을 맞추었지만, MADM은 MAS 개발 과정에서 빈번히 발생하는 설계 단계 실패 요인을 해소하는 데 초점을 둔다. MADM은 환경분석(Environmental Analysis), AI 에이전트 설계(AI Agent Design), 멀티 에이전트 시스템 구현(Multi-Agent System Implementation)의 세 단계(Phase)로 구성되며, 각 단계는 여러 개의 세부 활동(Activity)으로 세분화한다.

MADM은 네 가지 원칙에 기반한다. 첫째, 설계 우선 접근으로, MAS 실패의 약 76.5%가 초기 설계 미흡에서 비롯되므로[2] 설계 프로세스의 체계화를 중심에 둔다. 둘째, RACID 모델을 통한 역할 기반 협업 설계로 역할 중복이나 충돌을 방지한다. 셋째, 하향 설계와 상향 구현으로 오류를 조기에 발견하고 수정한다. 넷째, 계층 구조 설계로 상위 에이전트는 목표 설정과 방향 조정을, 하위 에이전트는 작업 실행을 담당한다. 이러한 원칙은 MAS 개발의 복잡성을 줄이고, 설계 단계에서의 불확실성을 최소화하기 위한 MADM의 기본 철학을 반영한다.

3-1 환경분석(Environmental Analysis)

환경 분석 단계는 문제 도메인의 특성을 파악하고 시스템의 요구사항을 명확히 정의하는 단계이다. 현재 상태 분석, 문제 도메인 정의, 핵심 업무 영역 식별, 요구사항 도출의 네 가지 활동으로 구성된다.

그림 3. 환경분석의 세부 활동 프로세스
Fig. 3. Workflow of the Environmental Analysis Stage

1) 현재 상태 분석(As-Is Analysis)

현재 상태 분석에서는 MAS가 해결해야 할 문제의 본질과 목표를 명확히 한다. 기존 시스템의 구조와 한계를 파악하고, 자동화나 지능화가 필요한 지점을 확인한다. 사용자 인터뷰, 기존 시스템 분석, 도메인 전문가의 자문 등을 통해 현재 상태를 다각도로 진단하며, MAS 도입을 통해 기대하는 효과를 정량적, 정성적 평가 지표를 설정한다.

2) 문제 영역 정의

MAS의 효과적인 설계는 문제 도메인을 명확히 정의하는 것에서 시작한다. 본 연구에서는 문제 명확성(Problem Clarity)과 방법 명확성(Method Clarity)을 축으로 하는 이차원 분류 매트릭스를 제안한다. 문제 명확성은 목표와 현재 상태 간 격차의 명확성을 의미하고, 방법 명확성은 문제 해결을 위한 절차적 지식의 구체화 정도를 의미한다[14]. 본 연구에서는 두 축을 기준으로 에이전트를 네 가지 유형으로 분류한다. 먼저 실행형 에이전트는 문제와 해결 방법이 모두 명확한 상황에서 반복적이거나 규칙 기반의 작업을 수행하는 데 적합하다. 반면 탐색형 에이전트는 문제는 명확하지만, 해결 방법이 불분명한 경우, 시행착오를 통해 최적의 접근법을 찾아간다. 정의형 에이전트는 문제가 모호하지만, 해결 방법이 분명한 경우에, 문제를 재정의하고 구체화하는 역할을 수행한다. 마지막으로 상호 작용형 에이전트는 문제와 방법 모두 불확실한 상황에서 여러 에이전트가 협력하여 해결 방향을 모색하는 유형이다.

3) 핵심 기능 영역 식별 및 우선순위화

핵심 기능의 중요도와 기술적 실현 가능성을 종합적으로 평가한다. 2 x 2 매트릭스를 통해 기능 영역을 분류하고 우선순위를 결정하며, 높은 우선순위 기능은 초기 단계에서 구현하고 부가 기능은 점진적 개발 전략을 수립한다.

표 1. 중요도와 실현 가능성 매트릭스
Table 1. Importance and Feasibility Matrix

Importance / Feasibility	High Feasibility	Low Feasibility
High Importance		
Low Importance		

4) 요구사항 도출

MAS는 다중 에이전트 간의 상호작용과 역할 분담까지 함께 고려해야 한다. 이해관계자를 전문가, 일반 사용자, 관리자로 분류하고 직접 사용자와 간접 영향자로 매핑하여 시스템에 요구되는 기능적·비기능적 요구사항을 도출한다.

요구사항은 두 가지 관점에서 도출한다. 비효율이나 오류 발생 지점을 식별하는 문제 중심 접근법, 실제 사용자의 관점에서 시스템의 흐름과 요구하는 기대를 파악하는 사용자 중심 접근법이다. 이를 바탕으로 사용자 시나리오를 구성하고, AI 에이전트가 기억해야 할 정보의 종류와 지속성을 가져야 하는 요구를 명시한 메모리 요구사항을 정의하여 설계 단계의 기반을 마련한다.

3-2 AI 에이전트 설계(Design)

AI 에이전트 설계 단계는 환경 분석을 통해 정의된 요구사항을 바탕으로, 실제 작동 가능한 MAS의 구조적 청사진을 구체화하는 과정이다. 이 단계에서는 기능 단위의 분해, 역할 체계 설계, 에이전트 명세화, 메모리 구조 설계, 관계 모델링, 그리고 시스템 다이어그램 작성까지 일련의 과정을 거친다.

그림 4. AI 에이전트 설계 단계의 세부 활동 프로세스
Fig. 4. Workflow of the AI Agent Design Stage

1) 기능 분석 및 에이전트 배정

사용자 시나리오와 요구사항을 기반으로 전체 프로세스를 작업 단위로 세분화한다. 각 작업은 IPO(Input-Process-Output)의 관점에서 정의하고 논리적 순서를 명확히 한다. 세분화된 기능은 단일 책임 원칙에 따라 그룹화되며, 에이전트는 역할과 책임 범위에 따라 계층 구조로 설계된다. 역할 구분을 명확히 하기 위해, 상위 수준의 주요 기능 영역을 최상위 레벨로 두고 점차 하위 기능 군과 세부 기능으로 분해한다. 본 연구는 이를 네 단계의 개념적 레벨로 제시한다. 레벨 0(조정형, Orchestrator)은 시스템 전체의 목표를 설정하고 대규모 워크플로를 조정하는 최상위 에이전트이다. 레벨 1(관리형, Coordinator)은 특정 도메인이나 기능 영역을 관리하는 중간 레벨의 에이전트이다. 레벨 2(처리형, Executor)는 도메인별 전문 지식을 바탕으로 구체적인 작업을 실행하는 에이전트이다. 레벨 3(행동형, Operator)은 세부적인 도구 사용이나 단순 반복 작업을 수행하는 최하위의 레벨 에이전트이다. 이 구조는 시스템의 복잡성과 규모에 따라 레벨을 유연하게 확장하거나 축소할 수 있다.

2) RACID 모델 기반의 역할 체계 설계

본 연구에서는 MAS를 하나의 조직(Organization)으로, 각 AI 에이전트를 개별 구성원(Team member)으로 간주한다. 이에 따라 전통적인 RACI 매트릭스를 MAS의 특성에 맞게 확장한 RACID 모델을 제안한다. 이 모델은 에이전트의 자율성과 동적 역할 변동성을 반영하여, 시스템 내 모든 에이전트의 책임과 권한을 명확히 규정한다[9].

역할 수행 여부는 다음과 같이 표기한다. ● (기본 역할), ○(상황별 적응 가능), △(조건부), -(역할 없음).

RACID 매트릭스는 작성 시 다음 사항을 검증해야 한다. 먼저, 각 주요 작업에 대해 반드시 하나의 A 역할만 지정되어야 한다. 또, 모든 작업에 하나 이상의 R 역할이 배정되어야 한다. 세 번째, D 역할은 일반적으로 레벨 0, 레벨 1에 부여한다. 이렇게 완성된 RACID 매트릭스는 다음 단계인 에이전트 상세 명세와 관계 모델 설계의 기반 자료로 활용된다.

표 2. RACID 역할 정의
Table 2. RACID Role Definition

Role	Name	Definition
R	Responsible	Performs the assigned task and generates outputs by invoking tools or processing data. Multiple R roles may work sequentially or in parallel.
A	Accountable	Holds final authority and ensures task quality. Reviews R outputs and may request revisions. Only one A per task, typically a top-level orchestrator.
C	Consulted	Provides domain expertise or information to support decisions. Does not execute tasks directly but assists through knowledge retrieval or APIs.
I	Informed	Receives progress and result updates without direct task involvement. Mainly monitors, logs, or interfaces with users.
D	Dynamic Coordinator	Manages task reassignment and workflow changes during exceptions, ensuring adaptability and system resilience. Usually a Level 0 orchestrator.

표 3. RACID 매트릭스
Table 3. RACID Matrix

Agent ID	Name	Lv	R	A	C	I	D	Autonomy Level	Notes
A001	Orchestrator	0	-	●	○	●	●	L5	System coordinator
A001	Collaborator	1	●	-	●	○	△	L4	Domain oversight
A001	Consultant	2	●	-	○	-	-	L2	Task execution
…	…	…	…	…	…	…	…	…	…

3) AI 에이전트 상세 명세

본 연구는 각 AI 에이전트의 행동 원칙(Persona)과 기술적 능력(Capability)을 중심으로 설계된다. 페르소나는 에이전트의 전문 영역, 의사결정 방식, 대화 스타일 등을 포함한 행동 지침으로, LLM 기반 에이전트의 응답 일관성과 신뢰성에 직접적인 영향을 미친다[11]. 본 연구는 다중 페르소나(Multi-Persona) 접근을 적용해, 각 에이전트가 맡은 역할에 따라 고유한 판단 기준과 성격적 특성을 갖도록 설계했다. 페르소나는 RACID 역할과 긴밀히 연계되며, 사용자 경험의 일관성을 유지하도록 조정된다.

능력 명세(Capability Specification)는 에이전트가 역할을 수행하는 데 필요한 기능을 정의하는 과정이다. 핵심 기능은 RACID 역할과 직접 연결되고, 보조 기능은 이를 지원한다. 각 기능은 IPO 흐름을 따르며, JSON 스키마로 명세 되어 구현 시 참조 기준으로 활용된다.

4) 메모리 아키텍처 설계

메모리는 'Lost in Conversation' 문제를 극복하기 위한 핵심 단계이다. 메모리 설계는 구조 정의, 활용 전략, 접근 권한 설정의 세 측면으로 이루어진다. 먼저, 메모리는 정보 지속성에 따라 단기(STM)와 장기(LTM)로 구분되며, 각각 저장할 데이터의 종류를 명확히 지정해야 한다[10]. 다음으로, 메모리 활용 전략은 시스템의 특성에 따라 선제적, 선택적, 또는 도구 기반 방식 중 적합한 접근을 선택한다[15]. 마지막으로, 메모리 접근 권한은 데이터의 민감도와 보안 요구 수준을 고려하여 에이전트별 접근 범위를 설정한다.

메모리 아키텍처 단계의 주요 산출물은 메모리 스키마 명세서, 활용 전략 정의서, 그리고 에이전트별 접근 매트릭스로, 이는 구현 단계에서 저장소 선정과 코드 구현의 기준 자료로 활용된다.

5) 관계모델 설계

관계 모델은 AI 에이전트 간 상호작용 방식을 정의한다. 즉, 에이전트들이 어떤 방식으로 협력하고 정보를 교환하며, 의사결정과 조율을 수행하는지를 규정한다[8]. 에이전트 관계는 계층 레벨에 따라 크게 두 가지로 구분된다. 수직적 관계는 서로 다른 계층 레벨 사이에 존재하게 되고, 통제와 위임에 기반한 계층형 관계이다. 수평적 관계는 동일한 계층 레벨의 에이전트들 간에 발생한다. 이 관계에서는 협력형, 경쟁형, 혼합형의 관계가 모두 가능하다. 관계 모델링 결과물은 에이전트 간 상호작용 다이어그램 형태로 표현되며, 전체 시스템 내의 통신 흐름을 시각적으로 명확히 보여준다.

6) 시스템 아키텍처 다이어그램

에이전트 설계 단계를 마치면 시스템 아키텍처 다이어그램을 작성한다. 다이어그램은 개발자와 이해관계자 간 의사소통을 원활하게 하고 시스템 구조를 직관적으로 이해할 수 있게 한다. 다이어그램의 포함 요소는 에이전트 목록 및 계층 레벨, 에이전트 간 역할 조정, 정보 흐름의 방향이다. 본 설계 단계에서 생성되는 핵심 산출물은 기능 명세서, RACID 매트릭스, 페르소나 및 능력 명세서, 메모리 스키마 명세서, MAS 설계 다이어그램이다. 이렇게 생성된 산출물은 구현 단계의 기준 문서로 활용되며, MAS의 일관성, 재현 가능성, 품질 확보를 위한 핵심 자료가 된다.

3-3 멀티 에이전트 시스템 구현(MAS Implementatin)

구현 단계는 설계 단계에서 정의된 청사진을 바탕으로 실제 작동하는 MAS를 구축하는 과정이다. MADM은 AI 에이전트의 네 가지 핵심 구성요소 (LLM, Memory, Tool, Autonomy)를 중심으로 구현된다.

1) 에이전트 프레임워크 선정

MAS 구현의 첫 단계는 설계 요구사항에 부합하는 에이전트 프레임워크를 선택하는 일이다. 프레임워크는 에이전트의 생성, 실행, 통신을 지원하는 기반 구조로, 전체 시스템의 확장성과 안정성을 좌우한다. 프레임워크 선정 시에는 다음의 기준을 종합적으로 고려한다.

① 시스템의 목표와 정합성을 평가: 빠른 프로토타이핑과 검증이 목표인지, 엔터프라이즈급의 안정성과 확장성이 중요한지에 따라 적합한 프레임워크가 달라진다[13].

② 기술 생태계 및 인력 확보 용이성을 검토: 기존 인프라

(클라운, 데이터베이스, 인증 시스템)와의 호환성, 개발 인력 확보 용이성, 커뮤니티 활성화 수준을 함께 검토해야 한다.

③ 라이선스 조건 및 비용 구조 검토: 오픈소스인지, 상업적 이용에 제약이 있는지, 유료 API 제한이 있는지 등을 확인해야 한다.

④ 유지 보수 및 발전 가능성을 평가

대표적인 에이전트 프레임워크로는 LangChain, AutoGen, CrewAI, LangGraph 등이 있다.

2) 에이전트 시스템 구현

이 단계에서는 설계 명세서를 바탕으로 선택된 프레임워크 내에서 에이전트를 실제로 구현한다. 에이전트는 LLM, Memory, Tool, Autonomy의 네 가지 핵심 구성요소로 구현된다.

① LLM: 에이전트의 핵심은 언어 모델 선택이다. LLM 모델은 추론력, 속도, 비용, 안정성 등에서 큰 차이를 보이므로,

작업 목적과 성격에 맞는 모델을 선택해야 한다. 고난도 의사결정이나 다단계 추론이 필요한 경우에는 GPT-4나 Claude Opus 등과 같은 고성능 모델이 적합하고, 실시간성과 비용 효율이 중요한 환경에서는 Gemini Flash, Claude Haiku, GPT-3.5 등의 경량 모델이 적합하다.

또한, 작업 유형에 따라 CoT(Chain-of-Thought), ToT(Tree-of-Thought), ReAct 등의 추론 프레임워크를 적용해 효율을 높일 수 있다.

② Memory: 메모리 스키마 명세서와 활용 전략 정의서, 접근 매트릭스를 기반으로 구현한다. VectorDB, RAG, MemoryBank 등 적합한 메모리 프레임워크를 선정하고, STM과 LTM 요구사항에 맞춰 데이터 구조를 실제 저장소에 구현한다. 또한 선택된 활용 전략을 코드로 구현하며, 각 에이전트별 접근 권한을 설정한다.

③ Tool: 도구는 LLM이 수행할 수 있는 범위를 "생성"에서 "실행과 상호작용"으로 확장시키는 핵심 구성요소이다.기능적 목적에 따라 Extensions, Functions, Data Stores의 세 가지 유형으로 나뉜다. 각각은 외부 API 호출, 실시간 데이터 접근, 내부 DB 통합 등 다양한 역할을 수행한다. Tool을 선택할 때는 보안 수준, 응답 속도, 외부 시스템 연동성을 종합적으로 고려해야 한다[17].

④ Autonomy: 자율성 구현은 RACID 매트릭스에 정의된 자율성 단계(1~5)에 따라 달라진다. 낮은 자율성 단계(1~2)에서는 사용자 명령에 따라 단일 작업을 수행하는 간단한 루프를 구성하며, 중간 단계(3~4)에서는 중간 수준의 계획 수립과 사용자 확인을 포함한 의사결정이 가능하도록 구성한다. 높은자율성 단계(4~5)에서는 Self-Refine, Reflection 등으로 출력의 품질을 반복적으로 개선하거나 실패 경험에서 학습할 수 있도록 구현한다[7].

3) 에이전트 간 통신 및 오케스트레이션

MAS의 핵심은 에이전트 간 협력과 정보 교환의 효율성이다. 이를 위해 적절한 통신 프로토콜과 오케스트레이션 구조가 필요하다. 관계 모델 정의서와 시스템 아키텍처 다이어그램은 이 단계의 기본 참조 자료로 활용된다.

① 통신 프로토콜 선정: MAS에서는 에이전트가 외부 시스템 및 다른 에이전트와 상호작용을 하기 위한 표준화된 프로토콜이 필요하다. Context-Oriented Protocols(MCP 등)는 에이전트와 외부 시스템 간 연동에 사용되며, Inter-Agent Protocols(A2A, ACP, ANP 등)은 에이전트 간 통신에 사용된다[12]. 프로토콜 선정 시 에이전트의 자율성 수준, 통신 유연성, 인터페이스 표준화 정도, 보안을 종합적으로 고려해야 한다[13].

② 통신 구조 구현: 설계 단계에서 정의된 관계 유형에 따라 통신 구조를 구현한다. 수직적 관계(계층형)에서는 Lv0 → Lv1 → Lv2 → Lv3 순으로 작업을 분배한다. 수평적 관계에서 협력형의 경우 작업 결과를 공유하고 통합하는 파이프라인을, 경쟁형의 경우 독립적 솔루션 제시 및 최적 결과 선택 메커니즘을, 혼합형의 경우 상황에 따른 동적 전환 로직을 구현한다. RACID 매트릭스에 정의된 역할에 따라 실행-검증-자문-모니터링-조율의 흐름을 코드 수준에서 구체화한다.

③ 오케스트레이션 방식 구현: 설계 단계에서 결정된 계획 수립 방식(CPDE 또는 DPDE)에 따라 구현한다[8].

구현이 완료된 에이전트는 단위 테스트(Unit Test)와 통합 테스트를 거쳐 기능 및 상호작용 정확성을 검증한 뒤,

실제 운영 환경에 배포된다.

Ⅳ. 적용 사례

MADM 방법론의 실효성을 검증하기 위해 일상적인 의사결정 시나리오인 식사 메뉴 선정 시스템을 설계하고 구현하였다. 이 시스템은 사용자의 선호도, 건강 상태, 예산, 시장 상황 등 다양한 제약 조건을 고려하여 최적의 식사 메뉴를 제안하는 MAS이다.

이 도메인을 선택한 이유는 세 가지이다. 첫째, 누구나 경험하는 일상적 문제로 도메인 이해가 용이하며 요구사항 정의가 직관적이다. 둘째, 영양, 선호도, 예산, 문화적 배경 등 복수의 판단 기준이 상충하는 상황에서 MAS의 협업 효과를 명확히 보여줄 수 있다. 셋째, 전체 계획 수립, 세부 영역 관리, 구체적 실행의 계층 구조가 자연스럽게 형성된다. 본 시스템은 7개의 에이전트로 구성되며 4단계 계층 구조를 가진다. 시스템의 최종 목표는 사용자 만족도를 극대화하는 동시에 건강과 경제적 제약을 준수하는 균형 잡힌 식사 메뉴를 제공하는 것이다.

4-1 환경 분석 적용

1) 현재 상태 분석

일상에서 식사 메뉴를 결정할 때 사용자는 여러 가지 문제에 직면한다. 대표적으로는 정보 과부하, 건강 상태 반영의 부재, 영양, 예산, 선호도를 동시에 고려해야 하는 복합적 판단의 어려움이 있다. 기존의 검색 기반 서비스는 단순 키워드

매칭에 의존해, 개인화된 추천이나 제약 조건 반영이 어렵다.

이에 본 연구는 MAS 도입을 통해 건강, 선호도, 예산을 통합적으로 고려한 맞춤형 의사결정 시스템을 구현하였다.

시스템 성능 평가는 다음 기준에 따라 수행된다.

표 4. 적용 사례: 시스템 평가 기준
Table 4. Applied Case: System Evaluation Criteria

Evaluation Criteria	Target Value	Measurement Method
Initial Accuracy	70% or higher user satisfaction	Pre/post survey on actual meal selection
Response Time	Within 3 seconds	Measure results delivery time
Constraint Compliance	95% or higher	Verification of health/budget constraint compliance
User Satisfaction	4.0 or higher on a 5-point scale	Survey

2) 문제 영역 정의

문제 영역을 문제 명확성과 방법 명확성의 두 축으로 분석한 결과, "영양, 선호도, 예산을 모두 만족하는 식사 메뉴 추천"은 목표가 뚜렷하므로 높은 문제 명확성을 가진다.

반면, 상충하는 제약 조건 내에서 최적 조합을 찾아야 하기 때문에 탐색형 문제(Exploratory type) 로 분류된다.

3) 핵심 기능 영역 식별 및 우선순위화

핵심 기능을 중요도와 실현 가능성을 기준으로 평가하여 우선순위를 설정하였다.

표 5. 적용 사례: 기능 우선순위 매트릭스
Table 5. Applied Case: Funtional Priority Matrix

Category	High Feasibility	Low Feasibility
High Importance	User preference analysis, health condition consideration, budget and market analysis, menu planning	Inventory linkage, social networking features
Low Importance	Recipe execution, cultural adaptation	Voice interface

4) 요구사항 도출

요구사항은 이해관계자 분석을 통해 도출했다. 직접 사용자는 일반 사용자와 건강 제약(당뇨, 알레르기 등)이 있는 사용자로 구분되며, 전문가 그룹에는 영양사와 요리사 등 도메인 지식 제공자가 포함된다. 간접 영향자로는 시스템 관리자와 외부 데이터 제공자 등이 있다.

표 6. 적용 사례: 기능적 요구사항
Table 6. Applied Case: Funtional Requirments

Requirement ID	Requirement Description	Priority
FR-01	The system shall learn and apply the user's past preference history.	Mandatory
FR-02	The system shall consider the user's health information (e.g., allergies, illnesses).	Mandatory
FR-03	The system shall recommend menus within the predefined budget range.	Mandatory
FR-04	The system shall categorize and provide menus based on cultural types (Korean, Chinese, Japanese).	Recommended
FR-05	The system shall provide detailed recipes for the final selected menu.	Recommended
FR-06	The system shall adjust the menu based on the user's real-time feedback.	Optional

표 7. 적용 사례: 비기능적 요구사항
Table 7. Applied Case: Non-Funtional Requirments

Category	Requirement
Performance	The response time for menu recommendations shall be within 3 seconds.
Reliability	The violation rate of health restriction conditions shall be maintained at 0%.
Usability	The system shall provide an intuitive, dialogue-based interface using natural language.
Scalability	The system shall allow seamless addition of new recipe categories.
Transparency	The system shall explicitly present the rationale behind each recommendation.

표 8. 적용 사례: 메모리 요구사항
Table 8. Applied Case: Memory Requirments

Memory Type	Stored Information	Usage Timing
SMT	Current dialogue context, temporary menu candidates, session feedback	Created/deleted per session
LMT	User profile (health, allergies), past menu selections, budget patterns, feedback records	Persistently stored and auto-retrieved

대표적인 사용자 시나리오는 다음과 같다. 첫 사용자가 접속하면 기본 정보를 수집하고 "저녁 메뉴 추천해 줘"라는 요청에 세 가지 메뉴 옵션을 제시한 후 선택 시 상세 레시피를 제공한다. 당뇨 환자가 "혈당 관리에 좋은 점심 메뉴"를 요청하면 저당 식단을 필터링하고 영양 균형을 맞춘 메뉴를 혈당 지수 정보와 함께 제공한다. 예산 제약이 있는 사용자가 "1만 원 이내로 3명이 먹을 저녁"을 요청하면 시장 가격 정보를 수집하고 가성비 높은 메뉴 조합을 예상 총비용 및 재료 구매처 정보와 함께 제공한다.

4-2 AI 에이전트 설계 적용

1) 기능 분석 및 에이전트 배정

시스템 기능을 일곱 가지 작업으로 분해하고, 단일 책임 원

칙에 따라 4단계 계층 구조로 배정했다. 시스템은 Lv0의 조정자 1개, Lv1의 계획자 1개, Lv2의 분석자 2개, Lv3의 레시피 생성자 6개로 구성된다.

표 9. 적용 사례: 에이전트 배정 명세서
Table 8. Applied Case: Agent Assignment Specification

Agent ID	Agent name	Parent Agent	Level	Core Role Priority
A001	System Orchestrator	-	Lv0	Essential
A101	Menu Planner & Mediator	A001	Lv1	Essential
A201	User Preference & Health Analyzer	A101	Lv2	Essential
A202	Budget & Market Analyzer	A101	Lv2	Essential
A301	Korean Chef(Health-Aware)	A201	Lv3	Recommended
A302	Japanese Chef(Health-Aware)	A201	Lv3	Recommended
A303	Chinese Chef(Health-Aware)	A202	Lv3	Recommended
A311	Korean Chef (Budget-Aware)	A202	Lv3	Recommended
A312	Japanese Chef (Budget-Aware)	A202	Lv3	Recommended
A313	Chinese Chef (Budget-Aware)	A202	Lv3	Recommended

2) RACID 모델 기반 역할 체계 설계

RACID 모델을 적용해 책임 분담을 명확히 했다. 자율성 수준은 각 에이전트의 의사결정 복잡도에 따라 독립적으로 설정하였다.

표 10. 적용 사례: RACID 매트릭스
Table 10. Applied Case: RACID Matrix

Agent ID	Name	Lv	R	A	C	I	D	Autonomy	
								Level	Rationale
A001	System Orchestrator	0	-	●	○	●	●	L4	Oversees the overall workflow and arbitration
A101	Menu Planner & Mediator	1	●	-	●	○	○	L3	Generates and validates menu combinations
A201	User Preference & Health Analyzer	2	●	-	○	-	●	L3	Performs health analysis and risk detection
A202	Budget & Market Analyzer	2	●	-	○	-	●	L3	Attempts budget optimization
A301	Korean Chef	3	●	-	●	-	-	L2	Generates health-aware recipes from templates
A302	Japanese Chef	3	●	-	●	-	-	L2	Generates health-aware recipes from templates
A303	Chinese Chef	3	●	-	●	-	-	L2	Generates health-aware recipes from templates

3) 에이전트 상세 명세

각 에이전트의 기능, 입출력, 자율성 수준, 메모리 접근 권한을 명세했다. A001은 전체 조정과 최종 의사결정을 담당하고, A101은 메뉴 계획과 중재를, A201/A202는 각각 건강/예산 분석을, A301~A313은 문화권 별 레시피 생성을 담당한다. 대표 예시로 A001의 명세를 제시한다.

표 11. 적용 사례: 에이전트 상세 명세(대표 예시: A00)
Table 11. Applied Case: Agent Specification (Example: A001)

Item	Description
Agent ID	A001
Agent Name	System Orchestrator
Hierarchy Level	Lv0 (Orchestrator)
Core Function	Interprets user requests, delegates tasks, and makes final decisions
Input	User's natural language requests, results from lower-level agents
Output	Task coordination instructions, final menu decision
Autonomy Level	L4 (Self-evaluation and improvement)
Memory Access	Full access to all memory (admin privileges)
Tools Used	None (coordination-only role)

4) 메모리 아키텍처 설계

메모리는 STM과 LTM으로 구분하였으며, 선제적 접근 전략을 기반으로 설계했다. A001과 A201은 세션 시작 시 사용자 프로필과 선호도 이력을 자동으로 로드하고, A202는 필요 시에만 예산 패턴을 조회하는 선택적 방식을 병행했다. 메모리 접근 권한의 경우에는 보안 수준에 따라 단계적으로 차등 적용했다. A001은 전체 시스템을 조율하는 역할 특성상 모든 메모리에 대한 관리자 권한을 가지고, 건강 정보를 다루는 A201은 사용자 프로필과 선호도 이력에 읽기/쓰기 권한을 가지며, 최하위 레벨의 A301~A303/A311~A313 셰프 에이전트는 레시피 검색을 위한 읽기 권한만 가진다.

표 12. 적용 사례: 메모리 아키텍처
Table 12. Applied Case: Memory Architecture

Memory Type	Stored Content	Access Rights	Lifecycle
STM	Current conversation history	Readable by all agents	Deleted at session end
STM	Temporary menu candidate list	Read/Write by A001, A101	Deleted at session end
LTM	User profile	Full access by A001, Read/Write by A201	Permanently retained
LTM	Preference history	Full access by A001, Read/Write by A201	Permanently retained
LTM	Budget pattern	Full access by A001, Read/Write by A202	Permanently retained
LTM	Recipe database	Readable by all agents, Write by admin	Permanently retained

5) 관계 모델 설계

관계 구조를 다음과 같이 다이어그램으로 표현했다.

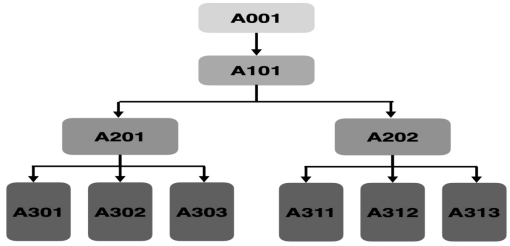

그림 4. 적용 사례: 다중 에이전트 시스템 관계 모델
Fig. 4. Applied Case: MAS Relationship Model

4-3 MAS 구현 적용

1) 에이전트 프레임워크 선정

시스템 요구사항을 바탕으로 주요 멀티 에이전트 시스템 프레임워크를 비교 분석한 결과, Google ADK로 프레임워크를 선정하였다. 선정 배경에는 네 가지 주요 요인이 있다. 첫째, Gemini 모델과의 네이티브 통합을 통해 API 호출이 최적화되고 비용 효율성을 확보할 수 있다. 둘째, 엔터프라이즈급 안정성과 확장성을 제공함으로써 상용 서비스 구현이 가능하다. 셋째, Vertex AI, BigQuery 등 Google Cloud 생태계와의 유기적 연동이 가능하다[17]. 넷째, 계층 구조 표현 및 메모리 관리 측면에서 유연한 커스터마이징이 가능하다.

2) 에이전트 시스템 구현

① LLM: 모든 에이전트에 Gemini 2.5 Flash를 적용하여 일관된 응답 품질과 시스템 간 호환성을 유지하고, 단일 모델 사용으로 운영 복잡도를 낮췄다. 추론 프레임워크는 에이전트별로 차등 적용했다. A001은 ReAct로 동적 상황 판단을 수행하고, A101은 CoT로 단계적 메뉴 조합 추론을 하며, A201/A202는 CoT와 Self-Refine을 결합하여 분석 결과를 재검토한다. A301~A313은 Few-shot Prompting으로 일관된 레시피 형식을 유지한다.

② Memory: STM은 세션 단위로 유지되며 현재 대화 맥락과 임시 작업 상태를 저장한다. LTM은 구조화된 정보(사용자 프로필, 알레르기, 예산 패턴)를 관계형 데이터베이스에, 비구조화된 정보(과거 선호도, 피드백)를 벡터 임베딩 형태로 저장하여 의미 기반 검색을 가능하게 했다. 선제적 접근 방식을 통해 A001은 사용자 요청 전에 프로필을 자동 로드하고, A201은 세션 시작 시 선호도 이력을 사전 검색한다. 접근 권한은 A001(전체 메모리), A201(건강 데이터), A202(예산 정보), A301~A303/A311~A313(레시피 데이터베이스)로 차등 설정했다.

③ Tools: A202는 시장 가격 API를 통해 실시간 가격 정보를 수집하고, A201은 영양 정보 API를 호출하며, A301~A303/A311~A313은 레시피 데이터베이스를 쿼리하도록 했다. 도구 호출은 Google ADK의 Function Calling 기능을 통해 구조화되며, 각 도구는 입력/출력 스키마를 정의했다[17].

④ Autonomy: 자율성은 에이전트별 의사결정 루프 구조로 구현했다. A001(L4)은 하위 에이전트의 작업 결과를 평가하고 재작업을 지시하며 제약 조건 충돌 시 자율적으로 중재안을 도출한다. A101, A201, A202(L3)는 각자의 영역에서 독립적으로 분석과 문제 해결을 시도하되, 처리 범위 초과 시 A001로 에스컬레이션한다. 반면, A301~A303/A311~A313의 에이전트(L2)는 단순 실행 루프 기반으로 설계되어 템플릿 기반의 레시피를 검색하고, 제한적인 범위 내에서만 조정을 수행하도록 제한했다.

3) 에이전트 간 통신 및 오케스트레이션

① 통신 프로토콜 선정: 외부와의 통신을 위해 MCP를 선정했다. 본 시스템은 세 가지 MCP 서버를 활용한다. MCP Filesystem Server는 레시피 데이터베이스와 사용자 프로필 파일 접근에 사용되고, MCP Google Drive Server는 장기 메모리 저장소로 활용되며, MCP Fetch Server는 외부 API 호출(시장 가격, 영양 정보)에 사용한다.

② 오케스트레이션 방식: 이 시스템에서는 CPDE 방식을 채택했다[8]. A001이 전체 계획을 수립하고 작업을 분배하는 구조이기 때문에, 시스템의 일관성 유지와 디버깅 용이성을 위해 CPDE를 선택했다. A001의 오케스트레이션은 다음 흐름으로 진행된다: 사용자 요청 해석 → 메모리 로딩 → 작업 분배 계획 → A201/A202 병렬 실행 → 제약 조건 충돌 감지 → 중재 프로세스 → A101 메뉴 생성 → 셰프 에이전트 선택 → 결과 제시.

③ 중재 메커니즘: 예를 들어, 식사메뉴 추천 시 A201(건강)과 A202(예산의 제약)가 충돌하는 상황이라면, A001이 양측의 분석 결과를 종합해 이를 중재하여 결과를 도출한다.

4-4 구현 결과

표 13. 적용 사례: 시스템 평가 결과
Table 14. Applied Case: Systm Evaluation Results

Evaluation Criteria	Target Value	Measured Result	Achievement	Measurement Method
Initial Accuracy	70% or higher user satisfaction	82.0%	O	Pre/post survey on actual meal selection
Response Time	Within 3 seconds	15 seconds	X	Measure results delivery time
Constraint Compliance	95% or higher	96.0%	O	Verification of health/budget constraint compliance
User Satisfaction	4.0 or higher on a 5-point scale	4.2 points	O	Survey

MADM 방법론을 적용한 식사 메뉴 추천 시스템의 성능을 평가하였다. 5명의 사용자가 각각 10회씩 메뉴 추천을 요청

하여 총 50회의 테스트를 수행하였다. 평가 결과, 정확도는 82.0%로 목표치인 70%를 초과하였으며, 제약 조건 준수율은 96.0%로 목표인 95%를 넘었다. 사용자 만족도는 평균 4.2점으로 목표 수준과 유사하게 나타났다.

반면, 응답 시간은 평균 15초로 목표(3초)를 초과하였다.

이는 다수의 에이전트 간 순차 통신과 외부 API 호출 과정에서의 지연 때문으로 분석된다. 향후에는 병렬 처리 및 캐싱 전략을 적용하여 응답 속도를 개선할 필요가 있다.

V. 결 론

본 연구는 MAS 개발의 높은 실패율을 해결하기 위해 설계 중심의 방법론인 MADM을 제안했다. 기존 연구에서 MAS 실패의 약 44.2%가 초기 설계 문제에서 기존 연구에 따르면 MAS 실패의 약 44.2%가 초기 설계 단계의 문제에서 비롯되며, 이는 구현 도구나 프레임워크 수준의 접근만으로는 근본적 해결이 어렵다는 점을 시사한다. 따라서 본 연구는 기존의 기술 중심 패러다임에서 벗어나, 분석과 설계 단계를 체계적으로 구조화하는 접근에 초점을 두었다. MADM은 세 단계의 프로세스로 구성된다. 환경 분석 단계에서는 문제 도메인 정의와 요구사항 도출을 통해 설계의 방향을 확립하고, AI 에이전트 설계 단계에서는 RACID 모델을 기반으로 역할과 책임을 구체화하며, 구현 단계에서는 설계 산출물을 실제 시스템으로 구체화한다. 이 세 단계는 상향식 구현이 아닌, 하향식 설계−상향식 검증의 순환 구조를 형성하여 MAS 개발 과정 전반의 일관성을 확보하도록 설계했다.

본 연구의 주요 기여는 다음과 같다. 첫째, 전통적인 RACI 매트릭스를 AI 에이전트의 특성에 맞게 확장한 RACID 모델을 제안했다.이 모델은 D(Dynamic Coordinator) 역할을 추가함으로써 동적인 역할 조정과 자율성 수준 간의 연계를 가능하게 했다. 둘째, 계층적 에이전트 구조를 기본 설계 원칙으로 채택하여 역할 분담을 명확히 하고 시스템의 복잡성을 관리 가능한 수준으로 분해하였다. 셋째, 설계 단계에서 생성되는 산출물(RACID 매트릭스, 에이전트 명세서, 메모리 스키마 등)이 구현 단계의 기준 문서로 활용되어 설계와 구현 간의 일관성을 확보했다.

식사 메뉴 추천 시스템에 MADM을 적용한 결과, 정확도 82.0%, 제약 조건 준수율 96.0%, 사용자 만족도 4.2점을 기록하여, 제안한 방법론의 적용 가능성과 실무적 잠재력을 확인할 수 있었다. 다만, 응답 시간(15초)은 목표치인 3초를 달성하지 못하였으며, 이는 에이전트 간 순차적 통신과 외부 API 호출 과정에서의 지연이 주요 원인으로 분석된다.

본 연구는 MADM을 하나의 사례에 적용하여 그 가능성을 탐색하였다는 점에서 의미가 있으나, 단일 도메인 중심의 제한적 적용이라는 한계를 가진다. 또한 구현 단계에서 제시된 지침이 개념적 수준에 머물러 있어, 실제 산업 환경에서 바로 적용하기에는 구체성이 다소 부족하다.

향후 연구에서는 다양하고 복잡한 도메인과 규모의 MAS 프로젝트에 MADM을 적용하여 그 확장성 및 실효성을 검증할 필요가 있다. 특히, 에이전트의 수와 상호작용의 복잡성이 증가할 때 MADM이 제공하는 계층적 설계 구조와 역할 조정 메커니즘이 얼마나 유효한지를 실증적으로 평가해야 한다.

또한, MAS를 평가하기 위한 표준화된 평가 기준(Metric)의 확립도 필요하다. 현재까지 MAS 성능 평가는 주로 응답 시간이나 정확도 등 정량적 지표에 국한되어 있으나, 향후에는 협업 효율성, 자율성 수준, 의사결정 품질, 시스템 신뢰성 등 다면적 요소를 종합적으로 고려한 평가 프레임워크가 마련되어야 한다. 결론적으로, MADM은 MAS 개발 과정에서 체계적 설계의 중요성을 강조하며, 개발자가 경험이나 직관에 의존하지 않고 명확한 절차와 설계 원칙에 따라 안정적이고 일관된 시스템을 구축할 수 있도록 지원하는 실질적 방법론으로 발전할 가능성을 보여주었다.

참고문헌

[1] M. A. Ferrag, N. Tihanyi, and M. Debbah, &From LLM Reasoning To Autonomous AI Agents: A Comprehensive Review,& arXiv preprint arXiv:2504.19678, 2025. https://doi.org/10.48550/arXiv.2504.19678

[2] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. Zaharia, J. E. Gonzalez, and I. Stoica, &Why Do Multi-Agent LLM Systems Fail?&, arXiv preprint arXiv:2503.13657, Mar. 2025. https://doi.org/10.48550/arXiv.2503.13657

[3] J. Gardner and V. A. Baulin, &Is The &Agent& Paradigm A Limiting Framework For Next-Generation Intelligent Systems?&, arXiv preprint arXiv:2509.10875, 2025. https://doi.org/10.48550/arXiv.2509.10875

[4] B. Atil, S. Aykent, A. Chittams, L. Fu, R. J. Passonneau, E. Radcliffe, G. R. Rajagopal, A. Sloan, T. Tudrej, F. Ture, Z. Wu, L. Xinyu, and B. Baldwin, &Non-Determinism Of &Deterministic& LLM Settings,& arXiv preprint arXiv:2408.04667, 2024. https://doi.org/10.48550/arXiv.2408.04667

[5] H. P. Zou, W.-C. Huang, Y. Wu, Y. Chen, C. Miao, H. Nguyen, Y. Zhou, W. Zhang, L. Fang, L. He, Y. Li, Y. Cao, D. Li, R. Jiang, and P. S. Yu, &LLM-Based Human-Agent Collaboration and Interaction Systems: A Survey,& arXiv preprint arXiv:2505.06120, 2025. https://www.semanticscholar.org/paper/bf14244f64db9a743944123558280294f7cd7e29

[6] H. Derouiche, Z. Brahmi, and H. Mazeni, &Agentic AI Frameworks: Architectures, Protocols, And Design Challenges,& arXiv preprint arXiv:2508.10146, 2025. https://doi.org/10.48550/arXiv.2508.10146

[7] K. J. V. Feng, D. W. McDonald, and A. Y. Zhang, &Levels Of Autonomy For AI Agents,& arXiv preprint arXiv:2506.12469, 2025. https://doi.org/10.48550/arXiv.2506.12469

[8] Y. Cheng, C. Zhang, Z. Zhang, X. Meng, S. Hong, W. Li, Z. Wang, F. Yin, J. Huo, and X. He, &Exploring Large Language Model Based Intelligent Agents: Definitions, Methods, And Prospects,& arXiv preprint arXiv:2401.03428, 2024. https://doi.org/10.48550/arXiv.2401.03428

[9] R. Suhanda and D. Pratami, &RACI Matrix Design For Managing Stakeholders In Project Case Study Of PT. XYZ,& International Journal of Industrial Engineering and Science, pp. S02.134, July 2021. https://doi.org/10.25124/ijies.v5i02.134

[10] P. Laban, H. Hayashi, Y. Zhou, and J. Neville, &LLMs Get Lost In Multi-Turn Conversation,& arXiv preprint arXiv:2505.06120, 2025. https://doi.org/10.48550/arXiv.2505.06120

[11] Z. Long, S. Guo, W. Wu, F. Wei, T. Ji, and H. Ji, &Unleashing The Emergent Cognitive Synergy In Large Language Models: A Task-Solving Agent Through Multi-Persona Self-Collaboration,& arXiv preprint arXiv:2307.05300, 2023. https://doi.org/10.48550/arXiv.2307.05300

[12] Y. Yang, H. Cai, Y. Song, S. Qi, M. Wen, N. Li, J. Liao, H. Ju, J. Lin, G. Liu, W. Liu, Y. Wen, Y. Yu, and W. Zhang, &A Survey Of AI Agent Protocols,& arXiv preprint arXiv:2504.16736, 2025. https://doi.org/10.48550/arXiv.2504.16736

[13] Josh (Juhwan Lee), AI Agent Ecosystem: A New Paradigm Opened by Frameworks and Protocols, Roadbook, 2025.

[14] Korea Database Agency, Data Analysis Expert Guide, Revised Edition, p.101, 2016.

[15] KT, Framework and Trends for AI Agent Development[Internet]. Available: https://kode.kt.com/blog/article/3895

[16] Microsoft Learn, Agent Memory[Internet]. Available: https://learn.microsoft.com/ko-kr/agent-framework/user-guide/agents/agent-memory?pivots=programming-language-csharp

[17] J. Wiesinger, P. Marlow, and V. Vuskovic, Agents, Kaggle, Feb. 2025 [Internet]. Available: https://www.kaggle.com/whitepaper-agents

천지영(Ji-Yeong Cheon)

2025년 : 명지대학교 정치외교학과 (학사)

2025년～현 재: 명지대학교 기록정보과학전문대학원 AI정보과학전공 석사과정

※관심분야 : AI Agents, 데이터 분석, 멀티모달 분석, 딥러닝

윤석용(Seok-Yong Yun)

2000년 : 숭실대학교 정보과학대학원 정보산업학과 (이학석사)

2018년 : 숭실대학교 일반대학원 IT정책경영학과 (공학박사)

1990년 ~ 1994년 (주)포스코 전산시스템부 사원

1994년 ~ 1999년 (주)현대정보기술 IT컨설팅팀 책임

2000년 ~ 2015년 (주)포스코경영연구원 빅데이터TFT 부장

2015년 ~ 2020년 (주)베가스 빅데이터컨설팅팀 부사장

2019년 ~ 현재 명지대학교 기록정보과학전문대학원 AI정보과학전공 주임교수

* 관심분야 : 데이터분석, 기계학습, 인공지능, 자연어처리, LLM/sLLM, 데이터베이스, 데이터 거버넌스, CDS