Semantic Foundations for Modeling Agentic AI Systems

Kabul Kurniawan[*]
kabul.kurniawan@ugm.ac.id
Universitas Gadjah Mada
Yogyakarta, Indonesia

Andreas Ekelhart
andreas.ekelhart@univie.ac.at
University of Vienna & SBA Research
Vienna, Austria

Elmar Kiesling
elmar.kiesling@wu.ac.at
Vienna University of Economics and Business
Vienna, Austria

ABSTRACT

Agentic AI systems – autonomous, goal-directed entities capable of adaptive decision making – are gaining traction in domains that require the orchestration of complex tasks. Despite their growing adoption, current frameworks often lack a formalized model and architecture. Hence, many implementations remain ad-hoc, relying on simplistic data structures and monolithic designs that hinder scalability, reusability, and interoperability. This paper addresses these limitations by introducing an OWL/RDF-based ontology and accompanying knowledge graph that formally represent the core concepts, components, and interactions that underpin agentic AI workflows. Our ontology provides a standardized vocabulary for modeling agentic patterns – including agents, tasks, workflows, and resource dependencies. Building on this foundation, we present a modular architecture designed to populate and integrate agentic patterns from frameworks across application domains. We evaluate our approach through three real-world use cases, i.e., declarative reconstruction of agentic patterns, cross-context reuse of tasks and agents, and implementation auditing. Our results demonstrate the potential of semantic technologies to bring structure, reusability, and transparency to agentic AI systems.

KEYWORDS

Ontology, Agentic AI, LLMs, Knowledge Graph

ACM Reference Format:
Kabul Kurniawan, Andreas Ekelhart, and Elmar Kiesling. 2021. Semantic Foundations for Modeling Agentic AI Systems. In *The Thirteenth International Conference on Knowledge Capture, December 10–22, 2025, Ohio, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3465481.3465767

1 INTRODUCTION

Agentic AI systems, characterized by their autonomy, goal-directed behavior, and capacity for adaptive decision-making, are increasingly deployed in domains that demand the orchestration of complex, multi-step workflows—ranging from cybersecurity[16], finance [25], industrial automation[3], and healthcare[15] to scientific research[10]. These systems promise enhanced flexibility, coordination, and scalability by delegating tasks to intelligent agents capable of interacting, reasoning, and responding to dynamic environments [34].

The year 2025 in particular could become a pivotal moment for agentic AI according to industry leaders and analysts who predict a surge in the adoption and implementation of agentic AI systems across various sectors[1] and expect large growth of the global agentic AI market in the next years[2].

Despite growing adoption, current implementations of such systems typically rely on frameworks that are characterized by tight coupling of ad-hoc data structures and configurations. This includes modern agentic AI frameworks such as Microsoft's AutoGen [35] CrewAI[3], LangGraph [33] and Mastra AI[4], which typically encode logic and workflows directly into code, resulting in hard-coded and brittle monolithic software architectures. This lack of standardization and formalization not only limits system scalability and maintainability, but it also hampers reusability, traceability, and interoperability across tools, platforms, and domains. Furthermore, the absence of a shared vocabulary and well-defined design patterns prevents practitioners from reusing existing agentic solutions and systematically managing workflows, goals, and resource dependencies.

To address these challenges, we introduce semantic foundations for modeling agentic AI systems. Specifically, we conceptualize agentic AI systems and provide an Resource Description Framework (RDF)/Web Ontology Language (OWL)-based ontology that formally defines the key components of agentic patterns and architectures such as agents, goals, tasks, tools, resources, and workflows, as well as their interrelations. The ontology serves as a standardized vocabulary that enables modular design, reusability, facilitates reasoning over agentic configurations, and supports system auditing and traceability. Accompanying the ontology is a knowledge graph that instantiates agentic patterns and allows integration with heterogeneous frameworks and application contexts.

Building on this foundation, we propose a modular architecture that leverages semantic technologies to bridge the gap between abstract agent design and concrete implementation. Our approach enables the declarative construction, reuse, and analysis of agentic workflows, empowering developers and researchers to design more organized, transparent, and interoperable agentic AI systems.

We validate the feasibility of the proposed ontology and Knowledge Graph (KG) through three real-world use cases: *(i)* declarative reconstruction of agentic AI patterns for visual inspection and reasoning, *(ii)* reusability of agent-task configurations across problem

K-CAP 2025, December 10–12, 2025, Ohio, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9051-4/21/08…$15.00
https://doi.org/10.1145/3465481.3465767

[1]https://www.gartner.com/en/articles/intelligent-agent-in-ai
[2]https://www.deloitte.com/global/en/about/press-room/deloitte-globals-2025-predictions-report.html
[3]https://crewai.com
[4]https://mastra.ai

contexts, and *(iii)* auditing of execution traces for transparency and explainability. These use cases highlight the potential of semantic technologies to serve as a foundational layer for robust and extensible agentic AI development.

To summarize, our **main contributions** of this paper are as follows: *(i)* we provide a comprehensive conceptualization that serves as a foundational model for the development of agentic AI systems[5]; *(ii)* we propose the first standardized conceptual model for agentic AI, formalized as an ontology that captures core concepts, relationships, workflow structures, and agentic interaction patterns;[6] *(iii)* we transform and integrate existing agentic AI patterns from heterogeneous frameworks across various application domains into a unified KGs; and *(iv)* we demonstrate the practical utility of our semantic model through three real-world use cases.

The remainder of this paper is organized as follows: Section 2 provides background agentic AI systems; Section 3 reviews related work in the area of agentic AI system, multi-agent AI approach, semantic workflow modelling and existing agentic AI framework; Section 4 discuss conceptual modelling of agentic AI system and the development of the ontology Section 5 describes the overall architecture and KGs construction pipeline for agentic patterns and illustrates the usefulness of the model by means of three use-case applications; and Section 6 discusses some limitations, concludes, and gives an outlook on future work.

2 BACKGROUND

In this section, we provide a brief introduction to Agentic AI Systems and essential components such as Large Language Models (LLMs) and Retrieval Augmented Generation (RAG).

2.1 Large Language Models (LLMs) and Retrieval Augmented Generation (RAG)

Modern LLMs like GPT-4, Claude, LLaMA, Deepseek, and Gemini have revolutionized the field of artificial intelligence, demonstrating remarkable capabilities in “understanding” and generating human like-text. They show exceptional prowess in tasks ranging from question answering to creative content generation. State-of-the-art LLMs are trained on massive corpora of textual data through self-supervised learning, enabling them to acquire linguistic knowledge and reasoning capabilities without explicit task-specific training [37]. Despite their impressive capabilities, LLMs face significant limitations that constrain their practical utility. Fundamental challenges include *(i)* knowledge cutoffs restricting them to training data [8], *(ii)* hallucinations, producing plausible but incorrect information [14], *(iii)* context windows hampering long-document processing [19], and *(iv)* inability to interact with external tools [21].

RAG has emerged as a prominent technique to address some of these limitations, particularly in accessing and utilizing specific knowledge [18]. RAG makes it possible for LLMs to access external knowledge sources such as vector databases, document stores, or knowledge graphs, retrieving relevant information and incorporating it into the generation process. The core concept behind it involves decomposing the response generation process into three distinct phases: retrieval, augmentation and generation. Figure 1 illustrates the general RAG architecture. Based on a user query, the system first searches external sources such as databases to identify relevant information (*retrieval*), which is then provided as additional context together with the original question (*augmentation*) to the LLM which generates an answer in natural-language (*generation*).

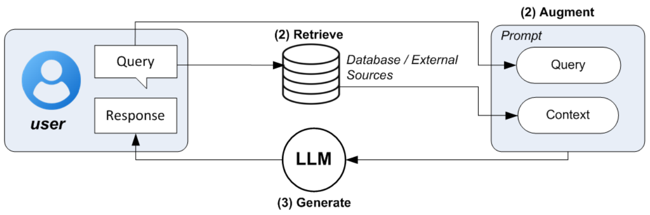

Figure 1: Retrieval Augmented Generation (RAG) Architecture

RAG offers several advantages over traditional LLMs, including *(i)* enhanced factual accuracy by grounding responses in retrieved documents [30], *(ii)* improved knowledge recency by accessing up-to-date information beyond the model's training cutoff date, and *(iii)* better domain adaptability by adjusting the retrieval corpus without retraining the underlying model [17].

2.2 Agentic AI Systems

Agentic AI represents the next evolutionary step beyond RAG systems. Agentic AI Systems consist of autonomous, goal-oriented software agents capable of perceiving their environment, making decisions, and executing actions to achieve specific objectives [22]. Unlike traditional AI pipelines that operate in a single-pass or stateless manner, agentic AI incorporates several elements such as planning, tool usage, reflection, and multi-agent collaboration to enable iterative and adaptive behavior. This paradigm enables agents to move beyond reactive tasks into more strategic and emergent problem solving.

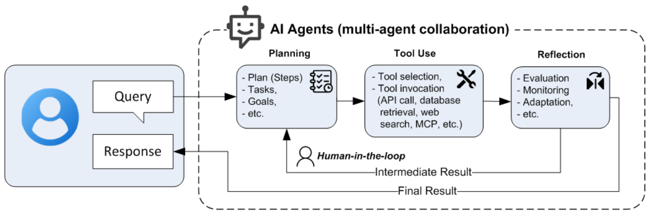

Figure 2: Agentic AI System Architecture

Figure 2 shows an agentic AI architecture. It comprises four prevalent ideas for constructing such intelligent systems [23]: *(i) Planning*: the agent decomposes high-level goals into actionable subtasks, sequences them, and executes them iteratively; *(ii) Tool Use*: the agent leverages external tools or APIs (e.g., web search, databases, MCP, etc.) to extend its capabilities beyond its native reasoning abilities; *(iii) Reflection*: the agent periodically evaluates its progress

[5]Available at: https://github.com/agentic-patterns/agentic-ai-onto.git
[6]Available at: https://w3id.org/agentic-ai/onto

toward a goal, updates its internal memory, and adjusts its plan accordingly; *(iv) Multi-Agent Collaboration*: multiple agents with specialized roles coordinate, communicate, and collaborate to solve complex problems. This collaboration can take several forms such as *Sequential* (agents pass outputs to one another in a defined order), *Parallel* (agents work independently on subtasks simultaneously), and *Nested* (agents spawn sub-agents to handle subtasks hierarchically)[7]. Incorporating human oversight at critical junctures within multi-agent systems (human-in-the-loop) enables the provision of feedback and guidance.

3 RELATED WORK

3.1 Semantic Web

The original semantic web (SW) proposal already envisioned "intelligent agents" performing tasks such as searching for information, completing transactions, and making decisions on behalf of humans [6]. To enable software agents to interact with the web on behalf of their users, explicit semantic representations were supposed to create an environment to (eventually) enable large-scale, agent-based mediation [29]. Although some technical foundations for this vision have been developed, however, the envisioned SW environment for agents never fully materialized. A related line of work focused on adding a semantic layer to traditional web services in order to enable automatic discovery, execution, and composition of web services [7, 20] based on technologies such as WSMO/WSML (Web Service Modeling Ontology) [28, 28], SWSO/SWSL [4, 5], and WSDL-S [2]. This was supposed to enable Agent-based Semantic Web Services [9]. However, at least partly due to the high modeling complexity, the wide range of proposed standards to encode metadata describing machine-readable interfaces, limited tool support, as well as lacking incentives to do so, this vision did not materialize either. A survey of critiques of the Semantic Web in 2020 [13] suggested that more recently, there is a growing recognition that these visions may be made redundant by advances in Machine Learning.

The arrival of LLMs provides strong evidence for this hypothesis, given their impressive ability to process natural language without explicit "machine-readable" semantics. The role of semantic descriptions and agents in this context can now be considered inverted w.r.t. the original SW vision – reasoning agents leveraging knowledge graphs may participate in agentic workflows (e.g., in hybrid artificial intelligence (AI) scenarios), but more commonly, the issue of "machine-interpretability" and interfaces is now largely solved with natural language as a lingua franca. In this context, semantic descriptions are becoming increasingly critical for describing, composing, and coordinating agentic workflows, surpassing their traditional role in representing inputs and outputs or facilitating interoperability – as was the vision for Semantic Web Services initiatives [9]. This fills an important gap, as related recent work on the semantic modeling of agentic workflow patterns (and their dynamic instantiation and composition) in the LLM era is still very scarce, despite a growing recognition that a systems-level perspective is required for designing agentic AI systems and mitigating their risks (including modeling emergent mechanisms, which we leave out of the scope of this paper) [22].

[7]https://github.com/ksm26/AI-Agentic-Design-Patterns-with-AutoGen

3.2 Agentic AI

A wealth of recent work has explored agentic AI systems based on LLMs with the idea to allow LLMs not only to generate text, but also to plan actions and use tools. The ReAct (Reason+Act) paradigm [36], for instance, combines chain-of-thought reasoning with task-specific actions. Building on this idea, the possibility of coordinating agents with heterogeneous knowledge and expertise has further been explored and formalized [11, 12, 31]. Frameworks such as AutoGen [35] allow multiple LLM agents to converse and coordinate to solve tasks. This follows the principle of division of labor, mirroring the concept of teamwork in human collaboration. Other approaches structure agents into specialized roles with defined interaction protocols. ChatDev [27], for example, simulates a software team by assigning different roles (e.g., developer, tester, manager) to LLM agents and orchestrating their communication through a predefined workflow.

Another line of work focuses on extending LLM agents with short-term and long-term memory concepts and planning subsystems [1, 26, 27]. One example is [26], where the agent records each experience in a natural language memory stream, synthesizes higher-level reflections, and retrieves relevant memories to inform its plans. Similarly, Voyager [32] demonstrated an LLM-driven agent that iteratively learns in an environment (Minecraft) by generating code, executing it, and storing new skills in a growing library.

Researchers have also begun to formalise agent interactions: e.g., Google's Agent-to-Agent (A2A)[8] communication protocol and Anthropic's Model-Context Protocol (MCP)[9] aim to define standard interfaces for LLM agents to communicate or use tools. Despite this progress, current academic frameworks remain limited in their semantic expressiveness and reusability. Many systems rely on natural language prompts or hard-coded pipelines to orchestrate agent behavior. There is no widely adopted ontology for describing agent capabilities, goals, or world knowledge – each framework uses its own task definitions and role descriptors, making it difficult to transfer knowledge between systems.

Several frameworks have emerged from both research and industry that put agentic concepts into practice. Among them, AutoGen [35] stands out as a prominent open-source multi-agent framework. It allows agents to communicate in natural language to coordinate on tasks, supporting customizable conversation patterns such as sequential, group and nested chats. This conversation orchestration is centralised in the sense that AutoGen manages the message passing between agents and agents can be powered by tool calls or human input (human-in-the-loop) as needed. CrewAI[10] is an open source Python framework that orchestrates role-playing autonomous AI agents to complete tasks. Each agent in CrewAI has a defined role (with its own tools and focus), and tasks are assigned and coordinated by a management process. CrewAI combines the free-form conversational style of AutoGen with more structured process control inspired by research such as ChatDev [27]. Like AutoGen, the communication schema is largely defined in the framework's code and prompts. Mastra[11] is a cloud-native, open-source

[8]https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/
[9]https://www.anthropic.com/news/model-context-protocol
[10]https://www.crewai.com
[11]https://mastra.ai

TypeScript framework designed for building AI agents with persistent memory and structured workflows. Agents are defined by a name, role-specific instructions (prompts), an LLM model, and optional tools or storage attachments. A standout feature is its support for long-term memory within threads, enabling agents to maintain context across interactions. Mastra also offers a graph-based workflow engine, inspired by state machines, to orchestrate complex LLM operations. While agent behaviors can be modular and reusable, the semantics of these workflows are primarily defined within code. LangGraph [33] is a framework based on the LangChain ecosystem designed for stateful orchestration of LLM agents using graph structures. Developers define workflows as graphs, where nodes represent actions or sub-agents, and edges dictate information flow. This approach models task dependencies and parallelism, offering fine-grained control over execution order. A persistent system state acts as a shared memory, accessible and modifiable by all nodes throughout the workflow. Nodes and edges have to be explicitly defined for each application, with node behaviors specified through Python functions. Another recent framework is Letta[12], which provides an orchestration environment focused on production-ready AI agents with lifecycle management, context sharing, and extensibility. While Letta and similar frameworks address operational aspects of deploying agentic systems, they typically encode semantics and workflows directly in code or proprietary schemas. Finally, other frameworks such as OpenAI's Agents SDK[13] implement a more lightweight approach in which one agent can dynamically delegate specific tasks to other agents.

4 CONCEPTUALIZATION

In this section, we present a conceptual foundation for developing a semantic model for agentic AI systems. Based on Sections 2 and 3, we identify and define essential components, interactions and architectural patterns that characterize agentic AI systems. The goal of this section is to provide a standardized ontology for representing agentic patterns, constructing an agentic pattern KG, and support building agentic AI systems.

4.1 Modelling Agentic AI Systems

We follow a bottom-up conceptual modeling approach [24] guided by a combination of domain analysis, literature review, design patterns and implementation techniques observed in widely used agentic AI frameworks. The conceptualization is carried out through four systematic stages as follows:

1. Component Identification. In this initial stage, we extract core components that appear consistently across existing implementations of agentic AI systems. For example, in the *CrewAI* framework, agents are instantiated with a defined `Role` (e.g., *Writer, Reviewer*, etc.), a specific `Task` (e.g., *Writing literature, Evaluate written content*, etc.), and associated `Tools` (e.g., *SearchTool*, etc.). The identified concepts are then systematically organized and characterized with associated attributes and properties relevant to their function within a system. For example, the `Agent` concept is described using metadata such as `title, description, role`, etc. Figure 3 shows an example of identified concept and their attributes.

[12]https://www.letta.com
[13]https://github.com/openai/openai-agents-python

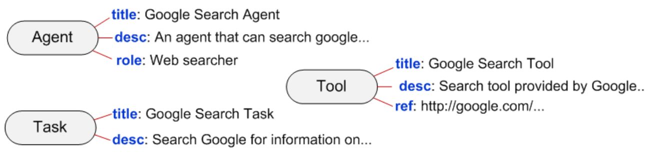
Agent
title: Google Search Agent
desc: An agent that can search google...
role: Web searcher
Tool
title: Google Search Tool
desc: Search tool provided by Google..
ref: http://google.com/...
Task
title: Google Search Task
desc: Search Google for information on...

Figure 3: Agentic AI concepts and their attributes (excerpt)

2. Relationship Modeling. Once the core components are identified, we define the key relationships and dependencies between these concepts, forming a coherent semantic network. These relationships are critical for modeling how each concept interacts with others and representing the behavior of an agentic AI System. For example, in agentic frameworks such as *AutoGen or CrewAI*, an `Agent` is typically assigned one or more `Tasks`, where each task represents a concrete action or objective. Agents may also use specific `Tools` (e.g., databases, APIs, web search) to achieve a specific `Goal` set by an agent. To formally capture this interaction, we defined semantic relationships such as: *(i)* `hasTask` - links an `Agent` to one or more `Task`; *(ii)* `usesTool` - captures the tools or external resources that are used by an `Agent` to accomplish `Tasks`. Figure 4 shows an example of agent relationship model.

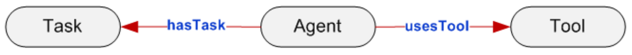

Figure 4: Agentic relationship model (excerpt)

3. Pattern Formalization. In this stage, we capture reusable behavioral patterns that represent recurring sequences of actions, decisions and interactions commonly found in agentic AI systems. These patterns represent abstract workflow or coordination structures that can be reused across tasks, domains, and frameworks. For example, a `literature review` task may involve multiple agents collaborating in a sequential workflow: *(i)* a `Google Search` agent is tasked with searching articles on the web; *(ii)* an `Arxiv Search` agent then refines the search by querying an academic database; finally *(iii)* a `Report` agent compiles and synthesizes the results into a structured report. This sequence *Search* $\longrightarrow$ *Refine* $\longrightarrow$ *Report* forms a **Sequential Pattern**, where the output of one agent serves as the input to the next. To this end we identified three mains workflow patterns i.e., *(i)* Sequential Pattern, a workflow where tasks are executed one after another, in a linear order. *(ii)* Parallel Pattern, involves multiple agents working independently at the same time on different sub-tasks. The results are later combined or aggregated *(iii)* Nested Pattern, refers to workflows where one or more agents invoke internal sub-workflows that can themselves be Sequential, Parallel, or even Nested. Figure 5, Figure 7 and Figure 6 show agentic Pattern workflow composed of identified concepts and relationship mentioned above.

4. Ontology Mapping. This final stage of the conceptualization process involves translating the previously defined model into a formal ontology using Semantic Web Standards based on RDF/OWL. Each core component identified in the *Component Identification* phase — such as `Agent`, `Task`, `Tool`, etc. — is formally defined as a **Class** in the ontology. These classes are enriched with **Data**

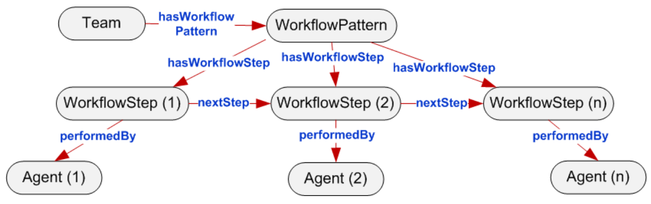

Figure 5: Sequential pattern workflow model example

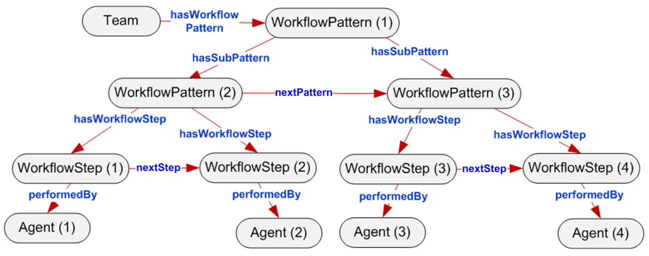

Figure 6: An example of "nested pattern" workflow model

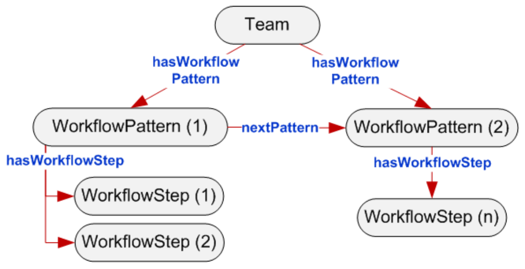

Figure 7: An example of "parallel pattern" workflow model

Properties (e.g., agent `title, description, role`, etc.) that capture attributes and metadata. Meanwhile, the relationships identified during the *Relationship Modelling* phase — such as `hasTask, usesTool, hasGoal, etc.` — are formalized as **Object Properties**, which link instances of one class to another.

4.2 Agentic AI Ontology

Building upon the conceptual model established in the previous section, we propose a formal ontology that defines the structure and semantics of agentic AI systems based on RDF/OWL. This ontology referred to as **AgentO** (Agentic AI Ontology) is designed to enable modular integration and interoperability across agentic frameworks.

Figure 8 shows an overview of *AgentO*, which formalizes the conceptual model and captures the essential components of agentic systems — including `Agent, Task, Tool, Goal, WorkflowPattern, etc.` These components are connected through object properties (e.g., `hasTask`, `usesTool`, `hasGoal`, `nextStep`, etc.) and data properties (e.g., agent `title, description, step order`, etc.). To support various type of agentic patterns (i.e., sequential, parallel, and nested workflow patterns), we introduce a combination of classes and relationships in the ontology.

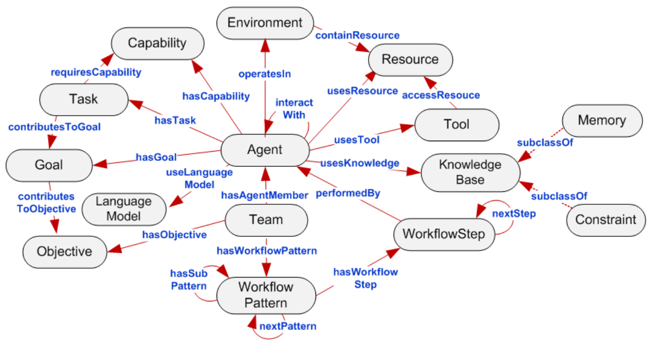

Figure 8: Overview of the Agentic AI Ontology (AgentO)

Specifically, the class `:WorkflowPattern` is used to model composite structures of workflows; individual actions are captured as instances of `:WorkflowStep`. The relationships `:hasWorkflowStep`, `:hasNext`, and `:hasSubPattern` allow us to represent linear sequences, branching parallel flows, and recursive nesting of subpatterns. Due to space limitations, full documentation and the complete ontology specification are provided in an external repository for interested readers.

5 IMPLEMENTATION AND USE CASES

In this section, we discuss the implementation of our approach and present three practical use-cases that demonstrate the utility and expressiveness in real-world scenarios.

5.1 Implementation

To operationalize the ontology and support the retrieval over agentic AI design patterns, we construct KGs that instantiates the ontology with representative patterns extracted from existing agentic AI frameworks. We collected about 50 agentic AI patterns from four popular frameworks i.e., AutoGen, CrewAI, LangGraph, and Mastra AI. Most of these are provided as templates or tutorial examples, officially published on their websites or shared by their respective communities.

Listing 1 shows an example of an AI agent configuration (written in *Python*) for a `literature review` task in the CrewAI framework. The agent is defined with a role, goal, tools and is tasked with a specific objective with expected outputs. These are then assembled into a “crew” which coordinates execution.

```python
from crewai import Agent, Task, Crew
from tools.search_tools import SearchTool
# Define an AI agent to act as a research assistant
literature_reviewer = Agent(
    role="AI Research Assistant",
    goal="Review and summarize recent academic papers on Agentic
      AI...",
    backstory="An expert in AI research who analyzes and
      synthesizes...",
    tools=[SearchTool()], memory=True  )
# Define a task for the agent to perform
review_task = Task(
    description="Identify, retrieve, and summarize at least five
      ...",
    expected_output="A concise literature review summarizing
    objective, methodologies, and unique features of each
      framework.",
    agent=literature_reviewer)
```

```
# Initialize the Crew with the agent and the task
crew = Crew(agents=[literature_reviewer], tasks=[review_task])
crew.kickoff() # Start the process
```

Listing 1: Agentic AI pattern configuration in the CrewAI framework.

Next, we parse the collected aggentic pattern files and extract relevant structures such as agent roles, assigned tasks, tool usage, workflow sequence, etc. We use prompt-guided LLM generation to generate semantically map and transform the preprocessed data into RDF statements based on the agentO schema and extracted context. For example, a pattern containing a `Researcher Agent` using `Google Search Tool` to perform `literature search` would be transformed into triples as shown in Listing 2.

```
@prefix : <http://www.w3id.org/agentic-ai/onto#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

:LiteratureReviewTeam a :Team ;
    dcterms:title "Literature Review Research Team" ;
    :hasWorkflowPattern :ResearchWorkflowPattern ;
...
:ResearchWorkflowPattern a :WorkflowPattern ;
    dcterms:title "Research Workflow Pattern" ;
    :hasStep :GoogleSearchStep, :ArxivSearchStep,
    :reportGenerationStep ;
...
:GoogleSearchStep a :WorkflowStep ;
    dcterms:title "Google Search Step" ;
    :performedBy :GoogleSearchAgent ;
    :nextStep :ArxivSearchStep ;
    :stepOrder 1 ;
...
:GoogleSearchAgent a :Agent ;
    dcterms:description "An agent that can search Google...";
    :hasTask :GoogleSearchTask ;
    :usesTool :GoogleSearchTool ;
...
:GoogleSearchTask a :Task ;
    dcterms:title "Google Search Task" ;
    dcterms:description "Task to search Google for information on
      no-code multi-agent AI tools." ;
...
```

Listing 2: Agentic AI pattern for "Literature Review" in RDF Turtle. (excerpt)

To ensure correctness and adherence to the ontology, we manually review and validate the generated RDF triples, correcting semantic mismatches (e.g., incorrect class assignment), validating property usage, completing missing relationships, etc.

The resulting KGs is deployed to a triple store and made publicly available through a SPARQL endpoint[14]. This provides access to agentic AI patterns for downstream applications such as agent composition, design inspection, and explainability reasoning. For example, developers can query the KG to retrieve all agent configurations involved in a “data analysis” pipeline or inspect common tools used in “report generation” tasks.

5.2 Use-Case Application

Use-Case 1: Agentic AI Pattern Reconstruction. In this use case, we demonstrate how the Agentic-AI Ontology and its associated KGs can be used to reconstruct agentic patterns in a fully declarative manner. This is particularly useful for users who want to understand the characteristics and behaviour of particular agentic

[14]http://w3id.org/agentic-ai/sparql

Table 1: Retrived Agentic-AI Pattern Components

Agent	Task	LLM	Tool	Resource
`:Google Search Agent`	Task to search Google for information on no-code multi-agent AI tools.	gpt-4o	Search Google for information, returns results with a snippet and body content	Google Search Engine
`:Arxiv Search Agent`	Task to search Arxiv for academic papers on no-code multi-agent AI tools.	gpt-4o	Search Arxiv for papers related to a given topic, including abstracts	Arxiv Database
`:Report Agent`	Task to synthesize search results into a literature review report.	gpt-4o	—	—

workflows in solving specific problems. Users can more easily identify the interconnected elements of agentic AI systems (e.g., goal to achieve, the tasks to perform, the tool use.) rather than manually inspecting hard-coded agentic template. By formulating a SPARQL CONSTRUCT query, as shown in Listing 1, users can retrieve and generate a subgraph pattern from existing agentic workflows from the KGs.

```
PREFIX : <http://www.w3id.org/agentic-ai/onto#>
CONSTRUCT {  ?team  :hasAgentMember ?agent ;
                # similar to triple pattern after WHERE clause..
} WHERE { ?team :hasAgentMember ?agent ; :hasObjective ?objective ;
             :hasWorkflowPattern ?workflowPattern .
             ?agent :hasTask ?task ; :hasGoal ?goal .
    OPTIONAL {?workflowPattern :hasWorkflowStep ?workflowStep .
             ?workflowStep :hasAssociatedTask ?task ;
             :performedBy ?agent ; :nextStep ?nextWorkflowStep . }
    OPTIONAL { ?agent :usesTool ?tool .}
    OPTIONAL { ?agent :usesResource ?resource .}
    OPTIONAL { ?tool   :accessesResource ?resource .}
    FILTER (regex(str(?objective),"Travel Planning","i")) }
```

Listing 1: SPARQL Query for Constructing Agentic AI Patterns

The query retrieve and reconstruct a subgraph of an agentic workflow related to a specific objective—in this case, "Travel Planning"—including the agents, their tasks, goals, tools, and resources involved from the KGs.

Figure 9 shows a graph visualization of the resulting query in Listing 1. It highlights how each agent contributes to a shared team objective through assigned tasks and workflow steps. The generated subgraph shows the connected elements of agentic patterns necessary for achieving the "Travel Planning" objective.

Use-Case 2: Reusability of Agentic AI components. This use case showcases how the Agentic AI Ontology and its KGs facilitate the discovery and reuse of agentic components such as tasks, agents, tools, and resources for addressing complex problem scenarios. For example, a researcher may need to build an agentic AI system to automate a *literature review* task. Instead of designing the agents and their configurations from scratch, they can query the KG and reuse the existing concepts and agentic patterns.

As shown in Listing 2, the formulated SPARQL query enables users to retrieve previously instantiated Agentic-AI pattern compositions and configurations aligned with a specific objective description. The ability to semantically query reusable components based on high-level objectives provides modularity, reduces duplication, and accelerates system development in agentic AI workflows. Table 1 shows the result with three agent definitions along with their

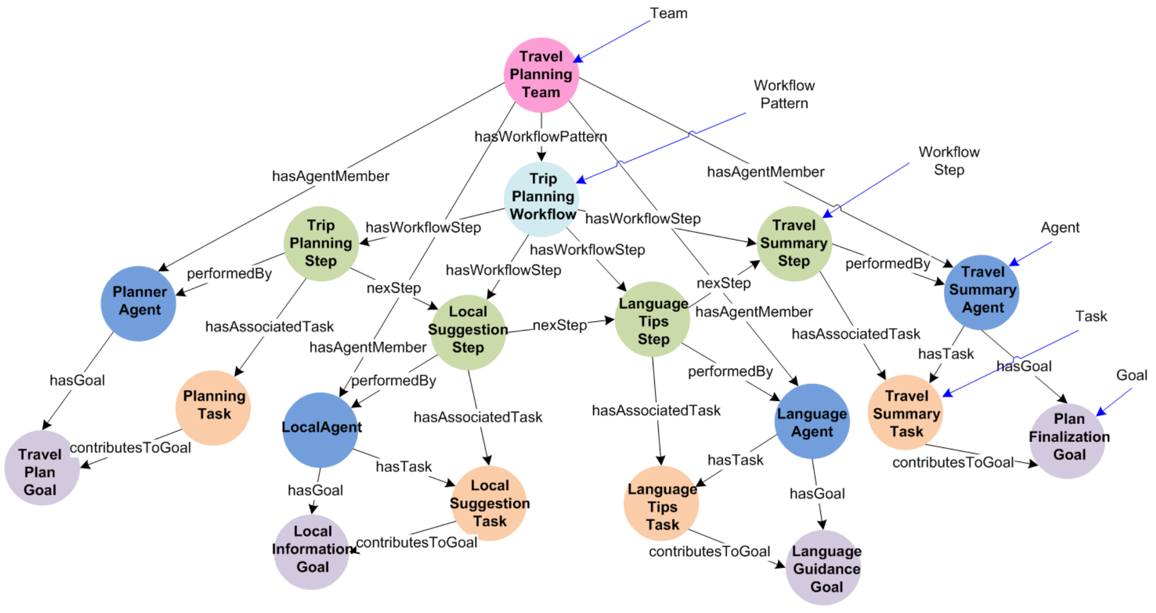

Figure 9: Visualization of constructed agentic patterns for Travel Planning

```
PREFIX : <http://www.w3id.org/agentic-ai/onto#>
PREFIX dcterms: <http://purl.org/dc/terms/>
SELECT DISTINCT ?agent ?dtask ?dlm ?dtool ?dresource
WHERE { # Find team based the objective
  ?team :hasObjective ?objective . ?objective dcterms:description
  ↪  ?Objdesc .
  # Find agent and task related to team,
  ?team :hasAgentMember ?agent . ?agent :hasTask ?task .
  ?task dcterms:description ?dtask
  # Find language model used by agents
  OPTIONAL { ?agent :usesLanguageModel ?lm . ?lm dcterms:hasVersion
  ↪  ?dlm}
  # Find tools/resource used by agents
  OPTIONAL { ?agent :usesTool ?tool . ?tool dcterms:description
  ↪  ?dtool}
  OPTIONAL { ?tool :accessesResource ?resource . ?resource
  ↪  dcterms:description ?dresource}
FILTER(regex(?Objdesc, "Literature Review", "i" )) }
```

Listing 2: SPARQL Query for existing agentic patterns for an objective

associated tasks, language models, tools, and resources required to perform "Literature Review".

Use-Case 3: Agentic AI Implementation Auditing. In this use case, we demonstrate how the Agentic AI Ontology and its KGs enable the auditing of agent-driven workflow pattern, i.e., to support explainability, transparency, and traceability in multi-agent AI systems. The SPARQL query in Listing 3 targets a specific workflow pattern e.g., `:FraudDetectionWorkflowPattern` and retrieves the sequence of workflow steps `:hasWorkflowStep` along with their order, associated task descriptions, responsible agents, and optionally the tools used.

The result, illustrated in Section 5.2, presents a traceable view of the AI process, detailing a multi-step investigation involving a

```
PREFIX : <http://www.w3id.org/agentic-ai/onto#>
PREFIX dcterms: <http://purl.org/dc/terms/>
SELECT distinct ?step ?stepOrder ?taskDescription ?agent ?dtool
WHERE { :FraudDetectionWorkflowPattern :hasWorkflowStep ?step .
        ?step :stepOrder ?stepOrder ; :hasAssociatedTask ?task ;
              :performedBy ?agent .
        ?task dcterms:description ?taskDescription .
    OPTIONAL { ?agent :usesTool ?tool . ?tool dcterms:title ?dTool .
    ↪  } }
ORDER BY ?stepOrder
```

Listing 3: SPARQL Query to investigate a specific workflow pattern

financial forensics analyst, compliance officer, and risk assessment analyst.

Table 2: Task Steps, Descriptions, and Associated Agents for the FraudDetectionWorkflowPattern

Step	Step Order	Task Description	Agent	Tool
:Financial Forensics Step	1	Analyze the transactional behaviors and patterns for {customer_name} ({customer_info}) to identify...	:Financial Forensics Analyst	Serper Dev Tool
:Compliance Step	2	Conduct a thorough review of compliance for {customer_name} ({customer_info}) within the {topic} sector. Evaluate adherence to legal standards and internal policies...	:Compliance Officer	Scrape Website Tool
:Risk Assessment Step	3	Perform an extensive risk assessment for {customer_name} ({customer_info}), analyzing their profile...	:Risk Assessment Analyst	Serper Dev Tool

6 CONCLUSIONS AND FUTURE WORK

In this paper, we introduced an ontology and the accompanying KG to address the limitations in designing current agentic AI systems, which are often implemented in an ad-hoc, monolithic manner. By capturing the fundamental concepts such as agents, tasks, goals, and interactions in a standardized RDF/OWL based representation, our approach facilitates modularity, interoperability, and reusability across diverse agentic AI frameworks. Through three use cases, we demonstrated how our ontology enables the declarative construction of agentic AI workflows, the reuse of existing agent-task configurations for new problems, and the auditing of agentic AI implementations for explainability and traceability. Future work will focus on extending the ontology to cover dynamic execution traces of agentic AI systems, enabling semantic representation of runtime behaviors and decision flows. Additionally, we aim to develop an open-source software that can (semi-automatically) generate agentic AI templates tailored to specific frameworks. For example, depending on users choice regarding the targeted framework.

ACKNOWLEDGMENTS

SBA Research (SBA-K1 NGC) is a COMET Center within the COMET – Competence Centers for Excellent Technologies Programme and funded by BMIMI, BMWET, and the federal state of Vienna. The COMET Programme is managed by FFG. This research was also funded in part by the Austrian Science Fund (FWF) 10.55776/COE12.

REFERENCES

[1] Aadharsh Aadhithya A, Sachin Kumar S, and Soman K. P. 2024. Enhancing Long-Term Memory using Hierarchical Aggregate Tree for Retrieval Augmented Generation. arXiv:2406.06124 [cs.CL] https://arxiv.org/abs/2406.06124

[2] Rama Akkiraju, Joel Farrell, John A Miller, Meenakshi Nagarajan, Amit P Sheth, and Kunal Verma. 2005. Web service semantics-wsdl-s. (2005). https://corescholar.libraries.wright.edu/knoesis/69

[3] Batin Latif Aylak. 2025. SustAI-SCM: Intelligent Supply Chain Process Automation with Agentic AI for Sustainability and Cost Efficiency. *Sustainability* 17, 6 (2025). https://doi.org/10.3390/su17062453

[4] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, et al. 2005. Semantic web services language (SWSL). *W3C Member Submission* 9 (2005), 1–61.

[5] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, et al. 2005. Semantic web services ontology (swso). *Member submission, W3C* (2005).

[6] Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The semantic web. *Scientific American* 284, 5 (2001), 34–43.

[7] Liliana Cabral, John Domingue, Enrico Motta, Terry Payne, and Farshad Hakimpour. 2004. Approaches to Semantic Web Services: an Overview and Comparisons. In *The Semantic Web: Research and Applications*, Christoph J. Bussler, John Davies, Dieter Fensel, and Rudi Studer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 225–239.

[8] Jeffrey Cheng, Marc Marone, Orion Weller, Dawn Lawrie, Daniel Khashabi, and Benjamin Van Durme. [n. d.]. Dated Data: Tracing Knowledge Cutoffs in Large Language Models. In *First Conference on Language Modeling*.

[9] Nicholas Gibbins, Stephen Harris, and Nigel Shadbolt. 2003. Agent-based semantic web services. In *Proceedings of the 12th international conference on World Wide Web*. 710–717.

[10] Mourad Gridach, Jay Nanavati, Khaldoun Zine El Abidine, Lenon Mendes, and Christina Mack. 2025. Agentic ai for scientific discovery: A survey of progress, challenges, and future directions. *arXiv preprint arXiv:2503.08979* (2025).

[11] Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhuo Xu, and Chaoyang He. 2024. LLM Multi-Agent Systems: Challenges and Open Problems. arXiv:2402.03578 [cs.MA] https://arxiv.org/abs/2402.03578

[12] Rui Hao, Linmei Hu, Weijian Qi, Qingliu Wu, Yirui Zhang, and Liqiang Nie. 2023. ChatLLM Network: More brains, More intelligence. arXiv:2304.12998 [cs.AI] https://arxiv.org/abs/2304.12998

[13] Aidan Hogan. 2020. The semantic web: two decades on. *Semantic Web* 11, 1 (2020), 169–185.

[14] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.* 55, 12, Article 248 (March 2023), 38 pages. https://doi.org/10.1145/3571730

[15] Nalan Karunanayake. 2025. Next-generation agentic AI for transforming healthcare. *Informatics and Health* 2, 2 (2025), 73–83. https://doi.org/10.1016/j.infoh.2025.03.001

[16] Nir Kshetri. 2025. Transforming cybersecurity with agentic AI to combat emerging cyber threats. *Telecommunications Policy* (2025), 102976. https://doi.org/10.1016/j.telpol.2025.102976

[17] Kabul Kurniawan, Elmar Kiesling, and Andreas Ekelhart. 2024. CyKG-RAG: Towards knowledge-graph enhanced retrieval augmented generation for cybersecurity. (2024).

[18] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[19] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. https://doi.org/10.1162/tacl_a_00638

[20] Sheila A McIlraith and David L Martin. 2005. Bringing semantics to web services. *IEEE Intelligent systems* 18, 1 (2005), 90–93.

[21] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented Language Models: a Survey. *arXiv e-prints* (2023), arXiv–2302.

[22] Erik Miehling, Karthikeyan Natesan Ramamurthy, Kush R Varshney, Matthew Riemer, Djallel Bouneffouf, John T Richards, Amit Dhurandhar, Elizabeth M Daly, Michael Hind, Prasanna Sattigeri, et al. 2025. Agentic AI needs a systems theory. *arXiv preprint arXiv:2503.00237* (2025).

[23] Andrew Ng. 2024. Agentic Design Patterns Part 1.

[24] Natalya F Noy, Deborah L McGuinness, et al. 2001. Ontology development 101: A guide to creating your first ontology.

[25] Izunna Okpala, Ashkan Golgoon, and Arjun Ravi Kannan. 2025. Agentic AI Systems Applied to tasks in Financial Services: Modeling and model risk management crews. *arXiv preprint arXiv:2502.05439* (2025).

[26] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. arXiv:2304.03442 [cs.HC] https://arxiv.org/abs/2304.03442

[27] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative Agents for Software Development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 15174–15186. https://doi.org/10.18653/v1/2024.acl-long.810

[28] Dumitru Roman, Uwe Keller, Holger Lausen, Jos De Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. 2005. Web service modeling ontology. *Applied ontology* 1, 1 (2005), 77–106.

[29] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. 2006. The semantic web revisited. *IEEE intelligent systems* 21, 3 (2006), 96–101.

[30] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval Augmentation Reduces Hallucination in Conversation. *Findings of the Association for Computational Linguistics: EMNLP 2021* (2021).

[31] Yashar Talebirad and Amirhossein Nadiri. 2023. Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents. arXiv:2306.03314 [cs.AI] https://arxiv.org/abs/2306.03314

[32] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291 [cs.AI] https://arxiv.org/abs/2305.16291

[33] Jialin Wang and Zhihua Duan. 2024. Agent AI with LangGraph: A Modular Framework for Enhancing Machine Translation Using Large Language Models. arXiv:2412.03801 [cs.CL] https://arxiv.org/abs/2412.03801

[34] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[35] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155* (2023).

[36] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[37] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* 1, 2 (2023).