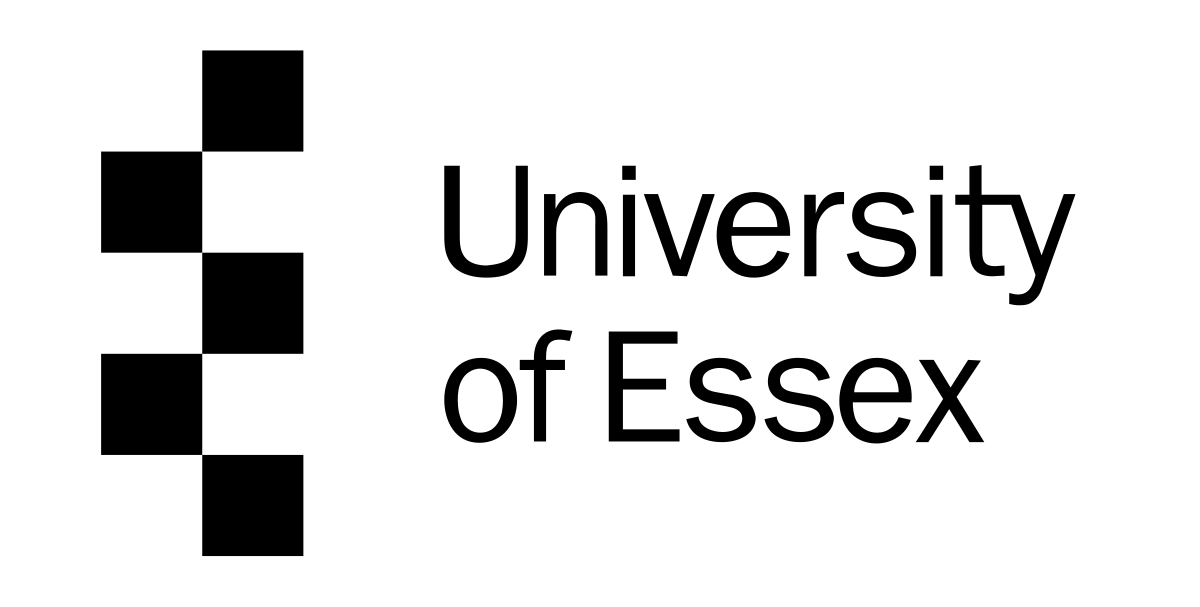
****

**Intelligent Academic Research Agent System**

**System Requirements**

We will utilize Python as the primary programming language to develop an agent that can search for online academic research findings based on specific search phrases (e.g., from social media platforms or search engines). Python is perfect because of its ease of use, robust libraries, and extensive support for data management and web scraping.  
  
The libraries listed below will be necessary:

* Requests: To communicate with websites or APIs by sending HTTP requests.
* BeautifulSoup: To parse HTML and retrieve pertinent data from the Document Object Model (DOM) of the webpage.
* Selenium: This program will behave like a web browser to collect data from websites with a lot of JavaScript, if content that changes needs to be extracted.
* Pandas: To process, store, and work with the extracted data in an organized style similar to DataFrames.

Web access rights and the capacity to effectively parse HTML or JSON data are additional system needs. Finally, data storage formats like JSON or CSV will allow the processed results to be saved offline in a machine-readable way (Li & Wang, 2020). A crucial component of the architecture will be making sure the agent can keep the results offline in a secure location, either by communicating with cloud storage services or producing local files.

**Design Process**  
  
The design of the system will prioritize flexibility, scalability, and modularity. The agent will be organized into a multi-step pipeline with distinct tasks at each stage. The input and query formulation will be handled by the first module, which will also decide which search phrases to use and where to look. The results will be retrieved by the second module using either Selenium for dynamic, JavaScript-rendered data or web scraping for static data.  
  
The decision to use BeautifulSoup for parsing static HTML originates from its simplicity and reliability when extracting structured elements like titles, links, and metadata (Davis, 2021). In contrast, Selenium was chosen for more complex websites that require interaction. For instance, some academic platforms load content only after scrolling or clicking, and Selenium allows the agent to behave like a real user, improving extraction accuracy (Gupta, 2022).   
  
Additionally, we chose to use Python because of its well-established application in data analysis and web scraping jobs.

**2.0 Approaches to System Creation**

The Academic Research Online agent system is developed based on an Agent-Oriented Software Engineering (AOSE) paradigm and is inspired by the structured design traditions of role allocation through modularity and goal-oriented decomposition (often linked to well-known AOSE methodologies, e.g., Gaia-inspired role modelling). The methodology advocates complex distributed systems that are constituted of autonomous agents by allowing systematic separation of responsibilities, communication design, and behavioral reasoning. Task decomposition is the initial step of the design process, and it splits the academic research process, query formulation, data retrieval, filtering, and offline storage into specialized agents, which would allow scalability, control clarity, and behavioral transparency.

The system follows a multi-agent processing pipeline that has four major agents: (1) the Query Agent that converts the topics provided by the user into structured search intentions; (2) the Retrieval Agent, which generates HTTP requests to an academic search interface or API and retrieves structured metadata; (3) the Processing Agent, which performs relevance ranking, filtering thresholds, and weighting of keywords; and (4) the Storage Agent, which formats and stores validated outputs in an offline repository. An event-based execution cycle based on structured performatives based on Agent Communication Languages (ACLs) enables agents to exchange messages of type inform, request, and achieve, as seen in the KQML model by Finin et al. (1994), by locating, executing, and communicating. Such a communication protocol provides a predictable pattern of coordination, modularity to extension, and traceable decision flows based on well-defined semantic structures.

**2.1 Agent Architecture, Reasoning and Processing Logic**

An agent architecture based on a hybrid approach is followed that combines deliberative planning and reactive responsiveness to achieve a balance between computational efficiency and strategic reasoning. The Query Agent has a reasoning model based on BDI, wherein beliefs are held topics and metadata retrieved, desires are organizational research objectives in the long term, and intentions are plans of action, queries, and monitoring strategies. This arrangement is in line with rational, resource-bounded planning that is in accordance with Bratman et al.'s (1988) scheme of practical reasoning.

The Retrieval and Storage Agents mostly use reactive behavior and hence react fast to external stimuli and incoming data streams, whereas the Processing Agent adds lightweight deliberation by using threshold-based scores of filtering and prioritization. The fully reactive architectures (Brooks, 1991) were rejected because of their poor representation ability, and the purely symbolic planners posed the danger of high computational cost and low real-time responsiveness. The hybrid solution is thus a practical compromise, which makes sure that it is flexible without compromising on representational coherence. Nonetheless, the complexity might escalate as the number of agents scales, which may demand coordination overhead and traffic messages and controlled communication approaches to ensure efficiency in the system, as proposed by dynamic multi-agent environments (Wooldridge, 2009).

**2.2 Literature-Based Technical Rationale**

The system's intelligent agent theories form the basis of the architectural choices of the system. The BDI model (Bratman et al., 1988) presents a sound model to develop autonomous decision-making and long-term goal monitoring. Wooldridge's (2009) definitions of the hybrid multi-agent architectures justify this decision, as it has proven effective in the context in which strategic planning and real-time reactivity are needed. The competence-based approach developed by Maes (1991) also supports the design of distributed agents whose behavioral responsibilities are specialized.

The use of agent communication is based on Searle's (1969) speech act theory and the operationalization of this theory via the use of KQML performatives (Finin et al., 1994) to allow agents to coordinate themselves semantically. Moreover, Russell and Norvig's (2021) principles of knowledge representation guide the conversion of retrieved academic metadata into organized and logically consistent internal structures, increasing its reliability for onward storage and analysis in the offline repository.

**3.1. Data Quality and Retrieval Challenges**

A potential challenge that the Academic Research Online agent system may face is improper search results particularly where the query is general and websites are dynamic, which can be mitigated using BDI-based reasoning, allowing the Query Agent to have consistent beliefs regarding the user goals and modify intentions when poor quality or restrictive outputs are identified (Bordini, R. H., El Fallah Seghrouchni, A., Hindriks, K., Logan, B., & Ricci, A., 2020). The other highly similar problem is that of data inconsistency, which involves various platforms having different metadata structures or missing fields. The challenge may be resolved with the help of the mechanisms of negotiation and alignment that allow the Processing Agent a reconciliation of discrepancies based on the schema-matching and prioritisation rules (Hung, N. Q. V., Luong, X. H., Miklós, Z., Quan, T. T., & Aberer, K., 2013).

**3.2. Agent Workload and System Performance Issues**

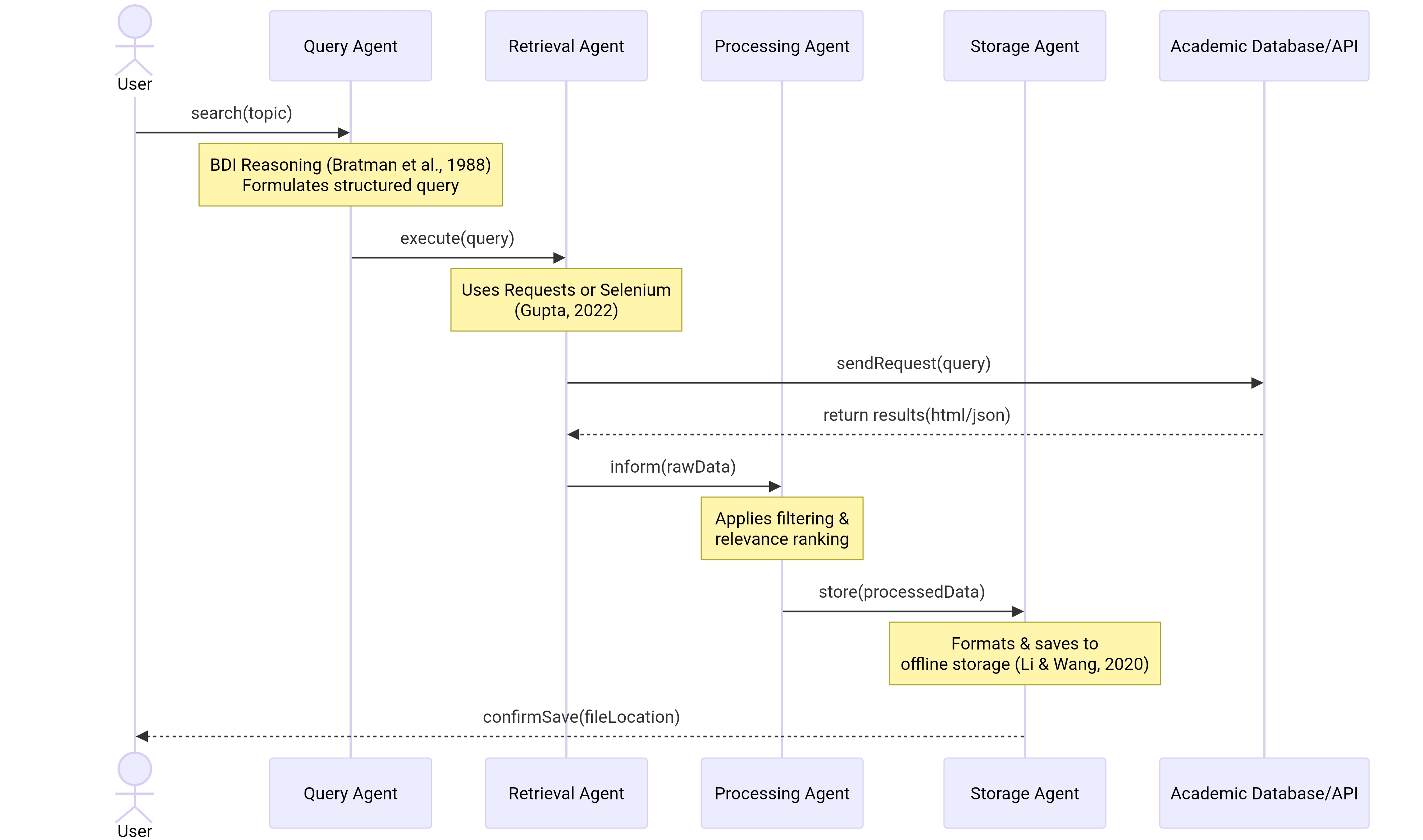
Another risk is that of overloaded agents, and this is especially relevant when the Retrieval Agent makes multiple requests to busy academic platforms. In them, the hybrid agent paradigms are an appropriate solution since it can balance between reactive responses and deliberate scheduling so as to avoid resource exhaustion (Petrenko, 2025). There may also be bottlenecks of communication; pipeline efficiency may be lowered or broken by the fact that communication between agents requires time. To counter this fact, structured ACL-inspired communication protocols provide a predictable message flow, acknowledgement and handling of timeouts.

**3.3. Interpretation, Relevance, and Semantic Alignment Problems**

Relevancy filtering errors are another challenge during which the Processing Agent misclassifies results or accidentally pares out useful academic information. Rational intention revision and adaptive filtering theories endorse a process of narrowing the threshold so that accuracy can be enhanced over time. Lastly, ontology mismatch, whereby a source varies in terms of terminology and categorisation, may deal a setback to consistent interpretation (Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., & Antoniou, G., 2008). The methods based on the knowledge-representation theory assist in aligning extracted concepts as well as semantically coherent across the system.

**4. Graphical Designs**

**4.1 Sequence Diagram**



***Figure 1: Sequence Diagram***

***Source: Author Made***

This diagram illustrates the chronological flow of messages and interactions between the system’s components to fulfill a user request, demonstrating the event-driven communication protocol inspired by KQML (Finin et al., 1994).

**Main Components and Flow:**

Actors/Objects: User, Query Agent, Retrieval Agent, Processing Agent, Storage Agent, and the external Academic Database/API.

Initialization: The User initiates the process by sending a search(topic) request to the Query Agent.

Query Formulation: The Query Agent performs its BDI-based reasoning (Bratman et al., 1988) to formulate a structured query, which it then sends (execute(query)) to the Retrieval Agent.

Data Retrieval: The Retrieval Agent interacts with the Academic Database/API. For static sites, it uses HTTP requests (Requests library); for dynamic content requiring JavaScript, it employs browser automation (Selenium) (Gupta, 2022). The raw search results are returned.

Processing & Storage: Results are passed to the Processing Agent for filtering and ranking. Validated data is then sent to the Storage Agent, which formats and saves it to an offline repository (e.g., CSV/JSON) (Li & Wang, 2020), before finally confirming completion to the user.

actor User

participant QA as Query Agent

participant RA as Retrieval Agent

participant PA as Processing Agent

participant SA as Storage Agent

participant DB as Academic Database/API

User->>QA: search(topic)

Note over QA: BDI Reasoning (Bratman et al., 1988)<br/>Formulates structured query

QA->>RA: execute(query)

Note over RA: Uses Requests or Selenium<br/>(Gupta, 2022)

RA->>DB: sendRequest(query)

DB-->>RA: return results(html/json)

RA->>PA: inform(rawData)

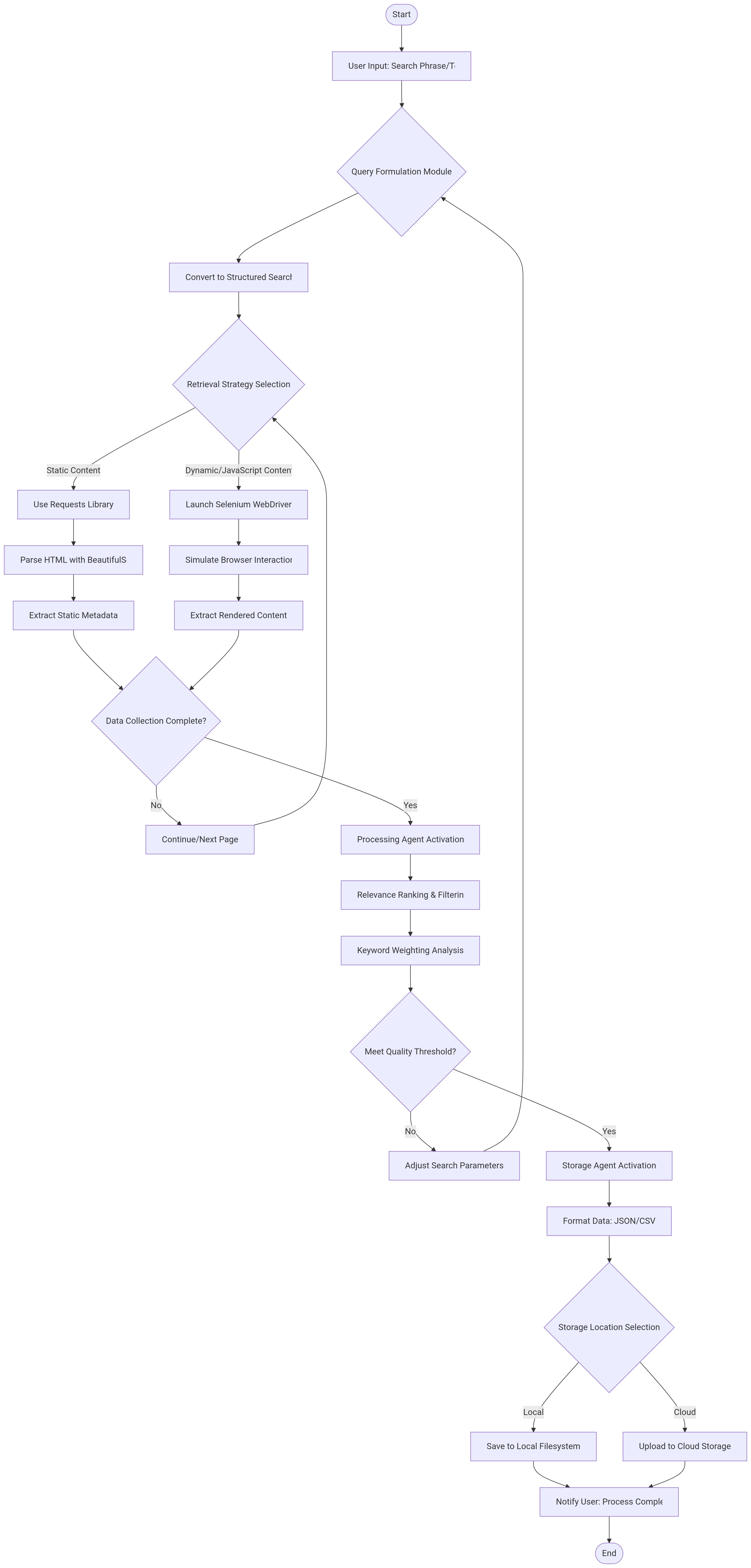
Note over PA: Applies filtering &<br/>relevance ranking

PA->>SA: store(processedData)

Note over SA: Formats & saves to<br/>offline storage (Li & Wang, 2020)

SA-->>User: confirmSave(fileLocation)

**4.2 Activity Diagram**

****

***Figure 2: Activity Diagram***

***Source: Author Made***

This diagram models the system’s overall control flow and decision points as a continuous process, from trigger to final storage, highlighting its hybrid reactive-deliberative architecture (Wooldridge, 2009).

**Main Components and Flow:**

Start & Initial Action: The workflow begins with the “Receive User Search Topic” action.

Parallel Processing: After query formulation, two key activities often occur in parallel: “Retrieve Data from Web/API” and “Monitor for Results/Errors,” showcasing reactive responsiveness.

Critical Decision Points:

1. “Content Dynamic?” determines whether to use Selenium or BeautifulSoup for parsing (Davis, 2021).

2. “Data Quality & Relevance OK?” represents the Processing Agent’s threshold-based filtering, a lightweight deliberative step.

3. “Schema Match?” addresses potential ontology mismatches, invoking alignment mechanisms before storage (Flouris et al., 2008).

End: The process concludes with the “Store in Offline Repository” action and a final flow end.

Start([Start]) --> A[Receive User Search Topic]

A --> B[Formulate Structured Query<br/>BDI Model]

B --> C{Content Dynamic?}

C -- Yes --> D[Use Selenium for Rendering]

C -- No --> E[Use Requests & BeautifulSoup]

D & E --> F[Parse & Extract Data]

F --> G{Data Quality & Relevance OK?<br/>Processing Agent Filter]

G -- No --> H[Discard Result]

G -- Yes --> I{Schema Match?<br/>Align Metadata]

I -- No --> J[Reconcile Data Structure]

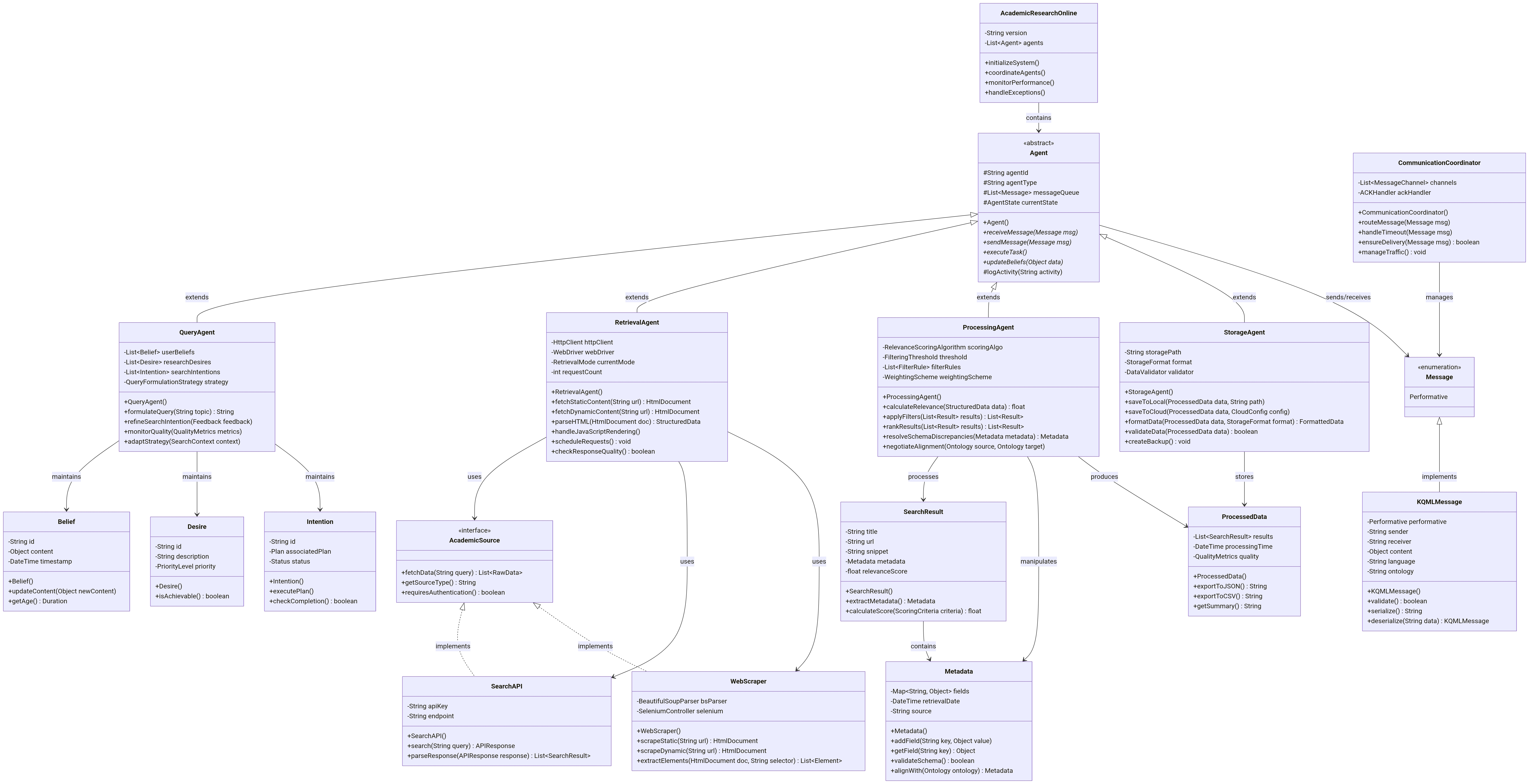
I -- Yes --> K[Store in Offline Repository<br/>CSV/JSON File]

J --> K

H --> B

K --> End([End])

**4.3 Class Diagram**

****

***Figure 3: Class Diagram***

***Source: Author Made***

This diagram defines the static structure of the system, showing the key classes, their attributes, operations, and relationships, reflecting the Agent-Oriented Software Engineering (AOSE) modularity.

**Main Components and Relationships:**

Core Agent Classes: The four specialized agents (QueryAgent, RetrievalAgent, ProcessingAgent, StorageAgent) are central. Each has a -role attribute and core methods aligning with their responsibilities.

Key Attributes & Methods:

QueryAgent contains BDI model lists (beliefs, desires, intentions) and a formulateQuery() method (Bratman et al., 1988).

RetrievalAgent holds the parser\_type and uses fetchStatic() or fetchDynamic() methods (Gupta, 2022).

ProcessingAgent applies filterThreshold and rankResults().

StorageAgent manages storage\_path and handles saveToCSV()/saveToJSON() (Li & Wang, 2020).

Supporting Classes: SearchQuery, ResearchPaper, and DataStorage encapsulate data. SearchQuery has a -strategy for different academic platforms.

Relationships: Agents are associated with the data classes they process. The Coordinator class (optional) manages the agent lifecycle and interaction, reducing communication bottlenecks (Petrenko, 2025).

**References**

1. Davis, J. (2021) ‘Introduction to Web Scraping with Python: BeautifulSoup and Requests’, *Real Python*. Available at: <https://realpython.com/python-web-scraping-practical-introduction/>
2. Li, X. & Wang, W. (2020) ‘Automating Data Collection for Research: A Case Study Using Python’, *Journal of Digital Information and Research*, 14(3), pp. 112-126. Available at: <https://www.digitalresearchjournal.com/automating-data-collection-python>
3. Gupta, R. (2022) ‘Selenium for Web Scraping: Advanced Techniques for Data Extraction’, *Web Scraping Today*. Available at: <https://www.webscrapingtoday.com/selenium-advanced-techniques>
4. Bratman, M.E., Israel, D.J. and Pollack, M.E. (1988) ‘Plans and resource-bounded practical reasoning’, Computational Intelligence, 4(3), pp. 349–355.
5. Brooks, R.A. (1991) ‘Intelligence without representation’, Artificial Intelligence, 47(1–3), pp. 139–159.
6. Finin, T., Fritzson, R., McKay, D. and McEntire, R. (1994) ‘KQML as an agent communication language’, Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM), pp. 456–463.
7. Maes, P. (1991) ‘The Agent Network Architecture (ANA)’, SIGART Bulletin, 2(4), pp. 115–120.
8. Reynolds, C.W. (1987) ‘Flocks, herds and schools: A distributed behavioural model’, Computer Graphics, 21(4), pp. 25–34.
9. Russell, S. and Norvig, P. (2021) Artificial Intelligence: A Modern Approach. 4th edn. Harlow: Pearson Education.
10. Searle, J.R. (1969) Speech Acts: An Essay in the Philosophy of Language. Cambridge: Cambridge University Press.
11. Wooldridge, M. (2009) An Introduction to MultiAgent Systems. 2nd edn. Chichester: John Wiley & Sons.
12. Payne, T.R. and Tamma, V. (2014) ‘Negotiating over ontological correspondences with asymmetric and incomplete knowledge’, Autonomous Agents and Multi-Agent Systems, 28(3), pp. 517–524.
13. Kaelbling, L.P. (1991) ‘A situated-automata approach to the design of embedded agents’, SIGART Bulletin, 2(4), pp. 85–88.
14. Bordini, R. H., El Fallah Seghrouchni, A., Hindriks, K., Logan, B., & Ricci, A., 2020. Agent programming in the cognitive era. *Autonomous Agents and Multi-Agent Systems,* 34(2), p. 37.
15. Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., & Antoniou, G., 2008. Ontology change: classification and survey. *The Knowledge Engineering Review,* 23(2), pp. 117-152.
16. Hung, N. Q. V., Luong, X. H., Miklós, Z., Quan, T. T., & Aberer, K., 2013. Collaborative Schema Matching Reconciliation. *In OTM Conferences,* pp. 222-240.
17. Petrenko, A., 2025. Agent-based approach to implementing artificial intelligence (AI) in service-oriented architecture (SOA). *System research and information technologies,* Volume 1, pp. 104-123.