JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 27, No. 2, pp. 519-531, Feb. 2026

Check for updates

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

천 지 영[1] · 윤 석 용[2*]
[1]명지대학교 AI정보과학전공 석사과정 [2]명지대학교 AI정보과학전공 교수

Multi-Agent Design Methodology (MADM) for AI Systems: 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 유용하지만, 설계 단계의 복잡성과 불확실성으로 인해 구현 과정에서 높은 실패율이 보고되고 있다. 기존 개발 방식과 구현 중심 프레임워크는 이러한 구조적 복잡성과 핵심 설계를 체계적으로 다루기 어렵다. 본 연구는 이를 보완하기 위해 설계 중심 방법론인 MADM을 제안한다. MADM은 환경 분석, AI 에이전트 설계, MAS 구현의 세 단계로 구성되며, RACI 매트릭스를 확장한 RACID 모델로 계층 구조와 책임을 명시한다. 제안 방법론은 복합 제약 조건의 식사 메뉴 추천 시스템에 개념증명(PoC)으로 적용하고, 동일한 구현 조건에서 비교 실험을 수행하였다. 그 결과, MADM 적용 그룹은 평균 점수와 성공 비율에서 개선 경향을 보였으며, 안전·의학 제약 시나리오에서 상대적으로 큰 향상을 보였다. 본 연구는 MAS 개발에서 설계 산출물 기반 체계화의 필요성을 제시하고, MADM의 초기 적용 가능성을 제안한다.

[Abstract]

Multi-agent systems (MAS) are effective for complex problem solving; however, they often exhibit high failure rates because of design-stage complexity and uncertainty. Existing implementation-oriented frameworks provide limited support for systematically structuring key design decisions. This paper proposes a multi-agent design-centric methodology (MADM) to address the lack of systematic agent design methodologies in multi-agent systems development. The proposed methodology comprises environmental analysis, artificial intelligence (AI) agent design, and multi-agent systems (MAS) implementation, while incorporating the RACID(Responsible, Accountable, Consulted, Informed, and Dynamic Coordinator) role model to define hierarchical responsibilities. MADM is evaluated at a proof-of-concept level by using a meal recommendation system with complex constraints under controlled implementation conditions. The results indicate improvement trends in output quality, particularly in safety- and medical-constrained scenarios, indicating the effectiveness of design artifact-based systematization in 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.2026.27.2.519

Received 02 December 2025; **Revised** 02 January 2026
Accepted 29 January 2026

***Corresponding Author; Seok-Yong Yun**
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 설계 결함에서 비롯된 사양 설계 실패가 전체의 44.2%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[3]. 이는 MAS 개발에서 설계 단계의 중요성을 시사하지만, 현재 산업 현장에서는 설계 중심의 체계적인 접근보다는 개발자의 경험과 직관에 의존한 구현 중심 접근이 일반적으로 사용되고 있다.

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[4], 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, LLM 기반 에이전트의 비결정성이다. 에이전트의 행동은 프롬프트, 도구, 메모리 구조 등 설계자가 정의한 요소에 의존하며[5],[6], 이러한 요소가 체계적으로 설계되지 않는다면 협업 효과는 급격히 떨어지고 시스템 또한 불안정해진다[7]. 셋째, 구현 중심 프레임워크의 방법론적 공백이다. AutoGen, LangGraph, Google ADK 등은 구현 도구를 제공하지만, '무엇을, 왜, 어떻게 설계해야 하는가'에 대한 가이드라인을 제공하지 못한다. 이로 인해 MAS 개발 과정에서는 설계 산출물의 일관성 부족과 반복적인 시행착오가 발생할 가능성이 높다. 이러한 배경에서 본 연구는 MAS 개발 과정에서 설계 단계의 구조화를 지원하기 위한 설계 중심 방법론인 MADM(Multi Agent Design Methodology)을 제안한다.

본 연구는 MAS 설계 방법론 부재가 초래하는 구조적 문제를 규명하고, 이를 완화할 수 있는 개념적 프레임워크를 제시하는 탐색적 연구이다. 기존 연구들이 구현 도구와 프레임워크 개발에 집중한 반면, 본 연구는 설계 단계 자체의 체계화에 초점을 둔다. 따라서 본 연구의 목적은 MADM의 일반적 성능 우수성을 입증하기보다는, 체계적 설계 접근이 결과물 품질에 미치는 영향을 개념증명 (PoC; Proof-of-Concept) 수준에서 탐색하는 데 있다. 현재 MAS의 '설계 결함(design-stage defect)'에 대한 정량적 정의와 측정 기준이 충분히 합의되지 않았으므로, 본 연구는 설계 프로세스 자체를 직접 측정하는 대신, 최종 출력물의 품질 지표(제약 준수율, 목표 달성률, 실패 유형 분포 등)를 통해 설계 방법론의 효과를 간접적으로 평가한다. 이러한 결과물 중심 평가는 설계 단계의 체계성이 실제 시스템 산출물에 어떻게 반영되는지를 관찰할 수 있는 현 시점에서 타당한 초기 검증 방법으로 판단하였다. 따라서 본 연구의 결과는 특정 도메인·구현 환경에 대한 일반적인 성능 비교가 아니라, 설계 중심 접근이 출력 안정성과 실패 양상에 유도하는 경향적 변화를 제시하는 사례로 해석되어야 한다.

실험 범위는 3~10개의 에이전트로 구성된 중소규모 MAS로 한정하며, 본 연구는 체계적 설계가 결과물에 미치는 영향을 탐색하는 초기 단계 연구로 위치한다. MADM은 환경 분석, AI 에이전트 설계, MAS 구현의 세 단계로 구성된다.

그림 1. MADM 방법론의 3단계
Fig. 1. The three phases 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[3]. 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[3]. 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 오픈 소스 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), 역할 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 보고되었다. 특히 System Design Issues(44.2%)는 설계 단계 결함으로 직접 분류되며, Inter-Agent Misalignment(32.3%) 역시 역할 정의·조정 구조의 미흡 등 설계 결정의 영향을 크게 받는 범주로 보고된다. 이에 본 연구는 두 범주를 '설계에 강하게 연동된 실패'로 묶어 해석하고, 해당 비중을 76.5%(44.2%+ 32.3%)로 요약하였다[2]. 이는 MAS 실패가 구현 오류뿐 아니라 설계 단계의 역할·책임 정의와 협업 구조 설정에 의해 크게 좌우될 수 있음을 시사한다.

본 연구에서 설계 결함(design-stage defect)은 MAST의 설계 오류(System Design Issues)와 설계 단계에서 기인한 역할 오류(Inter-Agent Misalignment) 범주를 중심으로, 구현 이전 단계에서 (1) 요구사항/제약 조건의 누락 및 모순, (2) 역할/책임의 불명확성 또는 중복, (3) 협업 및 중재 구조의 부재 또는 부적절한 설계, (4) 메모리/도구/자율성 설정의 범위 불일치로 인해 시스템의 의사결정이 일관되게 수행되지 못하는 경우를 포함한다.

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[3]. 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[7]. 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[3].

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[8]. 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

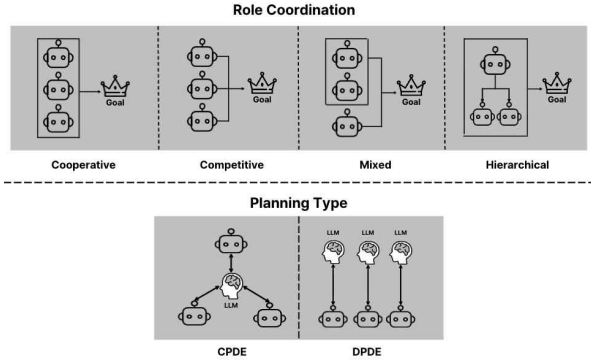

그림 2. MAS에서 역할 조정 및 계획 유형
Fig. 2. Role coordination types and planning structures in MAS

MAS의 조직구조는 그림 2와 같이 에이전트 간 관계 유형과 계획 수립 방식에 따라 다양하게 구분된다[9].

에이전트 간 관계 유형은 협력형, 경쟁형, 혼합형, 계층형으로 나눌 수 있다. 협력형(Cooperative)은 모든 에이전트가 동일한 목표를 공유하며, 문제를 세분화하여 병렬로 처리한다. 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[9].

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[10].

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.

2-8 AI 에이전트 메모리

메모리는 LLM과 AI 에이전트를 구별하는 핵심 요소로, 대화 맥락을 유지하고 사용자 선호나 과거 상호작용을 기억함으로써 개인화된 응답과 일관된 행동을 가능하게 한다[11]. 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[12]. 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[3]. 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[13].

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. Multi Agent Design 방법론

MADM(Multi-Agent Design Methodology)은 AI 에이

전트 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, 구현 이전 단계에서 요구사항, 제약조건, 역할 구조를 명확히 정의한다. 둘째, RACID 모델을 통한 역할 기반 협업 설계로 역할 중복이나 충돌을 최소화한다. 셋째, 하향 설계와 상향 구현으로 오류를 조기에 발견하고 수정 가능하도록 한다. 넷째, 계층적 조정 구조를 도입하여 상위 에이전트는 목표 설정과 방향 조정을, 하위 에이전트는 작업 실행을 담당함으로써 결정의 일관성을 확보한다. 이러한 원칙은 MAS 개발에서 빈번히 발생하는 설계 단계 불확실성을 구조적으로 완화하고, 시스템의 재현 가능성과 안정성을 확보하기 위한 MADM의 핵심 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. The step-by-step process of the AI agent design phase

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 기능군과 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 최하위 레벨의 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[10].

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
A101	Collaborator	1	●	-	●	○	△	L4	Domain oversight
A301	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[15]. 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[12]. 다음으로, 메모리 활용 전략은 시스템의 특성에 따라 선제적, 선택적, 또는 도구 기반 방식 중 적합한 접근을 선택한다[13]. 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[9]. 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 Implementation)

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. 대표적인 에이전트 프레임워크로는 LangChain, AutoGen, CrewAI, LangGraph 등이 있으며, 프레임워크 선정 시에는 다음의 기준을 종합적으로 고려한다.

① 시스템의 목표와 정합성을 평가: 빠른 프로토타이핑과 검증이 목표인지, 엔터프라이즈급의 안정성과 확장성이 중요한지에 따라 적합한 프레임워크가 달라진다[3].

② 기술 생태계 및 인력 확보 용이성을 검토: 기존 인프라(클라우드, 데이터베이스, 인증 시스템)와의 호환성, 개발 인력 확보 용이성, 커뮤니티 활성화 수준을 함께 검토해야 한다.

③ 라이선스 조건 및 비용 검토: 오픈소스 유무, 상업적 이용 제약의 유무, API 제한이 있는지 등을 확인해야 한다.

④ 유지 보수 및 발전 가능성을 평가

2) 에이전트 시스템 구현

이 단계에서는 설계 명세서를 바탕으로 선택된 프레임워크 내에서 LLM, Memory, Tool, Autonomy의 네 가지 핵심 구성요소를 바탕으로 구현한다.

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 효율 위주의 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[16].

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. 높은 자율성 단계(4~5)에서는 Self-Refine, Reflection 등으로 출력의 품질을 반복적으로 개선하거나 실패 경험에서 학습할 수 있도록 구현한다[8].

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[17]. 프로토콜 선정 시 에이전트의 자율성 수준, 통신 유연성, 인터페이스 표준화 정도, 보안을 종합적으로 고려해야 한다[3].

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 관계의 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[9].

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

실제 운영 환경에 배포된다.

Ⅳ. 적용 사례

4-1 실험 설계 및 목표

1) 목적

본 실험은 MADM 방법론의 일반적 성능 우수성을 입증하기보다는, 복합 제약 조건 문제 영역에서 체계적 설계 접근이 결과물에 미치는 영향을 탐색적으로 확인하기 위한 개념증명(PoC)을 목적으로 한다. 이상적으로는 설계 단계 자체의 결함을 직접 측정하고 정량화하는 것이 바람직하나, 현재 MAS 분야에서는 설계 결함에 대한 정량적 정의와 측정 기준이 충분히 합의되지 않은 상황이다. 이에 본 연구는 설계 프로세스 자체를 직접 계량화하는 대신, 최종 출력물의 품질 지표를 통해 설계 방법론의 효과를 간접적으로 평가하는 접근을 채택하였다. 이러한 결과물 중심 평가는 설계 단계의 체계성이 실제 시스템 산출물에 어떻게 반영되는지를 관찰할 수 있는 현 시점에서 타당한 초기 검증 방법으로 판단하였다.

2) 실험 설계

① 비교 그룹 설정: MADM을 통한 체계화된 설계 접근과 구현 중심 설계 접근 간의 차이를 정량적으로 비교하기 위해 두 개의 그룹을 구성하였다. Group A(대조군)은 명확한 설계 방법론 없이 직관과 경험에 기반해 개발하였으며, 현재 산업 현장의 일반적 개발 관행을 대표한다. Group B(실험군)는 MADM 방법론에 기반해 개발되었으며, RACID 모델, 우선순위 계층 모델을 적용하였다.

② 비교 그룹의 대표성과 통제: 본 실험의 Group A(대조군)는 AutoGen, CrewAI 등 기존 구현 중심 MAS 프레임워크 문서에서 제시하는 기본 패턴을 적용한 설계 수준을 반영한다. 즉, 기능별 에이전트 분리, 단순한 제약 처리 우선순위 규칙, 기본적인 메모리 구조화 및 자율성 설정은 포함하되, RACID 매트릭스와 같은 역할·책임 명세, 계층적 조정 구조, 설계 산출물 문서화는 적용하지 않았다. 반면, Group B는 이러한 기본 설정에 더해, MADM 방법론에 따라 설계 산출물을 사전에 정의하고 이를 구현 단계에서 반영하였다. 이에 따라 두 그룹 간 비교는 산출물 기반의 설계 방법론 적용 여부에 따른 차이를 관찰하도록 구성하였다.

③ 시스템 및 환경 통제: 두 그룹 모두 복합 제약 조건을 가진 '영양, 선호도, 예산을 동시에 만족하는 식사 메뉴 선정 시스템'을 실험 도메인으로 선정했다. 또한, 에이전트 구성(10개), LLM 모델(Gemini 2.0 Flash), 프레임워크(Google ADK), 도구(RAG, Function Calling)를 동일하게 설정하여 관측된 차이가 설계 체계화 수준 이외의 요인에서 발생하지 않도록 통제하였다.

표 4. AI 에이전트 목록
Table 4. AI agent list

Agent ID	Agent name
A001	System Orchestrator
A101	Menu Planner & Mediator
A201	User Preference & Health Analyzer
A202	Budget & Market Analyzer
A301	Korean Chef(Health-Aware)
A302	Japanese Chef(Health-Aware)
A303	Chinese Chef(Health-Aware)
A311	Korean Chef (Budget-Aware)
A312	Japanese Chef (Budget-Aware)
A313	Chinese Chef (Budget-Aware)

④ 입력 시나리오 구성: 실제 사용자의 다양한 표현 방식을 반영하기 위해 복합 제약 조건을 포함하는 10개의 테스트 시나리오를 설계하였다. LLM 사용 경험이 있는 20대 성인 5명이 각 시나리오에 대해 서로 다른 표현의 프롬프트를 작성하였다. 1인당 10개의 프롬프트를 생성하여 총 50개의 프롬프트를 확보하였고, 이를 두 그룹에 각각 동일하게 입력하여 총 100회의 실험을 수행하였다

⑤ 테스트 시나리오: 각 시나리오는 서로 다른 제약 조건 충돌 패턴과 실패 위험도를 포함한다.

3) 평가 기준 및 측정 방법

각 그룹당 50회의 응답을 평가하였다. 평가는 실험을 진행한 5명이 진행하였다. 각 응답은 제약 조건 준수, 영양 목표 달성, 예산 준수, 실행 가능성의 4개 차원에서 각 25점씩 배점되며, 총 100점 만점으로 환산된다. 생명이나 건강에 직접적 위험을 초래하는 경우는 Critical Failure(0점), 핵심 제약 조건을 충족하지 못한 경우는 Major Failure(1~41점), 실행 가능성이나 응답의 완결성이 부족한 경우는 Minor Failure(42~70점), 비교적 준수한 응답인 경우에는 Success(71점 이상)으로 정의하였다. 부가적으로는 응답 시간을 비교한다.

본 실험은 결과물의 절대적 성능 우위를 판단하기 위한 것이 아니라, 설계방법론(MADM)에 기반한 설계 체계화(RACID 기반 역할·책임 명세, 우선순위 계층화, 계층적 중재

표 5. 실험 시나리오 및 제약 조건
Table 5. Experimental scenarios and constraint settings

Scenario ID	Scenario Name	Constraints	Conflict Type	Failure Risk
S1	Budget-Constrained Single Household	Protein ≥ 60 g/day + Budget ≤ 35,000 KRW/week	Nutrition-Budget	Medium
S2	Elderly with Diabetes Preferring Chinese Food	GI ≤ 55 + Preference for Chinese cuisine	Health-Preference	High
S3	Multiple Food Allergies	Exclusion of shellfish, nuts, buckwheat + Korean cuisine	Safety-Preference	High
S4	Gestational Diabetes	GI ≤ 55 + Calcium ≥ 1,000 mg/day + Exclusion of large fish	Health-Nutrition	High
S5	Hypertension with Low Sodium	Sodium ≤ 2,000 mg/day + Taste prioritized + Budget ≤ 9,000 KRW	Health-Preference-Budget	High
S6	Diabetes, Hypertension, and Medication	GI ≤ 55 + ACE inhibitor use + High-potassium food avoidance	Health-Medication	Very High
S7	Picky Eating Child	Vegetable refusal + High protein/calcium + No liquid fructose	Preference-Nutrition	Medium
S8	Six-Member Multigenerational Family	Dysphagia + Regular diet + Meat preference + Budget ≤ 50,000 KRW	Multiple Demands	High
S9	Marathon Runner (D-3)	Carbohydrates ≥ 70% + High digestibility + Low irritation	Nutrition-Preference	Medium
S10	Elderly on Warfarin	Strict vitamin K restriction + Dysphagia	Medication-Form	Very High

구조)가 출력 안정성과 실패 유형 분포의 변화를 유도하는 경향을 관찰하는 데 목적이 있다.

표 6. 실패 분류 기준
Table 6. Failure classification criteria

Category	Critical Failure	Major Failure
Constraint	Medical violation	Missing constraints
Nutrition	< 50% of target achieved	< 80% of target achieved
Budget	> 200% of budget	> 150% of budget
Feasibility	Infeasible execution	Incomplete execution

4-2 Group A(대조군)

Group A는 프레임워크 기본 패턴 수준의 설계를 반영한다. 별도의 설계 산출물 문서화 없이 기능 요구사항 중심으로 개발하였으며, 개발 절차는 다음과 같다.

1) 에이전트 구성 및 역할 정의: 표 4에 제시된 10개의 에이전트를 생성하고, 각 에이전트의 역할을 직관적으로 정의하였다. 예를 들어, Korean Chef(Health-Aware)는 "당신은 한식 레시피 전문가입니다. 사용자의 건강을 고려해 최적의 레시피를 작성하세요."와 같이 전문성을 강조하는 지침으로 구성하였다.

2) 제약 충돌 규칙: 복합 제약 조건이 충돌하는 경우, "건강 → 예산 → 선호도" 순서의 단순 규칙을 적용하였다.

3) 메모리 및 자율성 설계: STM(대화 세션 메시지)과 LTM(사용자 프로필, 선호도, 건강 정보, 레시피 등)으로 구분하였으며, 선제적 활용 전략을 사용했다. 메모리 접근 권한은 모든 에이전트에게 동일하게 부여했다. 자율성 수준은 각 에이전트의 기능에 맞게 설정하였다.

4) 도구 및 추론프레임워크: 추론프레임워크는 CoT를 적용하였으며, 식재료 가격 조회 및 영양소 정보 조회를 위해 RAG와 통신을 위해 MCP를 사용하였다.

4-3 Group B(실험군)

Group B는 MADM 방법론에 따라 설계 산출물을 사전에 정의하고 구현에 반영하였다.

1) 환경 분석
"영양, 선호도, 예산을 동시에 만족하는 식사 메뉴 추천" 문제를 탐색형 문제(Exploratory type)로 분류하고, 요구사항을 도출하여 문서화하였다.

2) AI에이전트 설계
10개의 에이전트를 4단계 계층 구조로 배치하고 단일 책임 원칙에 따라 역할을 분리하였다. A001(System Orchestrator)은 Accountable로서 최종 의사결정 권한, A101(Menu Planner)은 Dynamic Coordinator로써 제약 충돌 시 중재를 담당한다. A201/A202는 각각 건강 분석과 예산 분석에 대한 Responsible 역할을 수행한다. 제약 조건 충돌 시 우선순위는 "생명 안전 → 의학적 제약 → 영양 목표 → 예산 → 선호도"로 계층화하였다. 메모리 활용 전략은 기본적으로 선제적 활용 전략을 선택하였다. 아키텍처는 STM과 LTM으로 구분하였으며, 에이전트 역할에 따라 접근 권한을

차등 설정하였다. A001만 전체 메모리에 대한 관리자 권한을 가지며, 나머지 에이전트는 담당 영역에 대해서만 읽기 또는 쓰기 권한을 부여받는다.

표 7. 그룹 B: RACID 매트릭스
Table 7. Group B: 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

표 8. 그룹 B: 메모리 아키텍처
Table 8. Group B: 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

설계한 최종 MAS의 다이어그램은 다음과 같다.

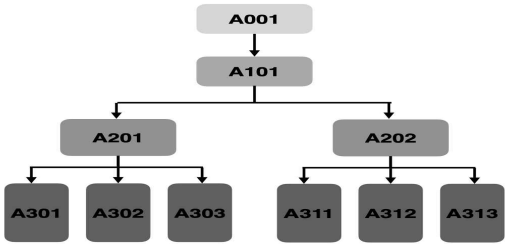

그림 5. 그룹 A: 다중 에이전트 시스템 관계 모델
Fig. 5. Group A: MAS relationship model

3) MAS 구현

실험 설계에서 정의한 대로 물리적, 기술적 조건을 동일하게 적용하였다. 에이전트 자율성은 계층별 의사결정 루프 구조로 구현하였다. A001(L4)은 하위 에이전트의 결과를 평가하고 필요 시 재작업을 지시하며, 제약 조건 충돌 발생 시 자율적으로 중재안을 도출한다. A101, A201, A202(L3)는 각자의 영역에서 독립적으로 분석과 문제 해결을 수행하되, 처리 범위를 초과하는 경우 상위 레벨 에이전트(A001)에게 판단을 위임하는 에스컬레이션 구조를 통해 의사결정의 일관성을 유지하도록 설계하였다. 반면, A301~A303/A311~A313의 에이전트(L2) 단순 실행 루프 기반으로 구현되어 템플릿 기반 레시피 검색과 제한된 범위 내 조정만 수행하도록 제한하였다. 외부 통신을 위해 MCP를 적용하였으며, 시스템 오케스트레이션 방식으로는 CPDE를 채택하였다[8].

4-4 결과

1) 전체 성능 비교

50개의 프롬프트를 각 그룹에 입력한 결과, MADM 방법론을 적용한 Group B는 Group A(Baseline) 대비 평균 점수와 Success 비율에서 상대적으로 높은 값을 보였다.

Group B의 평균 점수는 Group A보다 10.24점 높게 나타났으며, 평가 기준을 충족하는 출력(Success)의 발생 빈도 역시 더 자주 관찰되었다. 또한 Major Failure 비율은 6.0%p 감소하여, 일부 시나리오에서 핵심 제약 조건 위반이 상대적

표 9. 그룹별 전체 성능 비교
Table 9. Overall performance comparison by group

Metric	Group A (Baseline)	Group B (MADM)	Improvement
Average Score	56.24	66.48	+10.24 (+18.2%)
Standard Deviation	16.24	21.22	+5.0
Success Rate (≥71pts)	18.0% (9/50)	56.0% (28/50)	+38.0%p
Major Failure (≤40pts)	20.0% (10/50)	14.0% (7/50)	−6.0%p
Average Response Time (sec)	24.65	28.07	+3.42

으로 적게 발생했음을 확인하였다.

2) 평가 차원별 성능 분석

4개 평가 차원(제약 준수, 영양 목표, 예산 준수, 실행 가능성)을 기준으로 분석한 결과, Group B는 제약 준수, 영양 목표, 예산 준수 차원에서 점수 증가가 관찰되었으며, 실행 가능성 차원에서는 소폭 감소가 나타났다.

점수 차이는 특히 예산 준수와 영양 목표 달성 차원에서 크게 나타났는데, 이는 Group B가 제약 조건과 목표를 구조적으로 처리하도록 설계되었기 때문으로 해석할 수 있다. 반면 실행 가능성 점수의 감소는 Group B가 상위 제약 충족에 상대적으로 더 집중하면서, 출력의 구체성이나 단계적 설명이 일부 희석된 결과로 볼 수 있다. 이러한 결과는 MADM이 제약 중심 문제 해결에 강점을 보이지만, 실행 관점에서는 추가적인 보완 여지가 있음을 시사한다.

표 10. 평가 차원별 상세 점수
Table 10. Detailed scores by evaluation dimension

Dimension	Group A	Group B	Improvement
Constraint Compliance (25pts)	14.46	16.62	+2.16
Nutrition Goal Achievement (25pts)	14.48	18.26	+3.78
Budget Compliance (25pts)	12.50	17.90	+5.40
Execution Feasibility (25pts)	14.80	13.70	−1.10

3) 시나리오별 성능 분석

10개의 테스트 시나리오 중 복합 제약 조건이 명확한 시나리오에서 Group B의 점수 차이가 비교적 크게 나타났으며, 일부 시나리오에서는 Group A가 더 높은 점수를 기록하였다.

특히 S3(다중 알레르기), S6(당뇨+ 고혈압+ 약물), S10(와파린 복용)과 같이 안전 및 의학적 제약이 중첩되는 시나리오에서는 Group B의 점수가 더 높게 관찰되었다. 이는 MADM의 우선순위 계층화와 역할 분리가 복합 제약 상황에서 일정 수준의 안정성을 제공했을 가능성을 보여준다. 반면 S7(편식 아동), S8(대가족), S9(마라톤 D-3)와 같이 특수 요구사항 또는 선호 중심 문제에서는 Group B의 점수가 낮게 나타났다. 이는 현재 MADM의 우선순위 체계가 안전 및 제약 중심으로 설계되어 있어, 특수 영양 비율이나 선호 조정이 핵심이 되는 시나리오에서는 충분히 대응하지 못했음을 시사한다. 본 결과는 MADM의 적용 범위와 한계를 함께 보여주는 사례적 결과로 해석하는 것이 적절하다.

표 11. 시나리오별 성능 및 개선율
Table 11. Performance and improvement rate by scenario

Scenario	Failure Risk	Group A	Group B	Improvement
S1	Medium	60.4	84.4	+24.0
S2	High	59.6	62.8	+3.2
S3	High	33.0	67.4	+34.4
S4	High	58.8	80.2	+21.4
S5	High	65.6	79.0	+13.4
S6	Very High	44.2	80.4	+36.2
S7	Medium	56.0	41.6	−14.4
S8	High	57.0	49.2	−7.8
S9	Medium	69.6	37.6	−32.0
S10	Very High	58.2	82.2	+24.0

4) 응답 시간 분석

MADM의 구조적 설계는 응답 시간 증가로 이어지는 경향을 보였다. Group B는 Group A 대비 평균 응답 시간이 3.42초(13.9%) 더 길게 나타났다. 다만 두 그룹 모두 제약 조건에 대한 우선순위 규칙을 적용하고 있다는 점에서, 이러한 차이는 우선순위 제약의 존재 자체보다는 제약을 처리하는 방식의 차이에서 기인한 것으로 해석된다.

Group A에서는 우선순위 규칙이 단일 단계의 판단 로직으로 적용된 반면, Group B에서는 RACID 기반 역할 분리와 계층적 중재 구조를 통해 제약 충돌이 단계적으로 처리되었다. 이 과정에서 분석, 중재, 통합 단계가 명시적으로 분리되어 수행되었으며, 특히 복합 제약이 포함된 시나리오(S6, S10)에서는 해당 절차가 반복적으로 수행되었다. 이러한 결과는 MADM이 설계 품질과 안전성을 중시하는 과정에서 추가적인 처리 비용이 발생할 수 있음을 시사한다.

표 12. 응답 시간 비교
Table 12. Response time comparison

Metric	Group A	Group B	Difference
Average (sec)	24.65	28.07	+3.42 (+13.9%)
Std Dev (sec)	5.51	4.16	−1.35
Min (sec)	16.4	15.6	−0.8
Max (sec)	39.9	37.8	−2.1

5) 종합 분석 및 한계

본 실험에서 MADM을 적용한 Group B는 Group A 대비 평균 점수(+10.24점, +18.2%)와 Success 비율(+38.0%p)이 높은 경향을 보였고, Major Failure 비율은 6.0%p 감소하였다. 특히 안전 및 의학 제약이 중첩되는 시나리오(S3, S6, S10)에서 Group B의 점수 개선폭이 상대적으로 크게 관찰되어, RACID 기반 역할 분리와 우선순위 계층화가 제약 충돌 처리에 구조적 이점을 제공할 가능성을 시사한다. 반면 선호/비율 조정이 핵심인 시나리오(S7, S8, S9)에서는 성능 저하가 관찰되었고, 응답 시간도 13.9% 증가하여 역할 분리와 계층적 중재 구조가 추가적인 처리 비용을 수반할 가능성을 확인하였다.

본 연구는 다음과 같은 한계를 가진다. 첫째, 설계 프로세스 자체를 직접 측정하지 못했다. 본 연구는 설계 결함 감소를 문제의식으로 삼았으나, MAS 설계 결함에 대한 정량적 정의와 측정 기준이 아직 충분히 합의되지 않은 상황에서 설계 오류 발생 빈도, 역할 충돌 횟수, 재설계 횟수, 요구사항 변

경 빈도, 설계 소요 시간 등 설계 단계의 정량 지표를 수집이 어렵다. 이에 따라 관찰된 출력 품질 향상이 설계 품질 개선의 직접적 결과인지, 혹은 프롬프트 품질이나 모델 변동과 같은 다른 요인의 영향인지 명확히 분리하여 설명하기 어렵다. 둘째, 우선순위 체계의 도메인 적응 방법을 제시하지 못했다. 안전 중심 우선순위가 특정 시나리오에서 성능 저하로 이어질 수 있음을 관찰했으나, 도메인별로 우선순위를 조정의 원칙이나 기준을 본 연구에서 제시하지 못했다. 셋째, 일반화 가능성이 제한적이다. 단일 도메인(식사 메뉴 추천), 중소규모 에이전트(10개 에이전트), 50개 테스트 케이스에 한정되어 다양한 환경에서의 재현성은 추가 검증이 필요하다.

V. 결 론

본 연구는 MAS 개발의 높은 실패율 문제에 대한 해결 가능성을 탐색하기 위해 설계 중심 방법론인 MADM을 제안하였다. 기존 연구에서 MAS 실패의 약 44.2%가 초기 설계 단계 문제에서 비롯된다고 보고되었으나[2], 현재 산업 현장에서는 구현 중심 프레임워크에 의존한 개발이 일반적이며 체계적 설계 접근은 부재한 상황이다. 이러한 배경에서 MADM은 환경 분석, AI 에이전트 설계, MAS 구현의 세 단계로 구성되며, 특히 RACID 모델을 통한 역할·책임 명시화, 계층적 조정 구조, 우선순위 계층화 등 설계 산출물 기반의 체계적 접근을 제시한다. 또한 하향식 설계와 상향식 검증의 순환 구조를 통해 MAS 개발 전반의 일관성과 재현 가능성을 확보하는 방향을 지향한다.

다만 본 연구는 다음과 같은 한계를 가진다. 첫째, 설계 결함 자체를 직접 계량화하지 못했다. MAS 설계 결함의 정량적 정의와 측정 기준에 대한 합의가 부족한 상황에서 출력물 품질 지표를 통해 간접 평가를 수행하였으므로, 관찰된 품질 변화가 설계 개선에 의해 발생한 효과인지 다른 요인과 명확히 분리하여 설명하는 데 제약이 있다. 둘째, 단일 도메인(식사 메뉴 추천), 중소규모 구성(10개 에이전트), 제한된 테스트 케이스(50개)에 기반한 실험으로 일반화 가능성이 제한적이다. 셋째, 평가자 간 신뢰도 검증과 통계적 유의성 검정을 포함하지 않은 개념증명(PoC) 수준의 탐색적 실험이다. 넷째, 우선순위 체계의 도메인 적응 원칙을 충분히 제시하지 못하여, 안전 중심 설계가 특정 문제 유형에서 부적합할 수 있음을 관찰했음에도 이를 조정하는 구체적 기준을 제안하지 못했다.

이러한 한계 범위 내에서 수행한 PoC 실험에서, MADM 적용 그룹은 비교 그룹 대비 평균 점수(+18.2%), Success 비율(+38.0%p) 증가 및 Major Failure 비율(−6.0%p) 감소의 경향을 보였다. 특히 안전 및 의학 제약이 중첩되는 복합 시나리오에서 개선폭이 상대적으로 크게 관찰되어, RACID 기반 역할 분리와 우선순위 계층화가 제약 충돌 처리 과정에서 구조적 이점을 제공했을 가능성을 시사한다. 반면 선호 또는 특수 요구사항 중심 시나리오에서는 성능 저하와 응답 시간 증가가 함께 관찰되어, MADM의 적용이 문제 유형에 따라 트레이드오프를 수반할 수 있음을 확인하였다. 따라서 본 결과는 MADM의 일반적 우수성을 입증하기보다는, 설계 체계화가 특정 문제 유형(복합 제약 조건)에서 출력 안정성과 실패 양상에 미칠 수 있는 영향의 방향성을 제시하는 초기 사례로 해석되어야 한다.

본 연구의 기여는 다음과 같다. 첫째, MAS 개발에서 설계 방법론 부재가 초래할 수 있는 구조적 문제를 논의하고 설계 중심 접근의 필요성을 제기하였다. 둘째, RACID 역할 모델을 기반으로 역할·책임·조정 구조를 명시하는 설계 프레임워크(MADM)를 제안하여 향후 연구의 개념적·방법론적 출발점을 제공하였다. 셋째, 구현 중심 접근과 설계 중심 접근을 동일한 구현 조건에서 비교함으로써, 설계 산출물 기반 체계화가 결과물 품질에 영향을 미칠 수 있는 가능성을 탐색적으로 제시하였다.

향후 연구에서는 (1) 설계 단계 결함을 직접 측정할 수 있는 정량 지표 및 계량 절차 개발, (2) 도메인별 우선순위 조정 원칙과 적용 기준 수립, (3) 다양한 도메인·규모에서의 재현 실험 및 통계적 검증, (4) 평가자 간 신뢰도 검증을 포함한 엄밀한 실험 설계를 통해 MADM의 효과와 적용 범위를 보다 명확히 규명할 필요가 있다. 본 연구는 MAS 개발 실패 감소를 위한 설계 중심 접근의 초기 제안으로서, 향후 실제 프로젝트와 다양한 도메인에 적용·검증됨으로써 그 유효성이 확장되기를 기대한다.

참고문헌

[1] M. A. Ferrag, N. Tihanyi, and M. Debbah, “From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review,” arXiv:2504.19678, 2025. https://arxiv.org/abs/2504.19678

[2] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, ... and I. Stoica, “Why Do Multi-Agent Llm Systems Fail?,” arXiv:2503.13657, 2025. https://arxiv.org/abs/2503.13657

[3] J. Lee, *AI Agent Ecosystem: A New Paradigm Opened By Frameworks and Protocols*, Roadbook, Seoul, 2025.

[4] J. Gardner and V. A. Baulin, “Is the ‘Agent’ Paradigm a Limiting Framework for Next-Generation Intelligent Systems?,” arXiv:2509.10875, 2025. https://arxiv.org/abs/2509.10875

[5] B. Atil, S. Aykent, A. Chittams, L. Fu, R. J. Passonneau, E. Radcliffe, ... and B. Baldwin, “Non-Determinism of ‘Deterministic’ LLM Settings,” arXiv:2408.04667, 2025. https://arxiv.org/abs/2408.04667

[6] H. P. Zou, W.-C. Huang, Y. Wu, Y. Chen, C. Miao, H.

Nguyen, ... and P. S. Yu, “LLM-Based Human-Agent Collaboration and Interaction Systems: A Survey,” arXiv:2505.00753, 2025. https://arxiv.org/abs/2505.00753

[7] H. Derouiche, Z. Brahmi, and H. Mazeni, “Agentic AI Frameworks: Architectures, Protocols, and Design Challenges,” arXiv:2508.10146, 2025. https://arxiv.org/abs/2508.10146

[8] K. J. K. Feng, D. W. McDonald, and A. X. Zhang, “Levels of Autonomy for AI Agents,” arXiv:2506.12469, 2025. https://arxiv.org/abs/2506.12469

[9] Y. Cheng, C. Zhang, Z. Zhang, X. Meng, S. Hong, W. Li, ... and X. He, “Exploring Large Language Model Based Intelligent Agents: Definitions, Methods, and Prospects,” arXiv:2401.03428, 2024. https://arxiv.org/abs/2401.03428

[10] R. D. P. Suhanda and D. Pratami, “Raci Matrix Design for Managing Stakeholders in Project Case Study of PT. XYZ,” *International Journal of Innovation in Enterprise System*, Vol. 5, No. 2, pp. 122-133, 2021. https://doi.org/10.25124/ijies.v5i02.134

[11] Microsoft Learn. Agent Memory [Internet]. Available: https://learn.microsoft.com/ko-kr/agent-framework/user-guide/agents/agent-memory.

[12] P. Laban, H. Hayashi, Y. Zhou, and J. Neville, “LLMs Get Lost in Multi-Turn Conversation,” arXiv:2505.06120, 2025. https://arxiv.org/abs/2505.06120

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

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

[15] Z. Wang, S. Mao, W. Wu, F. Wei, T. Ge, and H. Ji, “Unleashing the Emergent Cognitive Synergy in Large Language Models: A Task-Solving Agent through Multi-Persona Self-Collaboration,” arXiv:2307.05300, 2023. https://arxiv.org/abs/2307.05300

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

[17] Y. Yang, H. Cai, Y. Song, S. Qi, M. Wen, N. Li, ... and W. Zhang, “A Survey of AI Agent Protocols,” arXiv:2504.16736, 2025. https://arxiv.org/abs/2504.16736

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～현 재: 명지대학교 기록정보과학전문대학원 AI정보과학전공 주임교수

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