DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Agentic AI Architecture Design

Workflow Automation in Compliance with Operational Excellence

Bachelor's Thesis

Wirtschaftsinformatik—Business Engineering
Duale Hochschule Baden-Württemberg
Ravensburg

Francisco Rodriguez Müller, on September 23, 2025
Mat. Nr.: 2775857, Course: RV-WWIBE122
Supervisor: Prof. Dr. Paul Kirchberg

Declaration of Authenticity

I hereby declare that I have written the present bachelor's thesis independently and that I have not used any sources or aids other than those indicated. All passages that were taken from published or unpublished works are clearly marked as such. This thesis has not been submitted to any other examination authority in the same or a similar form.

Thesis Title:
Agentic AI Architecture Design
Workflow Automation in Compliance with Operational Excellence

Ravensburg, September 23, 2025

(Place and Date) *(Signature)*

Acknowledgments

I would like to thank the *All for One Group SE* for providing the opportunity and resources to write this thesis. I also extend my gratitude to my academic supervisor, *Prof. Dr. Paul Kirchberg* for his guidance throughout the process; my professional advisor, *Emily Celen* for helping me with the scope and realistic expectations for this work; and my direct supervisor and friend *Ralph Thomaßen* for his wise advice and relentless support throughout my academic journey.

Abstract

[Placeholder, will write it once the thesis is finished.]

Contents

Abbreviations

A2A	Agent to Agent
DSR	Design Science Research
GaaS	Governance as a Service
GenAI	Generative Artificial Intelligence
IPA	Intelligent Process Automation
ISD	Information Systems Design
LLM	Large Language Model
MAS	Multi-Agent System
MBSE	Model-Based Systems Engineering
MCP	Model Context Protocol
OpEx	Operational Excellence
QCA	Qualitative Content Analysis
RE	Requirements Engineering
ReAct	Reason & Action
RPA	Robotic Process Automation
SLA	Service Level Agreement
SysML	Systems Modeling Language
WfMS	Workflow Management Systems

Listings

1 Introduction

Organizations across industries continue to face persistent challenges in achieving operational excellence (OpEx). Fragmented processes, manual interventions, and inconsistent data quality undermine efficiency and decision-making. Legacy workflows and siloed systems exacerbate these inefficiencies, while traditional automation approaches often lack the adaptability needed in dynamic business environments. For companies, this translates into slower response times, higher compliance risks, and limited scalability—issues that directly threaten competitiveness.

Agentic AI, building on the advances of generative artificial intelligence (GenAI), opens new possibilities to extend automation beyond deterministic scripts. While GenAI provides the cognitive and generative capabilities, agentic AI leverages these to create adaptive, tool-using agents that can plan, act, and coordinate—thereby supporting governance, decision quality, and organizational agility. Despite this potential, both practice and academic literature lack structured strategies and conceptual frameworks for embedding such agentic capabilities into operational workflows in a scalable and value-driven way. This gap motivates the present research.

In this context, multi-agent systems (MAS) can serve as a reference architecture for integrating GenAI-enabled agentic AI into enterprise workflow automation. The central research question is:

How can a MAS architecture be designed to integrate GenAI capabilities into workflow automation, in order to enhance agility, compliance, and decision quality to achieve OpEx?

To answer this question, the study addresses the following sub-questions:

- *Which design requirements are necessary to align a multi-agent architecture with the goals of OpEx?*
- *How should a MAS be architected to fulfill these requirements?*
- *Under which conditions is deploying a generative multi-agent architecture justified over traditional automation approaches?*

Methodologically, the thesis applies Design Science Research (DSR) to develop a conceptual reference architecture. The approach synthesizes requirements from academic literature and OpEx principles, models agent roles and interactions, and derives applicability conditions for real-world deployment.

The core contribution of this work is a conceptual design of a MAS that leverages GenAI to support OpEx in enterprise workflows. Specifically, it delivers:

- *A structured synthesis of system requirements derived from academic literature and OpEx principles.*
- *A conceptual architecture detailing agent roles, interactions, and integration points.*
- *A set of applicability conditions and design considerations to guide future deployment and evaluation of generative multi-agent architectures in practice.*

The scope is limited to conceptual design; formal evaluation and technical implementation are proposed as future work. Although the architecture is designed to remain industry-agnostic, a use case from the financial services sector is introduced to illustrate how the conceptual model can be instantiated in a regulated, legacy-intensive environment.

After this introduction in Section 1, the thesis is structured as follows: Section 2 outlines the research methodology, including the use of Design Science Research (DSR) and supporting methods. Section 3 presents a literature review on operational excellence, workflow automation, and agentic AI. Section 4 details the synthesis and modeling of requirements. Section 5 develops the conceptual multi-agent architecture. Section **??** discusses the applicability of MAS in workflow automation use cases, and Section **??** concludes with reflections and directions for future research.

2 Methodology

This thesis applies DSR methodology to create a conceptual artifact—a multi-agent architecture for workflow automation. Practically, the approach unfolded in three steps: (1) *reviewing the literature* on OpEx, workflow automation, and agentic AI; (2) *deriving and structuring requirements* from literature and case material into a requirements model; and (3) *designing a conceptual system architecture* using Systems Modeling Language (SysML).

Supporting methods included Mayring-style qualitative content analysis (QCA) for the review, requirements engineering (RE) and systems analysis for the requirements model, and information systems design (ISD) to structure the architecture and ensure requirement-to-design traceability, supported by SysML modeling practices from Model-Based Systems Engineering (MBSE). Within DSR, the work focuses on

problem identification, objective definition, and conceptual design, while instantiation/demonstration and formal evaluation are out of scope given the bachelor-thesis format and resource constraints. This scoping maintains methodological rigor while keeping the contribution focused: a well-argued reference architecture ready for subsequent implementation and empirical evaluation.

2.1 Qualitative Content Analysis

To ensure a systematic and structured literature review, this thesis employed QCA following the principles of Mayring and Fenzl (2022). As a rule-based method for synthesizing insights from textual sources, QCA was used within the DSR framework to support the problem identification and objective definition phases (Hevner et al. 2004; Peffers et al. 2007). In this thesis, it was applied in a literature-focused manner to structure the review and provide a traceable basis for subsequent RE.

The analysis was scoped along three dimensions. The *analysis unit* was defined as the overall body of literature addressing operational excellence, workflow automation, and agentic AI. The *context unit* consisted of individual publications (books, peer-reviewed articles, industry reports, standards). The *coding unit* was defined as discrete statements or conceptual claims relevant to the intersection of OpEx, automation paradigms, and AI-based multi-agent systems.

A mixed deductive-inductive approach was used. Deductive categories were derived from established theory, including OpEx dimensions such as adaptability, compliance, and decision quality, as well as prior automation frameworks (e.g., Robotic Process Automation, RPA; Intelligent Process Automation, IPA). Inductive categories emerged from the material itself, capturing issues highlighted repeatedly in the sources, such as observability, traceability, and governance in agentic AI. This balance ensured that both established and novel concerns were systematically reflected.

The outcome of this categorization was not a formal codebook, but a set of thematic clusters that guided the narrative structure of Section 3. Each subsection of the literature review is organized around these categories, which in turn serve as the input for the elicitation lists presented at the beginning of Section 4. In this way, QCA provides both a conceptual ordering of the literature and a direct bridge into the requirements engineering process.

2.2 Requirements Engineering

Following the IEEE (1990) definition, a requirement is a *condition or capability needed by a user to solve a problem or achieve an objective.* RE provides the systematic means to derive such objectives. In this thesis, RE was applied in the early phases to ensure that the conceptual architecture rests on precise, validated needs rather than general aspirations.

The synthesis of requirements followed a structured but literature-driven process. Recurring design concerns were identified from the results of the QCA, documented in *elicitation lists*, and then consolidated into a unified set of requirement candidates. Consistent with Glinz et al. (2020), each candidate was reformulated into an atomic, unambiguous, and verifiable "shall" statement and classified into *functional requirements* (system behaviors), *quality requirements* (non-functional attributes such as performance or compliance), or *constraints* (technological or regulatory limits).

While this procedure draws on the phases described by Herrmann (2022) (elicitation, documentation, analysis, management), it was adapted to the scope of this thesis: instead of stakeholder workshops, the primary elicitation source was the systematically coded literature. To situate the requirements, a complementary systems analysis defined the system boundary, identified stakeholders and external actors, and clarified interface obligations—helping prevent scope creep and omissions.

Requirements were then represented in SysML requirement diagrams. Trace links connect each documented requirement to the respective architecture elements, enabling full requirement-to-design traceability via ≪`satisfy`≫ and ≪`verify`≫ relationships. This model-based approach ensures that design decisions can always be traced back to validated needs and that no requirement was overlooked.

In summary, the integration of RE and systems analysis provided a structured, traceable, and quality-assured requirement set. This foundation anchors the subsequent conceptual architecture in rigorously defined objectives, ensuring consistency with both DSR methodology and operational excellence goals.

2.3 Information Systems Design

In line with the DSR approach, this thesis models the architecture in SysML v2 and applies MBSE practices to ensure requirements-to-design traceability. Although MBSE originates in systems engineering, its discipline transfers well to information systems and supports the systematic development of conceptual architectures. As

MAS grow in scope and complexity, a formal modeling framework becomes essential for maintaining control, observability, and consistency. SysML v2 is explored here as that framework, providing precise semantics and a version-controllable, analyzable representation of the artifact.

MBSE is used to transform the synthesized requirements into a coherent, analyzable system model and to validate the design at the conceptual level. Concretely, each requirement is represented as a SysML «`requirement`» element and linked via «`satisfy`» and, where applicable, «`verify`» relations to architectural elements (agent roles, interactions, and policies). This establishes requirements-to-design traceability and enables early checks against stakeholder needs and constraints. The resulting model offers a unified view of structure and behavior, making interface obligations, coordination patterns, and exception paths inspectable before implementation.

The conceptual architecture is captured as a SysML v2 model using a plain-text, Eclipse-based workflow. SysML v2's textual notation enables precise, tool-agnostic definitions of structure and behavior under version control. Compared with SysML v1 and informal diagramming, SysML v2 offers clearer semantics and richer constructs for specifying MAS concerns such as coordination patterns, policy enforcement points, interfaces, and exception pathways. In this work, the SysML v2 model functions as a machine-interpretable blueprint of the architecture: it supports early validation against the requirements, consistent terminology and interfaces, and a stable foundation for subsequent instantiation and evaluation.

3 Literature Review

The literature review is organized into three subsections that together frame the problem context of this thesis. It begins with operational excellence, which defines the strategic objectives—adaptability, compliance, decision quality—that guide enterprise transformation efforts. The second subsection addresses workflow automation, as the established technological approach for operationalizing these objectives in practice. The third subsection turns to agentic AI, a rapidly emerging paradigm that extends automation beyond deterministic scripts toward adaptive, tool-using agents. This sequence—objectives, established solutions, emerging solutions—provides a logical progression from strategic goals to current practice and then to prospective innovations. It ensures that the requirements synthesized in Section 4 are grounded

in both enduring management principles and the latest technological developments relevant to workflow design.

3.1 Operational Excellence

OpEx originated as a management philosophy in the manufacturing sector, particularly in the automotive industry to optimize quality and efficiency. In this classical context, OpEx focused on minimizing defects, eliminating waste, and embedding continuous improvement practices into organizational routines (Womack and Jones 1997). While these roots remain important, they provide only a partial foundation for understanding OpEx in today's IT-driven enterprises, which operate in volatile environments shaped by rapid technological change, regulatory complexity, and global competition.

One central dimension of OpEx is ADAPTABILITY AND AGILITY. In IT-driven firms, where OpEx is defined less by physical production flows and more by the ability to execute strategies effectively while maintaining innovation, adaptability refers to the capacity of processes to be reconfigured in response to volatility, ensuring resilience in dynamic environments. Agility emphasizes change-readiness and rapid adaptation, qualities increasingly recognized as indicators of organizational excellence in globalized and unpredictable markets. While adaptability enhances resilience, it can reduce efficiency if frequent changes disrupt standardization; conversely, a strong focus on efficiency can make processes rigid and less responsive to unexpected events. In practice, organizations must reconcile continuous improvement with agility, balancing stability and flexibility in their operational routines (cf. Carvalho et al. 2023, p. 1599).

Another recurring theme in the literature is COMPLIANCE AND RISK MANAGEMENT. OpEx in regulated industries demands that workflows embed mechanisms for ensuring transparency, auditability, and regulatory adherence. Compliance safeguards minimize operational risk but can introduce bureaucratic overhead and slow decision-making. The key challenge is balancing strict rule enforcement with the flexibility needed to respond to novel business conditions, a tension especially visible in digital service environments with evolving legal frameworks (cf. Owoade et al. 2024, p. 687).

A further category is DECISION QUALITY. A core aim of OpEx is not only to accelerate decision-making but to improve its reliability and evidential grounding (cf.

Owoade et al. 2024, p. 685). Automation can help by aggregating relevant data and reducing errors, but it also risks opacity and overconfidence when human oversight is limited. The central trade-off is speed versus quality: excessive automation may produce faster but less accountable outcomes, while excessive oversight slows operations. For sustainable excellence, systems must therefore support evidence-based choices and make decision pathways transparent (Carvalho et al. 2023).

Another key dimension is EFFICIENCY AND CONTINUOUS IMPROVEMENT. Rooted in Lean and TQM traditions, efficiency emphasizes minimizing waste and reducing manual effort, while continuous improvement institutionalizes iterative refinements in processes. Together, these principles enhance operational reliability and cost-effectiveness, yet they can conflict with the need for flexibility in volatile environments. The challenge is to design workflows that are optimized for today while remaining adaptable for tomorrow (cf. Juran and Godfrey 1999, p. 14.16 & p. 8.1).

Customer-facing outcomes are captured by CUSTOMER-CENTRICITY. OpEx emphasizes that processes must be aligned with user needs and service-level commitments, ensuring reliability, responsiveness, and satisfaction. A strong customer focus can create pressure to customize and accelerate processes, which may undermine efficiency or compliance. The literature stresses that sustainable excellence requires balancing external demands with internal consistency (Womack and Jones 1997; Juran and Godfrey 1999).

The category of USER EMPOWERMENT AND CULTURE recognizes that operational excellence is not solely technical but also organizational. Effective improvement requires systems that support transparency, collaboration, and employee engagement. Empowerment fosters ownership and participation, but decentralizing authority can introduce inconsistency and conflict with standardization goals. Culture therefore functions as both an enabler and a constraint for process excellence, shaping how automation is accepted and leveraged by human actors (cf. Juran and Godfrey 1999, p. 15.2-3) (cf. Womack and Jones 1997, p.3).

Finally, TECHNOLOGY INTEGRATION AND SCALABILITY reflects the increasing role of digital platforms in enabling OpEx. Modern enterprises rely on architectures that integrate automation, AI, and cloud services to achieve scalable and resilient operations (Owoade et al. 2024). Integration enables end-to-end process coverage and agility, but also increases dependency on heterogeneous systems and external vendors. Scalability promises growth and innovation, yet without careful governance it can amplify complexity and risk.

3.2 Workflow Automation

Building on the need to balance stability and agility in operational processes, workflow automation emerged as a means to systematically coordinate and streamline business workflows. It refers to the use of software systems (workflow management systems, WfMS) to orchestrate tasks, information flows, and decisions along a predefined business process model. According to industry standards, workflow automation entails routing documents, information, or tasks between participants according to procedural rules, with the goal of reducing manual effort and variability in execution (cf. Basu and Kumar 2002, p. 2). Early WfMS in the late 20th century were designed to make work more efficient, to integrate heterogeneous applications, and to support end-to-end processes even across organizational boundaries (cf. Stohr and Zhao 2001, p. 281).

One foundational aspect of workflow automation is PROCESS ORCHESTRATION. By encoding business procedures into formal process models that are executed by a *workflow engine*, organizations could enforce consistent process flows, improve speed and accuracy, and embed compliance checks into routine operations. In essence, pre-AI workflow automation provided a structured, deterministic way to implement business processes in software, directly addressing chronic issues like fragmented manual tasks and data silos in pursuit of OpEx (cf. Gadatsch 2012, p. 231). Orchestration denotes the centralized coordination of tasks and activities according to a defined process logic. In a typical WfMS, a workflow engine enacts the process model, dispatching tasks to the right resources (human or machine) in the correct sequence and enforcing the business rules at each step (Basu and Kumar 2002). This engine-driven coordination brings predictability and repeatability to workflows: tasks are executed in a fixed, optimized order with minimal ad-hoc variation. By systematically controlling task flow, early workflow systems could eliminate many manual hand-offs and delays, thereby boosting efficiency and consistency in outcomes. The orchestration approach essentially translated managerial routines into software: for example, an order processing workflow would automatically route an order through credit check, inventory allocation, shipping, and billing steps without needing human coordination at each transition. Such deterministic sequencing was crucial for achieving the quality and reliability targets of OpEx in an era before adaptive AI capabilities (cf. Stohr and Zhao 2001, pp. 283-285).

A closely related design concern is INTEGRATION. Workflow automation inher-

ently requires linking together diverse people, departments, and IT systems into an end-to-end process. Literature emphasizes that WfMS must integrate heterogeneous application systems and data sources to allow seamless information flow across functions (cf. Stohr and Zhao 2001, pp. 289-290). For instance, a procurement workflow might connect an ERP inventory module, a supplier's database, and a financial system so that each step can automatically consume and produce the necessary data. This integration extends beyond technical connectivity; it also encompasses coordinating work across organizational boundaries. As e-business initiatives grew in the 1990s and early 2000s, workflows increasingly spanned multiple organizations (suppliers, partners, customers), demanding inter-organizational process integration (Basu and Kumar 2002). Research in this period identified the need for distributed workflow architectures that could bridge independent systems and companies. Georgakopoulos et al. (1995) noted that existing workflow tools had limitations in complex environments, calling for infrastructure to handle heterogeneous, autonomous, and distributed information systems. In practice, this led to the development of interoperability standards (e.g., XML-based process definitions, web service interfaces) and process choreography protocols to ensure that a workflow could progress smoothly even when multiple organizations or platforms were involved. Effective integration was thus an essential condition for workflow automation, enabling the end-to-end automation of processes that formerly stopped at organizational or system boundaries.

To manage complexity and change, MODULARITY in workflow design became another important principle. Rather than hard-coding monolithic process flows, architects sought to break workflows into modular components or sub-processes that could be reused and reconfigured as needed. This component-based approach was accelerated by the rise of service-oriented architectures and e-business "workflow of services" concepts. For example, composite e-services and e-hubs allow organizations to construct complex workflows by composing smaller service modules. A modular workflow architecture improves maintainability: if a business rule changes or a new subprocess is required, one can update or insert a module without redesigning the entire workflow from scratch. Modularity also underpins adaptability. Ideally, a workflow systematizes routine functions but can be adjusted to accommodate new requirements or variations in the process (cf. Basu and Kumar 2002, p. 10). In other words, the literature suggests that well-designed workflow automation should combine standardization with flexibility: processes are structured into clear modules for the

"happy path" of routine operations, yet those modules can be reorchestrated or overridden in exceptional cases. This design philosophy reflects an early recognition that no single process model can anticipate all future conditions, so a degree of configurability must be built in (Georgakopoulos et al. 1995).

Despite efforts to introduce flexibility, traditional workflow automation faced notable challenges with EXCEPTION HANDLING. Exception handling refers to the ability of a system to cope with deviations, errors, or unforeseen scenarios that fall outside the predefined process flow. Basu and Kumar (2002) candidly observe that existing WfMS "tend to fall short whenever workflows have to accommodate exceptions to normal conditions"—i.e., when something unexpected occurs that was not explicitly modeled, the system often cannot resolve it autonomously, forcing human intervention. Typically, designers might anticipate a limited number of exception scenarios and build alternate paths for those (e.g., an approval escalation if a manager is absent). However, if a novel exception arises (say, a new regulatory requirement or an unplanned system outage affecting a step), the rigid workflow cannot handle it, and manual workarounds are needed. This problem of early workflows under dynamic conditions was widely acknowledged (Georgakopoulos et al. 1995). In the pre-AI era, most workflow automation remained predominantly rule-driven and inflexible outside of predefined contingencies. Exception handling thus stood out as a critical limitation of classical automation approaches, highlighting a gap between the desire for end-to-end automation and the reality of complex, ever-changing business environments.

Another salient theme in the literature is WORKFLOW GOVERNANCE—the structures and mechanisms for overseeing automated workflows and aligning them with organizational policies. As companies entrusted core business processes to software, ensuring the correct and intended execution of those processes became vital. Key governance considerations include monitoring, auditing, and controlling workflows (cf. Georgakopoulos et al. 1995, p. 131). A WfMS typically provides monitoring dashboards and logs so that managers can track the state and performance of process instances (e.g., to identify bottlenecks or errors). It also enforces role-based access control, ensuring that only authorized personnel perform certain tasks or approvals, which is essential for compliance in regulated industries. Basu and Kumar (2002) highlight the importance of organizational "metamodels" and control mechanisms that tie workflows to an enterprise's structure—for example, defining which organizational roles are responsible for each task and how escalation or overrides should

happen under specific conditions. Additionally, governance extends to establishing standards and best practices for workflow design and deployment. Industry coalitions and standards bodies (such as the WFMC in the 1990s) issued reference models and interface standards to promote consistency and interoperability in workflow implementations. In the context of inter-organizational workflows, governance also means agreeing on protocols and service-level commitments between partners so that automated interactions remain trustworthy and transparent. Overall, robust governance in workflow automation ensures not only efficiency but also accountability, security, and compliance. It addresses the managerial and oversight challenges that arise once processes are no longer directly handled by individuals but by software agents following prescribed logic.

3.3 Agentic Artificial Intelligence

Agentic AI refers to systems composed of multiple interacting generative agents that autonomously collaborate to achieve complex goals. It represents a shift beyond single AI agents toward orchestrated multi-agent ecosystems, enabled largely by recent advances in the field. Whereas traditional automation (e.g. rule-based RPA) executes predefined steps, an agentic architecture features adaptive, goal-directed agents that can perceive context, make decisions, and act with minimal hard-coded instructions (cf. Jennings et al. 1998, pp. 8-9). This idea builds on classic MAS principles of autonomy and social action (Castelfranchi 1998; Ferber 1999), but now agents are augmented with learning and reasoning capabilities from large language models (LLMs). Sapkota et al. (2025) highlight this paradigm shift by highlightin the difference between AI agents and agentic AI, framing the later as “multi-agent collaboration, dynamic task decomposition, persistent memory, and orchestrated autonomy” in pursuit of flexible problem-solving.

A defining trait of agentic AI is a higher degree of AUTONOMY. Agents can operate without constant human or central control, making and executing decisions in real time. Early MAS research already emphasized agent autonomy—e.g. agents as entities with independent control over their actions and state. Modern generative agents greatly amplify this autonomy by leveraging LLM-based reasoning to plan multi-step actions toward goals. For instance, frameworks like AutoGPT (Yang et al. 2023) demonstrated that a single LLM-based agent can iteratively break down objectives, choose actions, and adjust based on feedback without human

intervention. This autonomy promises agility and decision quality (agents can respond to situational changes or large search spaces beyond rigid scripts), but it also introduces risks. As Russell et al. (2015) caution, each additional decision delegated to an opaque AI agent shifts "ethical control" away from human operators. In enterprise settings, uncontrolled autonomous decisions might lead to policy violations or unsafe actions (Gaurav et al. 2025). Thus, an architectural challenge is balancing agent freedom with mechanisms to supervise or constrain critical decisions. In practice, this means designing agents with clearly scoped authorities, fail-safes, or escalation paths (e.g. requiring human confirmation for high-impact actions) to align autonomy with organizational policies.

TOOL-USE CAPABILITY is another hallmark of agentic AI architectures. Agents are not limited to their built-in knowledge; they can invoke external tools, APIs, or other services as part of their reasoning loop. This extends an agent's functionality—for example, an AI agent might call a database, run a code snippet, or query web services to gather real-time information. Research shows that augmenting LLM agents with tool integration significantly improves their problem-solving scope and accuracy. Notably, the ReAct paradigm interleaves an agents chain-of-thought with tool calls, allowing it to perceive (via queries) and act (via external operations) iteratively (Yao et al. 2023). Such designs transform static LLMs into dynamic cognitive agents that can perceive, plan, and adapt—a critical capability for complex, multi-step workflows.

For workflow automation, this means an agent can not only parse instructions but also execute parts of a workflow (e.g. trigger an RPA bot or send an alert) and then reason over the results. Architecturally, enabling tool use requires adding interface layers for the agent to safely interact with enterprise systems (APIs, databases, RPA scripts), along with policies on allowed tools. It's worth noting that tool integration adds orchestration complexity and potential error propagation paths (Sapkota et al. 2025). Therefore, designs often include an orchestration layer or planner agent that manages when and how tools are invoked, checks tool outputs, and handles exceptions (e.g. what if a tool fails or returns unexpected data). In summary, tool-use greatly enhances agent capabilities, but it demands careful architectural planning to manage the added complexity and ensure robust tool-agent interaction.

Agentic AI systems are inherently multi-agent comprising not one but many agents, often with SPECIALIZED roles, that must COORDINATE their efforts. This multi-agent approach stems from the insight that complex workflows can be decomposed: instead

of one monolithic AI agent trying to do everything, a team of agents can each handle subtasks and then combine results. Such specialization aligns with principles of OpEx (e.g. division of labor and expertise) and has been shown to improve performance. For example, Shu et al. (2024) report that a collaborative team of LLM-based agents achieved up to 70% higher success rates on complex tasks compared to a single-agent approach.

Architecturally, coordination mechanisms are crucial to harness these gains. Agents need to communicate their intentions, share data or results, and synchronize plans. The literature distinguishes coordination structures along two dimensions: hierarchy vs. flat and centralized vs decentralized decision-making. In a centralized hierarchical design, a top-level planner/manager agent delegates tasks to subordinate agents and integrates their outputs (akin to a project manager overseeing specialists). This can simplify global coordination and ensure alignment with a single source of truth (the planner's goal), at the cost of a single point of failure or bottleneck. Conversely, decentralized teams use peer-to-peer negotiation or voting; all agents are more equal and collectively decide on task assignments or conflict resolution (drawing on concepts from distributed AI and game theory). For instance, one recent system had developer agents jointly agree on a solution design without a central boss, mimicking consensus decision-making (Qian et al. 2024). Each approach has trade-offs: hierarchical control can be more efficient for well-structured processes (e.g., *workflow automation*), while decentralized collaboration may be more robust to single-agent failure and better for ill-structured problems (e.g., research).

Inter-agent communication protocols (what messages agents send and when) are another design facet—simple cases use direct message passing or shared memory, whereas more complex setups might use asynchronous communication. Importantly, specialization means each agent can be bounded in scope (e.g. a compliance checker agent vs. a data retrieval agent), which helps with scalability: each agent's LLM or reasoning module can operate within a focused context window, and different team members can even use different model types suited to their niche. This modularity and specialization, orchestrated through well-defined coordination logic, is a key architectural strength of agentic AI. It mirrors how human organizations structure teams for efficiency, and indeed is crucial for aligning multi-agent AI workflows with complex enterprise processes.

As agent behaviors become more autonomous and distributed, ensuring OBSERVABILITY of the system is vital. Observability here means that the internal states,

decisions, and actions of agents can be monitored and understood by humans or supervisory systems. Traditional MAS literature often dealt with observability in terms of state visibility (e.g. in partially observable environments), but in an enterprise context it translates to runtime transparency and traceability of what agents are doing and why. One challenge is that LLM-driven agents reason in natural language (or latent vectors), making their decision process somewhat opaque. Sapkota et al. (2025) highlight that AI agents "lack transparency, complicating debugging and trust", and they advocate for robust logging and auditing pipelines to make agent operations inspectable. In practice, agentic architectures include components to log key events: each prompt an agent generates, each tool API call and its result, each decision or plan the agent commits to, etc. Such audit logs enable post- hoc analysis, error tracing, and explanations—for example, if a workflow failed or a compliance issue occurred, developers can replay the agent interactions to pinpoint the cause. Some frameworks even expose an agent's chain-of-thought (the intermediate reasoning steps) in a controlled way for debugging or compliance review.

Beyond logging, observability can be enhanced through dashboarding and alerts: e.g. real-time monitors that track agent performance metrics or detect anomalies (like an agent taking too long on a task or generating an out-of-bounds output). The end goal is to treat an agentic AI system not as a "black box" automation, but as an observable workflow that operations teams can supervise akin to any critical IT system. This also ties into explainability: by capturing the rationale behind decisions (even if only in approximate form, such as storing the intermediate reasoning text), the system can later provide explanations for its actions, which is invaluable for trust and for continuous improvement. Overall, the literature suggests that designing for transparency—instrumenting agents with logging, and perhaps even designing agents to self-report their status—is a best practice to ensure agentic AI doesn't become an inscrutable tangle of automations. High observability supports OpEx principles by enabling traceability, accountability, and faster incident response when something goes wrong.

Finally, a recurrent theme is the need for strong GOVERNANCE mechanisms in agentic AI architectures to ensure alignment with rules, ethics, and organizational policies. By their nature, autonomous agents may produce unexpected or undesired outcomes—a risk amplified in multi-agent settings where interactions are complex and no single agent has full oversight. Without proper governance, an agentic system could easily violate compliance requirements or strategic constraints, undermining

OpEx goals (e.g. a well-intentioned agent might inadvertently expose sensitive data or execute an unauthorized transaction). In fact, Gaurav et al. (2025) warn that the "absence of scalable, decoupled governance remains a structural liability" in today's agentic AI ecosystems. To address this, researchers are exploring policy-enforcement layers that sit between the agents and the outside world. One such approach is Governance-as-a-Service (GaaS), a framework that intercepts agent actions at runtime and checks them against explicit rules or constraints. Rather than trusting each agent to self-regulate, an external governance layer can block or redirect high-risk actions, log rule violations, and even adapt penalties or restrictions on agents that exhibit misbehavior over time. This effectively creates an oversight controller for the MAS—analogous to a "compliance officer" in a human organization—that ensures no single agent can compromise the system's integrity.

Key design elements include declarative policy rules (defining allowable vs. disallowed outputs or tool uses), a mechanism to monitor all agent outputs (to flag violations), and possibly a trust score or reputation model to quantify an agent's reliability based on past behavior. Beyond automated enforcement, governance also encompasses human oversight: for example, requiring human approval for certain agent decisions (human-in-the-loop checkpoints) or having a fallback where a human operator can intervene if the agents encounter an ambiguous ethical situation.

In classical MAS research, analogous concepts existed like normative agents and electronic institutions that enforce "rules of engagement" among agents; the new twist is that with LLM-based agents we must often treat the models as black boxes, so governance can't be injected into their internal logic easily and must surround them instead. The overarching recommendation is that any architecture for generative multi-agent workflows should bake in governance from the start—not as an afterthought—to manage risk. As one author succinctly put it, such a system "does not teach agents ethics; it enforces them". This governance emphasis aligns tightly with OpEx goals of compliance, risk management, and trustworthiness (Gaurav et al. 2025).

In summary, the literature portrays agentic AI as an emerging paradigm for workflow automation that, if well-designed, can dramatically enhance agility, adaptability, and decision quality in operations. By combining autonomous, tool-using agents into coordinated architectures, organizations can automate complex processes that previously required human judgment. At the same time, achieving sustainable excellence with such systems demands careful attention to transparency and control:

architects must ensure agents remain observable and governable to uphold compliance and reliability standards. These insights set the stage for the next section of this thesis, which will integrate OpEx principles, workflow automation requirements, and agentic AI capabilities into a unified reference architecture. The themes of autonomy, coordination, and governance identified here directly inform the design choices and requirements elaborated in the subsequent chapters.

4 Requirements Modeling

The literature review identified recurring design concerns across operational excellence, workflow automation, and agentic AI. Synthesizing these insights yields *elicitation lists* which represent the initial outcome of requirements documentation based on systematically coded sources. A subsequent *clustering* step consolidated overlapping items into a unified set of requirement candidates.

Each candidate was then reformulated into an atomic, unambiguous, and verifiable "shall" statement, following the best-practice formulation rules of Glinz et al. (2020). The final requirements were organized into *functional requirements*, *quality requirements*, and *constraints*, in line with the Glinz taxonomy.

Requirements were then represented in SysML v2 as dedicated requirement elements, with their textual statements captured in the description field. Trace links connect each requirement to its source in the elicitation lists and to the architecture elements that «`satisfy`» it. This model-based representation ensures that design decisions remain traceable to validated needs and that requirement coverage can later be verified systematically.

4.1 Clustering and Consolidation

In order to derive actionable system requirements from the literature, a structured clustering process was applied. This eliminated redundancies, consolidated overlapping issues, and normalized vocabulary across disciplines. The process began by translating *coded statements* from each domain into elicitation items—discrete, *design-relevant units* derived from the coding units identified in the QCA process described in Section 2.1.

O — OPERATIONAL EXCELLENCE

1. ADAPTABILITY AND AGILITY — processes must remain reconfigurable in response to volatile conditions, ensuring resilience in dynamic environments.
2. COMPLIANCE AND RISK MANAGEMENT — regulatory adherence and transparency must be embedded into workflows to minimize compliance risks.
3. DECISION QUALITY — automation should enable data-driven, timely, and well-informed decisions rather than simply increasing speed.
4. EFFICIENCY AND CONTINUOUS IMPROVEMENT — workflows should reduce manual effort, eliminate waste, and institutionalize iterative refinements.
5. CUSTOMER-CENTRICITY — operations must align with user needs and service-level commitments to sustain value delivery.
6. USER EMPOWERMENT AND CULTURE — systems should support collaboration, transparency, and employee engagement in improvement processes.
7. TECHNOLOGY INTEGRATION AND SCALABILITY — architectures must accommodate automation, AI, and cloud services to enable sustainable innovation.

W — WORKFLOW AUTOMATION

1. PROCESS ORCHESTRATION — workflow engines must enforce task sequences and business rules to guarantee reliable execution.
2. INTEGRATION AND INTEROPERABILITY — automation must seamlessly connect heterogeneous applications, data sources, and organizational boundaries.
3. MODULARITY AND REUSABILITY — workflows should be composed of modular tasks or subprocesses that can be reused and reconfigured with minimal effort.
4. EXCEPTION HANDLING AND FLEXIBILITY — systems must detect, manage, and escalate deviations rather than failing in unforeseen scenarios.
5. WORKFLOW GOVERNANCE — monitoring, audit trails, and role-based controls must ensure accountability and compliance throughout automated processes.

A — AGENTIC AI

1. AUTONOMY IN DECISION-MAKING — agents should operate independently within clearly scoped authority to enhance agility while managing risks.
2. TOOL USE AND INTEGRATION — agents must invoke external tools, APIs, or services reliably, requiring robust interfaces and safeguards.
3. COORDINATION AND SPECIALIZATION — multi-agent systems should divide labor through explicit roles and structured coordination mechanisms.

4. OBSERVABILITY AND TRANSPARENCY — all agent actions and decisions must be logged and explainable to support trust, debugging, and compliance.
5. GOVERNANCE AND COMPLIANCE — oversight mechanisms, including policy enforcement layers and human-in-the-loop checkpoints, are essential to align agent behavior with organizational and ethical standards.

To reduce redundancy and ensure conceptual clarity, the elicitation items from each domain were compared and consolidated based on semantic similarity and functional overlap. Particular attention was given to cross-domain intersections—such as governance appearing in both workflow automation and agentic AI—and to areas where multiple coding units aligned on a shared concern. The result of this consolidation step is a reduced set of requirement candidates, each traceable to one or more domain-specific clusters. These are summarized in Table 4.1, which maps the original coded clusters to the consolidated requirement areas used for formulation.

CLUSTER IDS	SOURCE DOMAINS	CONSOLIDATED REQ. AREA
O1	OPEX	ADAPTABILITY AND AGILITY
O4	OPEX	CONTINUOUS IMPROVEMENT
O5	OPEX	CUSTOMER-CENTRICITY
O6	OPEX	USER EMPOWERMENT
O7	OPEX	TECH. INTEGRATION & SCAL.
O2, W5, A5	ALL THREE	GOVERNANCE & COMPL.
O3, A1	OPEX, AGENTIC AI	DECISION QUALITY
W1, W3	WORKFLOW AUTOMATION	ORCHEST. & MODULARITY
W4, A3	WORK. AUT., AGENTIC AI	EXCEP. HANDLING & COORD.
A4, O2	AGENTIC AI, OPEX	OBSERVABILITY & TRAC.
W2, A2	WORK. AUT., AGENTIC AI	TOOL INTEGRATION

Table 4.1: Mapping of domain-specific clusters to consolidated requirement areas (in-scope areas highlighted in gray).

This structured consolidation establishes a coherent foundation for refining architecture-level requirements. To maintain architectural focus and avoid inflation of the requirement set, areas originating solely from managerial OpEx—namely, adaptability and agility, continuous improvement, customer-centricity, user empowerment,

and technology integration and scalability—were *not* modeled. Their architecturally relevant aspects were absorbed into cross-domain clusters, while non-architectural concerns were acknowledged as higher-level managerial frameworks, guidelines, and principles rather than system requirements.

4.2 Reformulation and Classification

The logical next step, consistent with RE methodology, is to dissect each consolidated requirement area individually and distill it into a set of atomic "shall" statements. This transformation emphasizes clarity, necessity, and verifiability, ensuring that each requirement stands alone as an actionable directive. At this stage, the focus remains on precision and alignment with design intent; formal classification into functional, quality, or constraint types follows only after this refinement process. Several requirement areas share overlapping architectural concerns. This reflects systemic interdependencies, particularly in adaptive, agent-based architectures and, more broadly, in complex systems. To avoid redundancy, each requirement is assigned to the cluster that most directly motivates it.

To address the tradeoff between best-practices and readability, a deliberate compromise was made in the formulation of requirements: strict atomic decomposition was applied selectively. In cases where overly granular formulation would hinder readability or inflate the requirement set without clear architectural benefit, semantically related concerns were consolidated into cohesive statements. Furthermore, the scope of formal requirements was restricted to those directly affecting system architecture—such as structure, behavior, and coordination—while non-architectural concerns (e.g., organizational or cultural aspects) were acknowledged in the literature but excluded from the formal specification.

GOVERNANCE AND COMPLIANCE The literature consistently frames governance and compliance as architectural, not merely legal, concerns. Compliance should be embedded as a first-class constraint, with governance mechanisms preventing compliance drift as systems optimize or adapt. Classical workflow automation contributes monitoring, audit trails, and escalation, and has evolved toward policy-driven designs that decouple governance rules from core logic. Agentic AI requires a decoupled policy layer capable of intercepting and vetoing high-risk actions, complemented by human-in-the-loop checkpoints and comprehensive auditability. Across domains,

governance and compliance must operate as active, adaptable components—enforcing policy at runtime while preserving transparency and alignment with organizational and legal constraints (Basu and Kumar 2002; Gaurav et al. 2025).

- [FR] POLICY ENGINE
- [C] SEGREGATION OF DUTIES
- [FR] RISK-BASED APPROVALS AND ESCALATION
- [C] AUDIT LOGGING AND RETENTION
- [FR] COMPLIANCE MAPPING AND DRIFT CHECKS

DECISION QUALITY The literature portrays decision quality as producing reliable, evidence-based outcomes rather than merely increasing speed. In operational excellence, this means balancing rapid execution with transparency and accountability. Classical workflow automation contributes rule enforcement, role clarity, monitoring, and escalation to standardize and audit decision points. Agentic AI extends decision reach but must be bounded and observable: agents should operate within scoped authority, log the context of their choices (prompts, tool interactions, plan commitments), and escalate high-impact actions to human review. Together, these insights imply that architecture must combine evidence aggregation with rule-based consistency, human checkpoints for risk, and traceability of decision pathways to sustain quality under change. Based on these insights, the following requirements were derived:

- [FR] EVIDENCE-BASED DECISION SUPPORT
- [FR] RULE-ENFORCED DECISION POINTS
- [FR] DECISION TRACE AND RATIONALE
- [FR] RISK-BASED HUMAN APPROVAL
- [C] BOUNDED DECISION AUTONOMY

ORCHESTRATION AND MODULARITY The literature describes classical workflow automation as the enactment of formal process models by a central engine that coordinates human and machine tasks in a defined order and enforces business rules for predictable, repeatable execution. To manage change and complexity, workflows should be composed of modular subprocesses or services that can be reused and reconfigured without redesign, often realized through service-oriented compositions. Interoperability standards and clear interfaces enable module composition across

heterogeneous systems and organizational boundaries. In OpEx terms, parameterized designs allow variation without structural overhaul, preserving efficiency while enabling adaptation. Based on these insights, the following requirements were derived:

- [FR] PROCESS ORCHESTRATION ENGINE
- [C] MODULAR PROCESS UNITS
- [C] INTERFACE-BASED COMPOSITION
- [FR] PARAMETERIZED SUBPROCESSES
- [FR] HUMAN-MACHINE TASK ORCHESTRATION

EXCEPTION HANDLING AND COORDINATION The literature notes that classical workflow systems struggle with unforeseen exceptions; predefined alternates help, but novel cases often require human intervention. Architectures should therefore detect and route deviations to defined exception paths with appropriate escalation. In agentic AI, multi-agent designs decompose complex work into specialized roles, which demands explicit coordination structures—ranging from hierarchical planner-specialist patterns to decentralized negotiation—plus clear communication protocols (message passing or shared memory for asynchronous cases). Together, robust exception pathways and well-specified coordination ensure that workflows remain reliable when reality diverges from the "happy path," while agent teams synchronize decisions without conflict or drift. Based on these insights, the following requirements were derived:

- [FR] EXCEPTION DETECTION AND ROUTING
- [FR] ESCALATION TO HUMAN AUTHORITY
- [C] EXPLICIT COORDINATION MODEL
- [FR] INTER-AGENT COMMUNICATION PROTOCOL
- [FR] TOOL FAILURE HANDLING IN EXCEPTIONS
- [FR] CONFLICT RESOLUTION

OBSERVABILITY AND TRACEABILITY The literature emphasizes that enterprise automation must be inspectable at runtime and explainable post hoc. Classical workflow systems contribute monitoring and logs for process execution visibility. Agentic AI adds opacity risks; to mitigate them, architectures should log agent prompts, tool interactions, and plan/decision commitments, expose dashboards and

alerts for anomaly detection, and support replay so that failures and outcomes can be reconstructed and explained. Across domains, observability and traceability provide accountability, faster incident response, and the basis for explaining why a particular decision or action occurred. Based on these insights, the following requirements were derived:

- [FR] EVENT LOGGING PIPELINE
- [FR] DECISION TRACE AND RATIONALE
- [FR] DASHBOARDS AND ALERTS
- [FR] REPLAY FOR POST-HOC ANALYSIS
- [C] AGENT STATUS SELF-REPORTING

TOOL INTEGRATION The literature presents integration as foundational to both classical workflow automation and agentic AI. Workflow systems must connect heterogeneous applications and data sources, often across organizational boundaries, using clear interfaces and interoperability standards. Agentic AI adds tool-use: agents invoke enterprise APIs, databases, or RPA scripts through interface layers, while an orchestration component manages when and how tools are called and checks results. Architecturally, tool integration therefore centers on well-defined adapters and contracts, protocol-level interoperability, and data transformation to enable end-to-end workflows without brittle coupling. Based on these insights, the following requirements were derived:

- [FR] INTEGRATION CONNECTORS
- [FR] AGENT TOOL ADAPTERS
- [C] INTERFACE CONTRACTS AND SCHEMAS
- [FR] INTER-ORGANIZATIONAL INTEROPERABILITY
- [FR] DATA TRANSFORMATION LAYER
- [C] INVOCATION SAFEGUARDS

The requirements were then consolidated into a final list, grouped by class—*functional requirements* and *constraints*—to provide a single, unambiguous baseline for the subsequent SysML formalization. Here an example of each requirement class:

FR-09 POLICY ENGINE The system shall provide a decoupled policy evaluation component that can validate, veto, or redirect workflow executions and agent actions

at runtime.

C-03 BOUNDED DECISION AUTONOMY The system shall constrain agent decision-making to clearly scoped authority levels aligned with organizational objectives.

The complete list of requirements can be found in Appendix A.1. As the reader can see, *quality requirements* were left out of scope at this stage because this thesis focuses on specifying *what* the system must do and *which* governance and operational constraints it is subject to. Moreover, quality attributes vary significantly by deployment context and would require domain-specific targets, which remain out of scope here. Some quality-related concerns (e.g., auditability, interoperability) were operationalized as *constraints* rather than standalone quality requirements.

While the list optimizes precision, it is not yet visually informative nor operationally easy to manage. In the next section, the requirements are recast as SysML «`requirement`» elements and diagrams to improve visibility, traceability, and change control—linking them explicitly to their sources and preparing `satisfy`/`verify` relations for the architectural and evaluation work that follows.

4.3 Model-Based Representation

The requirements were encoded into a SysML v2 model that serves as the central design artifact for the remainder of this work. Concretely, each refined "shall" statement is represented as a dedicated SysML `<<requirement>>` element whose description preserves the validated text verbatim. To make the model auditable and operable, each element carries lightweight metadata (stable identifier, short title, source, and cluster) and is *anchored* to the system-of-interest via the element's *subject*. This anchoring establishes the scope of each requirement without committing to concrete architectural structure at this stage. For navigability and later analysis, requirements are organized into subpackages that mirror the six clusters consolidated in §4.2 (governance and compliance; decision quality; orchestration and modularity; exception handling and coordination; observability and traceability; tool integration). The model further *prepares* standard SysML relations (`<<satisfy>>`, `<<verify>>`) that will be instantiated in §5 once architectural elements are introduced; this sequencing preserves methodological clarity while ensuring traceability is planned

from the outset. Taken together, the requirement model forms a coherent, machine-interpretable blueprint that unifies functional directives and constraints in one representation and supports systematic evolution throughout the design process (IEEE 1990).

This subsection focuses exclusively on how the requirements are represented; architectural structure and behavior are treated in §5. The following decisions govern the model:

- *Requirement element and metadata.* Each validated statement is captured as a SysML `<<requirement>>` with: (i) a stable ID matching the consolidation in §4.2 (e.g., `FR-01`, `C-03`), (ii) a short title to aid readability, (iii) the original text in the description (self-documenting model), (iv) a `source` tag referencing the consolidation artifact, and (v) a `cluster` tag indicating its organizational grouping. This keeps the model faithful to the source while enabling queries and views.
- *Subject anchoring.* Every requirement has a *subject* set to the system-of-interest. The subject establishes what the requirement constrains or obligates, while deliberately avoiding bindings to specific blocks or interfaces before the architecture is introduced. This prevents premature design commitments while keeping semantics precise.
- *Granularity and verifiability.* Composite statements are decomposed until each requirement is atomic, necessary, and verifiable. Where a statement mixes an obligation with a performance bound or policy constraint, it is split into a functional requirement (the obligation) and one or more constraint requirements (the bounds). This respects the consolidated content yet ensures each element is testable.
- *Functional vs. constraint tagging.* The model distinguishes requirement *intent* via a lightweight tag: *Functional* for obligations on behavior/capability; *Constraint* for limits, policies, or non-functional bounds. This tagging supports later viewpoints (e.g., verification planning) without introducing parallel taxonomies.
- *Derivation and refinement.* When a consolidated statement is systematically specialized (e.g., split by scenario or mode), the resulting elements are connected with `derive` or `refine` relationships. These links are intra-requirement only at this stage; links to design are deferred to §5.
- *Verification planning metadata.* Each requirement optionally records a *planned*

verification method (Analysis, Inspection, Test, or Demonstration) as metadata. The corresponding `<<verify>>` links are created in §5 when test or analysis artifacts exist, keeping §4.3 strictly about requirement representation while making verification intent explicit (IEEE 1990).

- *Organization by cluster.* Requirements are grouped into six subpackages, one per cluster from §4.2. Cross-cutting concerns (e.g., governance or observability) remain in their home cluster but may later be *traced* to multiple architectural elements; organizing by cluster improves readability without pre-judging component boundaries.
- *Change provenance.* Each requirement preserves a brief rationale note (when relevant) and the source reference to the consolidation step. This enables impact analysis: a change in a requirement's text or status can be traced back to the originating consolidation item.

The following textual snippet shows how a consolidated statement becomes a model element while remaining architecture-agnostic:

```
requirement def FR_01_ProcessOrchestrationEngine {
  attribute id     = "FR-01";
  attribute title  = "Process orchestration engine";
  attribute cluster= "OrchestrationAndModularity";
  attribute source = "Consolidation §4.2 / Appendix A.1";
  subject soi : SystemOfInterest;
  doc /* The system shall execute workflow models by dispatching tasks
        to human or software actors according to model control flow
        and business rules. */
 // Trace and verification links are prepared here, and instantiated in §\ref{sec:mod-mas}.
}
```

This form preserves the validated text, records minimal yet sufficient metadata, and binds the requirement to the system-of-interest. Additional derived or refined elements (e.g., for parameterized subprocesses or exception routing) are represented similarly and related via `derive`/`refine` within the requirement package.

The model establishes *where* trace links will exist and *how* they will be expressed, but it intentionally defers instantiation of `<<satisfy>>` and `<<verify>>` until §5 introduces architectural and verification artifacts. This preserves a clean separation of concerns while aligning with IEEE's notion of traceability as the systematic linkage of requirements, design, and verification artifacts (IEEE 1990). Once §5 defines

architectural elements, each requirement receives explicit <<satisfy>> links from the corresponding structural or behavioral elements, and verification cases attach via <<verify>> according to the planned methods.

Representing the requirements in SysML v2 yields three immediate benefits: (i) it consolidates all validated needs into a single, analyzable model with explicit scope and provenance; (ii) it enforces conceptual integrity by keeping terminology and interfaces consistent in one place and enables early review against the requirements (e.g., checking that exception scenarios can be given handling paths once behavior is introduced); and (iii) it creates a stable basis for subsequent instantiation, allowing the model to guide development as a living blueprint (Peffers et al. 2007). These outcomes support the design-science contribution while remaining faithful to the constraint that architecture-specific content is presented in §5.

The complete SysML v2 requirement package (textual listing) is provided in Appendix A.1.

5 Architecture Modeling

This section translated the validated requirement model from §4 into an architectural description suitable for design, analysis, and later verification. To preserve separation of concerns, the representation choices of §4.3 were taken as given; instantiation of trace links (<<satisfy>>, <<verify>>) and the introduction of concrete design elements were deliberately performed here. The architecture was kept technology-agnostic and context-aware: it respected legacy coexistence, policy governance, and auditability-by-design while avoiding premature selection of platforms or products.

The section was organized to make the path to the result explicit. First, the scope, assumptions, and architectural drivers were derived from the requirement clusters, and the modeling viewpoints were fixed to address those drivers systematically (§

5.1 Architectural Drivers and Viewpoints

Before introducing concrete structures and behaviors, the scope of the architecture was first defined, the principal assumptions and constraints were recorded, and the architectural drivers were derived from the requirement clusters in §4.2-§4.3 (see Appendix A.1 for the complete list). On this basis, the modeling viewpoints to be

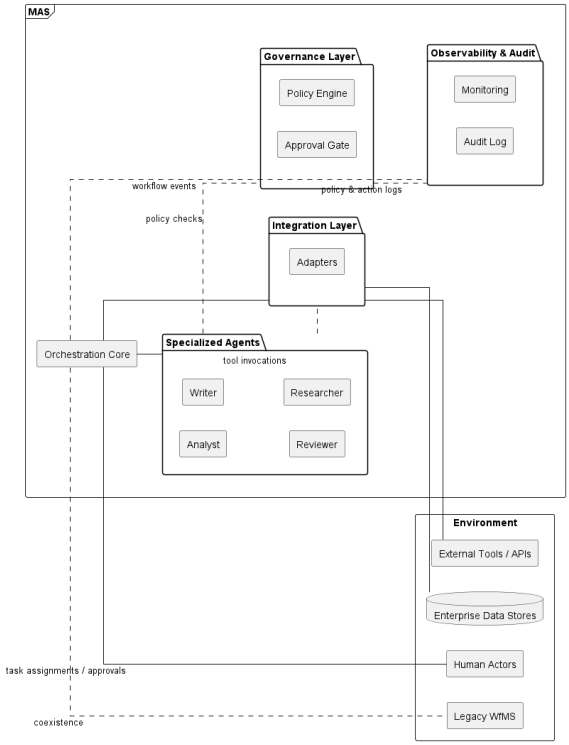

Figure 5.1: The system-of-interest was bounded with respect to human actors and legacy systems, highlighting the orchestration core, specialized agents, governance, integration, and observability.

used in §5 were selected so that the path from validated needs to design decisions remained explicit and inspectable (Peffers et al. 2007).

The architecture targeted a conceptual, technology-agnostic multi-agent system (MAS) for enterprise workflow automation. The *system-of-interest* comprised: (i) an orchestration core coordinating human and software tasks, (ii) specialized agents contributing analysis, drafting, or review capabilities, (iii) a governance layer enforcing policies and approvals, (iv) an integration layer connecting external tools and data sources, and (v) observability mechanisms for runtime inspection and audit. Organizational processes, cultural programs, or purely managerial OpEx initiatives were treated as context; only their architecturally relevant aspects (e.g., policies constraining runtime behavior) were carried into the design scope.

The following design assumptions and boundary conditions governed subsequent choices:

- *Legacy coexistence.* Existing systems (WfMS, data stores, line-of-business applications) remained in place; the architecture had to integrate without

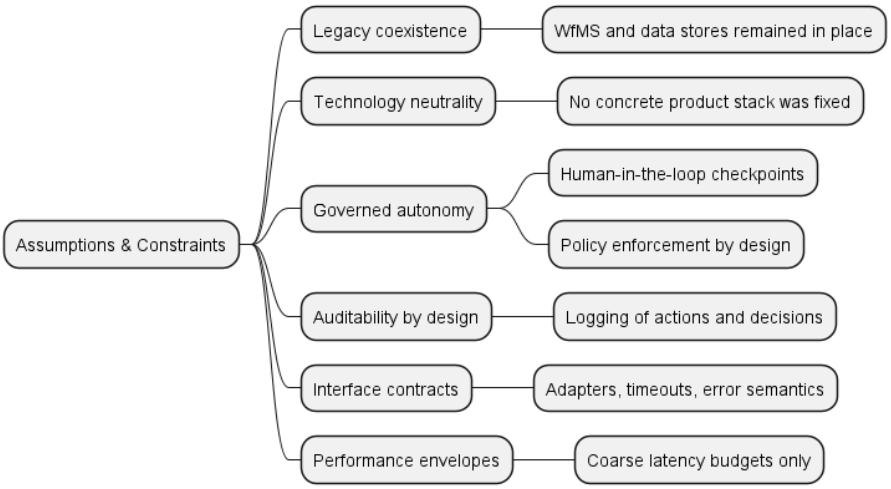

Figure 5.2: The principal assumptions and constraints framed subsequent architectural choices (legacy coexistence, technology neutrality, governed autonomy, auditability by design, interface contracts, and coarse performance envelopes).

requiring systemic replacement.

- *Technology neutrality.* No concrete product stack was prescribed in §5. The description remained at the level of components, roles, and interfaces so that multiple implementations would remain feasible.
- *Governed autonomy.* Agents operated under explicit policies and human oversight; high-risk actions required approvals. This constraint was treated as first-class rather than an afterthought.
- *Auditability by design.* All material actions (workflow transitions, tool invocations, agent decisions) were required to be observable and reconstructable for verification and compliance purposes (IEEE 1990).
- *Interface contracts.* External tool access occurred through well-defined adapters with error handling and fallbacks; direct, ad hoc coupling to third-party APIs was excluded.
- *Performance envelopes.* Only coarse bounds (e.g., end-to-end latency budgets sufficient for human-in-the-loop work) were imposed in §5. Quantitative tuning was deferred to implementation-specific work.

Consistent with the consolidated requirement clusters, the following concerns drove the architecture and framed trade-offs:

- *Governance and Compliance.* Policy enforcement, approval gates, and end-

to-end audit trails were required to be embedded into workflow and agent interactions, not bolted on later (Cluster: Governance&Compliance).

- *Decision Quality.* Decisions produced or assisted by agents had to be explainable and reviewable, with recorded inputs, rationales, and thresholds for escalation (Cluster: DecisionQuality).
- *Orchestration and Modularity.* Processes were to be composed from reusable units (subprocesses, services, agent capabilities) coordinated by an orchestration core, enabling substitution and incremental extension (Cluster: Orchestration&Modularity).
- *Exception Handling and Coordination.* Defined patterns for anomaly detection, recovery, and human escalation were required; multi-agent coordination had to avoid deadlocks and ensure progress (Cluster: ExceptionHandling&Coordination).
- *Observability and Traceability.* Uniform logging of states, prompts/tool calls, and workflow transitions was mandated, enabling replay, monitoring, and audit (Cluster: Observability&Traceability).
- *Tool Integration.* Access to heterogeneous external systems occurred via adapters with contracts for data shape, error semantics, and timeouts (Cluster: ToolIntegration).

These drivers operationalized the validated requirements into concrete concerns to be satisfied by the architecture. They also provided the anchor for trace links from requirements to design artifacts and later verification items (IEEE 1990).

To keep the description focused and consistent with §4.3, four complementary SysML viewpoints were chosen. Each viewpoint addressed specific driver concerns and prepared traceability for §5.2-§5.3:

- *Requirements/Traceability View.* The requirement elements from §4.3 (with IDs and cluster tags) were taken as the canonical source and linked to architectural elements via `<<satisfy>>` once those elements were introduced in §5.2. This view ensured that governance, decision-quality, and observability constraints were explicitly mapped rather than assumed (IEEE 1990).
- *Structural View.* Using block-definition and internal-block diagrams, the static composition (orchestrator, agent roles, governance services, adapters, data stores) and their interfaces were specified. This view made modularity and integration decisions explicit and provided the loci for `<<satisfy>>` links in

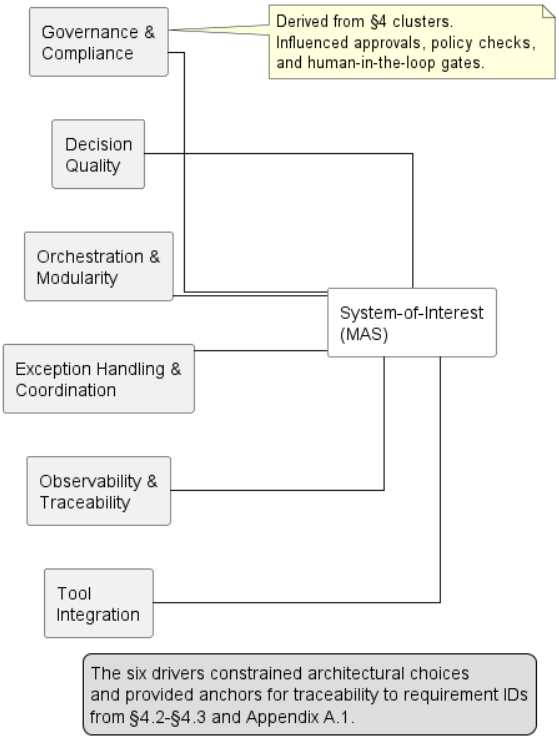

Figure 5.3: The six architectural drivers were derived from the consolidated clusters in §4 and were used to steer design decisions.

§5.2.

- *Behavioral View.* Activity and sequence diagrams captured coordination patterns, nominal flows, and exception paths (including human-in-the-loop checkpoints). This view instantiated decision-quality and exception-handling drivers and prepared `<<verify>>` hooks for test scenarios in §5.3.
- *Parametric/Constraints View.* Where relevant, policy and performance envelopes were represented as constraints tied to structural or behavioral elements (e.g., approval thresholds, timeout policies). Detailed quantitative models were deliberately deferred; only architecturally binding constraints were recorded here.

By defining scope and assumptions up front, deriving drivers from §4's clusters, and selecting viewpoints aligned to those drivers, the basis for §5.2 (structural decomposition and interfaces) and §5.3 (behavioral coordination and traceability) was established. The organization ensured that each design decision could be traced back to a specific driver and, ultimately, to a validated requirement in Appendix A.1, thereby preserving the separation of concerns introduced in §4.3 while enabling

systematic instantiation in §5 (IEEE 1990; Peffers et al. 2007).

5.2 Structural Decomposition and Interfaces

Guided by the drivers in §?? and the validated requirements in §4.3 (Appendix A.1), the *system-of-interest* was instantiated as a layered multi-agent system. The structure comprised: (i) an orchestration core coordinating role-specialized agents, (ii) a governance/policy service enforcing approvals and guardrails, (iii) an integration/adapter layer encapsulating access to external systems, and (iv) observability components for audit and replay. Each element was introduced with explicit interface contracts so that the path from requirements to design decisions remained inspectable and traceable via `<<satisfy>>` links (IEEE 1990).

Orchestration core. The *orchestrator* acted as control plane for task decomposition, assignment, and aggregation. It maintained workflow state, routed *TaskSpec* objects to agents, coordinated human-in-the-loop checkpoints, and consulted the governance service prior to high-risk actions. The orchestrator also accessed shared adapters where cross-cutting lookups were required. This element provided the structural locus for mapping orchestration and exception-handling requirements to concrete design artifacts.

Agent roles. Role-specialized agents (*planner*, *researcher*, *analyst*, *reviewer*, *writer*) encapsulated distinct capabilities. Each agent implemented a uniform task port and produced typed *TaskResult* outputs. This separation supported modular substitution and progressive refinement of capabilities without affecting the orchestrator or other layers. Where peer collaboration was beneficial, controlled agent-to-agent exchanges were permitted using the same message schema to avoid ad hoc couplings.

Governance/policy service. The *compliance* service externalized policies, approvals, and guardrails. Agents and the orchestrator performed policy checks before executing sensitive operations and recorded decisions for audit. Treating governance as a first-class service ensured consistent enforcement across the MAS and enabled explicit trace links from governance requirements to concrete policy checks (IEEE 1990).

Integration/adapter layer. The *integrations* package (*mcp*, *a2a*, *toolProxy*) encapsulated access to heterogeneous backends through stable contracts. Adapters exposed named *operations* with declared parameter/response shapes and error semantics. Agents invoked tools only through these adapters; direct coupling to third-party

APIs was excluded by design. This layer localized variability of external systems and preserved technology neutrality set in §??.

External systems and observability. External data stores, line-of-business applications, and services were represented as environment elements behind adapters. The *observer* and *auditLog* components captured material actions (workflow transitions, tool invocations, policy outcomes) to enable replay, inspection, and verification in §5.3, thereby operationalizing auditability-by-design (IEEE 1990).

Interface contracts. To make composition explicit and to support later verification, the following contracts governed interactions across layers. Each contract was specified with message names, payload fields, and error semantics (timeouts, retries, compensations), and was realized as provided/required ports in the structural view:

- *Task interface (orchestrator–agent).* The orchestrator provided `assignTask(taskSpec)` and `getResult(taskId)`; each agent required these and returned a typed *TaskResult*. Idempotency keys and correlation identifiers were mandatory to ensure replay safety.
- *Peer interface (agent–agent).* Where permitted, agents exchanged `requestInfo(query)` and `provideInfo(data)` using the same schema as the task interface. Policy tags were attached so that governance checks could be enforced uniformly.
- *Policy service (client–compliance).* Clients issued `checkPolicy(policyId, context)` and recorded decisions via `logEvent(event)`. Negative decisions included explanatory rationales to support decision quality and later review.
- *Tool invocation (client–adapter).* Clients called `invokeTool(toolId, params)` and received `ToolResult` with standardized success/failure envelopes. Adapters declared timeout budgets and compensations for partial failures.
- *Observability sink (producer–observer/auditLog).* Producers emitted `emit(event)` for state transitions, prompts, tool calls, and policy outcomes. Event schemas included timestamps, actor identifiers, and integrity digests to support verification.

5.3 Behavioral Coordination and Verification

This section realizes the structural design from §5.2 by defining the dynamic workflow of agent interactions and the artifacts used for verification. A central *orchestrator* supervises the process, dispatching work to specialized agents (planner, researcher,

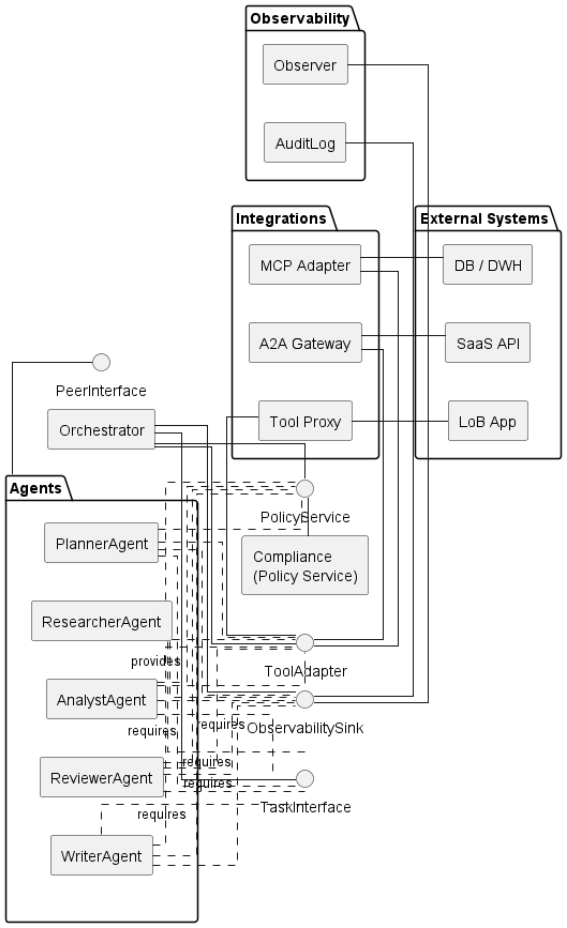

Figure 5.4: Composite structural view and contracts. The *orchestrator* provided a uniform task interface to role-specialized agents; agents and orchestrator invoked tools via adapters; policy checks were routed to *compliance*; and material events were emitted to *observer*/*auditLog*. External systems remained reachable only behind adapters.

analyst, writer, reviewer) and enforcing governance. Each agent executes a clearly-scoped task via the `Task` interface, while the orchestrator invokes external tools only through the `ToolAdapter` and checks every critical action via the `PolicyService`. For example, the orchestrator sequences tasks and treats sub-agents as "tools", and a decoupled policy engine can validate or veto actions at runtime. All agent actions and decisions emit structured events to the `ObservabilitySink` (observer/auditLog channels) to build an execution trace. This coordination supports the architecture drivers (e.g., GovernanceCompliance through policy gates and audit logging; OrchestrationModularity via clear task handoffs) and produces artifacts (logs, traces)

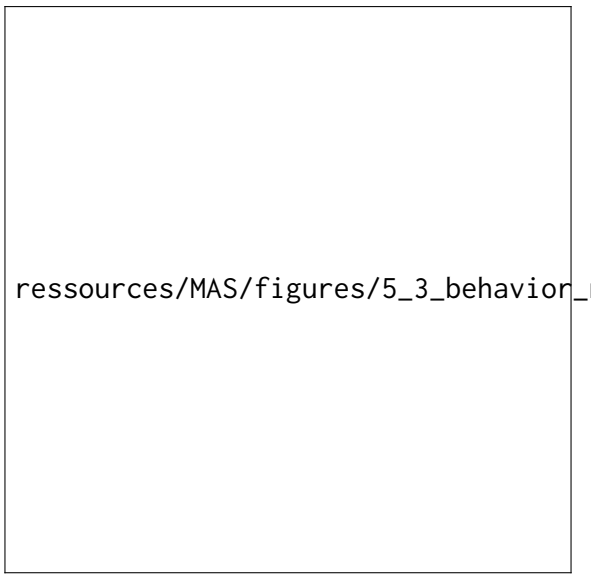

Figure 5.5: Nominal behavioral coordination. The *orchestrator* delegated tasks to role-specialized agents, invoked tools exclusively via adapters, enforced policy checks at defined gates, and emitted material events to *observer*/*auditLog*.

that tie back to requirements for verification. In the nominal case, the orchestrator begins by receiving a top-level task and decomposes it into subtasks. It assigns these in sequence: first the planner, then the researcher, followed by the analyst, writer, and finally the reviewer. Each agent performs its duty and returns results to the orchestrator via `Task`. External services or tool calls (e.g. via `mcp` or `a2a` interfaces or a `toolProxy`) are invoked only through the `ToolAdapter`, ensuring uniform request/response handling. Before executing any step that modifies data or affects outcomes, the orchestrator calls the `PolicyService`; the policy engine can approve the action or veto/redirect it according to rules. At each step, the system emits a timestamped event to the `ObservabilitySink`, building an audit log of decisions and tool invocations. This nominal coordination pattern (illustrated in Figure 5.5) ensures that each agent's role is clear, all tool uses are mediated by adapters, and governance checks and observability are applied at defined gates. The architecture also addresses concurrency and synchronization. To improve throughput, independent subtasks (especially data-retrieval or "reading" tasks) can be executed by spawning multiple agents in parallel under orchestrator control. However, the design avoids uncontrolled parallelism: all tasks carry a unique correlation ID via the `Task` interface, enabling matching of responses and idempotent retries. For operations that produce shared outputs, parallel execution is carefully managed. In

particular, parallel "write" or content-generation tasks are serialized or merged by a dedicated integrator agent to prevent inconsistency. In general, the system favors a breadth-then-depth strategy: multiple agents may gather information in parallel, but final synthesis or authoring is done by a single agent under the orchestrator's coordination. A shared context (e.g. a blackboard) holds intermediate results so that each agent sees the relevant state. These measures – task correlation via unique IDs, adapter-level idempotency, and controlled merging of results – ensure consistent state across agents while allowing safe parallelism.

- *Timeouts and retries.* The `ToolAdapter` enforces timeouts on external calls and automatically retries them as needed (using idempotency keys), recording each retry or failure event to the auditLog.
- *Policy denial with rationale.* If a planned action is disallowed by the PolicyService, the orchestrator aborts it and emits an audit event that includes the policy rule or justification for denial.
- *Low-confidence escalation.* The orchestrator detects low-confidence or undefined scenarios (fail-safe conditions) and escalates these cases to a human supervisor or higher authority. Such escalations are logged, and automated agents pause until resolution.
- *Conflict resolution.* When agents' outputs or decisions conflict, the orchestrator invokes a resolution mechanism (e.g. replanning or consensus procedures) in accordance with the conflict-resolution requirements. The outcome and any agent negotiations are logged.
- *Compensation for partial failure.* If an agent or tool invocation fails persistently (even after retries), the system triggers predefined compensation or recovery actions (as specified by exception subprocesses). Recovery steps (such as invoking alternative tools or notifying humans) are recorded in the audit trail.

The verification of this behavioral model is achieved via explicit links between behaviors and requirements. Each interaction scenario is tagged with relevant requirement IDs from Appendix A.1 (e.g. FR-09, FR-10 for policy checks, FR-07/FR-08 for exception handling). UML «verify» relations connect sequence diagrams to these requirements, and test cases use the auditLog as an oracle to check expected event sequences and outputs. For example, an acceptance test might replay a task sequence and assert that denied actions produce a corresponding "policy denied" event and no unauthorized changes. Message schemas and policy decisions are validated against the

defined interface contracts and policy rules, and performance bounds (response times, throughput) are checked against the envelopes from §5.1. This approach follows standard verification practices (e.g. IEEE software engineering definitions), ensuring that each behavior satisfies its design constraints and acceptance criteria. Each element of the coordination behavior maps back to the architecture's requirement clusters. Policy enforcement and audit logging fulfill the GovernanceCompliance and ObservabilityTraceability drivers (audit trails and human checkpoints), structured task delegation implements OrchestrationModularity, error paths address ExceptionHandlingCoordination, and tool-adapter patterns satisfy ToolIntegration. In this way, every behavioral flow can be traced to the corresponding requirement cluster (e.g. governance checks to GovernanceCompliance, decision trace logging to ObservabilityTraceability).

References

Basu, Amit and Akhil Kumar (2002). "Research Commentary: Workflow Management Issues in e-Business". *Information Systems Research* 13, pp. 1–14.

Carvalho, André M. et al. (2023). "Operational excellence, organizational culture, and agility: bridging the gap between quality and adaptability". *Total Quality Management & Business Excellence* 34, pp. 1598–1628.

Castelfranchi, Cristiano (1998). "Modelling social action for AI agents". *Artificial Intelligence* 103, pp. 157–182.

Ferber, Jacques (1999). Multi-agent systems: an introduction to distributed artificial intelligence. 1st ed. Harlow Bonn: Addison-Wesley, pp. 479–498.

Gadatsch, Andreas (2012). Grundkurs Geschäftsprozess-Management: Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker. 7. Wiesbaden: Vieweg+Teubner Verlag.

Gaurav, Suyash et al. (2025). Governance-as-a-Service: A Multi-Agent Framework for AI System Compliance and Policy Enforcement.

Georgakopoulos, Diimitrios et al. (1995). "An overview of workflow management: From process modeling to workflow automation infrastructure". *Distributed and Parallel Databases* 3, pp. 119–153.

Glinz, Martin et al. (2020). Handbook for the CPRE Foundation Level according to the IREB Standard. International Requiremenst Engineering Board.

Herrmann, Andrea (2022). Grundlagen der Anforderungsanalyse: Standardkonformes Requirements Engineering. Wiesbaden & Heidelberg: Springer Vieweg.

Hevner, Alan R. et al. (2004). "Design Science in Information Systems Research". *MIS Quarterly* 28, pp. 75–105.

IEEE (1990). Standard Glossary of Software Engineering Terminology. Corrected ed. New York: Institute of Electrical and Electronics Engineers.

Jennings, Nicholas R. et al. (1998). "A Roadmap of Agent Research and Development". *Autonomous Agents and Multi-Agent Systems* 1, pp. 7–38.

Juran, Joseph Moses and A. Blanton Godfrey (1999). Juran's quality handbook. 5th ed. McGraw Hill.

Mayring, Philipp and Thomas Fenzl (2022). “Qualitative Inhaltsanalyse”. Handbuch Methoden der empirischen Sozialforschung. Springer Fachmedien Wiesbaden, pp. 691–706.

Owoade, Samuel et al. (2024). “Systematic Review of Strategic Business Administration Practices for Driving Operational Excellence in IT-Driven Firms”. *International Journal of Scientific Research in Science and Technology* 11, pp. 680–700.

Peffers, Ken et al. (2007). “A design science research methodology for information systems research”. *Journal of Management Information Systems* 24, pp. 45–77.

Qian, Chen et al. (2024). ChatDev: Communicative Agents for Software Development.

Russell, Stuart et al. (2015). “Research Priorities for Robust and Beneficial Artificial Intelligence”. Version 1.

Sapkota, Ranjan et al. (2025). “AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges”. *Information Fusion* 126, pp. 103–599.

Shu, Raphael et al. (2024). Towards Effective GenAI Multi-Agent Collaboration: Design and Evaluation for Enterprise Applications.

Stohr, Edward A. and J. Leon Zhao (2001). “Workflow Automation: Overview and Research Issues”. *Information Systems Frontiers* 3, pp. 281–296.

Womack, J P and D T Jones (1997). “Lean Thinking—Banish Waste and Create Wealth in your Corporation”. *Journal of the Operational Research Society* 48.11, pp. 1148–1148.

Yang, Hui et al. (2023). Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions.

Yao, Shunyu et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models.

Appendix

Requirements List

Organized by Requirement Type

FUNCTIONAL REQUIREMENTS

- FR-01 PROCESS ORCHESTRATION ENGINE — The system shall execute workflow models by dispatching tasks to human or software actors according to model control flow and business rules.
- FR-02 PARAMETERIZED SUBPROCESSES — The system shall support parameterized subprocesses and rules so behavioral variants can be expressed without altering the underlying process structure.
- FR-03 HUMAN-MACHINE TASK ORCHESTRATION — The system shall coordinate both human and automated tasks within the same orchestrated process model.
- FR-04 EXCEPTION DETECTION AND ROUTING — The system shall detect deviations from the nominal process and route cases to defined exception subprocesses.
- FR-05 ESCALATION TO HUMAN AUTHORITY — The system shall escalate unresolved or unmodeled exceptions to designated human roles with clear ownership.
- FR-06 INTER-AGENT COMMUNICATION PROTOCOL — The system shall specify message formats and interaction rules for agent collaboration (e.g., direct messaging or shared memory; event-bus/blackboard where asynchronous exchange is required).
- FR-07 TOOL FAILURE HANDLING IN EXCEPTIONS — The system shall validate tool/API outputs and trigger defined recovery or fallback steps when invocations fail.
- FR-08 CONFLICT RESOLUTION — The system shall provide mechanisms for agents to resolve task or decision conflicts (e.g., delegation to a planner, negotiation, or voting).
- FR-09 POLICY ENGINE — The system shall provide a decoupled policy evaluation component that can validate, veto, or redirect workflow executions and agent actions at runtime.
- FR-10 RISK-BASED APPROVALS AND ESCALATION — The system shall require human approval and/or escalation for actions or decisions exceeding defined risk or impact thresholds.
- FR-11 COMPLIANCE MAPPING AND DRIFT CHECKS — The system shall map actions and policies to applicable regulations or internal rules and perform runtime checks to detect and block compliance drift.
- FR-12 EVIDENCE-BASED DECISION SUPPORT — The system shall aggregate relevant data at decision points to support evidence-based choices and reduce errors.
- FR-13 RULE-ENFORCED DECISION POINTS — The system shall enforce business rules and role responsibilities at decision points to ensure consistent, auditable outcomes.
- FR-14 DECISION TRACE AND RATIONALE — The system shall persist a human-readable decision trace for automated or assisted decisions, including inputs, tool/API calls and

results, and concise rationale summaries.

- FR-15 EVENT LOGGING PIPELINE — The system shall instrument the workflow engine and agents to emit structured, timestamped events for key actions (e.g., prompts, tool/API invocations and results, plan/decision commits, and state changes).
- FR-16 DASHBOARDS AND ALERTS — The system shall provide real-time dashboards and alerting for observability metrics (e.g., execution latency, error/anomaly rates, blocked actions).
- FR-17 REPLAY FOR POST-HOC ANALYSIS — The system shall support reconstruction and replay of workflow and agent interactions from logged events to enable root-cause analysis and explanation of outcomes.
- FR-18 INTEGRATION CONNECTORS — The system shall provide connectors to integrate heterogeneous applications and data sources required by the workflows.
- FR-19 AGENT TOOL ADAPTERS — The system shall expose a uniform adapter interface for agents and workflows to invoke external tools, APIs, databases, or RPA scripts.
- FR-20 INTER-ORGANIZATIONAL INTEROPERABILITY — The system shall support protocol and interface interoperability suitable for cross-organizational workflows.
- FR-21 DATA TRANSFORMATION LAYER — The system shall provide mapping and transformation capabilities to reconcile data across integrated systems.

CONSTRAINTS

- C-01 SEGREGATION OF DUTIES — The system shall enforce role-based access control and segregation of duties for configuration changes and sensitive actions.
- C-02 AUDIT LOGGING AND RETENTION — The system shall produce tamper-evident audit trails of agent actions, policy decisions, and configuration changes, retained according to the applicable compliance policy.
- C-03 BOUNDED DECISION AUTONOMY — The system shall constrain agent decision-making to clearly scoped authority levels aligned with organizational objectives.
- C-04 MODULAR PROCESS UNITS — The system shall structure workflows as modular subprocesses or services that can be reused and reconfigured without redesign.
- C-05 INTERFACE-BASED COMPOSITION — The system shall expose clear process and service interfaces compatible with established interoperability standards to enable composition across heterogeneous systems and organizations.
- C-06 EXPLICIT COORDINATION MODEL — The system shall define and enforce a coordination structure (e.g., hierarchical planner-specialists or decentralized collaboration) for multi-agent work.
- C-07 AGENT STATUS SELF-REPORTING — The system shall require agents to periodically self-report status and progress (e.g., current task, step outcome, next planned action) to improve runtime transparency.
- C-08 INTERFACE CONTRACTS AND SCHEMAS — The system shall define input/output contracts and validate request/response schemas at adapter boundaries.
- C-09 INVOCATION SAFEGUARDS — The system shall enforce adapter-level safeguards (e.g., timeouts, retries, and idempotency keys) to limit side effects of failed or repeated tool calls.

Organized by Requirement Cluster

GOVERNANCE AND COMPLIANCE

- FR-09 POLICY ENGINE — The system shall provide a decoupled policy evaluation component that can validate, veto, or redirect workflow executions and agent actions at runtime.
- FR-10 RISK-BASED APPROVALS AND ESCALATION — The system shall require human approval and/or escalation for actions or decisions exceeding defined risk or impact thresholds.
- FR-11 COMPLIANCE MAPPING AND DRIFT CHECKS — The system shall map actions and policies to applicable regulations or internal rules and perform runtime checks to detect and block compliance drift.
- C-01 SEGREGATION OF DUTIES — The system shall enforce role-based access control and segregation of duties for configuration changes and sensitive actions.
- C-02 AUDIT LOGGING AND RETENTION — The system shall produce tamper-evident audit trails of agent actions, policy decisions, and configuration changes, retained according to the applicable compliance policy.

DECISION QUALITY

- FR-12 EVIDENCE-BASED DECISION SUPPORT — The system shall aggregate relevant data at decision points to support evidence-based choices and reduce errors.
- FR-13 RULE-ENFORCED DECISION POINTS — The system shall enforce business rules and role responsibilities at decision points to ensure consistent, auditable outcomes.
- C-03 BOUNDED DECISION AUTONOMY — The system shall constrain agent decision-making to clearly scoped authority levels aligned with organizational objectives.

ORCHESTRATION AND MODULARITY

- FR-01 PROCESS ORCHESTRATION ENGINE — The system shall execute workflow models by dispatching tasks to human or software actors according to model control flow and business rules.
- FR-02 PARAMETERIZED SUBPROCESSES — The system shall support parameterized subprocesses and rules so behavioral variants can be expressed without altering the underlying process structure.
- FR-03 HUMAN-MACHINE TASK ORCHESTRATION — The system shall coordinate both human and automated tasks within the same orchestrated process model.
- C-04 MODULAR PROCESS UNITS — The system shall structure workflows as modular subprocesses or services that can be reused and reconfigured without redesign.
- C-05 INTERFACE-BASED COMPOSITION — The system shall expose clear process and service interfaces compatible with established interoperability standards to enable composition across heterogeneous systems and organizations.

EXCEPTION HANDLING AND COORDINATION

- FR-04 EXCEPTION DETECTION AND ROUTING — The system shall detect deviations from the nominal process and route cases to defined exception subprocesses.

- FR-05 ESCALATION TO HUMAN AUTHORITY — The system shall escalate unresolved or unmodeled exceptions to designated human roles with clear ownership.
- FR-06 INTER-AGENT COMMUNICATION PROTOCOL — The system shall specify message formats and interaction rules for agent collaboration (e.g., direct messaging or shared memory; event-bus/blackboard where asynchronous exchange is required).
- FR-07 TOOL FAILURE HANDLING IN EXCEPTIONS — The system shall validate tool/API outputs and trigger defined recovery or fallback steps when invocations fail.
- FR-08 CONFLICT RESOLUTION — The system shall provide mechanisms for agents to resolve task or decision conflicts (e.g., delegation to a planner, negotiation, or voting).
- C-06 EXPLICIT COORDINATION MODEL — The system shall define and enforce a coordination structure (e.g., hierarchical planner-specialists or decentralized collaboration) for multi-agent work.

OBSERVABILITY AND TRACEABILITY

- FR-14 DECISION TRACE AND RATIONALE — The system shall persist a human-readable decision trace for automated or assisted decisions, including inputs, tool/API calls and results, and concise rationale summaries.
- FR-15 EVENT LOGGING PIPELINE — The system shall instrument the workflow engine and agents to emit structured, timestamped events for key actions (e.g., prompts, tool/API invocations and results, plan/decision commits, and state changes).
- FR-16 DASHBOARDS AND ALERTS — The system shall provide real-time dashboards and alerting for observability metrics (e.g., execution latency, error/anomaly rates, blocked actions).
- FR-17 REPLAY FOR POST-HOC ANALYSIS — The system shall support reconstruction and replay of workflow and agent interactions from logged events to enable root-cause analysis and explanation of outcomes.
- C-07 AGENT STATUS SELF-REPORTING — The system shall require agents to periodically self-report status and progress (e.g., current task, step outcome, next planned action) to improve runtime transparency.

TOOL INTEGRATION

- FR-18 INTEGRATION CONNECTORS — The system shall provide connectors to integrate heterogeneous applications and data sources required by the workflows.
- FR-19 AGENT TOOL ADAPTERS — The system shall expose a uniform adapter interface for agents and workflows to invoke external tools, APIs, databases, or RPA scripts.
- FR-20 INTER-ORGANIZATIONAL INTEROPERABILITY — The system shall support protocol and interface interoperability suitable for cross-organizational workflows.
- FR-21 DATA TRANSFORMATION LAYER — The system shall provide mapping and transformation capabilities to reconcile data across integrated systems.
- C-08 INTERFACE CONTRACTS AND SCHEMAS — The system shall define input/output contracts and validate request/response schemas at adapter boundaries.
- C-09 INVOCATION SAFEGUARDS — The system shall enforce adapter-level safeguards (e.g., timeouts, retries, and idempotency keys) to limit side effects of failed or repeated tool calls.

A.2 Requirements SysML v2 Model

```
package MASRequirements {
  private import Requirements::*;

  // --- Multi-Agent System ---
  part def MAS;

  // ================================================
  // Governance & Compliance
  // ================================================
  package GovernanceAndCompliance {

    requirement def FR_09_PolicyEngine {
      attribute id = "FR-09";
      attribute title = "Policy engine";
      attribute intent = "Functional";
      attribute cluster = "GovernanceAndCompliance";
      attribute source = "Consolidation 4.2 / Appendix A.1";
      attribute plannedVerification = "Analysis"; // to be refined in 5
      subject soi : MAS;
      doc /* The system shall provide a decoupled policy evaluation component
          that can validate, veto, or redirect workflow executions and agent
          actions at runtime. */
    }

    requirement def FR_10_RiskBasedApprovalsAndEscalation {
      attribute id = "FR-10";
      attribute title = "Risk-based approvals & escalation";
      attribute intent = "Functional";
      attribute cluster = "GovernanceAndCompliance";
      attribute source = "Consolidation 4.2 / Appendix A.1";
      attribute plannedVerification = "Demonstration";
      subject soi : MAS;
      doc /* The system shall require human approval and/or escalation for
          actions or decisions exceeding defined risk or impact thresholds. */
    }

```

```
34    requirement def FR_11_ComplianceMappingAndDriftChecks {
35      attribute id = "FR-11";
36      attribute title = "Compliance mapping & drift checks";
37      attribute intent = "Functional";
38      attribute cluster = "GovernanceAndCompliance";
39      attribute source = "Consolidation 4.2 / Appendix A.1";
40      attribute plannedVerification = "Analysis";
41      subject soi : MAS;
42      doc /* The system shall map actions and policies to applicable
            regulations or internal rules and perform runtime checks to detect
            and block compliance drift. */
43    }
44
45    requirement def C_01_SegregationOfDuties {
46      attribute id = "C-01";
47      attribute title = "Segregation of duties";
48      attribute intent = "Constraint";
49      attribute cluster = "GovernanceAndCompliance";
50      attribute source = "Consolidation 4.2 / Appendix A.1";
51      attribute plannedVerification = "Inspection";
52      subject soi : MAS;
53      doc /* Enforce role-based access control and segregation of duties for
            configuration changes and sensitive actions. */
54    }
55
56    requirement def C_02_AuditLoggingAndRetention {
57      attribute id = "C-02";
58      attribute title = "Audit logging & retention";
59      attribute intent = "Constraint";
60      attribute cluster = "GovernanceAndCompliance";
61      attribute source = "Consolidation 4.2 / Appendix A.1";
62      attribute plannedVerification = "Inspection";
63      subject soi : MAS;
64      doc /* Produce tamper-evident audit trails of agent actions, policy
            decisions, and configuration changes, retained according to the
            applicable compliance policy. */
65    }
```

```
}

// ================================================
// Decision Quality
// ================================================
package DecisionQuality {

  requirement def FR_12_EvidenceBasedDecisionSupport {
    attribute id = "FR-12";
    attribute title = "Evidence-based decision support";
    attribute intent = "Functional";
    attribute cluster = "DecisionQuality";
    attribute source = "Consolidation 4.2 / Appendix A.1";
    attribute plannedVerification = "Demonstration";
    subject soi : MAS;
    doc /* The system shall aggregate relevant data at decision points to
        support evidence-based choices and reduce errors. */
  }

  requirement def FR_13_RuleEnforcedDecisionPoints {
    attribute id = "FR-13";
    attribute title = "Rule-enforced decision points";
    attribute intent = "Functional";
    attribute cluster = "DecisionQuality";
    attribute source = "Consolidation 4.2 / Appendix A.1";
    attribute plannedVerification = "Test";
    subject soi : MAS;
    doc /* The system shall enforce business rules and role responsibilities
         at decision points to ensure consistent, auditable outcomes. */
  }

  requirement def C_03_BoundedDecisionAutonomy {
    attribute id = "C-03";
    attribute title = "Bounded decision autonomy";
    attribute intent = "Constraint";
    attribute cluster = "DecisionQuality";
    attribute source = "Consolidation 4.2 / Appendix A.1";
```

```
    attribute plannedVerification = "Analysis";
    subject soi : MAS;
    doc /* Constrain agent decision-making to clearly scoped authority
        levels aligned with organizational objectives. */
  }
}

// ================================================
// Orchestration & Modularity
// ================================================
package OrchestrationAndModularity {

  requirement def FR_01_ProcessOrchestrationEngine {
    attribute id = "FR-01";
    attribute title = "Process orchestration engine";
    attribute intent = "Functional";
    attribute cluster = "OrchestrationAndModularity";
    attribute source = "Consolidation 4.2 / Appendix A.1";
    attribute plannedVerification = "Demonstration";
    subject soi : MAS;
    doc /* The system shall execute workflow models by dispatching tasks to
        human or software actors according to model control flow and
        business rules. */
  }

  requirement def FR_02_ParameterizedSubprocesses {
    attribute id = "FR-02";
    attribute title = "Parameterized subprocesses";
    attribute intent = "Functional";
    attribute cluster = "OrchestrationAndModularity";
    attribute source = "Consolidation 4.2 / Appendix A.1";
    attribute plannedVerification = "Analysis";
    subject soi : MAS;
    doc /* The system shall support parameterized subprocesses and rules so
        behavioral variants can be expressed without altering the underlying
         process structure. */
  }
```

```
133
134   requirement def FR_03_HumanMachineTaskOrchestration {
135     attribute id = "FR-03";
136     attribute title = "Humanmachine task orchestration";
137     attribute intent = "Functional";
138     attribute cluster = "OrchestrationAndModularity";
139     attribute source = "Consolidation 4.2 / Appendix A.1";
140     attribute plannedVerification = "Demonstration";
141     subject soi : MAS;
142     doc /* The system shall coordinate both human and automated tasks within
            the same orchestrated process model. */
143   }
144
145   requirement def C_04_ModularProcessUnits {
146     attribute id = "C-04";
147     attribute title = "Modular process units";
148     attribute intent = "Constraint";
149     attribute cluster = "OrchestrationAndModularity";
150     attribute source = "Consolidation 4.2 / Appendix A.1";
151     attribute plannedVerification = "Inspection";
152     subject soi : MAS;
153     doc /* Structure workflows as modular subprocesses or services that can
           be reused and reconfigured without redesign. */
154   }
155
156   requirement def C_05_InterfaceBasedComposition {
157     attribute id = "C-05";
158     attribute title = "Interface-based composition";
159     attribute intent = "Constraint";
160     attribute cluster = "OrchestrationAndModularity";
161     attribute source = "Consolidation 4.2 / Appendix A.1";
162     attribute plannedVerification = "Analysis";
163     subject soi : MAS;
164     doc /* Expose clear process and service interfaces compatible with
           established interoperability standards to enable composition across
           heterogeneous systems and organizations. */
165   }
```

```
}

// ================================================
// Exception Handling & Coordination
// ================================================
package ExceptionHandlingAndCoordination {

  requirement def FR_04_ExceptionDetectionAndRouting {
    attribute id = "FR-04";
    attribute title = "Exception detection & routing";
    attribute intent = "Functional";
    attribute cluster = "ExceptionHandlingAndCoordination";
    attribute source = "Consolidation 4.2 / Appendix A.1";
    attribute plannedVerification = "Test";
    subject soi : MAS;
    doc /* The system shall detect deviations from the nominal process and
        route cases to defined exception subprocesses. */
  }

  requirement def FR_05_EscalationToHumanAuthority {
    attribute id = "FR-05";
    attribute title = "Escalation to human authority";
    attribute intent = "Functional";
    attribute cluster = "ExceptionHandlingAndCoordination";
    attribute source = "Consolidation 4.2 / Appendix A.1";
    attribute plannedVerification = "Demonstration";
    subject soi : MAS;
    doc /* The system shall escalate unresolved or unmodeled exceptions to
        designated human roles with clear ownership. */
  }

  requirement def FR_06_InterAgentCommunicationProtocol {
    attribute id = "FR-06";
    attribute title = "Inter-agent communication protocol";
    attribute intent = "Functional";
    attribute cluster = "ExceptionHandlingAndCoordination";
    attribute source = "Consolidation 4.2 / Appendix A.1";
```

```
201       attribute plannedVerification = "Test";
202       subject soi : MAS;
203       doc /* The system shall specify message formats and interaction rules
              for agent collaboration (e.g., direct messaging or shared memory;
              event-bus/blackboard where asynchronous exchange is required). */
204     }
205
206     requirement def FR_07_ToolFailureHandlingInExceptions {
207       attribute id = "FR-07";
208       attribute title = "Tool failure handling in exceptions";
209       attribute intent = "Functional";
210       attribute cluster = "ExceptionHandlingAndCoordination";
211       attribute source = "Consolidation 4.2 / Appendix A.1";
212       attribute plannedVerification = "Test";
213       subject soi : MAS;
214       doc /* The system shall validate tool/API outputs and trigger defined
              recovery or fallback steps when invocations fail. */
215     }
216
217     requirement def FR_08_ConflictResolution {
218       attribute id = "FR-08";
219       attribute title = "Conflict resolution";
220       attribute intent = "Functional";
221       attribute cluster = "ExceptionHandlingAndCoordination";
222       attribute source = "Consolidation 4.2 / Appendix A.1";
223       attribute plannedVerification = "Demonstration";
224       subject soi : MAS;
225       doc /* The system shall provide mechanisms for agents to resolve task or
               decision conflicts (e.g., delegation to a planner, negotiation, or
              voting). */
226     }
227
228     requirement def C_06_ExplicitCoordinationModel {
229       attribute id = "C-06";
230       attribute title = "Explicit coordination model";
231       attribute intent = "Constraint";
232       attribute cluster = "ExceptionHandlingAndCoordination";
```

```
233       attribute source = "Consolidation 4.2 / Appendix A.1";
234       attribute plannedVerification = "Analysis";
235       subject soi : MAS;
236       doc /* Define and enforce a coordination structure (e.g., hierarchical
              plannerspecialists or decentralized collaboration) for multi-agent
              work. */
237     }
238   }
239
240   // ================================================
241   // Observability & Traceability
242   // ================================================
243   package ObservabilityAndTraceability {
244
245     requirement def FR_14_DecisionTraceAndRationale {
246       attribute id = "FR-14";
247       attribute title = "Decision trace & rationale";
248       attribute intent = "Functional";
249       attribute cluster = "ObservabilityAndTraceability";
250       attribute source = "Consolidation 4.2 / Appendix A.1";
251       attribute plannedVerification = "Inspection";
252       subject soi : MAS;
253       doc /* The system shall persist a human-readable decision trace for
              automated or assisted decisions, including inputs, tool/API calls
              and results, and concise rationale summaries. */
254     }
255
256     requirement def FR_15_EventLoggingPipeline {
257       attribute id = "FR-15";
258       attribute title = "Event logging pipeline";
259       attribute intent = "Functional";
260       attribute cluster = "ObservabilityAndTraceability";
261       attribute source = "Consolidation 4.2 / Appendix A.1";
262       attribute plannedVerification = "Inspection";
263       subject soi : MAS;
264       doc /* The system shall instrument the workflow engine and agents to
              emit structured, timestamped events for key actions (e.g., prompts,
```

```
        tool/API invocations and results, plan/decision commits, and state
        changes). */
265   }
266
267   requirement def FR_16_DashboardsAndAlerts {
268     attribute id = "FR-16";
269     attribute title = "Dashboards & alerts";
270     attribute intent = "Functional";
271     attribute cluster = "ObservabilityAndTraceability";
272     attribute source = "Consolidation 4.2 / Appendix A.1";
273     attribute plannedVerification = "Demonstration";
274     subject soi : MAS;
275     doc /* The system shall provide real-time dashboards and alerting for
            observability metrics (e.g., execution latency, error/anomaly rates,
             blocked actions). */
276   }
277
278   requirement def FR_17_ReplayForPostHocAnalysis {
279     attribute id = "FR-17";
280     attribute title = "Replay for post-hoc analysis";
281     attribute intent = "Functional";
282     attribute cluster = "ObservabilityAndTraceability";
283     attribute source = "Consolidation 4.2 / Appendix A.1";
284     attribute plannedVerification = "Demonstration";
285     subject soi : MAS;
286     doc /* The system shall support reconstruction and replay of workflow
            and agent interactions from logged events to enable root-cause
            analysis and explanation of outcomes. */
287   }
288
289   requirement def C_07_AgentStatusSelfReporting {
290     attribute id = "C-07";
291     attribute title = "Agent status self-reporting";
292     attribute intent = "Constraint";
293     attribute cluster = "ObservabilityAndTraceability";
294     attribute source = "Consolidation 4.2 / Appendix A.1";
295     attribute plannedVerification = "Test";
```

```
296       subject soi : MAS;
297       doc /* Require agents to periodically self-report status and progress (e.
              g., current task, step outcome, next planned action). */
298     }
299   }
300
301   // ================================================
302   // Tool Integration
303   // ================================================
304   package ToolIntegration {
305
306     requirement def FR_18_IntegrationConnectors {
307       attribute id = "FR-18";
308       attribute title = "Integration connectors";
309       attribute intent = "Functional";
310       attribute cluster = "ToolIntegration";
311       attribute source = "Consolidation 4.2 / Appendix A.1";
312       attribute plannedVerification = "Demonstration";
313       subject soi : MAS;
314       doc /* The system shall provide connectors to integrate heterogeneous
              applications and data sources required by the workflows. */
315     }
316
317     requirement def FR_19_AgentToolAdapters {
318       attribute id = "FR-19";
319       attribute title = "Agent tool adapters";
320       attribute intent = "Functional";
321       attribute cluster = "ToolIntegration";
322       attribute source = "Consolidation 4.2 / Appendix A.1";
323       attribute plannedVerification = "Test";
324       subject soi : MAS;
325       doc /* The system shall expose a uniform adapter interface for agents
              and workflows to invoke external tools, APIs, databases, or RPA
              scripts. */
326     }
327
328     requirement def FR_20_InterOrganizationalInteroperability {
```

```
    attribute id = "FR-20";
    attribute title = "Inter-organizational interoperability";
    attribute intent = "Functional";
    attribute cluster = "ToolIntegration";
    attribute source = "Consolidation 4.2 / Appendix A.1";
    attribute plannedVerification = "Analysis";
    subject soi : MAS;
    doc /* The system shall support protocol and interface interoperability
        suitable for cross-organizational workflows. */
  }

  requirement def FR_21_DataTransformationLayer {
    attribute id = "FR-21";
    attribute title = "Data transformation layer";
    attribute intent = "Functional";
    attribute cluster = "ToolIntegration";
    attribute source = "Consolidation 4.2 / Appendix A.1";
    attribute plannedVerification = "Test";
    subject soi : MAS;
    doc /* The system shall provide mapping and transformation capabilities
        to reconcile data across integrated systems. */
  }

  requirement def C_08_InterfaceContractsAndSchemas {
    attribute id = "C-08";
    attribute title = "Interface contracts & schemas";
    attribute intent = "Constraint";
    attribute cluster = "ToolIntegration";
    attribute source = "Consolidation 4.2 / Appendix A.1";
    attribute plannedVerification = "Inspection";
    subject soi : MAS;
    doc /* Define input/output contracts and validate request/response
        schemas at adapter boundaries. */
  }

  requirement def C_09_InvocationSafeguards {
    attribute id = "C-09";
```

```
363       attribute title = "Invocation safeguards";
364       attribute intent = "Constraint";
365       attribute cluster = "ToolIntegration";
366       attribute source = "Consolidation 4.2 / Appendix A.1";
367       attribute plannedVerification = "Test";
368       subject soi : MAS;
369       doc /* Enforce adapter-level safeguards (e.g., timeouts, retries,
              idempotency keys). */
370     }
371   }
372 }
```