AgentO: An Ontology for Modeling Agentic AI Systems

Kabul Kurniawan*[1][0000−0002−5353−7376], Fajar J. Ekaputra[2][0000−0003−4569−2496], Elmar Kiesling[2][0000−0002−7856−2113], and Andreas Ekelhart[3,4][0000−0003−3682−1364]

[1] Universitas Gadjah Mada, Yogyakarta, Indonesia
[2] Vienna University of Economics and Business, Austria
[3] SBA Research, Favoritenstraße 16, Vienna, Austria
[4] University of Vienna, Austria

Abstract. Agentic AI systems are rapidly being deployed as autonomous, goal-directed entities to manage the orchestration of complex, multi-step workflows across diverse domains. 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 AgentO, 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. To build and evaluate AgentO, we developed an automated LLM-driven process and translated 66 agentic workflows from four different agentic AI frameworks. We further evaluated our approach through three real-world use cases: declarative reconstruction of agentic patterns, cross-context reuse of tasks and agents, and agentic AI workflow 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
Resource type: Ontology

DOI: 10.5281/zenodo.18342624
URL: https://w3id.org/agentic-ai/onto

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

* Corresponding Author {first.last}@ugm.ac.id

cybersecurity[16], finance [22], industrial automation[3], and healthcare[15] to scientific research[11]. These systems promise enhanced flexibility, coordination, and scalability by delegating tasks to intelligent agents capable of interacting, reasoning, and responding to dynamic environments [30].

The year 2025 is 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[5] and expect large growth of the global agentic AI market in the next years[6].

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 [31] CrewAI[7], LangGraph [29] and Mastra AI[8], 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 AgentO, an Ontology for modeling agentic AI systems[9]. The ontology provide concepts of agentic AI systems and formally defines the key components of agentic patterns such as agents, goals, tasks, tools, resources as well as their interrelations and workflows. The ontology serves as a standardized vocabulary (built based on Resource Description Framework (RDF)/Web Ontology Language (OWL)) 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 existing agentic patterns and allows integration with heterogeneous agentic AI frameworks and application contexts. This resource 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 contexts, and *(iii)* auditing of execution traces for transparency and explainability.

To summarize, our **main contributions** of this paper are as follows: *(i)* we propose the first standardized conceptual model for agentic AI as a reusable

[5] https://www.gartner.com/en/articles/intelligent-agent-in-ai
[6] https://www.deloitte.com/global/en/about/press-room/deloitte-globals-2025-predictions-report.html
[7] https://crewai.com
[8] https://mastra.ai
[9] https://w3id.org/agentic-ai/onto

resource, formalized as an ontology that captures core concepts, relationships, workflow structures, and agentic interaction patterns;[10] [11] *(ii)* we transform and integrate 66 existing agentic AI patterns from heterogeneous frameworks across various application domains into a unified KG and share the Large Language Model (LLM)-based pipeline[12], and *(iii)* 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 presents related work in the area of agentic AI systems and Semantic Web; Section 3 discusses the conceptual modeling and the development of the ontology; Section 4 describes the overall KG construction and refinement; Section 5 presents three use-case applications; and Section 6 discusses limitations, concludes, and gives an outlook on future work.

2 Related Work

2.1 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 [26]. 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 [18,7] based on technologies such as WSMO/WSML (Web Service Modeling Ontology) [25], SWSO/SWSL [5,4], and WSDL-S [2]. This was supposed to enable Agent-based Semantic Web Services [10]. Due to the high modeling complexity, the wide range of proposed standards, 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 [14] 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

[10] Available at: https://w3id.org/agentic-ai/onto
[11] Available at: https://github.com/agentic-patterns/agentic-ai-onto.git
[12] Available at: https://github.com/agentic-patterns/agentic-ai-onto.git

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 [10].

2.2 Agentic AI 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 [19]. 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.

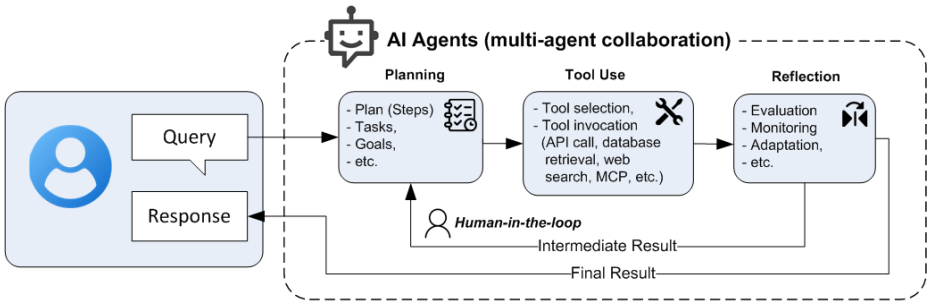

Fig. 1: Agentic AI System Architecture

Figure 1 shows an agentic AI architecture, which comprises four prevalent ideas for constructing such intelligent systems [20]: *(i) Planning*: the agent decomposes high-level goals into actionable sub-tasks, 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 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. Incorporating human oversight at critical junctures within multi-agent systems (human-in-the-loop) enables the provision of feedback and guidance.

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 [32], 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 [27,12,13]. Frameworks such as AutoGen [31] 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 [24], 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 [23,1,24]. One example is [23], 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 [28] 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)[13] communication protocol and Anthropic's Model-Context Protocol (MCP)[14] 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 [31] 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. CrewAI[15] is an open source Python framework that orchestrates role-playing autonomous AI agents to complete tasks. Mastra[16] is a cloud-native, open-source TypeScript framework designed for building AI agents with persistent memory and structured workflows, and LangGraph [29], a framework based on the LangChain ecosystem designed for stateful orchestration of LLM agents using graph structures. We will focus on these four popular frameworks in this work.

3 Conceptualization

In this section, we present a conceptual foundation for developing a semantic model for agentic AI systems. Based on Section 2, we identify and define essential components, interactions and architectural patterns that characterize agentic

[13] https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/
[14] https://www.anthropic.com/news/model-context-protocol
[15] https://www.crewai.com
[16] https://mastra.ai

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.

3.1 Modelling Agentic AI Systems

We follow a bottom-up conceptual modeling approach [21] 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:

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.), specific `Tasks` (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 2 shows examples of identified concepts and their attributes.

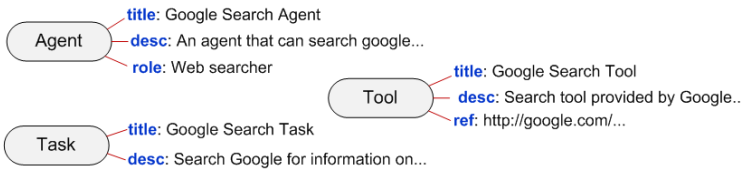

Fig. 2: Agentic AI concepts and their attributes (excerpt)

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 performs 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)* `responsibleAgent` - links a `Task` with a responsible `LLMAgent`; *(ii)* `performedBy` - captures the tools or external resources that are used by an `LLMAgent` to accomplish `Tasks`. Figure 3 shows an example of the agent relationship model.

Fig. 3: Agentic relationship model (excerpt)

3. Pattern Identification 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* → *Refine* → *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 nested. Figure 4 shows agentic patterns composed of the identified concepts and relationships. (See Apendix for other patterns, i.e., Figure 8 and Figure 7)

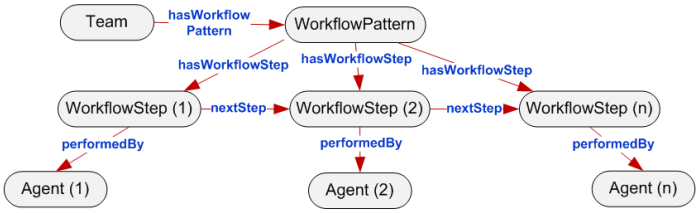

Fig. 4: Sequential pattern workflow model example

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 Properties** (e.g., agent `title, description, role`, etc.) that capture attributes and metadata. Meanwhile, the relationships identified during the *Relationship Modelling* phase — such as `hasAssociatedTask, performedBy,`

`hasAgentGoal` — are formalized as **Object Properties**, which link instances of one class to another.

3.2 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. To this end, we built upon existing concepts from existing ontologies, specifically PROV-O [17](i.e., `prov:Agent`), P-Plan [9] (`pplan:Plan` and `pplan:Step`), and BEAM [8] (`beam:System`,`beam:Context`and `beam:Resource`).

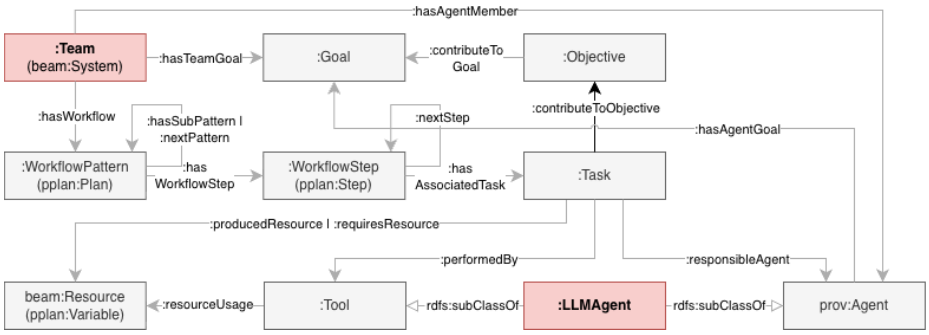

Fig. 5: Overview of the Agentic AI Ontology (AgentO)

Figure 5 shows an overview of the core classes *AgentO* and their relations, capturing the essential components of agentic systems including `Team, Agent, Task, Goal, Objective, WorkflowPattern, WorkflowStep, Tool`, and `Resource` To support various types of agentic patterns (i.e., sequential, parallel, and nested), we introduce a combination of classes and relationships in the ontology. Specifically, the class `:WorkflowPattern` is used to model composite structures of workflows; individual actions are captured as instances of `:WorkflowStep`. The relationships `:hasWorkflowStep`, `:hasNextStep`, and `:hasSubPattern` allow us to represent linear sequences, branching parallel flows, and recursive nesting of sub-patterns. Due to space limitations, full documentation and the complete ontology specification are provided online[17].

4 Knowledge Graph Construction and Refinement

To operationalize the ontology and support the retrieval over agentic AI design patterns, we construct a KG that instantiates the ontology with representative

[17] Agentic AI Ontology (AgentO): http://w3id.org/agentic-ai/onto

patterns extracted from existing agentic AI frameworks. We designed an automated process including LLMs to populate the ontology and discover current issues and potential extensions. The results were then manually evaluated and used to refine the ontology. The KG construction and refinement process consisted of several stages:

1. Source Pattern Collection We collected 66 existing agentic AI templates and design patterns from several framework repositories (AutoGen[18], CrewAI[19], LangGraph[20], and Mastra AI[21]). An overview can be found in Table 6. These templates are typically hardcoded configurations within *Python* or *JavaScript* files. Listing 1.1 shows an illustrative 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 the execution.

```
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 1.1: Agentic AI pattern configuration in the CrewAI framework

2. Pattern Extraction and Ontology Mapping Next, we developed an automated workflow[22] to parse the raw agentic pattern files and related source code to extract relevant concepts. We use prompt-guided LLM generation to semantically map and transform the framework patterns into RDF statements based on the AgentO schema. Since the frameworks use different structures, all source code files from each pattern were passed as context to the LLM. All extractions were done with `gpt-5-mini` via API access. The extraction time per pattern varied between 47.72 and 166.60 seconds. Approximately 2.8 million input and 570,000 output tokens were processed for parsing the 66 patterns[23].

[18] https://github.com/ksm26/AI-Agentic-Design-Patterns-with-AutoGen 6 patterns
[19] https://github.com/crewAIInc/crewAI-examples 16 patterns
[20] https://github.com/langchain-ai/langgraphjs-gen-ui-examples 9 patterns
[21] https://github.com/mastra-ai/mastra 35 patterns
[22] The source code is available here: https://github.com/kabulkurniawan/agenticPatternExtractor
[23] The API costs for transforming the 66 patterns to AgentO totaled $2.72

A consistent output structure was generated for each pattern translation, including execution time and the used model, issues and assumptions from the LLM, followed by the instance definition in .ttl format. In Listing 1.2 we show an excerpt of the CrewAI trip planner output as an example.

```
# Execution time: 107.83 seconds
# Model used: gpt-5-mini

# Issues / Assumptions:
# - The ontology lacks explicit properties for backstory, verbose flags, tool endpoints, and agent prompt/backstory fields. These are represented using dcterms:description literals on Agent and Tool individuals.
# - There is no dedicated property to link an Agent to the API keys or environment variables;
...

@prefix : <http://www.w3id.org/agentic-ai/onto#> .
...

:TripPlannerCrew rdf:type :Team ;
    dcterms:title "Trip Planner Crew" ;
    dcterms:description "A team composed of multiple agent roles (city selector, local expert, travel concierge) collaborating to identify a city, gather local knowledge, and produce a 7-day travel itinerary." ;
    :hasAgentMember :Agent_CitySelection ,
                    :Agent_LocalExpert ,
                    :Agent_TravelConcierge ;
    :hasWorkflowPattern :Workflow_TripPlanning .
...
```

Listing 1.2: Translation output for the Crew AI trip planner (abbreviated)

3. Human-in-the-Loop Verification and Refinement Next, we manually selected and reviewed 6 translated patterns from each framework (24 in total). This included a review of the results, and a discussion of the identified issues. For changes we considered helpful, we extended the ontology. In Table 1 we show an excerpt of the 27 common issues we discovered and how we addressed them.

Table 1: Identified issues and resolutions during ontology refinement (excerpt)

#	Source	Issue	Resolution
1	All	Missing axioms: Prompt & Config Classes and properties; temporal properties	Add Prompt class and relevant properties (e.g., prompt, role); introduce a Config class (runtime/design-time).
2	CrewAI, AutoGen, Mastra	Missing axioms: datatype properties for Agent runtime (e.g., allow_delegation, verbose).	We will not model these implementation details.
3	All	Confusion between Agent and Tool.	Tool is a superclass of LLMAgent.
4	All	No structured representation for function calls, loops, or invocation semantics.	We will not model these implementation details.

After the extension of the ontology, we re-ran the automated extraction process with the updated ontology and compared them to the initial round. The new concepts and changes were picked up and integrated.

4. KG Generation and Publishing The result triples from the translations were compiled into a coherent RDF graph. The resulting KG is in a triple store and publicly available through a SPARQL endpoint[24]. This provides access to the agentic AI patterns for downstream applications such as agent composition, design inspection, and explainability reasoning.

5 Use Cases Application

In this section, we present three practical use-cases that demonstrate the utility and expressiveness of AgentO in real-world scenarios. The KG 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[25].

5.1 Use-Case Application

```
PREFIX :    <http://www.w3id.org/agentic-ai/onto#>
PREFIX dct: <http://purl.org/dc/terms/>

CONSTRUCT {  ?team  :hasAgentMember    ?agent ;
             # similar to triple pattern after WHERE clause..
} WHERE {?team a :Team ; :hasAgentMember    ?agent ;
                  :hasWorkflowPattern ?workflowPattern ;
                   dct:title   ?teamTitle .
         ?agent :hasAgentGoal ?goal .
  OPTIONAL { ?agent :agentToolUsage ?tool . }
  OPTIONAL { ?workflowPattern :hasWorkflowStep ?workflowStep .
             ?workflowStep :hasAssociatedTask ?task ;
                           :relatedStep   ?nextWorkflowStep .
             ?task :performedByAgent ?agent .
   OPTIONAL { ?task :requiresResource ?resource . }
  } FILTER (regex(str(?teamTitle), "Trip Planner", "i"))
}
```

Listing 1: SPARQL Query for Constructing Agentic AI Patterns

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

[25] An excerpt of the generated KG in RDF turtle format can be found in Listing 1.3 in the Appendix.

Use-Case 1: Agentic AI Pattern Reconstruction In this use case, we demonstrate how the AgentO Ontology and its associated KG 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 workflows in solving specific problems. Users can more easily identify the interconnected elements of agentic AI systems (e.g., goals, the tasks to perform, tool use) rather than manually inspecting hard-coded agentic templates and source code. By formulating a SPARQL CONSTRUCT query, as shown in Listing 1, users can retrieve and generate a subgraph pattern from existing agentic workflows.

The query retrieves and reconstructs a subgraph of an agentic workflow related to a specific objective, in this case, "Trip Planner"—including the agents, their tasks, goals, tools, and resources involved. Figure 6 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.

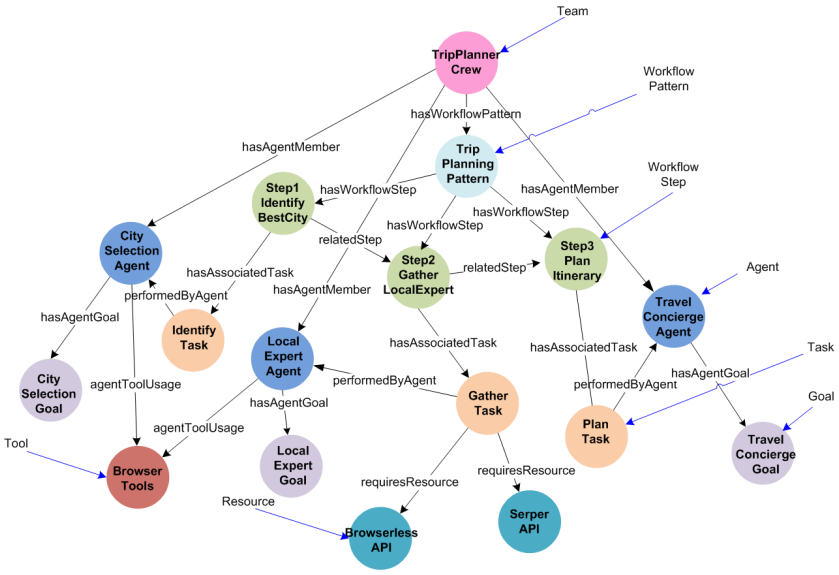

Fig. 6: Visualization of the constructed agentic pattern for Travel Planning

Use-Case 2: Reusability of Agentic AI components This use case showcases how the AgentO Ontology and its KG facilitate the discovery and reuse of agentic components such as tasks, agents, tools, and resources for addressing complex problem scenarios. For example, a travel application may require

a system that selects the best destination for a traveler, gathers local insights, and composes a detailed multi-day itinerary. Instead of designing the agents and their configurations from scratch, we 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. 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 2 shows the result with three agent definitions along with their associated tasks, language models, tools, and resources required to perform trip planning.

```
PREFIX : <http://www.w3id.org/agentic-ai/onto#>
PREFIX dct: <http://purl.org/dc/terms/>

SELECT DISTINCT ?agent ?dtask ?dlm ?dtool ?dresource
WHERE { ?team a :Team ; dct:description ?Objdesc ;
                :hasAgentMember ?agent .
        ?task :performedByAgent ?agent ; dct:description ?dtask .
        OPTIONAL {  ?agent :useLanguageModel ?lm .
                    ?lm dct:description ?dlm . }
        OPTIONAL {  ?agent :agentToolUsage ?tool.
                     ?tool dct:description ?dtool .}
        OPTIONAL {   { ?tool :resourceUsage ?resource } UNION
                      { ?task :requiresResource ?resource }
                  ?resource dct:description ?dresource .}
        FILTER(regex(?Objdesc, "Trip", "i"))
}
```

Listing 2: SPARQL Query for existing agentic patterns

Table 2: Retrieved Agentic-AI Pattern Components for Travel Agents

Agent	Task	LLM	Tool	Resource
:CitySelection Agent	Analyze and choose best city for trip based on weather, events, and travel cost	OpenAI LLM client	Internet search (Serper)	Organic search results
:LocalExpert Agent	Create in-depth city guide with attractions, culture, hidden gems, and tips	OpenAI LLM client	Internet search (Serper)	Organic travel insights
:TravelConcierge Agent	Expand guide into 7-day itinerary: daily plans, hotels, restaurants, budget	OpenAI LLM client	Internet search + itinerary planning tools	Real-world travel info

Use-Case 3: Agentic AI Implementation Auditing In this use case, we demonstrate how the Agentic AI Ontology and its KG enable the auditing of agent-driven workflow patterns, 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., `:RecruitmentWorkflow` and retrieves the sequence of workflow steps `:hasWorkflowStep` along with their order, associated task descriptions, responsible agents, and optionally the tools used.

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

SELECT DISTINCT ?step ?stepOrder ?taskDescription ?agent ?tool
WHERE { :RecruitmentWorkflow :hasWorkflowStep ?step .
        ?step :stepOrder ?stepOrder ; :hasAssociatedTask ?task .
        ?task  :performedByAgent ?agent .
    ?task dcterms:description ?taskDescription .
   OPTIONAL {?agent :agentToolUsage ?tool .}
} ORDER BY ?stepOrder
```

Listing 3: SPARQL Query to investigate a specific workflow pattern

The results, illustrated in Table 3, present a traceable view of the AI-driven recruitment process, detailing a multi-step workflow including researching job candidates, evaluating suitability, engaging selected candidates, and summarizing hiring recommendations.

Table 3: Retrieved Agentic-AI Workflow Components: Recruitment Use Case

Step	Order	Task	Agent	Tool
`ws_research`	1	Research job candidates using online sources; ensure requirement match	Researcher Agent	SerperDev Search API
`ws_match_and_score`	2	Evaluate and score candidate-job suitability	Matcher Agent	SerperDev Search API
`ws_outreach`	3	Engage and reach selected candidates	Communicator Agent	SerperDev Search API
`ws_report`	4	Summarize findings and recommend best candidates	Reporter Agent	—

6 Conclusions and Future Work

In this paper, we introduced the AgentO 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 additional aspects, including dynamic execution traces of agentic AI systems, enabling semantic representation of runtime behaviors and decision flows. We will continue the development and maintenance of the ontology as a sustained effort within the BILAI (Bilateral AI Clusters of Excellence) project, which will run until autumn 2029. In addition, as part of another research project, we aim to develop and maintain an open-source software that can (semi-automatically) translate AgentO workflows to specific target frameworks. Users can then model agentic workflows independently using AgentO and select a target framework for execution. We further plan to use AgentO as basis for the automated construction of agentic workflows from natural-language specifications.

Acknowledgments

This research is funded by the Indonesian Endowment Fund for Education (LPDP) on behalf of the Indonesian Ministry of Higher Education, Science and Technology and managed under the EQUITY Program (Contract Number: 4301/B3/DT.03.08/2025 and 10107/UN1.P/Dit-Keu/HK.08.00/2025).

This work was supported by the Austrian Science Fund (FWF) Bilateral AI projects (Grant Nr. 10.55776/COE12) and the Austrian Research Promotion Agency (FFG) FAIR-AI project (Grant Nr. FO999904624). 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.

References

1. A, A.A., S, S.K., P, S.K.: Enhancing long-term memory using hierarchical aggregate tree for retrieval augmented generation (2024), https://arxiv.org/abs/2406.06124
2. Akkiraju, R., Farrell, J., Miller, J.A., Nagarajan, M., Sheth, A.P., Verma, K.: Web service semantics-wsdl-s (2005), https://corescholar.libraries.wright.edu/knoesis/69
3. Aylak, B.L.: 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, https://www.mdpi.com/2071-1050/17/6/2453

4. Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., et al.: Semantic web services language (swsl). W3C Member Submission **9**, 1–61 (2005)
5. Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., et al.: Semantic web services ontology (swso). Member submission, W3C (2005)
6. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American **284**(5), 34–43 (2001)
7. Cabral, L., Domingue, J., Motta, E., Payne, T., Hakimpour, F.: Approaches to semantic web services: an overview and comparisons. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) The Semantic Web: Research and Applications. pp. 225–239. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
8. Ekaputra, F.J., Prock, A., Kiesling, E.: Towards supporting ai system engineering with an extended boxology notation. In: The 2nd International Workshop on Knowledge Graphs for Responsible AI (KG-STAR) Co-located with the Extended Semantic Web Conference (ESWC 2025), CEUR-WS (2025)
9. Garijo Verdejo, D., Gil, Y.: Augmenting prov with plans in p-plan: scientific processes as linked data. CEUR Workshop Proceedings (2012)
10. Gibbins, N., Harris, S., Shadbolt, N.: Agent-based semantic web services. In: Proceedings of the 12th international conference on World Wide Web. pp. 710–717 (2003)
11. Gridach, M., Nanavati, J., Abidine, K.Z.E., Mendes, L., Mack, C.: Agentic ai for scientific discovery: A survey of progress, challenges, and future directions. arXiv preprint arXiv:2503.08979 (2025)
12. Han, S., Zhang, Q., Yao, Y., Jin, W., Xu, Z., He, C.: Llm multi-agent systems: Challenges and open problems (2024), https://arxiv.org/abs/2402.03578
13. Hao, R., Hu, L., Qi, W., Wu, Q., Zhang, Y., Nie, L.: Chatllm network: More brains, more intelligence (2023), https://arxiv.org/abs/2304.12998
14. Hogan, A.: The semantic web: two decades on. Semantic Web **11**(1), 169–185 (2020)
15. Karunanayake, N.: Next-generation agentic ai for transforming healthcare. Informatics and Health **2**(2), 73–83 (2025). https://doi.org/https://doi.org/10.1016/j.infoh.2025.03.001, https://www.sciencedirect.com/science/article/pii/S2949953425000141
16. Kshetri, N.: Transforming cybersecurity with agentic ai to combat emerging cyber threats. Telecommunications Policy p. 102976 (2025). https://doi.org/https://doi.org/10.1016/j.telpol.2025.102976, https://www.sciencedirect.com/science/article/pii/S0308596125000734
17. Lebo, T., Sahoo, S., McGuinness, D., Belhajjame, K., Cheney, J., Corsar, D., Garijo, D., Soiland-Reyes, S., Zednik, S., Zhao, J.: Prov-o: The prov ontology (2013)
18. McIlraith, S.A., Martin, D.L.: Bringing semantics to web services. IEEE Intelligent systems **18**(1), 90–93 (2005)
19. Miehling, E., Ramamurthy, K.N., Varshney, K.R., Riemer, M., Bouneffouf, D., Richards, J.T., Dhurandhar, A., Daly, E.M., Hind, M., Sattigeri, P., et al.: Agentic ai needs a systems theory. arXiv preprint arXiv:2503.00237 (2025)
20. Ng, A.: Agentic design patterns part 1 (2024)
21. Noy, N.F., McGuinness, D.L., et al.: Ontology development 101: A guide to creating your first ontology (2001)

22. Okpala, I., Golgoon, A., Kannan, A.R.: Agentic ai systems applied to tasks in financial services: Modeling and model risk management crews. arXiv preprint arXiv:2502.05439 (2025)
23. Park, J.S., O'Brien, J.C., Cai, C.J., Morris, M.R., Liang, P., Bernstein, M.S.: Generative agents: Interactive simulacra of human behavior (2023), https://arxiv.org/abs/2304.03442
24. Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., Sun, M.: ChatDev: Communicative agents for software development. In: Ku, L.W., Martins, A., Srikumar, V. (eds.) Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 15174–15186. Association for Computational Linguistics, Bangkok, Thailand (Aug 2024). https://doi.org/10.18653/v1/2024.acl-long.810, https://aclanthology.org/2024.acl-long.810
25. Roman, D., Keller, U., Lausen, H., De Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web service modeling ontology. Applied ontology **1**(1), 77–106 (2005)
26. Shadbolt, N., Berners-Lee, T., Hall, W.: The semantic web revisited. IEEE intelligent systems **21**(3), 96–101 (2006)
27. Talebirad, Y., Nadiri, A.: Multi-agent collaboration: Harnessing the power of intelligent llm agents (2023), https://arxiv.org/abs/2306.03314
28. Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., Anandkumar, A.: Voyager: An open-ended embodied agent with large language models (2023), https://arxiv.org/abs/2305.16291
29. Wang, J., Duan, Z.: Agent ai with langgraph: A modular framework for enhancing machine translation using large language models (2024), https://arxiv.org/abs/2412.03801
30. Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al.: A survey on large language model based autonomous agents. Frontiers of Computer Science **18**(6), 186345 (2024)
31. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., et al.: Autogen: Enabling next-gen llm applications via multi-agent conversation. arXiv preprint arXiv:2308.08155 (2023)
32. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models (2023), https://arxiv.org/abs/2210.03629

Appendix

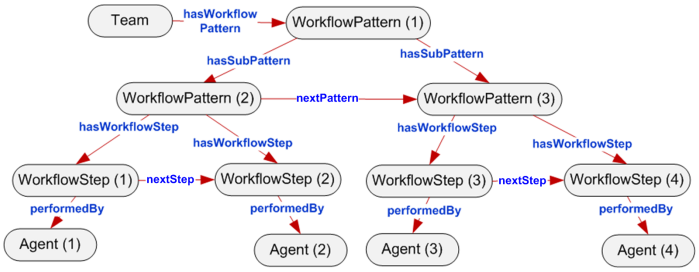

Fig. 7: An example of nested pattern" workflow model

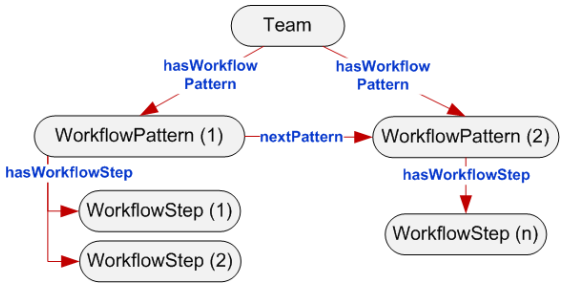

Fig. 8: An example of "parallel pattern" workflow model

Prompt Instructions

You are an expert in agent systems and ontology population.
You will be given: 1) An existing ontology file in Turtle format (http://www.w3id.org/agentic-ai/onto) ontology 2) The source code and configuration of an agent-based solution (with agents, tasks, tools, workflows, prompts, etc.) {{source_code}}
Your task:
1. Study the ontology file. - Treat it as a fixed schema with all classes and properties already defined. - Do NOT add, modify, or remove any classes or properties in the schema!
2. Study the source code and configuration. - Extract all instance-level information needed to fully describe the solution (agents, tasks, tools, workflows, prompts, parameters, data/artifacts, etc.).
3. Populate the ontology with individuals based on the extracted information. The goal is that another LLM can reconstruct the agent solution from the ontology instances you create. Do not add source code because we do not know the target framework to recreate the agent solution, so keep the semantic meaning and logic instead. - Use ONLY the existing classes and properties from the ontology! - Create individuals for all relevant entities and connect them with the appropriate properties. - Preserve all information from the source code (including prompts, parameters, and important logic) as literals or links between individuals. Fidelity is important, do not change or condense information.
4. If some aspects of the solution cannot be modeled with the current ontology: - Do NOT invent new classes or properties! - Do NOT model programming information (e.g., specific code structures, language-specific constructs, or implementation details), we are interested in the semantic meaning. Do NOT model framework specific functions and SDKs! Do NOT model UI components! - Model them as closely as possible with the existing schema but do not abuse the schema (e.g., do not press different classes and info in literals and descriptions). - If concepts are missing: list any missing concepts, limitations, or necessary extensions in an “Issues / Assumptions” comment block at the top of the Turtle output. Do not put it into the ontology in descriptions, etc.
Output format:
- Respond with the instance information in .ttl format. - At the very top of the Turtle content, write a comment block:
Issues / Assumptions: - <issue 1 or ”No issues detected”> - <issue 2> ...

Do not ask for confirmation or clarifications, just produce the output as specified.

```
1
2  @prefix : <http://www.w3id.org/agentic-ai/onto#> .
3  @prefix pp: <http://purl.org/net/p-plan#> .
4  @prefix owl: <http://www.w3.org/2002/07/owl#> .
5  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
6  @prefix xml: <http://www.w3.org/XML/1998/namespace> .
7  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
8  @prefix beam: <http://w3id.org/beam/core#> .
9  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
10 @base <http://www.w3id.org/agentic-ai/onto#> .
11
12 :TripPlannerCrew a :Team ;
13     <http://purl.org/dc/terms/title>  Trip Planner Crew  ;
14     <http://purl.org/dc/terms/description>  A crew composed of multiple LLM
       agents and tools to plan trips: city selection, local expertise gathering
       , and travel concierge planning.  ;
15     :hasAgentMember :CitySelectionAgent , :LocalExpertAgent , :
       TravelConciergeAgent ;
16     :hasWorkflowPattern :TripPlanningPattern ;
17     :hasSystemConfig :EnvConfig .
18
19 :CitySelectionAgent a :LLMAgent ;
20     :agentID  city_selection_agent  ;
21     :agentRole  City Selection Expert  ;
22     <http://purl.org/dc/terms/description>  An expert in analyzing travel
       data to pick ideal destinations  ;
23     :agentPrompt :CitySelectionAgent_BackstoryPrompt ;
24     :hasAgentGoal :CitySelectionGoal ;
25     :hasAgentCapability :CitySelectionCapability ;
26     :agentToolUsage :SearchTools , :BrowserTools ;
27     :useLanguageModel :OpenAI_LanguageModel ;
28     :hasAgentConfig :OpenAIConfig ;
29     :interactsWith :LocalExpertAgent , :TravelConciergeAgent .
30 ...
31
32 :CitySelectionAgent_BackstoryPrompt a :Prompt ;
33     :promptInstruction  An expert in analyzing travel data to pick ideal
       destinations  ;
34     :promptContext  Role: City Selection Expert; purpose: select the best
       city based on weather, season, and prices.  ;
```

Table 4: Agentic AI Core Concepts. (Ag = AutoGen, Cr = CrewAI, Lg = LangGraph, Ma = Mastra AI); ✓ = supported; × = not explicitly supported by the respective framework.

Class	Description	Ag	Cr	Lg	Ma
Agent	An entity capable of perceiving its environment, processing information, and acting to achieve goals.	✓	✓	✓	✓
LLMAgent	An AI agent based on a language model capable of autonomous task execution.	✓	✓	✓	✓
HumanAgent	A human participating or collaborating with agentic workflows.	✓	✓	×	✓
Capability	An ability that can be performed by an agent.	✓	✓	✓	✓
Tool	An instrument used by an agent to enhance its capabilities.	✓	✓	✓	✓
Resource	Any asset that can be utilized by an agent or tool to perform tasks.	✓	✓	✓	✓
Task	A specific activity that contributes to achieving objectives or goals.	✓	✓	✓	✓
WorkflowStep	An individual action or phase within a workflow pattern.	✓	✓	✓	✓
StartStep	The initial step in a workflow pattern.	✓	✓	✓	✓
EndStep	The final step in a workflow pattern.	✓	✓	✓	✓
WorkflowPattern	A reusable structured workflow template.	✓	✓	✓	✓
Goal	A desired state that an agent or team aims to achieve.	✓	✓	✓	✓
Objective	A collective objective assigned to a team.	✓	✓	×	×
Team	A coordinated group of agents.	✓	✓	×	×
Environment	The surroundings or context in which an agent operates.	✓	✓	✓	✓
KnowledgeBase	A structured collection of information that an agent can reference.	✓	✓	✓	✓
Memory	Stores and retrieves past information to support reasoning and decision-making.	✓	✓	✓	✓
Prompt	A structured instruction or input given to an agent.	✓	✓	✓	✓
Constraint	A rule or condition that restricts or guides agent decisions or actions.	✓	✓	✓	✓
Config	Configuration settings for agents, tools, or environments.	✓	✓	✓	✓
Context	A conceptual frame containing relevant situational data for reasoning and action.	✓	✓	✓	✓
Instance	A data object or content referenced/used within workflows.	✓	✓	✓	✓
LanguageModel	The model underlying LLM Agents.	✓	✓	✓	✓

Table 5: Identified Relationships Model for Agentic AI Systems

Property	**Domain**	**Range**	**Description**
participatedIn	Agent (prov)	Task	Agent participated in task execution.
humanParticipatedIn	HumanAgent	Task	Human agent participated in a task.
agentPrompt	LLMAgent	Prompt	Prompt used by an agent independently of tasks.
taskPrompt	Task	Prompt	Prompt used within a task.
hasPrompt	Agent or Task	Prompt	Associates prompts with agents or tasks.
agentResourceUsage	LLMAgent	Resource	Agent directly consumes a resource.
resourceUsage	Tool	Resource	Tool accesses a resource on behalf of the agent.
containsResource	Environment	Resource	Environment includes resources.
producedResource	Task	Resource	Resource produced as task output.
requiresResource	Task	Resource	Task requires a resource.
agentToolUsage	LLMAgent	Tool	Agent directly uses a tool.
toolUsage	Tool	Tool	A tool uses another tool during execution.
hasAgentCapability	LLMAgent	Capability	Capability an agent can perform.
requiresCapability	Task	Capability	Capability needed by a task.
hasCapability	Tool	Capability	Capability provided by a tool.
hasAgentConfig	LLMAgent	Config	Configuration settings for an LLM agent.
hasEnvironmentConfig	Environment	Config	Environment configuration settings.
hasSystemConfig	Team	Config	Configuration settings specific to team system.
hasToolConfig	Tool	Config	Configuration for tool.
hasConfig	Any	Config	Configuration associated with any component.
hasAgentGoal	LLMAgent	Goal	Goal assigned to an agent.
hasTeamGoal	Team	Goal	Goal assigned to a team.
hasGoal	Agent or Team	Goal	Relates entity to its goals.
contributesToGoal	Objective	Goal	Objective contributes to goal.
contributesToObjective	Task	Objective	Task contributes to objective.
hasObjective	Agent or Team	Objective	Assigned objective of a team/agent.
hasKnowledge	LLMAgent	KnowledgeBase	Knowledge possessed by an agent.
hasWorkflowPattern	Team	WorkflowPattern	Assigned workflow pattern of a team.
hasWorkflowStep	WorkflowPattern	WorkflowStep	Step that is part of workflow pattern.
hasAssociatedTask	WorkflowStep	Task	Workflow step corresponds to a task.
relatedStep	WorkflowStep	WorkflowStep	Relationship between workflow steps.
nextStep	WorkflowStep	WorkflowStep	Next step in workflow sequence.
hasSubPattern	WorkflowPattern	WorkflowPattern	Workflow sub-pattern relationship.
nextPattern	WorkflowPattern	WorkflowPattern	Next workflow pattern in sequence.
hasRelatedPattern	WorkflowPattern	WorkflowPattern	Pattern relationship.
hasAgentMember	Team	LLMAgent	Team includes agent members.
interactsWith	LLMAgent	LLMAgent	Agents interacting with each other.
operatesIn	LLMAgent	Environment	Agent's operational environment.
performedBy	Task	Tool	Tool that performs the task.
performedByAgent	Task	LLMAgent	Agent that performs the task.
useLanguageModel	LLMAgent	LanguageModel	Associates agent with language model.

Table 6: Agentic Pattern Collected from AutoGen, CrewAI, LangGraph and Mastra AI Repository (✓ = provided; × = not explicitly provided).

#	Title	Framework	Tool Use	Multi-Agent	Workflow Pattern
AutoGen					
1	Multi-Agent Conversation and Stand-up Comedy	AutoGen	×	✓	Sequential
2	Sequential Chats and Customer Onboarding	AutoGen	×	✓	Sequential
3	Reflection and Blogpost Writing	AutoGen	×	✓	Nested-sequential
4	Tool Use and Conversational Chess	AutoGen	✓	✓	Hybrid
5	Coding and Financial Analysis	AutoGen	×	✓	Hybrid
6	Planning and Stock Report Generation	AutoGen	×	✓	Hybrid
CrewAI					
7	Game Builder Crew	CrewAI	×	✓	Sequential
8	Industry Agents	CrewAI	✓	✓	Sequential
9	Instagram Post	CrewAI	✓	✓	Sequential
10	Job Posting	CrewAI	✓	✓	Sequential
11	Landing Page Generator	CrewAI	✓	✓	Sequential
12	Markdown Validator	CrewAI	✓	×	Single-step
13	Match Profile to Positions	CrewAI	✓	✓	Sequential
14	Marketing Strategy	CrewAI	✓	✓	Sequential
15	Meta Quest Knowledge	CrewAI	×	×	Sequential
16	Prep for a Meeting	CrewAI	✓	✓	Sequential
17	Recruitment	CrewAI	✓	✓	Sequential
18	Screenplay Writer	CrewAI	✓	✓	Sequential
19	Starter Template	CrewAI	✓	×	Single-step
20	Stock Analysis	CrewAI	✓	✓	Sequential
21	Surprise Trip	CrewAI	✓	✓	Sequential
22	Trip Planner	CrewAI	✓	✓	Sequential
LangGraph					
23	Chat Agent	LangGraph	×	×	Single-step
24	Email Agent	LangGraph	×	×	Hybrid
25	Open Code	LangGraph	✓	✓	Sequential
26	Pizza Orderer	LangGraph	✓	×	Sequential
27	Stockbroker	LangGraph	✓	×	Sequential
28	Supervisor	LangGraph	✓	✓	Hybrid
Mastra AI					
29	A2A	Mastra AI	×	✓	Sequential
34	Bird Checker with Express	Mastra AI	✓	×	Single-step
35	Bird Checker with NextJS	Mastra AI	✓	×	Sequential
36	Bird Checker with NextJS and Eval	Mastra AI	✓	×	Sequential
37	Client Side Tools	Mastra AI	✓	×	Single-step
38	Crypto Chatbot	Mastra AI	✓	✓	Single-step
39	DANE	Mastra AI	✓	✓	Nested-sequential
41	Fireworks R1	Mastra AI	×	×	Single-step
42	Heads Up Game	Mastra AI	✓	✓	Sequential
43	Memory per Resource Example	Mastra AI	✓	×	Sequential
44	Memory Todo Agent Memory	Mastra AI	✓	×	Sequential
45	Memory with Context	Mastra AI	×	×	Single-step
46	Memory with LibSQL	Mastra AI	×	✓	Sequential
47	Memory with MongoDB	Mastra AI	✓	✓	Sequential
48	Memory with Postgres	Mastra AI	✓	✓	Sequential
49	Memory with Processors	Mastra AI	✓	✓	Sequential
50	Memory with Upstash	Mastra AI	✓	✓	Sequential
51	MCP Configuration	Mastra AI	✓	×	Single-step
52	MCP Registry	Mastra AI	✓	×	Sequential
53	OpenAPI Spec Writer	Mastra AI	✓	×	Sequential
54	Quick Start	Mastra AI	✓	×	Single-step
55	Stock Price Tool	Mastra AI	✓	×	Single-step
56	Travel App	Mastra AI	✓	✓	Sequential
57	Weather Agent	Mastra AI	✓	×	Sequential
58	Workflow AI Recruiter	Mastra AI	✓	×	Sequential
59	Workflow with Inline Steps	Mastra AI	×	✓	Sequential
60	Workflow with Memory	Mastra AI	✓	×	Hybrid
62	Workflow with Separate Steps	Mastra AI	×	×	Sequential
63	Workflow with Suspend Resume	Mastra AI	×	×	Hybrid
64	YC Directory	Mastra AI	✓	×	Sequential
65	AI SDK v5	Mastra AI	✓	×	Single-step
66	AI SDK UseChat	Mastra AI	✓	✓	Sequential

```
    :promptOutputIndicator  Use tools to find weather patterns, seasonal
    events, and travel costs. Produce a detailed report.  .
...

:CitySelectionGoal a :Goal ;
    <http://purl.org/dc/terms/title>  Select the best city  ;
    <http://purl.org/dc/terms/description>  Select the best city based on
    weather patterns, seasonal events, and travel costs  .
...
:CitySelectionCapability a :Capability ;
    <http://purl.org/dc/terms/description>  Analyze and compare cities by
    weather conditions, events, and travel costs; deliver a detailed city
    selection report.  .
```

Listing 1.3: Agentic AI pattern for "Trip Planner" in RDF Turtle. (excerpt)