systems

MDPI

Article

An MBSE-Driven Digital Twin Framework with Semantic Enhancement for Cross-Phase Collaborative Management in Complex Product Systems

Zhu Xiang †, Minghao Li †, Tianyang Lei, Kewei Yang *, Guopeng Song and Jiang Jiang

College of Systems Engineering, National University of Defense Technology, Changsha 410073, China; xiangzhu_xz@163.com (Z.X.); liminghao4869@nudt.edu.cn (M.L.); leitianyang20@163.com (T.L.); guopeng.song@nudt.edu.cn (G.S.); jiangjiangnudt@nudt.edu.cn (J.J.)

* Correspondence: kayyang27@nudt.edu.cn

† These authors contributed equally to this work.

Highlights

Please indicate how your work links to systems science via your contributions to systems practice, theory, and/or methodology.

- An MBSE-driven digital twin framework combines a four-layer architecture and a collaboration-oriented SysML Profile to support structured and traceable cross-phase management.
- Semantic enhancement enables runtime state synchronization, cross-phase impact reasoning, and adaptive decision support under development disturbances.

What are the main findings and/or the implications of the main findings?

- Demonstrates a feasible engineering pathway from perceptive digital twins toward reasoning-enabled collaborative decision support while retaining human involvement in management decisions.
- Shows that semantic reasoning can propagate disturbance impacts across development phases and translate them into decision-ready constraints, supporting adaptive cross-phase collaborative management.

Abstract

Cross-phase collaborative management in complex product systems (CoPS) development is inherently challenged by heterogeneous organizational coupling and stochastic disturbances. Although digital twin (DT) and model-based systems engineering (MBSE) technologies provide foundations for physical–virtual synchronization and model traceability, existing approaches remain fragmented in three respects: insufficient requirements-traceable architectural integration, limited cross-phase semantic interoperability and runtime evolution, and weak operational links between semantic reasoning and adaptive decision models. To address these gaps, this paper proposes an MBSE-driven digital twin framework with semantic enhancement. First, a four-layer architecture is derived using the MagicGrid methodology, encompassing physical–virtual mapping, semantic reasoning, decision support, and service interaction. Second, a collaboration-oriented SysML profile is developed to standardize the representation of tasks, resources, materials, disturbances, and management constraints across engineering phases. Third, a knowledge-driven adaptive collaboration mechanism maps runtime disturbance inputs into semantic states, propagates their cross-phase impacts, supports process-topology reconfiguration, and generates decision-ready constraints for adaptive management. A case-based prototype for aero-engine turbofan blade development demonstrates the feasibility of the mapping–reasoning–decision chain and provides case-level evidence of improved cross-phase coordination under controlled

Check for updates

Academic Editor: James Cale

Received: 4 March 2026
Revised: 9 August 2026
Accepted: 18 August 2026
Published: 24 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

disturbance scenarios. The results indicate a feasible engineering pathway from perceptive DT functions toward reasoning-enabled collaborative decision support.

Keywords: collaborative management; digital twin; model-based systems engineering (MBSE); knowledge-driven; systems modeling language (SysML)

1. Introduction

Complex product systems (CoPSs), exemplified by aerospace engines and heavy-duty gas turbines, represent capital-intensive assets where development management efficiency attracts critical attention [1]. Unlike mass production, the development of these systems necessitates intricate collaboration among heterogeneous organizations across design, manufacturing, and testing phases. However, this coupled process is inherently subject to stochastic disturbances, including design changes, test failures, and resource shortages, which often trigger cascading delays throughout the entire value chain [2]. Traditional management systems lack the capability to autonomously perceive real-time physical disturbances or dynamically reconfigure cross-phase resources, functioning primarily in an offline and open-loop mode [3].

Digital twin (DT) technology has emerged as a transformative paradigm for achieving cross-phase collaboration and whole-process management [4]. By establishing a holistic, dynamic virtual mapping of physical entities, this technology lays the technical foundation for real-time perception, virtual-physical interaction, and closed-loop intelligent decision-making [5]. Nevertheless, the majority of studies focus on isolated technical issues such as equipment-level monitoring [6] or specific process simulations [7], lacking a layered and modular design for the overall system architecture [8]. This structural deficiency prevents integrating isolated modules—from perception to decision support—into a coherent closed loop, leaving the system unable to adapt autonomously to real-time physical disturbances. For example, a blade design revision may be recorded in the design environment as an updated CAD model, while downstream manufacturing and testing systems require explicit information about affected activities, precedence relations, resource states, and execution conditions. Without a common cross-phase representation and runtime propagation mechanism, the implications of the same change may be interpreted inconsistently and may not be translated into updated scheduling constraints.

To address this structural fragmentation, model-based systems engineering (MBSE) provides a rigorous foundation for constructing standardized system architectures [9]. MBSE formally represents system requirements, functions, components, interfaces, and their dependencies within an interconnected model, thereby providing a common architectural reference for defining module responsibilities, data exchanges, and interface contracts. This formal integration enables heterogeneous DT modules to be developed against a consistent system definition and reduces module isolation [10]. Moreover, lifecycle traceability links stakeholder requirements with architectural elements, implementation artifacts, and verification evidence, allowing design-time changes to be traced across model artifacts and inconsistencies to be identified during system development and evolution [11].

However, MBSE alone primarily provides architectural coherence and design-time traceability, while conventional MBSE models remain phase-independent and largely static during execution. The application to DT-enabled cross-phase collaborative management still faces fundamental disconnects in both spatial and temporal dimensions. The preceding scenario reveals these two disconnects. In this study, spatial semantic heterogeneity refers to inconsistent representations and interpretations of cross-phase collaborative information,

including tasks, resources, materials, disturbances, and constraints, across organizational and lifecycle boundaries [12]. It concerns whether heterogeneous participants can interpret the collaborative implications of the same engineering change consistently. The temporal semantic gap refers to the inability of static architectural models to remain synchronized with runtime changes and to use updated model states for decision support. It concerns whether an identified disturbance can update affected task and resource states, propagate through cross-phase dependencies, and be converted into constraints that a scheduling model can process. Existing studies have improved lifecycle information continuity [12] and runtime adaptation under unexpected conditions [13]. However, the integration of these capabilities into a cross-phase mechanism that propagates runtime impacts and generates decision-ready constraints remains insufficiently developed.

Consequently, this work proposes a digital twin framework with semantic enhancement for cross-phase collaborative management in the development of CoPS. Grounded in the MagicGrid methodology of MBSE, the framework follows a systematic top-down process to guide the transition from the problem domain to the solution domain. First, a standardized four-layer architecture is constructed as the core solution to meet the identified collaborative requirements, encompassing physical-virtual mapping, semantic reasoning, decision support, and service interaction. On this architectural foundation, two complementary mechanisms are introduced. A collaboration-oriented SysML profile standardizes the representation of cross-phase collaborative information, while a knowledge-driven adaptive collaboration mechanism synchronizes runtime states, propagates disturbance impacts, and transforms reasoning results into executable inputs for the decision support layer. The primary contributions of this work are summarized as follows:

- A standardized DT architecture: Based on the MagicGrid methodology of MBSE, we proceed from the problem domain to design a four-layer architecture within the solution domain. This architecture integrates physical-virtual mapping, semantic reasoning, decision support, and service interaction to overcome structural fragmentation.
- A collaboration-oriented SysML profile: To mitigate collaboration-level spatial semantic heterogeneity, we develop a cross-phase extension of SysML that standardizes the representation of tasks, resources, materials, disturbance events, and decision constraints across participating organizations.
- A knowledge-driven adaptive collaboration mechanism: To bridge the runtime model-decision disconnect, we integrate ontologies, reasoning rules, and decision strategies to synchronize dynamic states, infer cascading impacts, and generate quantifiable logic constraints for closed-loop adaptation.
- System instantiation and verification: We instantiated the proposed framework for an aero-engine turbofan blade development scenario to validate its feasibility, demonstrating the capability to operationalize closed-loop adaptation and effectively mitigate cross-phase uncertainties.

2. Literature Review

This section reviews the theoretical foundations of DT, MBSE, and ontology-driven reasoning, concluding with a gap analysis that motivates the proposed framework.

2.1. Digital Twin for Product Development

The DT concept, tracing back to Grieves's proposal in 2002 [14] and formally defined by ISO 23247 [15], characterizes physical entities and processes through virtual–physical synchronization [16]. By integrating heterogeneous data and models, DT provides a technical foundation for lifecycle collaboration, state monitoring, and closed-loop decision support [17,18].

Existing studies have investigated DT applications across different lifecycle phases. In the design phase, DT-driven product design frameworks have been developed to support iterative design analysis and engineering decision-making [19]. Integrated DT frameworks have also been proposed to connect complex-product design with manufacturing activities [20]. In the manufacturing phase, DT workshop architectures provide a foundation for virtual–physical integration and shop-floor management [21]. DT-enabled job-shop scheduling and adaptive workflow management have been used to respond to production disturbances and operation reworking [22,23]. Other studies have explored DT composition and operational-strategy adjustment under unexpected manufacturing conditions [13,24]. DT-based anomaly detection and predictive monitoring have also been investigated for manufacturing equipment and processes [25,26]. In the testing phase, DT-based virtual testing has been applied to autonomous vehicles [27], while DT-assisted state reconstruction methods have supported the monitoring of complex testing processes [28]. Collectively, these studies demonstrate the value of DT within individual lifecycle phases or bounded operational contexts. However, cross-phase collaboration requires information, model relationships, and disturbance impacts to remain interpretable across design, manufacturing, and testing boundaries.

To support such connectivity, recent research has explored standardized lifecycle infrastructures, particularly the Asset Administration Shell (AAS) and Digital Thread. AAS represents industrial assets through standardized metamodels, submodels, semantic identifiers, and service interfaces, thereby facilitating information exchange and interoperability among heterogeneous systems and lifecycle applications [29,30]. Digital Thread approaches establish persistent links among authoritative data, engineering models, and change records across lifecycle stages, supporting information continuity, traceability, and change management [31,32]. Recent studies have further connected AAS-based semantic DTs with scheduling modules, demonstrating that standardized asset representations can serve as an integration backbone for dynamic production applications [33].

Taken together, AAS and Digital Thread provide important foundations for cross-phase DT integration through standardized representation, information exchange, and lifecycle traceability. These infrastructures can support downstream reasoning and scheduling applications, but they do not by themselves specify the collaboration logic required to interpret runtime disturbances across design, manufacturing, and testing. It therefore remains insufficiently addressed how detected changes should update cross-phase task and resource states, propagate through lifecycle dependencies, and be operationalized as inputs for adaptive decision-making.

2.2. MBSE and Its Application for Digital Twin

MBSE is the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities from conceptual design through later lifecycle phases [34]. Compared with text-based systems engineering, which relies heavily on documents and may introduce ambiguities and traceability difficulties [35], MBSE organizes requirements, functions, structures, behaviors, and interfaces through interconnected models. This integrated representation improves model consistency and reduces ambiguity [36]. It also supports information traceability and model reuse during complex system development [37]. In practice, MBSE is implemented through the coordinated use of methodologies, modeling languages, and modeling tools to support the systematic construction of complex system architectures [38]. An MBSE methodology defines the principles, processes, and modeling steps used to address an engineering problem [39]. It generally guides system development from requirement elicitation and definition through architecture design and detailed design to verification, following a logic consistent with

the V-model [8]. MagicGrid provides one representative methodology for structuring this progression and maintaining relationships among requirements, architecture, and implementation models [40]. Within an MBSE methodology, SysML serves as a general-purpose systems modeling language for formally representing system requirements, structure, behavior, and parametric relationships [41]. Its standardized syntax and explicit model relationships help stakeholders maintain a consistent understanding of the system.

Existing MBSE–DT research can be organized around three main functions: lifecycle framing, systematic DT construction, and runtime model operation. Madni et al. framed DT as an overarching extension of MBSE and emphasized the role of formal models in maintaining lifecycle consistency and supporting simulation and analysis. However, their work primarily established a conceptual vision rather than an implemented mechanism for disturbance-driven process adaptation [42]. At the methodological level, MBSE has been used to organize DT requirements, functions, architectures, and implementation models [8], while standards-informed approaches have guided the systematic engineering of manufacturing DT systems [43]. These studies reduce model fragmentation and improve architectural consistency, but mainly focus on DT definition and construction. Other studies have extended MBSE–DT integration toward model heterogeneity, reconfiguration, verification, and runtime operation. Zhang et al. introduced heterogeneous model integration, variant management, and model reconfiguration for industrial robot DTs, thereby providing configuration-level dynamism [44]. Collaboration between system and multiphysics DT models has supported requirements verification and architecture trade-off [45]. Executable MBSE specifications have further enabled digital continuity, runtime simulation, system monitoring, and operational-state feedback [46], while bidirectional connections between SysML models and physical systems have supported runtime model updating and model-based testing [47].

Taken together, existing MBSE–DT studies have progressed beyond static architectural definition by supporting model configuration and reconfiguration [44], as well as executable models and physical–virtual synchronization [46,47]. However, these capabilities primarily concern the construction, execution, and updating of DT models; they do not by themselves specify how synchronized runtime information should be represented as collaboration-level states, interpreted against cross-phase dependencies, and operationalized for adaptive decision-making. Consequently, a unified and MBSE-traceable mechanism linking runtime state updates with disturbance interpretation, impact propagation across design–manufacturing–testing processes, and decision-ready constraint generation remains insufficiently developed.

2.3. Ontology-Driven Reasoning for Product Development Management

In DT-enabled complex product development, ontology-driven reasoning provides machine-interpretable representations of tasks, resources, materials, requirements, and runtime states, enabling engineering models and operational data to be connected through explicit semantic relationships and rules. One research stream operationalizes engineering models by translating semantic knowledge into executable or reconfigurable logic. Ontology reasoning has been used to map SysML designs to IEC 61499 control implementations for automatic control-code generation [48]. OWL-based representations have supported the dynamic reconfiguration of IoT control software [49], while SPARQL-based reasoning has enabled manufacturing-control reconfiguration under environmental disturbances [50]. Semantic descriptions of products, resources, skills, and environments have also been used to generate assembly plans and executable robot programs [51].

A second research stream applies semantic relationships and rules to dependency and impact reasoning. SysML-linked knowledge graphs have been used to identify fault-

propagation paths and reliability risks [52], while semantic rules have supported the analysis of requirement-change impacts on downstream architecture elements [53]. Semantic capability models have further matched changed manufacturing demands with resources, supply-chain alternatives, and business constraints, supporting coordinated responses to disruptions [54]. These studies extend semantic reasoning from knowledge representation toward runtime state assessment, impact identification, and response-option generation.

A third research stream couples semantic models with scheduling, optimization, and industrial decision support. An AAS- and ontology-based semantic DT has been connected with a simulation-based scheduler for dynamic production scheduling [33]. A knowledge-driven DT that uses semantic knowledge to drive geometric and decision models has been implemented in an aircraft manufacturing company [55]. Manufacturing knowledge graphs and graph-neural-network reasoning have also supported machining-process decisions [56], while NGSI-LD-based semantic models have been connected with agent-based optimization modules across machining, assembly, and inspection processes [57].

These studies confirm that semantic reasoning can support adaptive planning, scheduling, and industrial decision-making. However, the reviewed implementations are predominantly designed for specific production processes or bounded decision scenarios, and the mapping from semantic inferences to decision variables and constraints remains tied to the target application. A reusable and MBSE-traceable interface for propagating runtime impacts across design–manufacturing–testing dependencies and transforming them into executable decision inputs therefore remains insufficiently developed.

2.4. Knowledge Gaps and Motivations

Despite progress in DT, MBSE, and knowledge engineering, their capabilities remain fragmented when applied to cross-phase collaborative management. Table 1 compares the representative approaches reviewed above and summarizes the remaining requirements.

As summarized in Table 1, phase-specific DT applications have established monitoring, simulation, diagnosis, and adaptation capabilities within individual lifecycle stages, while AAS and Digital Thread approaches have improved standardized representation, information exchange, lifecycle connectivity, and traceability. MBSE-based DT studies have further supported systematic architecture definition, model integration, reconfiguration, execution, and physical–virtual synchronization. Ontology- and knowledge-graph-based methods enable semantic inference and impact analysis, and semantic scheduling approaches demonstrate that such knowledge can support optimization and decision-making. However, these capabilities are commonly distributed across separate solutions rather than integrated into a continuous cross-phase management mechanism.

Three connected gaps therefore remain. First, an architectural integration and traceability gap persists because physical–virtual mapping, semantic reasoning, decision support, and service functions are not generally organized within a common requirements-traceable system definition. Second, a cross-phase semantic interoperability and runtime evolution gap remains because standardized connectivity and model synchronization do not automatically ensure consistent representations of collaborative tasks, resources, materials, disturbances, dependencies, and evolving runtime states. Third, a reasoning-to-decision operationalization gap remains because inferred states and impacts often require process-specific or solver-specific mappings before they can be expressed as decision variables, boundaries, objectives, and constraints. The first two gaps make the third more difficult by depriving downstream decision modules of traceable architectural context and consistent cross-phase semantics. Nevertheless, resolving architectural and semantic fragmentation

alone is insufficient, because an explicit interface is still required to transform inferred impacts into executable decision inputs.

Table 1. Comparison of related approaches.

Approach	Main Contribution	Remaining Gap for Cross-Phase Collaboration	Required Advancement
Phase-specific DT applications [19,22,25]	Monitoring, simulation, diagnosis, and adaptation within individual lifecycle phases	Cross-phase dependencies and propagated disturbance impacts are typically not the primary focus	A requirements-traceable architecture for design–manufacturing–testing collaboration
AAS-based integration [29,30]	Standardized asset models, semantic submodels, identifiers, and interfaces	Standardized representation does not automatically support collaboration-specific reasoning and decision mapping	Collaboration-specific semantics and cross-phase dependency reasoning
Digital Thread approaches [31,32]	Lifecycle information continuity, model linkage, traceability, and change management	Lifecycle connectivity is not necessarily operationalized for runtime reasoning and adaptive decisions	Operationalization of lifecycle links for runtime impact propagation and decision support
Existing MBSE-based DT frameworks [8,42–44,47]	Architecture definition, model integration, variant management, traceability, and physical–virtual synchronization	Runtime functions mainly address model updating, verification, or reconfiguration within specific systems	Runtime semantic evolution and cross-phase impact reasoning within a traceable architecture
Ontology- and knowledge-graph-driven reasoning [50,52–54]	Semantic representation, rule-based inference, state assessment, and impact analysis	Inferred results often require application-specific mappings for use in decision models	A typed interface between semantic inference and decision models
Semantic scheduling and optimization [33,55–57]	Semantic support for disruption response, dynamic scheduling, optimization, and decision-making	Decision variables and constraints are often predefined or process-specific	Runtime derivation of decision inputs and constraints from propagated impacts

Accordingly, this study is motivated to develop an integrated approach that closes these gaps in a coordinated manner. Such an approach should provide a requirements-traceable architecture linking physical–virtual mapping, semantic reasoning, decision support, and service interaction. It should also establish consistent representations and runtime evolution mechanisms for cross-phase collaborative information, including tasks, resources, materials, disturbances, dependencies, and states. Finally, an explicit reasoning-to-decision interface is needed to transform inferred states and propagated impacts into executable decision inputs and constraints. These requirements provide the motivation and design basis for the framework introduced in the following section.

3. Proposed System Architecture and SysML Profile

3.1. The MagicGrid Methodology and V-Model Alignment

To ensure the rigorous development of the DT architecture, this study adopts the MagicGrid methodology. As depicted in the systems engineering V-model in Figure 1, the development of the system requires a symmetrical process of top-down decomposition

and bottom-up verification. MagicGrid operationalizes this philosophy by structuring the modeling process into three hierarchical layers: problem, solution, and implementation, as shown in Figure 2. By leveraging this formalized derivation, the proposed approach replaces ambiguous textual descriptions with standardized SysML models, effectively transforming the system architecture from a conceptual sketch into a computable blueprint. This alignment ensures complete traceability, guaranteeing that the instantiated system is strictly verified against the initial stakeholder requirements.

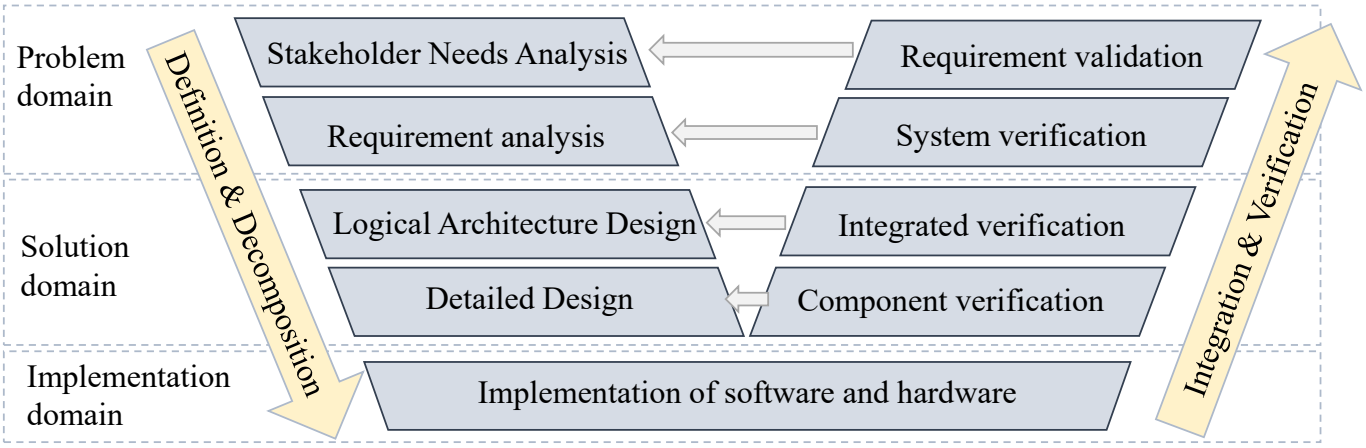


Figure 1. Methodological alignment between MagicGrid and the systems engineering V-model.

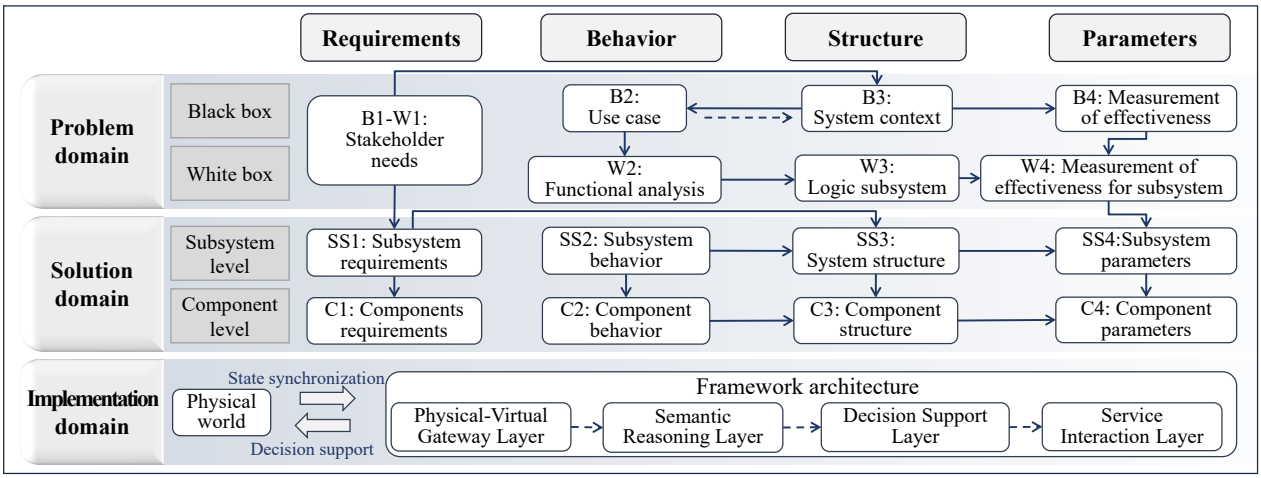


Figure 2. Adaptation of the MagicGrid methodology for the proposed framework.

3.2. Stakeholder Needs and Requirements

In the problem domain, the operational boundaries and performance goals are defined based on stakeholder needs. As the primary stakeholder, the overall project manager requires a system capable of managing cross-phase uncertainties and maintaining systemic adaptability. Figure 3 synthesizes the high-level stakeholder needs.

To operationalize these needs, the system context is modeled in Figure 4. The target collaborative system functions as a centralized hub, interacting with three heterogeneous physical phases: design, manufacturing, and testing. However, the integration of these disparate sources faces a critical challenge: the lack of a unified system structure to manage data flow from physical perception to closed-loop decision support. Consequently, a standardized layered architecture is first established to define the functional skeleton of the system, followed by the construction of tailored engineering semantic models to support its cross-phase implementation.

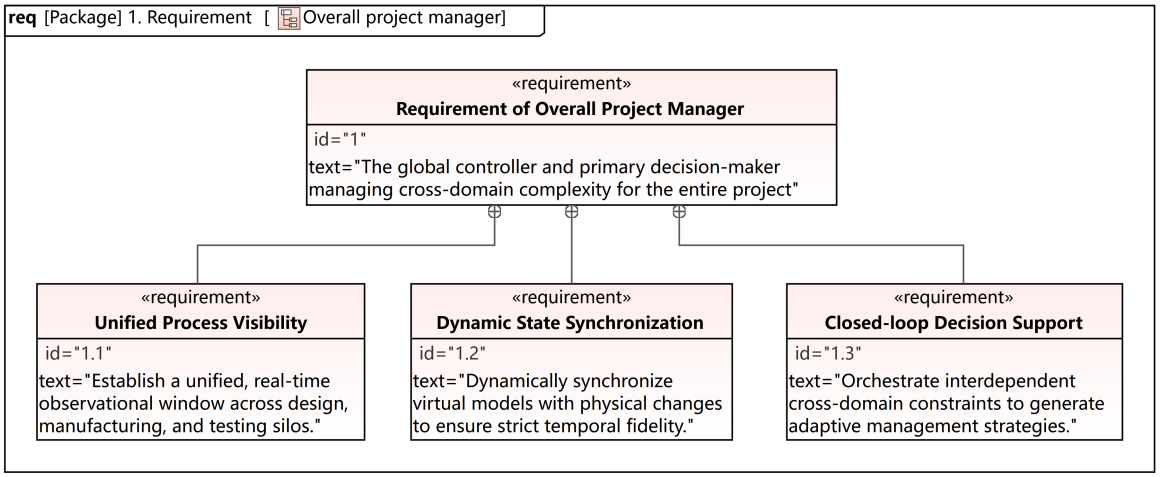


Figure 3. Stakeholder needs analysis definition (B1-W1).

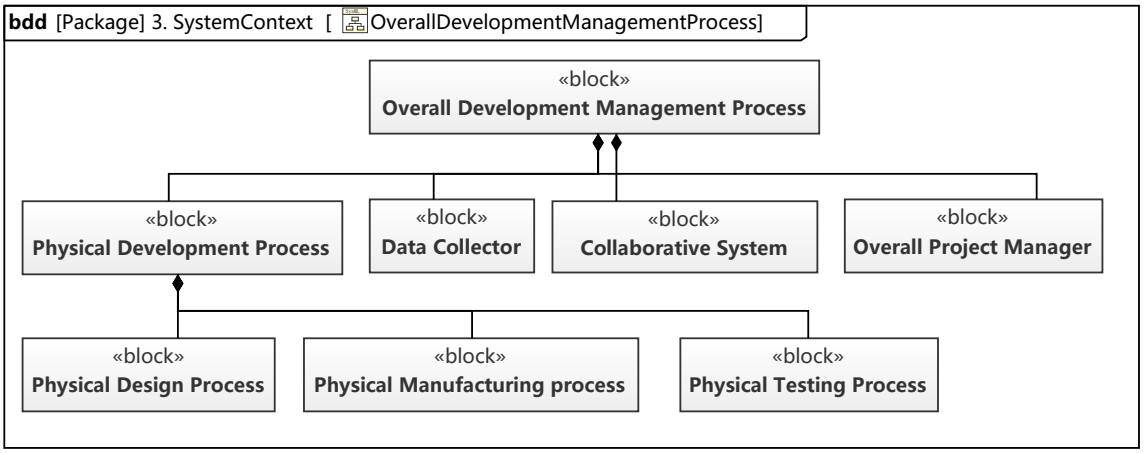


Figure 4. System context and operational boundary definition (B3).

3.3. The Layered Logic Architecture

In the solution domain, a four-layer architecture is established to organize the capabilities required for an end-to-end cross-phase collaboration loop, including physical–virtual mapping, semantic reasoning, adaptive decision support, and service interaction. As illustrated in the SysML block definition diagram in Figure 5, the architecture formally defines the system composition and the specialized modules within each layer.

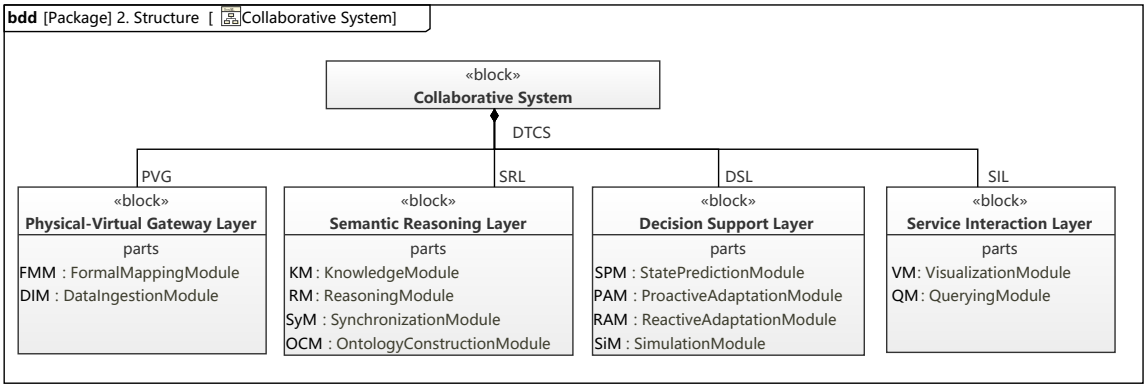


Figure 5. SysML BDD of the framework's layered logic architecture (SS3-1).

Compared with generic Edge–PaaS–SaaS architectures that primarily organize device connectivity, platform capabilities, and user applications, the proposed four-layer archi-

tecture is organized according to the management process required to respond to runtime disturbances across design, manufacturing, and testing. The comparison is summarized in Table 2. Its MBSE-driven characteristic is reflected in the derivation and formal definition of layer responsibilities, internal modules, and interfaces through the MagicGrid process and SysML models. Its semantic-enhanced characteristic is reflected in the use of collaboration-specific information representations, ontology-based state synchronization and impact reasoning, and the explicit transformation of reasoning results into decision inputs. The distinction is therefore reflected in both the specialized functions within the layers and the interaction mechanism between them, particularly the continuous connection from physical information and semantic interpretation to adaptive decision-making.

Table 2. Comparison between generic Edge–PaaS–SaaS architectures and the proposed four-layer functional architecture.

Comparison Aspect	Generic Edge–PaaS–SaaS Architecture	Proposed Four-Layer Functional Architecture
Primary organizing concern	Device connectivity, platform capabilities, and application-service delivery	End-to-end collaborative management across design, manufacturing, and testing
Layer responsibilities	Data acquisition and preprocessing, platform computing and data services, and user-facing applications	Physical–virtual mapping, semantic state synchronization and impact reasoning, adaptive decision support, and service interaction
Inter-layer information	Device data, platform messages, API requests, and application results	Typed task, resource, material, event, semantic-state, decision-constraint, and decision-result objects
Reasoning-to-decision connection	Defined according to the requirements of individual applications	Inferred states and propagated impacts are explicitly transformed into LogicConstraint inputs for decision models
Engineering traceability	Determined by the modeling and implementation method adopted for the target system	Stakeholder needs, requirements, layer functions, internal modules, interfaces, and validation evidence are linked through the MagicGrid and SysML models
Adaptive collaboration loop	Constructed within individual applications when required	Physical mapping–semantic reasoning–constraint generation–adaptive decision–service interaction

3.3.1. Hierarchical Structural Definition

The physical-virtual gateway functions as the bottom-level data foundation, enabling the connectivity required by upper-level requirements. By employing a data ingestion module to handle heterogeneous protocols and a formal mapping module to execute the synchronization mechanism, this layer transforms raw signals into standardized instances. This data standardization process is a prerequisite for achieving unified process visibility and provides the necessary real-time input stream for the upper layers.

Above the gateway, the semantic reasoning layer constructs the DT context that bridges data and decisions. The ontology construction module maintains static knowledge, while the synchronization module updates entity states based on real-time data streams. This layer does not function in isolation but collaboratively supports dynamic state synchronization by ensuring semantic consistency. Furthermore, the reasoning module provides

the inferential basis required to identify hidden logic constraints, such as implicit resource conflicts, which are invisible to traditional monitoring systems.

The decision support layer serves as the computational core, leveraging the semantic state provided by the reasoning layer to execute the predictive and adaptive management loop. The state prediction module assesses probabilistic risks such as potential delays, while the proactive and reactive adaptation modules generate cross-phase management strategies. The effectiveness of these decision models relies on the high-fidelity state data from the lower layers, ensuring that collaborative decisions are grounded in physical reality.

Finally, the service interaction layer functions as the integrated interface where the functionalities of all underlying layers converge. Through the visualization and querying modules, the layer aggregates raw data, semantic states, and scheduling schemes into user-friendly dashboards. This design culminates in the realization of unified process visibility and provides the human-in-the-loop interaction capabilities necessary to validate and enact the collaborative scheduling results.

3.3.2. Inter-Layer Interaction Mechanism

To formalize the information exchange that drives the adaptive collaboration loop, the SysML internal block diagram is presented in Figure 6. The diagram defines the ports, information flows, and interface objects connecting the four layers. The collaboration-specific information types shown in Figure 6 are formally specified through the SysML Profile in Section 3.4.

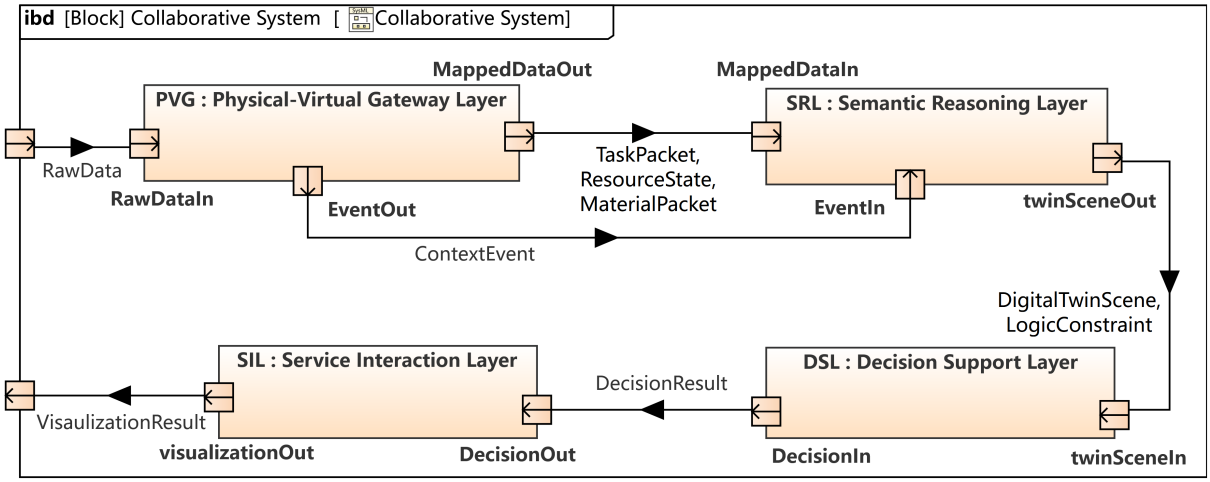


Figure 6. SysML IBD defining cross-layer data interactions and flows (SS3-2).

Regarding physical data ingestion and mapping (PVG → SRL), the Data Ingestion Module receives heterogeneous operational records and event streams from the design, manufacturing, and testing phases and performs basic protocol and format processing. The Formal Mapping Module then organizes and validates the processed records according to the SysML Profile. The resulting task, resource, and material instances are transmitted to the SRL through the MappedData interface, while detected disturbances are transmitted as ContextEvent signals.

Within the SRL, the MappedData received from the PVG enters the Ontology Construction Module through the ProcessedInitialData port. The OCM transforms these inputs into EntityData, including ontology individuals, properties, and relations, and transmits them to the Synchronization Module. The SyM maintains the current runtime states and passes the relevant IndividualData to the Reasoning Module when attributes or dependency relationships change. The ReM applies the rules retrieved from the Knowledge Module and returns the resulting ReasonedData to the SyM, as detailed in Figure 7.

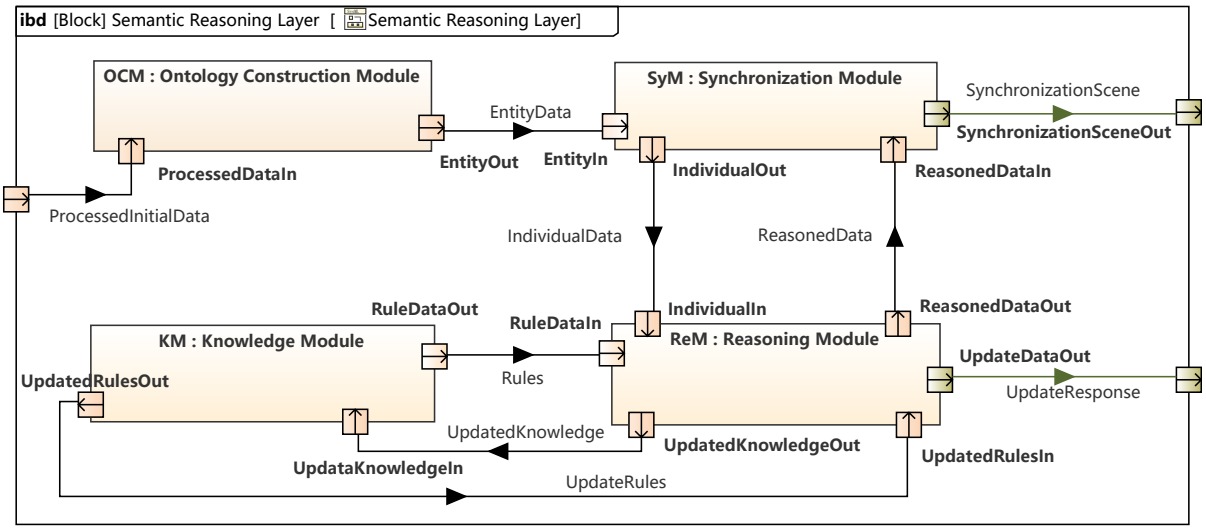


Figure 7. IBD of the SRL (C3-1).

Regarding reasoning-to-decision interaction (SRL → DSL), the synchronized scene generated by the SyM is exposed at the SRL boundary as the DigitalTwinScene. In addition, restrictions and propagated impacts identified by the reasoning process are packaged as decision-constraint objects, denoted as LogicConstraint and formally defined in Section 3.4. The DigitalTwinScene describes the current task, resource, material, and dependency states, whereas the decision constraints specify the affected subjects and the corresponding operational restrictions. The DSL therefore receives a decision problem that has already been updated according to the semantic reasoning results, rather than directly processing raw physical data.

Finally, regarding decision execution and service interaction (DSL → SIL), the DSL invokes the appropriate proactive or reactive adaptation strategy based on the DigitalTwin-Scene and the reasoning-derived constraints. The generated scheduling and simulation results are encapsulated in a unified DecisionResult, which is transmitted to the SIL through the ResultVisualization interface for visualization, querying, and human confirmation.

3.3.3. Structural Requirement Traceability and Verification

A critical tenet of MBSE is the verifiable alignment between the problem domain and the solution domain. To strictly validate the logical consistency of the proposed framework, a traceability analysis was performed, as presented in the satisfaction matrix in Figure 8.

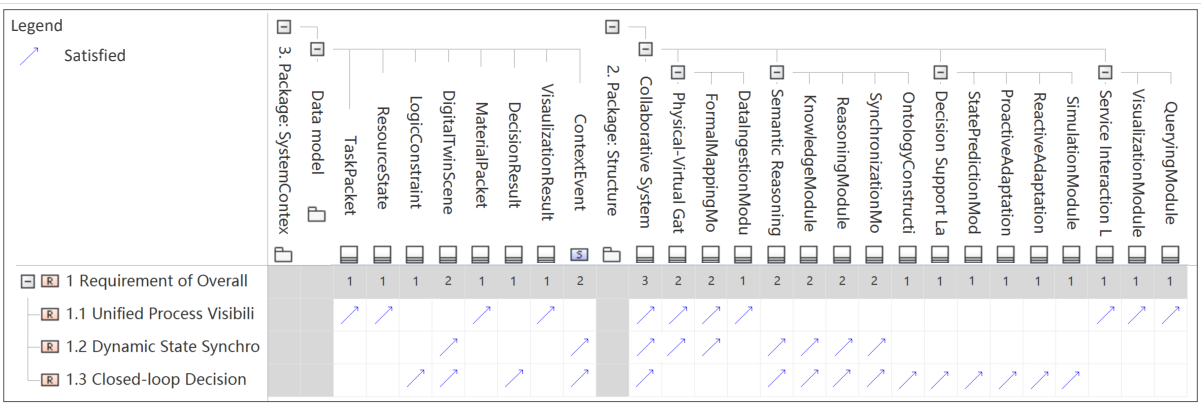


Figure 8. Requirement satisfaction matrix for architectural verification.

Regarding the requirement for bridging information silos (Req 1.1), the architecture establishes the necessary foundation as indicated in the first row. Specifically, the matrix demonstrates that the physical-virtual gateway achieves data normalization through the TaskPacket definition, while the service interaction layer enables interactive data retrieval via the querying module. Furthermore, the traceability links validate the event-driven nature of the system (Req 1.2). The design explicitly maps physical disturbances to ContextEvent signals, which are captured by the formal mapping module and subsequently propagated by the synchronization module to update the DigitalTwinScene in real-time. Finally, the matrix confirms the realization of closed-loop decision support (Req 1.3). The design grounds the decision support layer in tailored engineering rules. By feeding the decision models with semantically derived LogicConstraint objects rather than raw data, the framework ensures that the generated DecisionResult is both computationally derived and semantically valid.

3.4. Collaboration-Oriented SysML Profile

To resolve the spatial semantic heterogeneity among participating units, we construct a tailored cross-phase SysML Profile. The Profile extends selected SysML metaclasses through stereotypes, tagged values, and constraints, thereby providing a common representation of the information exchanged across the design, manufacturing, and testing phases. Rather than attempting to represent every domain concept throughout the complete lifecycle of a complex product system, the Profile defines a core set of five stereotypes according to their roles in the cross-phase collaboration loop. These stereotypes are organized into two functional groups: collaborative entity representations and dynamic interaction mechanisms.

As summarized in Table 3 and formally specified in Tables 4 and 5, TaskPacket, ResourceState, and MaterialPacket represent the three primary types of managed information required for collaborative task execution: process activities, execution capabilities, and material inputs. They provide normalized representations of task properties and dependencies, heterogeneous resource states and capabilities, and material availability and quality conditions. The other two stereotypes define the dynamic interaction mechanisms of the collaboration loop. ContextEvent represents runtime disturbances that trigger state updates, impact propagation, or dependency changes. LogicConstraint encapsulates reasoning-derived restrictions as explicit and machine-processable inputs to the decision support layer, thereby allowing decision models to use the reasoning results without directly querying the complete ontology graph. Together, the five stereotypes cover the core information chain from cross-phase state representation and disturbance perception to reasoning-to-decision interaction.

The five stereotypes are considered functionally sufficient for the scope of this study because they support the information required for cross-phase task execution, resource and material readiness, disturbance representation, semantic impact reasoning, and reasoning-to-decision interaction. This sufficiency refers to the mapping–synchronization–reasoning–decision loop demonstrated in this study; the Profile is not intended to constitute an exhaustive lifecycle metamodel for all complex product system domains.

Quality- and risk-related information is partially represented in the current core Profile. Material quality conditions are captured by MaterialPacket.qualityStatus, while task uncertainty and event criticality are represented by TaskPacket.uncertaintyFactor and ContextEvent.severityLevel, respectively. More detailed quality standards, risk states, cost parameters, configuration versions, or compliance requirements may be introduced as domain-specific stereotypes and linked to the existing core types when required by a particular application.

Table 3. Coverage of cross-phase information requirements by the core SysML profile.

Cross-Phase Information Requirement	Profile Stereotype	Representative Attributes	Role in the Collaboration Loop
Task identity, duration, priority, and dependency	TaskPacket	taskID, plannedDuration, priorityLevel, predecessorID	Process-topology representation, task-state synchronization, and delay-impact propagation
Resource availability and execution capability	ResourceState	resourceID, category, isAvailable, healthIndex, capabilitySet	Resource-feasibility evaluation, capability matching, and disruption-impact analysis
Material availability and quality state	MaterialPacket	materialID, arrivalStatus, qualityStatus, quantity	Material-readiness evaluation and identification of downstream supply constraints
Runtime disturbances and their affected scope	ContextEvent	eventID, eventType, impactScope, validDuration, severityLevel	Triggering runtime state updates, impact propagation, or dependency-topology changes
Reasoning-derived decision boundaries	LogicConstraint	triggerEvent, targetSubject, boundaryExpression, policyType, penaltyFactor	Transferring inferred restrictions to scheduling, simulation, and optimization models

Table 4. Formal specification of core collaborative entities.

1. TaskPacket (Process standardization)
stereotype TaskPacket extends Action {
  tag taskID : String [0..∗];
  tag plannedDuration : Real [1..1] {unit = “Hour”; pattern = “val $>$ 0.0”;};
  tag priorityLevel : Integer [0..1] {defaultValue = 3; range = $[1, 5]$;};
  tag predecessorID : String [0..∗];
  tag uncertaintyFactor : Real [0..1];
  constraint ValidTime {context TaskPacket inv: self.plannedDuration $\leq$ 200.0};
}
2 . ResourceState (Capacity standardization)
stereotype ResourceState extends Block {
  tag resourceID : String [1..1] {pattern = “RES-\d{4}”;};
  tag category : Enumeration {literals = {Human, Equipment, Space};};
  tag isAvailable : Boolean [1..1];
  tag healthIndex : Real [0..1] {description = “0.0=Failure, 1.0=Perfect”;};
  tag capabilitySet : String [1..∗];
}
3. MaterialPacket (Supply standardization)
stereotype MaterialPacket extends Block {
  tag materialID : String [1..1] {pattern = “MAT-\d{6}”;};
  tag arrivalStatus : Enumeration {literals = {Arrived, InTransit, Delayed};};
  tag qualityStatus : Enumeration {literals = {PendingInspection, Qualified, Defective};};
  tag quantity : Integer [1..1] {pattern = “val $\geq$ 0”;};
  constraint InspectionStateConsistency {
    context MaterialPacket inv:
    (self.qualityStatus $\neq$ PendingInspection) implies
    (self.arrivalStatus == Arrived)
  };
}

Table 5. Formal specification of control logic interfaces and contextual mechanisms.

```
4. LogicConstraint (The decision boundary interface)
stereotype LogicConstraint extends ConstraintBlock {
   tag triggerEvent : String [1..1] {
      allowedValues = {"OnDelay", "OnMachineFailure", "OnUrgentInsert"};
   };
   tag targetSubject : String [1..*];
   tag boundaryExpression : String [1..1];
   tag policyType : Enumeration  literals = HardBoundary, SoftPenalty;
   tag penaltyFactor : Real [0..1];
}
5. ContextEvent (The macro disruption)
stereotype ContextEvent extends Signal {
   tag eventID : String [1..1] {
      pattern = "EVT-\d{4}";
   };
   tag eventType : Enumeration {
      literals = {
         DesignChange, // Technical disruption
         PolicyChange, // Managerial disruption
         UrgentOrder // Commercial disruption
      };
   };
   tag impactScope : String [1..1] {
      description = "Target ID (Task/Resource) or 'Global"';
   };
   tag validDuration : Real [0..1] {
      unit = "Hour";
      description = "Estimated duration of the impact";
   };
   tag severityLevel : Integer [1..5]; // 1=Minor, 5=Critical
}
```

3.5. Formal Mapping Mechanism for Physical-Virtual Synchronization

This section formalizes the mapping function $\mathcal{M}$, which transforms heterogeneous data from the cross-phase physical development processes into the strictly typed SysML profile instances.

We formalize the physical development space as a tuple $\mathcal{P}_{phy} = \{A, R, M, E, L\}$, representing the sets of development activities, resources, component supplies, environmental events, and management policies. The mapping $\mathcal{M} : \mathcal{P}_{phy} \rightarrow \mathcal{V}_{model}$ defines how each physical element is transformed into a Profile-aligned virtual representation.

For any development activity $a_i \in A$, the function generates a TaskPacket instance via Equation (1).

$$\mathcal{M}(a_i) = \langle id_i, t_i, p_i, d_i, u_i \rangle \tag{1}$$

id_i ensures system-wide traceability. t_i populates the plannedDuration attribute based on historical cycle data, while p_i quantifies the strategic importance for priorityLevel. d_i maps to predecessorID to reconstruct the collaborative project topology, and u_i captures the R&D stochasticity within the uncertaintyFactor attribute.

Resources $r_j \in R$ are mapped to ResourceState instances via Equation (2).

$$\mathcal{M}(r_j) = \langle id_j, cat_j, s_j, h_j, cap_j \rangle \tag{2}$$

The binary variable s_j corresponds to the isAvailable tag. h_j normalizes sensor data into the dimensionless healthIndex, representing capability saturation rather than simple mechanical wear. cap_j populates the capabilitySet to list specific cross-phase qualifications.

Component supplies $m_k \in M$ are transformed via Equation (3).

$$\mathcal{M}(m_k) = \langle id_k, as_k, qs_k, q_k \rangle \tag{3}$$

This mapping directly populates the arrivalStatus and qualityStatus enumerations through variables as_k and qs_k. This enables the reasoning layer to identify potential starvation risks where downstream tasks lack certified inputs.

Environmental events $e_l \in E$ are formalized into ContextEvent signals via Equation (4).

$$\mathcal{M}(e_l) = \langle id_l, type_l, scope_l, dur_l, sev_l \rangle \tag{4}$$

The function categorizes the disruption source into eventType via $type_l$ and defines the impact boundary through $scope_l$. dur_l and sev_l provide the quantitative basis for impact analysis by populating the validDuration and severityLevel attributes.

To bridge physical directives and the virtual reasoning engine, management policies $l_n \in L$ are formalized via Equation (5).

$$\mathcal{M}(l_n) = \langle trig_n, pol_n, ref_n \rangle \tag{5}$$

This mapping defines a baseline policy-to-rule specification for the subsequent derivation of LogicConstraint objects, rather than directly producing a complete constraint instance. Specifically, $trig_n$ corresponds to triggerEvent, pol_n determines policyType, and ref_n identifies the associated SWRL rule. During runtime reasoning, this rule is evaluated together with the current ontology state and event context to derive targetSubject, boundaryExpression, and, where applicable, penaltyFactor, thereby forming a complete LogicConstraint for downstream decision support.

At the software level, Equations (1)–(5) are implemented as declarative field-level mapping specifications rather than SWRL reasoning rules. The FMM operationalizes them through field extraction, data-type conversion, Profile-constraint validation, and typed-object construction. Validated objects are subsequently converted by the OCM into ontology individuals and property assertions; for example, TaskPacket is represented as an Activity individual, plannedDuration as hasDuration, and predecessorID as hasPredecessor. A ContextEvent remains a typed trigger input, while Equation (5) supplies the policy-to-rule association. These validated facts and relations provide the structural inputs for the reasoning mechanisms in Section 4.2: the mapping specifications generate ontology facts, whereas the SWRL rules infer state propagation, topology changes, and decision constraints. A lightweight executable demonstration is presented in Section 5.2.

4. Knowledge-Driven Adaptive Collaboration Mechanism

The structural framework established in Section 3 effectively ensures spatial syntactic interoperability through standardized data profiles. However, static data mapping alone remains insufficient for dynamic control under stochastic disturbances. To bridge the temporal gap between the static model and the time-varying physical process, this section proposes a knowledge-driven adaptive collaboration mechanism. Its theoretical core is the knowledge metamodel ($\mathcal{K}$), formally defined as a tuple:

$$\mathcal{K} = \langle \mathcal{O}, \mathcal{R}, \mathcal{S} \rangle \tag{6}$$

where $\mathcal{O}$ denotes the ontological knowledge, $\mathcal{R}$ denotes the procedural knowledge, and $\mathcal{S}$ denotes the strategic knowledge. Building upon $\mathcal{O}$, SRL maintains the synchronized static DT model. Meanwhile, through $\mathcal{R}$, it reasons about the model's dynamic evolution, and guided by $\mathcal{S}$, it generates adaptive management strategies.

4.1. Ontological Knowledge Structure ($\mathcal{O}$)

The ontology $\mathcal{O}$ constitutes the semantic backbone of the framework. It transforms the standardized but flat data packets from the PVG into a multidimensional semantic graph. We define $\mathcal{O}$ as a 6-tuple:

$$\mathcal{O} = \langle C, I, R_o, R_d, Ax, \Sigma \rangle \tag{7}$$

The constituent elements of Equation (7) are defined as follows:

(1) Classes (C): C denotes the set of class definitions acting as the schema for domain entities. To ensure strict structural consistency, elements in C are derived directly from the collaboration-oriented SysML profile, as specified in Equation (8).

$$Class = \langle Activity, Resource, State, Skill, Condition \rangle \tag{8}$$

The architectural logic seamlessly translates communication packets into semantic states. The classes *Activity* and *Resource* provide the fundamental schema for digital instances, where materials are structurally subsumed under the *Resource* class to maintain a streamlined semantic taxonomy. Crucially, dynamic disruptions (transmitted from physical development space as ContextEvent signals) do not generate isolated event individuals. Instead, these signals trigger real-time state mutations—such as fluctuations in availability or health indices—within the corresponding *Activity* or *Resource* individuals. The ontology then autonomously infers the semantic implications of these state changes. Finally, lightweight profile attributes are elevated into semantic concepts; for instance, elements in a capability set are instantiated as individuals of class *State*, enabling logical inference rather than simple data storage.

(2) Individuals (I) and Properties (R_o, R_d): I represents the set of individuals instantiated from the validated Profile objects generated by the formal mapping mechanism ($\mathcal{M}$) in Section 3.5. The OCM converts the resulting $\mathcal{M}(a_i)$ and $\mathcal{M}(r_j)$ objects into individuals asserted to the Activity and Resource classes, respectively. Similarly, the material mapping $\mathcal{M}(m_k)$ is represented as a supply individual subsumed under the Resource taxonomy. During this transformation, plannedDuration is represented through the data property hasDuration, while predecessorID values are converted into hasPredecessor object-property assertions.

To articulate the multidimensional semantic network between these individuals, two property types are utilized: object properties (R_o) and data properties (R_d). R_o establishes logical dependencies between two individuals, while R_d binds literal values to specific individuals. Crucially, the environmental event mapping $\mathcal{M}(e_l)$ (Equation (4)) does not create standalone event individuals within I. Instead, the validated ContextEvent object is supplied as a trigger input to the procedural rules, which determine whether and how the corresponding R_o or R_d of the affected activity or resource individuals should be updated.

(3) Axioms (Ax) and Constraints (Σ): Ax governs the foundational TBox logic, establishing fundamental rules such as class disjointness and property domains. In addition, Σ represents the set of validity rules used to enforce data integrity. Crucially, these rules are not arbitrary but are mapped directly from the validation constraints defined in the SysML profile (Section 3.4), ensuring that any instantiated individual violating the original design specifications is automatically flagged as invalid.

4.2. Procedural Knowledge ($\mathcal{R}$) and Event-Driven Reasoning

While the ontology ($\mathcal{O}$) defines the static graph structure, the procedural knowledge ($\mathcal{R}$) governs the dynamic logic of state transitions. In this framework, $\mathcal{R}$ encapsulates the SWRL-based implication rules that process physical disruptions and ultimately generate the LogicConstraint instances (Section 3.4) for downstream decision support. It is formalized as a set of state-transition rules:

$$\mathcal{R} = \{r \mid r : \phi(\mathcal{O}_t, e_t) \rightarrow \psi(\mathcal{O}_{t+1})\} \tag{9}$$

where ϕ denotes a precondition predicate evaluated against the current state $\mathcal{O}_t$ and a detected physical disruption e_t (mapped via ContextEvent), while ψ represents the consequent state update. The antecedent facts evaluated by R are supplied by the mapping and ontology-instantiation process described in Sections 3.5 and 4.1, including entity individuals, data-property assertions, dependency relations such as hasPredecessor, and validated ContextEvent inputs. Thus, the mapping specifications generate the factual and structural graph inputs, whereas the procedural rules infer new states and relations from these inputs. To support adaptive process control, the execution of $\mathcal{R}$ drives two distinct graph evolution mechanisms:

(1) Parametric State Propagation: This mechanism addresses disturbances where the process structure remains invariant, yet data attributes require synchronization, such as execution delays or equipment efficiency drops. The reasoning engine executes a recursive update function that modifies the data property set (R_d) while maintaining the structural stability of individuals (I) and object relations (R_o). The dependency relations used by this function are instantiated from the predecessorID mappings defined in Section 3.5. The logic is formalized as Equation (10):

$$\begin{aligned} \forall x, y \in I : (\text{hasPredecessor}(y, x) \in R_o \wedge \Delta\text{State}(x) \in R_d) \\ \Rightarrow \text{UpdateState}(y) \in R'_d \end{aligned} \tag{10}$$

By generating the updated property set R'_d, this mechanism maintains state consistency across the dependency network and supports the inference of qualitative states without reconstructing the graph. For example, a numerical delay can be transformed into a State_Blocked status for an affected downstream activity.

(2) Topology Reconfiguration: Unlike traditional static management approaches, this framework supports dynamic structural evolution necessitated by macro-disturbances such as rework loops, task insertion, or cancellation. Upon detecting a structure-altering event e, the reasoning engine fundamentally modifies the population of individuals (I) and their logical connections (R_o). The transformation to the new ontology state $\mathcal{O}'$ is strictly aligned with the tuple defined in Equation (7) and is formalized as Equation (11):

$$f_{reconfig}(\mathcal{O}, e) \rightarrow \mathcal{O}' = \langle C, (I \cup I_{new}) \setminus I_{del}, (R_o \cup R_{new}) \setminus R^*_{del}, R_d, Ax, \Sigma \rangle \tag{11}$$

where I_{new} and I_{del} represent the sets of individuals to be added or removed, respectively, and R_{new}/R^*_{del} denote the corresponding mutation of object property relations. This structural reasoning enables the DT model to autonomously synchronize its semantic topology with the changing physical process.

4.3. Strategic Knowledge ($\mathcal{S}$) and Adaptive Decision Loop

Strategic knowledge ($\mathcal{S}$) functions as a library of adaptive management strategies designed to decouple the decision logic from the core framework execution. We formalize this knowledge as Equation (12):

$$\mathcal{S} = \{\langle s_q, \Gamma_q \rangle\} \tag{12}$$

where s_q represents an executable decision model, such as a heuristic rule set or a global optimization model, and Γ_q denotes its applicability context. In addition to computational complexity and latency characteristics, Γ_q records the supported disturbance type, severity range, and structural impact scope. These metadata provide the basis for selecting an appropriate strategy at runtime. Driven by the reasoning results from Section 4.2, the adaptive decision-making process executes through three coordinated phases:

(1) Context-aware strategy selection: Triggered by a detected state divergence ($\mathcal{O} \neq \mathcal{O}'$), the system first identifies the disturbance type and evaluates its severity and impact scope against the applicability context Γ_q of each decision model.

For a parametric disturbance represented by a predicted activity delay, the relative delay ratio is defined as Equation (13):

$$\rho_d = \frac{\Delta t_{\text{pred}}}{t_{\text{planned}}} \tag{13}$$

where Δt_{pred} denotes the predicted delay duration and t_{planned} denotes the planned duration of the affected activity. Given a configurable switching threshold θ_d, the strategy-selection rule is defined as Equation (14):

$$s_q = \begin{cases} s_{\text{Rule}}, & \rho_d < \theta_d \\ s_{\text{GA}}, & \rho_d \geq \theta_d \end{cases} \tag{14}$$

Accordingly, locally manageable deviations are handled by the low-latency heuristic strategy, whereas disturbances exceeding the configured severity threshold activate the global optimization model to support broader task and resource coordination.

For a topological disturbance, the strategy is selected according to its structural impact scope rather than the delay ratio. A local insertion affecting a single activity or dependency relation without project-wide resource reallocation is handled by s_{Rule}. In contrast, a structural disturbance involving multiple process branches, shared resources, or project-wide rescheduling activates s_{GA}.

The threshold θ_d is treated as a configurable engineering parameter rather than a universal constant. Depending on the application context, it may be calibrated using historical project records, organizational control rules, expert elicitation, or an explicit trade-off between decision quality and computation time. The case-specific threshold used in the validation is specified in Section 5.2.

(2) Semantic constraint injection: This phase bridges the gap between semantic reasoning ($\mathcal{R}$) and dynamic decision support ($\mathcal{S}$). Before execution, the selected decision model s_q is initialized with a dynamically formulated problem space derived from the reasoning outputs. Specifically, restrictive states inferred via Equation (10) are translated into strict operational boundaries (encapsulated as LogicConstraint objects). In contrast, structural expansions involving new individuals and relations derived from Equation (11) are integrated as adaptable decision parameters. This transformation ensures that the situational awareness of the DT strictly bounds the solution space of the decision model.

(3) Strategy formulation and closure: Finally, the decision model s_q executes the adaptation process utilizing the transformed constraints as boundaries. The resulting cross-phase management strategy (encapsulated as a DecisionResult) is transmitted to the SIL for visualization and human confirmation. This step effectively completes the closed-loop cycle for the collaborative management of complex product development.

5. Instantiation and Validation

To validate the feasibility and effectiveness of the proposed framework, we instantiated a digital twin collaborative system (DTCS) for the development of aero-engine turbofan blades. This component represents a quintessential CoPS, characterized by lengthy development cycles and rigorous interdisciplinary coupling across design, manufacturing, and testing phases.

5.1. Case Scenario Description

The validation scenario simulates a full-cycle development process derived from historical industrial data. As detailed in Table 6, the workflow consists of 14 coupled activities ($A1$–$A14$) and involves 15 heterogeneous resource types, ranging from human experts to specialized equipment.

Table 6. Activities in the DTCS-based case scenario.

ID	Activity Name	Duration (h)	Predecessor(s)	Resource Required
A1	System Requirement Definition	24.0	–	Engineer_Li, Engineer_Wang
A2	Conceptual Design	40.0	A1	Engineer_Wang, Engineer_Zhang
A3	Detailed Design & Simulation	72.0	A2	CAEWorkstation_01, Engineer_Liu
A4	Auxiliary Trial Casting	16.0	A1	Furnace_01, Engineer_Wang
A5	Material Coupon Benchmark Test	16.0	A1	CreepTester_01, Metallographic_01, Engineer_Liu
A6	Prototype Preparation	16.0	A3	Furnace_01, Engineer_Wang
A7	Test Rig Setup	16.0	A3	CreepTester_01, Metallographic_01, Engineer_Liu
A8	Prototype Casting	24.0	A3, A6	Furnace_01, Engineer_Wang
A9	Precision Machining	12.0	A8	5-axisMachiningCenter, Engineer_Li
A10	Surface Coating	8.0	A9	SprayRoom_01, Engineer_Zhang
A11	Heat Treatment	10.0	A10	HeatTreatFurnace_01, Engineer_Wang
A12	NDT Inspection	8.0	A11, A7	CMM_Machine_01, CT_Scanner_01, Operator_Zhao
A13	Performance Bench Test	24.0	A12	CreepTester_01, Metallographic_01, Engineer_Liu
A14	Qualification Test	48.0	A13	EnvironmentChamber_01, CreepTester_01, Engineer_Wang

This process is inherently vulnerable to stochastic disturbances due to its rigid cross-phase dependency structure. For instance, the Prototype Casting ($A8$) in the manufacturing phase cannot commence until both detailed design & simulation ($A3$) and prototype preparation ($A6$) are fully completed. Furthermore, critical resources such as the creep tester and metallographic analyzer are shared across multiple distributed stages ($A5$, $A7$, $A13$), creating potential bottlenecks where a single equipment failure can trigger cascading delays throughout the project network. The primary objective of the instantiated DTCS is to digitize this complex environment and provide adaptive cross-phase management capabilities under such uncertainty.

5.2. Framework Instantiation and Proof-of-Concept Implementation

Following the architectural blueprint defined in the solution domain, this phase executes the implementation-domain tasks through a case-level instantiation of the proposed framework. The implementation combines scenario-based ontology construction, a lightweight executable mapping proof of concept, and a web-based visualization demonstrator to illustrate the relationships among the ontological knowledge ($\mathcal{O}$), procedural knowledge ($\mathcal{R}$), and strategic knowledge ($\mathcal{S}$). Table 7 summarizes how the responsibilities of the four layers defined in Section 3.3 are represented in the case implementation and examined in the subsequent validation scenarios.

Table 7. Case-level realization of the proposed four-layer architecture.

Layer	Architectural Responsibility	Case-Level Realization	Validation Evidence
PVG	Physical data ingestion and formal mapping	Representative task, resource, material, event, and policy records are processed by the executable mapping proof of concept and converted into validated Profile objects	Representative valid mappings and deliberate invalid-input validation
SRL	Ontology construction, runtime synchronization, and semantic reasoning	Validated Profile objects are represented as ontology individuals, properties, and relations, while the SWRL-based state-propagation and topology-evolution mechanisms are demonstrated through the validation scenarios	Scenario-based downstream-state inference and dependency-relation reconstruction
DSL	Constraint-driven adaptive decision support	A configurable strategy library and constraint-processing logic are configured in Python (v3.10.4) for the case evaluation	Reasoning-derived constraints are represented as inputs to the decision models in the validation scenarios
SIL	Result visualization, querying, and human interaction	A web-based dashboard presents collaborative workflows and decision results	Adapted schedules and comparative performance results

5.2.1. Architectural Instantiation and Data Standardization

The implementation commenced with the instantiation of the four-layer architecture to bridge information silos. The PVG layer was initialized to ingest the heterogeneous process data defined in Table 6, which originates from fragmented departments including design, manufacturing, and testing. To unify these disparate data formats for global management, the formal mapping mechanism ($\mathcal{M}$) was executed. Taking the activity $A8$ as a representative instance, this task originates from the manufacturing sector, and the raw operational constraints were formalized into a standardized TaskPacket object: the execution window was mapped to the plannedDuration tag, while its dependencies on $A3$ and $A6$ were encoded in the predecessorID tag. Simultaneously, the required Vacuum Furnace was instantiated as a ResourceState object. Its physical nature was explicitly defined via the category tag as Equipment, while its real-time operational status was normalized into a quantitative healthIndex. This standardization enforces a unified semantic format, ensuring that all temporal and capacity constraints are strictly validated against the SysML profile before entering the reasoning layer.

To examine the software executability of the formal mapping mechanism, a lightweight Python proof of concept was developed. The five mappings defined in Equations (1)–(5) were implemented through field extraction, data-type conversion, Profile-constraint validation, and typed-object construction. The illustrative batch contained five valid records, with one representative record for each mapping type, and one deliberately invalid ResourceState record. The valid records produced typed Profile objects, and the applicable entity records were further serialized into representative RDF assertions. The invalid resource record was correctly rejected because its identifier did not follow the required RES- prefix followed by exactly four digits, while the remaining records continued to be processed. All eight unit tests passed. This proof of concept demonstrates the executable mapping and validation chain but does not execute SWRL reasoning or constitute an integrated digital-twin backend.

5.2.2. Ontological and Procedural Knowledge Construction

The ontology defines five core conceptual classes that abstract key elements of the development process based on Equation (8). On this basis, the semantic structure is specified by object properties R_o and data properties R_d based on Equation (7). The ontology was constructed and visualized using Protégé (v5.6.7) with the OntoGraf plugin. Object properties capture binary relations between entities, as illustrated in Figure 9. Data properties represent quantitative attributes associated with entities. To support the adaptive decisions in the Decision Support Layer (DSL), specific properties are defined as shown in Table 8.

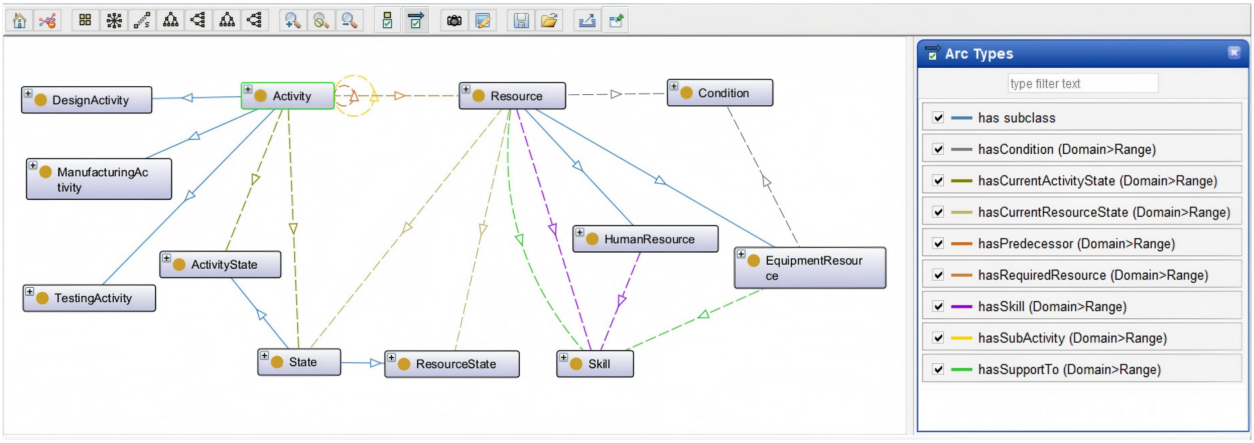


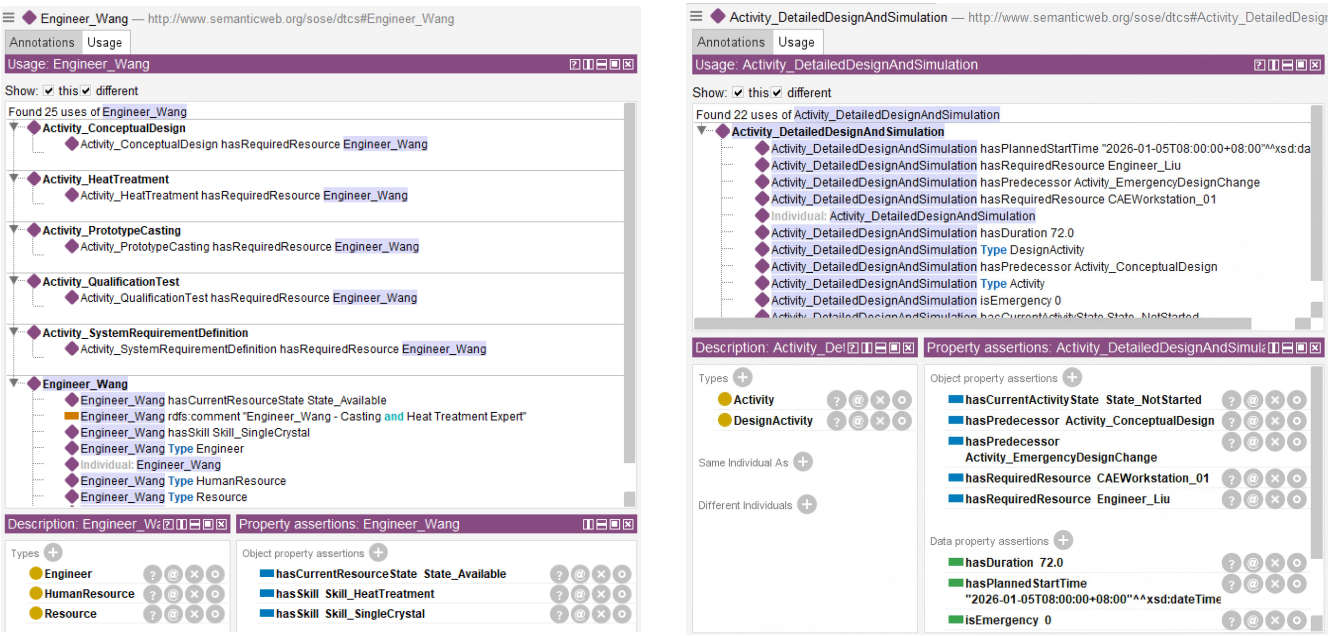
Figure 9. Core conceptual classes and semantic relations of the DTCS ontology.

Table 8. Data properties defined in the DTCS ontology.

Data Property	Domain	Range
hasActualStartTime	Activity	xsd:dateTime
hasCapabilityFactor	Resource	xsd:float
hasDuration	Activity	xsd:float
hasEfficiency	Resource	xsd:float
hasFailurePredictionTime	EquipmentResource	xsd:dateTime
hasFailureProbability	EquipmentResource	xsd:float
hasPlannedStartTime	Activity	xsd:dateTime
hasPredictedDelayDuration	Activity	xsd:float
isEmergency	Activity	xsd:boolean

While the KM provides the static schema, the OCM converts validated Profile objects generated from PVG data into ontology individuals and property assertions. For the

representative A8 record, the validated TaskPacket is represented as an Activity individual. Its plannedDuration is converted into a hasDuration data-property assertion, while the predecessor identifiers A3 and A6 are converted into two hasPredecessor object-property assertions. These assertions provide the factual and structural inputs required by the procedural reasoning rules. Figure 10 presents representative individuals instantiated from the case scenario.



(**a**) A resource individual sample (**b**) An activity individual sample

Figure 10. Representative individuals in the DTCS ontology system.

5.2.3. Strategic Knowledge Instantiation and Prototype Integration

To operationalize the adaptive decision loop, the DSL was configured as a strategy library ($\mathcal{S}$) in a Python-based case-evaluation environment. Following $\mathcal{S} = \{\langle s_q, \Gamma_q \rangle\}$, two decision models were configured. The genetic-algorithm-based model (s_{GA}) supports significant parametric disturbances and globally coupled changes requiring project-wide task and resource coordination. The heuristic rule-based model (s_{Rule}) provides low-latency responses to locally bounded disturbances, such as a single-activity insertion without global resource reallocation.

The strategy-selection mechanism follows the switching logic defined in Section 4.3. For the present case, the relative-delay threshold was configured as $\theta_d = 0.20$ using an engineering-rule-based setting. Predicted delays below 20% of the planned activity duration are handled by s_{Rule}, whereas delays reaching or exceeding 20% activate s_{GA}. For topological disturbances, locally bounded changes activate s_{Rule}, while changes involving multiple process branches or shared resources activate s_{GA}. The threshold is a case-configured parameter rather than a universal constant and may be recalibrated using historical records, organizational control rules, risk tolerance, and available computational resources.

Semantic constraint injection was represented by parsing incoming LogicConstraint objects and converting them into inputs for the selected strategy. For s_{GA}, constraints such as HardBoundary are incorporated into feasibility checks and fitness evaluation; for s_{Rule}, local precedence and topology constraints define rule conditions and adjustment boundaries. The resulting DecisionResult is transmitted to the SIL, where a web-based dashboard visualizes the collaborative workflow and management strategy, as shown in Figure 11.

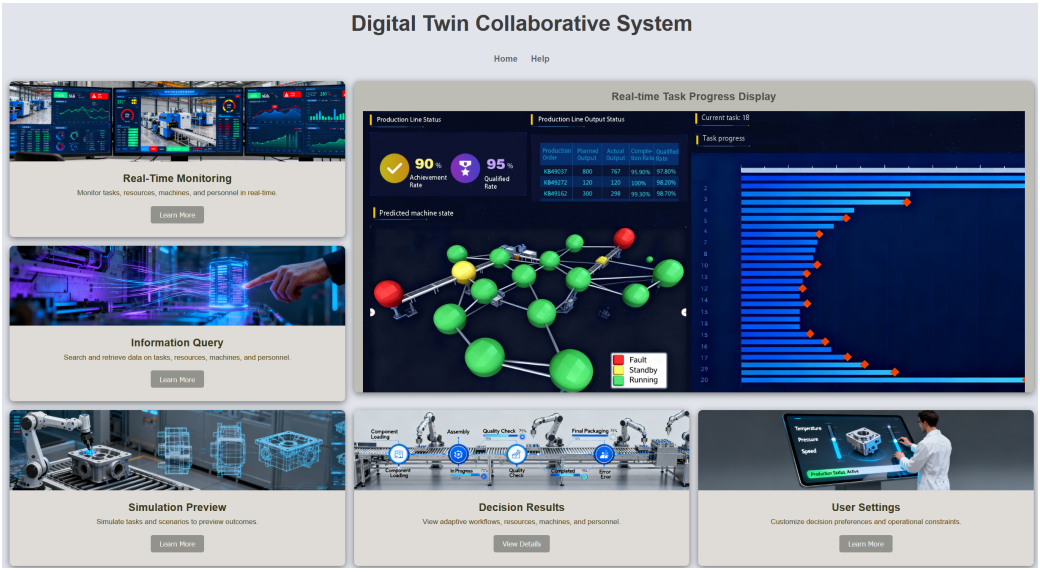


Figure 11. Prototype user interface of the DTCS. This figure is a schematic illustration of the prototype interface; the displayed graphical elements are for illustrative purposes only and do not represent additional quantitative results.

5.3. Validation of the Knowledge-Driven Adaptive Collaboration Mechanism

Building upon the architecture and SysML interfaces instantiated in Section 5.2, this subsection evaluates the core knowledge-driven mechanism. Based on the turbofan blade development scenario defined in Table 6, two targeted scenarios are examined to activate the two fundamental reasoning patterns: parametric state propagation and topology reconfiguration.

Each scenario is organized according to the four-layer closed loop: PVG mapping, SRL synchronization and semantic reasoning, DSL adaptive decision, and SIL result delivery. This organization demonstrates how physical disturbances are transformed into standardized runtime inputs, ontology facts and inferred relations, executable decision constraints, and human-interpretable management results.

5.3.1. Scenario I: Adaptive Collaboration via Parametric State Propagation

To ensure a controlled comparison, the DTCS-assisted mode and the baseline use the same activities, activity durations, resource capacities, precedence relations, disturbance input, and initial schedule. The baseline represents a conventional schedule-driven coordination mode without predictive early warning, ontology-based state propagation, semantic constraint generation, or automatic adaptive rescheduling. Under this mode, the participating departments continue to follow the original schedule until the delay of A3 is manually confirmed and communicated. The affected plan is then adjusted by project managers according to the existing precedence and resource constraints.

(1) PVG Mapping: In the case evaluation, the PVG receives a simulated runtime update indicating a predicted delay of 24 h for A3. The update represents the output of an external prediction module and is used as the runtime disturbance input for evaluating the downstream collaboration mechanism. It is associated with A3 and normalized through the formal mapping mechanism as a typed runtime-state input, including the validated activity identifier, delay duration, and data type. The standardized input is then supplied to the semantic layer.

(2) SRL Synchronization and Semantic Reasoning: Since the A3 activity individual has already been instantiated through the mapping and ontology-construction process described in Section 5.2, the OCM converts the validated runtime input into the data-

property assertion A3 hasPredictedDelayDuration 24.0, where the value is expressed in hours. This synchronization transforms the physical prediction into a machine-interpretable ontology fact, as shown in the top-left pane of Figure 12.

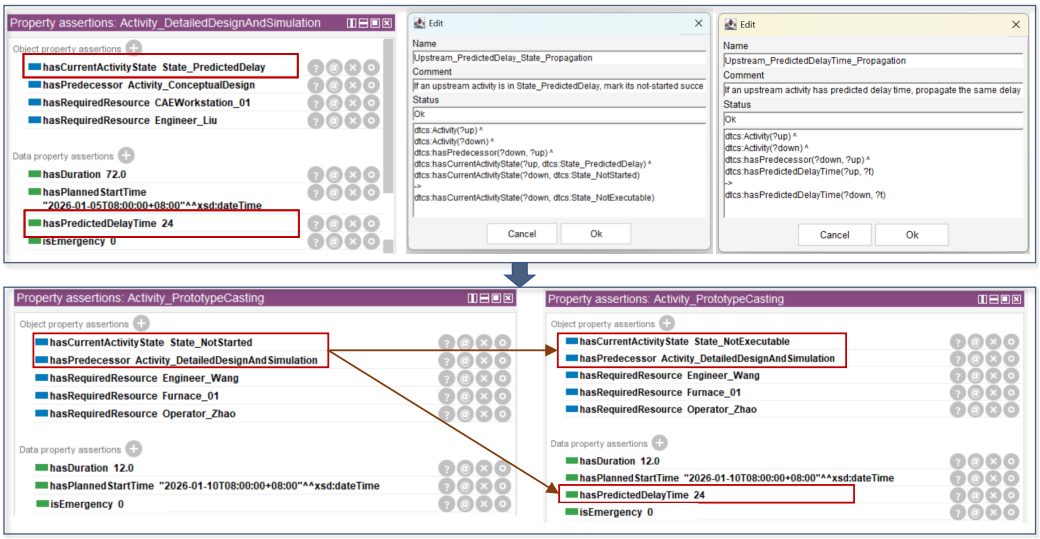


Figure 12. Ontology-based reasoning and delay propagation. The red boxes highlight the relevant ontology assertions before and after reasoning, while the arrows indicate the propagation of the delay impact from the predecessor activity to the downstream activity.

The updated ontology fact activates the Delay Propagation Rule defined in the procedural knowledge $\mathcal{R}$, as shown in the upper-right pane of Figure 12. Since A8 holds a direct hasPredecessor relation with A3, the rule propagates the impact to A8 and asserts A8 hasCurrentActivityState State_NotExecutable. The lower panels of Figure 12 show that the parametric disturbance changes the execution state of the dependent downstream activity without modifying the process topology.

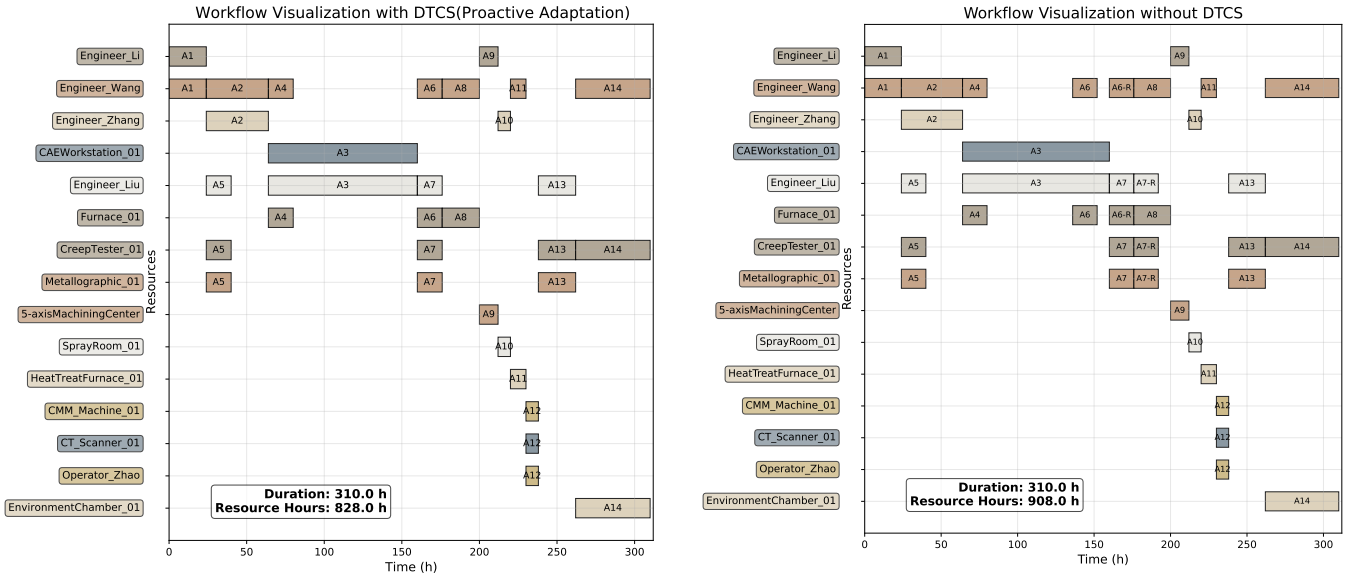
(3) DSL Adaptive Decision: The inferred state evolution activates the strategy-selection mechanism in $\mathcal{S}$. For A3, the relative delay ratio is calculated as $\rho_d = 24/72 = 0.333$, or 33.3%. Because this value exceeds the configured threshold $\theta_d = 0.20$, the DSL selects the global optimization model s_{GA}.

To enforce the inferred state within the decision space, the system generates a LogicConstraint object whose targetSubject is set to A8 to represent the inferred blockage. The constraint is incorporated into the feasibility checks and fitness evaluation of s_{GA}. The model then shifts flexible downstream manufacturing activities before the predicted delay causes cross-phase conflicts and generates the adapted DecisionResult.

(4) SIL Result Delivery: The generated DecisionResult is transmitted to the SIL for visualization and managerial review. As illustrated in Figure 13, the SIL presents the adapted workflow and its project-performance indicators. The DTCS-assisted schedule has a total development cycle of 310 h and consumes 828 resource hours.

Under the conventional manual-coordination baseline, the delay of A3 is not propagated to downstream activities until it is manually confirmed and communicated. Consequently, downstream preparatory activities continue according to the stale initial schedule. In the examined scenario, the premature executions of A6 (Prototype Preparation) and A7 (Test Rig Setup) become invalid after the actual delay is confirmed and are therefore repeated as the rework instances A6-R and A7-R, respectively, as shown in Figure 13. Although the baseline retains the same total project duration of 310 h, it consumes 908 resource hours because of the redundant executions, compared with 828 resource hours under the DTCS-assisted mode. Both modes start from the same 14-activity project definition, resource

capacities, precedence relations, initial schedule, and disturbance input. The comparison therefore highlights the benefit of DT-enabled state synchronization, semantic reasoning, and constraint-driven adaptation in avoiding redundant cross-phase resource occupation.



(**a**) Adaptive DTCS Execution (**b**) Conventional Manual-Coordination Baseline

Figure 13. Comparison between DTCS-assisted adaptation and the conventional manual-coordination baseline.

5.3.2. Scenario II: Adaptive Collaboration via Topology Reconfiguration

(1) PVG Mapping: In the case evaluation, the PVG receives a simulated ContextEvent representing an urgent design-change request during the manufacturing phase. Unlike the parametric disturbance in Scenario I, this event requires an additional activity to be incorporated into the existing process. The formal mapping mechanism validates the event type, impact scope, expected duration, and severity level, and converts the event into a typed trigger input for the semantic layer.

(2) SRL Synchronization and Semantic Reasoning: Based on the validated trigger input, the OCM materializes the event-derived individual Activity_EmergencyDesignChange with the data-property assertions isEmergency = 1 and hasDuration = 8.0. At this stage, the emergency activity is represented in the ontology, while the existing dependency relations remain unchanged, as shown in the top-left pane of Figure 14.

The emergency activity and its isEmergency assertion activate the Emergency Predecessor Rule defined in the procedural knowledge $\mathcal{R}$, as shown in the upper-right pane of Figure 14. The rule identifies eligible common activities whose current state is State_NotStarted and adds the emergency activity as an additional predecessor. Consequently, the inference engine asserts the object-property relation Activity_PrecisionMachining hasPredecessor Activity_EmergencyDesignChange. The lower panels of Figure 14 show that Activity_PrecisionMachining retains its original predecessor Activity_PrototypeCasting while acquiring the additional emergency predecessor, thereby completing the topology reconfiguration.

(3) DSL Adaptive Decision: The topological evolution of $\mathcal{O}$ activates the strategy-selection mechanism in $\mathcal{S}$. Because the disturbance involves a single emergency-activity insertion and a locally bounded precedence update without project-wide resource reallocation, it is classified as a local topological disturbance. The DSL therefore selects the low-latency heuristic strategy s_{Rule}.

The inferred precedence relation is encapsulated into a topology-type LogicConstraint and supplied to s_{Rule} as a rule condition and adjustment boundary. The strategy inserts the emergency cross-phase activity into the affected local workflow while preserving the remaining project structure and generates the corresponding DecisionResult.

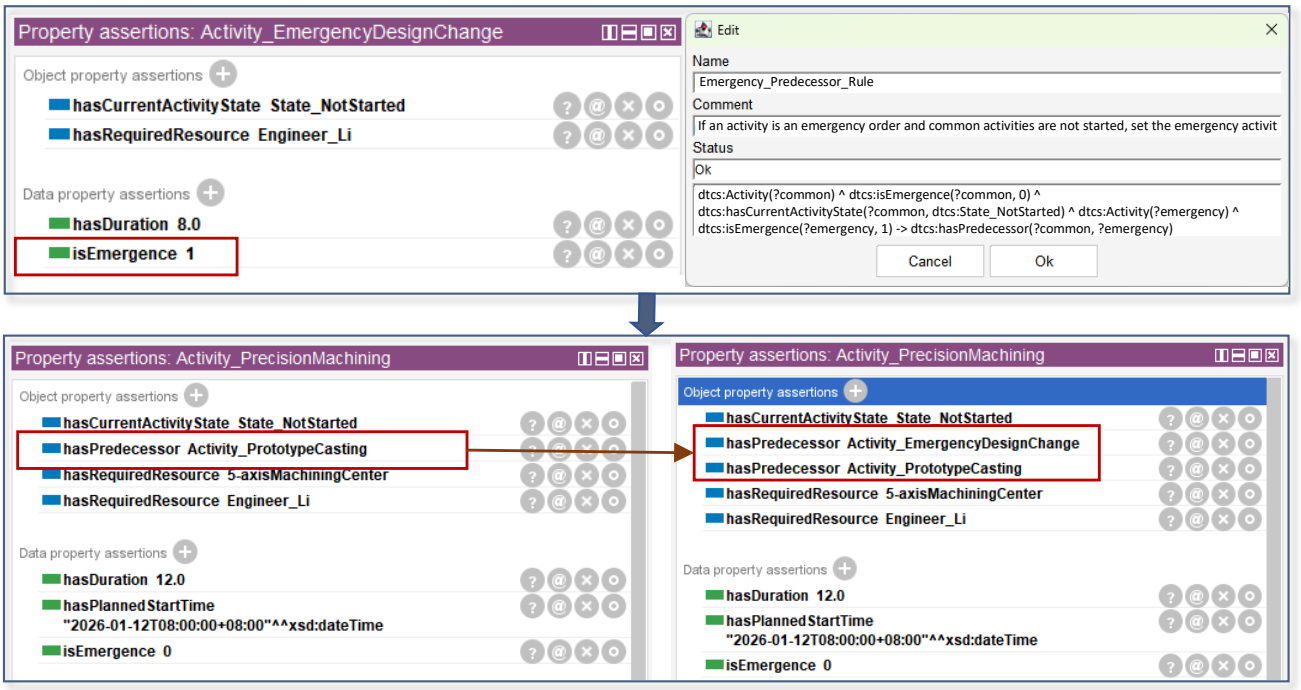


Figure 14. Ontology reasoning and topology reconfiguration. The red boxes highlight the key property assertions involved in the reasoning process. The blue arrow indicates the inference flow from the emergency-event assertion and reasoning rule to the downstream activity, while the red arrow indicates the topology reconfiguration in which the emergency activity is added as an additional predecessor while the original predecessor is retained.

(4) SIL Result Delivery: The generated DecisionResult is transmitted to the SIL, where the revised workflow, the inserted emergency activity, and the updated precedence relation are visualized for managerial review and human confirmation. The adapted workflow has a total project cycle duration of 310 h. The SIL presentation supports communication of the design change and the corresponding local schedule adjustment to the affected design and manufacturing participants.

5.4. Quantitative Assessment of Computational Performance and Collaboration Effectiveness

To evaluate both the operational feasibility and decision quality of the proposed framework, the quantitative assessment considers three complementary dimensions: system responsiveness, collaboration effectiveness, and adaptation efficiency under increasing project complexity. System responsiveness evaluates the latency of physical–virtual synchronization and semantic processing. Collaboration effectiveness evaluates cross-phase handoff timeliness, shared-resource coordination, and the ability to generate feasible responses to plan changes. Adaptation efficiency evaluates the computational cost of the selected decision strategies.

5.4.1. System Responsiveness Analysis

This metric evaluates the latency of the DTCS in reflecting physical changes, formally defined as the total time cost T_{sdt} required to traverse the pipeline from data ingestion to logic constraint generation. Based on the emergency design change experiment in

Scenario II, we measured the constituent time costs. Specifically, $T_{mapping}$ captures the time consumed by the OCM to instantiate the new emergency individual and populate its properties. Following this, $T_{reasoning}$ accounts for the duration required by the inference engine to execute SWRL rules and assert new relationships. Finally, T_{sync} represents the latency for encapsulating the reasoning result into a LogicConstraint and synchronizing it to the Decision Support Layer (DSL).

The measurement results over five independent trials are summarized in Table 9. The data indicates that the average synchronization delay time (T_{sdt}) is 18.4 ms. This validates the preliminary implementation of the framework's semantic processing and state synchronization mechanisms, demonstrating their potential to support real-time DT collaboration.

Table 9. Prototype-level synchronization latency measurements.

Trial	$T_{mapping}$ (ms)	$T_{reasoning}$ (ms)	T_{sync} (ms)	T_{sdt} (ms)
1	4	7	6	17
2	5	9	6	20
3	4	9	7	20
4	5	7	6	18
5	4	7	6	17
Avg.	4.4	7.8	6.2	18.4

5.4.2. Collaboration Effectiveness Analysis

For the effectiveness comparison, the DTCS-assisted mode was evaluated against a conventional schedule-driven manual-coordination baseline. The baseline follows the initial project schedule and does not include DT-enabled predictive synchronization, ontology-based impact reasoning, or automatic LogicConstraint generation. Disturbances and their consequences are incorporated only after manual confirmation and cross-phase communication, after which the affected activities are adjusted using the available precedence and resource information. Both modes use identical activities, resources, dependency relations, initial schedules, disturbance inputs, and management objectives; their principal difference is whether runtime changes are automatically propagated and transformed into adaptive decision constraints.

Collaboration effectiveness was evaluated using cross-phase task waiting time and shared-resource utilization. Let A_{cp} denote the set of downstream activities having at least one mandatory predecessor in a different phase. The release time is formalized as Equation (15) and the average waiting time is formalized as Equation (16).

$$t_j^{\text{release}} = \max_{i \in \text{Pred}(j)} t_i^{\text{finish}} \tag{15}$$

$$\overline{W}_{cp} = \frac{1}{|A_{cp}|} \sum_{j \in A_{cp}} \max\left(0, t_j^{\text{start}} - t_j^{\text{release}}\right) \tag{16}$$

This schedule-derived metric represents the handoff interval between the completion of mandatory predecessors and the actual start of a downstream cross-phase activity. Let R_s denote the critical resources shared across development phases. For each shared resource $r \in R_s$, its utilization is defined as Equation (17).

$$u_r = \frac{T_r^{\text{busy}}}{T_r^{\text{available}}}, \tag{17}$$

where T_r^{busy} and $T_r^{\text{available}}$ denote the busy time and available time of resource r, respectively. The mean utilization of the shared resources is then calculated as Equation (18).

$$\overline{u} = \frac{1}{|R_s|} \sum_{r \in R_s} u_r, \tag{18}$$

and the corresponding utilization variance is defined as Equation (19).

$$V_u = \frac{1}{|R_s|} \sum_{r \in R_s} (u_r - \overline{u})^2. \tag{19}$$

A lower utilization variance, interpreted together with mean utilization and total resource consumption, indicates more balanced shared-resource coordination. Table 10 compares the two modes under identical task, resource, dependency, initial-schedule, and disturbance settings.

Table 10. Collaboration-effectiveness comparison between the DTCS-assisted mode and the conventional manual-coordination baseline.

Metric	DTCS-Assisted Mode	Manual-Coordination Baseline
Total development cycle (h)	310	310
Total resource consumption (resource h)	828	908
Average cross-phase task waiting time (h)	6.67	8.00
Mean shared-resource utilization (%)	33.39	36.46
Shared-resource utilization variance	0.0404	0.0553

The DTCS-assisted mode reduced the average cross-phase task waiting time from 8.00 h to 6.67 h and decreased the shared-resource utilization variance from 0.0553 to 0.0404. Although its mean shared-resource utilization was slightly lower, the simultaneous reduction in total resource consumption from 908 to 828 resource h indicates reduced redundant resource occupation and more balanced shared-resource coordination under the examined case settings.

5.4.3. Adaptation Efficiency, Scalability, and Plan-Change Response

Adaptation timeliness measures the computational efficiency of the DSL in generating decision support. This metric serves as an indicator of the ability to maintain responsiveness as project complexity increases. To evaluate this, we recorded the computation time of the two strategies under their corresponding disturbance classes. The global optimization model s_{GA} was evaluated for predictive adaptation under significant parametric disturbances, whereas the low-latency heuristic model s_{Rule} was evaluated for locally bounded topology reconfiguration. To examine structural complexity rather than activity count alone, the baseline case was extended to synthetic 20- and 30-activity configurations with progressively more precedence relations, cross-phase dependencies, shared resource types, and bounded rework branches. The rework branches explicitly represent one-pass feedback paths from testing to design correction, manufacturing rework, and subsequent verification without introducing cyclic scheduling dependencies. Four plan-change instances were evaluated at each scale, covering activity delay, resource unavailability, emergency activity

insertion, and dependency change. The configurations and results are summarized in Table 11.

Table 11. Prototype-level scalability under increasing structural complexity.

Activities	Dependencies (Total/Cross-Phase)	Shared Resource Types	Bounded Rework Branches	s_{GA} Latency (ms)	s_{Rule} Latency (ms)	Feasible Responses
14	15/6	4	0	11	6	4/4
20	22/9	7	1	18	9	4/4
30	34/15	9	3	27	13	3/4

As the number of activities, cross-phase dependencies, shared resource types, and rework branches increased, both strategies exhibited higher decision latency but remained within the millisecond range for the tested configurations. The lower latency of s_{Rule} supports locally bounded adjustments, whereas s_{GA} remains computationally tractable for disturbances requiring broader task and resource coordination. These results demonstrate prototype-level scalability trends under increasing structural complexity rather than full industrial-scale validation.

In addition to decision latency, the robustness of collaborative adaptation is evaluated using the feasible plan-change response rate. For N_{change} disturbance instances, a response is regarded as successful when it is generated within the specified decision window, satisfies the precedence, resource-capacity, and inferred semantic constraints, and preserves workflow executability. The metric is defined as follows:

$$SR_{pc} = \frac{N_{\text{successful}}}{N_{\text{change}}} \times 100\%. \tag{20}$$

Across the 12 plan-change instances distributed over the three structural-complexity configurations, the framework generated 11 feasible responses, corresponding to an overall response rate of 91.7%. All four instances were successfully handled in the 14- and 20-activity cases, whereas three of four were successfully handled in the 30-activity case. The only unsuccessful instance involved an extended CT-scanner outage that prevented the workflow from remaining executable within the configured case horizon.

5.5. Discussion and Limitations

The instantiation suggests that the applicability of the framework is not limited to the specific context of turbofan blade development. By leveraging the MagicGrid methodology, the framework organizes complex development processes through a domain-reusable core semantic abstraction comprising generic activities, heterogeneous resources, disturbance events, and state or decision constraints. These generic concepts and the associated mapping–reasoning–decision pattern provide a reusable blueprint for DT-enabled cross-phase collaborative management. When transferred to other CoPS development contexts, however, domain-specific ontology concepts, reasoning rules, model parameters, and decision strategies need to be configured according to the characteristics of the target application.

However, the current study is subject to limitations regarding the fidelity of the prototype. First, the PVG currently operates through simulated data injection rather than raw high-frequency sensor streams. This setting supports scenario-level examination of the downstream semantic integration and decision-support logic while avoiding application-specific signal-processing requirements. Second, the configured decision models are simplified relative to industrial multi-criteria decision systems. The modular architecture never-

theless allows higher-fidelity perception modules and more advanced strategy libraries to be incorporated without changing the core framework logic. In addition, the 20% switching threshold is a case-configured engineering parameter used to demonstrate the strategy-selection mechanism rather than a universal industrial constant. Industrial deployment should recalibrate it using historical project performance, organizational control rules, risk tolerance, disturbance-propagation characteristics, and available computational resources.

Compared with the representative approaches summarized in Table 1, the contribution of the proposed framework lies not in the isolated use of MBSE, semantic modeling, or optimization, but in their operational integration for cross-phase collaboration. First, it advances fragmented DT functions toward an MBSE-traceable architecture that links collaboration requirements, functions, components, interfaces, and implementation elements. Second, it advances static information connectivity toward runtime semantic evolution through state synchronization, dependency-based impact propagation, and topology reconfiguration. Third, it advances semantic inference toward executable decision support by translating inferred impacts into inputs and constraints that can be processed by scheduling and optimization models.

The scalability results should likewise be interpreted as prototype-level trends obtained from synthetic configurations with up to 30 activities, increasing cross-phase dependencies, shared resources, and bounded rework branches. They demonstrate the framework's behavior under increasing structural complexity but do not constitute full validation of industrial-scale CoPS deployment.

These collaboration metrics are interpreted as case-level indicators: cross-phase waiting time is derived from schedule transitions, resource-utilization variance is interpreted together with mean utilization and total resource consumption, and the plan-change response rate reflects the 12 disturbance instances examined in the case-based evaluation rather than a statistical guarantee for all industrial projects.

6. Conclusions

To address the strong task coupling and stochastic disturbances inherent in CoPS development, this study proposed and evaluated a semantic-enhanced DT framework for cross-phase collaborative management. By integrating the MagicGrid methodology with a collaboration-oriented SysML profile, the framework establishes an MBSE-traceable architecture connecting collaboration requirements, system functions, interfaces, semantic models, and implementation elements. The knowledge-driven mechanism further connects runtime disturbance mapping, ontology synchronization, dependency-based impact reasoning, LogicConstraint generation, strategy selection, and result delivery through the PVG–SRL–DSL–SIL closed loop. Theoretically, this capability chain provides a feasible engineering pathway for advancing DTs from physical-state perception toward semantic reasoning and decision support: the PVG captures and maps runtime changes, the SRL interprets their semantic implications, the DSL converts inferred impacts into executable decision boundaries, and the SIL supports human-interpretable confirmation. It therefore identifies semantic representation, relation-aware reasoning, and constraint generation as important intermediate capabilities between perceptive and cognitive DTs. The framework also contributes to Model-Based Collaborative Engineering by extending MBSE models beyond design-time description and verification, enabling typed model elements, semantic relations, and executable constraints to support runtime cross-phase collaboration rather than relying solely on document exchange and manual interpretation.

Despite these contributions, the current implementation remains a case-based prototype with several limitations. The PVG relies on simulated data injection rather than raw high-frequency signals from industrial assets, and the configured decision models

are simplified relative to large-scale multi-criteria engineering applications. Moreover, the framework currently demonstrates semantic reasoning and constraint-driven decision support but does not yet provide continuous online learning, autonomous knowledge evolution, or fully automated decision execution; the generated results still require human confirmation. Accordingly, the study demonstrates a reasoning-enabled pathway toward cognitive DT capabilities rather than a fully autonomous cognitive twin or a universally applicable DT maturity model.

Future work will integrate higher-fidelity edge interfaces and industrial data streams, investigate online knowledge updating and adaptive decision agents, and employ distributed computing to support larger and more strongly coupled CoPS scenarios. Further research will also develop capability-assessment criteria for evaluating the progressive evolution from perceptive and semantic DT functions toward cognitive and autonomous collaborative capabilities.

Author Contributions: Conceptualization, Z.X. and T.L.; methodology, Z.X. and M.L.; software, Z.X.; validation, Z.X. and T.L.; formal analysis, Z.X. and M.L.; investigation, Z.X. and M.L.; resources, K.Y.; data curation, Z.X.; writing—original draft preparation, Z.X.; writing—review and editing, Z.X., M.L., T.L., G.S., and J.J.; visualization, Z.X. and M.L.; supervision, K.Y.; project administration, K.Y.; funding acquisition, K.Y. and J.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under grants 72231011 and 72431011.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Huang, Y.; Wang, G.; Wang, R.; Peng, T.; Li, H.; Yan, Y. A case-based knowledge graph with reinforcement learning for intelligent design approach of complex product. *J. Eng. Des.* **2025**, *36*, 1451–1478. [CrossRef]
2. Zhang, N.; Gao, W.; Li, Y.; Wang, Q. Complex product configuration updating oriented by changing customer requirements. *J. Eng. Des.* **2023**, *34*, 522–560. [CrossRef]
3. Zhang, Y.; Zhu, J.; Ge, C.; Huang, D.; Tong, H. Multi-Stage Incentive Contract Design of Aviation Complex Product Multi-Task Delivery Strategy. *Comput. Ind. Eng.* **2024**, *193*, 110297. [CrossRef]
4. Reim, W.; Andersson, E.; Eckerwall, K. Enabling Collaboration on Digital Platforms: A Study of Digital Twins. *Int. J. Prod. Res.* **2023**, *61*, 3926–3942. [CrossRef]
5. Bitencourt, J.; Wooley, A.; Harris, G. Verification and Validation of Digital Twins: A Systematic Literature Review for Manufacturing Applications. *Int. J. Prod. Res.* **2025**, *63*, 342–370. [CrossRef]
6. Wang, J.; Ye, L.; Gao, R.X.; Li, C.; Zhang, L. Digital Twin for Rotating Machinery Fault Diagnosis in Smart Manufacturing. *Int. J. Prod. Res.* **2019**, *57*, 3920–3934. [CrossRef]
7. Zhang, C.; Tao, F.; Qi, Q.; Cheng, Y.; Cheng, J.; Wang, B.; Nee, A.Y.C. Digital Twin-Based Shop-Floor Reconfiguration Design for Uncertainty Management. *Int. J. Prod. Res.* **2025**, 1–25. [CrossRef]
8. Liu, J.; Liu, J.; Zhuang, C.; Liu, Z.; Miao, T. Construction Method of Shop-Floor Digital Twin Based on MBSE. *J. Manuf. Syst.* **2021**, *60*, 93–118. [CrossRef]
9. Kuelper, N.; Jeyaraj, A.K.; Liscouët-Hanke, S.; Thielecke, F. Integration of a model-based systems engineering framework with safety assessment for early design phases: A case study for hydrogen-based aircraft fuel system architecting. *Results Eng.* **2025**, *25*, 104249. [CrossRef]
10. Zhang, Y.; Yu, X.; Liu, X.; Guo, J.; Zhang, W.; Zhang, C. Research on Model-Based Systems Engineering Approach for Information Technology Standard System Planning. *Systems* **2026**, *14*, 380. [CrossRef]

11. Mirabella, C.; Tuccillo, M.; Della Vecchia, P. Evaluating MBSE Approaches and Tools for Aircraft Design and Certification: A Comparative Perspective. *Systems* **2026**, *14*, 482. [CrossRef]
12. Monnier, L.; Shao, G.; Foufou, S. A Methodology for Digital Twins of Product Lifecycle Supported by Digital Thread. In Proceedings of the ASME 2022 International Mechanical Engineering Congress and Exposition (IMECE), Columbus, OH, USA, 30 October–3 November 2022.
13. Mingorance, R.; Crespo Pereira, D.; Uribetxebarria, J.; Leturiondo, U. A methodology leveraging digital twins to enhance the operational strategy of manufacturing plants in unexpected scenarios. *Results Eng.* **2025**, *27*, 106761. [CrossRef]
14. Grieves, M.; Vickers, J. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In *Transdisciplinary Perspectives on Complex Systems*; Kahlen, J., Flumerfelt, S., Alves, A., Eds.; Springer: Cham, Switzerland, 2017; pp. 85–113.
15. *ISO 23247-1:2021*; ISO Automation Systems and Integration—Digital Twin Framework for Manufacturing—Part 1: Overview and General Principles. International Organization for Standardization: Geneva, Switzerland, 2021.
16. Grieves, M. *Digital Twin: Manufacturing Excellence Through Virtual Factory Replication*; White Paper; Florida Institute of Technology: Melbourne, FL, USA, 2014.
17. Damjanovic-Behrendt, V.; Behrendt, W. An Open Source Approach to the Design and Implementation of Digital Twins for Smart Manufacturing. *Int. J. Comput. Integr. Manuf.* **2019**, *32*, 366–384. [CrossRef]
18. Lu, Y.; Zhao, Z.; Wang, W.; Zhang, K. Digital Twin Product Lifecycle System Dedicated to the Constant Velocity Joint. *Comput. Electr. Eng.* **2021**, *93*, 107264. [CrossRef]
19. Tao, F.; Sui, F.; Liu, A.; Qi, Q.; Zhang, M.; Song, B.; Guo, Z.; Lu, S.C.-Y.; Nee, A.Y.C. Digital Twin-Driven Product Design Framework. *Int. J. Prod. Res.* **2019**, *57*, 3935–3953. [CrossRef]
20. Li, H.; Tao, F.; Wang, H.; Song, W.; Zhang, Z.; Fan, B.; Wu, C.; Li, Y.; Li, L.; Wen, X.; et al. Integration Framework and Key Technologies of Complex Product Design-Manufacturing Based on Digital Twin. *Comput. Integr. Manuf. Syst.* **2019**, *25*, 1320–1336. (In Chinese) [CrossRef]
21. Tao, F.; Zhang, M.; Cheng, J.; Qi, Q. Digital Twin Workshop: A New Paradigm for Future Workshop. *Comput. Integr. Manuf. Syst.* **2017**, *23*, 1–9. (In Chinese) [CrossRef]
22. Fang, Y.; Peng, C.; Lou, P.; Zhou, Z.; Hu, J.; Yan, J. Digital-Twin-Based Job Shop Scheduling Toward Smart Manufacturing. *IEEE Trans. Ind. Inform.* **2019**, *15*, 6425–6435. [CrossRef]
23. Quan, Z.; Wang, Y.; Liu, X.; Ji, Z. Multi-Objective Evolutionary Scheduling Based on Collaborative Virtual Workflow Model and Adaptive Rules for Flexible Production Process with Operation Reworking. *Comput. Ind. Eng.* **2024**, *187*, 109848. [CrossRef]
24. De Giacomo, G.; Favorito, M.; Leotta, F.; Mecella, M.; Silo, L. Digital Twins Composition in Smart Manufacturing via Markov Decision Processes. *Comput. Ind.* **2023**, *149*, 103916. [CrossRef]
25. Li, Y.; Tao, Z.; Wang, L.; Du, B.; Guo, J.; Pang, S. Digital Twin-Based Job Shop Anomaly Detection and Dynamic Scheduling. *Robot. Comput.-Integr. Manuf.* **2023**, *79*, 102443. [CrossRef]
26. Yan, S.; Zhang, F.; Pei, A.; Guo, X.; Wu, H. Research on intelligent prediction model of hydraulic support load based on digital twin and multi-agent framework. *Results Eng.* **2025**, *28*, 107182. [CrossRef]
27. Meng, Z.; Zhao, S.; Chen, H.; Hu, M.; Tang, Y.; Song, Y. The Vehicle Testing Based on Digital Twins Theory for Autonomous Vehicles. *IEEE J. Radio Freq. Identif.* **2022**, *6*, 710–714. [CrossRef]
28. Wang, B.; Ke, X.; Song, Z.; Du, K.; Bi, X.; Hao, P.; Zhou, C. An Improved Strain Field Reconstruction Method Based on Digital Twin for Test Monitoring. *Exp. Mech.* **2024**, *64*, 519–537. [CrossRef]
29. Deuter, A.; Imort, S. Product Lifecycle Management with the Asset Administration Shell. *Computers* **2021**, *10*, 84. [CrossRef]
30. Shin, M.; Jung, J.-Y. An Edge AI System Framework Based on the Asset Administration Shell Standard. *Systems* **2026**, *14*, 205. [CrossRef]
31. Duverger, E.; Aubry, A.; Arista, R.; Levrat, E. An ontology-based digital thread framework to support early concurrent engineering in the aerospace industry. *J. Ind. Inf. Integr.* **2025**, *48*, 100984. [CrossRef]
32. Zhang, Q.; Liu, J.; Chen, X. A Literature Review of the Digital Thread: Definition, Key Technologies, and Applications. *Systems* **2024**, *12*, 70. [CrossRef]
33. Kosse, S.; Betker, V.; Hagedorn, P.; König, M.; Schmidt, T. A Semantic Digital Twin for the Dynamic Scheduling of Industry 4.0-Based Production of Precast Concrete Elements. *Adv. Eng. Inform.* **2024**, *62*, 102677. [CrossRef]
34. *INCOSE Systems Engineering Vision 2020*; Vision Document; INCOSE: San Diego, CA, USA, 2009.
35. Zhang, Y.; Zhao, Y.; Li, Z.; He, W.; Liu, Y. Research on the Digital Twin System for Rotation Construction Monitoring of Cable-Stayed Bridge Based on MBSE. *Buildings* **2025**, *15*, 1492. [CrossRef]
36. Ramos, A.L.; Ferreira, J.V.; Barceló, J. Model-Based Systems Engineering: An Emerging Approach for Modern Systems. *IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.)* **2012**, *42*, 101–111. [CrossRef]
37. Madni, A.M.; Purohit, S. Economic Analysis of Model-Based Systems Engineering. *Systems* **2019**, *7*, 12. [CrossRef]
38. Delligatti, L. *SysML Distilled: A Brief Guide to the Systems Modeling Language*; Addison-Wesley: Upper Saddle River, NJ, USA, 2014.

39. Ma, J.; Wang, G.; Lu, J.; Zheng, X.; Yan, Y. Application of Multi Architecture Modelling Method in Intelligent Electric-Vehicle Design. *Int. J. Prod. Res.* **2025**, *63*, 5493–5511. [CrossRef]
40. Aleksandraviciene, A.; Strolia, Z.; Jankauskas, O. Application of SysMLv1 vs SysMLv2 in the Scope of the MagicGrid Framework. *INCOSE Int. Symp.* **2024**, *34*, 2087–2109. [CrossRef]
41. Lutfi, M.; Valerdi, R. Integration of SysML and Virtual Reality Environment: A Ground Based Telescope System Example. *Systems* **2023**, *11*, 189. [CrossRef]
42. Madni, A.M.; Madni, C.C.; Lucero, S.D. Leveraging Digital Twin Technology in Model-Based Systems Engineering. *Systems* **2019**, *7*, 7. [CrossRef]
43. Gregory, C.; Mbolamananamalala, R.; Rabah, S.; Chapurlat, V. Model Based Systems Engineering Applied to Digital Twin Engineering: Why and How To? *IFAC-PapersOnLine* **2024**, *58*, 157–162. [CrossRef]
44. Zhang, X.; Wu, B.; Zhang, X.; Duan, J.; Wan, C.; Hu, Y. An Effective MBSE Approach for Constructing Industrial Robot Digital Twin System. *Robot. Comput.-Integr. Manuf.* **2023**, *80*, 102455. [CrossRef]
45. Bouhali, I.; Idasiak, V.; Martinez, J.; Mhenni, F.; Choley, J.-Y.; Palladino, L.; Kratz, F. A Collaboration Framework Using Digital Twin for Dynamic Simulation and Requirements Verification Based on MBSE and the MIC Concept. In Proceedings of the 2024 IEEE International Systems Conference (SysCon), Montreal, QC, Canada, 15–18 April 2024; pp. 1–8.
46. Mahboob, A.; Hussein, O.; Willrodt, L.; Raaj, V.R.T.R.A.; Takim, M.B.; Spaans, A.; Hasib, H.; Birkenfeld, K. C.Pulse: An Industrial Demonstrator for a Digital Twin Powered by MBSE for Achieving Digital Continuity during the Complete Development Process. In Proceedings of the 60th Ilmenau Scientific Colloquium, Ilmenau, Germany, 4–8 September 2023.
47. Ramirez, C.A.; Agrawal, P.; Thompson, A.E. An Approach Integrating Model-Based Systems Engineering, IoT, and Digital Twin for the Design of Electric Unmanned Autonomous Vehicles. *Systems* **2025**, *13*, 73. [CrossRef]
48. Cao, Y.; Liu, Y.; Wang, H.; Zhao, J.; Ye, X. Ontology-based model-driven design of distributed control applications in manufacturing systems. *J. Eng. Des.* **2019**, *30*, 523–562. [CrossRef]
49. Banerjee, A.; Choppella, V. A Knowledge-Driven Approach for Dynamic Reconfiguration of Control Design in Internet of Things and Cyber–Physical Systems. *IEEE Internet Things J.* **2025**, *12*, 5615–5641. [CrossRef]
50. Lin, Y.-J.; Hsieh, Y.-Y.; Huang, C.-Y. Ontology-based Manufacturing Control Systems (MCS). In Proceedings of the 25th International Conference on Production Research Manufacturing Innovation: Cyber Physical Manufacturing, Chicago, IL, USA, 9–14 August 2019; pp. 1906–1912.
51. Hoebert, T.; Lepuschitz, W.; Vincze, M.; Merdan, M. Knowledge-driven framework for industrial robotic systems. *J. Intell. Manuf.* **2023**, *34*, 771–788. [CrossRef]
52. Ren, S.; Gao, H.; Zhao, X.; Wang, J.; Shi, H.; Wang, C. A model and knowledge-driven reliability optimisation approach for complex systems. *J. Eng. Des.* **2025**, 1–38. [CrossRef]
53. Dong, M.; Wang, G.; Lu, J.; Wu, S.; Ma, J. Ontology formalism and semantic rules supporting traceability management in model-based systems engineering. *J. Eng. Des.* **2025**, *37*, 1255–1280. [CrossRef]
54. Martínez-Arellano, G.; Niewiadomski, K.; Mo, F.; Elshafei, B.; Chaplin, J.C.; McFarlane, D.; Ratchev, S. Enabling Coordinated Elastic Responses of Manufacturing Systems Through Semantic Modelling. *IFAC-PapersOnLine* **2023**, *56*, 7402–7407. [CrossRef]
55. Su, C.; Han, Y.; Tang, X.; Jiang, Q.; Wang, T.; He, Q. Knowledge-Based Digital Twin System: Using a Knowledge-Driven Approach for Manufacturing Process Modeling. *Comput. Ind.* **2024**, *159–160*, 104101. [CrossRef]
56. Su, C.; Jiang, Q.; Han, Y.; Wang, T.; He, Q. Knowledge Graph-Driven Decision Support for Manufacturing Process: A Graph Neural Network-Based Knowledge Reasoning Approach. *Adv. Eng. Inform.* **2025**, *64*, 103098. [CrossRef]
57. Juarez Juarez, M.G.; Giret, A.; Botti, V. Semantic and Modular Orchestration of AI-Driven Digital Twins for Industrial Interoperability and Optimization. *J. Ind. Inf. Integr.* **2025**, *48*, 100959. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.