Process-Driven RDF Knowledge Graphs as Semantic Digital Twins for Industry 5.0

Inauguraldissertation zur Erlangung des akademischen
Grades eines Doktors der Naturwissenschaften der
Universität Mannheim

vorgelegt von

M.Sc. Franz Krause

aus Greiz

Mannheim, 2026

Dekan: Professor Dr. Claus Hertling, Universität Mannheim
Referent: Professor Dr. Heiko Paulheim, Universität Mannheim
Korreferent: Priv.-Doz. Dr. Bernhard Moser, Universität Linz

Tag der mündlichen Prüfung: 22.06.2026

Abstract

Industry 5.0 envisions human-centric, resilient, and adaptive industrial systems in which processes are orchestrated across humans, machines, and intelligent services. Realizing such systems requires representations that capture system behavior over time while also providing semantic grounding, interpretability, and support for feedback and adaptation. This thesis investigates how process-driven knowledge modeling can achieve these goals by integrating process orchestration, semantic technologies, and graph-based learning.

Drawing on insights from the EU-funded Teaming.AI project, five key requirements for knowledge modeling in Industry 5.0 are derived: explicit coordination, temporal traceability, semantic grounding, integration of process and domain knowledge, and adaptive feedback. These requirements motivate a layered architecture in which process models define coordination, execution traces capture system dynamics, and semantic knowledge graphs provide conceptual grounding. Their integration yields semantically enriched, time-resolved representations of collaborative industrial systems in the form of semantic digital shadows.

To realize this integration, the sBPMN framework is introduced, providing an ontology that captures the core components of Business Process Model and Notation (BPMN). In addition to the ontology, sBPMN includes a software implementation for transforming BPMN workflows and their executions into dynamic knowledge graph trajectories, realized as a prototype within the Teaming.AI project. This enables the systematic mapping of collaborative processes into Resource Description Framework (RDF)-based representations, linking heterogeneous domain knowledge through their participation in shared process instances and thereby connecting previously isolated information across diverse application domains, including Industry 5.0.

Towards the realization of semantic digital twins, the thesis explores data-driven methods for enabling feedback and predictive capabilities. Graph Neural Networks (GNNs) are extended with explainable AI techniques to increase the transparency of downstream predictions on relational data via controlled graph traversals. In addition, a novel embedding paradigm based on local self-reconstruction enables adaptive and robust learning on incomplete and evolving graph structures without retraining, by reconstructing embeddings on-the-fly from local context.

The proposed GNN extensions are evaluated on generalized benchmark settings and demonstrate improved transparency, adaptability, and predictive performance, reflecting key real-world requirements. While not yet deployed in industrial production systems, they are designed to transfer across diverse domains. Overall, this thesis establishes a foundation for integrating process-centric and semantic representations with explainable and adaptive learning, advancing semantic digital shadows toward semantic digital twins for human-centered industrial systems.

Zusammenfassung

Industrie 5.0 beschreibt die Vision menschenzentrierter, resilienter und adaptiver industrieller Systeme, in denen Prozesse über Menschen, Maschinen und intelligente Dienste hinweg stattfinden. Die Realisierung solcher Systeme erfordert Repräsentationen, die das Systemverhalten zeitlich abbilden und zugleich semantische Fundierung, Interpretierbarkeit sowie Rückkopplung und Anpassungsfähigkeit unterstützen. Diese Dissertation untersucht, wie prozessgetriebene Wissensmodellierung durch die Integration von Prozessorchestrierung, semantischen Technologien und graphbasierten Lernverfahren zur Umsetzung dieser Anforderungen beitragen kann.

Auf Basis von Erkenntnissen aus dem EU-geförderten Teaming.AI-Projekt werden fünf zentrale Anforderungen an die Wissensmodellierung in Industrie 5.0 abgeleitet: explizite Koordination, zeitliche Nachvollziehbarkeit, semantische Fundierung, Integration von Prozess- und Domänenwissen sowie adaptives Feedback. Diese Anforderungen motivieren eine geschichtete Architektur, in der Prozessmodelle die Koordination strukturieren, Ausführungsspuren die Systemdynamik erfassen und semantische Wissensgraphen die konzeptuelle Grundlage bereitstellen. Ihre Integration führt zu semantisch angereicherten, zeitaufgelösten Repräsentationen kollaborativen Systemverhaltens in Form semantischer digitaler Schatten.

Zur Umsetzung dieser Integration wird das sBPMN-Framework eingeführt. Es stellt eine Ontologie zur Modellierung der zentralen Konzepte der Business Process Model and Notation (BPMN) sowie eine Softwareimplementierung bereit, die BPMN-Workflows und deren Ausführungen in dynamische Wissensgraphen überführt und im Rahmen von Teaming.AI prototypisch realisiert wurde. Dies ermöglicht die systematische Abbildung kollaborativer Prozesse in RDF-basierten Repräsentationen, wodurch heterogenes Domänenwissen über gemeinsame Prozessinstanzen hinweg verknüpft und zuvor isolierte Informationen über verschiedene Anwendungsdomänen hinweg – einschließlich Industrie 5.0 – miteinander verbunden werden.

Hinsichtlich der Realisierung semantischer digitaler Zwillinge untersucht die vorliegende Arbeit datengetriebene Methoden zur Unterstützung von Rückkopplung und prädiktiven Fähigkeiten. Graph-Neuronale Netze (GNNs) werden um erklärbare KI-Ansätze erweitert, um die Transparenz von Vorhersagen auf relationalen Daten durch kontrollierte Graph-Traversierungen zu erhöhen. Zusätzlich wird ein neuartiges Embedding-Paradigma auf Basis lokaler Selbstrekonstruktion eingeführt, das ein adaptives und robustes Lernen auf unvollständigen und wandelbaren Graphstrukturen ohne erneutes Training ermöglicht, indem Einbettungen dynamisch aus lokalem Kontext rekonstruiert werden.

Die vorgeschlagenen Erweiterungen für GNNs werden auf allgemeinen Benchmark-Datensätzen evaluiert und zeigen Verbesserungen hinsichtlich Transparenz, Anpassungsfähigkeit und Vorher-

sageleistung, wodurch zentrale Anforderungen realer Anwendungsszenarien adressiert werden. Auch wenn bislang keine Integration in industrielle Produktionssysteme erfolgt ist, sind die entwickelten Ansätze auf eine Übertragbarkeit auf unterschiedliche Anwendungsdomänen ausgelegt. Insgesamt schafft diese Arbeit eine Grundlage für die Verbindung prozessorientierter und semantischer Repräsentationen mit erklärbaren und adaptiven Lernverfahren und trägt dazu bei, semantische digitale Schatten in Richtung semantischer digitaler Zwillinge für menschenzentrierte industrielle Systeme weiterzuentwickeln.

Acknowledgements

Can I come out now? Is it really over? I suppose it is, and therefore it is time to spend a few words on the people who accompanied me throughout this journey at DWS. It was an exciting, interesting, and, let us be honest, often stressful time. Would I do it again? Absolutely. Would I do things differently? One hundred percent. But I suppose that is part of the PhD process. It is not only about research, but also about growing as a person and learning about yourself. My research group provided the perfect environment for that.

I would like to thank my supervisor Heiko Paulheim. Even though I had already decided after my M.Sc. that I would not pursue a PhD, I had one exception in mind: if it involved knowledge graphs and manufacturing, I would at least consider it. After our first call, I immediately knew that I wanted to do this, and sometimes first impressions are right. It has been a pleasure to be guided by you over the past years and to benefit from your advice and perspective. I would like to thank my colleagues at DWS, especially Martin, Andreea, Rita, and Sven, listed here in the order of their appearance in "MARS", who accompanied me throughout most of this journey.

Next, I would like to thank two people who played an important role in shaping my path. First, Markus Schiemann, who decided in 2017 that it might be a good idea to let me explore this thing called *knowledge graphs*, which his then-supervisor wanted to implement during my internship at Daimler. In hindsight, that decision seems to have worked out quite well. Second, Tobias Weller, who helped me settle into PhD life when I started at DWS and ultimately gave me the opportunity to take on t(his) position.

I would also like to thank the entire Teaming.AI consortium, particularly Bernhard Moser, who was a great project lead during the time we worked together and who also gave me the opportunity to visit SCCH in 2024 for a research stay. At the end of this journey, we meet again for my defense. There are many other people in the consortium whom I would like to thank for the time we spent together, the countless online meetings, but especially the enjoyable moments during our in-person gatherings. Among them, I would like to thank Pedro and Julen, who showed that European research projects can also lead to genuine European friendships.

Finally, I would like to thank my family and friends, who were always there for me, and give a few special acknowledgements. To my parents, who supported me throughout this journey and helped me find calm when I needed a break. To Coni, whose support during the final phase of this thesis had a greater impact than he may realize. To Alex, even though our joint publication has not worked out yet, sharing our experiences was always helpful to me. Most importantly, my deepest thanks go to my soon-to-be wife Tatjana, who faced many of the same challenges and reminded me that love is the most powerful way to overcome doubts, worries, and fears.

Contents

List of Figures **xiii**

List of Tables **xv**

List of Listings **xvii**

List of Acronyms **xx**

Introduction **1**
Contributions 3
Thesis Outline 4
Publications 5

I Background & Motivation **7**

1 Industry 5.0: Goals and Challenges **9**
1.1 Technological Foundations of Industry 4.0 10
1.2 Digital Representations of Industrial Systems 12
1.3 The Human Role within Industry 4.0 14
1.4 Principles of Industry 5.0 14
1.4.1 Human-Centricity 14
1.4.2 Sustainability 15
1.4.3 Resilience 15

2 Layered Knowledge Modeling with BPMN and RDF **17**
2.1 Layered Knowledge Modeling 17
2.2 Layered Knowledge Modeling with BPMN 20
2.2.1 BPMN as a Modeling Language 20
2.2.2 BPMN Process Models as Domain Model 22
2.2.3 BPMN Process Executions as Domain Population 24
2.3 Layered Knowledge Modeling with RDF 25
2.3.1 The RDF Data Model 25
2.3.2 RDF Ontology Languages as a Modeling Language 28

2.3.3 RDF Ontologies as Domain Model 31
2.3.4 RDF Triple Sets as Domain Population 34

3 RDF Knowledge Graphs 37
3.1 Graph-Theoretic Foundations 37
3.2 From RDF Triple Sets to RDF Knowledge Graphs 40
3.3 Temporal Extensions of RDF Knowledge Graphs 41
3.3.1 Time-Awareness in RDF Knowledge Graphs 42
3.3.2 Temporal RDF Knowledge Graphs 44
3.3.3 Temporal Trajectories of RDF Knowledge Graphs 45
3.3.4 Integrating Local and Global Temporality 46

4 Graph Representation Learning 49
4.1 Non-Parametric Graph Embedding Models 52
4.2 Parametric Graph Embedding Models 54
4.2.1 Factorization-Based Training Objectives 56
4.2.2 Transformation-Based Training Objectives 57
4.2.3 Random-Walk-Based Training Objectives 58
4.2.4 Contrastive Training Objectives 60
4.3 Graph Neural Networks 61

II Knowledge Modeling in Industry 5.0 67

5 Knowledge Modeling Requirements in Industry 5.0 69
5.1 The Teaming.AI Project 69
5.2 Key Challenges and Architectural Requirements 71
5.3 Semantic Digital Representations as Enablers of Industry 5.0 73

6 From Digital Shadows to Semantic Digital Twins 75
6.1 Digital Shadows via BPMN Knowledge Bases 75
6.1.1 From Industrial Control Logic to Collaborative Process Orchestration . 75
6.1.2 Design and Deployment Lifecycle 76
6.1.3 Execution and Workflow Engines in Industrial Environments 76
6.1.4 BPMN Knowledge Base Trajectories as Digital Shadows 77
6.2 Semantic Domain Knowledge via RDF Knowledge Bases 77
6.2.1 Ontology Modeling and Integration Paradigms 78
6.2.2 RDF Management Systems and Deployment Architectures 79
6.2.3 Population-Level Characteristics and Dynamic Limitations 80
6.3 From Orthogonality to Semantic Digital Shadows and Twins 81

III Process-Driven RDF Knowledge Graphs as Semantic Digital Shadows 83

7 From BPMN Knowledge Bases to RDF Knowledge Bases 85
7.1 Related Work . . . 86
7.2 Abstraction Levels in BPMN-to-RDF Representations . . . 87
7.3 Mapping BPMN Knowledge Bases to RDF Knowledge Bases . . . 88
7.3.1 Representing Process Models as Knowledge Graph Instances . . . 89
7.3.2 Representing Process Models as Ontology Concepts . . . 90
7.4 Implications for Process-Driven Knowledge Graphs . . . 91

8 The sBPMN Framework 93
8.1 The sBPMN Ontology . . . 95
8.1.1 Ontology Extraction from the BPMN Schema . . . 96
8.1.2 Constraint Modeling with SHACL . . . 99
8.2 sBPMN System Architecture and Implementation . . . 99
8.2.1 H2 Execution Log Schema . . . 100
8.2.2 Virtualization and Knowledge Graph Construction . . . 101
8.2.3 Dual-Endpoint Architecture . . . 103
8.2.4 SPARQL Query Federation . . . 103
8.3 Use Case: Human-AI Teaming in Industry . . . 103
8.3.1 External Knowledge Integration . . . 105
8.4 Discussion . . . 106

IV Enabling Semantic Digital Twins via Graph Learning 107

9 Explaining RGNN Predictions on RDF Graphs via LRP 111
9.1 Layer-Wise Relevance Propagation . . . 113
9.2 Related Work . . . 114
9.3 Methodology . . . 116
9.3.1 Applying LRP to Single-Layer RGNNs . . . 117
9.3.2 Applying LRP to Multi-Layer RGNNs . . . 118
9.4 Experimental Evaluation . . . 119
9.4.1 Edge/Stub-Level RGNN Explanations . . . 121
9.4.2 Node-Level RGNN Explanations . . . 122
9.4.3 Path-Level RGNN Explanations . . . 122
9.4.4 Embedding-Level RGNN Explanations . . . 122
9.4.5 Relevance Distribution Across Computation Graphs . . . 123
9.5 Discussion . . . 123

10 Reconstruction-Based Graph Embedding Models 125
10.1 Real-World Requirements for Graph Embeddings . . . 126
10.2 Methodology . . . 128
10.3 Experimental Evaluation . . . 131

10.3.1 Embedding Undirected and Homogeneous Graphs 132
10.3.2 Embedding Directed and Relational Graphs 133
10.3.3 Adaptability to Graph Updates 134
10.3.4 Separability in the Embedding Space 136
10.4 Discussion 138

V Conclusion and Outlook **139**

Summary 141
Limitations and Future Work 142

Bibliography **145**

List of Figures

1.1 ISA-95 hierarchy of industrial systems. 11
1.2 Industry 4.0 data integration and AI-enabled feedback architecture. 12

2.1 Layered knowledge modeling and knowledge base perspective. 18
2.2 Core categories of BPMN modeling constructs. 21
2.3 Sample BPMN process for AI-assisted quality control in manufacturing. . . . 23
2.4 Bootstrapped organization of the RDF/RDFS terminology. 29
2.5 Illustration of an RDF knowledge base. 35

4.1 Schematic representation of a graph embedding constructor. 54
4.2 Computation graph of a GNN during a forward pass. 64

6.1 Comparison of vertical vs. horizontal ontology integration. 81

7.1 Abstraction levels in BPMN-to-RDF representations. 88
7.2 Minimal BPMN process model of a human-AI collaboration pattern observed in the Teaming.AI use cases, covering several industrial components. 88
7.3 Knowledge graph encoding of process model and execution. 89

8.1 BPMN diagram with labeled elements and sequence flows. 94
8.2 sBPMN ontology: mapping BPMN XML schema to OWL and SHACL. . . . 95
8.3 Modular ontology architecture for sBPMN process modeling. 98
8.4 sBPMN system architecture integrating BPMN engine, RDF mapping, and SPARQL-based access. 101
8.5 Teaming.AI BPMN workflow for human–AI process diagnostics. 104

9.1 Semantic predictive loop: explaining RGNN predictions via LRP. 112
9.2 Dense layer forward pass and LRP-based relevance propagation. 114
9.3 Initializing LRP for RGNN outputs with relevance propagation. 116
9.4 Relevance redistribution in a single-layer RGNN to stubs and nodes. 117
9.5 Deletion-based LRP faithfulness for RGNNs. 121
9.6 Hop-distance distribution of most relevant nodes, edges, and paths in LRP-based RGNN explanations. 123

10.1 LoRE self-reconstruction via masked base embeddings. 129
10.2 Visual representation of δ_{LoRE} characteristics and training objectives. 130
10.3 PCA projections of AIFB embeddings for KGCL, RGAT, and LoRE. 137

List of Tables

5.1 Characteristics of digital representation layers in Industry 5.0 architectures. . . 74

9.1 GNN explanation methods: comparison of capabilities and properties. 115

10.1 Statistics of undirected and homogeneous graph embedding benchmarks. . . . 132
10.2 Classification accuracies (± std) on undirected and homogeneous benchmarks. 133
10.3 Statistics of directed and relational benchmarks after removing literals. 133
10.4 Classification accuracies (± std) on directed and relational benchmarks. 134
10.5 Classification accuracies (± std) before and after edge-level perturbations and embedding updates via neighborhood reconstruction. 135
10.6 Classification accuracies (± std) before and after node-level perturbations and embedding updates via neighborhood reconstruction. 136

List of Listings

2.1 Sample SQL domain model and population within a relational database. 19
2.2 Example RDF statements in TTL syntax. 27
2.3 RDFS-based description of the rdf:type property. 29
2.4 Excerpt from the OWL ontology describing owl:Class. 30
2.5 Sample declarations of domain classes and properties using OWL. 32
2.6 OWL someValuesFrom restriction for worker–organization relation. 33
2.7 SHACL minCount constraint for worker–organization relation. 34

3.1 Three ontological approaches to encoding the same temporal information. . . . 42
3.2 Statement-level temporal qualification using RDF reification. 43
3.3 Statement-level temporal qualification using RDF-star. 43

8.1 Excerpt of a BPMN 2.0 XML process definition. 94
8.2 OWL class for BPMN transaction with compensating method. 96
8.3 Sequence flow class restriction formulated in OWL. 97
8.4 SHACL default value for transaction method. 99
8.5 Camunda runtime properties. 102
8.6 Ontop mapping for Camunda logs. 102
8.7 SPARQL query retrieving process instances associated with a short-shot error. . 105

List of Acronyms

ABox Assertional Box.

AI Artificial Intelligence.

BPM Business Process Modeling.

BPMN Business Process Model and Notation.

CPPS Cyber-Physical Production System.

ERP Enterprise Resource Planning.

GAT Graph Attention Network.

GCN Graph Convolutional Network.

GNN Graph Neural Network.

HITL Human-In-The-Loop.

HMI Human-Machine Interface.

HOTL Human-On-The-Loop.

IoT Internet of Things.

IRI Internationalized Resource Identifier.

LRP Layer-Wise Relevance Propagation.

MES Manufacturing Execution System.

ML Machine Learning.

MQTT Message Queuing Telemetry Transport.

NN Neural Network.

NUNA Non-Unique Name Assumption.

OPC OLE for Process Control.

OPC UA OPC Unified Architecture.

OWA Open World Assumption.

OWL Web Ontology Language.

PCA Principal Component Analysis.

PLC Programmable Logic Controller.

RDF Resource Description Framework.

RDFS RDF Schema.

RGAT Relational Graph Attention Network.

RGCN Relational Graph Convolutional Network.

RGNN Relational Graph Neural Network.

SCADA Supervisory Control and Data Acquisition.

SHACL Shapes Constraint Language.

SPARQL SPARQL Protocol and RDF Query Language.

SWRL Semantic Web Rule Language.

TBox Terminological Box.

TTL Turtle.

URI Uniform Resource Identifier.

Introduction

Over the past decades, industrial production systems have undergone a series of paradigm shifts driven by increasing levels of automation, digitalization, and data availability. While earlier stages of industrial development primarily focused on efficiency, throughput, and cost optimization, the emergence of Industry 4.0 marked a decisive step towards highly automated and interconnected production environments. In this context, technologies such as Machine Learning (ML), industrial Internet of Things (IoT), and Cyber-Physical Production Systems (CPPSs) enabled the integration of physical processes with digital control and analytics [111, 139, 150, 221]. This transition was accompanied by an increasing delegation of decision-making and control tasks to Artificial Intelligence (AI)-driven components and autonomous system elements [169]. However, the strong emphasis on automation and optimization in Industry 4.0 has also revealed limitations, particularly with respect to system robustness, transparency, and the role of human expertise in complex and dynamic environments. Industry 5.0 therefore extends the Industry 4.0 vision by placing human-centricity, resilience, and sustainability at the core of industrial production systems [38, 256]. Achieving these goals requires digital infrastructures that not only integrate heterogeneous data sources, but also support interpretability, adaptability, and meaningful socio-technical collaboration [217]. The question of how collaborative scenarios can be explicitly coordinated and semantically enriched constitutes a central motivation of this thesis.

Another central question addressed in this thesis is whether and how Semantic Web technologies based on RDF, such as knowledge graphs and domain ontologies, can effectively support Industry 5.0 as semantic infrastructure. In current industrial practice, knowledge graphs and ontologies are primarily employed as static repositories of domain knowledge, serving purposes such as data integration, semantic standardization, and interoperability across heterogeneous information systems [22, 146]. They are typically constructed offline, curated by domain experts, and aligned with established ontologies in order to provide a shared conceptualization of the production domain that is both human-interpretable and machine-readable [105, 222]. While such implementations enable expressive querying and semantic enrichment of industrial data, their interpretability for both humans and machines is commonly exploited only in a descriptive or analytical manner. As a result, they are often treated as isolated artifacts that remain largely decoupled from the execution logic of production systems. Consequently, updates to the underlying processes, data streams, or system configurations are often reflected in the graph only with significant delay, if at all, limiting their ability to capture the dynamic nature of CPPSs. This, in turn, limits support for real-time decision-making and adaptive control within operational processes, such as those found in Industry 5.0 environments.

Building upon these observations, this work addresses the static nature of existing semantic infrastructures by enabling the systematic linking of decentralized knowledge fragments through process-driven semantic digital shadows. This is achieved through the integration of dynamic process executions, which are continuously mapped into a dynamic knowledge graph trajectory that captures the semantics of ongoing and past executions of industrial processes. This transformation shifts knowledge graphs from static repositories to operational, process-integrated representations of industrial system behavior. In contrast to traditional automation-centric designs, the underlying process modeling paradigms in this thesis are inherently human-centered, ensuring that the resulting semantic representations remain aligned with human decision-making, interpretability, and domain expertise.

The concept of process knowledge graphs as dynamic digital shadows also provides a natural foundation for advancing towards digital twins, which require not only the passive recording of process states but also the ability to simulate, predict, and reason about future system states [126, 221]. In the context of Industry 5.0, digital twins play a central role in enabling resilient and human-centric production systems by supporting adaptive decision-making, scenario exploration, and transparent interaction between operators and automated components [25, 38]. By semantically enriching process executions and linking them to existing domain knowledge, RDF-based representations offer a structured and interpretable source of information on which digital twin capabilities can be developed, bridging physical system operations with their human-centered process logic. In particular, formal domain ontologies enable logical entailment and constraint validation [39, 96, 115, 102], allowing new knowledge to be inferred and process configurations to be verified through symbolic reasoning mechanisms.

In parallel, advances in graph learning show that ML techniques can infer new knowledge from graph-structured data [42, 113], although most methods are designed for static graphs and lack support for evolving structures [233, 242]. To address this gap, this work introduces an adaptive framework that aligns with Industry 5.0 requirements and enables predictive modeling for dynamic graph representations, leveraging GNNs to handle process-based structural evolution without requiring complete retraining. Moreover, we show how GNNs can be combined with explainable AI approaches that attribute predictions to relevant graph fragments within the knowledge graph. This enables a bidirectional feedback loop in which predictive insights are not only generated but also contextualized, empowering human actors to understand and validate automated reasoning in complex industrial environments. While the proposed semantic integration and learning-based approaches are evaluated on generalized benchmark settings and controlled environments, they have not yet been deployed in operational industrial production systems, but are designed to be transferable to a wide range of process-driven domains by adhering to the industrial requirements outlined in this work.

This thesis was conducted in the context of the Teaming.AI research project, which received funding from the European Commission's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 957402. Although the Teaming.AI project provided an industrial setting for the investigation of knowledge graph technologies in production environments and informed the selection of use cases and practical constraints considered in this work, the methodological contributions and conclusions of the thesis remain self-contained.

Contributions

This thesis makes four key contributions at the intersection of process modeling, semantic technologies, and graph-based machine learning for dynamic industrial environments in the context of Industry 5.0. The contributions span conceptual, methodological, and technical advances and were developed and evaluated within the EU-funded Teaming.AI project.

(C1) Knowledge modeling requirements for Industry 5.0. This thesis derives five key requirements for knowledge modeling in Industry 5.0 based on empirical insights from the Teaming.AI project: (i) explicit process orchestration, (ii) temporal traceability, (iii) formal semantic grounding, (iv) integration of process and domain knowledge, and (v) adaptive feedback mechanisms. Building on these requirements, a layered architecture is introduced that combines process models and semantic knowledge bases into semantic digital shadows and, ultimately, semantic digital twins. A central insight is that process-centric representations (e.g., BPMN) and semantic domain knowledge (RDF) are complementary: processes provide coordination and dynamics, and semantic models provide meaning and interoperability, enabling system-spanning, semantically grounded representations of collaborative workflows.

(C2) The sBPMN framework for semantic process knowledge graphs. To operationalize this integration, the thesis introduces the sBPMN framework, which transforms BPMN-based process models and execution traces into RDF knowledge graphs. The framework consists of (i) an ontology covering the core concepts of the BPMN specification and (ii) a concrete software implementation that generates process knowledge graphs from workflow engine executions (e.g., Camunda). This enables the creation of semantic digital shadows that capture both process dynamics and domain semantics in a unified representation. The framework was deployed and validated within the final demonstrator of the Teaming.AI project, demonstrating its applicability in real-world industrial use cases.

(C3) Explainable graph learning on knowledge graphs. The thesis integrates GNNs for learning on RDF-based knowledge graphs with explainable AI techniques based on Layer-Wise Relevance Propagation (LRP). Existing approaches for graph learning are extended from unlabeled graphs to labeled RDF graphs, enabling predictions on semantically structured data. By combining relational GNNs with LRP, the approach provides explanations for predictions in terms of relevant subgraphs, thereby making automated reasoning transparent and interpretable. This contribution establishes a key mechanism for transparent feedback within semantic digital twins, allowing predictions to be understood, validated, and acted upon by human operators.

(C4) Local self-reconstruction for adaptive graph embeddings. To address the limitations of existing GNNs in dynamic settings, this thesis introduces local self-reconstruction as a novel graph embedding paradigm that enables adaptation to evolving graph structures without requiring retraining. Two implementations, NAVI and LORE, are proposed, differing in their assumptions regarding the availability of initial node embeddings. Both models support inductive learning on dynamically evolving knowledge graphs and achieve improved adaptability and predictive performance in downstream tasks compared to conventional GNNs. Experimental results on generalized benchmark settings show that new data can be incorporated efficiently while maintaining or even improving overall model performance in downstream tasks.

Thesis Outline

Taken together, the contributions C1–C4 establish a unified framework for semantic digital twins that combines process-centric modeling, semantic knowledge representation, and adaptive, explainable graph learning to support human-centered, resilient, and interpretable industrial systems. These contributions are developed through the following structure of this thesis.

Part I (Background & Motivation) introduces the conceptual and technological groundwork of the thesis. Chapter 1 contextualizes the evolution towards Industry 5.0, analyzing the limitations of Industry 4.0 with respect to coordination, transparency, and human integration. Chapter 2 introduces layered knowledge modeling using BPMN and RDF, establishing a unified perspective on process models and semantic representations. Chapter 3 formalizes RDF knowledge graphs and their temporal extensions as a foundation for representing dynamic systems. Chapter 4 provides an overview of graph representation learning, covering both classical embedding approaches and graph neural networks as a basis for learning from structured data.

Part II (Knowledge Modeling in Industry 5.0) derives requirements and conceptual foundations for semantic digital representations in industrial environments. Chapter 5 analyzes knowledge modeling requirements in Industry 5.0, including insights from the Teaming.AI project, and identifies key architectural challenges and requirements. Chapter 6 bridges digital shadows and digital twins by integrating process-centric representations with semantic domain knowledge, establishing the conceptual basis for semantic digital twins.

Part III (Process-Driven RDF Knowledge Graphs as Semantic Digital Shadows) develops the core representation framework of this thesis. Chapter 7 presents the transformation of BPMN knowledge bases into RDF knowledge bases, thereby enabling process-driven knowledge graphs. Chapter 8 introduces the sBPMN framework, including its ontology, constraint modeling using SHACL, system architecture, and a real-world use case for human–AI teaming.

Part IV (Enabling Semantic Digital Twins via Graph Learning) extends the representation framework with learning-based capabilities. Chapter 9 introduces methods for explaining relational GNN predictions using LRP, enabling interpretability of learned representations by propagating relevance to graph structures within the corresponding computation graphs. Chapter 10 presents reconstruction-based graph embedding models that support adaptation to dynamic graph updates and provide robust representations for evolving knowledge graphs.

Part V (Conclusion and Outlook) summarizes the contributions of this thesis and discusses implications, limitations, and directions for future research in the context of semantic digital twins and human-centered industrial systems.

This thesis is grounded in the industrial context of the Teaming.AI project, yet the proposed concept of process-driven semantic digital twins extends beyond this domain. It applies to settings in which process executions can be observed and enriched with semantic domain knowledge. By combining dynamic process knowledge from execution traces with static domain knowledge in semantic representations, the approach provides a general framework for consistent, interpretable, and adaptable system representations across diverse application domains.

Publications

The research presented in this thesis builds upon the following selection of peer-reviewed publications resulting from the work conducted in the course of this dissertation:

- Franz Krause, Tobias Weller, and Heiko Paulheim. *"On a Generalized Framework for Time-Aware Knowledge Graphs."* In: *Towards a Knowledge-Aware AI*, IOS Press, 2022. *(Best Research Paper Award)*

- Franz Krause, Heiko Paulheim, Elmar Kiesling, Kabul Kurniawan, Maria Leva, Hector Diego Estrada-Lugo, Gernot Stübl, Nazim Üre, Javier Dominguez, Maqbool Khan, Pedro Demolder, Hans Gaux, Bernhard Heinzl, Thomas Hoch, Jorge Martinez-Gil, Agastya Silvina, and Bernhard A. Moser. *"Managing human-AI collaborations within Industry 5.0 scenarios via knowledge graphs: key challenges and lessons learned."* In: *Frontiers in Artificial Intelligence*, 2024.

- Franz Krause, Kabul Kurniawan, Elmar Kiesling, Heiko Paulheim, and Axel Polleres. *"On the Representation of Dynamic BPMN Process Executions in Knowledge Graphs."* In: *Knowledge Graphs and Semantic Web*, Springer, 2023.

- Franz Krause, Heiko Paulheim, Amparo Vázquez, and Alejandro Espert. *"sBPMN: A Framework for Capturing Business Process Semantics in Dynamic Knowledge Graphs."* In: *Proceedings of the 13th International Conference on Knowledge Capture*, ACM, 2025.

- Franz Krause and Heiko Paulheim. *"Explaining RGNN Predictions on Knowledge Graphs via LRP."* In: *Proceedings of the 22nd International Conference on Semantic Systems (SEMANTiCS)*, IOS Press, 2026.

- Franz Krause. *"Dynamic Knowledge Graph Embeddings via Local Embedding Reconstructions."* In: *Proceedings of the ESWC 2022 Satellite Events*, Springer, 2022.

- Franz Krause and Heiko Paulheim. *"LoRE: Adaptive and Robust Graph Embeddings via Local Self-Reconstruction."* In: *Machine Learning and Knowledge Discovery in Databases. Research Track*, Springer, 2026.

PART I

Background & Motivation

Great things are not done by impulse,
but by a series of small things brought together.

Vincent van Gogh

This part introduces the background and preliminaries for the remainder of the thesis. We begin by outlining the principles and challenges of Industry 5.0, emphasizing human-centered collaboration, resilience, and sustainability, which motivate the need for advanced knowledge modeling in industrial environments. We adopt a layered perspective on knowledge modeling and consider two complementary formalisms for representing domain knowledge: BPMN for modeling collaborative industrial processes and RDF for structured knowledge representation. Building on this foundation, RDF knowledge graphs are introduced as labeled, directed graph representations of domain entities, together with graph embedding methods that provide numerical representations of graph structure. We explore different graph embedding paradigms, ranging from classical self-contained models to more recent graph neural network methods. The concepts introduced in this part form the basis for the semantic digital twin approach developed in this work.

Chapter 1

Industry 5.0: Goals and Challenges

Industrial production systems have undergone a sequence of transformations, commonly described as industrial revolutions, each characterized by distinct technological innovations and organizational paradigms. Industry 1.0 introduced mechanization through water and steam power, enabling the transition from manual craftsmanship to machine-supported production. Industry 2.0 advanced industrialization through electrification and assembly lines, allowing for mass production and standardized manufacturing processes. Industry 3.0 marked the rise of automation through electronics, programmable logic controllers, and information technology, enabling increased precision, efficiency, and control over industrial processes [148].

Building upon these developments, Industry 4.0 has emerged as a paradigm of digitalized and interconnected production environments. It is characterized by the integration of Cyber-Physical Production Systems (CPPSs) into manufacturing ecosystems, enabled by Internet of Things (IoT) infrastructures and advanced analytics capabilities [74, 111, 150, 228]. These technologies enable continuous data exchange between machines, sensors, software services, and enterprise applications across systems and organizational boundaries. As a result, modern production systems evolve into highly interconnected and information-rich environments, supporting capabilities such as real-time monitoring, predictive maintenance, and adaptive process optimization [129, 148, 221, 238]. However, the increasing degree of digitalization also leads to complex socio-technical systems composed of heterogeneous and independently developed components. In such environments, interactions between machines, software systems, and particularly human operators are typically realized through distributed interfaces and event-driven communication mechanisms [9, 37, 216]. While Industry 4.0 enables connectivity and data exchange across systems, it does not inherently provide a coherent representation of how these systems coordinate their behavior. Consequently, coordination logic often remains implicit and distributed across multiple systems, rather than being represented as a unified process model that captures interactions between machines, software systems, and human operators.

Hence, processes that span multiple systems and involve both human and automated decision-making cannot be directly analyzed or adapted as coherent units. This lack of coordination reduces transparency, limits traceability, and complicates the analysis and adaptation of system behavior across system boundaries [155, 256]. At the same time, the strong focus on automation and efficiency in Industry 4.0 architectures tends to underrepresent human roles and broader

system-level objectives [217]. In response to these challenges, Industry 5.0 has been introduced as a complementary paradigm that extends the technological foundations of Industry 4.0 by emphasizing human-centeredness, sustainability, and resilience [38]. Rather than replacing existing technologies, it reframes industrial systems as collaborative socio-technical environments in which humans, machines, and software services interact within shared and coordinated processes, whose explicit representation is essential for achieving transparency, adaptability, and human-centered decision-making [45].

In this chapter, we first review the technological foundations of Industry 4.0. We then analyze the limitations of automation-centric production with respect to coordination, transparency, and system integration. Building on this analysis, the chapter introduces the core principles of Industry 5.0 and discusses their implications for the design of future industrial information systems. These limitations motivate two key requirements: (i) process-centric representations that explicitly capture collaborative workflows across system boundaries, and (ii) semantic representations of domain knowledge that can be dynamically attached to process executions and integrated via a shared conceptualization.

1.1 Technological Foundations of Industry 4.0

As a successor to Industry 3.0, Industry 4.0 builds upon the automation principles established in Industry 3.0, which form the foundation of modern industrial control systems. In Industry 3.0, communication at the field level is enabled between sensors, actuators, and Programmable Logic Controllers (PLCs), which observe and control production processes via embedded control logic [9]. To ensure deterministic real-time communication between PLCs and field devices, specialized communication technologies such as fieldbus systems or Ethernet-based industrial protocols are employed [24]. Together, these technologies enable the execution of automated production steps within manufacturing environments with real-time characteristics.

To coordinate and supervise production processes, additional hierarchical layers are considered, in which human decision-making plays a central role. These layers are formalized in the ISA-95 standard [104], which defines a technology-independent hierarchy for manufacturing systems, as illustrated in Figure 1.1. At the lower levels 0–1 of operational technology, PLCs act as central control units, while higher-level systems such as Human-Machine Interfaces (HMIs) and Supervisory Control and Data Acquisition (SCADA) systems provide operators with access to process information and control capabilities. SCADA systems typically aggregate data from multiple PLCs and offer a unified interface for monitoring and interacting with production processes [37]. Such settings are commonly referred to as Human-In-The-Loop (HITL) scenarios, where operators observe system states, receive alerts, and intervene when necessary [196].

Above the SCADA layer, Manufacturing Execution Systems (MESs) coordinate production workflows and provide a link between operational control and business-level planning. At the highest level, Enterprise Resource Planning (ERP) systems manage business processes such as scheduling, logistics, and resource allocation [144]. The ISA-95 model thus provides a conceptual framework for structuring industrial systems across hierarchical layers [104]. However, while ISA-95 defines the structural organization of these systems, it does not prescribe specific communication technologies. In Industry 3.0 environments, communication between PLCs and

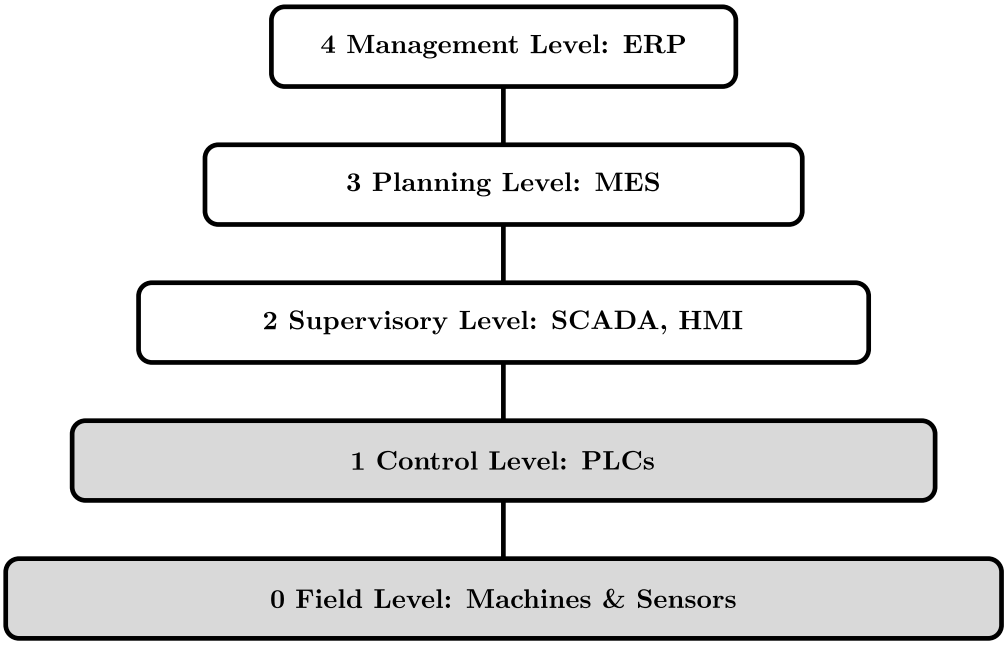


Figure 1.1: The ISA-95 hierarchy illustrating the separation of industrial systems into field, control, supervisory, planning, and management levels. Lower levels (0–1) correspond to operational technology, focusing on real-time control of physical processes, while higher levels (2–4) represent information technology, supporting human interaction, production planning, and business processes. Standardized interfaces enable data exchange across these layers, forming the basis for interconnected and data-driven Industry 4.0 systems.

higher-level systems was often realized using vendor-specific interfaces or technologies such as OLE for Process Control (OPC) [82]. Although widely adopted, these approaches lacked platform independence, scalability, and semantic interoperability. As a result, integration across systems remained complex, tightly coupled, and difficult to generalize.

The transition to Industry 4.0 addresses these limitations by introducing standardized, interoperable, and network-oriented communication paradigms. In particular, OPC Unified Architecture (OPC UA) provides a platform-independent and semantically rich interface for exposing industrial data, enabling systems to exchange not only data values but also structured and semantically enriched information [132]. Building on this foundation, lightweight messaging protocols such as Message Queuing Telemetry Transport (MQTT) facilitate scalable, decoupled, and event-driven data distribution across distributed systems [205]. This combination allows industrial data to be consumed not only by human-facing systems but also by automated, software-based entities such as analytics services, Artificial Intelligence (AI) agents, and digital twins, which are discussed in detail in Section 1.2.

The resulting connectivity is supported by distributed computing infrastructures. Edge computing enables low-latency processing close to physical processes, while cloud computing provides scalable resources for large-scale data storage and advanced analytics [228, 238]. Consequently,

CPPSs continuously generate heterogeneous operational data streams reflecting the dynamic state of machines, processes, and products. Industry 4.0 thus extends the classical human-centric consumption of industrial data towards a broader ecosystem in which both human and machine consumers interact with production systems. This shift enables new capabilities such as predictive maintenance, adaptive process optimization, and autonomous decision support, forming the basis for data-driven and intelligent manufacturing [129, 169]. This architectural shift is illustrated in Figure 1.2.

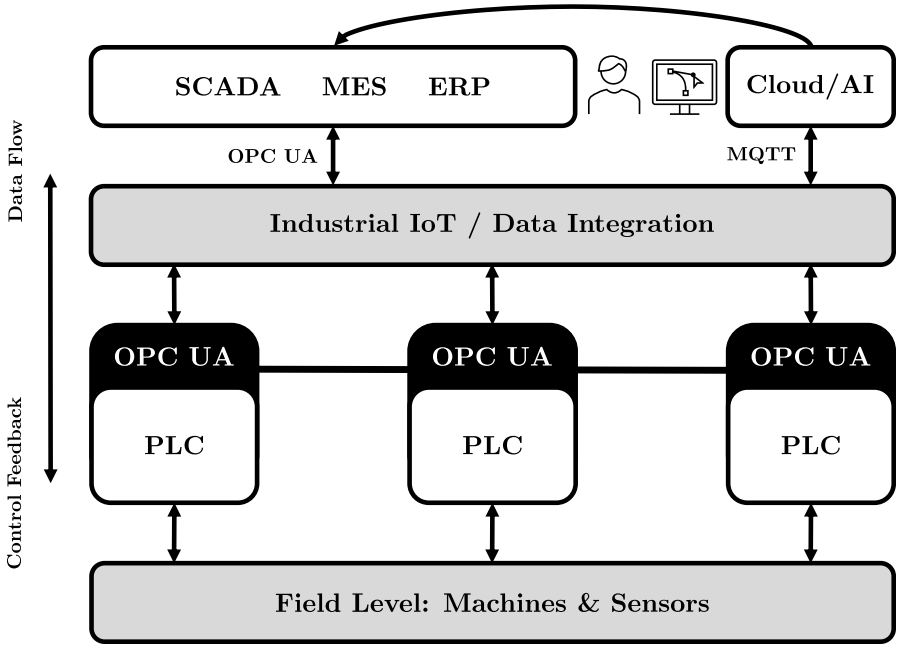


Figure 1.2: Illustration of a typical Industry 4.0 architecture in which industrial data is exposed from PLCs via standardized interfaces such as OPC UA and aggregated through an Industrial IoT and data integration layer. Higher-level systems, including SCADA, MES, and ERP, as well as cloud-based AI services, consume this data for monitoring, planning, and analysis. In addition to the predominantly upward data flow from operational technology to information technology, AI-based services introduce feedback by providing predictions, recommendations, or decision support to higher-level systems. While this architecture enables interoperability and scalable data exchange across system boundaries, it does not provide explicit representations of coordination or collaborative processes between heterogeneous actors.

1.2 Digital Representations of Industrial Systems

A central characteristic of Industry 4.0 is the continuous acquisition and integration of operational data from interconnected industrial systems. Enabled by technologies such as the IoT, CPPSs, and standardized communication interfaces (e.g., OPC UA), physical production systems become increasingly observable and accessible within networked environments. This per-

vasive data availability forms the foundation for creating digital representations of industrial systems, which support monitoring, analysis, and optimization across system boundaries. Digital representations capture structural properties, operational states, and historical behavior of physical assets and processes. Depending on the degree of synchronization and interaction between the physical and digital domains, these representations are categorized into digital models, digital shadows, and digital twins [126, 221]. This classification reflects a progression from static representations to data-driven systems that actively participate in industrial decision-making.

Digital Model

A digital model represents the structural and conceptual aspects of an industrial system without continuous synchronization with its physical counterpart. It is typically created during design or engineering phases and used for documentation, configuration, or offline simulation. As such, digital models provide a static abstraction of a system, capturing its intended structure and behavior but not its current operational state [126].

Digital Shadow

A digital shadow extends this concept by incorporating continuous or event-driven data acquisition from the physical system. Information flows unidirectionally from the physical system to the digital representation, enabling the latter to reflect the current state of the system based on sensor data, machine logs, or production databases. Digital shadows therefore enable real-time monitoring and retrospective analysis but do not actively influence the physical system [31, 126].

Digital Twin

A digital twin represents the most advanced form of digital representation, characterized by a bidirectional and dynamic coupling between the physical system and its digital counterpart. In addition to continuously receiving data, the digital twin can generate predictions, optimization strategies, or control actions that influence the physical system. This enables adaptive and data-driven behavior, supporting use cases such as predictive maintenance, process optimization, and autonomous decision support [184, 221]. Within Industry 4.0, digital twins play a key role in transforming industrial systems from passive data sources into active participants in decision-making processes. They serve as integration points where data from heterogeneous sources is aggregated, analyzed, and translated into actionable insights. However, existing digital twin approaches are predominantly machine-centric and focus on physical assets and their operational states. As a result, they often lack explicit, system-level representations of process execution and human-machine interactions. They are limited in their ability to capture and analyze collaborative processes that span multiple systems and involve both human and automated actors [138].

This limitation highlights the need for process-centric and semantically enriched representations that extend beyond asset-level modeling and capture interactions between heterogeneous actors, e.g., by incorporating human roles and decision-making within industrial processes. This enables a shift from simulation-based optimization of individual assets to the analysis and optimization of collaborative, system-level scenarios.

1.3 The Human Role within Industry 4.0

In addition to automation and system integration, Industry 4.0 environments incorporate human actors and intelligent software components into industrial processes. Human–machine collaboration manifests in both physical and cognitive forms. Physical collaboration is realized through collaborative robots that operate in shared workspaces with human operators, assisting in tasks such as assembly and material handling [26]. Cognitive collaboration involves AI-based decision support systems that provide predictions, anomaly detection, and optimization recommendations, while human operators contribute contextual understanding, domain expertise, and final decision-making [2, 154].

These interaction patterns are commonly described using concepts such as HITL and Human-On-The-Loop (HOTL), which denote different degrees of human involvement in automated decision processes [72]. In HITL scenarios, human operators actively participate in decision-making by validating or modifying system outputs, whereas HOTL approaches involve supervisory roles where humans monitor system behavior and intervene when necessary.

While these paradigms enable meaningful collaboration between humans and intelligent systems, their realization is typically confined to individual systems or applications. Human interaction is often mediated through system-specific user interfaces, dashboards, or control panels, while AI components are embedded within localized processing pipelines. As a result, human and AI involvement remains embedded within the internal logic of individual systems, without an explicit, system-spanning representation of collaborative processes.

1.4 Principles of Industry 5.0

Industry 5.0 has been proposed in response to the limitations of predominantly automation- and efficiency-driven Industry 4.0 systems, offering a broader perspective on the role of industry within society. Rather than pursuing autonomous optimization as an end in itself, it advocates a balanced orientation in which technological innovation supports human-centered work practices and robust industrial operations [38, 155, 256]. According to the European Commission, Industry 5.0 positions human needs and planetary boundaries as central to production [38]. It does not replace Industry 4.0, but builds upon its technological foundations while reframing industrial progress toward systems that integrate human expertise, support transparent decision-making, and remain adaptable under evolving conditions. Three interrelated principles characterize Industry 5.0: human-centricity, sustainability, and resilience, each addressing limitations of current industrial systems and motivating new directions in system design [155, 256].

1.4.1 Human-Centricity

Human-centricity emphasizes the central role of human skills, expertise, and creativity in industrial environments [217]. While Industry 4.0 often positions humans primarily as supervisors of automated systems, Industry 5.0 promotes technologies that augment human capabilities and enable meaningful collaboration between humans and intelligent systems. This includes both physical collaboration, e.g., through collaborative robotics, and cognitive collaboration, where

humans and AI jointly contribute to analysis and decision-making [5, 217]. Industrial processes are thus understood as socio-technical systems in which human judgment and contextual knowledge remain indispensable. Beyond merely including humans in the operational loop, systems must enable humans to understand system behavior, interact effectively, and contribute expertise at meaningful points in the workflow. This perspective is closely related to HITL and HOTL paradigms [72], but requires systems to be transparent, controllable, and trustworthy [70, 225]. However, current industrial systems lack representations of such collaborative processes, resulting in fragmented and implicit coordination across heterogeneous components.

1.4.2 Sustainability

Sustainability extends industrial objectives by integrating ecological and societal considerations into system design and operation [38, 54]. It encompasses energy-efficient production, responsible resource use, and waste minimization. Digital technologies support these goals by enabling monitoring and optimization across production stages and supply networks, promoting systems that balance efficiency with long-term environmental and societal viability [226].

1.4.3 Resilience

Resilience refers to the ability of industrial systems to anticipate, adapt to, and recover from disruptions. In complex and interconnected environments, systems optimized solely for efficiency often lack the flexibility required to handle uncertainty [38]. Industry 5.0 therefore promotes adaptive architectures that integrate real-time data and dynamic process models to enable rapid reconfiguration. This requires explicit representations of processes that can be adapted at runtime. Resilience is closely linked to human-centricity, as human judgment and situational awareness remain essential for managing unforeseen events [217].

Together, these principles highlight the need for systems that integrate heterogeneous data, support transparent human–machine collaboration, and remain adaptable under changing conditions. This motivates semantic representations that unify domain knowledge, machine data, human roles, and process knowledge within a common framework [22, 146]. In particular, semantic process models and knowledge graphs enable the representation of both static knowledge and process execution, providing a foundation for contextualization, reasoning, and collaborative decision-making [19, 45, 112]. This thesis therefore focuses on human-centric, process-based digital twins that explicitly represent collaborative workflows involving human operators, software systems, and physical machines. In contrast to asset-centric digital twin approaches, the emphasis lies on representing process execution, coordination logic, and human decision-making within system-spanning, collaborative workflows [2, 70]. While sustainability and resilience are considered where relevant, the primary objective is to enable transparent, adaptable, and semantically grounded representations of collaborative industrial processes. The following chapter introduces knowledge bases as schema-based representations of domain knowledge. This unified perspective encompasses both the Business Process Model and Notation (BPMN) modeling formalism and the Resource Description Framework (RDF) data model, providing the conceptual foundation for the semantic, process-centric digital twin formalism developed in the remainder of this thesis.

Chapter 2

Layered Knowledge Modeling with BPMN and RDF

Digital systems increasingly require representations that make domain knowledge explicit, interpretable, and reusable across organizational and technical boundaries [162, 232]. This need is particularly pronounced in the context of Industry 5.0, where human-centricity, resilience, and the integration of heterogeneous systems demand shared representations of processes, entities, and their relationships [38]. Accordingly, information must be organized in forms that capture both conceptual and assertional knowledge [167]. In information science, such representations are commonly subsumed under the notion of a knowledge base, understood as a technology-independent abstraction for organizing domain knowledge through a shared terminology [95, 194]. In this chapter, we adopt a layered knowledge modeling perspective and interpret this structure through complementary knowledge base views. Building on this foundation, the chapter relates this abstraction to two realizations central to this thesis: BPMN knowledge bases for process modeling and execution tracing [164], alongside RDF knowledge bases for structuring and instantiating domain knowledge based on Semantic Web technologies [50].

2.1 Layered Knowledge Modeling

From an engineering perspective, systems for representing domain knowledge are typically constructed along a three-layer structure that separates the means of expression from the models formulated in it and from the instance-level data described by those models [16, 136]:

1. A **modeling language** specifies which modeling statements can be expressed. It defines the available primitives, their syntax, and their intended semantics.
2. A **domain model** is a concrete specification formulated in that modeling language, introducing domain-specific categories, relations, constraints, and structural patterns. It thereby provides a formal conceptualization of the domain.
3. A **domain population** is a collection of observations described using the domain model. It captures operational data, including concrete entities and their relationships.

These three layers form a successive chain of specification. The domain model is formulated using the constructs of the modeling language, and the population is described using the vocabulary introduced by the domain model. Each layer therefore provides the vocabulary used to formulate statements at the next layer. In knowledge representation theory, this layered construction is analyzed through the notion of knowledge bases, comprising two complementary components: a terminological component and an assertional component [95]. In the context of Description Logics, these are referred to as the Terminological Box (TBox) and the Assertional Box (ABox), respectively [81]. The TBox provides the conceptual vocabulary and logical axiomatization that define the structure of the modeled domain, such as classes, relations, and constraints, while the ABox contains assertions formulated using that vocabulary. This distinction therefore does not introduce an additional engineering layer. Instead, it represents a logical perspective applied to a selected pair of the previously introduced layers, as illustrated in Figure 2.1.

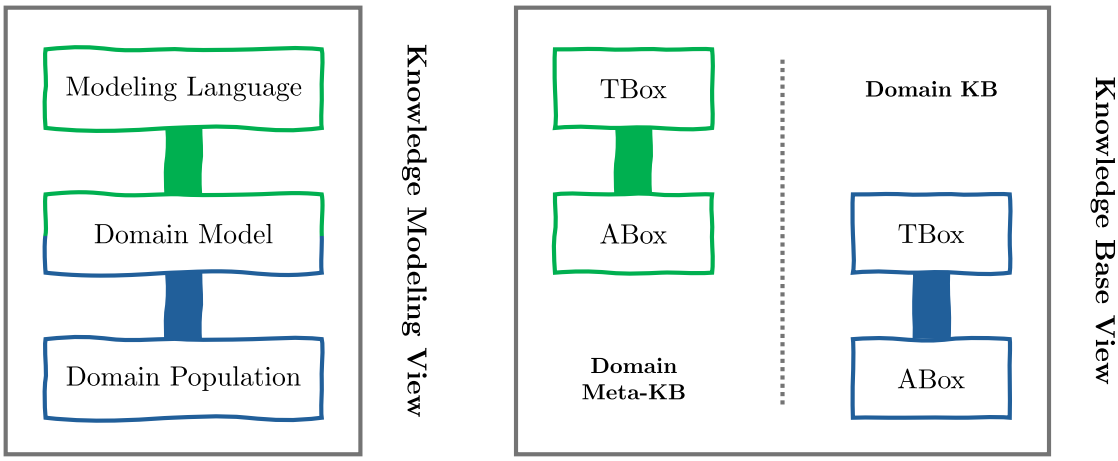


Figure 2.1: Relation between the layered knowledge modeling view (left) and the knowledge base perspective (right). Combining the domain model with its population yields a domain knowledge base (blue), whereas combining the modeling language with the domain model yields a domain meta-knowledge base (green).

Crucially, the role of the domain model is relative to the chosen viewpoint. The three-layer knowledge modeling structure can therefore be interpreted as two complementary knowledge bases that share the domain model as a common component. When considered together with its population, the domain model represents the TBox, while the population forms the corresponding ABox. Conversely, when the modeling language serves as the terminological reference, the domain model constitutes the ABox of the corresponding domain meta-knowledge base. The distinction between instance and abstraction is therefore defined only with respect to the knowledge base under consideration [194]. In this work, we adopt the perspective of the domain when referring to knowledge bases. Accordingly, the term knowledge base denotes the domain knowledge base, i.e., the combination of a domain model and its corresponding population. Unless stated otherwise, all subsequent references to a knowledge base in this thesis implicitly adopt this domain-level interpretation.

Relational databases illustrate the layered knowledge modeling perspective in a transparent way [117], as shown in Listing 2.1, using the domain of music as an example. The SQL Data Definition Language (DDL) acts as the modeling language by providing native constructs such as CREATE TABLE. The resulting table declarations collectively constitute the domain model, as they define the structural vocabulary of the music domain. The rows stored in these tables form the domain population, expressing assertions about concrete artists and songs. The SQL Data Manipulation Language (DML) provides the operations used to create and modify such population-level assertions.

```
-- Domain Model: defined using SQL DDL
CREATE TABLE Artist (
    id       INTEGER PRIMARY KEY,
    name     VARCHAR(200),
    country  VARCHAR(100) );
CREATE TABLE Song (
    id            INTEGER PRIMARY KEY,
    title         VARCHAR(200),
    artist_id     INTEGER,
    release_date  DATE,
    FOREIGN KEY (artist_id) REFERENCES Artist(id) );

-- Domain Population: rows inserted via SQL DML
INSERT INTO Artist (id, name, country) VALUES
    (1, 'Nine Inch Nails', 'USA'),
    (2, 'Massive Attack', 'UK');
INSERT INTO Song (id, title, artist_id, release_date) VALUES
    (1, 'Hurt', 1, '1995-04-17'),
    (2, 'The Perfect Drug', 1, '1997-05-13'),
    (3, 'Teardrop', 2, '1998-04-27');
```

Listing 2.1: SQL domain model and population for a music domain within a relational database.

The example further illustrates that the distinction between instantiation and abstraction emerges relative to the chosen knowledge base perspective. Within the meta-knowledge base, the modeling vocabulary provided by SQL acts as the terminological component, while the table declarations Artist and Song appear as assertions describing instances of the modeling construct TABLE. Within the knowledge base, by contrast, the table declarations belong to the TBox, where they represent instantiatable abstractions of artists and songs. The rows inserted into the tables Artist and Song then form the corresponding ABox, constituting the domain population. The domain model thus changes logical roles across perspectives, as it constitutes assertional knowledge relative to SQL DDL and terminological knowledge relative to the actual data in the database.
In general, this layered separation provides a unifying abstraction across knowledge representation paradigms. In this work, we employ BPMN to represent domain knowledge from a process perspective, capturing process models and their executions, whereas the RDF stack is used to represent domain knowledge in terms of ontological conceptualizations and their instantiations.

When representing domain knowledge via BPMN, the BPMN specification serves as the modeling language [164]. A collection of BPMN process models constitutes the domain model, while concrete execution traces form the respective domain population. For RDF, corresponding ontology languages collectively serve as the modeling language [96, 39, 50]. Domain ontologies expressed in these languages constitute the domain model, whereas collections of RDF assertions describing concrete individuals form the population [99].

2.2 Layered Knowledge Modeling with BPMN

BPMN is one of the most widely adopted standards for modeling business processes [204]. Developed and maintained by the Object Management Group (OMG), it evolved from earlier workflow and process modeling notations such as Unified Modeling Language (UML) activity diagrams and event-driven process chains [63, 109]. Its design goals include providing a notation that is understandable to both business stakeholders and technical implementers, together with a machine-readable XML serialization that supports interchange and execution [164]. Within the layered knowledge modeling perspective introduced in Section 2.1, BPMN itself functions as a modeling language for representing domain knowledge in terms of abstract but executable process models. Consequently, domain knowledge is modeled with respect to the processes that take place within the domain and the activities, events, and decision points that structure their execution. A collection of BPMN process models constitutes the domain model, while concrete executions recorded as event logs form the corresponding domain population [52, 145]. Interpreted through the complementary knowledge base perspective from Section 2.1, the domain model represented by BPMN process models assumes terminological or assertional roles, depending on whether the domain knowledge base or meta-knowledge base is considered.

2.2.1 BPMN as a Modeling Language

In its current and widely adopted form, BPMN 2.0 constitutes the de-facto standard for business process modeling [164]. It is widely used in domains such as business administration, service orchestration, and organizational workflow management [78]. However, in industrial environments shaped by automation-centric paradigms and Industry 4.0 technologies, it has so far played only a limited role, as these settings typically rely on lower-level control and orchestration models tailored towards machine-driven execution [147, 189]. Nevertheless, with the growing emphasis on human-centricity, adaptability, and cross-system integration, its comparatively human-oriented perspective is becoming increasingly relevant for modeling supervisory and collaborative processes in industrial scenarios [171], particularly in the context of Industry 5.0 [256]. As a modeling language in the sense introduced above, BPMN provides a formalized set of modeling constructs, a graphical notation, and an execution-oriented semantics for describing business processes. It is specified through a conceptual definition of its abstract syntax, an XML-based serialization that enables machine-readable exchange, and an extensive specification document that explains the intended semantics and permissible combinations of modeling elements [164]. Together, these components ensure that BPMN process models can be created, validated, exchanged, and executed consistently across tools and platforms [8].

At the core of the language are flow objects, which represent fundamental behavioral constructs [63]. Activities describe units of work and include user tasks, manual tasks, service tasks, script tasks, and other specialized variants. Events designate points at which something happens during a process, including start, intermediate, and end events, with refinements such as message, timer, error, escalation, and signal events. Gateways define routing decisions and synchronization points. Exclusive gateways model choices, parallel gateways enable concurrent paths, and inclusive or event-based gateways support more complex control-flow behavior.

Beyond flow objects, BPMN incorporates additional constructs that structure and contextualize processes. Sequence flows encode permissible execution order, while message flows capture interactions between process participants. Pools and lanes partition a model into organizational or system-level responsibilities. Data objects represent information produced or consumed by activities, data stores provide persistent repositories, and data associations link activities to the data they read or write. Although BPMN is not intended as a full data-modeling language, these constructs integrate essential data-flow aspects into the control-flow perspective [8].

The BPMN 2.0 specification prescribes how these elements may be combined and defines the token-flow semantics governing their behavior during execution [128]. From the viewpoint adopted in this thesis, it can therefore be understood as specifying both the vocabulary and the rules governing how its constructs may be combined. Its formal specification and XML schema capture the core structural aspects, while much of the additional behavioral interpretation is conveyed through descriptive text. In this sense, the BPMN specification plays a role comparable to that of a meta-model by defining admissible modeling constructs and their combinations, although its semantics are partly formalized and partly described in natural language [164]. The main modeling constructs, including graphical representations, are illustrated in Figure 2.2.

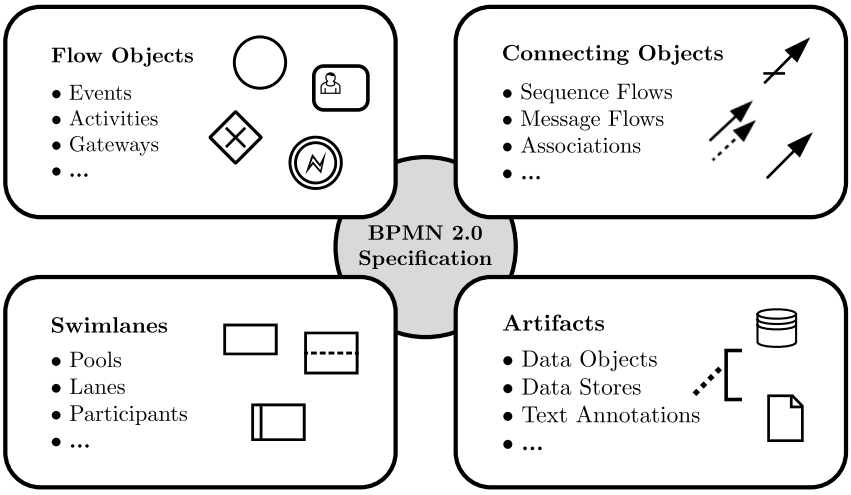


Figure 2.2: Illustrative overview of the four main categories of BPMN modeling constructs as defined in the BPMN 2.0 specification [164]. Flow objects capture the core behavior of a process, connecting objects define control-flow and communication relations, swimlanes structure responsibilities across participants, and artifacts provide supplementary information such as data and annotations. Together, they constitute the core vocabulary of the BPMN modeling language.

2.2.2 BPMN Process Models as Domain Model

Concrete BPMN process models are formulated using the constructs of the BPMN modeling language. While BPMN defines the available modeling vocabulary, individual models arrange these constructs to represent domain-specific operational behavior. Modeling domain knowledge through BPMN thus presupposes a process-oriented domain in which operational procedures form a central aspect of the system under consideration, as is the case in the industrial environments considered in this thesis. Within the layered knowledge modeling perspective introduced above, such process models therefore constitute the domain model.

In practice, a BPMN-based domain model is rarely represented by a single process model. Instead, it is typically realized as a modular collection of BPMN process models, each describing a particular process or operational scenario within the domain [220]. Taken together, these models form a structured representation of how activities are coordinated, responsibilities are distributed, and information flows between participating actors and systems.

A BPMN domain model thus captures knowledge about how processes are intended to operate by specifying their control-flow logic, organizational responsibilities, and relevant data interactions [8]. At the control-flow level, a process model defines sequencing and dependency relations between activities, conditions for branching, and synchronization points. At the organizational level, pools and lanes assign responsibilities to actors, departments, systems, or external partners. At the data level, the model represents the information consumed or produced by activities, how data objects move through the process, and how persistent data stores are accessed [63].

BPMN further supports hierarchical modeling through subprocesses, call activities, and event subprocesses, enabling modularization, reuse, and structured representation of complex behavior [195]. Variants of a process, such as conditional subprocesses or alternative execution paths, can be expressed within a single model, allowing it to capture operational variability within a domain. The XML serialization prescribed by the BPMN 2.0 specification ensures that these models are machine-readable and can be interpreted consistently by execution or analysis tools [164]. The graphical notation and the XML representation encode the same domain model by defining the structural constraints that admissible process executions must follow. Figure 2.3 shows a representative example of a BPMN process model, whose elements constrain the ordering of activities, the allocation of responsibilities, and the permitted flow of information.

From the layered knowledge modeling perspective introduced in Section 2.1, the role of process models depends on the level of consideration. During the design phase, they act as instances of the BPMN modeling language and therefore constitute the assertional component of a BPMN meta-knowledge base, specifying which modeling constructs are used and how they are combined in a domain. Once deployed, however, the same process models assume the terminological role within a BPMN knowledge base by defining the structural constraints to which process executions must conform. Observed executions, captured as event logs or runtime traces [3], then populate this structure with instance-level data. The process model shown in Figure 2.3 illustrates this principle using two interacting processes organized as separate BPMN participants. One process represents the production cell controlling the manufacturing cycle, while the other represents an external AI-based inspection service. The processes communicate through message events, thereby forming a nested collaboration in which overall operational behavior emerges from their interactions. The following subsection elaborates on this perspective.

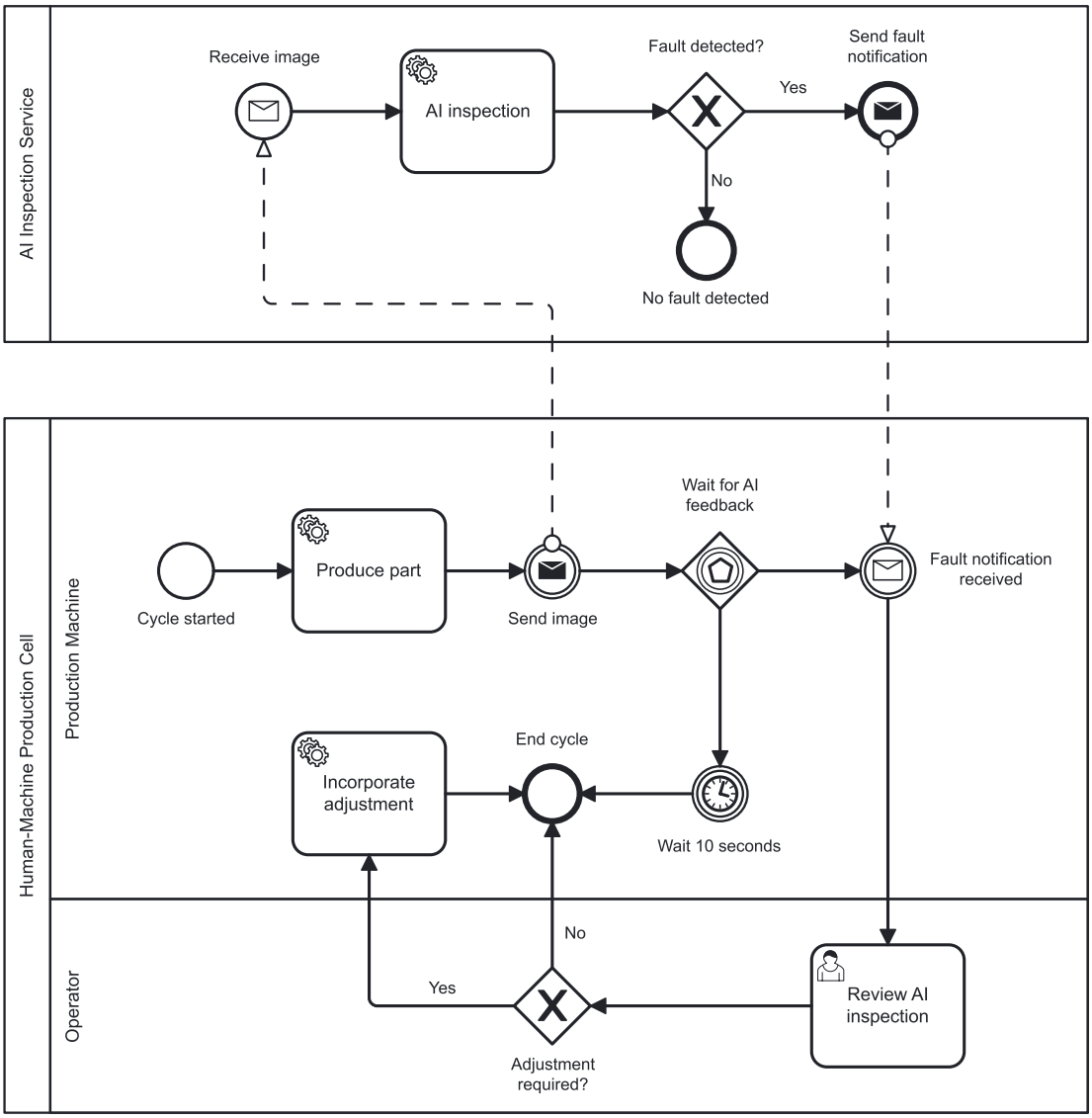


Figure 2.3: Example BPMN process model illustrating an asynchronous interaction between automated production and an external AI-based inspection service. The process is structured using pools and lanes that separate responsibilities of the production machine, the operator, and the inspection service. After a production cycle, the machine sends an inspection image to the AI service. The factory process then waits for either a fault notification or a timeout using an event-based gateway. If a defect is predicted, a human operator reviews the inspection result and decides whether a machine adjustment is required. The diagram demonstrates typical BPMN elements such as service and user tasks, message flows, gateways, timer events, and pooling, which together constrain the admissible execution behavior of the process.

2.2.3 BPMN Process Executions as Domain Population

Given a deployed BPMN domain model, consisting of one or more process models that form the terminological component of the BPMN knowledge base, the corresponding population arises from concrete executions carried out by workflow or orchestration engines such as Camunda, Flowable, or Activiti [163]. These engines interpret the XML serializations of BPMN process models and instantiate their elements at runtime, maintaining execution states and emitting events that document the behavior of process instances in operational environments [145]. Such events are typically recorded as event logs or audit trails and organized into traces, where each trace represents the execution history of a single process instance [52], i.e., a temporally ordered sequence of events such as activity starts and completions, timer triggers, or message correlations between interacting processes. Figure 2.3 illustrates a representative example. During runtime, the execution of activities, such as AI inspection activities, gateway decisions, timer events, and message exchanges, generate corresponding events in the execution log. Each production cycle thus gives rise to an execution trace that records the temporal order in which the modeled elements are instantiated.

Depending on the execution environment, recorded events may contain additional metadata such as timestamps, resource identifiers, user roles, tenant identifiers, process variables, or contextual information obtained from surrounding systems, for example IoT-connected production equipment [177]. These records are commonly persisted in structured storage systems such as relational databases or log repositories, from which execution traces can be reconstructed and analyzed. Each event thereby represents the concrete realization of a modeled element: the start of a user task instantiates the corresponding activity, a timer trigger realizes a modeled timer event, and a message correlation corresponds to a defined inter-process communication. Execution traces thus provide the assertional content associated with a BPMN domain model by grounding its abstract control-flow structure in observed runtime behavior. While the BPMN process model typically evolves slowly, the population of execution traces grows continuously as processes are executed. This separation between a comparatively stable structural description and a dynamically evolving body of observations makes BPMN-based knowledge representations well suited for monitoring, conformance checking, and the analysis of collaborative workflows [78]. From the perspective adopted in this thesis, a process-driven BPMN domain knowledge base therefore consists of process models as TBox that define the structural constraints of admissible executions together with the execution traces that instantiate them as ABox.

However, the expressivity of BPMN remains largely confined to the control-flow and coordination aspects of processes. While the notation captures how activities are ordered, how participants interact, and how decisions affect execution paths, it provides only limited means for representing the broader semantics of heterogeneous domains, such as industrial production environments. In the example shown in Figure 2.3, the process model describes the sequence of activities involved in production and inspection, yet it does not formally represent the produced parts, the machines performing the operations, the operators participating in the workflow, or the specific AI models used for inspection. Such domain-level semantics lie outside the expressive scope of BPMN itself. Integrating these additional conceptual relations therefore requires complementary general-purpose knowledge representation formalisms, which motivates the use of RDF-based knowledge modeling discussed in the following section.

2.3 Layered Knowledge Modeling with RDF

The Resource Description Framework (RDF) is a W3C standard for representing information in a machine-processable and human-interpretable form [50, 99], constituting a core technology of the Semantic Web initiative [56, 175]. Originally designed to enable interoperable data exchange across distributed systems, it has evolved into a general-purpose framework for modeling and instantiating heterogeneous domain knowledge. In contrast to BPMN, which functions as a process-centered modeling formalism for representing domain knowledge, RDF defines a data model for expressing information as subject–predicate–object triples with formally specified semantics. In this sense, RDF at its core provides a representation paradigm rather than a modeling language in the strict sense. In the layered knowledge modeling perspective introduced in Section 2.1, RDF does not syntactically enforce a separation between modeling language, domain model, and domain population. Instead, it provides a uniform representation in which terminological and assertional knowledge can be expressed using the same structural mechanism. However, RDF itself defines only a minimal vocabulary for describing resources, which is extended and reused by ontology languages built on top of RDF, most prominently RDF Schema (RDFS) [39] and the Web Ontology Language (OWL) [96]. These languages enable the definition of a domain model in the form of ontologies encoded using the RDF data model. Within an RDF meta-knowledge base, such ontologies constitute the assertional component relative to the ontology languages. When considered together with concrete instance data, however, the same ontologies assume the terminological role within an RDF knowledge base. The flexibility of reusing RDF representations across modeling layers has made RDF-based technologies a central approach for semantic data integration, knowledge representation, and interoperability across organizational boundaries [15, 99, 107].

2.3.1 The RDF Data Model

The RDF data model is a foundational representation formalism of the Semantic Web for describing knowledge about resources within a domain [50]. As a web-native representation, RDF employs globally unique identifiers to denote resources unambiguously. The classical implementation mechanism for such identifiers is the Uniform Resource Identifier (URI) as defined in RFC 3986 [32], which specifies an ASCII-based syntax consisting of a scheme, an authority, and a path. DBpedia, for example, is a well-known service that provides RDF data by extracting structured information from Wikipedia and making it available for semantic reuse on the web [17]. The entry of the Swedish city Malmö is identified by the following URI:

https://dbpedia.org/resource/Malm%C3%B6

To accommodate the full range of Unicode characters, RFC 3987 extends this notion to Internationalized Resource Identifiers (IRIs) [65]. IRIs preserve the structural components of URIs while allowing Unicode characters natively. The same DBpedia entry can thus be expressed as:

https://dbpedia.org/resource/Malmö

Any IRI can be converted into a valid URI through percent encoding of non-ASCII characters. Apart from this encoding step, URIs and IRIs share the same semantic interpretation in RDF.

Within the Semantic Web community, the term IRI is therefore the standard for identifying resources, and is adopted throughout this work [99]. In practice, IRIs are often decomposed into a namespace and a local name to obtain shorter identifiers. For example,

https://dbpedia.org/resource/

may be associated with the prefix dbr, yielding compact notations such as dbr:Malmö. Prefixes are purely syntactic conveniences, and the full IRI is obtained by concatenation.
Besides global IRIs, RDF also includes blank nodes as locally scoped resources without globally meaningful identifiers [50]. They act as existential placeholders whose internal labels (e.g., _:b1) have no meaning beyond the local document. In this thesis, however, we do not distinguish further between IRIs and blank nodes, since both denote resources and differ only in the scope of their identifiers. Without loss of generality, resources may thus be treated as represented by IRIs, since blank nodes can be replaced by IRIs without affecting structural properties [98].
The third category of RDF terms are literals, which function as data values [50]. A literal is a Unicode string optionally annotated with a datatype IRI (for example "42"^^xsd:integer) or with a language tag (for example "Malmö"@sv). Literals are not considered resources, as they denote data values whose interpretation must be considered with respect to their context.
Let $\mathfrak{I}$ denote the set of all IRIs, $\mathfrak{B}$ the set of all blank nodes, and $\mathfrak{L}$ the set of all literals. We define the set of resources as $\mathfrak{R} = \mathfrak{I} \cup \mathfrak{B}$. Domain knowledge is therefore represented as triple facts $(s, p, o) \in \mathfrak{R} \times \mathfrak{I} \times (\mathfrak{R} \cup \mathfrak{L})$, where subjects must be resources, predicates must be IRIs, and objects may be resources or literals. When $o \in \mathfrak{L}$ is a literal, (s, p, o) expresses a literal-valued attribute, and when $o \in \mathfrak{R}$ identifies a resource, it expresses a resource-valued fact. Accordingly, an RDF triple set is a collection of subject-predicate-object facts

$$\mathcal{F} \subseteq \mathfrak{R} \times \mathfrak{I} \times (\mathfrak{R} \cup \mathfrak{L}).$$

Each RDF triple set structures knowledge about the RDF terms occurring in subject and object positions, which we collectively call the arguments of the triples

$$\mathcal{A}(\mathcal{F}) \;:=\; \{\, s \mid (s,p,o) \in \mathcal{F} \,\} \cup \{\, o \mid (s,p,o) \in \mathcal{F} \,\} \;\subseteq\; \mathfrak{R} \cup \mathfrak{L}.$$

Analogously, given a triple set $\mathcal{F}$, we define the set of induced predicates as

$$\mathcal{P}(\mathcal{F}) \;:=\; \{\, p \mid (s,p,o) \in \mathcal{F} \,\} \;\subseteq\; \mathfrak{I}.$$

Using these induced sets, the triple set $\mathcal{F}$ can equivalently be expressed as

$$\mathcal{F} \subseteq \mathcal{A}(\mathcal{F}) \times \mathcal{P}(\mathcal{F}) \times \mathcal{A}(\mathcal{F}),$$

which provides a bootstrapped, more precise characterization of the same triple set by restricting the Cartesian product to the arguments and predicates that actually occur in $\mathcal{F}$. To differentiate between resource and literal arguments, we define the corresponding restrictive notations

$$\mathcal{A}_{\mathfrak{R}}(\mathcal{F}) := \mathcal{A}(\mathcal{F}) \cap \mathfrak{R} \quad \text{and} \quad \mathcal{A}_{\mathfrak{L}}(\mathcal{F}) := \mathcal{A}(\mathcal{F}) \cap \mathfrak{L}.$$

Moreover, RDF triple sets can be decomposed into resource- and literal-valued triple subsets

$$\mathcal{F}_{\mathfrak{R}} := \{\,(s,p,o) \in \mathcal{F} \mid o \in \mathfrak{R}\,\} \subseteq \mathfrak{R} \times \mathfrak{I} \times \mathfrak{R}, \quad \mathcal{F}_{\mathfrak{L}} := \{\,(s,p,o) \in \mathcal{F} \mid o \in \mathfrak{L}\,\} \subseteq \mathfrak{R} \times \mathfrak{I} \times \mathfrak{L},$$

with $\mathcal{F} = \mathcal{F}_{\mathfrak{R}} \uplus \mathcal{F}_{\mathfrak{L}}$. Hence, $\mathcal{A}(\mathcal{F}_{\mathfrak{R}}) \subseteq \mathcal{A}_{\mathfrak{R}}(\mathcal{F})$ holds and $\mathcal{A}(\mathcal{F}_{\mathfrak{R}})$ may be a proper subset of $\mathcal{A}_{\mathfrak{R}}(\mathcal{F})$, since a resource argument may participate exclusively in literal-valued triples. Moreover, $\mathcal{A}_{\mathfrak{L}}(\mathcal{F}) \subseteq \mathcal{A}(\mathcal{F}_{\mathfrak{L}})$ holds, with equality if $\mathcal{F}_{\mathfrak{L}} = \emptyset$. Analogous to the decomposition of the argument set, we define the (not necessarily disjoint) induced resource and literal predicates:

$$\mathcal{P}_{\mathfrak{R}}(\mathcal{F}) := \{\,p \mid (s,p,o) \in \mathcal{F}_{\mathfrak{R}}\} = \mathcal{P}(\mathcal{F}_{\mathfrak{R}}) \quad \text{and} \quad \mathcal{P}_{\mathfrak{L}}(\mathcal{F}) := \{\,p \mid (s,p,o) \in \mathcal{F}_{\mathfrak{L}}\} = \mathcal{P}(\mathcal{F}_{\mathfrak{L}}).$$

RDF semantics adopts two assumptions that govern how facts are interpreted [50]. First, RDF-modeled data follows the Open World Assumption (OWA), under which missing information is treated as unknown rather than false, reflecting the inherent incompleteness of distributed web data. Second, the Non-Unique Name Assumption (NUNA) specifies that different identifiers are not presumed to denote different resources. These assumptions facilitate the integration of heterogeneous sources and underpin the flexible, extensible modeling style of the Semantic Web. Because RDF defines an abstract, fact-based data model, RDF triple sets must be serialized for storage and exchange. Common serialization formats include RDF/XML, N-Triples, JSON-LD, and Turtle (TTL) [178], the latter of which we use throughout this thesis whenever RDF snippets are presented. These formats differ in readability and tooling convenience, yet all share the same model-theoretic semantics and thus represent equivalent RDF encodings.
Listing 2.2 shows a minimal example expressing that Malmö is a city located in Sweden with a population count of 339316. All statements are expressed uniformly as triples, regardless of whether they describe concrete entities, their relationships, or references to more general categories. The RDF data model itself does not prescribe a separation between TBox and ABox data. Such distinctions arise only through how particular IRIs are used within a dataset.

```
@prefix dbr: <https://dbpedia.org/resource/> .
@prefix dbo: <https://dbpedia.org/ontology/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

dbr:Malmö rdf:type dbo:City ;
    dbo:country dbr:Sweden ;
    dbo:populationTotal "339316"^^xsd:nonNegativeInteger .
```

Listing 2.2: Example RDF statements in TTL syntax.

RDF nevertheless defines a small core vocabulary in the rdf namespace. This vocabulary includes terms such as rdf:type and rdf:Property, which provide basic mechanisms for stating that a resource is an instance of another resource or that a predicate is used as a property [50]. By convention, these terms are employed to relate descriptive resources to more abstract ones, thereby enabling representations that resemble the terminological-assertional organization of knowledge bases discussed in Section 2.1. Importantly, this structuring is not enforced by the RDF data model itself but arises through the reuse of shared vocabularies within its uniform

triple representation. To support the explicit formulation of RDF domain ontologies and richer conceptual constraints, the RDF vocabulary is extended by RDF ontology languages such as RDFS and OWL. These languages remain fully grounded in the RDF data model while introducing additional vocabulary and semantics for defining domain-level abstractions. Their role in enabling RDF domain models is examined in the following subsection.

2.3.2 RDF Ontology Languages as a Modeling Language

As outlined above, the RDF specification already provides a small built-in vocabulary in the rdf namespace that enables basic typing and the identification of predicates [50]. This vocabulary, however, is intentionally minimal and does not yet provide the expressive means required to define abstract classes, hierarchies, or domain constraints. In order to formulate domain conceptualizations that function as a TBox within an RDF knowledge base, additional ontology languages are required. The Semantic Web stack therefore introduces RDFS as a first modeling language building on RDF, followed by the more expressive formalism OWL, both of which are formulated in the same RDF triple-based representation [96, 39]. Complementary languages such as Shapes Constraint Language (SHACL) and Semantic Web Rule Language (SWRL) extend this framework with validation and rule-based reasoning capabilities [102, 115].

RDF and RDF Schema

At the level of the RDF data model, the vocabulary defined in the rdf namespace constitutes a collection of predefined terms intended for reuse in describing resources [50]. Terms such as rdf:type and rdf:Property are commonly employed to express instance relationships and predicate roles, while additional constructs including rdf:value, the reification vocabulary rdf:Statement, rdf:subject, rdf:predicate, and rdf:object, as well as collection constructs, support recurring representation patterns. These identifiers establish shared modeling conventions, yet they do not by themselves define class systems, hierarchies, or structural constraints. RDF Schema (RDFS) extends this initial vocabulary by introducing terms that allow such structures to be specified [39]. Notions like rdfs:Class, rdfs:Resource, rdfs:subClassOf, and rdfs:subPropertyOf provide identifiers that can be used to formulate class hierarchies and property specializations. The properties rdfs:domain and rdfs:range associate predicates with classes that characterize their admissible usage and enable the derivation of typing information. Annotation properties such as rdfs:label and rdfs:comment further allow resources to be documented in a human-readable manner. Together, the terms provided by RDF and RDFS form a combined terminology sufficient for defining lightweight ontologies that capture domain categories and their structural relations. Specifically, the official RDF and RDFS specification files are themselves expressed using this very terminology [39, 50]. Rather than being defined in a separate meta-language or plain text, the vocabularies are described by means of the same constructs that they introduce, resulting in a self-descriptive representation. Listing 2.3 illustrates how a core term such as rdf:type, originally stipulated as part of the RDF vocabulary, is characterized using descriptive terms supplied by RDF and RDFS in a structured manner.

```
rdf:type rdf:type rdf:Property ;
    rdfs:label "type" ;
    rdfs:comment "The subject is an instance of a class." ;
    rdfs:range rdfs:Class ;
    rdfs:domain rdfs:Resource .
```

Listing 2.3: RDFS-based description of the rdf:type property.

This organization yields a bootstrapped foundation in which the terminology used to model domain knowledge is also used to describe the terminology itself. The combined RDF/RDFS vocabulary therefore acts both as a representational basis and as a modeling language whose own meaning is specified within the same formalism. Statements such as the characterization of rdf:type as an instance of rdf:Property may initially appear circular. However, this reflects the deliberate reuse of the RDF/RDFS vocabulary to describe its own terms rather than the introduction of an external meta-language. Figure 2.4 illustrates this self-descriptive structure.

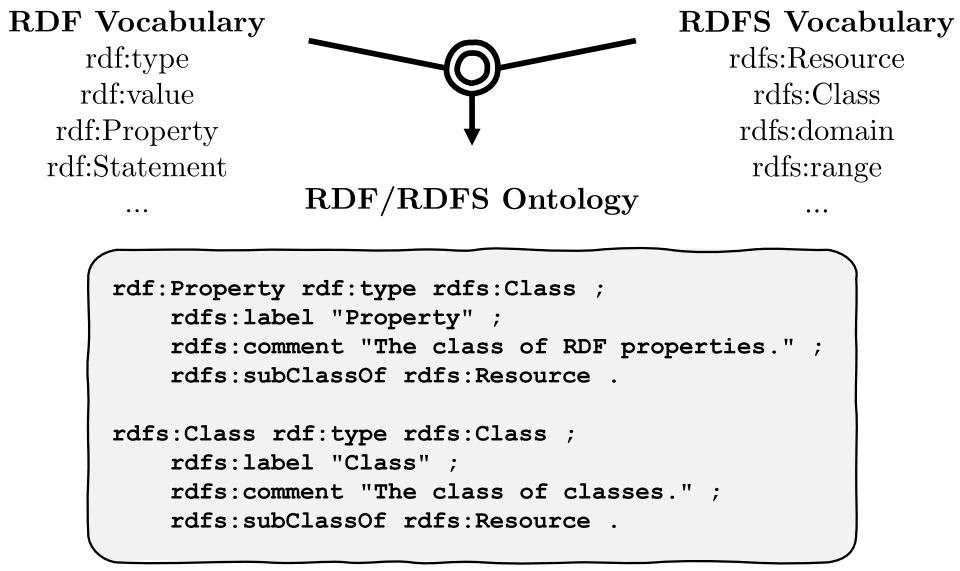


Figure 2.4: Bootstrapped organization of the RDF/RDFS terminology. The initial RDF vocabulary is extended by RDFS to express structural notions, and the combined terminology is then reused to formally describe both vocabularies, yielding a self-descriptive ontological foundation.

While RDF and RDFS are sufficient to formulate simple ontologies and taxonomic structures, their expressive power remains intentionally limited [175]. In particular, they do not provide constructs for richer logical axioms such as class equivalence, disjointness, or cardinality restrictions. To support more elaborate conceptual modeling and automated reasoning, the ontology language OWL was introduced as an extension of this RDF/RDFS foundation.

Web Ontology Language OWL

Most real-world RDF ontologies are formulated using the Web Ontology Language (OWL) [96] or one of its profiles, making it the de-facto standard for expressing domain conceptualizations on the Semantic Web [99]. OWL was introduced to overcome the limited expressivity of RDF and RDFS, which suffice for lightweight taxonomies but do not provide constructs for richer logical relationships such as class equivalence or disjointness [175].

Conceptually, OWL follows the same architectural design principle as RDFS. It introduces additional vocabulary terms while remaining fully grounded in the RDF data model. OWL therefore does not define a new representation formalism but extends the existing RDF/RDFS terminology with identifiers such as owl:Class, owl:ObjectProperty, owl:DatatypeProperty, and owl:Restriction. Crucially, OWL ontologies directly reuse the vocabulary provided by RDF and RDFS. As a consequence, model statements in OWL commonly combine terms from all three namespaces, for example by typing a resource as an owl:Class while simultaneously constraining properties using rdfs:domain and rdfs:range. OWL thus imports and builds upon the earlier terminology rather than replacing it.

The additional expressivity of OWL includes constructors for complex class expressions, such as intersections, unions, and complements, as well as quantified restrictions relating classes through properties. Furthermore, OWL provides axioms for characterizing properties, for example by declaring them transitive, symmetric, functional, or inverse-functional, and supports statements about equivalence and disjointness of classes and properties. These constructs enable automated reasoning tasks such as classification, inference of implicit relationships, and detection of logical inconsistencies. The formal semantics of these modeling primitives is based on Description Logics, which provide the logical underpinning for OWL-based reasoning systems [101].

Analogous to RDF and RDFS, the OWL vocabulary is itself specified using the same representational basis. Its defining resources are described as RDF triples that reuse the modeling terms introduced by RDF and RDFS. Listing 2.4 illustrates how a central notion such as owl:Class is characterized within this framework.

```
owl:Class rdf:type rdfs:Class ;
    rdfs:label "Class" ;
    rdfs:comment "The class of OWL classes." ;
    rdfs:subClassOf rdfs:Class .
```

Listing 2.4: Excerpt from the OWL ontology describing owl:Class.

While OWL substantially increases modeling expressivity, its semantics remain monotonic and open-world, focusing on deriving implicit knowledge rather than validating data against closed structural constraints. For applications requiring rule-based derivations or the verification of data against prescribed shapes, additional languages such as SWRL and SHACL complement the RDF/RDFS/OWL stack and are discussed next.

Extensions for Constraints and Rules

While OWL provides a highly expressive language for formulating ontologies and deriving implicit knowledge through logical inference, many applications require additional mechanisms that go beyond the semantics of Description Logics [193]. In particular, practical settings often demand the validation of data against structural expectations or the specification of rule-based derivations that are not naturally captured by OWL axioms. To address such needs, the Semantic Web ecosystem includes a range of extension languages that complement, rather than replace, the RDF/RDFS/OWL foundation [39, 50].

These extensions follow the same general design principle as the ontology languages themselves. They introduce additional vocabulary while remaining fully expressed in RDF triples and reusing the existing terminology [50]. SHACL, for example, defines terms like sh:NodeShape, sh:property, and sh:targetClass to describe structural conditions that RDF data should satisfy, including datatype restrictions, cardinalities, or path-based constraints [115]. Similarly, SWRL introduces constructs such as swrl:Imp, swrl:ClassAtom, and swrl:IndividualPropertyAtom to encode rule bodies and heads that derive new knowledge from existing assertions [102]. In both cases, the languages themselves are represented as RDF triple sets. Their elements are not intended to model domain entities but to specify constraints and derivation rules over such entities. Consequently, extensions like SHACL and SWRL do not define additional domain vocabularies in the sense of ontological classes that are instantiated in domain populations. Instead, they provide structural mechanisms that operate over ontologies formulated using RDF, RDFS, and OWL. They can thus be understood as operational extensions that refine how domain knowledge is validated or extended, while remaining grounded in the same representational formalism.

Taken together, these extensions illustrate how the Semantic Web stack incrementally enriches the shared RDF-based terminology without departing from its uniform triple representation: RDF and RDFS provide the foundational vocabulary, OWL adds expressive ontological modeling by means of Description Logics, and complementary frameworks supply validation and rule mechanisms required in practical knowledge-based applications [102, 115]. On this basis, RDF ontologies can be viewed as application-specific conceptualizations that employ RDF ontology languages to define a corresponding domain model.

2.3.3 RDF Ontologies as Domain Model

Within the layered modeling perspective adopted in this thesis, RDF ontologies expressed in RDF ontology languages such as RDF/RDFS and OWL are understood as domain models. Specifically, the vocabularies defined by RDF, RDFS, and OWL are themselves published as RDF ontologies and therefore constitute meta-ontologies that specify the modeling primitives from which domain ontologies can be constructed. In practice, a domain model is often not represented by a single ontology but by a modular collection of ontologies that describe different aspects of a domain [201]. Such ontologies may capture distinct conceptual layers, subdomains, or integration perspectives, while collectively constituting the domain model used to structure instance data [202]. In this sense, the RDF ontology languages together with domain ontologies formulated in these languages form an RDF meta-knowledge base. At this level, the ontology languages constitute the TBox, while domain ontologies appear as an ABox relative to them.

An RDF domain ontology is likewise an RDF triple set, yet its conceptual role differs from that of the ontology languages. Whereas the ontology languages provide the modeling primitives, the domain ontology defines the domain-specific vocabulary, categories, and structural constraints that organize subsequent population-level assertions. The distinction between ontology language, domain ontology, and domain population is therefore semantic and role-based rather than syntactically enforced by the RDF data model itself.
Using the terminology provided by the RDF ontology languages, domain classes are introduced as instances of the meta-class owl:Class, while domain properties are declared as instances of owl:ObjectProperty or owl:DatatypeProperty, which specialize the more general notion of rdf:Property. Although rdfs:Class already suffices to declare categories in a purely taxonomic model, owl:Class is typically used because it embeds these categories within the richer semantic framework of OWL, where they can participate in logical axioms such as disjointness, equivalence, cardinality constraints, or quantified restrictions. Formally, owl:Class is defined as a subclass of rdfs:Class. Every OWL class is therefore also an RDFS class, meaning that the OWL vocabulary extends rather than replaces the RDFS semantics by embedding class and property definitions into the Description Logic–based framework that enables automated reasoning [1].
Importantly, when an RDF domain ontology is formulated in OWL, logical entailment already applies at this terminological level. Axioms such as owl:disjointWith, rdfs:subClassOf, or domain and range declarations are interpreted under the model-theoretic semantics and may yield inferred statements independently of any instance data. In Listing 2.5, several such consequences arise already within the ontology. As a subclass of onto:Person, onto:Worker inherits all axioms applying to onto:Person, including its disjointness with onto:Organization. Since disjointness is symmetric, the axiom onto:Person owl:disjointWith onto:Organization entails the converse statement as well. Consequently, onto:Worker is also disjoint with onto:Organization. Such entailments hold independently of any instantiations and illustrate that reasoning in OWL affects the conceptual domain model.

```
@prefix onto: <https://example.org/ontology#> .

onto:Person rdf:type owl:Class ;
    owl:disjointWith onto:Organization .

onto:Worker rdf:type owl:Class ;
    rdfs:subClassOf onto:Person .

onto:Organization rdf:type owl:Class .

onto:worksFor rdf:type owl:ObjectProperty ;
    rdfs:domain onto:Worker ;
    rdfs:range onto:Organization .
```

Listing 2.5: Sample declarations of domain classes and properties using the OWL vocabulary.

The ontology fragment also implies consistency constraints. For instance, it disallows the definition of a new domain class onto:PersonOrganization as a subclass of both onto:Person and onto:Organization, since these classes are declared disjoint. In such a case, the ontology would become logically inconsistent. Consistency checks of this kind can be performed using ontology reasoning frameworks as well [1, 200, 203]. Concerning existential knowledge, however, the open-world character of OWL under the OWA becomes apparent. Suppose one intends to state that every worker must work for at least one organization. This can be expressed by an existential restriction, as shown in Listing 2.6.

```
onto:Worker rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty onto:worksFor ;
    owl:someValuesFrom onto:Organization
] .
```

Listing 2.6: OWL existential restriction stating under the OWA that every onto:Worker participates in at least one onto:worksFor relation to an onto:Organization.

Semantically, this axiom states that for every instance of onto:Worker there exists at least one onto:worksFor relation to an instance of onto:Organization. Under the OWA adopted by OWL, however, this statement is not interpreted as a data completeness requirement. If an individual is asserted to be an onto:Worker without an explicit onto:worksFor triple, this does not imply inconsistency. Instead, it is assumed that such a relation exists in the modeled world even if it is not currently represented. Missing information is therefore treated as unknown rather than false. OWL axioms describe necessary conditions in the intended interpretation of the domain, not mandatory patterns that must be explicitly present in a concrete dataset [193]. If, in contrast, one wishes to enforce that every recorded worker must explicitly be linked to at least one organization via onto:worksFor, constraint languages such as SHACL are required [115]. SHACL evaluates RDF triples under a validation-oriented, closed-world perspective in which absent triples are treated as violations rather than as unknown facts. The SHACL shape shown in Listing 2.7 complements the ontology by introducing such a validation requirement.

Accordingly, every instance of onto:Worker is required to have at least one onto:worksFor property pointing to an entity of onto:Organization. If such a triple is missing, the assertion violates the shape. Unlike OWL, which enriches domain knowledge through logical entailment under open-world semantics, SHACL validates instantiations by checking whether explicitly stated triples satisfy structural constraints. The coexistence of OWL and SHACL thus reflects two complementary perspectives within the same RDF framework. OWL provides a Description Logic–based method for modeling domain knowledge and deriving implicit facts, while SHACL introduces validation constraints that regulate the admissible structure of instance data. Both are expressed in RDF and reuse its triple-based representation, yet they serve distinct conceptual roles: ontological meaning and logical entailment on the one hand, and constraint-level validation under a closed-world interpretation on the other.

```
@prefix sh:   <http://www.w3.org/ns/shacl#> .

onto:WorkerShape
    rdf:type sh:NodeShape ;
    sh:targetClass onto:Worker ;
    sh:property [
        sh:path onto:worksFor ;
        sh:class onto:Organization ;
        sh:minCount 1 ;
    ] .
```

Listing 2.7: SHACL constraint introducing a closed-world validation requirement on worker instances, complementing the open-world, inference-oriented semantics of OWL.

2.3.4 RDF Triple Sets as Domain Population

Beyond schema-level reasoning, the ontology fragment in Listing 2.5 also induces entailments for instance data. For example, given an assertion

$$(x, \text{onto:worksFor}, y),$$

the declared domain and range imply that x is an instance of onto:Worker and y an instance of onto:Organization. Since onto:Worker is a subclass of onto:Person, it further follows that x is also an instance of onto:Person. Thus, the ontology not only provides a domain-specific vocabulary but also enriches explicitly stated instance facts through logical entailment [1, 96].
In general, when concrete instance data are introduced, we move to the perspective of RDF domain knowledge bases. In this setting, the domain model is represented by a domain ontology $\mathcal{O}$ (possibly composed of multiple ontology modules) formulated using an RDF ontology language $\mathcal{L}$, which assumes the terminological role in the respective meta-knowledge base. The assertional component consists of an RDF triple set as a collection of facts $\mathcal{F}$ describing the domain population by reusing the vocabulary provided by the ontology $\mathcal{O}$. Since RDF uses a uniform representation, the population, the domain ontology, and the ontology language are all encoded as RDF triple sets, i.e.,

$$\mathcal{F}, \mathcal{O}, \mathcal{L} \subseteq \mathfrak{R} \times \mathfrak{I} \times (\mathfrak{R} \cup \mathfrak{L}).$$

Instantiable classes and predicates are specified within the ontology as resource arguments:

$$\mathcal{C}_{\mathcal{O}}, \mathcal{P}_{\mathcal{O}} \subseteq \mathcal{A}_{\mathfrak{R}}(\mathcal{O}),$$

where $\mathcal{P}_{\mathcal{O}}$ can be subdivided into resource and literal predicates $\mathcal{P}_{\mathcal{O},\mathfrak{R}}, \mathcal{P}_{\mathcal{O},\mathfrak{L}} \subseteq \mathcal{P}_{\mathcal{O}}$. Accordingly, the induced predicate set $\mathcal{P}(\mathcal{F})$ as per Section 2.3.1 must satisfy

$$\mathcal{P}(\mathcal{F}) \subseteq \mathcal{P}_{\mathcal{O}} \cup \{\text{rdf:type}\}.$$

Notice that this does not exclude the use of general-purpose, language-level predicates (e.g., annotation properties), since the ontology may incorporate such terms through overlap with the

language vocabulary, i.e., $\mathcal{O} \cap \mathcal{L} \neq \emptyset$ may hold. In particular, rdf:type plays a special role in an RDF knowledge base, as it provides the only explicit structural link from the population to the ontology by connecting individuals to ontology-defined classes [50].
Analogous to predicates, let the set of classes induced by $\mathcal{F}$ be given by

$$\mathcal{C}(\mathcal{F}) := \{\, o \mid (s, \text{rdf:type}, o) \in \mathcal{F} \,\} \subseteq \mathcal{C}_{\mathcal{O}},$$

implying the set of entities described in the population

$$\mathcal{E}(\mathcal{F}) := \mathcal{A}_{\mathfrak{R}}(\mathcal{F}) \setminus \mathcal{C}(\mathcal{F}).$$

Using these notions, the resource-valued subset $\mathcal{F}_{\mathfrak{R}}$ can be decomposed into typing links and entity-to-entity relations with respect to domain resource properties defined by $\mathcal{O}$, i.e.,

$$\mathcal{F}_{\mathcal{C}} \subseteq \mathcal{E}(\mathcal{F}) \times \{\text{rdf:type}\} \times \mathcal{C}_{\mathcal{O}}, \qquad \mathcal{F}_{\mathcal{E}} \subseteq \mathcal{E}(\mathcal{F}) \times \mathcal{P}_{\mathcal{O},\mathfrak{R}} \times \mathcal{E}(\mathcal{F}),$$

such that $\mathcal{F}_{\mathfrak{R}} = \mathcal{F}_{\mathcal{C}} \uplus \mathcal{F}_{\mathcal{E}}$ holds. In accordance with the population-level abstraction introduced above, class identifiers thus occur in $\mathcal{F}$ exclusively as objects of rdf:type triples. Moreover, this yields the simplified characterization of the literal-valued subset $\mathcal{F}_{\mathfrak{L}} \subseteq \mathcal{F}$:

$$\mathcal{F}_{\mathfrak{L}} \subseteq \mathcal{E}(\mathcal{F}) \times \mathcal{P}_{\mathcal{O},\mathfrak{L}} \times \mathfrak{L}.$$

In summary, $\mathcal{F}$ constitutes the ABox of an RDF knowledge base by describing individuals as entities through the reuse of vocabulary specified in a corresponding RDF domain ontology $\mathcal{O}$, which functions as the TBox. Entities are explicitly linked to class types via rdf:type, related to each other via resource predicates, and enriched with literal-valued attributes.
Figure 2.5 schematically illustrates this decomposition of an RDF domain population into literal annotations, typing links, and entity-level relations with respect to the underlying ontology.

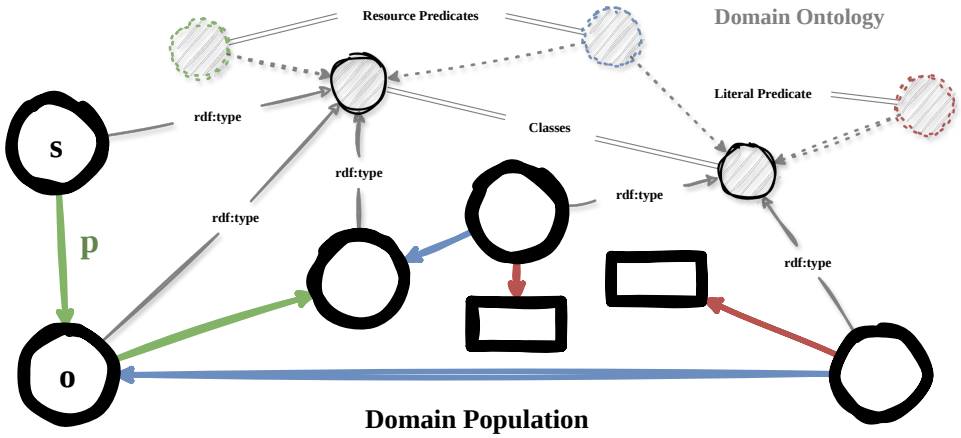


Figure 2.5: Illustration of an RDF knowledge base composed of entity-level knowledge (bold) and terminological knowledge (grey), connected via rdf:type links and reuse of predicates.

In the following, we motivate why RDF-encoded domain populations are commonly referred to as RDF knowledge graphs by interpreting RDF triple sets as graph structures. This viewpoint subsequently enables the application of graph representation learning methods to RDF data.

Chapter 3

RDF Knowledge Graphs

Graphs provide a flexible formalism for representing structured information, and many real-world datasets naturally admit a graph-based representation [12]. For instance, BPMN process models can be represented via directed edges between flow objects that reflect execution order. Analogously, RDF triples induce edges between their arguments, thereby forming a predicate-labeled graph. As a fundamental organizational principle, graph-structured data is often inherently interpretable, particularly when complemented by an underlying schema specifying the semantics of vertices and edges [250]. In the following, we introduce graphs from a general mathematical perspective, emphasizing edge incidences and labels as key structural characteristics used throughout this work. In this context, the term knowledge graph is linked to graph-structured data representing real-world entities enriched with semantics through an explicit domain conceptualization. Although the term is used in different ways in the literature, it is commonly associated with graphs induced by RDF data, typically constructed from instance-level RDF statements grounded in an accompanying ontology, even if this assumption remains implicit. Building on the layered knowledge modeling perspective developed in Chapter 2, we make this interpretation explicit. In this thesis, we thus understand an RDF knowledge graph as the directed, predicate-labeled graph induced by the ABox of an RDF domain knowledge base, while the corresponding RDF domain ontology provides the TBox that defines the admissible vocabulary of classes and predicates. Finally, we introduce a generalized framework for time-aware knowledge graphs, incorporating time both as local edge metadata and as a global dimension that governs the evolution of domain knowledge.

3.1 Graph-Theoretic Foundations

First, we formally introduce graphs from an abstract mathematical perspective, thereby providing a model-independent foundation for the subsequent formalization of RDF graphs and graph representation learning methodologies. Our aim is to isolate the structural core shared by all graph-based representations without committing to a particular data model such as the one imposed by RDF. In this sense, a graph is understood as an abstract structure that other systems may instantiate by supplying appropriate sets of vertices, edge labels, and asserted edges [231].

Definition 1. *Let $\mathcal{V}$ be a non-empty set of vertices (also called nodes) and let $\mathcal{M}$ be a non-empty set of edge metadata (also called labels). A graph over $(\mathcal{V},\mathcal{M})$ is a subset*

$$\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M},$$

and $(v,u,m) \in \mathcal{G}$ denotes an edge from $v \in \mathcal{V}$ to $u \in \mathcal{V}$ with metadata $m \in \mathcal{M}$. We define

$$\mathcal{V}(\mathcal{G}) := \{v \mid (v,u,m) \in \mathcal{G}\} \cup \{u \mid (v,u,m) \in \mathcal{G}\}, \quad \mathcal{M}(\mathcal{G}) := \{m \mid (v,u,m) \in \mathcal{G}\}$$

as the induced vertices and metadata. A graph $\mathcal{G}$ is said to cover $(\mathcal{V},\mathcal{M})$ if and only if

$$\mathcal{V}(\mathcal{G}) = \mathcal{V}, \quad \mathcal{M}(\mathcal{G}) = \mathcal{M}$$

holds, i.e., if every $v \in \mathcal{V}$ and every $m \in \mathcal{M}$ occur in some edge of $\mathcal{G}$.

Definition 1 admits a natural knowledge base interpretation as the pair $(\mathcal{V},\mathcal{M})$ specifies a graph-specific vocabulary, such that each graph $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M}$ represents a set of asserted facts over this vocabulary. However, $(\mathcal{V},\mathcal{M})$ itself does not provide any semantic grounding. Still, as shown later, this notation is fully compatible with RDF graphs and further allows us to encode special graph types within the same setting, such as directed and undirected graphs.

Definition 2. *A graph $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M}$ is called undirected if and only if it is required to be symmetric in its first two components, that is,*

$$(v,u,m) \in \mathcal{G} \iff (u,v,m) \in \mathcal{G}.$$

Otherwise, it is called directed.

In contrast to some related work [100], we do not model directedness as an intrinsic property of individual edges. Instead, directedness is defined at the level of the graph structure, while edges are uniformly represented as ordered tuples. In this sense, what is commonly viewed as an undirected edge corresponds in our setting to a pair of edges with exchanged endpoints.
Besides directedness, we also consider whether graphs carry distinguishable edge labels. In this context, we denote singleton sets by $\{*\}$, where $*$ represents a trivial element.

Definition 3. *Let $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M}$ be a graph. We call $\mathcal{G}$ homogeneous if $\mathcal{M} = \{*\}$ is a singleton set. In this case, no edge labels are distinguished and every edge is of the form*

$$(v,u,*).$$

A graph is called heterogeneous if it is not homogeneous, i.e., if $|\mathcal{M}| \geq 2$ holds.

Unlabeled graphs are often defined as subsets of $\mathcal{V} \times \mathcal{V}$ without an explicit label set. However, without loss of generality, such graphs can be represented in the framework of Definition 1 by introducing a singleton set $\mathcal{M} = \{*\}$, so that every edge takes the form $(v, u, *)$ and carries no additional information. Finally, when edge labels represent relation types between vertices, the graph is also referred to as a relational graph. The distinction between generalized graphs and relational graphs is mostly semantic. By assuming a relational graph, we consider a countable set of edge labels that carry relational meaning and are not permutation-invariant.

The preceding definitions describe a graph from a global perspective, specifying its structure as a whole through collections of vertices and edges. In many settings, however, it is necessary to adopt a local viewpoint centered at a particular vertex $v \in \mathcal{V}$. Such a perspective enables capturing structural information directly associated with a vertex, including incident edges and adjacent vertices. To formalize this localized view uniformly across directed, undirected, homogeneous, and heterogeneous graphs, we introduce the notions of vertex stubs and neighbors.

Definition 4. *Let $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M}$ be a graph. If $\mathcal{G}$ is directed, let $\mathcal{I} = \{\iota_{\text{out}}, \iota_{\text{in}}, \iota_{\text{id}}\}$ denote the set of admissible incidences (outgoing, incoming, and identity for self-loops). The incidence expansion $\mathcal{G}^+ \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M} \times \mathcal{I}$ of $\mathcal{G}$ is defined as*

$$\mathcal{G}^+ := \bigcup_{(v,u,m)\in\mathcal{G}} \begin{cases} \{(v,u,m,\iota_{\text{out}}), (u,v,m,\iota_{\text{in}})\}, & v \neq u, \\ \{(v,v,m,\iota_{\text{id}})\}, & v = u. \end{cases}$$

A directed edge between distinct vertices $v \neq u$ thus induces two entries in $\mathcal{G}^+$, while a self-loop induces a single entry. If $\mathcal{G}$ is homogeneous, then $\mathcal{M} = \{*\}$ holds. Analogously, if $\mathcal{G}$ is undirected, let $\mathcal{I} = \{*\}$, which simplifies $\mathcal{G}^+$ to

$$\mathcal{G}^+ := \{(v,u,m,*) \mid (v,u,m) \in \mathcal{G}\}.$$

For $v \in \mathcal{V}$, a tuple (u,m,ι) with $(v,u,m,\iota) \in \mathcal{G}^+$ is called a stub incident to *v. Accordingly, the set of all stubs incident to v is defined as*

$$\mathcal{S}_{\mathcal{G}}(v) := \{(u,m,\iota) \mid (v,u,m,\iota) \in \mathcal{G}^+\} \subseteq \mathcal{V} \times \mathcal{M} \times \mathcal{I}.$$

Given $\mathcal{S}_{\mathcal{G}}(v)$, we define the set of neighbors of $v \in \mathcal{V}$ as the vertices adjacent to it:

$$\mathcal{N}_{\mathcal{G}}(v) := \{u \mid (u,m,\iota) \in \mathcal{S}_{\mathcal{G}}(v)\} \subseteq \mathcal{V}.$$

Accordingly, a stub $s = (u,m,\iota) \in \mathcal{S}_{\mathcal{G}}(v)$ states that $v \in \mathcal{V}$ is connected to $u \in \mathcal{V}$ via $m \in \mathcal{M}$ with incidence $\iota \in \mathcal{I}$. In the following, we show how RDF triple sets can be interpreted as directed graphs with predicate types as edge labels, and under which conditions they represent knowledge graphs. Further, we introduce a generalized framework for time-aware graph extensions.

3.2 From RDF Triple Sets to RDF Knowledge Graphs

The abstract notion of a graph introduced in Definition 1 provides a natural foundation for relating formal graph structures to concrete data models. Many systems induce graphs by specifying how entities correspond to vertices and how statements correspond to edges. In this context, the term knowledge graph is widely used, yet no universally accepted definition exists. Broadly speaking, knowledge graphs describe real-world entities and their relationships as graph structures grounded in a shared domain conceptualization [99, 160, 168]. In this view, entities correspond to vertices and relationships to edges, while the conceptualization constrains their interpretation and highlights the relational nature of knowledge graphs.
In practice, the term knowledge graph is most commonly associated with graph representations induced by RDF triple sets [66, 252], reflecting both their inherent graph-like structure and their close connection to conceptualized knowledge representations based on RDF ontologies. Building on the layered knowledge modeling perspective developed in Chapter 2, we make explicit what often remains implicit in this usage: a knowledge graph arises as the graph induced by the ABox of an RDF domain knowledge base. First, if $\mathcal{F}$ constitutes the ABox of an RDF knowledge base with domain ontology $\mathcal{O}$, then the triples in $\mathcal{F}$ encode instance-level assertions with respect to $\mathcal{O}$. Second, each RDF triple set $\mathcal{F} \subseteq \mathfrak{R} \times \mathfrak{I} \times (\mathfrak{R} \cup \mathfrak{L})$ with induced arguments $\mathcal{A}(\mathcal{F}) \subseteq \mathfrak{R} \cup \mathfrak{L}$ and predicates $\mathcal{P}(\mathcal{F}) \subseteq \mathfrak{I}$ canonically induces a directed graph. Namely, as established in Section 2.3.1, every RDF triple set satisfies

$$\mathcal{F} \subseteq \mathcal{A}(\mathcal{F}) \times \mathcal{P}(\mathcal{F}) \times \mathcal{A}(\mathcal{F}).$$

By identifying each triple $(s,p,o) \in \mathcal{F}$ with a labeled edge (s,o,p), we obtain the RDF graph

$$\mathcal{G}(\mathcal{F}) \subseteq \mathcal{A}(\mathcal{F}) \times \mathcal{A}(\mathcal{F}) \times \mathcal{P}(\mathcal{F}),$$

covering its induced arguments $\mathcal{V} = \mathcal{A}(\mathcal{F})$ and predicates $\mathcal{M} = \mathcal{P}(\mathcal{F})$ as labels. In particular, every RDF graph is a graph over $(\mathfrak{R} \cup \mathfrak{L}, \mathfrak{I})$, while respecting the constraints imposed by the RDF data model, such as the restriction that literals may only appear as objects. In accordance with this graph interpretation, we adopt the terminology of an RDF knowledge graph for representing instance-level domain knowledge, analogous to the requirements outlined in [99].

Definition 5. *Let $(\mathcal{O}, \mathcal{F})$ be an RDF domain knowledge base, where $\mathcal{O}$ denotes the domain ontology and $\mathcal{F}$ the domain population. In this case, the graph $\mathcal{G}(\mathcal{F})$ induced by $\mathcal{F}$ is called an RDF knowledge graph with respect to the domain ontology $\mathcal{O}$.*

Definition 5 aligns with the abstract requirements outlined above. As the ABox of an RDF knowledge base, $\mathcal{F}$ captures knowledge about domain entities $\mathcal{E}(\mathcal{F})$ by means of

1. relations between individuals through resource-valued properties $\mathcal{P}_{\mathfrak{R}}(\mathcal{F}) \subseteq \mathcal{P}_{\mathcal{O},\mathfrak{R}}$,
2. literal-valued annotations through literal properties $\mathcal{P}_{\mathfrak{L}}(\mathcal{F}) \subseteq \mathcal{P}_{\mathcal{O},\mathfrak{L}}$,
3. type assertions to ontology-defined classes $\mathcal{C}(\mathcal{F}) \subseteq \mathcal{C}_{\mathcal{O}}$ via rdf:type.

As components of RDF knowledge bases, RDF knowledge graphs are therefore considered with respect to a given domain ontology. In this work, we restrict ourselves to such knowledge graphs encoded in RDF. Thus, throughout this work, we use the term knowledge graph to refer to RDF knowledge graphs. Moreover, we refer to a knowledge graph as a relational knowledge graph if $\mathcal{F} = \mathcal{F}_{\mathfrak{R}}$ holds, implying that edges either correspond to class assertions or relations between entities. Finally, we assume the ontology to be non-trivial, as the trivial ontology induced by an RDF graph $\mathcal{F}$ would merely introduce $|\mathcal{C}(\mathcal{F})| + |\mathcal{P}(\mathcal{F})| - 1$ declarations stating that the induced classes and predicates are OWL classes and RDF properties (excluding rdf:type).
In the following, we use the two equivalent RDF data views interchangeably, writing $\mathcal{F} \cong \mathcal{G}(\mathcal{F})$ whenever no confusion can arise. When referring to $\mathcal{F}$ as an RDF triple set, we consider the ordering (s, p, o), whereas when referring to it as an RDF graph, we adopt the ordering (s, o, p).

3.3 Temporal Extensions of RDF Knowledge Graphs

> Parts of this section are based on the publication *"On a Generalized Framework for Time-Aware Knowledge Graphs"* by Franz Krause, Tobias Weller, and Heiko Paulheim [124], published in *Towards a Knowledge-Aware AI* (IOS Press, 2022), which received the Best Research Paper Award.

Building upon the formalization of knowledge graphs as RDF triple sets constituting the ABox of an RDF knowledge base, we now turn to extensions that account for the temporal dimension of domain knowledge. In many real-world applications, such as industrial scenarios, domain knowledge is not static but evolves over time. Facts may become valid or invalid, new information may be added, and historical states may need to remain traceable. Consequently, a purely static view of knowledge graphs is often insufficient for modeling time-dependent domains. In the following, we ground temporal extensions of knowledge graphs on the generalized graph framework introduced in Section 3.1 and subsequently adapt it to the characteristics of RDF.
In the context of knowledge graph-based downstream applications, there has been increasing interest in incorporating temporal information directly into representation learning and handling dynamic knowledge graph updates [91, 236, 242]. However, the corresponding literature exhibits substantial terminological inconsistencies. In particular, notions such as *stationary*, *static*, *temporal*, and *dynamic* knowledge graphs are frequently used interchangeably, despite referring to conceptually distinct settings. A key distinction concerns the objective of time-awareness. On the one hand, time may describe the validity period of a fact, indicating the time period during which a triple holds in the modeled domain. On the other hand, time may capture the dynamics of the domain, reflecting when a fact was added, modified, or removed within the knowledge graph. These two perspectives lead to fundamentally different extensions of the underlying graph structure. In the following, we systematically distinguish between local and global temporal extensions of knowledge graphs, highlighting their orthogonality and differing requirements regarding time-awareness. By doing so, we provide a precise and unified terminology for this thesis and for research in the context of time-aware knowledge graph modeling more broadly, thereby facilitating clearer conceptualization and improved comparability in future research.

3.3.1 Time-Awareness in RDF Knowledge Graphs

The RDF data model naturally permits the representation of temporal information about resources via time-related literal values such as timestamps [50, 99]. Such temporal values are typically expressed as typed literals, e.g., "2026-02-11T09:00:00" with datatype xsd:dateTime, and linked via predicates defined in dedicated RDF ontologies. Prominent examples include schema.org [87], Dublin Core Terms [62], and OWL-Time [49], each providing a vocabulary for representing temporal aspects of domain entities. To highlight the structural differences between these ontological approaches, the same temporal information is encoded below using all three ontologies. In each case, a task is associated with an employee and spans the time interval from 09:00 to 17:00. As shown in Listing 3.1, the modeling approaches differ substantially.

```
# TBOX
@prefix schema:  <https://schema.org/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix time:    <http://www.w3.org/2006/time#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .

# ABOX
@prefix kg: <http://example.org/abox#> .

kg:task_3ou4gt a schema:Action ;
    schema:agent kg:employee_232so9 .

# (1) schema.org: temporal literals as attributes
kg:task_3ou4gt  schema:startDate "2026-02-11T09:00:00"^^xsd:dateTime ;
                schema:endDate   "2026-02-11T17:00:00"^^xsd:dateTime .

# (2) Dublin Core Terms: explicit PeriodOfTime
kg:task_3ou4gt dcterms:temporal [
    a dcterms:PeriodOfTime ;
    dcterms:start "2026-02-11T09:00:00"^^xsd:dateTime ;
    dcterms:end   "2026-02-11T17:00:00"^^xsd:dateTime ] .

# (3) OWL-Time: axiomatized interval structure
kg:task_3ou4gt time:hasTime kg:interval_1 .
kg:interval_1 a time:ProperInterval ;
    time:hasBeginning [
        a time:Instant ;
        time:inXSDDateTime "2026-02-11T09:00:00"^^xsd:dateTime ] ;
    time:hasEnd [
        a time:Instant ;
        time:inXSDDateTime "2026-02-11T17:00:00"^^xsd:dateTime ] .
```

Listing 3.1: Three ontological approaches to encoding the same temporal information.

Although these implementations differ in expressivity and formal grounding, they share a common modeling principle, as temporal information is attached to resources rather than to individual RDF statements. The relation between task and employee remains unchanged, while the temporal extent is modeled as metadata of the task entity. Thus, time is treated as an attribute of entities rather than as a qualifier of relational assertions. In many applications, however, the relevant question concerns the validity of a specific relation. For example, an employee may act as the schema:agent of a task only during a subinterval of its overall duration. Such temporally qualified assertions cannot be expressed by annotating participating entities alone. Instead, they must qualify the statement itself. This requirement has led to modeling constructs such as *n*-ary relation patterns, statement-level annotations, and graph versioning approaches. A classical solution is RDF reification [50], where a statement is represented as a resource of type rdf:Statement to which additional metadata can be attached, as shown in Listing 3.2.

```
kg:stmt_4711 a rdf:Statement ;
    rdf:subject   kg:task_3ou4gt ;
    rdf:predicate schema:agent ;
    rdf:object    kg:employee_232so9 ;
    schema:startDate "2026-02-11T10:00:00"^^xsd:dateTime ;
    schema:endDate   "2026-02-11T12:00:00"^^xsd:dateTime .
```

Listing 3.2: Statement-level temporal qualification using RDF reification.

While reification enables statement-level qualification, it introduces structural overhead by replacing original triples with multiple descriptive triples. To counteract this issue, RDF-star was introduced as a more compact alternative [94], allowing triples to be directly referenced and annotated without introducing explicit statement nodes, as shown in Listing 3.3.

```
« kg:task_3ou4gt schema:agent kg:employee_232so9 »
    schema:startDate "2026-02-11T10:00:00"^^xsd:dateTime ;
    schema:endDate   "2026-02-11T12:00:00"^^xsd:dateTime .
```

Listing 3.3: Statement-level temporal qualification using RDF-star.

Large collaborative knowledge graphs such as Wikidata [227] follow closely related paradigms in which statements are treated as first-class objects enriched with qualifiers and references. Approaches that attach timestamps or intervals directly to triples are commonly referred to as temporal knowledge graphs and represent a local form of time-awareness, as temporal information qualifies individual relational assertions.

In contrast, a second dimension of time-awareness concerns the global evolution of domain knowledge as a whole. Here, time does not qualify individual facts but instead indexes structural changes such as the insertion, deletion, or modification of triples. This perspective is typically operationalized through sequences of graph snapshots [212, 233, 236] and is frequently described in the literature under the label of dynamic knowledge graphs, but also referred to as

temporal knowledge graphs [77, 237]. The terminology is therefore often used without clearly distinguishing between fact-level temporal qualification and graph-level evolution. From a formal perspective, however, a graph considered in isolation constitutes a timeless mathematical structure, in the sense that it does not carry an external temporal parameter. Consequently, global time-awareness cannot be attributed to a single graph but arises only when graphs are organized into ordered families representing successive states of knowledge. This conceptual distinction motivates the need for a unified and formally precise specification of time-awareness in knowledge graphs. In the following section, we therefore introduce temporality separately for edge-level metadata and for graph trajectories, first defining both notions for generalized graphs and subsequently adapting them to RDF graphs, including RDF knowledge graphs.

3.3.2 Temporal RDF Knowledge Graphs

In the context of generalized graphs $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M}$ according to Definition 1, temporal graphs arise when the edge metadata space $\mathcal{M}$ contains temporal information $\mathcal{T}$, i.e., when edge labels are of the form $\mathcal{M} = \mathcal{T} \times \mathcal{M}'$, where $\mathcal{T}$ denotes temporal metadata and $\mathcal{M}'$ identifies additional, non-temporal metadata such as predicate types. Given a totally ordered time domain $\mathbb{T}$, such as $\mathbb{T} \subseteq \mathbb{N}$ or $\mathbb{T} \subseteq \mathbb{R}$, different types of temporal edge metadata can be constructed. We restrict ourselves to validity periods, i.e., $\mathcal{T} = \{(t_s, t_e) \in \mathbb{T} \times \mathbb{T} \mid t_s \leq t_e\}$ with start and end timestamps. If necessary, the entries $\{-\infty, \infty\}$ may be included to represent open intervals. When specializing this notion to RDF triple sets

$$\mathcal{F} \subseteq \mathcal{A}(\mathcal{F}) \times \mathcal{P}(\mathcal{F}) \times \mathcal{A}(\mathcal{F}),$$

an essential structural restriction arises, as edge metadata consists solely of predicate types. Consequently, core RDF graphs cannot attach additional metadata to relational assertions while preserving the native subject–predicate–object structure. In particular, temporal qualification of relations cannot be expressed at the level of plain triples. Hence, an RDF graph in its native form does not constitute a temporal graph in the sense defined above.

If an ontology provides temporal constructs for resources, such as instants, intervals, or validity predicates, these constructs can readily be used to describe the temporal properties of entities. However, using the same constructs to describe the temporal validity of a relation requires transforming the relation itself into a resource. In core RDF, this is achieved by reification, that is, by representing a triple $(s, p, o) \in \mathcal{F}$ as an explicit statement resource to which additional properties can be attached. Once relational assertions are reified, temporal predicates defined in the ontology can be used to associate validity intervals with these statement resources. Temporal metadata is thus no longer attached to entities but to representations of relations.

Although RDF graphs cannot directly induce temporal graphs by construction, we refer to an RDF graph as temporal if it contains reified statements enriched with temporal information from which a corresponding temporal graph representation can be obtained. In practice, approaches described as temporal RDF knowledge graphs frequently rely on extensions such as RDF-star [94] or comparable statement-level annotation mechanisms, which provide more compact syntactic support for attaching metadata to triples.

Temporal annotations must be interpreted with respect to a single temporal perspective. In this section, we restrict ourselves to validity time, i.e., the time during which a fact holds in the

modeled domain. Introducing additional dimensions, such as insertion or deletion time, would yield a multi-temporal setting requiring cross-perspective reasoning (e.g., determining when it became known that a validity period ended), which is beyond the scope of this thesis. We do not further elaborate on the technical realization of such local temporal extensions, which is discussed in detail in [124]. Instead, our objective is to contrast this local temporality, where time qualifies individual assertions, with the global notion introduced next, where time indexes successive graph states in a trajectory. Since such trajectories are often labeled both dynamic and temporal graphs in the literature, we resolve this terminological overloading by explicitly separating local temporal qualification from global temporality defined via graph trajectories.

3.3.3 Temporal Trajectories of RDF Knowledge Graphs

Graphs $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M}$ in the sense of Definition 1 are timeless mathematical objects. Timelessness here refers to the absence of a temporal parameter in the graph model itself and does not preclude the graph from containing time-related information encoded as resource attributes or as part of its edge metadata, as in temporal graph variants [77, 237]. Accordingly, RDF graphs, including RDF ontologies and knowledge graphs, are likewise timeless when considered in isolation. In practical applications, however, domain knowledge evolves, i.e., information is inserted, revised, or removed, and past states of knowledge may need to remain accessible. This evolution does not occur within a single graph, but across successive states of knowledge. Attempting to represent multiple such states within one timeless graph either leads to inconsistencies or obscures the temporal validity of facts. For instance, representing successive office holders of the presidency of the US in a single graph without temporal separation conceals when each fact was valid. To formally capture such evolution, we therefore introduce the notion of graph trajectories, which model ordered families of graph states indexed by a parameter.

Definition 6. *Let $\mathbb{I}$ be a totally ordered index set. A* trajectory of graphs *over $\mathbb{I}$ is a family*

$$(\mathcal{G}_i)_{i \in \mathbb{I}}$$

of graph states $\mathcal{G}_i \subseteq \mathcal{V}_i \times \mathcal{V}_i \times \mathcal{M}_i$. The trajectory is called static *if and only if all graph states coincide, i.e., $\mathcal{G}_i = \mathcal{G}_j$ for all $i, j \in \mathbb{I}$, and* dynamic *otherwise.*

If the index set carries a temporal interpretation, i.e., if $\mathbb{I} = \mathbb{T}$ is a totally ordered time domain such as $\mathbb{T} \subseteq \mathbb{N}$ or $\mathbb{T} \subseteq \mathbb{R}$, the trajectory is referred to as a temporal trajectory. In this case, each graph state $\mathcal{G}_t$ represents the domain knowledge available at time $t \in \mathbb{T}$. In the case of temporal trajectories, we also refer to graph states as snapshots. In particular, this formalization resolves a common ambiguity in the literature, where individual graphs are described as static or dynamic, although such attributes are not defined for timeless graph objects. Implicitly, these works refer to trajectories of graphs rather than to single graph instances.

The abstraction of Definition 6 applies directly to RDF graphs. However, since an RDF knowledge graph is defined as the ABox of an RDF knowledge base with respect to a corresponding ontology, temporal evolution must be considered at the level of the entire knowledge base.

Definition 7. *An RDF knowledge base trajectory is a family*

$$\big((\mathcal{O}_i, \mathcal{F}_i)\big)_{i \in \mathbb{I}}$$

over a totally ordered index set $\mathbb{I}$, where each pair $(\mathcal{O}_i, \mathcal{F}_i)$ constitutes an RDF knowledge base and induces a corresponding trajectory of RDF knowledge graphs

$$(\mathcal{F}_i)_{i \in \mathbb{I}}.$$

Within this framework, it is essential to distinguish whether evolution affects only the assertional component or also the ontological component. If the ontology remains fixed, i.e., $\mathcal{O}_i = \mathcal{O}_j$ for all $i, j \in \mathbb{I}$, the trajectory captures evolving instance-level knowledge under a stable conceptualization. If the ontology itself varies, even unchanged assertional fragments may acquire a different interpretation under the modified TBox, i.e., static knowledge graph trajectories may require dynamic interpretation.

Most real-world RDF knowledge graph systems represent temporal trajectories in the sense of Definition 7. Large collaborative graphs such as DBpedia [17] or Wikidata [227] continuously evolve and provide publicly available snapshots. A query evaluated at different times may therefore yield different results, even if the query itself remains unchanged. Nevertheless, in practice the term knowledge graph is often used both for a single snapshot and for the evolving sequence of such snapshots without explicitly distinguishing between these perspectives. Hence, when the term dynamic knowledge graph is used in the literature, it typically refers to a trajectory of RDF knowledge graphs whose assertional component changes over time. The framework introduced here makes this implicit understanding explicit by separating timeless graph structures from their temporal indexing and by distinguishing trajectories from locally temporalized graphs.

3.3.4 Integrating Local and Global Temporality

Local and global time-awareness constitute orthogonal extensions of graph representations, including knowledge graphs. Local temporality qualifies individual assertions with respect to the time they describe, whereas global temporality indexes successive graph states according to when knowledge becomes available. Therefore, temporality concerns the validity of individual assertions, while dynamics concern the evolution of graph states across a trajectory.

When both aspects are present, two distinct temporal domains must be distinguished: one indexing the evolution of the knowledge base and one describing the temporal dimension of the modeled reality. In general, we denote these domains by

$$\mathbb{T}_{\text{dyn}} \quad \text{and} \quad \mathbb{T}_{\text{temp}},$$

respectively. However, in this work we focus on temporal trajectories of atemporal RDF knowledge graphs. Temporal information produced during system execution is therefore treated as

evolving instance-level information without reifying assertional facts. The emphasis lies on modeling RDF knowledge base evolution rather than on fact-level temporal qualification.
The distinctions introduced above provide a useful perspective for structuring related work on temporal extensions of knowledge graphs. A temporal graph, in the sense considered here, remains a timeless graph object whose edge metadata incorporates temporal information describing the modeled domain. In contrast, temporal trajectories of graphs capture the global evolution of knowledge by organizing successive graph states along an ordered temporal index. This distinction also clarifies common terminology in the literature. What is often described as a *dynamic knowledge graph* typically refers to a trajectory of knowledge graphs whose states evolve over time, rather than to a single graph instance. Conversely, what is referred to as a *static knowledge graph* usually corresponds to a single timeless graph instance, which may equivalently be viewed as part of a static trajectory in which all graph states coincide. Thus, when speaking of dynamic or static graphs in this work, we refer to trajectories of graph objects.

Chapter 4

Graph Representation Learning

As described in Chapter 3, graph-structured data provides a natural, inherently interpretable format for representing domain knowledge, particularly when accompanied by an underlying conceptualization of the domain, such as RDF knowledge graphs, which represent assertions with respect to corresponding RDF ontologies. However, their semantics are symbolic and therefore do not inherently induce a metric structure suitable for similarity comparison, gradient-based optimization, or statistical generalization [160]. To bridge this gap, graph representation learning studies how symbolic graph structures can be mapped into continuous vector spaces [42, 84]. The resulting numerical embeddings provide vector representations of symbolic inputs that can be used in downstream learning and optimization tasks. We begin with a general notion of numerical embeddings.

Definition 8. *Let $\mathcal{X}$ be a set and $d \in \mathbb{N}$. A d-dimensional embedding of $\mathcal{X}$ is a mapping*

$$\phi : \mathcal{X} \to \mathbb{K}^d, \qquad \mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}.$$

The set $(\mathbb{K}^d)^{\mathcal{X}}$ denotes all such functions from $\mathcal{X}$ to $\mathbb{K}^d$, i.e., $\phi \in (\mathbb{K}^d)^{\mathcal{X}}$. If $\mathcal{X}$ is finite, i.e.,

$$\mathcal{X} = \{x_1, \ldots, x_{|\mathcal{X}|}\},$$

the embedding ϕ can equivalently be represented as a matrix

$$\vec{\phi} \in \mathbb{K}^{|\mathcal{X}| \times d} \quad \textit{with} \quad \phi(x_i) = \vec{\phi}_{i,:}^{\top}.$$

Definition 8 introduces the notion of a purely numerical embedding of a set. At this level, no structural assumptions are imposed. The mapping $\phi : \mathcal{X} \to \mathbb{K}^d$ may therefore be arbitrary and need not reflect relationships among elements of $\mathcal{X}$. In practice, useful embeddings are typically constructed with respect to some underlying structure defined on the set. In the present work, we consider graphs as structural components of interest.

Definition 9. *Let $\mathcal{X}$ be a set and $d \in \mathbb{N}$. Given sets of vertices $\mathcal{V}$ and edge labels $\mathcal{M}$, a graph embedding model of $\mathcal{X}$ over $(\mathcal{V}, \mathcal{M})$ is an operator*

$$\Phi : \Omega_{\mathcal{X}} \to (\mathbb{K}^d)^{\mathcal{X}}, \quad \Omega_{\mathcal{X}} := \Omega_{\mathcal{X}}^{(\mathcal{V},\mathcal{M})} \subseteq 2^{\mathcal{V}\times\mathcal{V}\times\mathcal{M}},$$

where $\Omega_{\mathcal{X}}$ denotes the set of graphs over $(\mathcal{V}, \mathcal{M})$ that encode structural information about $\mathcal{X}$. It assigns to each graph $\mathcal{G} \in \Omega_{\mathcal{X}}$ a graph embedding $\Phi(\mathcal{G}) \in (\mathbb{K}^d)^{\mathcal{X}}$ of $\mathcal{X}$. If $\mathcal{X}$ is finite, we again define

$$\vec{\Phi}(\mathcal{G}) \in \mathbb{K}^{|\mathcal{X}|\times d} \quad \textit{with} \quad \Phi(\mathcal{G})(x_i) = \vec{\Phi}(\mathcal{G})_{i,:}^{\top}.$$

Definition 9 introduces the notion of a graph embedding model but does not specify how embeddings are obtained. In practice, embeddings are produced by concrete computational or mathematical procedures that derive vector representations from graph structure [42, 51, 229]. A graph embedding model thus formalizes a mapping from a graph to a corresponding graph embedding of $\mathcal{X}$. A central characteristic of graph embeddings is their dependence on structure. Consider the same set $\mathcal{X}$ together with graphs $\mathcal{G}, \mathcal{G}' \in \Omega_{\mathcal{X}}$. Although the underlying set $\mathcal{X}$ remains the same, the induced embeddings typically differ, as they reflect different structural patterns, i.e., $\Phi(\mathcal{G})(x) \neq \Phi(\mathcal{G}')(x)$ for $x \in \mathcal{X}$ in general. Graph embeddings therefore do not arise from $\mathcal{X}$ alone, but from the interaction between its elements and a particular graph structure. The set $\Omega_{\mathcal{X}}$ specifies which graphs are associated with $\mathcal{X}$. Typically, components of a graph serve as the objects to be embedded. Depending on the application, one may embed vertices, edge labels, individual edges, higher-order substructures, or entire graphs as global objects [35]. In the present work, however, we restrict attention to embeddings of vertex sets $\mathcal{X} \subseteq \mathcal{V}$, implying the admissible set of graphs

$$\Omega_{\mathcal{X}} \subseteq \{\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M} \mid \mathcal{X} \subseteq \mathcal{V}(\mathcal{G})\},$$

i.e., all graphs over $(\mathcal{V}, \mathcal{M})$ that contain information about the vertices in $\mathcal{X}$. In this context, we distinguish between global and local graph embedding models: $\mathcal{X} = \mathcal{V}$ corresponds to a global model, whereas $\mathcal{X} \subset \mathcal{V}$ corresponds to a local one. Accordingly, we view a graph embedding model as mapping each graph $\mathcal{G}$ to an embedding

$$\phi = \Phi(\mathcal{G}) \in (\mathbb{K}^d)^{\mathcal{X}}, \quad \mathcal{X} \subseteq \mathcal{V}.$$

The restriction to vertex embeddings is natural in graph representation learning, where vertices typically represent entities of interest and edges describe relationships between them. Indeed, in much of the literature, the term *graph embedding* is commonly used to refer specifically to vertex-level embeddings with respect to a given graph [42, 84, 176].

It is important to note that Definition 9 is still conceptual. While it formalizes what constitutes a graph embedding model and a corresponding graph embedding, it does not specify how such a model is formalized or learned from data. The design of graph embedding models, that is, the question of how structural information is translated into meaningful vector representations, constitutes a central research question in graph representation learning [42].

Several graph embedding models have emerged to derive meaningful embeddings for different kinds of graphs. In particular, a large body of work considers embedding models for RDF knowledge graphs, commonly referred to as knowledge graph embeddings [51, 229]. However, only a few approaches explicitly incorporate ontology axioms or other schema-level constraints into representation learning. Such methods are typically restricted to simplified logical fragments, require additional reasoning steps, or focus on ontology-centric tasks rather than instance-level representations [46]. Moreover, empirical studies have not demonstrated consistent improvements over models that rely solely on the inherent RDF graph structure, i.e., treating RDF graphs as directed predicate-labeled graphs [34]. Consequently, and in line with the predominant body of related work, we treat RDF graphs $\mathcal{F}$, including knowledge graphs, as directed and heterogeneous graphs. In particular, we do not differentiate between induced classes $\mathcal{C}(\mathcal{F})$ and entities $\mathcal{E}(\mathcal{F})$, and we treat rdf:type as a regular relation type. Graph embeddings in this work are therefore defined with respect to this relational view of RDF, and the term knowledge graph embedding is used whenever the graph under consideration constitutes a knowledge graph.

Generally, RDF graphs may contain literal arguments, i.e., $\mathcal{A}_{\mathfrak{L}}(\mathcal{F}) \neq \emptyset$, which serve as terminal annotation values rather than structural entities. Analogous to class arguments, literals occur only as triple objects and cannot have outgoing edges. Unlike class resources, however, they do not represent unique graph resources but must be interpreted in the context of the subject–predicate pair in which they occur. As a result, they do not naturally fit the structural interpretation of vertices. Some existing approaches incorporate literal arguments into embedding models by reifying literal-valued annotations as statement resources or by predefining embeddings for literal values that remain fixed when constructing embeddings for resource vertices [79]. Nevertheless, such mechanisms do not affect the methodologies developed in this work. For the purposes of relational representation learning, we therefore restrict attention to the resource-level projection $\mathcal{F} = \mathcal{F}_{\mathfrak{R}}$, i.e., to relational RDF knowledge graphs. If $\mathcal{F}$ is a knowledge graph with respect to an ontology $\mathcal{O}$, its resource-level subgraph remains a valid knowledge graph with respect to $\mathcal{O}$ whenever $\mathcal{F}_{\mathfrak{R}} \neq \emptyset$, due to the OWA. However, the restriction to resource-level facts is not injective, as literal-valued triples are discarded. Resource arguments in $\mathcal{A}_{\mathfrak{R}}(\mathcal{F}) \setminus \mathcal{A}(\mathcal{F}_{\mathfrak{R}})$ that are exclusively linked to literal arguments may disappear from the resource-level graph. To retain this information, one may define an injective transformation of $\mathcal{F}$ into $(\mathcal{F}_{\mathfrak{R}}, \ell_{\mathcal{F}})$ via the mapping

$$\ell_{\mathcal{F}} : \mathcal{A}_{\mathfrak{R}}(\mathcal{F}) \to 2^{\mathcal{P}_{\mathfrak{L}}(\mathcal{F}) \times \mathcal{A}_{\mathfrak{L}}(\mathcal{F})}, \qquad \ell_{\mathcal{F}}(v) = \{(p,o) \mid (v,p,o) \in \mathcal{F}_{\mathfrak{L}}\}.$$

Moreover, we consider real-valued graph embeddings in this work, i.e., $\mathbb{K} = \mathbb{R}$. This reflects the predominant practice in the literature [51] and the fact that the methodologies developed in this thesis are primarily designed for real-valued embeddings. However, we indicate how the proposed constructions extend naturally to complex-valued embeddings when appropriate.

We distinguish three conceptually different categories of graph embedding models applicable to generalized graphs $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M}$, including RDF graphs. These categories differ in how embeddings are obtained, how structural context is incorporated, and in their modeling scope, for example, whether $\mathcal{G}$ must be reduced to undirected or homogeneous graphs. We restrict attention to finite graphs with $|\mathcal{G}| < \infty$, implying that both the vertex and label sets are finite:

$$\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}, \quad \mathcal{M} = \{m_1, \ldots, m_{|\mathcal{M}|}\}.$$

4.1 Non-Parametric Graph Embedding Models

Non-parametric graph embedding models constitute the most direct way of constructing embeddings of vertex sets from a graph, typically at a global level. In contrast to parametric approaches, these methods derive embeddings solely from the given graph structure without introducing trainable parameters or solving an optimization problem. The resulting representations are therefore determined by the input graph and remain fixed once computed [84].
The simplest non-parametric embeddings arise from encoding vertices directly with respect to their inherent ordering. Typical examples include one-hot encodings, where each vertex v_i is mapped to the i-th canonical basis vector of $\mathbb{R}^{|\mathcal{V}|}$, as well as neighborhood indicator vectors that represent a vertex by the characteristic vector of its adjacency pattern [42]. Further deterministic descriptors include handcrafted structural features such as degree, centrality measures, or motif counts [206]. These examples illustrate that non-parametric embedding models may be defined directly through fixed structural encodings of vertices. A particularly important subclass of such models is formed by algebraic constructions based on adjacency representations of graphs. In this setting, adjacency tensors are viewed as structural objects that uniquely encode the connectivity of graphs $\mathcal{G}$ over $(\mathcal{V}, \mathcal{M})$ and thus serve as a canonical algebraic representation:

$$\vec{\mathcal{G}} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|\times|\mathcal{M}|}, \qquad \vec{\mathcal{G}}_{i,j,k} = \begin{cases} 1 & \text{if } (v_i, v_j, m_k) \in \mathcal{G}, \\ 0 & \text{otherwise.} \end{cases}$$

Many classical approaches focus on homogeneous graphs $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \{*\}$. In this case, the adjacency tensor canonically reduces to the adjacency matrix $\vec{\mathcal{G}} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$. Deterministic graph embedding models are then obtained by applying predefined, non-learned transformations

$$f : \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|} \longrightarrow \mathbb{R}^{|\mathcal{V}|\times d},$$

yielding embedding matrices of the form

$$\vec{\phi} = f\left(\vec{\mathcal{G}}\right) \in \mathbb{R}^{|\mathcal{V}|\times d}.$$

In contrast to learning-based methods, f is fixed a priori and consists solely of well-defined matrix operations. Prominent examples of such algebraic deterministic embeddings include spectral methods, which derive vertex representations from eigendecompositions of matrices constructed from $\vec{\mathcal{G}}$, such as the graph Laplacian or normalized adjacency matrix [30]. The resulting coordinates reflect global structural properties including connectivity, diffusion behavior, and community structure. Another non-parametric strategy is given by matrix-analytic representations, where rows or columns of $\vec{\mathcal{G}}$, or functions thereof, are directly used as feature vectors, optionally after applying fixed transformations such as normalization, matrix powers, or kernel mappings capturing multi-hop connectivity [159]. However, many of these algebraic transformations implicitly assume symmetry of the adjacency matrix, as required, for instance, by spectral decompositions. Consequently, when applied to directed graphs, they effectively introduce undirectedness by symmetrizing edges. Still, given $\mathcal{M} = \{*\}$, we can derive an operator that constitutes a graph embedding model as per Definition 9 for $\mathcal{X} = \mathcal{V}$ via:

$$\vec{\Phi}(\mathcal{G})(v_i) = f(\vec{\mathcal{G}})_{i,:} \in \mathbb{R}^d. \tag{4.1}$$

For heterogeneous graphs, analogous constructions can be applied. The straightforward approach is to consider the canonical representation of $\mathcal{G}$ as a homogeneous graph

$$\mathcal{G}^* := \{(v,u,*) \mid (v,u,m) \in \mathcal{G}\}.$$

Thus, by applying a graph embedding model as per Equation 4.1, we obtain

$$\vec{\Phi}(\mathcal{G}) = f(\vec{\mathcal{G}}^*) \in (\mathbb{R}^d)^{\mathcal{V}},$$

such that label information (and potentially directionality) is discarded. To keep track of label structure, recall that a heterogeneous graph can be decomposed into $|\mathcal{M}|$ homogeneous graphs

$$\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(|\mathcal{M}|)}, \qquad \mathcal{G}^{(k)} = \{\, (v,u,m) \in \mathcal{G} \mid m = m_k \,\} \subseteq \mathcal{V} \times \mathcal{V} \times \{m_k\},$$

with the corresponding adjacency matrices

$$\vec{\mathcal{G}}^{(k)} = \vec{\mathcal{G}}_{:,:,k} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}.$$

Thus, an operator for homogeneous graphs as per Equation 4.1 can be applied, yielding

$$\vec{\phi}^{(1)}, \ldots \vec{\phi}^{(|\mathcal{M}|)} \quad \text{with} \quad \vec{\phi}^{(k)} = f(\vec{\mathcal{G}}^{(k)}) \in \mathbb{R}^{|\mathcal{V}|\times d}$$

as label-wise embedding matrices, which can then be combined via predetermined aggregation operators such as summation, concatenation, or block-matrix constructions [60], i.e.,

$$\vec{\phi} = \bigoplus\nolimits_{k\in\{1,\ldots,|\mathcal{M}|\}} \vec{\phi}^{(k)} \in \mathbb{R}^{|\mathcal{V}|\times d}.$$

Given such an aggregation, we can thus define a generalized operator

$$\vec{\Phi}(\mathcal{G}) = \bigoplus\nolimits_{k\in\{1,\ldots,|\mathcal{M}|\}} f(\vec{\mathcal{G}}^{(k)}) \in \mathbb{R}^{|\mathcal{V}|\times d}.$$

Another approach to deterministic, non-parametric graph embedding models is given by constructive methods that derive graph embeddings from deterministic base embeddings.

Definition 10. *Given vertices and edge labels $\mathcal{V}, \mathcal{M} \neq \emptyset$, a graph embedding constructor*

$$\mathcal{K} : (\mathbb{R}^{d'})^{\mathcal{V}} \times 2^{\mathcal{V}\times\mathcal{V}\times\mathcal{M}} \longrightarrow (\mathbb{R}^d)^{\mathcal{V}}$$

assigns to a base embedding $\psi \in (\mathbb{R}^{d'})^{\mathcal{V}}$ and a graph $\mathcal{G} \subseteq \mathcal{V}\times\mathcal{V}\times\mathcal{M}$ an embedding

$$\phi = \mathcal{K}(\psi, \mathcal{G}) \in (\mathbb{R}^d)^{\mathcal{V}}.$$

Given $\mathcal{X} \subseteq \mathcal{V}$ and $\psi \in (\mathbb{R}^{d'})^{\mathcal{V}}$, it induces a constructive graph embedding model

$$\Phi : \Omega_{\mathcal{X}} \to (\mathbb{R}^d)^{\mathcal{X}}, \quad \Phi(\mathcal{G})(v) = \mathcal{K}(\psi, \mathcal{G})(v).$$

In this context, base embeddings are independent of specific graph structures, while $\mathcal{K}$ incorporates structural information through each graph $\mathcal{G}$. Such construction methodologies are typically based on local aggregation schemes, such as neighborhood averaging, i.e.,

$$\mathcal{K}(\psi,\mathcal{G})(v) = \frac{1}{|\mathcal{N}_{\mathcal{G}}(v)|}\sum_{u\in\mathcal{N}_{\mathcal{G}}(v)}\psi(u) \in \mathbb{R}^d, \quad \text{for} \quad d' = d.$$

In such cases, the choice of $\mathcal{X} \subseteq \mathcal{V}$ and $\mathcal{G} \subseteq \mathcal{V}\times\mathcal{V}\times\mathcal{M}$ determines the required support of ψ. For example, the aggregation above only requires base embeddings for all neighbors

$$\psi \in (\mathbb{R}^{d'})^{\mathcal{N}_{\mathcal{G}}(\mathcal{X})}, \qquad \mathcal{N}_{\mathcal{G}}(\mathcal{X}) := \bigcup\nolimits_{v\in\mathcal{X}}\mathcal{N}_{\mathcal{G}}(v).$$

The framework can be extended to base embeddings of additional structural components, such as edge labels, which may be incorporated by aggregating embeddings associated with incident edges. However, in line with the present focus on vertex embeddings, we restrict attention to base embeddings defined on $\mathcal{V}$. Figure 4.1 illustrates the separation between structure-independent base embeddings ψ and a graph-dependent construction operator $\mathcal{K}$.

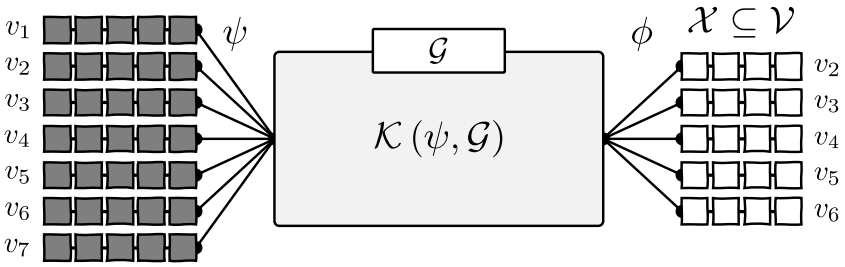


Figure 4.1: Schematic illustration of a graph embedding constructor $\mathcal{K}$ that combines base embeddings $\psi \in (\mathbb{R}^5)^{\mathcal{V}}$ with graph structure $\mathcal{G}$ to construct $\phi(v) = \mathcal{K}(\psi,\mathcal{G})\,|_{\mathcal{X}}\,(v) \in \mathbb{R}^4$.

In summary, non-parametric graph embedding models encompass a broad class of deterministic methods, including direct structural encodings, algebraic transformations of adjacency representations, and constructive approaches that derive embeddings from base representations while incorporating graph structure. Their interpretability and reproducibility make them attractive for structural analysis, but their inability to adapt representations to a specific predictive objective motivates the learning-based approaches introduced in the following sections.

4.2 Parametric Graph Embedding Models

Unlike deterministic, non-parametric graph embedding models, parametric variants do not determine graph embeddings through fixed transformations. Instead, given vertices $\mathcal{V}$ and edge labels $\mathcal{M}$, they specify a space of trainable parameter settings $\Theta := \Theta_{(\mathcal{V},\mathcal{M})}$ such that each parameter setting $\theta \in \Theta$ determines an $\mathbb{R}^d$-embedding of $\mathcal{X} \subseteq \mathcal{V}$ [113]:

$$\{\phi_\theta \mid \theta \in \Theta\}, \qquad \phi_\theta \in (\mathbb{R}^d)^{\mathcal{X}}.$$

Suitable parameters are obtained from graph data by minimizing a task-dependent loss function

$$\mathcal{L} : \Omega_{\mathcal{X}} \times \Theta \to \mathbb{R}.$$

The learning process can be described by a training operator $\mathcal{T} : \Omega_{\mathcal{X}} \to \Theta$, such that each $\theta \in \Theta$ determines an embedding $\phi_\theta \in (\mathbb{R}^d)^{\mathcal{X}}$. Training amounts to solving the optimization problem

$$\mathcal{T}(\mathcal{G}) \in \arg\min_{\theta \in \Theta} \mathcal{L}(\mathcal{G}, \theta).$$

Thus, a graph $\mathcal{G}$ is assigned a trained parameter configuration $\mathcal{T}(\mathcal{G}) \in \Theta$, inducing the graph embedding $\phi_{\mathcal{T}(\mathcal{G})}$. In this sense, parametric embedding models determine embeddings through optimization, i.e., each training operator induces a graph embedding model

$$\Phi : \Omega_{\mathcal{X}} \to (\mathbb{R}^d)^{\mathcal{X}}, \qquad \Phi(\mathcal{G}) := \phi_{\mathcal{T}(\mathcal{G})}.$$

Graph embeddings obtained by specifying embeddings ϕ_θ as explicit parameters are referred to as *self-contained*. A defining characteristic of self-contained graph embedding models is that graph structure influences the learned representations only indirectly through the objective optimized by $\mathcal{T}$, rather than through an explicit construction rule. Structural regularities are captured because the parameters are optimized with respect to patterns observed in $\mathcal{G}$, while the embeddings remain free variables throughout the learning process. Consequently, self-contained graph embedding paradigms do not differ in how embeddings are parameterized, but in how structural information from the graph is translated into an optimization objective. Concretely, the training signal may enforce global consistency with the observed edge set, impose relation-specific geometric constraints, or model stochastic proximities via graph traversals.

In contrast, constructive variants of parametric graph embedding models do not treat the target embeddings as explicit parameters, but as the result of a parameterized construction procedure. Formally, embeddings are obtained as

$$\phi_\theta = \mathcal{K}_\theta(\psi_\theta, \mathcal{G}), \quad \theta \in \Theta,$$

where $\mathcal{K}_\theta$ denotes a parameterized graph embedding constructor and ψ_θ a (potentially learned) base embedding. In this setting, the embeddings ϕ_θ are not free parameters, but are derived from the structure of $\mathcal{G}$ through $\mathcal{K}_\theta$. If the base embedding ψ_θ is learned jointly with the constructor, it induces representations for those components of the graph on which it is defined, typically a subset of vertices or auxiliary structural elements.

A representative example is NodePiece [75], which derives node representations from a base embedding defined only over a small set of anchor nodes $\mathcal{V}' \subset \mathcal{V}$. Another prominent family of constructive embedding models is given by Graph Neural Networks (GNNs) [41, 114, 197, 224, 234], which compute node representations through iterative, parameterized aggregation of local neighborhood information. These models will be discussed in detail in Section 4.3.

The following subsections introduce representative realizations of self-supervised graph embedding paradigms, originally developed for self-contained models but also applicable to constructive variants. While many constructive approaches still rely on such objectives, recent work has introduced training methodologies specifically tailored to parameterized constructors. These developments are discussed alongside the classical paradigms.

4.2.1 Factorization-Based Training Objectives

Factorization-based methods realize training by learning global graph embeddings (i.e., $\mathcal{X} = \mathcal{V}$) as explicit optimization variables to reconstruct the adjacency representation of a graph. In contrast to non-parametric algebraic embedding models, they consider a reconstructive mapping

$$f : (\mathbb{R}^d)^{\mathcal{V}} \to \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|\times|\mathcal{M}|},$$

which produces a tensor $f(\phi_\theta)$ intended to approximate the adjacency tensor $\vec{\mathcal{G}}$. Training consists of selecting a parameter setting that minimizes a factorization loss

$$\mathcal{L}_{\text{fac}} : \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|\times|\mathcal{M}|} \times \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|\times|\mathcal{M}|} \to \mathbb{R},$$

typically defined via an entry-wise loss $\ell : \{0,1\} \times \mathbb{R} \to \mathbb{R}$ against the graph's adjacency tensor:

$$\mathcal{L}(\mathcal{G}, \theta) = \mathcal{L}_{\text{fac}}\Big(\vec{\mathcal{G}}, f(\phi_\theta)\Big) = \sum_{i,j,k} \ell\Big(\vec{\mathcal{G}}_{i,j,k}, f(\phi_\theta)_{i,j,k}\Big).$$

The selected parameter setting is therefore the output of a training operator,

$$\mathcal{T}(\mathcal{G}) \in \arg\min_{\theta\in\Theta} \mathcal{L}(\mathcal{G}, \theta).$$

The learned graph embedding is then given by the component $\phi_{\mathcal{T}(\mathcal{G})}$. Analogous to deterministic algebraic graph embedding models, many factorization-based approaches are primarily formulated for homogeneous graphs, i.e., $\mathcal{M} = \{*\}$. In this case, the adjacency tensor again reduces to the adjacency matrix $\vec{\mathcal{G}} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, which is approximated by a bilinear form

$$f(\phi_\theta) = \vec{\phi}_\theta \cdot W_\theta \cdot \vec{\phi}_\theta^{\top} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|},$$

where $W_\theta \in \mathbb{R}^{d\times d}$ is an interaction matrix contained in θ. Structural constraints on W_θ (e.g., symmetry, diagonality, or low rank) give rise to different models and control expressiveness and complexity [6, 166, 211]. Edges are thus predicted directly from pairwise interactions of embeddings, so that vertices with compatible latent representations are reconstructed as adjacent. This perspective naturally gives rise to a link prediction interpretation, in which the learned model assigns scores to potential edges and is evaluated based on its ability to distinguish observed from non-observed connections. This formulation naturally extends to heterogeneous graphs via relation-specific interaction matrices and by reconstructing the tensor slice-wise [161, 219, 240]:

$$f(\phi_\theta)_{:,:,k} = \vec{\phi}_\theta \cdot W_\theta^{(k)} \cdot \vec{\phi}_\theta^{\top} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}.$$

Structural information is incorporated through the objective optimized by $\mathcal{T}$, rather than through an explicit construction of representations from neighborhoods. Even if the embeddings themselves are free optimization variables, the training objective constrains them to reflect the structural patterns observed in the graph. In practice, graphs are typically sparse, so evaluating entry-wise losses over all non-observed edges would be computationally prohibitive and would heavily bias the objective toward absent entries. Instead, the loss is usually computed over the observed edges together with a sampled subset of non-existing edges, which serve as negative samples during training [244]. The resulting embeddings appear directly as learned parameters and thus constitute self-contained representations of the graph. Because factorization-based objectives interpret sampled non-observed edges as negative evidence, they implicitly adopt a closed-world interpretation during training, which contrasts with the OWA underlying the RDF data model.

4.2.2 Transformation-Based Training Objectives

Transformation-based methods constitute a second realization of the training operator $\mathcal{T}$. While factorization-based approaches aim to reconstruct the adjacency tensor globally, these methods realize link prediction by modeling local compatibility constraints for individual edges [192]. Rather than predicting adjacency entries simultaneously, they assume that the validity of an observed edge $(v,u,m) \in \mathcal{G}$ can be evaluated through a label-specific transformation acting on the corresponding vertex embeddings. These approaches can thus be interpreted as local counterparts to factorization-based objectives, replacing global reconstruction of the adjacency tensor by edge-wise scoring of individual triples. While applicable to subsets $\mathcal{X} \subset \mathcal{V}$, they are typically trained globally. Concretely, each parameter setting $\theta \in \Theta$ implies a graph embedding $\phi_\theta \in (\mathbb{R}^d)^{\mathcal{V}}$ and metadata-dependent transformation parameters, reflecting that these models are primarily designed for heterogeneous, directed graphs. These define label-specific mappings

$$f_\theta^{(m)}, g_\theta^{(m)} : \mathbb{R}^d \to \mathbb{R}^{d'},$$

and the central modeling assumption is that observed edges satisfy the compatibility relation

$$f_\theta^{(m)}(\phi_\theta(v)) \approx g_\theta^{(m)}(\phi_\theta(u)).$$

Thus, labels act as operators that map vertex embeddings into a comparison space in which connected vertices should be close to each other [51]. Selecting $g_\theta^{(m)} = \mathrm{id}$ yields approximations

$$f_\theta^{(m)}(\phi_\theta(v)) \approx \phi_\theta(u),$$

which are called translational graph embedding paradigms. For instance, given parametrized label representations $\rho_\theta \in (\mathbb{R}^d)^{\mathcal{M}}$, TransE [36] realizes this idea through additive translations

$$f_\theta^{(m)}(\phi_\theta(v)) = \phi_\theta(v) + \rho_\theta(m) \approx \phi_\theta(u).$$

More expressive variants introduce projections or linear operators [106, 135, 230], replacing simple translations with structured transformations while preserving the same training principle. Given such transformations, edge plausibilities are quantified via a scoring function

$$S_\theta : \mathcal{V} \times \mathcal{V} \times \mathcal{M} \to \mathbb{R},$$

for instance, by measuring the ℓ^2-distance between transformed representations:

$$S_\theta(v,u,m) = -\left\| f_\theta^{(m)}(\phi_\theta(v)) - g_\theta^{(m)}(\phi_\theta(u)) \right\|_2.$$

The training operator $\mathcal{T}$ then selects a parameter setting whose scores separate observed edges from non-observed ones according to a ranking or contrastive objective. However, evaluating such objectives over all absent edges would be computationally infeasible for large and sparse graphs. In practice, the loss is therefore approximated using negative sampling: for each observed triple $(v,u,m) \in \mathcal{G}$, a small number of corrupted triples such as (v',u,m) or (v,u',m) are

generated and treated as negative samples [244]. Typically, multiple negative edges are sampled per positive edge, yielding a stochastic approximation of the ideal objective. A common formulation is the margin-based ranking loss

$$\mathcal{L}_{\mathrm{MRL}}(\mathcal{G},\theta) = \sum_{(v,u,m)\in\mathcal{G}} \sum_{(v',u',m)\in \mathrm{CorT}(v,u,m)} \max\big(0,\, \gamma + S_\theta(v',u',m) - S_\theta(v,u,m)\big),$$

where $\mathrm{CorT}(v,u,m)$ denotes a sampled set of corrupted triples and $\gamma > 0$ is a margin hyperparameter. The training operator is then defined by

$$\mathcal{T}(\mathcal{G}) \in \arg\min_{\theta\in\Theta} \mathcal{L}_{\mathrm{MRL}}(\mathcal{G},\theta).$$

As in the factorization-based case, the learned graph embedding $\phi_{\mathcal{T}(\mathcal{G})}$ may appear explicitly as part of the optimized parameter setting and thus constitute a self-contained representation of the graph. Even if the embedding candidates themselves are treated as free optimization variables, structural information is incorporated through the objective optimized during training, so that explicit graph-based construction mechanisms are not required.
Most transformation-based models operate in real-valued spaces, which is also the setting considered throughout this work. Some extensions, such as RotatE or ComplEx, employ complex-valued embeddings to represent additional structural patterns (e.g., asymmetry) via phase-based transformations [208, 219]. These models can be interpreted as operating over $\mathbb{C}^d$ without altering the underlying training-operator formulation described above.

4.2.3 Random-Walk-Based Training Objectives

Random-walk-based methods constitute a third realization of self-contained graph embedding models. In contrast to factorization-based approaches that reconstruct adjacency structure and transformation-based approaches enforcing edge compatibility constraints, these methods derive training signals from stochastic traversals of the graph. Structural information is thus not modeled explicitly, but implicitly through co-occurrence statistics along sampled walks.
Given a graph $\mathcal{G}$, the training operator first constructs a multiset of sequences by repeatedly performing random walks. A walk of length L starting at $v_0 \in \mathcal{V}$ is a sequence

$$(v_0, s_1, s_2, \ldots, s_L),$$

with stubs $s_i = (v_i, m_i, \iota_i) \in \mathcal{S}_{\mathcal{G}}(v_{i-1})$. Each transition is sampled according to a distribution $\mathbb{P}(s_i \mid v_{i-1})$. The precise structure of the generated sequences depends on the type of graph:

- For homogeneous and undirected graphs, $s_i = (v_i, *, *)$ holds, and walks reduce to

  $$(v_0, v_1, \ldots, v_L)$$

  as vertex sequences, as used in DeepWalk [170] and node2vec [86].

- For homogeneous but directed graphs, stubs $s_i = (v_i, *, \iota_i)$ record traversal directions, yielding sequences of the form

  $$(v_0, \iota_1, v_1, \iota_2, v_2, \ldots, \iota_L, v_L),$$

  and the parameter setting includes incidence representations $\{\rho_\theta \in (\mathbb{R}^d)^{\mathcal{I}} \mid \theta \in \Theta\}$.

- For heterogeneous but undirected graphs, stubs $s_i = (v_i, m_i, *)$ record labels, yielding sequences
$$(v_0, m_1, v_1, m_2, v_2, \ldots, m_L, v_L),$$
and the parameter setting induces label representations $\{\rho_\theta \in (\mathbb{R}^d)^{\mathcal{M}} \mid \theta \in \Theta\}$.
- For heterogeneous, directed graphs, stubs $s_i = (v_i, m_i, \iota_i)$ contain directed label information, yielding sequences
$$(v_0, (m_1, \iota_1), v_1, (m_2, \iota_2), v_2, \ldots, (m_L, \iota_L), v_L),$$
as in RDF2Vec [188], with directed label representations $\{\rho_\theta \in (\mathbb{R}^d)^{\mathcal{M}\times\mathcal{I}} \mid \theta \in \Theta\}$.

The resulting collection of walks is interpreted as a corpus $\mathcal{W}(\mathcal{G})$ of token sequences
$$(w_1, \ldots, w_n) \in \mathcal{W}(\mathcal{G}), \quad n \in \{L+1, 2L+1\}.$$
Each token represents either a vertex or an auxiliary symbol (e.g., a label, an incidence, or their combination). Each parameter setting $\theta \in \Theta$ thus induces representations for these tokens, including vertex embeddings $\phi_\theta \in (\mathbb{R}^d)^{\mathcal{V}}$ and auxiliary embeddings ϕ'_θ. Random-walk-based methods thereby transform the graph learning problem into a token-based representation learning problem analogous to natural language processing. The generated walks play the role of sentences, and vertices together with structural symbols act as words.
Learning is then performed using distributional objectives, such as the one originating from the Word2Vec framework [149]. A common formulation is the Continuous Bag-of-Words (CBOW) model with context window size $c \in \mathbb{N}$. For each position i in a sequence $(w_1, \ldots, w_n) \in \mathcal{W}(\mathcal{G})$, the surrounding tokens
$$C_i := \{\, w_{i+j} \mid -c \le j \le c,\ j \neq 0 \,\}$$
are used to predict the center token w_i. The training objective maximizes
$$\sum_{(w_1,\ldots,w_n)\in\mathcal{W}(\mathcal{G})} \sum_{i=1}^{n} \log \mathbb{P}_\theta\big(w_i \mid C_i\big),$$
yielding the loss
$$\mathcal{L}_{\mathrm{CBOW}}(\mathcal{G}, \theta) = -\sum_{(w_1,\ldots,w_n)\in\mathcal{W}(\mathcal{G})} \sum_{i=1}^{n} \log \mathbb{P}_\theta\big(w_i \mid C_i\big).$$
The conditional probability is parameterized directly by the token representations, typically by aggregating context embeddings (e.g., via averaging) and predicting the center token using a softmax model based on inner products. The training operator is thus given by
$$\mathcal{T}(\mathcal{G}) \in \arg\min_{\theta\in\Theta} \mathcal{L}_{\mathrm{CBOW}}\big(\mathcal{G}, \theta\big).$$
Computing these objectives exactly requires normalization over the entire token vocabulary and is often infeasible for large graphs. Therefore, Word2Vec-style training replaces the full softmax with negative sampling, i.e., a stochastic objective that contrasts observed token–context pairs with randomly drawn noise tokens. In contrast to factorization-based and transformation-based

methods, this negative sampling operates at the level of token co-occurrences rather than edges. Consequently, random-walk-based methods do not realize link prediction and do not require edge-level negative sampling.
Unlike the previous paradigms, similarity between vertices is not enforced through adjacency reconstruction or compatibility constraints, but emerges from shared walk contexts, i.e., vertices that occur in similar contexts receive similar embeddings. The representations thus arise from distributional properties of graph traversals while remaining explicit elements of the learned parameter setting and therefore constitute self-contained graph embeddings. Even if the embedding candidates themselves are treated as free optimization variables, the distributional objective constrains them to reflect structural patterns induced by the graph. From a semantic perspective, these methods do not impose negative evidence on non-observed edges and therefore preserve the OWA underlying the RDF data model. Potential inconsistencies arise only at the level of token representations, where distinct symbols may be mapped to similar embeddings despite representing different resources, thereby affecting assumptions such as the NUNA.

4.2.4 Contrastive Training Objectives

Contrastive learning has emerged as a further paradigm for training graph embedding models. Unlike factorization-based and transformation-based methods, which rely on edge-level supervision, and random-walk-based methods, which exploit co-occurrence statistics, contrastive approaches derive training signals from consistency across multiple perturbed graph views [248]. Given a graph $\mathcal{G}$, the training operator constructs stochastic augmentations

$$\tilde{\mathcal{G}}^{(1)}, \tilde{\mathcal{G}}^{(2)}, \dots$$

obtained through random perturbations such as edge dropout, subgraph sampling, or feature masking [110]. These perturbations preserve global structural characteristics while altering local connectivity patterns. A parametric graph embedding model produces representations

$$\phi_\theta^{(1)}, \phi_\theta^{(2)} \in (\mathbb{R}^d)^{\mathcal{X}}$$

for corresponding graph augmentations. The training objective then enforces that embeddings of the same vertex across different views remain similar, while embeddings of different vertices remain distinguishable. This is typically realized via contrastive objectives of the form

$$\mathcal{L}_{\mathrm{CL}}(\mathcal{G}, \theta) = \sum_{v \in \mathcal{X}} \ell_{\mathrm{con}}\big(\phi_\theta^{(1)}(v), \phi_\theta^{(2)}(v), \{\phi_\theta^{(2)}(u)\}_{u \neq v}\big),$$

where ℓ_{con} denotes a contrastive loss that separates positive pairs (same vertex across views) from negative pairs (different vertices). The training operator is then given by

$$\mathcal{T}(\mathcal{G}) \in \arg\min_{\theta \in \Theta} \mathcal{L}_{\mathrm{CL}}(\mathcal{G}, \theta).$$

A defining characteristic of contrastive training is that structural information is incorporated through the response of the embedding model to graph perturbations. In particular, the objective

enforces invariance of representations under stochastic modifications of the graph while preserving discriminability across vertices. Consequently, the effectiveness of such objectives relies on the embeddings being explicitly constructed from graph structure.
In contrast to the previously discussed paradigms, contrastive objectives are not meaningful for self-contained graph embedding models in which embeddings are treated as independent optimization variables. Since such embeddings do not depend explicitly on graph structure, perturbations of $\mathcal{G}$ do not induce corresponding changes in the representations, and therefore no meaningful contrastive signal can be obtained. Instead, contrastive learning is inherently tied to constructive graph embedding models with $\phi_\theta = \mathcal{K}_\theta(\psi_\theta, \mathcal{G})$, in which embeddings are generated through parameterized graph-dependent mappings. In practice, most contrastive graph embedding methods are therefore based on graph embedding constructors, in particular GNNs, which compute representations through iterative aggregation of local neighborhood information. Prominent examples include GRACE [255] and KGCL [243]. This observation motivates a closer examination of graph embedding constructors, with GNNs forming the most prominent and widely studied class. The following section therefore introduces GNNs as a general framework for parameterized, structure-dependent graph embedding models.

4.3 Graph Neural Networks

Neural Networks (NNs) transform numerical input representations into output representations via compositions of activated affine transformation layers [83, 130].

Definition 11. *Let $d', d \in \mathbb{N}$ denote input and output dimensions. A dense layer*

$$f : \mathbb{R}^{d'} \to \mathbb{R}^d, \quad f(x) = \sigma(W \cdot x + b)$$

is defined by a weight matrix $W \in \mathbb{R}^{d \times d'}$, a bias vector $b \in \mathbb{R}^d$, and a non-linear activation function $\sigma : \mathbb{R}^d \to \mathbb{R}^d$, producing the layer output $y = f(x)$.

Dense layers are typically optimized with respect to downstream tasks such as classification or regression using gradient-based learning [131]. They form the fundamental building blocks of NNs through compositions of such layers. For example, a multi-layer perceptron (MLP) with $L \in \mathbb{N}$ dense layers f_ℓ is given by

$$F : \mathbb{R}^{d'} \to \mathbb{R}^d, \quad y = F(x) = (f_L \circ \cdots \circ f_1)(x).$$

The layers are evaluated iteratively via

$$y_\ell = f_\ell(y_{\ell-1}) = \sigma_\ell\left(W_\ell \cdot y_{\ell-1} + b_\ell\right),$$

where $y_0 := x$, $y_L := y$, and $d'_\ell = d_{\ell-1}$ with $d'_0 := d'$ and $d_L := d$. More complex NN architectures follow the same principle of composing affine transformations with non-linearities, but differ in how these transformations are structured.

Graph Neural Networks (GNNs) aggregate local graph context per node to produce numerical representations of symbolic vertices. The local context is provided by the edges adjacent to a node, which induce its incident stubs and neighborhood. In the context of GNNs, existing work differentiates between relational and non-relational GNN approaches. Accordingly, these approaches consider a set of relation types $\mathcal{R}$ as edge labels, i.e.,

$$\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{R}, \quad \mathcal{M} = \mathcal{R}.$$

Recall that the stubs $\mathcal{S}_\mathcal{G}(v)$ of a node reduce to its neighborhood $\mathcal{N}_\mathcal{G}(v)$ in homogeneous and undirected graphs, which are considered by traditional GNN methods [114, 224, 234]. Given a vertex $v \in \mathcal{V}(\mathcal{G})$ and its neighborhood $\mathcal{N}_\mathcal{G}(v)$, these methods consider a base embedding

$$\psi : \mathcal{N}_\mathcal{G}(v) \cup \{v\} \to \mathbb{R}^{d'}$$

to produce an aggregated representation of v. Given a non-linear activation function

$$\sigma : \mathbb{R}^d \to \mathbb{R}^d,$$

traditional GNN layers can be expressed as

$$\phi(v) = \sigma\left(F_\mathrm{id}(\psi(v)) + \textstyle\sum_{u \in \mathcal{N}_\mathcal{G}(v)} c(u,v) \cdot F(\psi(u))\right),$$

where the NNs $F_\mathrm{id}, F : \mathbb{R}^{d'} \to \mathbb{R}^d$ compute self-messages and intermediate stub embeddings, weighted by coefficients $c : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$. Graph Convolutional Networks (GCNs) [114] consider degree-normalized coefficients and realize intermediate stub embeddings through shared linear transformations of adjacent embeddings:

$$F(\psi(u)) = W \cdot \psi(u) + \beta \quad \text{with} \quad W \in \mathbb{R}^{d \times d'},\ \beta \in \mathbb{R}^d.$$

Graph Attention Networks (GATs) [224] consider equivalent stub embeddings but learn attention coefficients for each neighbor from the base embeddings:

$$c(u,v) = \frac{\exp(a(\psi(u),\, \psi(v)))}{\sum_{u' \in \mathcal{N}_\mathcal{G}(v)} \exp(a(\psi(u'),\, \psi(v)))}. \tag{4.2}$$

Relational Graph Neural Networks (RGNNs) approaches such as Relational Graph Convolutional Networks (RGCNs) [197], Relational Graph Attention Networks (RGATs) [41], and CompGCN [223] extend traditional GNN layers by incorporating relation types

$$\phi(v) = \sigma\Big(\textstyle\sum_{r \in \overline{\mathcal{R}}} \sum_{u \in \mathcal{N}_{\overline{\mathcal{G}},r}(v)} c_r(u,v) \cdot F_r(\psi(u))\Big)$$

via relation-specific neighborhoods, coefficients, and NN transformations such as

$$F_r(\psi(u)) = \sigma_r\big(W_r \cdot \psi(u) + \beta_r\big), \quad r \in \mathcal{R}.$$

Instead of incorporating explicit self-messages, RGNNs consider the graph closure

$$\overline{\mathcal{G}} := \mathcal{G} \cup \{(v,v,r_\mathrm{id}) \mid v \in \mathcal{V}\}, \qquad \overline{\mathcal{R}} := \mathcal{R} \cup \{r_\mathrm{id}\}.$$

Self-messages are thus incorporated through the self-relation r_id. While applicable to heterogeneous graphs, RGNN formulations typically assume undirected graphs (or inverse relations [197]). To formally specify the incorporation of edge labels and directions into RGNN message passing, we provide a generalized definition of a GNN layer that covers homogeneous and heterogeneous graphs and incorporates directions, as well as arbitrary edge label types $\mathcal{M}$.

Definition 12. *Given vertices and labels $\mathcal{V}, \mathcal{M} \neq \emptyset$, a generalized GNN layer is defined as*

$$\kappa : (\mathbb{R}^{d'})^{\mathcal{V}} \times 2^{\mathcal{V}\times\mathcal{V}\times\mathcal{M}} \to (\mathbb{R}^{d})^{\mathcal{V}}, \;\; \kappa(\psi,\mathcal{G})(v) = \sigma\Big(\sum\nolimits_{(u,m,\iota)\in\mathcal{S}_{\mathcal{G}}(v)} c_{m,\iota}(u,v)\cdot F_{m,\iota}(\psi(u))\Big),$$

where $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ is a non-linear activation. Moreover,

$$\{c_{m,\iota}\}_{m\in\mathcal{M},\iota\in\mathcal{I}}, \qquad \{F_{m,\iota}\}_{m\in\mathcal{M},\iota\in\mathcal{I}}$$

are label- and incidence-specific coefficient functions and neural transformations with

$$c_{m,\iota} : \mathcal{V}\times\mathcal{V} \to \mathbb{R}, \qquad F_{m,\iota} : \mathbb{R}^{d'} \to \mathbb{R}^{d}.$$

If $\mathcal{G}$ is given, we also write $\kappa_{\mathcal{G}}(\psi) := \kappa(\psi,\mathcal{G})$.

While Definition 12 does not explicitly consider the closure of a graph, self-messages can be incorporated by setting $\mathcal{G} = \overline{\mathcal{G}}$. In practice, GNN layers are stacked sequentially, such that a GNN is a composition of $L \in \mathbb{N}$ GNN layers according to Definition 12:

$$\mathcal{K} : (\mathbb{R}^{d'})^{\mathcal{V}} \times 2^{\mathcal{V}\times\mathcal{V}\times\mathcal{M}} \to (\mathbb{R}^{d})^{\mathcal{V}}, \qquad \mathcal{K}(\psi,\mathcal{G}) = (\kappa_{\mathcal{G},L}\circ\cdots\circ\kappa_{\mathcal{G},1})(\psi).$$

In particular, generalized GNN layers represent graph embedding constructors. Given a subset $\mathcal{X} \subseteq \mathcal{V}$ and a base embedding $\psi \in (\mathbb{R}^{d'})^{\mathcal{V}}$, we obtain

$$\Phi : \Omega_{\mathcal{X}} \to (\mathbb{R}^{d})^{\mathcal{X}}, \quad \Phi(\mathcal{G})(v) = \mathcal{K}(\psi,\mathcal{G})(v),$$

with the resulting graph embedding $\phi = \Phi(\mathcal{G})$. The required support of ψ reduces to the L-hop neighbors of nodes $v \in \mathcal{X}$. In general, the n-hop neighborhood of $\mathcal{X}$ for $n \in \mathbb{N}$ is specified as

$$\mathcal{N}^n_{\mathcal{G}}(\mathcal{X}) := \bigcup\nolimits_{v\in\mathcal{X}} \mathcal{N}^n_{\mathcal{G}}(v) \quad \text{with} \quad \mathcal{N}^n_{\mathcal{G}}(v) = \bigcup\nolimits_{u\in\mathcal{N}^{n-1}_{\mathcal{G}}(v)} \mathcal{N}_{\mathcal{G}}(u), \quad \mathcal{N}^0_{\mathcal{G}}(v) := \{v\}.$$

The L-hop neighborhood is required since an L-layer GNNs performs L aggregation steps. Given $n_\ell = L - \ell + 1$, multi-layer GNNs can thus be decomposed into layers

$$\kappa_{\mathcal{G},\ell} : (\mathbb{R}^{d'_\ell})^{\mathcal{N}^{n_\ell}_{\mathcal{G}}(\mathcal{X})} \to (\mathbb{R}^{d_\ell})^{\mathcal{N}^{n_\ell-1}_{\mathcal{G}}(\mathcal{X})}, \quad \phi_\ell = \kappa_{\mathcal{G},\ell}(\psi_\ell).$$

For instance, for the first GNN layer $\kappa_{\mathcal{G},1}$, we obtain $n_1 = L$, and for the final GNN layer $\kappa_{\mathcal{G},L}$, we obtain $n_L = 1$. Analogous to MLPs, stacked GNNs are executed sequentially via $\phi_\ell = \kappa_{\mathcal{G},\ell}(\phi_{\ell-1})$, i.e., $d'_\ell = d_{\ell-1}$ with $d'_0 := d'$ and $d_L = d$, $\psi_\ell = \phi_{\ell-1}$ with $\phi_0 := \psi$ and $\phi_L = \phi$. In this work, we consider such sequential GNNs and node-specific outputs $\phi(v)$ used in downstream Machine Learning (ML) tasks such as node classification or link prediction. In particular, given the single-hop induced graph

$$\mathcal{H}_{\mathcal{G}}(v) := \{(u,w,m) \in \mathcal{G} \mid u = v \vee w = v\} \subseteq \mathcal{G}$$

we obtain the computation graph induced by an L-layer GNN for $\mathcal{X} \subseteq \mathcal{V}$:

$$\mathcal{H}^L_{\mathcal{G}}(\mathcal{X}) := \bigcup\nolimits_{v\in\mathcal{X}} \mathcal{H}^L_{\mathcal{G}}(v) \subseteq \mathcal{G} \quad \text{with} \quad \mathcal{H}^n_{\mathcal{G}}(v) = \bigcup\nolimits_{u\in\mathcal{N}^{n-1}_{\mathcal{G}}(v)} \mathcal{H}_{\mathcal{G}}(u).$$

Figure 4.2 illustrates the induced computation graph of a GNN during a single forward pass. Starting from base embeddings ψ, node representations are iteratively updated through successive aggregation layers, where each layer incorporates information from an expanding neighborhood. The depicted connections correspond to the edges in the computation graph $\mathcal{H}_{\mathcal{G}}^{L}(v)$, i.e., the subset of $\mathcal{G}$ contributing to the representation of a node v within L message-passing steps. The final embeddings $\phi(v)$ obtained at the output layer can then be used in downstream tasks.

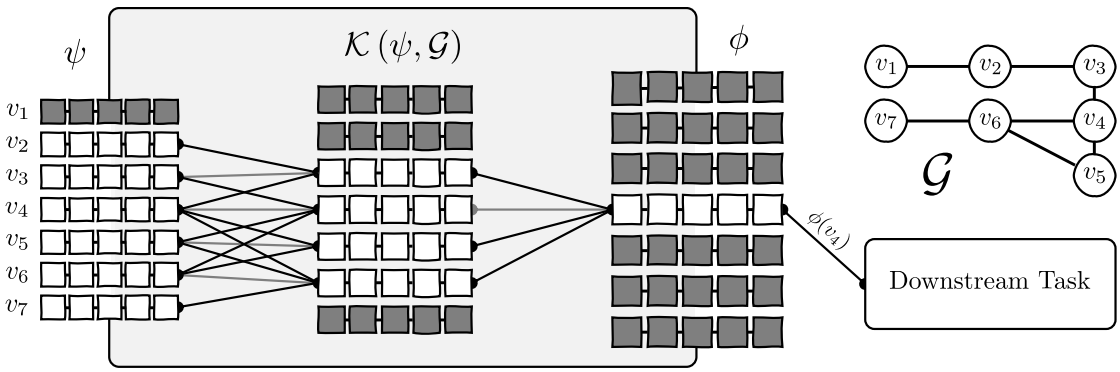


Figure 4.2: Computation graph of a GNN during a forward pass. Starting from base embeddings, node representations are iteratively refined through successive aggregation layers, each incorporating information from larger neighborhoods. The resulting embeddings are used as input for downstream tasks such as node classification or link prediction.

In practice, generalized GNN layers are typically not deterministically given, i.e., ψ and $\mathcal{K}$ are not provided. Rather, both the base embedding $\psi_\theta \in (\mathbb{R}^{d'})^{\mathcal{V}}$ and the neural transformations

$$F_{m,\iota,\theta} : \mathbb{R}^{d'} \to \mathbb{R}^{d}, \qquad c_{m,\iota,\theta} : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$$

are parametrized and learned from data. Thus, given $\mathcal{X} \subseteq \mathcal{V}$ and $\mathcal{G} \in \Omega_{\mathcal{X}}$, we obtain

$$\phi_\theta(v) = \mathcal{K}_\theta(\psi_\theta, \mathcal{G})(v) \;\text{ for }\; v \in \mathcal{X}$$

whose outputs depend jointly on the graph structure and parameters. These representations can be optimized under different training paradigms depending on the learning objective. In supervised settings, the embeddings ϕ_θ are trained with respect to downstream tasks such as node classification. In self-supervised settings, GNN outputs may be used within constructive graph embedding paradigms, for instance via contrastive learning objectives based on perturbed graph views. Alternatively, the outputs of a GNN can be incorporated into self-supervised objectives traditionally associated with self-contained models, such as factorization-based or transformation-based losses, thereby combining explicit graph-dependent construction with parameter optimization driven by structural objectives. As a result, we obtain a trained parameter setting $\theta^* \in \Theta$, which can be used to derive a deterministic graph embedding model for $\mathcal{X}' \subseteq \mathcal{V}$:

$$\Phi' : \Omega_{\mathcal{X}'} \to (\mathbb{R}^{d})^{\mathcal{X}'}, \quad \Phi'(\mathcal{G}')(v) = \mathcal{K}_{\theta^*}(\psi_{\theta^*}, \mathcal{G}')(v) \;\text{ for }\; v \in \mathcal{X}'.$$

Therefore, GNNs are often referred to as inductive, in the sense that they can adapt to graph updates and new nodes without retraining [191]. In contrast, graph embedding models that necessarily require retraining are referred to as transductive, including any self-contained variants. However, the inductive capabilities of GNNs are limited in standard implementations. Namely, training is based on a given node set $\mathcal{X} \subseteq \mathcal{V}$ and input graph $\mathcal{G}$, i.e., θ^* is optimized with respect to this information and any inductive behavior assumes that these trained parameters generalize across graphs. Moreover, assuming an L-layer GNN, only the base embeddings within the respective computation graphs are actually optimized, i.e., the ones of the nodes $\mathcal{V}(\mathcal{H}^L_{\mathcal{G}}(\mathcal{X}))$. Furthermore, intermediate representations are only meaningfully incorporated for corresponding labels $\mathcal{M}(\mathcal{H}^L_{\mathcal{G}}(\mathcal{X}))$. In fact, training of GNNs is usually restricted to these subsets, i.e., the information actually considered for training embeddings of nodes $\mathcal{X} \subseteq \mathcal{V}$ with respect to $\mathcal{G}$.
Importantly, the influence of newly added nodes on existing node representations depends critically on the depth of the model. For a single-layer GNN, newly inserted nodes cannot affect the representations of existing nodes at all, since message passing is applied only once and existing node embeddings are computed solely from their original neighborhoods. Consequently, newly added nodes are not incorporated into previously computed representations. Incorporating structural information from such nodes into existing embeddings therefore requires multi-layer GNNs, which enable information propagation across multiple hops. In such cases, however, the propagation still originates from the learned base embeddings ψ_{θ^*}, which remain tied to the original, transductive training setting of the base embeddings.
For instance, this limitation becomes particularly relevant under temporal constraints, i.e., when temporal trajectories of graphs are considered, with the abstract vertex set

$$\mathcal{V}_{\mathbb{T}} := \bigcup\nolimits_{t \in \mathbb{T}} \mathcal{V}_t,$$

such that $\mathcal{V}$ corresponds to $\mathcal{V}_{\mathbb{T}}$ and $\mathcal{X}$ corresponds to a snapshot $\mathcal{V}_t$ for $t \in \mathbb{T}$ with $\mathcal{V}_{\mathbb{T}} \setminus \mathcal{V}_t \neq \emptyset$. Such settings expose fundamental limitations of standard GNNs in dynamic environments. We revisit this scenario in Chapter 10, where we evaluate LoRE and NAVI, two approaches based on local self-reconstruction as a novel GNN-native graph embedding formalism. This formalism enables the incorporation of base embeddings for new or substantially updated nodes within single-hop procedures.

PART II

Knowledge Modeling in Industry 5.0

Learn how to see.
Realize that everything connects to everything else.

Leonardo Da Vinci

Building on the formal foundations of knowledge modeling with BPMN and RDF presented in Part I, this part examines technologies and approaches that support the management of human interventions in AI-assisted manufacturing processes under the dynamic conditions of Industry 5.0. The discussion builds on insights from the Teaming.AI project, which aimed to address Industry 5.0 requirements through knowledge graph technologies. While traditional knowledge graphs are largely static, with structures fixed at design time, the dynamic nature of inline human–AI collaboration in industrial environments requires a late-shaping design paradigm. In this context, we outline relevant approaches and lessons learned, addressing general challenges such as modeling domain expertise with a focus on knowledge integration, as well as challenges specific to industrial domains.

Chapter 5

Knowledge Modeling Requirements in Industry 5.0

As elaborated in Chapter 1, Industry 4.0 systems lack explicit system-spanning coordination and semantically grounded process execution. In response, Industry 5.0 introduces human-centeredness, resilience, and sustainability as guiding principles [38]. In contrast to automation-driven paradigms that primarily emphasize efficiency and technological autonomy [147], it repositions human expertise, judgment, and responsibility at the core of industrial production systems [155, 256], with AI conceived as a complementary capability embedded within socio-technical workflows rather than as a replacement for human decision-making [217]. Building on these conceptual foundations, this chapter translates the identified limitations into concrete requirements for knowledge modeling architectures in industrial environments. Collaborative industrial systems must coordinate heterogeneous actors, support structured intervention under changing conditions, and ensure traceability of decisions and responsibilities. From a knowledge modeling perspective, this implies representations that capture process dynamics, encode responsibility allocation, and maintain interpretable state trajectories over time. At the same time, industrial systems operate within semantically rich domains comprising machines, materials, quality standards, safety constraints, and organizational structures [183]. Effective Industry 5.0 architectures must therefore reconcile dynamic process representations with formal domain semantics. Using the Teaming.AI project as an empirical context, this chapter derives structural requirements for such integrated knowledge modeling approaches. It extends the notions of digital shadows and semantic domain knowledge bases toward semantic digital shadows and semantic digital twins, thereby establishing their architectural relationships. The remainder of this work further analyzes the process-centric and semantic perspectives in detail and demonstrates how their integration can act as an enabler of semantic digital twins for Industry 5.0.

5.1 The Teaming.AI Project

The Teaming.AI project provides the industrial and methodological foundation for the process-centric perspective considered in this thesis. As an EU-funded research initiative (Horizon 2020

ICT-38, January 2021–June 2024), it investigated how human–AI collaboration in manufacturing can be systematically structured in accordance with the principles of Industry 5.0, namely human-centeredness, resilience, and sustainability. Rather than positioning AI as an isolated assistance component, the project embedded human expertise and algorithmic capabilities into end-to-end production workflows, enabling coordinated decision-making and shared responsibility for system performance. A central objective was to design collaboration mechanisms that allow adaptive intervention and structured late shaping under changing production conditions.

Consortium and Industrial Use Cases

The consortium comprised 15 partners from eight European countries, including research institutions, universities, technology providers, and industrial enterprises. Coordinated by the Software Competence Center Hagenberg GmbH (SCCH), Austria, it combined expertise in artificial intelligence, machine learning, knowledge representation, human–machine interaction, software architecture, and industrial automation. This interdisciplinary composition ensured that both technical and socio-technical aspects of collaborative manufacturing processes were addressed, from algorithmic development to deployment and validation in operational environments. The developed concepts were validated in three industrial use cases representing complementary forms of structured human–AI collaboration:

1. **Transfer learning-based quality inspection** (Farplas, Turkey): AI-supported defect detection in injection moulding was combined with operator feedback and contextual expertise. Human interventions refined model behavior and increased robustness in the presence of data shifts and non-ideal production conditions.

2. **Machine diagnostics and parameter optimization** (Industrias Alegre, Spain): Machine learning methods supported anomaly detection and pattern recognition in injection moulding lines, while experienced operators contributed domain reasoning for root cause analysis and parameter adjustment during production.

3. **Ergonomics and risk prevention** (Goimek, Spain): AI-based monitoring systems were integrated with structured human assessments to analyze posture and movement, supporting continuous feedback and mitigation of occupational risks in large-part manufacturing.

Together, these use cases demonstrate how human-centered AI can be systematically integrated into operational workflows within real-world production environments as a socio-technical collaboration paradigm. They illustrate the necessity of explicit process models and orchestration mechanisms to coordinate agents, support adaptive intervention, and enable traceable collaboration in dynamic industrial environments. Beyond their application-specific objectives, these use cases exposed recurring structural challenges in managing collaborative human–AI systems. These challenges concerned the explicit coordination of tasks and responsibilities, the handling of model drift and evolving operational conditions, the traceability of decisions and interventions, and the integration of domain-specific knowledge into runtime processes. Rather than representing isolated technical issues, these observations point toward fundamental requirements for knowledge modeling architectures in Industry 5.0. The following section synthesizes these lessons and derives the corresponding structural implications.

5.2 Key Challenges and Architectural Requirements

This section builds upon the publication *"Managing human-AI collaborations within Industry 5.0 scenarios via knowledge graphs: key challenges and lessons learned"* by Franz Krause, Heiko Paulheim, Elmar Kiesling, Kabul Kurniawan, Maria Leva, Hector Diego Estrada-Lugo, Gernot Stübl, Nazim Üre, Javier Dominguez, Maqbool Khan, Pedro Demolder, Hans Gaux, Bernhard Heinzl, Thomas Hoch, Jorge Martinez-Gil, Agastya Silvina, and Bernhard A. Moser [122], published in *Frontiers in Artificial Intelligence* (2024).

Human–AI collaboration in industrial environments constitutes a socio-technical coordination problem rather than a purely technical integration task. Contemporary research in human–AI interaction emphasizes complementary cognitive strengths between automated systems and human decision-makers [11, 142, 158]. AI systems provide scalable pattern recognition and statistical generalization, whereas human operators contribute contextual awareness, normative reasoning, and adaptive problem solving. However, bounded rationality and cognitive limitations constrain human oversight, while AI models remain sensitive to dataset shifts and non-stationary operating conditions [28, 207]. Importantly, improvements in isolated model performance do not necessarily translate into improved socio-technical teaming performance [23, 156]. Effective collaboration depends on the alignment of interacting task models, teaming models, self-models, and shared representations. Industrial knowledge modeling must therefore operate at the system level rather than focusing solely on algorithmic accuracy. While the present discussion is motivated by human–AI teaming scenarios, the underlying challenges extend to socio-technical collaborations in Industry 5.0 more broadly, including interactions among humans, automated systems, and organizational structures. Based on empirical insights from the Teaming.AI project, this section derives five key knowledge modeling requirements for Industry 5.0 systems. These requirements constitute the central findings of this chapter and provide a conceptual foundation for the modeling approach developed in the subsequent sections.

R1: Explicit Coordination and Orchestration

Effective collaboration requires structured interaction patterns rather than informal supervision. Human-in-the-loop, human-on-the-loop, and governance-level intervention must be embedded within explicitly defined task sequences and escalation pathways. Coordination quality depends on clear responsibility allocation, transparent decision boundaries, and structured authority transitions [23, 156]. Moreover, collaboration unfolds across multiple phases, including initialization, operational execution, adaptation under drift, and retrospective reflection. These phases require explicit representation of intervention points and responsibility transfers.

> **R1:** Collaborative industrial systems must be represented through system-spanning, executable process models that explicitly define task sequencing, responsibility allocation, validation checkpoints, and escalation mechanisms.

Such representations provide the structural foundation for orchestrating heterogeneous actors within shared workflows and enable multi-level intervention across the system lifecycle.

R2: Temporal Operational Traceability

Industrial production systems operate under continuously evolving conditions. Dataset shift and concept drift are structural characteristics of real-world environments rather than exceptional disturbances [28, 207]. In practice, deviations often emerge gradually and require retrospective reconstruction to understand their causes. Because collaboration unfolds over time, knowledge modeling must preserve the historical trajectory of decisions, adjustments, and human interventions. Without temporal traceability, adaptation processes cannot be analyzed or validated.

> **R2:** The runtime behavior of collaborative processes must be recorded as structured, time-resolved execution traces that allow reconstruction of operational dynamics.

These temporally evolving traces form digital shadows of collaborative processes. They enable auditability, lifecycle analysis, compliance verification, and resilience assessment across phases of operation and adaptation.

R3: Formal Semantic Grounding

Industrial collaboration unfolds within semantically rich domains comprising machines, materials, quality standards, organizational roles, and regulatory requirements [40, 146]. These elements are typically distributed across heterogeneous systems and organizational boundaries. Research on knowledge representation and ontology-based integration highlights the necessity of shared conceptual models for ensuring interoperability and consistent interpretation across fragmented environments [55, 209]. Without formal semantic grounding, process traces remain syntactic records lacking shared meaning. Domain entities and their relations must therefore be represented independently of specific process executions.

> **R3:** Industrial knowledge architectures must include formally defined domain models that represent entities, relations, and constraints in a semantic manner.

Corresponding semantic domain knowledge bases ensure conceptual coherence across systems but do not, by themselves, capture the dynamic behavior of collaborative processes.

R4: Contextualized Integration of Process and Semantics

Neither execution traces nor semantic models alone enable context-aware interpretation of collaborative behavior. Shared situational awareness depends on linking runtime events to the semantic entities, constraints, and environmental conditions under which they occur [23, 108]. Trust calibration and explainability require decisions to be interpretable within their domain context rather than merely reconstructed chronologically. System-level coherence emerges only when process dynamics and domain semantics are integrated. This integration supports alignment of mental models and reduces ambiguity regarding authority and responsibility.

> **R4:** Operational execution traces must be semantically grounded in domain knowledge to enable context-aware interpretation of industrial processes.

This requirement motivates integrated representations in which digital shadows are enriched with formal semantics, combining temporal evolution with conceptual grounding.

R5: Adaptive Feedback and Evolution

Industry 5.0 emphasizes resilience, sustainability, and structured late shaping of socio-technical systems. Effective collaboration therefore requires not only descriptive reconstruction but also predictive reasoning, explainable feedback, and the ability to incorporate new observations without invalidating prior knowledge [11, 156]. Adaptation must occur at the system level, preserving traceability while enabling inductive updates under evolving conditions.

> **R5:** Knowledge representations must support predictive reasoning, explainable feedback, and inductive adaptation under evolving operational conditions.

When digital shadows are extended with feedback capabilities, they evolve into digital twins capable of shaping the underlying production system in a bidirectional manner [31, 126].

5.3 Semantic Digital Representations as Enablers of Industry 5.0

Taken together, the requirements derived above imply a layered architectural structure:

- Formal process models and orchestration mechanisms (R1),
- Runtime execution traces forming digital shadows (R2),
- Semantic domain knowledge as a semantic digital model (R3),
- Integrated semantic digital shadows (R4),
- Adaptive semantic digital twins with feedback capability (R5).

Each layer addresses structural limitations of the preceding one. Process models provide explicit coordination but lack runtime history. Digital shadows capture temporal dynamics but lack shared semantics. Semantic domain knowledge provides conceptual grounding but lacks operational evolution. Semantic digital shadows integrate time and domain knowledge, extended by semantic digital twins with adaptive feedback. Together, these layers establish a representational foundation required for human-centered, resilient, and sustainable Industry 5.0 systems.
More specifically, formal process models enable transparent coordination structures and clearly defined responsibilities, thereby supporting human-centered collaboration between operators, engineers, and AI components. Runtime execution traces and digital shadows preserve the temporal evolution of system behavior and allow deviations, interventions, and adaptations to be reconstructed, which is essential for resilient operation under changing industrial conditions. Semantic domain knowledge provides a shared conceptualization of machines, materials, organizational roles, and constraints, ensuring consistent interpretation across heterogeneous subsystems and enabling interoperable integration of industrial knowledge. When operational traces are grounded in such semantic knowledge, semantic digital shadows enable contextual interpretation of system behavior, support explainability of decisions, and facilitate informed human oversight. Finally, semantic digital twins extend this representation through predictive reasoning

and adaptive feedback mechanisms that enable continuous improvement of system performance, resource utilization, and process efficiency. In this way, the layered architecture derived from the requirements R1–R5 provides the knowledge representation capabilities necessary to operationalize the guiding principles of Industry 5.0 by supporting human-centered collaboration, resilient adaptation to evolving conditions, and sustainable optimization of industrial processes. Conventional digital representations often lack explicit semantic grounding across heterogeneous industrial subsystems [214]. Process execution logs derived from workflow systems naturally constitute digital shadows because they capture temporal trajectories of operational behavior. They record what has happened over time and therefore provide dynamic state traces of the real system. However, such traces are frequently syntactic records without formally defined domain semantics, limiting their interpretability beyond procedural reconstruction. Conversely, RDF domain knowledge bases provide shared conceptual structures and machine-interpretable semantics [50]. They formally describe entities, relations, and constraints within a domain and thereby enable logical inference and semantic validation. Yet these knowledge bases typically represent static snapshots of domain knowledge rather than time-resolved operational trajectories. As a result, they offer conceptual coherence without intrinsic dynamic evolution.

A semantic digital shadow emerges when temporally evolving process traces are explicitly grounded in formal domain knowledge, thereby combining dynamic evolution with conceptual coherence. This integration allows operational events to be interpreted within a shared semantic framework. Logical inference can derive implicit relations, detect inconsistencies, and provide structured explanations for observed system behavior. However, adaptation remains limited at this stage. While reasoning mechanisms can enrich and reinterpret the evolving state representation, they do not inherently establish a closed feedback loop toward the physical system. The semantic digital shadow remains observational and interpretative: it supports understanding and analysis of ongoing processes, but it does not autonomously influence system behavior.

A semantic digital twin extends this integration by incorporating predictive reasoning, decision support, and adaptive feedback mechanisms that actively affect the real system. In this setting, inferred knowledge can trigger structural adjustments, parameter adaptations, or coordinated human–AI interventions. The representation therefore becomes intervention- and feedback-capable and closes the loop between digital representation and physical operation. The distinguishing characteristics of these representation layers are summarized in Table 5.1.

Table 5.1: Characteristics of digital representation layers in Industry 5.0 architectures.

Layer	**Dynamics**	**Semantics**	**Adaptation**
Digital Shadow	✓	✗	✗
Semantic Domain Knowledge	✗	✓	✗
Semantic Digital Shadow	✓	✓	✗
Semantic Digital Twin	✓	✓	✓

The following chapter analyzes process and semantic knowledge bases realized via BPMN and RDF in industrial practice, examines their strengths and limitations in isolation, and motivates their integration into semantic digital shadows as a prerequisite for semantic digital twins.

Chapter 6

From Digital Shadows to Semantic Digital Twins

In the previous chapter, we derived structural requirements for knowledge modeling architectures in Industry 5.0 and introduced the architectural progression from digital representations to semantic digital representations of collaborative processes. These findings highlight the existence of two orthogonal knowledge branches within industrial knowledge architectures: process domain knowledge and semantic domain knowledge. This chapter analyzes these two branches in isolation and demonstrates why neither suffices on its own to fulfill the requirements R1–R5 derived in the previous chapter as prerequisites for enabling human-centeredness, resilience, and sustainability in Industry 5.0 scenarios. Their systematic integration motivates concrete realizations of semantic digital shadows and, ultimately, semantic digital twins.

6.1 Digital Shadows via BPMN Knowledge Bases

In this work, process-driven digital shadows arise within collaborative industrial environments through the deployment of BPMN knowledge base systems. While traditional industrial automation relies on embedded control logic and machine-centric coordination [189], Industry 5.0 scenarios increasingly require explicit modeling of socio-technical collaboration, responsibility allocation, and structured escalation [256]. In such settings, workflow orchestration systems become central architectural components. We therefore examine the transition from classical control logic to collaborative process modeling, describe the lifecycle of BPMN knowledge bases, and analyze how BPMN execution traces can give rise to knowledge graph trajectories as dynamic digital shadows within the Teaming.AI context.

6.1.1 From Industrial Control Logic to Collaborative Process Orchestration

As outlined in Chapter 1, industrial production systems are traditionally implemented using domain-specific control technologies such as programmable logic controllers, supervisory control and data acquisition systems, or manufacturing execution systems [216]. In highly automated environments, process logic is often embedded directly into machine control programs

or specialized orchestration layers optimized for deterministic, low-latency machine interaction [143]. These architectures prioritize efficiency and reliability but rarely externalize coordination logic in a form that explicitly models human participation or responsibility transitions.
Industry 5.0 scenarios characterized by human-centered collaboration fundamentally alter this architectural landscape [155, 217]. For instance, collaborative production workflows involve human operators validating AI predictions, engineers adjusting parameters under drift conditions, and supervisors managing exception handling and compliance requirements [196]. These interaction patterns resemble business processes as structured sequences of tasks executed by heterogeneous actors under explicitly defined coordination rules. Consequently, workflow modeling formalisms such as BPMN become suitable tools for representing collaborative industrial workflows [76, 171, 174], thereby fulfilling requirement R1.

6.1.2 Design and Deployment Lifecycle

As discussed in Section 2.2, the lifecycle of BPMN knowledge bases comprises design, deployment, execution, and logging phases. During the design phase, domain experts and process analysts model collaborative workflows using graphical modeling capabilities. Within the layered knowledge modeling formalism considered in this work, BPMN process models constitute the ABox of a BPMN meta-knowledge base. The models define abstract task sequences, branching conditions, synchronization points, message exchanges, and data interactions. They encode normative collaboration structures that specify how human and service agents are expected to interact. Human tasks, service tasks invoking AI components, and events can be represented within a unified formalism [174]. By externalizing collaboration logic, BPMN process models make coordination structures explicit and analyzable.
During deployment, the same BPMN models assume the role of the TBox within a BPMN knowledge base, as they provide the structural vocabulary for concrete process executions. At this stage, process models are serialized into XML according to the BPMN 2.0 specification [164] and deployed to workflow engines. Integration points with external systems are configured, including REST interfaces, database connectors, and bindings to AI components.

6.1.3 Execution and Workflow Engines in Industrial Environments

Execution of deployed BPMN models is managed by workflow and process orchestration engines such as Camunda [44], Flowable [73], Activiti [4], or jBPM [181]. These engines interpret process definitions, manage state transitions, coordinate parallel flows, and handle exception logic. In collaborative industrial environments such as those investigated in the Teaming.AI project, engines like Camunda are integrated into existing IT landscapes through service-oriented architectures [134]. Human operators interact with the engine via task dashboards, while automated service tasks invoke AI modules, analytics services, enterprise systems, or manufacturing execution systems. The workflow engine thus acts as coordination middleware between human agents, AI components, and legacy production systems, separating orchestration logic from machine-level control logic.

Each execution of a deployed process model instantiates a concrete process instance. During runtime, the workflow engine records detailed execution events, including activity lifecycle transitions, gateway evaluations, variable updates, and escalation events. These events are typically persisted in relational database tables maintained by the engine, forming audit logs and history tables. In modern architectures, events may additionally be propagated to event streaming infrastructures such as Apache Kafka [14] or stored in log archives for monitoring and analytics. The resulting data structures contain temporally ordered traces enriched with timestamps, resource identifiers, and contextual variables, enabling reconstructions of process trajectories.

6.1.4 BPMN Knowledge Base Trajectories as Digital Shadows

From the knowledge modeling perspective adopted in this thesis, these execution traces constitute a BPMN domain population that instantiates the structural vocabulary defined by the deployed process models. At each point in time, the set of deployed process models provides the terminological structure, while recorded execution events populate this structure with instance-level assertions. Over time, this yields a dynamically evolving trajectory of BPMN knowledge base states reflecting operational behavior. Importantly, this trajectory may also incorporate evolution at the model level, as process definitions can be versioned, redeployed, and migrated during operation. The resulting trajectories therefore capture both instance-level execution dynamics and controlled adaptations of the underlying domain model.
Such runtime trajectories constitute digital shadows, as they provide time-resolved, observational representations of collaborative industrial processes derived from workflow instrumentation. Information flows unidirectionally from the physical production environment to the digital representation. While BPMN deployments suffice to generate such trajectories of BPMN-based execution traces, these representations remain syntactic and lack formally grounded domain semantics. Digital shadows are therefore temporally evolving yet semantically shallow representations of collaborative production processes, fulfilling the requirement R2 of temporal traceability while lacking explicit semantic integration across industrial systems.

6.2 Semantic Domain Knowledge via RDF Knowledge Bases

Whereas BPMN knowledge bases externalize collaborative process logic and generate time-resolved execution trajectories, industrial environments simultaneously maintain extensive domain knowledge concerning production machines and materials, product specifications, quality standards, safety constraints, organizational structures, and regulatory requirements [139, 180, 183, 228]. In contemporary semantic information architectures, such domain knowledge is frequently represented using RDF-based ontologies and knowledge graphs. The adoption of RDF in industrial contexts is motivated by several factors. First, RDF constitutes a W3C standard for graph-based data representation, enabling syntactic interoperability across heterogeneous systems [50]. Second, ontology languages layered on top of RDF, such as RDFS and OWL, provide formally defined semantics that support logical reasoning, constraint validation, and classification [1, 96]. Third, RDF enables uniform representation of both TBox and ABox knowledge within a shared formalism, thereby facilitating integration of distributed data sources [15].

The semantic representation and integration of manufacturing knowledge have therefore become an important application area for Semantic Web technologies. Numerous ontology and knowledge graph initiatives have been proposed to structure industrial knowledge and enable interoperability across heterogeneous systems. Examples include asset administration shell ontologies for Industry 4.0 interoperability [103], manufacturing ontologies capturing production assets and capabilities [133], semantic models for predictive maintenance and condition monitoring [21, 182], knowledge graphs for production system modeling [22, 40, 97, 146, 214], and ontology-based representations of industrial standards and product specifications [59, 105, 222]. These approaches demonstrate how RDF-based semantic representations can provide a shared conceptual backbone for integrating distributed industrial information sources.

In the context of the Teaming.AI project, such semantic representations were employed to contextualize domain knowledge across the investigated industrial use cases. Domain ontologies were used to capture and structure engineering knowledge originating from various industrial artifacts and standards. Examples include ontological representations of failure mode and effects analysis for describing potential faults and mitigation strategies, industrial process documentation such as injection moulding process parameterizations, and machine learning components used within human-AI workflows. In addition, domain-specific industrial standards such as EUROMAP 77 [68] for injection moulding machine communication were incorporated to represent machine telemetry and operational parameters in a semantically structured form (cf. Chapter 8). Through such representations, heterogeneous domain knowledge concerning machines, processes, diagnostics, and analytics could be integrated into a unified semantic framework supporting cross-system interpretation and interoperability.

From the layered knowledge modeling perspective adopted in this thesis, RDF ontology languages constitute the modeling language, domain ontologies represent the domain model, and concrete RDF triple sets describe the corresponding domain population. In contrast to BPMN, which clearly separates model and execution, RDF encodes both terminological and assertional knowledge using the same structural mechanism. Its primary strength thus lies in the explicit formalization of conceptual structures and their reuse across heterogeneous technical systems.

6.2.1 Ontology Modeling and Integration Paradigms

In industrial practice, semantic domain models are rarely constructed monolithically. The diversity of subdomains involved in modern production environments, ranging from mechanical engineering and materials science to logistics, quality management, human resource management, and regulatory compliance, renders the construction of a single comprehensive ontology impractical. The heterogeneity of conceptual perspectives, terminologies, and modeling granularity across these domains makes fully horizontal integration in the sense of a unified monolithic ontology difficult, if not infeasible. Instead, ontology modeling typically follows a modular and layered approach [19, 61, 201, 202]. Domain knowledge is decomposed into subdomain ontologies, each capturing a coherent conceptual fragment. One module may describe production assets and machine capabilities, another AI specifications, and a third organizational roles and compliance constraints. Subject-matter experts contribute domain-specific distinctions, while knowledge engineers ensure logical consistency and controlled alignment.

Horizontal integration in its strict sense would entail constructing a single unified ontology covering all industrial subdomains within one coherent conceptual framework. In such an approach, concepts and relations across domains such as machine engineering, materials science, logistics, and organizational management would be represented within a single shared ontology. While theoretically appealing, this approach faces substantial practical challenges due to domain heterogeneity, differing abstraction levels, evolving terminology, and organizational boundaries. The conceptual diversity of industrial domains makes it difficult to maintain a single ontology that remains both expressive and manageable as systems evolve.

Consequently, ontology engineering in industrial environments typically adopts a vertical integration strategy [7, 93]. Instead of constructing a monolithic ontology, specialized subdomain ontologies are developed independently and subsequently aligned through a shared upper ontology or domain core ontology. In this approach, subdomain ontologies are connected through alignment to abstract superclasses and shared upper-level relations. For instance, machines and human operators may both be modeled as subclasses of a general agent concept, while physical resources may be subsumed under a shared asset abstraction. Vertical integration therefore contrasts with the monolithic ideal of horizontal integration by preserving modular subdomain models while establishing coherence through an abstract shared layer [47].

Such taxonomical alignment establishes terminological compatibility and supports logical reasoning across modules. However, the resulting integration primarily affects the terminological layer. While classes and properties become aligned through shared abstractions, instance-level data typically remain organized within their respective subdomain populations. Connectivity is therefore achieved at the conceptual level, whereas population-level relations across subdomains are not inherently induced by such taxonomical alignment (cf. Figure 6.1).

6.2.2 RDF Management Systems and Deployment Architectures

In industrial semantic architectures, RDF graphs are typically exposed through standardized access endpoints, which constitute the primary integration interface between semantic data and enterprise systems. SPARQL Protocol and RDF Query Language (SPARQL) is the W3C-standardized query and update language for RDF data [29, 179]. It enables declarative graph pattern matching over triple sets and supports operations such as selection, projection, aggregation, graph construction, and modification of triples. Through HTTP-based endpoints, enterprise applications, analytics services, and AI components can query and update semantic domain knowledge in a uniform and technology-independent manner. From an architectural perspective, the SPARQL endpoint defines the decisive system boundary for semantic integration. External systems interact with semantic domain knowledge exclusively through this interface, independently of how the underlying RDF graph is implemented.

It is therefore essential to distinguish between an RDF graph and the infrastructure that governs it. An RDF graph, as elaborated in Section 3.3, is a timeless set of triples. The system behind a SPARQL endpoint manages such graphs and may allow their modification via update operations or through changes in connected data sources. Through successive insertions, deletions, or transformations, different graph states may arise over time. In accordance with the terminology introduced earlier, the temporally ordered succession of such graph states is referred to as a graph

trajectory. What is commonly described in the literature as a dynamic knowledge graph therefore corresponds, in this thesis, to a graph trajectory governed by the underlying management infrastructure. Two principal architectural paradigms are used to implement such SPARQL-exposed RDF graphs in practice: virtualized infrastructures and materialized triplestore-based infrastructures. The following paragraphs briefly outline the characteristics of both approaches.

In virtualized deployments, RDF data is not physically stored. Instead, existing enterprise data sources, most notably relational databases, are exposed as virtual RDF graphs using ontology-based data access mechanisms [43]. Systems such as Ontop translate incoming SPARQL queries into SQL queries executed against underlying relational backends [235]. The ontology provides a semantic layer over pre-existing data without duplicating it in triple form. In industrial practice, virtualization is predominantly applied to population-level data [85, 152]. Large-scale instance data such as asset registries, production records, maintenance logs, or configuration data remain stored in relational systems while being semantically exposed through mappings. Domain ontologies themselves are typically maintained as curated RDF triple sets due to their comparatively small size and logical structure. In virtualized settings, the exposed RDF graph reflects the current state of the underlying data sources at query time.

In materialized deployments, RDF data is physically stored within dedicated triplestores [99]. Systems such as Apache Jena TDB [213], Virtuoso [67], and GraphDB [165] persist triples in index structures optimized for graph-shaped data and provide transactional storage and scalable query processing. Ontologies and instance-level data are serialized in formats such as TTL or RDF/XML and loaded into the materialized system [50, 178]. The triplestore exposes a SPARQL endpoint through which queries and updates are executed. Generally, both virtualized and materialized infrastructures provide uniform semantic access via SPARQL. Their architectural role in industrial systems is primarily the harmonization and structured exposure of domain knowledge rather than the intrinsic modeling of operational coordination.

6.2.3 Population-Level Characteristics and Dynamic Limitations

Although both virtualized and materialized infrastructures technically allow graph modification and iterative graph trajectories, industrial RDF knowledge bases predominantly exhibit catalogue-like characteristics [22]. They describe assets, product classifications, configuration states, documentation references, and regulatory constraints [146]. While instance data may evolve, such evolution typically concerns descriptive or configurational updates rather than structured operational traces. High-frequency operational data, including sensor streams, machine telemetry, or real-time measurements, are usually managed in specialized time-series or streaming infrastructures [189]. Although such data can be transformed into RDF representations, large-scale triple generation for fine-grained measurements introduces scalability and maintenance challenges. Consequently, current industrial RDF deployments primarily capture structural aspects of the domain rather than continuously evolving characteristics.

Virtualized endpoints may reflect changing enterprise data more frequently, since updates occur in the underlying operational systems. Materialized triplestores, by contrast, often contain curated semantic data that change comparatively slowly. In both cases, however, the generation of graph trajectories is not driven by explicit coordination logic but by external data updates

or manual synchronization processes. The management infrastructure governs graph trajectories only insofar as the surrounding systems produce updates. It does not inherently encode workflow semantics or collaborative process structure.

From the perspective of the requirements derived in Chapter 5, semantic domain knowledge bases fulfill the need for formal semantic grounding and interoperability across heterogeneous systems (R3). They provide conceptual coherence, logical consistency checking, and machine-interpretable meaning. However, they do not inherently provide explicit coordination structures (R1) or systematically generate execution trajectories required for operational traceability (R2). In isolation, semantic domain knowledge bases are therefore semantically rich but isolated and operationally passive. The graph trajectories governed by their infrastructures typically represent evolving semantic catalogues rather than digital shadows of collaborative industrial processes.

6.3 From Orthogonality to Semantic Digital Shadows and Twins

The preceding sections reveal a structural orthogonality between BPMN and RDF knowledge bases. BPMN knowledge bases inherently generate execution traces grounded in explicit coordination logic but lack formal semantic integration across heterogeneous subdomains. RDF knowledge bases, by contrast, provide rich terminological grounding and theoretical interoperability but do not intrinsically encode operational trajectories tied to collaborative workflows.

This orthogonality suggests a complementary integration strategy. Instead of attempting to vertically align heterogeneous subdomain ontologies exclusively through abstract upper-level connectors, the process dimension itself can be introduced as an explicit horizontal integration layer. Industrial production is inherently process-driven and coordinated through task sequences, resource allocations, and responsibility transfers [116]. By representing BPMN process models and their execution traces as RDF graphs, process semantics can be incorporated directly into the semantic domain layer. Conceptually, this transformation yields a process ontology that is not merely an abstract meta-layer but an operational connector ontology. Instances of heterogeneous subdomain ontologies, such as machines, operators, materials, or AI agents, become related through their participation in shared process instances. In contrast to vertical integration, which primarily aligns classes taxonomically, horizontal process-based integration induces explicit instance-level connectivity through operational participation, as shown in Figure 6.1.

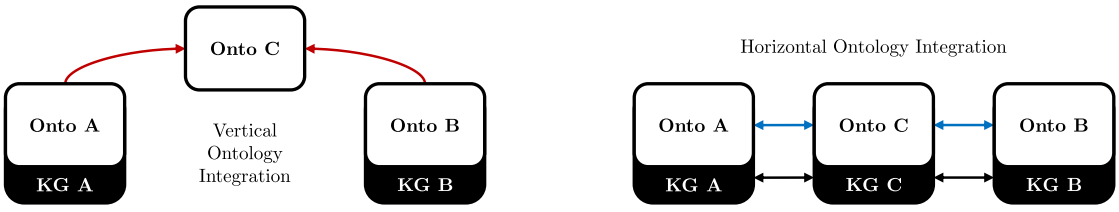


Figure 6.1: Comparison of ontology integration paradigms in industrial semantic architectures. Left: Vertical integration aligns ontologies via a shared upper ontology, establishing conceptual compatibility without inducing instance-level connectivity. Right: Mediated integration enables explicit instance-level relations across subdomains, thereby supporting operational connectivity.

Through this integration, BPMN-derived execution trajectories are represented as graph trajectories within the semantic domain layer. The resulting RDF knowledge base trajectories no longer represent static catalogues but time-resolved, semantically grounded components of collaborative industrial processes. Operational events become linked to domain entities, contextual constraints, and organizational roles, allowing system behavior to be interpreted within a shared conceptual framework. This integration fulfills Requirement R4 by grounding process execution traces in formal domain semantics and thereby enabling context-aware interpretation of industrial processes. The resulting representations correspond to *semantic digital shadows*: dynamically evolving yet semantically interpretable models of real-world industrial operations.
A *semantic digital twin* emerges when such semantically grounded trajectories are further coupled with reasoning, validation, and adaptive feedback mechanisms [126]. In this setting, the knowledge representation does not merely describe past and present system states but actively supports predictive reasoning and decision-making. For example, learned representations derived from graph-based ML models, such as adaptive knowledge graph embeddings or GNNs, can identify structural patterns in the evolving knowledge graph and generate explainable predictions regarding system behavior or process configurations. When these predictive capabilities are integrated into the operational environment, inferred knowledge can trigger adjustments of process parameters, support human decision-making, or initiate coordinated interventions. Such feedback mechanisms realize Requirement R5 by enabling adaptive feedback under evolving operational conditions while preserving semantic traceability of system behavior.
The orthogonality between process-centric and ontology-centric knowledge bases is therefore not resolved by dissolving their distinctions but by systematically integrating their complementary strengths within a unified RDF-based representation. Process knowledge contributes structured execution trajectories, while semantic domain knowledge provides conceptual grounding and interoperability. Their integration produces semantically enhanced digital shadows of collaborative processes, which can subsequently evolve into adaptive semantic digital twins when combined with reasoning and learning mechanisms.
In the following, due to the implementation decisions made within the Teaming.AI project, the focus of this work lies on BPMN-based process executions in industrial environments under the vision of Industry 5.0. Nevertheless, the conceptual findings presented throughout the remainder of this thesis are not limited to industrial production systems. They can be transferred analogously to other process-driven domains in which formalized process modeling languages and corresponding execution logs exist and where semantic domain knowledge can be attached to these executions. Examples include healthcare workflows and clinical treatment pathways, logistics and supply chain coordination processes, business process management in enterprise information systems, energy grid operation and infrastructure management, as well as scientific workflows and data processing pipelines.
Consequently, the architectural principles derived in this work are not specific to industrial manufacturing but characterize a general paradigm for integrating operational process traces with semantic knowledge representations, thereby enabling semantically grounded digital shadows and adaptive digital twins across real-world systems.

PART III

Process-Driven RDF Knowledge Graphs as Semantic Digital Shadows

---

What we observe is not nature itself,
but nature exposed to our method of questioning.

Werner Heisenberg

This part investigates how BPMN-based process knowledge can be integrated with heterogeneous RDF knowledge bases to jointly represent process structure and execution in a semantically enriched manner. Existing approaches typically model process elements as instance-level entities, resulting in an abstraction mismatch that limits their role as integrative components within a knowledge base. To address this, we represent process models at the ontology level, thereby capturing process executions as semantically grounded knowledge graph instances. Building on this formalization, we introduce sBPMN together with a corresponding system architecture based on BPMN orchestration engines such as Camunda, which enables the systematic derivation of semantic digital shadows from process executions. In doing so, we demonstrate how the knowledge modeling requirements R1–R4 formulated in Chapter 5 can be realized in real-world industrial settings, reflecting the final outcome of the Teaming.AI project.

Chapter 7

From BPMN Knowledge Bases to RDF Knowledge Bases

Parts of this chapter build upon the publication *"On the Representation of Dynamic BPMN Process Executions in Knowledge Graphs"* by Franz Krause, Kabul Kurniawan, Elmar Kiesling, Heiko Paulheim, and Axel Polleres [119], published in *Knowledge Graphs and Semantic Web*, Springer (2023).

In the previous chapter, we established that BPMN- and RDF-based knowledge bases provide complementary perspectives on industrial knowledge modeling. While BPMN knowledge bases externalize collaborative process logic and generate operational execution trajectories, RDF knowledge bases provide a semantically grounded formalism for representing heterogeneous domain knowledge. These paradigms therefore capture complementary aspects of industrial systems: dynamic process behavior on the one hand and formally structured domain semantics on the other. Realizing semantic digital shadows as requirement R4 from Chapter 5 requires integrating these perspectives so that process execution trajectories and domain entities can be represented within a shared knowledge graph. In this sense, a semantic digital shadow can be understood as a trajectory of knowledge graphs driven by process execution. Within this trajectory, concrete process executions and production entities form the domain population of shared RDF knowledge bases. Domain entities such as machines, operators, materials, or quality artifacts are described through corresponding ontologies and instantiated within knowledge graphs, enabling semantic integration and reasoning across heterogeneous subdomains. To realize such representations, process execution traces must therefore be linked both to the structural roles they play within a process model and to the surrounding domain knowledge.

This requirement implies that a knowledge base for semantic digital shadows must capture two types of information. On the one hand, it must represent concrete process executions as instance-level observations that evolve over time. On the other hand, it must integrate these observations with semantic descriptions of the participating domain concepts and their relationships to process models at the ontology level. Consequently, the knowledge graph must provide a representation of process executions that connects them with domain semantics.

In this chapter, we therefore investigate how such representations can be realized when integrating BPMN-based process knowledge with RDF knowledge bases. We first review existing approaches that represent BPMN constructs through ontological models. In these approaches, the BPMN modeling language itself is typically formalized as an ontology, effectively treating BPMN as a domain of discourse. Hence, concrete process models are represented as instance-level descriptions within the resulting knowledge graph. While this representation enables semantic descriptions of process models, it implicitly positions process models at the same abstraction level as domain entities encoded in RDF knowledge graphs. Building on this observation, we analyze the implications of such representations for the integration objectives introduced in the previous chapters. In particular, representing process models as instance-level structures limits their ability to serve as horizontal integration mechanisms across industrial subdomain ontologies. Instead, we argue that process models should be treated as part of the domain model itself, defining structural roles that connect domain entities during process execution. This perspective preserves the abstraction hierarchy between process models and process executions: process models become part of the ontology layer of the RDF knowledge base, and concrete process executions populate the corresponding knowledge graph. Such a representation provides a principled foundation for expressing process execution trajectories as semantically grounded knowledge graph trajectories and thereby enables the realization of semantic digital shadows as knowledge modeling requirement R4 for Industry 5.0 environments.

7.1 Related Work

Several approaches have explored the use of ontologies to formalize concepts from Business Process Modeling (BPM). In this context, BPMN ontologies provide machine-interpretable representations of process modeling constructs and their relationships. Examples include the BPMN-native ontologies [157, 190], and the BPMN Business Ontology (BBO) [13]. These ontologies define vocabularies and formal semantics for BPMN elements such as tasks, gateways, sequence flows, and events, thereby providing a conceptual foundation for representing BPMN-based process knowledge within semantic frameworks. Building on such conceptualizations, ontology-based process modeling (OBPM) aims to enhance the semantic clarity and interoperability of BPM systems [215]. In OBPM approaches, process models can be annotated with concepts from domain ontologies to provide additional contextual meaning and enable semantic reasoning [58, 71, 186]. These approaches facilitate semantic enrichment of process models, but their adoption in industrial environments remains limited, partly due to the considerable effort required to develop and maintain application-specific BPM ontologies [48].

Beyond semantic annotations, several works have investigated the representation of BPMN models directly within RDF knowledge graphs. In such approaches, BPMN constructs are represented using ontological vocabularies, allowing process models to be translated into RDF graphs that can be queried, integrated with other semantic data sources, and analyzed using Semantic Web technologies. For instance, BPMN2KG [20] represents BPMN elements as individuals that instantiate concepts defined in the BBO ontology, thereby enabling the integration of process knowledge with other instance data. Although these approaches enable semantic representations of process models, they primarily focus on describing the structure of BPMN models rather

than the dynamic executions of business processes. In practice, the runtime behavior of processes is typically recorded separately in event logs generated by workflow engines and stored in standardized formats such as XES [3]. These logs capture the sequence of executed process elements but remain largely disconnected from both the formal process model and surrounding domain knowledge represented in knowledge graphs. Consequently, existing approaches often treat BPMN process models themselves as instance-level structures within a knowledge graph, while concrete execution traces are either omitted entirely or represented outside the semantic knowledge base. From the perspective of semantic digital shadows, however, this separation is problematic. To enable semantically grounded representations of process execution trajectories, execution observations must appear as individuals within the knowledge graph and must be linked both to their structural roles in the process model and to the participating domain entities.

7.2 Abstraction Levels in BPMN-to-RDF Representations

All approaches discussed above provide ontologies that capture the BPMN modeling language or fragments of it. Accordingly, concrete process models are represented as instances within resulting knowledge graphs. In this perspective, BPMN constitutes the domain of discourse, i.e., the ontology describes BPMN constructs such as tasks, gateways, and events, and process models appear as assertions of these constructs. While this representation is consistent with the meta-model perspective of BPMN, it leads to a shift in abstraction levels when attempting to represent process executions within the same knowledge base. In particular, process models occupy the instance layer of the knowledge base, leaving no dedicated population layer for representing concrete process executions. As a result, execution traces must either be represented outside the knowledge base or merged with the abstraction level of process models.

However, from the perspective of semantic digital shadows, the domain of interest is not the BPMN modeling language itself but the industrial processes described by BPMN models. In this setting, process models act as abstract structural descriptions of the process domain, and concrete process executions represent instance-level observations within that domain. As a consequence, process models should form part of the RDF domain model, whereas execution traces populate the corresponding domain population. Figure 7.1 illustrates this abstraction mismatch. BPMN ontology approaches model the domain of BPMN itself, causing process models to appear as individuals within the knowledge graph. In contrast, the representation required for semantic digital shadows models the domain of industrial process models, where process models define the domain model and process executions populate the knowledge graph.

In the following, we therefore analyze how existing approaches that represent process models as individuals within a knowledge graph can still be reused, since this representation naturally follows the abstraction hierarchy between the BPMN modeling language and concrete BPMN process models. However, treating process models as instance-level entities prevents them from acting as a subdomain module for enabling horizontal integration across existing domain ontologies. To preserve the abstraction hierarchy and still enable semantically grounded execution trajectories, we instead incorporate both the BPMN modeling language and concrete process models into the TBox of an RDF knowledge base, with process executions populating the ABox.

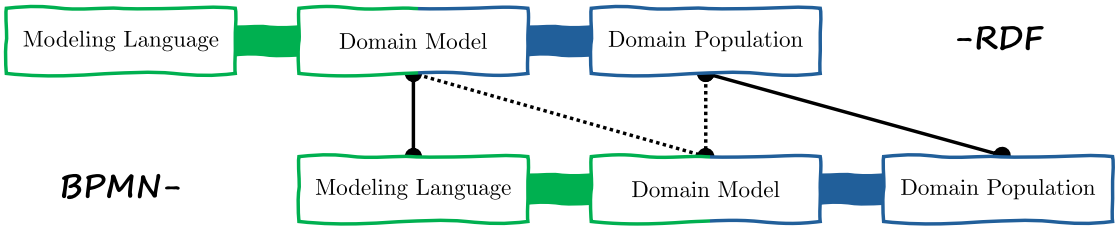


Figure 7.1: Abstraction levels in BPMN-to-RDF representations. BPMN ontologies represent the BPMN modeling language as the domain, whereas semantic digital shadows require modeling the domain of industrial processes with process executions as domain population.

7.3 Mapping BPMN Knowledge Bases to RDF Knowledge Bases

The abstraction mismatch discussed above becomes particularly evident when process execution data must be integrated with domain knowledge in real-world industrial scenarios. To illustrate the practical implications of the representation problem discussed above, we consider a recurring human-AI collaboration pattern observed across several Teaming.AI use cases. The scenario reflects typical interactions between automated analysis components and human operators in industrial production environments. Figure 7.2 shows an abstracted BPMN process in which an automated quality inspection component is combined with human validation and potential machine adjustment, based on static encodings of failure mode adjustments.

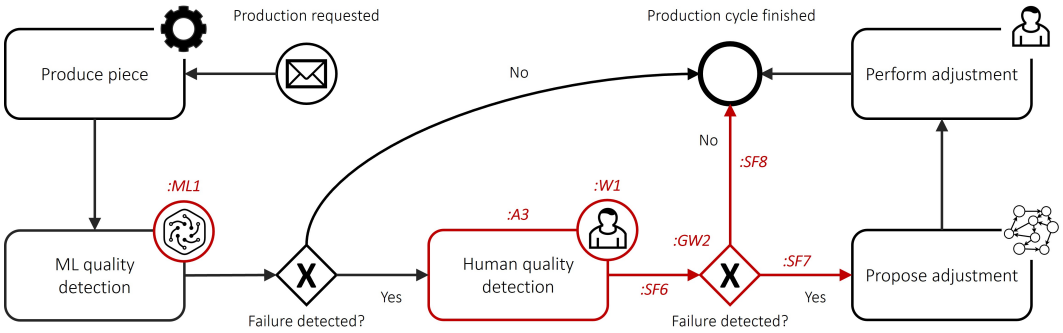


Figure 7.2: Minimal BPMN process model of a human-AI collaboration process :P0 observed in the Teaming.AI use cases, covering several industrial components.

Monitoring the quality of manufactured parts and improving product quality through machine setting recommendations requires combining process knowledge, operational data, and domain expertise. In the motivating scenario, a production machine manufactures parts that are evaluated against quality requirements. Automated visual inspection based on ML reduces the workload of human operators but requires structured intervention when a prediction indicates a failure. Once a failure is confirmed, operators may adjust machine parameters. Domain knowledge about failure types and appropriate countermeasures can be retrieved from an RDF domain knowledge base to support such decisions [21]. While such knowledge bases can support static lookups

(e.g., retrieving known failure types and adjustment strategies), the collaborative dynamics arise at runtime. For example, it may be of interest which combinations of AI agents :ML1 and human agents :W1 have led to the execution of the sequence flow :SF8, representing a worker vetoing a preceding ML prediction. Such analyses could reveal whether specific ML models, training datasets, or operational contexts lead to systematic differences between automated predictions and human assessments. Addressing these questions requires execution traces to be represented as queryable individuals that are linked (i) to the structural roles they play within the process model and (ii) to the participating domain entities. Consequently, process executions must become part of a semantically grounded RDF knowledge graph trajectory that integrates process behavior with domain knowledge. The appropriate representation level of process models and their components, such as the sequence flow :SF8, is therefore non-trivial.

7.3.1 Representing Process Models as Knowledge Graph Instances

Most existing approaches that translate BPMN models into RDF knowledge bases treat process models and their components as individuals in the ABox. Examples include systems such as BPMN2KG [20], where BPMN constructs are represented as instances of BPMN ontology classes defined in BPMN ontologies such as BBO. In this representation, activities, gateways, and sequence flows appear as individuals that instantiate the corresponding BPMN concepts defined in the TBox. In the following, we also consider BBO as an example BPMN ontology to illustrate these representations.

Process models are therefore encoded as instance-level structures that explicitly describe the control-flow relations between process elements. This encoding corresponds to the blue structures shown in Figure 7.3, where process components such as activities and gateways are represented as individuals connected via instance-level properties.

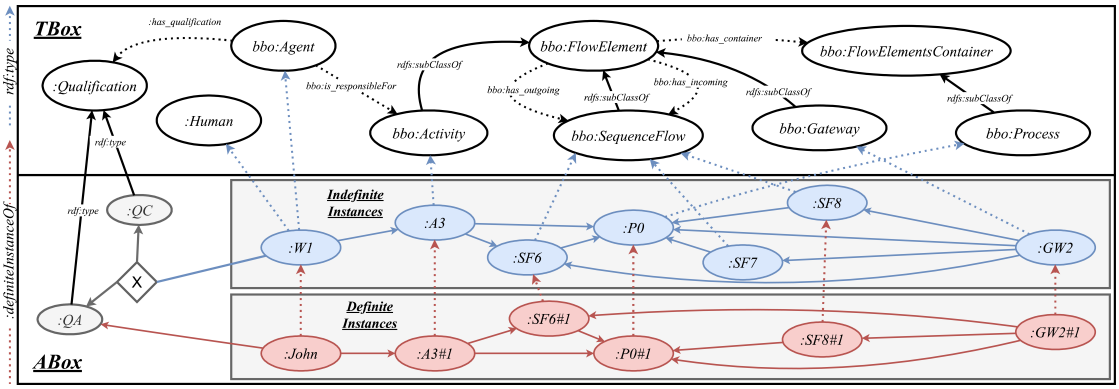


Figure 7.3: Knowledge graph encoding of a process model (blue) and process execution (red).

Within this representation, process executions can be incorporated by introducing additional individuals that represent executed process elements. To maintain the distinction between model elements and their runtime executions, an additional abstraction layer can be introduced by distinguishing between indefinite instances and definite instances. Model elements such as :A3

represent indefinite instances describing the process model, whereas concrete execution events such as :A3#1 correspond to definite instances of these model elements.
Using this mechanism, dynamic BPMN executions can be represented as individuals within the knowledge graph and linked to participating agents, resources, and contextual domain entities. While this approach enables the representation of execution traces within RDF knowledge graphs, it introduces conceptual limitations. Since process models appear as instance-level entities in the ABox, they cannot act as structural components of the ontology layer. As a result, process models cannot serve as reusable ontology modules that connect heterogeneous domain ontologies through OWL axioms. This limitation becomes particularly relevant when BPMN processes are intended to act as horizontal integration mechanisms between subdomain ontologies. If process models are encoded solely as instance-level structures, OWL-based class hierarchies and semantic constraints cannot directly reference the structural elements of the process model. Thus, the ability to use process models as ontological connectors between heterogeneous industrial subdomains is significantly reduced.

7.3.2 Representing Process Models as Ontology Concepts

To address this limitation, BPMN process models can instead be incorporated into the TBox of the knowledge base. In this representation, process model elements are defined as subclasses of the corresponding concepts specified in BPMN ontologies such as BBO.
For example, the BPMN process :P0 in Figure 7.2 can be encoded as a subclass of bbo:Process, whereas the gateway :GW2 represents a subclass of bbo:Gateway. OWL class expressions can then be used to specify structural constraints that characterize valid process executions. For instance, the class :GW2 can be defined to be equivalent to

(bbo:has_container **exactly 1** :P0) **and** (bbo:has_incoming **exactly 1** :SF6)
and (bbo:has_outgoing **exactly 1** (:SF7 **or** :SF8)).

OWL constructs such as owl:Restriction, owl:qualifiedCardinality, and owl:onClass enable the formal specification of such execution constraints and domain policies. As another example, quality assurance :QA or quality control :QC can represent the required qualifications of agents performing the activity :A3, as implied by its superclass

inverse bbo:is_responsibleFor **only** (:has_qualification **min 1** ({:QA, :QC})).

TBox encodings of BPMN process models capture valid process flows as well as constraint-based execution policies that can be enriched with domain knowledge. By representing BPMN process models as TBox concepts, the abstraction hierarchy between process models and process executions is preserved. Process models become part of the domain ontology, with concrete executions populating the ABox as individuals of these model-level concepts. This representation enables process models to function as structural connectors between heterogeneous domain entities and allows OWL axioms to directly reference process structures.
However, even though this representation allows expressive modeling of execution constraints, OWL lacks the ability to express certain overlapping constraints. For example, in the equivalent

class expression for :GW2, it cannot be specified that an instance of :SF6 must be contained within the same instance of :P0. More expressive constraint- or rule-based formalisms such as SHACL [115] or SWRL [102] are therefore required, increasing the complexity of the ontology and potentially impairing the adoption of ontology-based process modeling (cf. Section 7.1). Moreover, although no restrictions exist regarding the dynamics of BPMN executions represented as individuals in the ABox, this is not the case for BPMN models encoded through OWL axioms. Since process flows are captured through class axioms, updates to process models may lead to consistency violations with respect to previously recorded executions.

7.4 Implications for Process-Driven Knowledge Graphs

In this chapter, we discussed two approaches for representing BPMN process models in RDF knowledge bases. Existing works commonly encode process models as individuals in the ABox. While this allows process executions to be represented alongside the model structure, it prevents process models from acting as ontology components that connect heterogeneous domain ontologies through OWL axioms. In contrast, incorporating process models into the TBox preserves the abstraction hierarchy between process models and process executions and enables structural constraints and domain policies to be expressed using OWL. Although OWL cannot capture certain overlapping execution constraints, these limitations arise precisely because ontology-based representations provide expressive modeling capabilities that instance-level encodings do not offer. We therefore propose to extend BPMN ontology formalisms with concrete domain process models that collectively form the TBox of an RDF knowledge base, with the ABox reserved for actual individuals such as process executions and participating domain entities. This separation preserves the conceptual distinction between models and instances and is consistent with the interpretation of knowledge graphs as representations describing real-world individuals and their relationships. The following chapter introduces the sBPMN framework, which operationalizes this representation by introducing a core BPMN ontology and providing a concrete implementation for generating process-driven RDF knowledge graph trajectories.

Chapter 8

The sBPMN Framework

This chapter builds upon the publication *"sBPMN: A Framework for Capturing Business Process Semantics in Dynamic Knowledge Graphs"* by Franz Krause, Heiko Paulheim, Amparo Vázquez, and Alejandro Espert [123], published in the *Proceedings of the 13th Knowledge Capture Conference (K-CAP 2025)*, ACM (2025).

The previous chapter established that integrating BPMN-based process knowledge with RDF knowledge bases requires representations that capture both process structure and concrete process executions. Existing approaches that translate BPMN models into semantic representations typically encode process models and their components as individuals in the ABox of an RDF knowledge base. This allows process structures to be represented at the assertional level, but it prevents process models from acting as ontology components in the TBox that connect heterogeneous domain ontologies through OWL axioms.

Several works have investigated semantic representations of BPMN models using Semantic Web technologies. Early approaches based on BPMN 1.1 lack support for important BPMN 2.0 constructs such as choreography, conversation diagrams, and standardized execution metadata, which are required for compatibility with modern orchestration engines [80, 190]. Among BPMN 2.0-based approaches, two limitations frequently arise. First, many ontologies are designed independently of the official BPMN specification [13, 90]. While these ontologies provide conceptual vocabularies for representing BPMN 2.0 constructs, mismatches between their modeling assumptions and the syntax of executable BPMN files hinder alignment with BPMN-driven tools such as orchestration engines. Second, approaches that explicitly consider the BPMN 2.0 specification often rely on its extended textual documentation rather than the formal XML schema [89, 157]. Because the documentation omits several elements defined in the schema and introduces implicit or compositional semantics, such ontologies do not fully capture the structural constraints defined at the schema level. A notable exception is the IPPR ontology[1], which translates the BPMN 2.0 XML schema into OWL. However, this approach introduces additional abstraction layers and inconsistencies with BPMN execution semantics, limiting its usefulness for reasoning over executable process models.

[1] `https://github.com/reiterma13/IPPR`

A further limitation concerns the representation of runtime behavior. Most semantic BPMN approaches focus on static process models and neglect the integration of actual process executions. BPMN2KG [20] partially addresses this issue by representing process models as punned entities, i.e., resources that appear both as abstract classes and as individuals (cf. Chapter 7). Although this structure allows processes to be instantiated within semantic representations, it prevents process models from serving as horizontal connectors between subdomain ontologies. More generally, in the course of the previous chapter, we highlighted that representing process models as ontology classes improves modularity and interpretability.

To address the limitations of encoding BPMN knowledge bases within RDF knowledge bases, we introduce semantic BPMN (sBPMN), a framework for representing BPMN process knowledge in a manner that preserves the abstraction hierarchy between process models and process executions, ultimately enabling semantic digital shadows driven by BPMN process executions as requirement R4 from Chapter 5. BPMN process models are thus defined through XML documents conforming to the official BPMN 2.0 schema [164] and translated into the domain ontology layer. Concrete process executions populate the knowledge graph and are linked to participating domain entities. Listing 8.1 shows a simplified fragment of an XML process definition.

```
<bpmn:process id="pr967">
  <bpmn:startEvent id="ev845" name="Start"/>
  <bpmn:userTask id="ta100" name="Review Request"/>
  <bpmn:endEvent id="ev541" name="End"/>
  <bpmn:sequenceFlow id="sf228"
                     sourceRef="ev845"
                     targetRef="ta100"/>
  <bpmn:sequenceFlow id="sf035"
                     sourceRef="ta100"
                     targetRef="ev541"/>
</bpmn:process>
```

Listing 8.1: Excerpt of a BPMN 2.0 XML process definition.

Accordingly, Figure 8.1 shows the corresponding diagrammatic representation of the induced, minimal process model. The red labels indicate the identifiers of the BPMN elements defined in the XML representation and were added for illustration purposes.

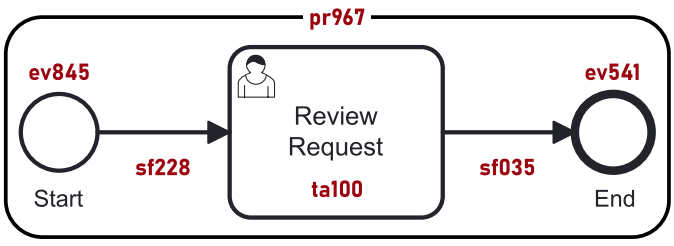


Figure 8.1: BPMN diagram with labeled start event, user task, end event, and sequence flows.

The structural elements defined in such executable process descriptions form the basis for the semantic representation proposed in this work. The following section therefore introduces the sBPMN ontology, which derives its conceptual vocabulary directly from the BPMN 2.0 XML schema and provides the foundation for representing process models and their executions within dynamic RDF knowledge graph trajectories as semantic digital shadows within Teaming.AI.

8.1 The sBPMN Ontology

To enable semantic enrichment of BPMN process models and executions, we introduce the sBPMN ontology[2]. As a direct translation of the core BPMN 2.0 XML schema and its semantics into OWL axioms and SHACL validity constraints, it ensures interoperability across BPMN tools and extensions. Figure 8.2 illustrates the hierarchy of BPMN 2.0-based process definitions and the central role of the semantic schema file, including a snapshot that illustrates the taxonomical structure of userTask. It also demonstrates how these concepts are represented in OWL and SHACL using the sBPMN prefixes sbpmnc (classes) and sbpmnp (properties).

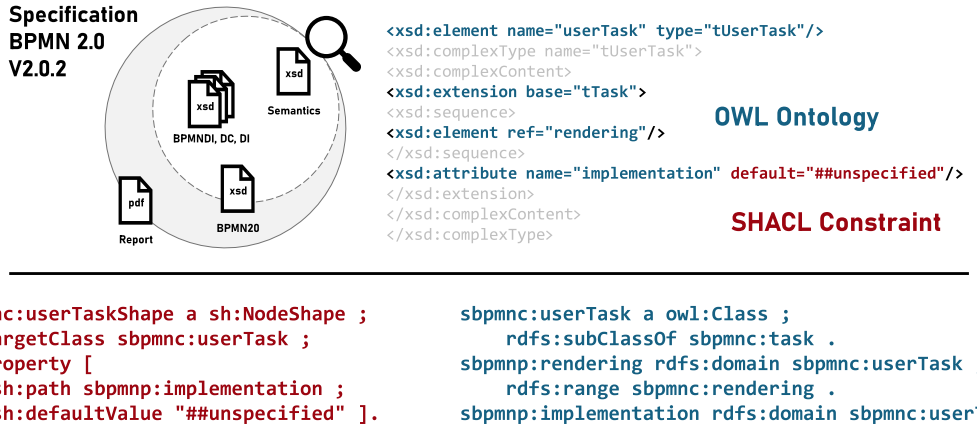


Figure 8.2: Overview of the sBPMN ontology extraction, illustrating how the core BPMN 2.0 XML schema is translated into OWL classes and properties and constrained using SHACL shapes. This enables formal reasoning and validation of BPMN models.

Executions are asserted through rdf:type within the knowledge graph, enabling real-time querying and reasoning over process instances as soon as dynamic executions are observed, thereby inducing evolving knowledge graph trajectories. In this context, domain-specific extensions can be defined via rdfs:subClassOf relations to base BPMN concepts. Given a156-8c53e6 as the identifier of a concrete execution of the user task onto:ta100 from the BPMN engine, we obtain:

```
onto:ta100 owl:subClassOf sbpmnc:userTask .
kg:a156-8c53e6 rdf:type onto:ta100 .
```

[2] https://sBPMN.github.io/2.0/index.html

8.1.1 Ontology Extraction from the BPMN Schema

The sBPMN ontology is derived from the Semantic.xsd schema provided by the OMG as part of the BPMN 2.0 standard [164]. This schema defines the core XML structure of valid BPMN process definitions and constitutes the formal specification used by BPMN modeling tools and orchestration engines. Although not an ontology itself, the schema implies a conceptual structure that can be interpreted as an ontology through its hierarchy of types and attribute definitions.
Complex types such as tTask and tGateway correspond to domain ontology classes in sBPMN (e.g., sbpmnc:task, sbpmnc:gateway). Inheritance relations expressed through xsd:extension are translated into subclass hierarchies using owl:subClassOf.
Sequence elements represent structural links between BPMN components. These relations are encoded as OWL object properties (e.g., sbpmnp:targetRef), whose domains and ranges are aligned with the corresponding type constraints defined in the XML schema.
Data attributes are modeled as OWL data properties. An exception is made for attributes of type xsd:anyURI or xsd:IDREF, which reference other BPMN elements and are therefore represented as object properties.
Typed attributes are defined via xsd:simpleType restrictions that constrain values to enumerated xsd:string or xsd:token literals.[3] While this representation suffices for XML validation, it conflicts with Semantic Web principles, as semantics are encoded as strings rather than structured resources. We therefore introduce a dedicated OWL class for each enumerated value and link it to its parent class through equivalence axioms, expressed through owl:equivalentClass. In addition to IRI-based typing in SPARQL queries and knowledge graph applications, this modeling approach enables logical inference by OWL reasoners. As exemplified in Listing 8.2, a BPMN transaction with the attribute method="##Compensate" is inferred as an instance of sbpmnc:compensateTransaction:

```
sbpmnc:compensateTransaction rdf:type owl:Class ;
    rdfs:subClassOf sbpmnc:transaction ;
    owl:equivalentClass [ owl:intersectionOf (
        sbpmnc:transaction
        [   rdf:type owl:Restriction ;
            owl:onProperty sbpmnp:method ;
            owl:hasValue "##Compensate" ] ) ] .
```

Listing 8.2: OWL class definition for a BPMN transaction with an attribute value, where the attribute method="##Compensate" induces class membership via owl:equivalentClass.

As a subproperty of dcterms:isPartOf, sbpmnp:containedIn is introduced to account for nested XML elements. It represents hierarchical structures of process components in a semantically meaningful way, such as a sbpmnc:process containing an sbpmnc:sequenceFlow. Ultimately, the ontology was generated semi-automatically, resulting in a total of **166** classes, including **31** equivalent classes, **127** object properties, and **44** data properties. Notably, BPMN process

[3] In the BPMN 2.0 XML schema, "##" prefixes denote enumerated string tokens.

dynamics are primarily expressed through sequence flows and events, and activities themselves are modeled as atomic units [8]. Listing 8.3 demonstrates how sBPMN can be used to represent the onto:sf228 sequence flow class from the example in Figure 8.1, whose sbpmnp:sourceRef is an instance onto:ev845 and whose sbpmnp:targetRef is an instance of onto:ta100. Moreover, it is contained in exactly one instance of the process onto:pr967.

```
onto:sf228 rdfs:subClassOf [
    owl:intersectionOf (
        sbpmnc:sequenceFlow
        [ rdf:type owl:Restriction ;
          owl:onProperty sbpmnp:sourceRef ;
          owl:onClass onto:ev845 ;
          owl:qualifiedCardinality 1 ]
        [ rdf:type owl:Restriction ;
          owl:onProperty sbpmnp:targetRef ;
          owl:onClass onto:ta100 ;
          owl:qualifiedCardinality 1 ]
        [ rdf:type owl:Restriction ;
          owl:onProperty sbpmnp:containedIn ;
          owl:onClass onto:pr967 ;
          owl:qualifiedCardinality 1 ] ) ] .
```

Listing 8.3: OWL class restriction for a BPMN sequence flow specifying its source, target, and containing process via qualified cardinality constraints.

In BPMN files, identifiers such as sf228 are scoped locally, which can lead to naming conflicts across files. To preserve locality while avoiding such collisions, we interpret identifiers as relative to the source file. The URI prefix of each resource is derived from the corresponding file path, ensuring consistent differentiation of identically named elements across BPMN files:

```
@prefix onto: <{file_path}/{file_name}.bpmn#> .
```

Accordingly, we propose a modular domain ontology design, as shown in Figure 8.3, in which each BPMN file is enhanced by a corresponding OWL ontology. Thus, the BPMN-based RDF domain model contains a collection of such files. This setup supports decentralized development, allowing domain experts to define BPMN models independently. As a result, it also reduces upfront complexity by letting each process model focus on its domain, thereby maintaining locality and conceptual clarity, since cross-model dependencies are only defined when required. An integration layer manages abstract, reusable process components to enable modularity and reasoning over shared concepts. For instance, an abstract user task integration:reviewTask with integration:reviewTask rdfs:subClassOf sbpmnc:userTask may specify high-level semantics, such as alignment with domain knowledge, while omitting execution-level structural constraints such as incoming or outgoing sequence flows. Executable tasks such as onto:ta100 from the model layer can then be declared as subclasses, i.e., onto:ta100 rdfs:subClassOf integration:reviewTask. This design ensures global consistency and preserves local autonomy of BPMN models.

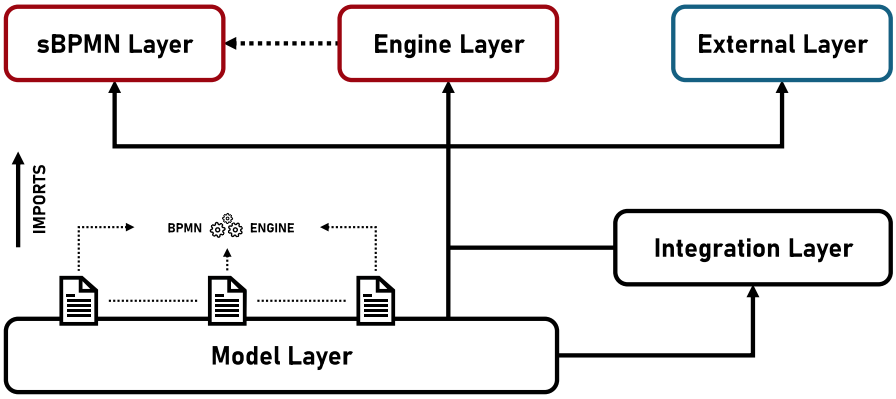


Figure 8.3: Modular process model ontology architecture: executable BPMN process models are represented in the process model layer, while the integration layer defines abstract, reusable process components. They are specified as extensions of the sBPMN layer and may be aligned with BPMN engine data and external knowledge.

Beyond supporting independent process model development and the maintenance of abstract, reusable BPMN components, the modular ontology design proposed in Figure 8.3 also facilitates integration with BPMN engine metadata and external knowledge resources. In particular, the integration layer serves as the horizontal connector of subdomain ontologies, i.e., we can specify how production entities are incorporated into process executions. While many BPMN engines introduce concepts not covered by the BPMN 2.0 specification (such as tenants in Camunda [44]), most remain conceptually aligned with BPMN and can be mapped to sBPMN concepts. Similar to the BPMN 2.0 XML schema, they are not encoded as ontologies and thus require transformations before they can be used with sBPMN (e.g., to specify which tenants may perform a task). Section 8.2 discusses how this can be achieved for Camunda.
Regarding the integration with external data, common alignment targets include domain ontologies or general-purpose ontologies such as FOAF [64], DC Terms [62], or schema.org [87]. By linking processes and their components encoded via sBPMN to external concepts, they gain contextual expressiveness and reusability across organizational boundaries. For instance, the BPMN user task ta100 from the example in Figure 8.1 can be linked to a public concept from DBpedia, either directly or through its integration-level superclass reviewTask as follows:

```
@prefix dbpedia: <http://dbpedia.org/resource/> .

onto:ta100 owl:subClassOf integration:reviewTask .
integration:reviewTask owl:subClassOf dbpedia:Review .
```

Thus, onto:ta100 (process model layer) inherits characteristics from integration:reviewTask (integration layer), which in turn is linked to dbpedia:Review (external layer). Such relationships can be instantiated in corresponding knowledge graphs as property assertions via RDF updates, enabling real-time monitoring and cross-system reasoning over business operations, e.g., every instance of onto:ta100 is inferred to be an instance of dbpedia:Review.

8.1.2 Constraint Modeling with SHACL

To formalize and validate complex structural constraints defined in the BPMN 2.0 XML schema, we adopt SHACL. In our context, SHACL enables the validation of process executions against schema-defined rules. We use SHACL to capture:
Cardinality constraints, which define how many times an attribute may occur, e.g., requiring at least or at most one value.
Type attribute classes, which denote attributes restricted to a predefined set of values (e.g., enumerated types).
Default values, which represent fallback values defined in the BPMN schema when no instance-level attribute value is provided.

As shown in Listing 8.4, if no value is specified, sbpmnc:transaction is assigned the default sbpmnp:method value "##Compensate". Together with the owl:equivalentClass axioms discussed in Section 8.1.1, OWL reasoning and SHACL validation can jointly infer the corresponding classes or identify constraint violations.

```
sbpmnc:transactionShape a sh:NodeShape ;
sh:targetClass sbpmnc:transaction ;
sh:property [ sh:path sbpmnp:method ;
              sh:defaultValue "##Compensate" ] .
```

Listing 8.4: SHACL node shape defining a default value for the attribute sbpmnp:method of sbpmnc:transaction, supporting validation and inference in combination with OWL axioms.

8.2 sBPMN System Architecture and Implementation

Building on the semantic representation of process models introduced through sBPMN in the previous section, we provide a dockerized system architecture[4], as shown in Figure 8.4. This architecture enables the derivation of process-driven trajectories of knowledge graphs that constitute semantic digital shadows of BPMN process executions orchestrated by a BPMN engine. The architecture was developed as part of the Teaming.AI project within the industrial environment of Industrias Alegre. The corresponding insights are discussed in Section 8.3.
The sBPMN framework, grounded in the sBPMN ontology, provides a vocabulary for encoding process executions within a knowledge graph. However, capturing these executions requires extracting information from orchestration engines that manage workflow instances at runtime. In this work, we consider Camunda [44], a widely used open-source BPMN 2.0 engine deployed at Industrias Alegre. We focus on Camunda 7, which shares its Activiti-based relational logging mechanism with other engines such as Flowable [134]. As illustrated in Figure 8.4, this H2 logging database is virtualized to retrieve real-time execution data encoded via sBPMN-based process descriptions and linked to both the ontology and additional domain knowledge.

[4] https://github.com/sBPMN/2.0

It is important to note that Camunda 7 is not the most current release at the time of submitting this thesis. Starting with version 8, Camunda introduced a Zeebe-based message-stream architecture that replaced the former Activiti foundation. The Activiti-based H2 execution database assumed in this work is therefore no longer a component of newer versions. Instead, Camunda 8 continuously emits process event streams that can be consumed by subscribing to them. While this simplifies data retrieval and improves integration flexibility, it shifts responsibility for storage and aggregation to the integrator. Nevertheless, even though the internal architecture has changed significantly, the principles of dynamic RDF data retrieval remain similar in practice. Execution log data can still be stored in a relational database or transformed directly into RDF, although relational storage often integrates more easily with existing infrastructures and provides a stable, scalable foundation for semantic access. The architecture described in the following therefore applies similarly to Camunda 8, with the relational layer being optional. We argue, however, that even for Camunda 8 it is advantageous to persist process data in a relational database and virtualize it on demand. In this work, we thus focus on Camunda 7, as it reflects the setup at Industrias Alegre and the prevalent use of Activiti-based engines in current BPM practice. A simplified prototype demonstrating how the same principles apply to Camunda 8 through a single-triplestore setup is available in the project repository.

8.2.1 H2 Execution Log Schema

Camunda 7 (and other Activiti-based engines) can be configured to persist execution-related data in a lightweight H2 relational database following schema conventions introduced by Activiti [163]. The schema contains several logging tables, including:

- **ACT_HI_PROCINST** records completed process instances together with start and end times, durations, and instance states.
- **ACT_HI_ACTINST** captures traces of execution logs such as tasks, gateways, and events.
- **ACT_HI_TASKINST** extends activity data for user tasks with additional attributes such as assignees, claim times, and timestamps.

These logs provide execution-level metadata including performers (ASSIGNEE_), timestamps (START_TIME_, END_TIME_), hierarchical relations (SUPER_PROCESS_INSTANCE_ID_), and engine-specific fields such as business keys. This information can be essential for constructing a dynamic knowledge graph trajectory as a digital shadow of process executions. However, it extends beyond standard BPMN 2.0 semantics and is therefore not represented in the sBPMN ontology. To bridge this gap, an engine-specific ontology can be derived from the official documentation[5] and formalized within a dedicated engine module of the ontology. This allows the logs to be exposed as virtualized knowledge graphs, enabling semantic access to execution data.

[5] https://docs.camunda.org/manual/latest/reference/bpmn20/custom-extensions/

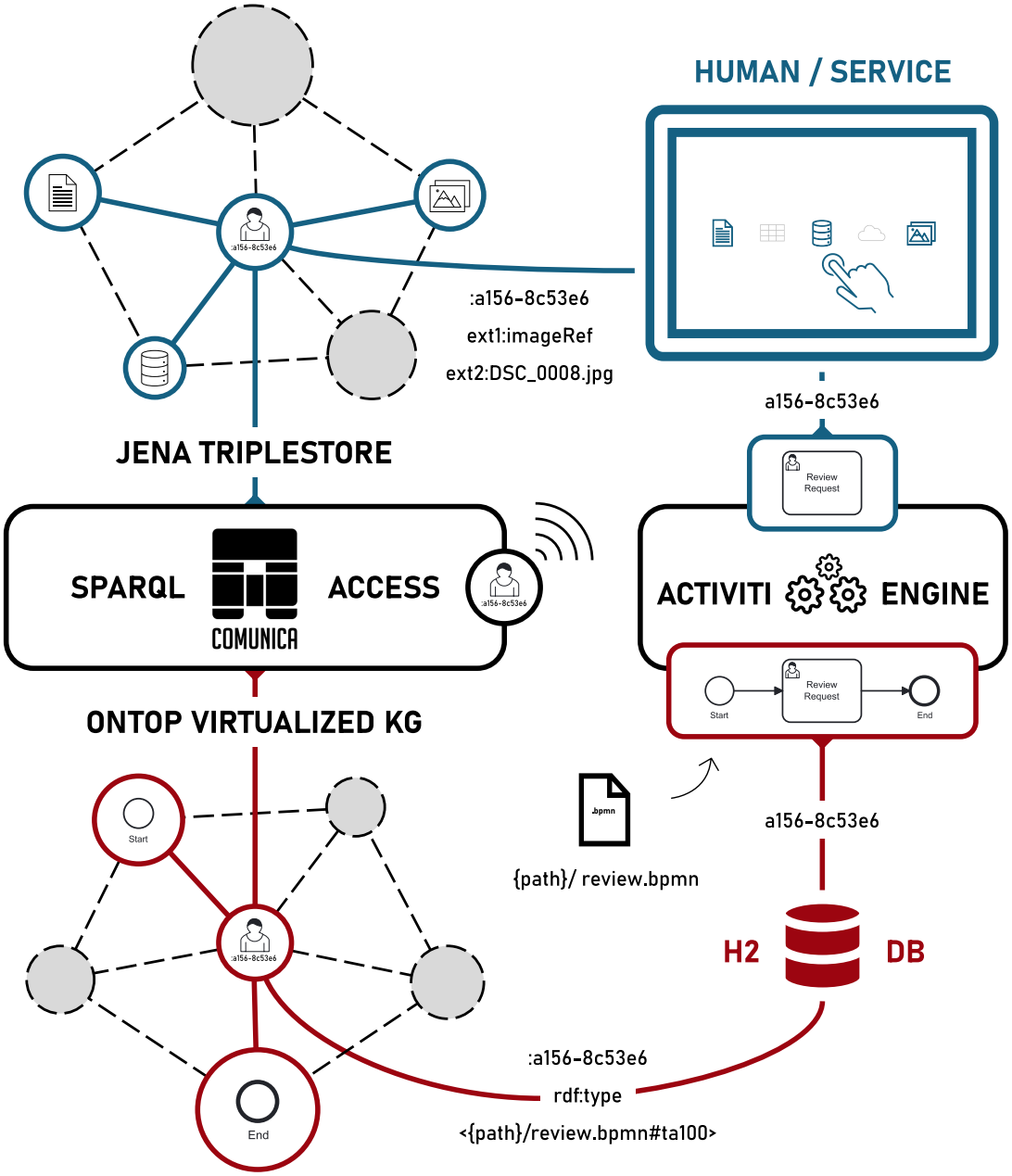


Figure 8.4: sBPMN system architecture integrating an Activiti-based BPMN engine (such as Camunda 7), a virtualized SPARQL endpoint (Ontop), and a materialized triplestore (Jena). Execution data is mapped to RDF and accessed via SPARQL, enabling federated queries (Comunica) across both runtime process knowledge and semantic domain knowledge.

8.2.2 Virtualization and Knowledge Graph Construction

The Camunda data schema extends the BPMN 2.0 schema and can thus be aligned with the sBPMN ontology once transformed into an ontological representation. In the following, we con-

sider an auxiliary Camunda 7 ontology that complements the sBPMN core ontology by adding classes and properties to capture runtime metadata, such as user assignments and temporal attributes, as shown in Listing 8.5. Once these extensions are defined, the H2 logging data can be exposed as a SPARQL-queryable RDF knowledge graph using the Ontop virtualized knowledge graph system [235]. Ontop enables the declarative mapping of relational database schemas to RDF ontologies following the Ontology-Based Data Access (OBDA) paradigm [43]. Given an ontology and a set of mappings, Ontop provides reasoning capabilities and allows relational data to be accessed and queried as an RDF graph via SPARQL, as illustrated in Listing 8.6.

```
camunda:assignee   rdf:type      owl:ObjectProperty ;
                   rdfs:domain   sbpmnc:userTask ;
                   rdfs:range    camunda:user .
camunda:startTime  rdf:type      owl:DatatypeProperty ;
                   rdfs:domain   sbpmnc:activity ;
                   rdfs:range    xsd:dateTime .
```

Listing 8.5: Excerpt of Camunda-based sBPMN extensions introducing runtime-related properties for user assignment and temporal attributes.

```
[MappingDeclaration] @collection [[
mappingId   activityTypeMapping
target      :{ACT_TYPE_}  owl:subClassOf  sbpmnc:activity .
            kg:{ID_}      rdf:type        onto:{ACT_TYPE_} ;
                          camunda:assignee  kg:user_{ASSIGNEE_} ;
                          camunda:startTime "{START_TIME_}" ;
                          camunda:endTime   "{END_TIME_}" .
source      SELECT ID_, ASSIGNEE_, ACT_TYPE_,
                   START_TIME_, END_TIME_
            FROM ACT_HI_ACTINST
]]
```

Listing 8.6: Ontop OBDA mapping exposing Camunda execution log data as RDF triples by linking relational attributes to ontology classes and properties.

These mappings are materialized as RDF triples during SPARQL execution, enabling real-time querying over the execution log database. Depending on the orchestration engine and its metadata conventions, OBDA configurations may vary. However, since Activiti-based engines typically extend the Activiti logging schema, identical mappings can usually be applied, differing only in identifiers. The virtualized RDF approach captures the semantics of BPMN process executions as observed within the orchestration engine. To integrate this runtime perspective with complementary data sources and domain-level context conceptualized in the external layer of the modular ontology, we adopt a dual-endpoint architecture.

8.2.3 Dual-Endpoint Architecture

In addition to the virtualized RDF endpoint, we deploy a materialized triplestore using Apache Jena TDB [213] to store the modular ontology and maintain a background knowledge base that enriches process models and their executions with external knowledge. This includes domain ontologies and contextual data managed not by Camunda itself but by the external layer of the modular ontology setup. Execution metadata, such as task or user IDs, serve as stable identifiers for linking both graphs, forming a semantically enhanced digital shadow of runtime process instances. For example, a human actor performing a user task is identified by its internal Camunda user ID, which is reused in the background knowledge graph to attach additional information about the individual, for instance via foaf:name or schema:email.
Similarly, external systems such as IoT-enabled machines can contribute associated metadata. Semantic enrichment is facilitated through an agent-based design, in which agents, either human users or automated services, contribute context-specific knowledge that is not explicitly encoded in the process ontology or in the H2 logs of Camunda. For example, a material class used in a production step may be defined in an external materials ontology and linked to an sBPMN task instance via shared identifiers, semantically aligned through the integration or model layer. This information can then be instantiated in the background graph and becomes accessible alongside the execution data from the virtualized endpoint.
Agents interact with the process at a local level without requiring visibility into the full orchestration model. This approach is also consistent with the Zeebe architecture of Camunda 8. The orchestration engine exposes only the currently assigned task, referenced by its internal identifier. This localized view simplifies access control and reduces cognitive and technical overhead while still enabling rich semantic interactions by means of the shared domain knowledge. Through this mechanism, agents can attach or query contextual information relevant to their current task, such as media annotations, sensor data, or material properties, thereby supporting both interoperability and knowledge-driven process execution as key enablers of Industry 5.0.

8.2.4 SPARQL Query Federation

To provide unified access across the virtualized execution endpoint and the background triplestore, we employ a federation layer based on the Comunica query engine [210]. Comunica supports SPARQL 1.1 federation across multiple endpoints, allowing both graphs to be queried in parallel. This configuration facilitates hybrid reasoning and cross-graph inference over live process executions and domain knowledge. Queries may combine BPMN runtime data, such as start times or user assignments, with external information, such as material specifications or user roles, thereby supporting semantically enriched process analysis. In this way, the dual-endpoint architecture forms the foundation for capturing and reasoning over dynamic process executions, as demonstrated in the real-world industrial use case discussed in the following section.

8.3 Use Case: Human-AI Teaming in Industry

The sBPMN framework represents one of the key outcomes of the Teaming.AI project, which developed a human-centric platform for human–AI collaboration in industrial environments. In

this chapter, we exemplify the capabilities of sBPMN through the human-on-the-loop use case implemented at Industrias Alegre, a manufacturer specialized in plastic components for the automotive sector. It supplies major OEMs and Tier-1 manufacturers and commits to innovation in quality assurance and production traceability. The company relies on widely adopted standards such as MES and ERP systems to manage orders, tasks, and resources, while OPC UA standardizes access to machine-level data (cf. Chapter 1). For instance, a customer order is initiated in the ERP, scheduled through the MES, and executed on the shop floor via OPC UA-connected machines. Although these systems enable structured data exchange, they are primarily designed for task-level orchestration and do not capture the dynamics of human–AI collaboration [153], such as operator decisions informed by ML predictions. Moreover, without a shared semantic model, these components remain loosely coupled and difficult to reason over [10]. To address this limitation, teaming scenarios are modeled using BPMN and semantically enriched through sBPMN, enabling integrated and adaptive process representations.

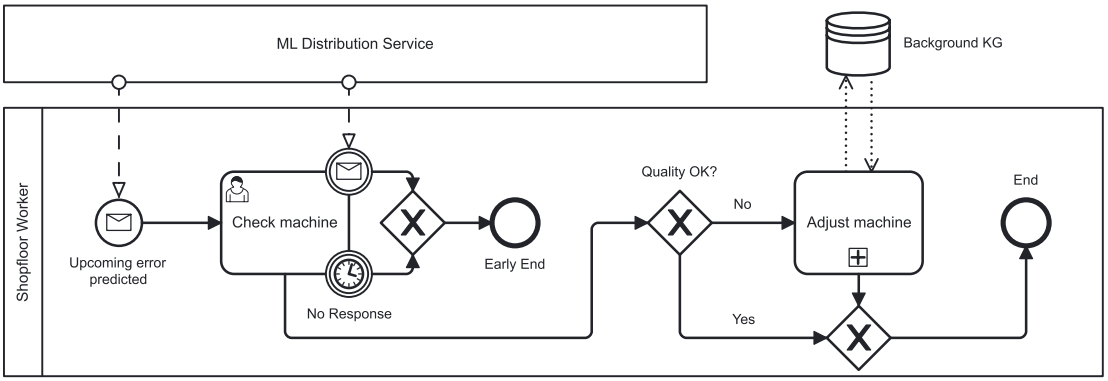


Figure 8.5: BPMN diagram of a human-AI teaming scenario in the plastic moulding industry. When an upcoming machine error is predicted, operators are notified and requested to check the machine. The process continues either when the check is completed or a catch event occurs (e.g., timeout or a new prediction supersedes the current one). If the check is performed, a quality assessment determines whether to trigger an ad-hoc machine adjustment subprocess supported by the semantic domain knowledge.

The Teaming.AI demonstrator at Industrias Alegre focuses on machine and process diagnostics as well as human interaction in thermoplastic injection moulding. The workflow spans pre-processing (e.g., mounting and task selection), the injection phase (e.g., mould temperature, injection pressure, and cycle time), and post-processing (e.g., cooling and quality control). The goal is to detect and prevent potential quality deviations by combining sensor-based ML predictions with semantically enriched domain knowledge encoded in knowledge graphs, as illustrated in Figure 8.5. The resulting execution traces are captured through RDF mappings of Camunda's H2 logging database and subsequently enriched with external knowledge sources, as described in the following sections. Beyond demonstrating human-centered collaboration between operators and AI systems, this use case also addresses the broader goals of Industry 5.0, namely sustainability and resilience. By representing process executions as semantic digital

shadows through sBPMN, the resulting knowledge graph trajectories enable improved analysis of production processes. In particular, they support the identification of patterns related to waste generation, contributing to more sustainable manufacturing, and facilitate the forecasting of process deviations, thereby strengthening the resilience of industrial production systems.

8.3.1 External Knowledge Integration

To ensure standardized interpretation of machine signals and process states, the Euromap 77 specification [68] is adopted as domain terminology within the ontology's external layer. This standard defines OPC UA-based interfaces for injection moulding machines and provides a shared vocabulary for key machine data points. For example, the class e77:InjectionUnit contextualizes physical components such as e77:BarrelId and e77:ScrewDiameter. Similarly, the group e77:InjectionCycleParameters captures operational metrics including e77:InjectionTime and e77:SpecificPressureMaximum, while the class e77:Jobs describes task-level execution metadata such as job status and associated machine units.
These concepts are instantiated in the Jena knowledge graph as contextual background knowledge supporting ML-based error prediction. The underlying ML models are trained on sensor data to forecast potential machine errors in upcoming production cycles. When a prediction exceeds a predefined threshold, an alert is generated and forwarded to the operator. In addition to estimating the likelihood of an error, the system also classifies its expected type, such as burn marks or short shots, by leveraging domain knowledge encoded in the external ontology and linking it to BPMN events. Each error type is associated with recommended countermeasures, such as adjusting injection speed or mould temperature. When an alert occurs, the machine state is referenced and contextual information is stored in the background knowledge graph.
Because all relevant entities, including machine states, process events, error types, and recommended actions, are semantically represented, they can be queried using SPARQL to support runtime reasoning and context-aware decision-making. Listing 8.7 illustrates a query retrieving all process instances involving the use-case-specific user task model:checkMachineTask that is linked to a e77:shortShotError.

```
SELECT ?processInstance WHERE {
    ?error rdf:type e77:shortShotError .
    ?task rdf:type model:checkMachineTask ;
          onto:errorObserved ?error .
    ?processInstance sbpmnp:contains ?task . }
```

Listing 8.7: SPARQL query retrieving process instances associated with a short-shot error.

The countermeasures associated with these error types are encoded as BPMN tasks and organized into ad-hoc BPMN subprocesses, that is, modular process fragments triggered dynamically at runtime [21]. These subprocesses may include parameter adjustments, inspection steps, or calls to maintenance personnel. Their execution follows the system architecture introduced in Section 8.2, where Camunda serves as the BPMN orchestration engine and dynamically logs process executions.

8.4 Discussion

The sBPMN framework demonstrates the feasibility of encoding BPMN-based process knowledge within RDF knowledge bases. Its development constituted the final outcome of the Teaming.AI project, providing a unified approach to fulfill the Industry 5.0 knowledge modeling requirements R1–R4 derived in Chapter 5. Preliminary implementations addressing R5 were explored using GNN models operating on static domain knowledge sources. However, a full integration of graph learning on dynamic semantic digital shadows was beyond the scope of the final project demonstrator. While motivated by industrial use cases, the framework is not restricted to manufacturing settings but is applicable to any process-driven domain with inherent semantic domain knowledge. Process models are encoded as ontology components, and execution logs generated by BPMN engines are transformed into RDF triples populating the ABox, i.e., the knowledge graph. As process executions evolve, successive assertional states form dynamic knowledge graph trajectories that constitute semantic digital shadows of workflows. Within Teaming.AI, semantic mappings enabled the reuse of recurring process patterns across use case partners. At Industrias Alegre, operators reported increased transparency of ML-based predictions through access to past process instances and their associated metadata in the knowledge graph, enabling traceability and explainability by linking predictions to domain context, operator actions, and process execution.

From a knowledge modeling perspective, sBPMN establishes a principled mechanism for integrating process knowledge with heterogeneous domain ontologies. By representing process models as ontology components, BPMN processes become a subdomain of the knowledge base and act as horizontal connectors between otherwise loosely coupled ontologies. At the same time, concrete executions in the ABox can be linked to instance-level entities such as machines, operators, or production orders, enabling a unified representation of workflows and contextual knowledge. The current implementation should be understood as a preliminary realization that requires further evaluation regarding scalability and automation. In particular, the transformation of BPMN models into ontology representations and the integration of execution logs remain partially manual. Future work should therefore focus on automated mapping pipelines and scalable reasoning approaches, for example by extending tools such as BPMN2KG [20] to derive sBPMN-specific mappings with reduced manual effort.

Despite these limitations, the framework provides a concrete methodology for constructing semantic digital shadows of process-driven systems. Building on this foundation, the following chapter investigates how such semantic digital shadows can be transformed into semantic digital twins through dual feedback mechanisms and adaptive graph learning. In particular, we derive requirements for feedback loops in Industry 5.0 settings, while formulating the findings in a general context applicable to arbitrary graph learning approaches beyond industrial domains.

PART IV

Enabling Semantic Digital Twins via Graph Learning

Wisdom is the right use of knowledge.

Charles Spurgeon

This part establishes graph learning as a key enabler of semantic digital twins in Industry 5.0. Building on insights obtained from the Teaming.AI project, we abstract from a concrete application scenario to derive general principles for learning on graph-structured data, while highlighting their relevance for industrial settings. While feedback loops can be realized through logical inference over domain ontologies, data-driven approaches based on graph embeddings are increasingly used to exploit complex relational data. However, such models typically lack transparency, thereby undermining a central advantage of knowledge graphs. We first show how predictions of relational graph neural networks can be explained using layer-wise relevance propagation, restoring interpretability at the model level. Building on these insights, we identify key limitations of existing graph embedding methods in real-world settings, in particular with respect to structural consistency and dynamic adaptability. To address these challenges, we introduce local self-reconstruction as a novel graph embedding paradigm that enables robust, adaptive, and semantically aligned representations for evolving graph data.

Graph Learning as a Feedback Mechanism

The semantic digital shadows introduced in the previous chapters provide a structured representation of industrial processes as dynamic trajectories of RDF knowledge graphs. While these representations enable the integration and semantic interpretation of heterogeneous data sources, they do not yet constitute fully operational semantic digital twins as required by R5 (cf. Chapter 5). A defining property of digital twins is the existence of feedback mechanisms that allow the represented system to adapt its behavior based on the knowledge captured in its digital representation. Consequently, transforming a semantic digital shadow into a semantic digital twin requires mechanisms that derive actionable insights from the knowledge graph and feed them back into the operational system. Within the Teaming.AI project, initial efforts explored the use of GNNs as such feedback mechanisms, for instance to predict suitable ML model instances for given collaborative scenarios. However, these approaches remained out of scope for the final demonstrator, which primarily consisted of the semantic digital shadow provided by sBPMN (cf. Chapter 8). Instead, feedback was realized through symbolic mechanisms, in particular logical inference and constraint validation, which can be applied as soon as corresponding axioms are encoded in the domain model, i.e., the corresponding domain ontologies.

One class of feedback loops is therefore provided by symbolic reasoning mechanisms inherent to RDF-based technologies (cf. Section 2.3). Logical entailment using formalisms such as RDFS and OWL allows implicit knowledge to be derived from explicitly asserted triples, thereby enriching the knowledge graph with additional facts. In addition, constraint validation languages such as SHACL enable consistency checking of instance-level data against formally specified structural constraints. Within semantic digital shadows, these mechanisms support the verification of process executions, the identification of constraint violations, and the inference of implicit relationships between entities. In this sense, axiomatic inference and constraint-based validation already establish a fundamental form of feedback loop by enabling the system to detect inconsistencies and derive new knowledge that can influence subsequent decisions.

However, symbolic reasoning alone does not exploit statistical regularities that emerge from large collections of observed process executions and their associated domain context. Beyond data integration and semantic representation, RDF-based knowledge graphs therefore increasingly serve as input to graph learning algorithms [229] (cf. Chapter 4). These approaches leverage the structural properties of graphs to perform downstream tasks such as node classification, link prediction, or recommendation. In contrast to purely symbolic reasoning, graph

learning methods can capture latent structural patterns that are not explicitly encoded in ontological axioms, thereby providing an additional mechanism for establishing feedback loops based on learned predictive models. To enable such ML-based methods, graph elements are typically transformed into numerical vector representations known as graph embeddings. These embeddings encode structural and semantic properties of nodes and relations in a continuous vector space, enabling the use of standard ML techniques. Within semantic digital shadows, such embeddings can support predictive analyses of process behavior, the recommendation of corrective actions, and the identification of recurring structural patterns in industrial workflows.

Despite this potential, the application of graph learning to dynamic knowledge graph trajectories in Industry 5.0 introduces two fundamental challenges. First, although graph learning models operate on inherently interpretable graph structures, their predictions are typically produced by opaque numerical representations. Once a graph is embedded into a latent vector space, the connection between predictions and the underlying knowledge graph becomes difficult to interpret, limiting their suitability for transparent decision support. Second, most existing graph embedding methods are designed for static graphs. In typical settings, embeddings are learned for a fixed snapshot, reflecting the traditional view of graphs as static objects. In contrast, semantic digital shadows evolve continuously as new process executions and observations are incorporated. As a result, embeddings quickly become outdated, and adapting them requires costly retraining [27]. Enabling graph learning as a practical feedback mechanism therefore requires methods that can efficiently adapt embeddings to dynamic graph updates. In the following, we consider GNNs as a key enabler of such data-driven feedback mechanisms. Although not part of the official Teaming.AI demonstrator, we study their role in a generalized setting with established benchmarks and address the two central limitations identified above: the explainability of graph-based predictions with respect to the underlying knowledge graph structure (Chapter 9) as well as the adaptability and robustness of embeddings in dynamic environments such as process-driven semantic digital shadows (Chapter 10).

Taken together, these enhanced feedback mechanisms enable the transition to bidirectional semantic digital representations. By augmenting symbolic reasoning with data-driven graph learning, knowledge graphs evolve from passive representations into active components of adaptive systems. This is particularly relevant in the context of Industry 5.0, where transparency, adaptability, and human-centric decision support are key objectives. At the same time, these principles are not limited to industrial applications, but extend to domains in which dynamic knowledge graph trajectories serve as representations of complex systems.

Chapter 9

Explaining RGNN Predictions on RDF Graphs via LRP

This chapter builds upon the publication *"Explaining RGNN Predictions on Knowledge Graphs via LRP"* by Franz Krause and Heiko Paulheim [120], published in the *Proceedings of the 22nd International Conference on Semantic Systems*, IOS Press (2026).

Knowledge graphs are central to modern data management and reasoning, providing semantically rich representations of domain entities and their relationships. Typically represented as RDF graphs [50] and supported by ontologies, they combine machine-readable structure with human-interpretable semantics. This enables integration, interoperability, and reasoning across heterogeneous data sources [15, 99]. At the same time, knowledge graphs serve as graph-structured inputs for ML tasks such as node classification and link prediction [160].

As described in Section 4.3, RGNNs represent a powerful tool for learning on relational graph structures such as RDF knowledge graphs. In particular, RGNNs exploit local graph topology and relation types to capture semantic dependencies via multi-hop message passing, achieving state-of-the-art performance on benchmark tasks [41, 197, 246]. However, this integration introduces a fundamental tension: while knowledge graphs are designed to support transparent and interpretable reasoning, RGNN models act as black-box predictors whose internal decision processes are difficult to interpret. Consequently, predictions derived from semantically structured knowledge graphs may lose the very transparency they were intended to provide.

This challenge is not merely conceptual but highly relevant in real-world deployments. Knowledge graph-based systems are increasingly used in industrial decision-support scenarios, where predictions can have operational or societal impact [40, 112]. In the previous chapters, we established how such systems can be grounded in a semantic digital shadow for Industry 5.0, providing a unified, semantically enriched representation of industrial processes and their execution data. Building upon this foundation, RGNNs enable predictive capabilities on top of these structured representations. However, ensuring that such predictions remain interpretable is essential for enabling human oversight, fostering trust, and supporting regulatory requirements such as those outlined in emerging AI governance frameworks [33, 69].

Addressing this challenge requires explaining predictions directly in terms of the underlying knowledge graph structure. Ideally, predictions produced by relational graph learning models should be traceable to the entities and relations that contributed to them. In this work, we formalize this idea as a *semantic predictive loop*: knowledge graph structure serves as input to relational graph learning models, and the resulting predictions are explained again with respect to the same graph. Given the semantic digital shadow introduced in Chapter 8 and RGNN-based predictions built upon it, this loop provides feedback on automated predictions and enables transparent, structure-aware interpretation of model behavior.

To realize this loop, we propose a generalized framework for explaining RGNN predictions using Layer-Wise Relevance Propagation (LRP). The approach traces predictions through the relational message passing process and redistributes signed relevance to nodes, edges, and multi-hop paths. As illustrated in Figure 9.1, an RGNN first produces a prediction over the knowledge graph, which is then traced back via LRP to assign signed relevance scores to graph components (e.g., nodes, edges, and paths), thereby effectively closing the semantic predictive loop. Beyond extending relevance propagation to relational message passing, the framework unifies structure-oriented and embedding-oriented explanations by linking graph components with embedding-level contributions. Experiments on standard knowledge graph learning benchmarks demonstrate that the proposed approach yields faithful and interpretable explanations of RGNN predictions across multiple structural levels.

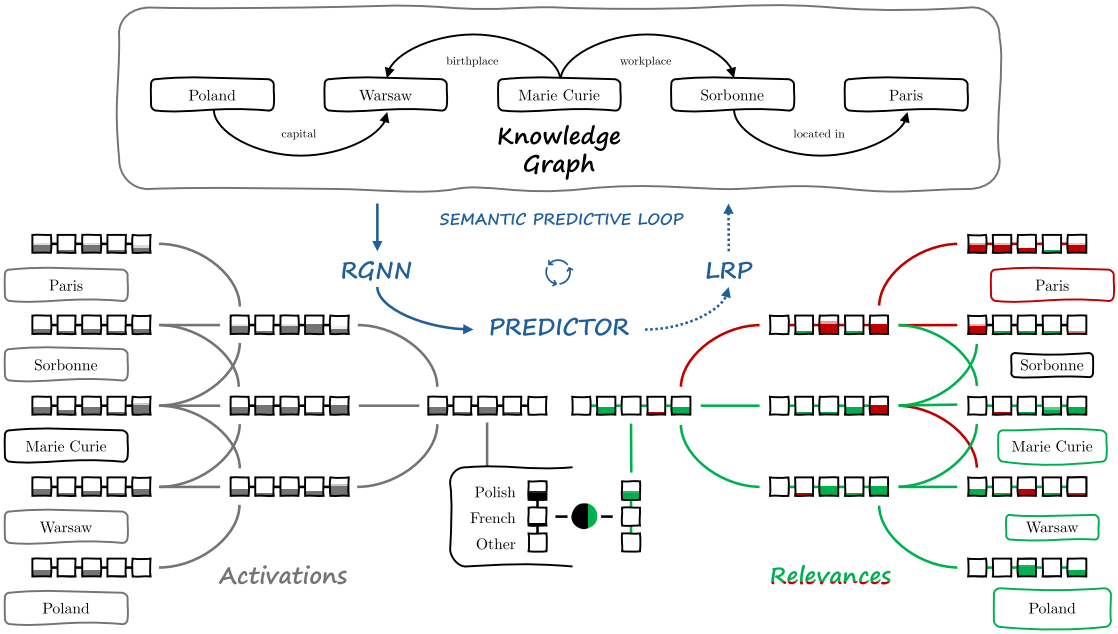


Figure 9.1: Illustration of the semantic predictive loop: an RGNN predicts the nationality of Marie Curie from a relational knowledge graph, while LRP traces the prediction back through relational message passing to assign signed relevances to nodes, edges, and embedding dimensions (green: supportive, red: opposing).

9.1 Layer-Wise Relevance Propagation

In this section, we first introduce preliminaries on LRP and its applicability to NN-based predictors, building on the notation established in Section 4.3, particularly Definition 11. We then review existing approaches for explaining GNN- and RGNN-based predictions, focusing on their characteristics and practical limitations in real-world settings. Throughout this chapter, we consider directed relational graphs, i.e.,

$$\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{R}$$

with relation types $\mathcal{R}$. In the context of RDF graphs $\mathcal{F}$ as per Chapter 3, these relations correspond to the induced predicates, i.e., we consider $\mathcal{R} = \mathcal{P}(\mathcal{F}_{\mathfrak{R}})$ and $\mathcal{V} = \mathcal{A}(\mathcal{F}_{\mathfrak{R}})$.

Derived from deep Taylor decomposition, LRP explains NN forward passes by conserving output relevance and redistributing it to input features [18, 151]. Signed input relevances indicate which features increase or decrease the output.

Definition 13. *Let $f : \mathbb{R}^{d'} \to \mathbb{R}^d$ be a dense layer as per Definition 11 with component-wise monotonic activation and output $y = f(x)$. Further, let $\hat{y} \in \mathbb{R}^d$ be a relevance vector, where $\hat{y}_j$ quantifies the relevance of the output neuron y_j. LRP computes relevance vectors $\hat{x} \in \mathbb{R}^{d'}, \hat{b} \in \mathbb{R}^d$ for the dense layer's input and bias, respectively:*

$$\hat{x}_i = \sum_{j=1}^{d} \frac{W_{j,i}\, x_i}{b_j + \sum_{k=1}^{d'} W_{j,k}\, x_k}\, \hat{y}_j c, \qquad \hat{b}_j = \frac{b_j}{b_j + \sum_{k=1}^{d'} W_{j,k}\, x_k}\, \hat{y}_j.$$

This redistribution preserves total relevance: $\sum_{i=1}^{d'} \hat{x}_i + \sum_{j=1}^{d} \hat{b}_j = \sum_{j=1}^{d} \hat{y}_j$.

Accordingly, relevance is redistributed in proportion to each input neuron's contribution to each output neuron. The activation $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ is called component-wise monotonic if the following holds for any $\tau, \tau' \in \mathbb{R}^d$:

$$\tau_j \geq \tau'_j \text{ and } \tau'_k = \tau_k \text{ for all } k \neq j \quad \Longrightarrow \quad \sigma(\tau)_j \geq \sigma(\tau')_j.$$

This property ensures that relevance can be propagated at the pre-activation level $W \cdot x + b$, so σ can be treated as the identity during relevance propagation, as illustrated in Figure 9.2. Standard activation functions such as ReLU, sigmoid, and softmax satisfy this property. In this work, we use the original LRP formulation. Stabilized variants exist to avoid near-zero denominators [151]. Although we did not observe numerical issues, some applications may require such extensions, which typically break exact relevance conservation.
To apply LRP, an output index $j^* \in \{1, \ldots, d\}$ is selected, such as the one corresponding to a predicted class. The initial relevance is set to $\hat{y}_{j^*} = y_{j^*}$ and $\hat{y}_j = 0$ for all $j \neq j^*$. Relevance is then propagated backward according to Definition 13, yielding explanations by identifying how input or hidden units contribute to the output of interest. The decomposition into $\hat{x}$ and $\hat{b}$ also separates input-driven evidence from contributions encoded in the bias.

Moreover, LRP extends naturally to composite NNs through layer-wise propagation. For example, given an MLP $F(x) = y \in \mathbb{R}^d$ and initial relevance $\hat{y}$, applying LRP to the final layer f_L yields $\hat{y}_{L-1} = \hat{x}_L \in \mathbb{R}^{d'_L}$ and $\hat{b}_L \in \mathbb{R}^{d_L}$. Recursively applying the rule to layers f_ℓ for $\ell = 1, \ldots, L$ produces relevance vectors $\hat{y}_{\ell-1} = \hat{x}_\ell \in \mathbb{R}^{d'_\ell}$ and $\hat{b}_\ell \in \mathbb{R}^{d_\ell}$.

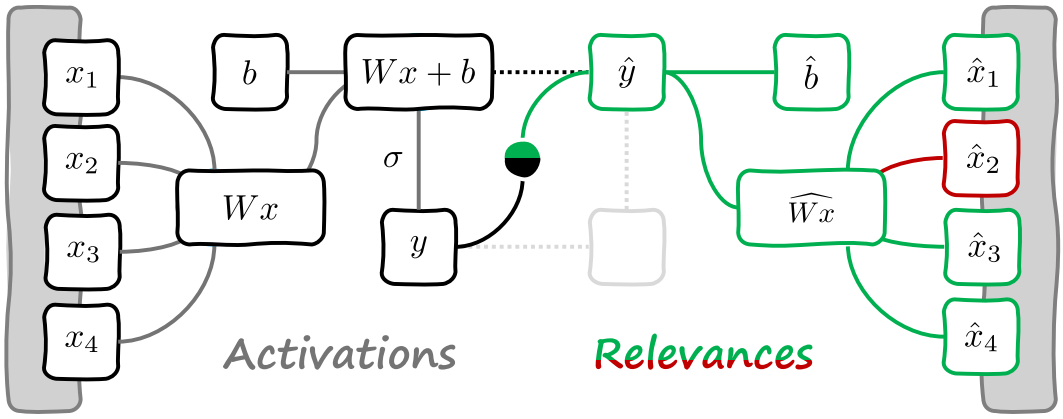


Figure 9.2: Forward pass and relevance propagation for an output $y = \sigma(Wx+b) \in \mathbb{R}$ in a dense layer. LRP redistributes the output relevance $\hat{y} \in \mathbb{R}$ back to the pre-activation contributions, yielding signed relevance scores in $\hat{x} \in \mathbb{R}^4$ and $\hat{b} \in \mathbb{R}$ (green: supportive, red: opposing).

9.2 Related Work

Explanations of individual GNN predictions target either the embedding space or the underlying graph structure [137]. Model-agnostic methods such as LIME [185] and SHAP [140] can be applied to GNNs by perturbing node features or learned embeddings and observing prediction changes. However, these approaches scale poorly, ignore message passing, and are not designed for structural or relational reasoning. Embedding-oriented methods instead assign semantics to latent dimensions, for example via correlations with structural statistics [53], dimension-level interpretability [172, 199], or disentangled representations [173]. While improving transparency in latent space, they do not explain how predictions arise from graph structure. Structure-oriented explainers address this by analyzing the computation graph used by the model for a prediction and are grouped into three families.

A: Edge-elimination methods start from the full computation graph and identify explanations by masking edges whose removal minimally changes the model output. GNNExplainer [247] learns such masks via per-instance optimization, which must be repeated for every prediction and downstream task. PGExplainer [141] replaces this with a global surrogate network trained to predict edge relevance from node embeddings, but a separate surrogate must be trained for each task (e.g., each node classification task or relation type in link prediction). Given that both methods operate on homogeneous adjacency matrices, adapting them to RGNNs requires extensions to handle typed message passing.

B: Path-synthesis methods generate explanations by assembling multi-hop relational paths rather than pruning the full graph. RAW-Explainer [127] performs guided random walks whose transition probabilities depend on the model-internal message coefficients $c_{r,\iota}(u,v)$ per stub $s = (u, r, \iota)$ according to Definition 12. RelEx [253] learns relation-type importance scores through continuous relation-level masks and then constructs relational paths consistent with these scores. Because relation scores in RelEx are computed for individual link prediction queries, the resulting explanations are not applicable to node classification settings.

C: Propagation-based methods trace contributions through the message-passing architecture and thereby yield structure-oriented explanations. FlowX [88] and GradXKG [249] rely on gradients of the downstream losses, which make the methods sensitive to activation saturation and unable to produce signed attributions. GNN-LRP [198] applies LRP to homogeneous GNNs and produces signed relevance scores reflecting positive or negative evidence. However, its formulation is restricted to untyped message passing and does not support relational aggregation functions. In this work, we extend GNN-LRP to RGNNs to derive signed relevances of knowledge graph components and embedding dimensions.

As summarized in Table 9.1, existing graph-specific explanation approaches only partially satisfy the requirements of RGNNs. In particular, they often lack support for relation types, global validity without per-instance optimization, signed relevance scores distinguishing positive and negative evidence, applicability across different downstream tasks, full coverage of the computation graph, and interpretability at the level of embedding dimensions. Among these approaches, PGExplainer serves as a baseline for edge- and node-level explanations. While it does not explicitly model negative relevance, it distinguishes between relevant and irrelevant edges and can be extended to incorporate relation types. For path-level explanations, we consider RAW-Explainer as a complementary baseline. Although its random walk generation may not capture the entire computation graph, it supports multiple downstream task settings and thus provides a useful reference for path-based explainability.

Method	Relational	Global	Signed	NC/LP	Full	Embedding
GNNExplainer (A)	×	×	positive-only	✓/×	✓	×
PGExplainer (A)	×	✓	positive-only	✓/✓	✓	×
RelEx (B)	✓	×	positive-only	×/✓	×	×
RAW-Explainer (B)	✓	✓	positive-only	✓/✓	×	×
FlowX (C)	×	✓	×	✓/×	✓	✓
GradXKG (C)	✓	✓	×	×/✓	✓	✓
GNN-LRP (C)	×	✓	✓	✓/✓	✓	✓

Table 9.1: Comparison of GNN explanation methods. Columns indicate support for relational message passing (Relational), global explanations without per-instance optimization (Global), signed relevance (Signed), applicability to NC/LP tasks (NC/LP), coverage of the full computation graph (Full), and attribution to embedding dimensions (Embedding).

9.3 Methodology

We extend GNN-LRP to RGNNs operating on relational graphs $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{R}$. Analogous to prior RGNN works, we treat them as equivalent to their closures, i.e., $\mathcal{G} = \overline{\mathcal{G}}$ (cf. Section 4.3), implying $v \in \mathcal{N}_{\mathcal{G}}(v)$. The aggregation steps within RGNN layers as per Definition 12 can be expressed as compositions of dense layers, thus allowing for the propagation of relevance to edge and node neighborhoods via intermediate embeddings. As shown in Figure 9.3, we consider an individual RGNN output $\phi(v) = \mathcal{K}(\psi, \mathcal{G})(v)$, based on an RGNN graph embedding of $\mathcal{X} \subseteq \mathcal{V}$:

$$\mathcal{K}_{\mathcal{G}} := \mathcal{K}(\cdot, \mathcal{G}) : (\mathbb{R}^{d'})^{\mathcal{N}_{\mathcal{G}}^{L}(\mathcal{X})} \to (\mathbb{R}^{d})^{\mathcal{X}}, \qquad \phi = \mathcal{K}_{\mathcal{G}}(\psi) = (\kappa_{\mathcal{G},L} \circ \cdots \circ \kappa_{\mathcal{G},1})(\psi).$$

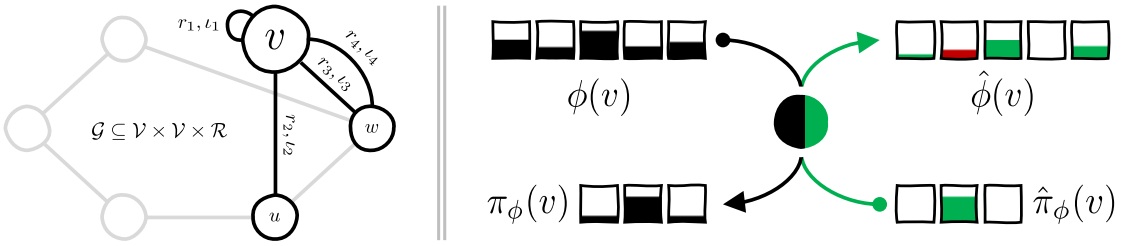


Figure 9.3: LRP initialization for RGNN pipelines: Initial relevances $\hat{\pi}_\phi(v)$ of downstream predictions $\pi_\phi(v)$ are redistributed to RGNN outputs via $\hat{\phi}(v)$ (green: supportive, red: opposing).

Given an initial relevance vector $\hat{\phi}(v)$, we propose two paradigms for distributing relevance: Macro LRP provides aggregated, layer-wise explanations at the node level, whereas micro LRP preserves the full path structure and enables fine-grained attribution to multi-hop relational dependencies. Intuitively, the goal is to trace how predictive relevance propagates through relational message passing back to the graph structure, thereby enabling a semantic predictive loop. To obtain the initial relevance, an LRP-compatible predictor $\pi : \mathbb{R}^{d} \to \mathbb{R}^{d_\pi}$ is assumed which uses $\phi(v)$ as input, i.e.,

$$\pi_\phi(v) := \pi(\phi(v)) \in \mathbb{R}^{d_\pi}.$$

In line with LRP objectives, we aim to explain predictions by addressing the question:

Why was the j^*-th output entry $\pi_\phi(v)_{j^*}$ of the prediction $\pi_\phi(v)$ high (or low)?

A typical example is a classifier, where $\pi_\phi(v)_{j^*}$ is the probability of class j^*. Initial relevance $\hat{\pi}_\phi(v)$ is defined as $\hat{\pi}_\phi(v)_{j^*} = \pi_\phi(v)_{j^*}$ and $\hat{\pi}_\phi(v)_j = 0$ for $j \neq j^*$. Since the predictor operates on $\phi(v)$, this relevance is transferred to $\hat{\phi}(v)$. Analogously, link prediction assigns a plausibility score to a stub $s \in \mathcal{V} \times \mathcal{R} \times \mathcal{I}$ expressing how likely it is that $s \in \mathcal{S}_{\mathcal{G}}(v)$ holds. Such scores are computed via scoring functions $S_\phi : \mathcal{V} \times (\mathcal{V} \times \mathcal{R} \times \mathcal{I}) \to \mathbb{R}$ with

$$S_\phi(v, s) := S\big(\phi(v), \phi(u), r, \iota\big), \qquad S : \mathbb{R}^{d} \times \mathbb{R}^{d} \times \mathcal{R} \times \mathcal{I} \to \mathbb{R},$$

where higher scores indicate greater plausibility of the hypothesis $s = (u, r, \iota) \in \mathcal{S}_{\mathcal{G}}(v)$. Accordingly, we consider the downstream prediction

$$\pi_\phi(v; s) := S_\phi(v, s) \in \mathbb{R}.$$

Since link prediction scores are scalar values, $\hat{\pi}_\phi(v;s) = \pi_\phi(v;s)$ serves as initial relevance, yielding $\hat{\phi}(v)$. Given an edge (v,u,r) with fixed u and r, we use $\pi_\phi(v;(u,r,\iota_{\text{out}}))$, whereas fixing v and r yields $\pi_\phi(u;(v,r,\iota_{\text{in}}))$. Thus, node classification performs predictions over predefined class labels, whereas link prediction assigns a plausibility score to a stub indicating how likely it is to be incident to a node. In this context, each relation type implies a distinct downstream task. In the following, we first apply LRP to single RGNN layers and subsequently extend the procedure to multi-layer architectures. Throughout, we assume that no relevance is absorbed by biases, i.e., $\sum_{i=1}^{d} \hat{\phi}(v)_i = \pi_\phi(v)_{j^*}$, which is obtained by using RGNN models without biases.

9.3.1 Applying LRP to Single-Layer RGNNs

Let $\kappa : (\mathbb{R}^{d'})^{\mathcal{N}_\mathcal{G}(\mathcal{X})} \to (\mathbb{R}^d)^{\mathcal{X}}$ be an RGNN layer with input $\psi \in (\mathbb{R}^{d'})^{\mathcal{N}_\mathcal{G}(\mathcal{X})}$, yielding

$$\phi(v) = \kappa_\mathcal{G}(\psi)(v) \in \mathbb{R}^d.$$

Furthermore, we assume an initial relevance vector $\hat{\phi}(v) \in \mathbb{R}^d$. Figure 9.4 illustrates how $\hat{\phi}(v)$ can be redistributed to stubs $s = (u,r,\iota) \in \mathcal{S}_\mathcal{G}(v)$ and the neighborhood $\mathcal{N}_\mathcal{G}(v)$. For readability, we write $F_\psi(s) := F_{r,\iota}(\psi(u))$ and $c(s,v) := c_{r,\iota}(u,v)$ for $s = (u,r,\iota) \in \mathcal{S}_\mathcal{G}(v)$ in the following.

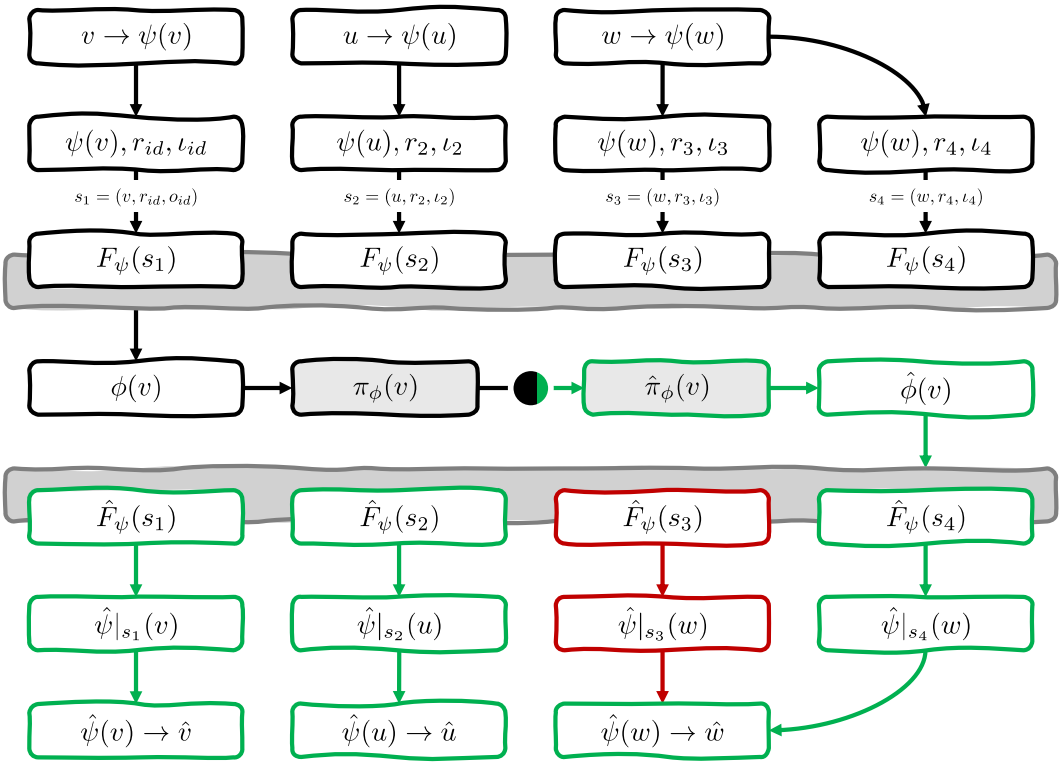


Figure 9.4: Decomposition of a single-layer RGNN output relevance vector $\hat{\phi}(v)$. Relevance is assigned to stubs $s \in \mathcal{S}_\mathcal{G}(v)$ and aggregated into node-level vectors and scalar relevance scores (green: supportive, red: opposing).

Component-wise monotonic activations are assumed. Hence, the weighted intermediate stub embeddings $\{F_\psi(s) \mid s \in \mathcal{S}_\mathcal{G}(v)\}$ yield stub-level relevances:

$$\hat{F}_\psi(s) = \big(c(s,v) \cdot F_\psi(s)\big) \oslash \Big(\sum\nolimits_{s' \in \mathcal{S}_\mathcal{G}(v)} c(s',v) \cdot F_\psi(s')\Big) \odot \hat{\phi}(v).$$

In this context, vector operations are elementwise, denoted by the Hadamard product/division $\odot/\oslash$. Since intermediate embeddings in standard RGNNs are produced by unbiased and unactivated dense layers [41, 114, 197, 224], i.e., $F_\psi(s) = W_{r,\iota} \cdot \psi(u)$ with $s = (u,r,\iota)$, the relevance $\hat{\psi}\,|_s\,(u)$ corresponds to the relevance assigned to $\psi(u)$ via s. These stub relevances can be aggregated further to obtain node embedding relevances

$$\hat{\psi}(u) = \sum\nolimits_{s=(u',r,\iota) \in \mathcal{S}_\mathcal{G}(v)} \big(\mathbb{1}_{u=u'} \cdot \hat{\psi}\,|_s\,(u')\big) \in \mathbb{R}^{d'} \quad \text{for all} \quad u \in \mathcal{N}_\mathcal{G}(v).$$

Scalar relevances $\hat{u} \in \mathbb{R}$ are obtained by component-wise summation of $\hat{\psi}(u)$.

9.3.2 Applying LRP to Multi-Layer RGNNs

Let $\mathcal{K}_\mathcal{G} = \kappa_{\mathcal{G},L} \circ \cdots \circ \kappa_{\mathcal{G},1}$ be a multi-layer RGNN with $L > 1$ layers. Section 9.3.1 describes how relevance vectors $\hat{\psi}_L\,|_s\,(u) \in \mathbb{R}^{d'_L}$ are computed for each stub $s = (u,r,\iota) \in \mathcal{S}_\mathcal{G}(v)$. Analogous to single-layer RGNNs, these can be aggregated to obtain node-level relevances $\hat{\psi}_L(u) \in \mathbb{R}^{d'_L}$ and scalars $\hat{u}_L \in \mathbb{R}$ for all $u \in \mathcal{N}_\mathcal{G}(v)$. Thus, one must decide whether (i) to perform relevance aggregation at each layer (macro LRP) or (ii) to assign relevance to paths as sequences of stubs without intermediate node-level aggregation (micro LRP).

Macro LRP for Multi-Layer RGNNs

Given relevance vectors $\hat{\psi}_L(u) \in \mathbb{R}^{d'_L}$ for $u \in \mathcal{N}_\mathcal{G}(v)$, macro LRP produces relevances $\hat{\psi}_{L-1}(u)$ for $u \in \mathcal{N}^2_\mathcal{G}(v)$ via κ_{L-1}. Iterative application yields relevance sets

$$\{\hat{\psi}_\ell(u) \mid u \in \mathcal{N}^{n_\ell}_\mathcal{G}(v)\}.$$

Unlike the output layer κ_L, layers with $\ell < L$ may require propagating multiple relevance vectors. We therefore define the relevance assigned to $\psi_\ell(u)$ from $\hat{\phi}_\ell(w)$ as

$$\hat{\psi}_\ell(u;w) \quad \text{for} \quad (u,w) \in \mathcal{N}^{n_\ell}_\mathcal{G}(v) \times \mathcal{N}^{n_\ell-1}_\mathcal{G}(v) \quad \text{with} \quad w \notin \mathcal{N}_\mathcal{G}(u) \Rightarrow \hat{\psi}_\ell(u;w) = 0.$$

Accordingly, the total relevance $\hat{\psi}_\ell(u)$ is obtained by node-wise summation:

$$\hat{\psi}_\ell(u) = \sum\nolimits_{w \in \mathcal{N}^{n_\ell-1}_\mathcal{G}(v)} \hat{\psi}_\ell(u;w).$$

Since $\mathcal{N}^0_\mathcal{G}(v) = \{v\}$ holds, this notation also applies to the RGNN output layer, where $\hat{\phi}_L(v)$ is the only relevance vector, implying $\hat{\psi}_L(u) = \hat{\psi}_L(u;v)$.

Micro LRP for Multi-Layer RGNNs

Instead of aggregating stub relevances at each layer, micro LRP groups stubs into paths

$$(s_1,\dots,s_n) \in P_n(v) \subseteq (\mathcal{V}\times\mathcal{R}\times\mathcal{I})^n$$

with $s_i = (u_i, r_i, \iota_i) \in \mathcal{S}_\mathcal{G}(u_{i-1})$ and $u_0 := v$. Each stub $s_1 \in \mathcal{S}_\mathcal{G}(v)$ defines a single-step path $s_1 \in P_1(v)$, which serves as the basis for all two-step paths $(s_1,s_2) \in P_2(v)$ with $s_2 \in \mathcal{S}_\mathcal{G}(u_1)$. Iteratively, all paths of length n from v are obtained via

$$P_n(v) = \{(s_1,\dots,s_n) \mid s_1 \in \mathcal{S}_\mathcal{G}(v),\dots,s_n \in \mathcal{S}_\mathcal{G}(u_{n-1})\}\,.$$

As a result, the total number of paths considered is

$$|P_n(v)| = \sum\nolimits_{s_1\in\mathcal{S}_\mathcal{G}(v)} \sum\nolimits_{s_2\in\mathcal{S}_\mathcal{G}(u_1)} \cdots \sum\nolimits_{s_n\in\mathcal{S}_\mathcal{G}(u_{n-1})} 1\,.$$

In the computation subgraph $\mathcal{H}^n_\mathcal{G}(v)$, a path $p = (s_1,\dots,s_{n_\ell}) \in P_{n_\ell}(v)$ corresponds to the reverse-ordered chain of intermediate embeddings

$$\psi_\ell(u_{n_\ell}),\dots,\psi_L(u_1).$$

Using LRP, a relevance vector $\hat{p} \in \mathbb{R}^{d_\ell}$ can be propagated to all one-step extensions

$$p' \in \{(s_1,\dots,s_{n_\ell},s_{n_{\ell-1}}) \mid s_{n_{\ell-1}} \in \mathcal{S}_\mathcal{G}(u_{n_\ell})\} \subseteq P_{n_{\ell-1}}(v).$$

Because multi-layer RGNNs are sequential, we have $\phi_{\ell-1}(u_{n_\ell}) = \psi_\ell(u_{n_\ell})$. By setting

$$\hat{\phi}_{\ell-1}(u_{n_\ell};p) = \hat{\psi}_\ell(u_{n_\ell};p) =: \hat{p}$$

and $\hat{\phi}_{\ell-1}(u;p) = 0$ for $u \neq u_{n_\ell}$, layer $\kappa_{\ell-1}$ distributes relevance to one-hop extensions

$$p' = (s_1,\dots,s_{n_\ell},s_{n_{\ell-1}})$$

with the corresponding relevances

$$\hat{p}' = \hat{\phi}_{\ell-2}(u_{n_{\ell-1}};p') = \hat{\psi}_{\ell-1}|_{s_{n_{\ell-1}}}(u_{n_{\ell-1}};p') \in \mathbb{R}^{d_{\ell-1}}. \tag{9.1}$$

As shown in Section 9.3.1, the base relevance $\hat{s}_1 = \hat{\psi}_L|_{s_1}(u)$ is directly obtained for any stub $s_1 \in \mathcal{S}_\mathcal{G}(v)$. Thus, repeated application of Equation (9.1) yields relevances for all multi-hop paths $(s_1,\dots,s_n) \in P_n(v)$ with $n > 1$. This ultimately provides the relevances of all full-length paths $P_L(v)$ used in the RGNN computation graph $\mathcal{H}^L_\mathcal{G}(v) \subseteq \mathcal{G}$ originating from v.

9.4 Experimental Evaluation

We evaluate LRP for RGNNs on knowledge graphs via node classification and link prediction, abbreviated as NC and LP in the following. We analyze relevance at three structural levels:

edges, nodes, and multi-hop paths. Faithfulness is assessed using signed top-k deletion experiments, measuring how removing the most positive or most negative components affects the model output. This deletion-based evaluation follows established practice in graph explainability [247, 251]. In addition, node relevance vectors are aggregated to interpret embedding dimensions as features of RGNN predictors. The code implementation is available online.[1]

Experimental Setup. We train sequential RGCN and RGAT models of depth $L = 4$ with embedding dimension $d = 256$. Following standard implementations, identity relations r_{id} are included such that $\mathcal{G} = \overline{\mathcal{G}}$. Model parameters, including initial node embeddings $\psi(v)$, are optimized using Adam (learning rate 10^{-3}). For NC, we use the AIFB and MUTAG knowledge graph benchmarks [187, 197] (AIFB: 4 classes, MUTAG: 2 classes). RGNN embeddings $\phi(v)$ are passed to a two-layer MLP predictor π, whose logits define the initial relevance vectors $\hat{\phi}(v)$ via LRP. For LP, we use the FB15K-237 [218] and WN18RR [57] benchmarks. Models are trained using negative sampling with 20 corrupted triples per positive triple. Among several scoring functions (DISTMULT [240], TRANSE [36], COMPLEX [219], ROTATE [208]), DISTMULT performed best, yielding link scores

$$\pi_\phi(v;s) := S_\phi(v,s) = w(s)^\top \phi(v), \qquad w(s) = \phi(u) \odot \rho(r,\iota),$$

for $s = (u,r,\iota)$, where $\rho(r,\iota) \in \mathbb{R}^d$ denotes an auxiliary directed relation embedding. Since the score is a weighted sum over components of $\phi(v)$, LRP applies directly, yielding initial relevances $\hat{\phi}(v) = \phi(v) \odot w(s)$.

Evaluation Protocol. Each benchmark provides training, validation, and test splits. For NC, these consist of node–label pairs. For LP, they consist of positive triples, with training triples defining $\mathcal{G}$. Although our focus is explainability rather than predictive performance, all models achieve competitive results. Relevance is assigned to a single output activation (class logit or link score), and signed top-k deletion tests perturb this activation. To ensure comparability across benchmarks, we apply the hyperbolic tangent tanh as activation to all logits and scores and ℓ_2-normalize base embeddings after each training step. While we focus on single-activation explanations, the framework extends naturally to comparative settings via LRP, e.g., explaining why $\pi_\phi(v)_j - \pi_\phi(v)_i$ was high or low.

Evaluation Results. Faithfulness is evaluated by deleting or inverting the highest- and lowest-scoring graph components or embedding dimensions. After updating the computation graph (or embedding input), we recompute the prediction $\pi'_\phi(v)$ and measure the change in output activation using the following activation ratio as evaluation metric:

$$\rho(a,b) = \mathbb{1}_{a<b}(1 - a/b) + \mathbb{1}_{a\geq b}(b/a - 1), \qquad \rho(\pi'_\phi(v), \pi_\phi(v)) \in [-1,1).$$

Negative values indicate decreased activation, while positive values indicate an increase. Ratios are aggregated across all NC and LP benchmarks. The results are summarized in Figure 9.5, which reports two boxplots per method and architecture: blue corresponds to removing or inverting negatively relevant components (activation increase expected), and red to positively relevant components (activation decrease expected).

[1] GitHub Repository: `https://github.com/Krusinaldo9/LRP4RGNN`.

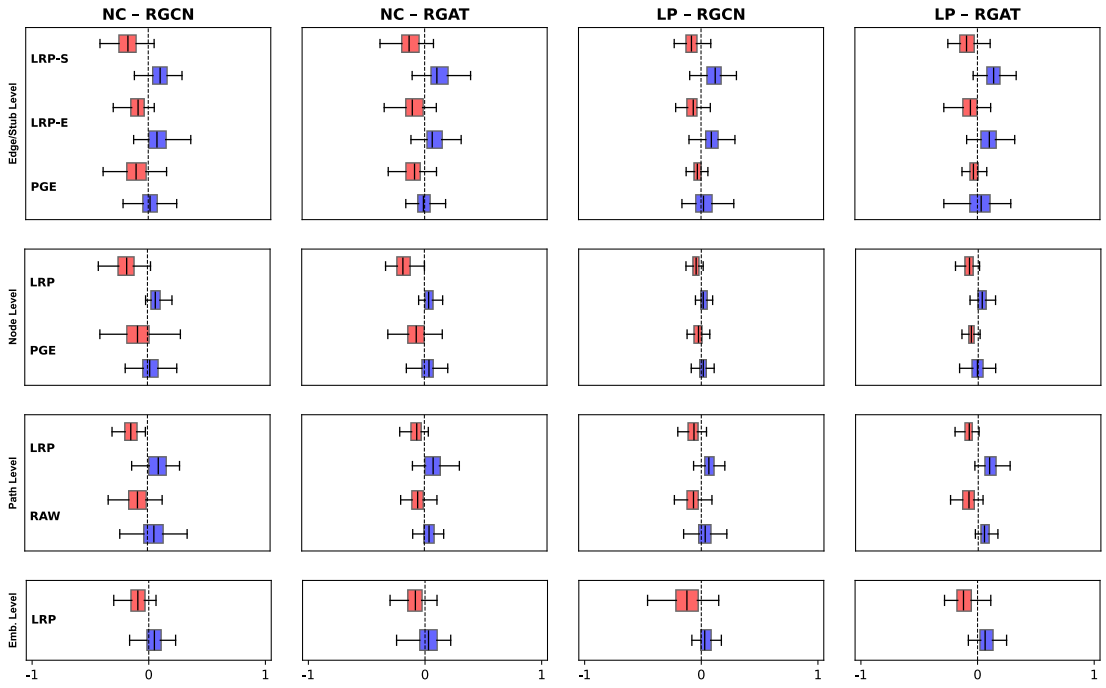


Figure 9.5: Deletion-based faithfulness w.r.t. activation ratio ρ (x-axis, $\rho \in [-1,1)$) across edge/stub, node, path, and embedding levels. Red boxplots indicate removal of the most positively relevant components, while blue boxplots indicate the most negative ones.

9.4.1 Edge/Stub-Level RGNN Explanations

We begin by evaluating explanations with respect to edges and their associated stubs in the computation graph using macro LRP. For a target node $v \in \mathcal{V}$, we consider the triples participating in paths $P_L(v)$. For every input-layer edge $e = (v,u,r)$ in this computation graph, we compute a relevance vector $\hat{e} \in \mathbb{R}^{d'}$. If $v \neq u$, the directed edge e yields two node–stub tuples, $(v,(u,r,\iota_{\text{out}}))$ and $(u,(v,r,\iota_{\text{in}}))$. Hence, unlike existing approaches, each directed edge receives two relevance vectors, combined as

$$\hat{e} = \hat{\psi}|_{(u,r,\iota_{\text{out}})}(v) + \hat{\psi}|_{(v,r,\iota_{\text{in}})}(u) \in \mathbb{R}^{d'}.$$

For self-edges $e = (v,v,r)$ we obtain $\hat{e} = \hat{\psi}|_{(v,r,\iota_{\text{id}})}(v)$. In both cases, scalar relevance scores are obtained by summing vector components. For LP tasks, we restrict ourselves to tail predictions. Given a test triple (v,u,r), a candidate tail

$$u^* = \arg\max_{u' \in \mathcal{V}} \pi_\phi\left(u'; (v,r,\iota_{\text{in}})\right)$$

is predicted and compared to u. However, our goal is not predictive performance but identifying which edges influenced the model's preference for u^*. As discussed in Section 9.2, PGExplainer serves as an unsigned baseline. Faithfulness is assessed by deleting the top-10 highest- and lowest-scoring edges or stubs. If a deletion would isolate a node, its weakest edge or stub is retained. Both PGExplainer (denoted as PGE in Figure 9.5) and LRP identify positively

relevant edges, but PGExplainer cannot capture negative evidence. LRP, in contrast, reveals both supportive and inhibitory contributions, which become even more pronounced at the stub level (denoted as LRP-E / LRP-S in Figure 9.5).

9.4.2 Node-Level RGNN Explanations

Node-level relevances are obtained by aggregating edge- and stub-level scores. For macro LRP, we apply the aggregation rule from Equation 9.3.1 to compute relevance vectors $\hat{\psi}(u) \in \mathbb{R}^{d'}$ for all $u \in \mathcal{N}_{\mathcal{G}}^{L}(v)$, whose component sums yield scalar scores $\hat{u}$. PGExplainer scores are aggregated analogously by distributing each directed edge's relevance to its incident nodes. Since removing a node implicitly removes all adjacent edges, we delete only the single highest- or lowest-scoring node, respectively. Faithfulness is evaluated using the output ratio ρ, comparing $\pi_{\phi}(v)$ with the post-deletion prediction $\pi'_{\phi}(v)$ and aggregating results across all NC and LP benchmarks. LRP again produces more faithful explanations, with the removal of a highly relevant node yielding a stronger activation drop than for PGExplainer. Ratios are generally smaller for LP tasks, which is expected since deleting a single node perturbs fewer paths than deleting individual edges or stubs. Although PGExplainer recovers some negative evidence after aggregation, LRP consistently identifies both supportive and inhibitory nodes.

9.4.3 Path-Level RGNN Explanations

Path-level explanations assess how multi-hop message passing influences a prediction. As a baseline, we employ RAW-Explainer (denoted as RAW in Figure 9.5), which samples random-walk paths and ranks them to construct explanatory subgraphs. For comparability, micro LRP is restricted to the same set of sampled walks rather than all paths in $P_L(v)$, whose full enumeration would be infeasible in practice. Since our RGNNs have depth $L = 4$, deleting two paths removes at least five edges from the computation graph. Thus, for both methods, we remove only the top-2 highest- or lowest-scoring paths. We then recompute the prediction and evaluate the resulting activation change using the ratio ρ. Both methods identify supportive and inhibiting paths. RAW-Explainer performs competitively despite lacking explicit signed relevance propagation, while micro LRP retains a slight advantage overall. The differences remain modest, reflecting that path-level perturbations affect multiple hops simultaneously and therefore produce more stable output shifts.

9.4.4 Embedding-Level RGNN Explanations

Embedding-level explanations reveal which dimensions of $\psi(v) \in \mathbb{R}^{d'}$ are most influential after LRP propagation. Since individual embedding dimensions do not have predefined semantics, we aggregate relevance by dimension to quantify their contribution to the downstream objective. For LP tasks, each prediction produces a single score. We therefore group predictions by relation type and sum their macro-LRP relevance vectors $\hat{\psi}(u)$ to identify the most positive and most negative embedding dimensions associated with each relation. For NC tasks, each prediction yields multiple logits. We propagate relevance only from the true class and aggregate the resulting relevance vectors per class. To assess faithfulness, we invert the five most or least

relevant embedding dimensions and measure the resulting activation change using the ratio ρ. Figure 9.5 shows that inverting highly relevant dimensions consistently reduces the activation. In contrast, inverting dimensions with low relevance produces only minor changes, indicating that RGNN predictions rely on a relatively small subset of strongly positive embedding dimensions and negatively relevant dimensions play a less dominant role.

9.4.5 Relevance Distribution Across Computation Graphs

We additionally examine how far the most influential structural components lie from the target node $v \in \mathcal{V}$ in order to understand whether RGNN predictions are primarily driven by local evidence or by information propagated across multiple hops. Node- and edge-level explanations are obtained via macro LRP, whereas path-level explanations rely on micro LRP. For each prediction, we determine the hop distance of the most relevant node (self-distance equals 0), the hop distance of the most relevant edge (self-edge distance equals 0), and the number of non-self-edge hops contained in the most relevant path. Figure 9.6 shows that RGNNs concentrate most relevance on the self-embedding $\psi(v)$, particularly in node-level explanations, while the influence of more distant nodes, edges, and paths decreases with hop distance. This observation reflects how the trained RGNN models utilize their representations and highlights the strong influence of self-projections.

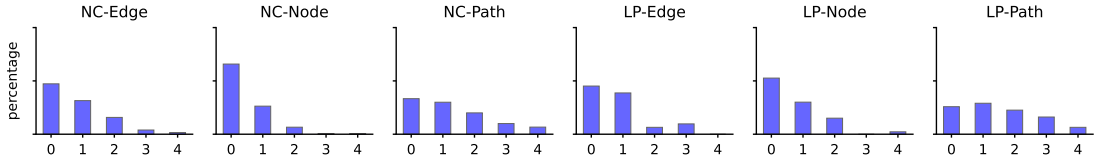


Figure 9.6: Hop distances (0–4) of the most relevant nodes, edges, and paths across NC/LP tasks. Percentages denote how often the most relevant component occurred at the respective distance from the prediction target. RGCN/RGAT results are combined due to similar distributions.

9.5 Discussion

The experimental evaluation demonstrates that LRP provides faithful and fine-grained explanations for RGNN predictions across multiple structural levels, including edges, nodes, embedding dimensions, and multi-hop paths. In particular, the results show that predictive behavior is often concentrated in a comparatively small subset of structural components and feature channels, and that LRP is able to identify both supportive and inhibiting contributions that unsigned baselines cannot recover. At the same time, the evaluation is primarily quantitative. We did not perform a detailed qualitative analysis of whether the identified explanations are semantically meaningful within the respective application domains. Such an analysis would require domain expertise as well as highly reliable underlying RGNN predictors, and constitutes an important direction for future work, particularly with respect to understanding how different model architectures influence relevance patterns.

These findings directly contribute to the broader objective of this thesis by enabling transparent feedback loops in semantic digital twins. As introduced in Chapter 5, semantic digital twins are characterized by the integration of a semantic digital shadow with predictive and decision-support capabilities. Given semantically structured representations such as knowledge graphs, RGNNs provide predictions on top of these representations. In Industry 5.0 scenarios, however, such predictions must remain transparent and interpretable to support human-centered decision-making in socio-technical systems. Accordingly, a human actor must be able to understand why a prediction was made in order to assess its validity and act upon it. The LRP-based framework proposed in this chapter addresses this requirement by grounding explanations directly in the underlying semantic structure, thereby enabling transparent and structure-aware feedback within the semantic digital twin.

At the same time, the presented approach assumes a static view of the underlying knowledge graph, as is common in most RGNN-based methods. This assumption is not fully aligned with the notion of a semantic digital shadow, which evolves continuously as new data becomes available. Consequently, an important open challenge lies in extending graph representation learning methods to account for dynamic updates while preserving structural consistency and interpretability. Addressing this challenge forms the focus of the next chapter, where we investigate how adaptability to evolving graph structures, alongside other real-world requirements in Industry 5.0 scenarios, can be incorporated into graph embedding approaches.

Chapter 10

Reconstruction-Based Graph Embedding Models

This chapter builds upon the following works:

"Dynamic Knowledge Graph Embeddings via Local Embedding Reconstructions" by Franz Krause [118], published in the *Proceedings of the ESWC Satellite Events*, Springer (2022).

"LoRE: Adaptive and Robust Graph Embeddings via Local Self-Reconstruction" by Franz Krause and Heiko Paulheim [121], published in *Machine Learning and Knowledge Discovery in Databases. Research Track*, Springer (2026).

As shown in Chapter 4, graph embeddings can be obtained deterministically through algebraic constructions or heuristics [92], but most approaches rely on gradient-based optimization. Learned embeddings have historically followed three main paradigms: factorization-based models [161, 166], transformation-based methods [36, 135], and random-walk-based techniques [86, 188]. In these approaches, embeddings are learned as self-contained parameters, with structural information incorporated through the training objective. More recently, GNNs introduced message-passing architectures that construct embeddings from local neighborhoods [41, 114, 197, 223, 224]. While these enable expressive representations, they are still often trained with objectives originally designed for self-contained embeddings, despite emerging GNN-specific paradigms such as contrastive learning [243, 255].

In practical deployments, graphs are frequently incomplete, dynamic, and structurally redundant [99, 239]. Requirement R5 for knowledge modeling in Industry 5.0 (Section 5.2) therefore calls for representations that are robust under incompleteness and adaptive to graph updates. In the context of semantic digital twins, this directly affects graph embeddings used in downstream learning. Namely, observations from the Teaming.AI project suggest that embeddings should remain structurally consistent, compatible with open-world semantics, sensitive to local distinctions, and adaptable to newly observed facts. However, many existing approaches rely on global edge-prediction objectives with negative sampling and remain inherently transductive, requiring retraining as graphs evolve [254]. At the same time, embeddings are often tailored to

specific downstream tasks such as link prediction or node classification. In contrast, we aim to learn a graph embedding of the vertex set that captures structural properties in a task-agnostic manner and enables reuse across applications. To this end, we introduce *self-reconstructive learning* as a graph embedding paradigm in which supervision is derived from local neighborhood context rather than individual relations. This paradigm is first instantiated by NAVI, which formulates embedding learning as the reconstruction of a fixed base embedding using a GNN, thereby enabling the adaptation of precomputed embeddings to graph updates. LORE (LOcally Reconstructed Embeddings) generalizes this idea by lifting the assumption of a fixed base embedding and enabling the joint learning of base embeddings and their reconstruction from local graph context in an end-to-end manner. This self-reconstruction formulation unifies representation learning and reconstruction within a single objective, yielding general-purpose embeddings while leveraging the inductive capabilities of GNNs (cf. Section 4.3).
By grounding learning in local self-reconstruction, these approaches are well suited to incomplete and evolving graphs, enabling the incorporation of newly observed facts without retraining. In particular, reconstructions are performed without directly relying on a node's own base embedding, thereby counteracting the observation from Section 9.4.5 that GNNs tend to concentrate a large share of predictive relevance on node-specific base representations. Moreover, we observe a regularizing effect of reconstruction, reflected in clearer separability of node representations in the embedding space. Experiments show that LORE achieves competitive performance compared to established embedding models in static settings, with particular advantages in relational and incomplete graph settings. NAVI is generally less performant than LORE and depends on a suitable base embedding. Nevertheless, it highlights the effectiveness of reconstruction for adapting existing, otherwise transductive embeddings. This is particularly relevant in scenarios where an existing base embedding is to be preserved. In this chapter, we therefore study reconstructive learning in both static and dynamic settings by considering $\mathcal{G} = \mathcal{G}_t$ as an initial snapshot of a graph trajectory $(\mathcal{G}_t)_{t\in\mathbb{T}}$. We first elaborate on the sub-requirements implied by R5, before formally introducing reconstructive learning as a graph embedding paradigm.

10.1 Real-World Requirements for Graph Embeddings

Insights obtained from the deployment of semantic digital shadows within the Teaming.AI project revealed four practical requirements for graph embeddings when used as enablers of feedback loops from semantic digital shadows to the represented system. These are formalized as sub-requirements R5.1–R5.4 of the general requirement R5, which establishes the need for such feedback mechanisms, for instance via predictions derived from GNN-based models. In this context, GNN-based embeddings must respect key domain characteristics such as incompleteness, dynamics, sparsity, and heterogeneity of real-world graphs [22, 99].

R5.1 - Structural Consistency:

In real-world graphs, multiple nodes may share identical topologies, for example when integrating heterogeneous data sources or representing entities participating in identical relationships. Such nodes are structurally indistinguishable and should ideally receive identical embeddings.

However, existing embedding methods do not explicitly enforce this requirement. Parametric graph embedding models inherently introduce noise through random initialization and stochastic training procedures, which can lead to different embeddings for structurally equivalent nodes. In self-contained models, node embeddings are independent trainable parameters determined only indirectly through the optimization objective, further amplifying this effect. A similar issue arises in standard GNNs, which, although they construct embeddings from local context, typically incorporate node-specific base embeddings into the forward pass, so that structurally equivalent nodes may still receive different embeddings due to differences in their base embeddings. Thus, structural symmetry is not preserved by default, even in message-passing architectures.

R5.2 - Open-World Compatibility:

Many real-world graphs, including the RDF graphs considered in this work for the construction of semantic digital shadows, follow the OWA, where the absence of an edge does not imply that the corresponding relation is false but merely unknown. In contrast, many graph embedding models rely on negative edge sampling during training, treating non-observed edges as negative instances and thereby implicitly assuming a closed-world interpretation. While this assumption is often acceptable for large-scale benchmark graphs such as DBpedia [17] or Wikidata [227], it becomes problematic in domain-specific settings, where graphs are typically smaller, denser, and inherently incomplete. In such cases, incorrectly treating unknown relations as false may suppress valid but unobserved knowledge and, in safety-relevant domains such as Industry 5.0, can even lead to critical misrepresentations. Consequently, negative edge sampling should either be carefully designed to remain compatible with open-world semantics or be omitted altogether.

R5.3 - Local Similarity Preservation:

Graph embeddings are typically trained using global distance-based similarity measures, and many training objectives enforce separation between positive and negative examples through margin-based or contrastive losses. While effective for tasks such as link prediction, these objectives may distort relational structure in heterogeneous graphs by emphasizing global separation over local consistency. In particular, nodes that are globally similar but locally distinguishable may be forced apart in the embedding space, weakening the representation of fine-grained differences within dense regions. Consequently, embedding models should preserve local distinctions induced by neighborhood context while maintaining similarity among structurally related nodes.

R5.4 - Dynamic Adaptability:

In practical deployments, such as the semantic digital shadows of BPMN-driven industrial processes introduced in this work, graphs are rarely static and may evolve continuously over time. However, most embedding models are inherently static and learn representations for a fixed graph during training. In particular, self-contained embeddings are transductive by design, as each node is associated with a dedicated trainable parameter that cannot be inferred for unseen nodes or updated structures without retraining. When the graph changes, such models therefore require costly retraining to incorporate new information, which prevents efficient or real-time

adaptation. While GNNs provide a partial remedy through their ability to construct embeddings from local neighborhoods, this inductive capability is inherently limited. Specifically, GNNs can infer embeddings $\phi(v)$ for updated or previously unseen nodes if their neighborhoods are known and neighboring nodes already possess base embeddings. However, this inductiveness applies only to the construction step, whereas the underlying base embeddings $\psi(v)$ remain transductive parameters learned during training. As a result, the model cannot fully generalize to structural updates without either approximating or retraining these base representations. Accordingly, embedding models for real-world applications should support efficient and localized updates, enabling the incorporation of new data without requiring retraining of model parameters.

10.2 Methodology

We introduce LoRE, a graph embedding framework that extends GNN architectures with local self-reconstruction as a novel self-supervised training paradigm designed to satisfy the requirements introduced above. Within this section, we first assume sets of nodes and edge labels $\mathcal{V}, \mathcal{M} \neq \emptyset$ and a graph $\mathcal{G} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{M}$ covering $(\mathcal{V}, \mathcal{M})$, i.e., $\mathcal{V}(\mathcal{G}) = \mathcal{V}$ and $\mathcal{M}(\mathcal{G}) = \mathcal{M}$. In this setting, we intend to build a general-purpose graph embedding $\phi \in (\mathbb{R}^d)^{\mathcal{V}}$, i.e., we consider $\mathcal{X} = \mathcal{V}$. We begin with the requirement of structural consistency (R5.1), which states that identical graph neighborhoods must yield identical embeddings, that is

$$\mathcal{S}_{\mathcal{G}}(v) = \mathcal{S}_{\mathcal{G}}(u) \;\Rightarrow\; \phi(v) = \phi(u).$$

Deterministic embedding constructions usually satisfy this requirement, unless external node embeddings are considered. Learned embeddings, on the other hand, typically start from randomly initialized base embeddings, which may break structural symmetry during training (cf. Section 4). To make constructed embeddings invariant to such initialization noise, LoRE actively ignores self-information within the GNN neighborhood aggregation process. In contrast to standard GNN implementations that assume a closed graph $\mathcal{G} = \overline{\mathcal{G}}$ with explicit self-relations, we instead consider the irreflexive reduction of the graph

$$\mathcal{G}^{\circ} := \{(v,u,m) \in \mathcal{G} \mid v \neq u\},$$

i.e., we assume $\mathcal{G} = \mathcal{G}^{\circ}$, as the graph obtained by removing all self-edges. This implies that nodes whose connectivity is defined solely by self-relations do not contribute to the construction of other node embeddings. As a result, node representations are constructed exclusively from neighborhood context rather than from node-specific base embeddings. Consequently, nodes with identical neighborhoods yield identical constructed embeddings. In the case of multi-layer GNN architectures, this modification only needs to be applied in the first layer per node, since subsequent layers already operate on constructed embeddings. Most importantly, this step ensures that the constructed embedding $\phi(v)$ does not depend on $\psi(v)$, which is the key requirement used by the end-to-end self-reconstruction mechanism of LoRE. The architectural implications are illustrated in Figure 10.1, highlighting the masking of respective base embeddings during the GNN forward pass. In typical GNN-based learning architectures, base embeddings $\psi \in (\mathbb{R}^{d'})^{\mathcal{V}}$ are first aggregated to constructed embeddings $\phi \in (\mathbb{R}^d)^{\mathcal{V}}$, which are subsequently used within self-supervised training paradigms such as edge-scoring or contrastive learning.

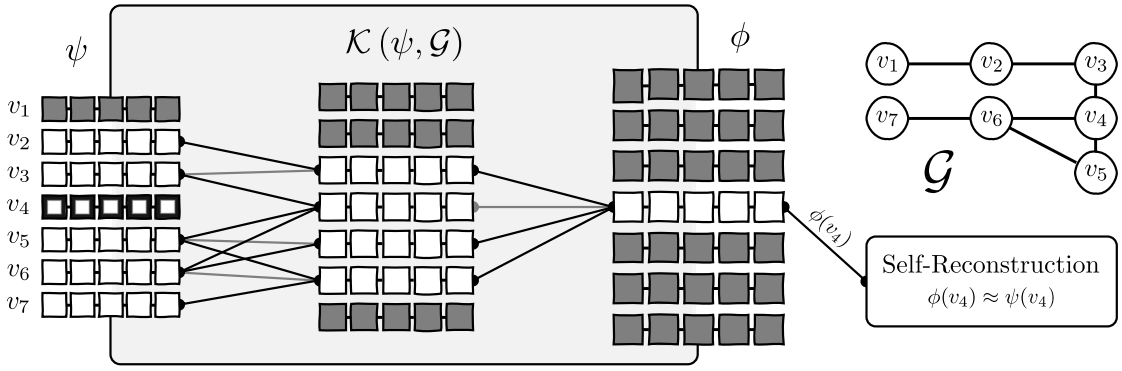


Figure 10.1: Self-reconstruction objective in LORE: a node-specific base embedding is masked during the forward pass and reconstructed from its neighborhood context.

In contrast, training in LORE builds upon the NAVI approach by replacing edge-level supervision with a node-level reconstruction objective. While NAVI reconstructs a given base embedding ψ via a trainable constructor $\phi_\theta = \mathcal{K}_\theta(\psi, \mathcal{G})$, LORE generalizes this idea by enabling the learning of base embeddings ψ_θ jointly in an end-to-end manner:

$$\phi_\theta = \mathcal{K}_\theta(\psi_\theta, \mathcal{G}) \approx \psi_\theta \,.$$

Instead of predicting edges or walk co-occurrences, local self-reconstruction is conceptually related to contrastive learning as it analyzes and optimizes consistency between different representations of the same node. However, unlike contrastive approaches, self-reconstruction operates within a single embedding space. The model reconstructs each node's base embedding $\psi_\theta(v)$ from the forwarded representation $\phi_\theta(v)$, using embeddings of equal dimensionality ($n' = n$). This formulation avoids negative edge sampling and evaluates similarity locally within the context of each node (R5.2). To facilitate reconstruction, base and constructed embeddings are constrained to lie in comparable regions of the embedding space. LORE applies ℓ^2-normalization to the base embeddings after each training step:

$$\|\psi_\theta(v)\|_2 = 1.$$

The embeddings $\phi(v)$ retain some flexibility through the element-wise activation $\sigma = \tanh$:

$$\phi_\theta(v) \in (-1, 1)^d.$$

Within its GNN constructor, LORE implements

$$F_{m,\iota,\theta}(\psi_\theta(u)) = \psi_\theta(u) + \left(1 - 2 \cdot \mathbb{1}_{\{\iota = \iota_{out}\}}\right) \cdot \rho_\theta(m)$$

as lean, translational stub embeddings. Since existing attention mechanisms incorporate a node's base embedding, as shown in Equation 4.2, we propose a self-attention pooling mechanism $c_{m,\iota,\theta}(u, v) = c_\theta(s, v)$ for stubs $s = (u, m, \iota)$, which is independent of the base embedding:

$$c_\theta(s, v) = \frac{1}{|\mathcal{S}_\mathcal{G}(v)|} \cdot \sum_{s' \in \mathcal{S}_\mathcal{G}(v)} \left(\left(\sum_{s'' \in \mathcal{S}_\mathcal{G}(v)} \exp\left(\frac{q_{\theta,s}^\top k_{\theta,s''}}{\sqrt{d}} \right) \right)^{-1} \cdot \exp\left(\frac{q_{\theta,s}^\top k_{\theta,s'}}{\sqrt{d}} \right) \right).$$

Stubs $s \in \mathcal{S}_\mathcal{G}(v)$ are mapped to $q_{\theta,s} = W_{q,\theta} \cdot z_\theta(s)$ and $k_{\theta,s} = W_{k,\theta} \cdot z_\theta(s)$ as queries and keys, with learnable matrices $W_{q,\theta}, W_{k,\theta} \in \mathbb{R}^{d \times d}$. Scaled dot-products compute pairwise similarities, and a stub-wise softmax yields the average attention weight assigned to s with respect to all other stubs, independently of $\psi_\theta(v)$.

LoRE is built upon a locality-preserving loss, which is based on the cosine distance

$$\delta_{\cos} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R},$$

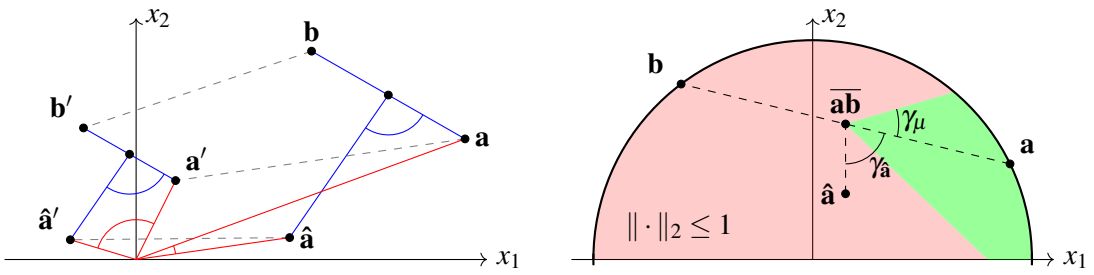
measuring the angular difference between vectors:

$$\delta_{\cos}(\mathbf{a}, \mathbf{a}') = 1 - \left(\mathbf{a} \cdot \mathbf{a}'\right) \cdot \left(\|\mathbf{a}\|_2 \cdot \|\mathbf{a}'\|_2\right)^{-1} \in [0,2], \quad \mathbf{a}, \mathbf{a}' \in \mathbb{R}^d.$$

In contrast to ℓ^p distances, which are translation-invariant but sensitive to scaling, $\delta_{\cos}$ is invariant under positive scalar multiplication but not translation-invariant. To incorporate translation- and scale-invariance, thereby enabling locality-awareness, we define δ_{LoRE} as a perspective-preserving cosine distance between a prediction $\hat{\mathbf{a}}$ and target $\mathbf{a}$ relative to a reference $\mathbf{b}$:

$$\delta_{\text{LoRE}}(\hat{\mathbf{a}}, \mathbf{a}; \mathbf{b}) = \delta_{\cos}(\hat{\mathbf{a}} - \overline{\mathbf{ab}}, \mathbf{a} - \overline{\mathbf{ab}}) \quad \text{with} \quad \overline{\mathbf{ab}} := (\mathbf{a} + \mathbf{b})/2.$$

The local anchor $\overline{\mathbf{ab}}$ re-centers the prediction and target with respect to the reference vector before measuring their angular difference. The ℓ^2-normalization positions base embeddings on the unit sphere, so the anchor $\overline{\mathbf{ab}}$ lies inside the unit ball. This adjustment allows the distance to be scale-invariant and locally translation-invariant relative to $\overline{\mathbf{ab}}$, as illustrated in Figure 10.2A. Standard margin-based optimization compares two distances and enforces a margin $\mu \geq 0$ between them. LoRE also supports margins, as illustrated in Figure 10.2B. The margin μ corresponds to an angle γ_μ and predictions $\hat{\mathbf{a}}$ with $\gamma_{\hat{\mathbf{a}}} > \gamma_\mu$ are penalized. Given an angle $\gamma_\mu \in [0, \frac{\pi}{2})$, the corresponding margin μ can be determined via $\mu = 1 - \cos(\gamma_\mu)$.



(A) Perspective-preserving cosine variant δ_{LoRE} (blue) compared to standard cosine distance (red).

(B) Interpretation of the δ_{LoRE} loss using angular constraints relative to the anchor vector $\overline{\mathbf{ab}}$.

Figure 10.2: Visual representation of δ_{LoRE} characteristics and training objectives.

The corresponding angle is defined locally, meaning that adequate angular separation can be obtained even when $\mathbf{a}$ and $\mathbf{b}$ are close in the embedding space, which is not possible with standard distances such as $\delta_{\cos}$ or ℓ^p. To preserve local similarity during training (R5.3), we thus use

$$\lambda_{\text{LoRE},\theta}(v,u) = \max\left(\delta_{\text{LoRE}}(\phi_\theta(v), \psi_\theta(v); \psi_\theta(u)) - \mu, 0\right)$$

as the reconstruction loss for $v \in \mathcal{V}$ with reference node $u \in \mathcal{V}$, where $u \neq v$.

The training procedure for graph embedding models introduced in Chapter 4 is thus defined as

$$\mathcal{L}_{\text{LoRE}}(\mathcal{G},\theta) = \sum\nolimits_{v,u\in\mathcal{V}} \lambda_{\text{LoRE},\theta}(v,u)\,.$$

The parameters are accordingly optimized via

$$\mathcal{T}(\mathcal{G}) \in \arg\min_{\theta\in\Theta} \mathcal{L}_{\text{LoRE}}(\mathcal{G},\theta),$$

yielding a parameter setting $\theta^* = \mathcal{T}(\mathcal{G}) \in \Theta$ and $\phi_{\theta^*} = \mathcal{K}_{\theta^*}(\psi_{\theta^*},\mathcal{G})$. In the case of NAVI, $\psi_{\theta^*} = \psi$ holds for a precomputed embedding $\psi \in (\mathbb{R}^d)^{\mathcal{V}}$.

As discussed in Section 4.3, GNN layers inherently provide single-hop inductiveness at the level of constructed embeddings. When new nodes or edges appear, a modified graph $\mathcal{G}'$ is processed by the constructor $\mathcal{K}_{\theta^*}$, yielding embeddings $\phi'_{\theta^*} = \mathcal{K}_{\theta^*}(\psi_{\theta^*},\mathcal{G}')$, provided that neighboring nodes with base embeddings exist [254]. LoRE extends this inductiveness to the base embedding level through its self-reconstruction objective. During training, base embeddings are implicitly masked and reconstructed from neighborhood context, simulating inference for previously unseen nodes. Consequently, if a new node connects to at least one node with an existing base embedding, its base embedding can be approximated using the same mechanism. The resulting constructed embedding $\phi'_{\theta^*}(v')$ is projected onto the unit sphere via ℓ^2 normalization to obtain $\psi'_{\theta^*}(v')$, which can subsequently participate in further reconstructions. This enables efficient adaptation of the base embeddings under graph updates (R5.4), effectively extending GNN inductiveness beyond the immediate neighborhood. In the case of retraining, LoRE can therefore also provide suitable initializations and no negative edge sampling is required.

10.3 Experimental Evaluation

We evaluate LoRE along three complementary dimensions: static embedding quality via node classification, adaptability under dynamic graph updates, and geometric separability of learned representations. Node classification is a standard downstream task for evaluating graph embeddings, where pretrained node representations are used to predict externally defined labels. We focus on this external evaluation rather than link prediction. While both tasks aim to predict missing information [176], link prediction relies on artificially generated negative samples that may violate the OWA, whereas node labels provide a stable and verifiable ground truth. We consider two graph settings: undirected and homogeneous graphs as well as directed and relational graphs. We compare LoRE against established graph embedding paradigms as baselines. All models use embeddings of dimension $n = 256$ trained with Adam using a learning rate $5 \cdot 10^{-5}$, batch size $\tau_b = 32$, and maximum stub count $\tau_s = 16$. The angular margin γ_μ of δ_{LoRE} is selected from $\{0, \pi/6\}$ via grid search, and a dropout rate of 0.2 is applied throughout. All models are trained for up to 100 epochs with early stopping based on validation performance. In settings without explicit validation splits, in particular for the knowledge graph benchmarks, both hyperparameters and stopping criteria are determined via nested cross-validation.
Except for standalone (R)GAT, all embedding models are pretrained in a self-supervised manner using a GAT-based GNN constructor, where the respective embedding paradigms define the

training objective, and the resulting embeddings are used as fixed features for downstream evaluation. Standalone GAT and RGAT serve as locally trained baselines that directly optimize node classification end-to-end using two dense layers and a softmax output. Downstream evaluation uses support vector machines (SVMs) and random forests (RFs), with $C \in \{0.01, 0.1, 1, 10, 100\}$ for SVMs and $\{100, 300, 500\}$ trees and depths $\{10, 30, 50\}$ for RFs. Embedding models and classifiers are trained independently ten times and results are averaged. Unless stated otherwise, we use single-layer GNNs, ensuring that node representations depend only on immediate neighborhoods. Consequently, newly added nodes cannot influence existing representations. Only in the dynamic update experiments, we additionally consider two-layer variants for the baselines to analyze how newly added nodes affect existing node representations via multi-hop message passing. Experiments were conducted on an NVIDIA RTX 2080 Ti GPU with 12 GB VRAM. The PyTorch implementation of LoRE is publicly available.[1]

10.3.1 Embedding Undirected and Homogeneous Graphs

We evaluate on the Planetoid citation networks Cora, CiteSeer, and PubMed [245] and the Amazon-Computers co-purchase graph from SNAP [241]. In these undirected graphs, nodes represent papers or products and edges represent citations or co-purchases. Dataset statistics are summarized in Table 10.1. We adopt the standard Planetoid and SNAP splits for comparability.

	Nodes	Edges	Classes	Train	Validation	Test
Cora	2,708	5,278	7	140	500	1,000
CiteSeer	3,327	4,552	6	120	500	1,000
PubMed	19,717	44,324	3	60	500	1,000
Amazon	13,752	245,861	10	8,251	2,750	2,751

Table 10.1: Statistics of undirected and homogeneous graph embedding benchmarks. Train/Validation/Test denote the node classification splits. Class balances are color-coded.

Baselines include HOPE [166], Node2Vec [86], GRACE [255], and an end-to-end GAT [224]. For fair comparison, embedding-based baselines are trained using the same embedding dimension, optimizer (Adam), learning rate, batch size, and dropout configuration, and cosine distance is used as the similarity metric. HOPE is tuned over embedding ranks and regularization strengths to match the target dimensionality and prevent overfitting. For Node2Vec, we consider random walk parameters with walk length 10, context size 5, and 20 walks per node. GRACE is trained using feature masking and edge dropout augmentations. The GAT baseline is trained end-to-end using a single attention layer with the same hidden dimensionality. Results are shown in Table 10.2. GAT, GRACE, and LoRE achieve the highest accuracies overall, with LoRE exhibiting the lowest variance across runs. Notably, downstream predictors based on GRACE and LoRE can match the performance of an end-to-end GAT explicitly targeted towards the classification objective considered in the evaluation setup.

[1] GitHub Repository: `https://github.com/Krusinaldo9/LoRE`

Dataset	HOPE	Node2Vec	GRACE	GAT	LoRE
Cora	81.73 ±1.62	82.21 ±1.48	**85.11** ±2.05	84.72 ±1.34	84.88 ± **0.95**
CiteSeer	70.88 ±1.90	72.15 ±1.74	73.40 ±2.12	**73.41** ±**0.96**	73.12 ±1.04
PubMed	79.64 ±1.72	79.12 ±1.55	81.95 ±1.63	**83.01** ±1.17	82.21 ±**1.14**
Amazon	85.91 ±1.81	86.74 ±1.44	88.18 ±1.13	88.44 ±0.99	**88.90** ±**0.96**

Table 10.2: Classification accuracies (± std) on undirected and homogeneous benchmarks.

10.3.2 Embedding Directed and Relational Graphs

We evaluate on the benchmark RDF knowledge graphs AIFB, MUTAG, BGS, and AM [187, 197]. Each dataset provides an RDF knowledge graph with literal annotations and predefined train–test splits of node IRIs and external class labels. Within our preprocessing, boolean literals are converted into instance–class relations by introducing two nodes (true and false) per property, while non-boolean literals are removed. Dataset statistics are summarized in Table 10.3.

	Nodes	Relations	Edges	Classes	Train	Test
AIFB	2,835	22	20,264	4	140	36
MUTAG	22,506	18	64,494	2	272	68
BGS	103,055	62	529,945	2	117	29
AM	958,232	45	3,299,143	11	802	198

Table 10.3: Statistics of directed and relational benchmarks after removing non-boolean literals. Class balances are color-coded.

Baselines include RESCAL [161], RDF2Vec [188], KGCL [243], and an end-to-end RGAT [41]. From the translational models (TransE [36], TransD [106], and TransR [135]), we report only TransR, which achieved the strongest overall performance in preliminary experiments. All baselines follow the unified training protocol described above. In particular, embeddings are ℓ^2-normalized and trained with identical dimensionality and optimizer settings, while hyperparameters for matrix factorization and translational models are selected via grid search. Since the benchmarks do not provide validation splits, nested cross-validation is used for model selection. The results are shown in Table 10.4. In the directed and relational setting, the contrastive baseline KGCL does not match the performance of LoRE. More importantly, downstream predictions based on LoRE embeddings consistently outperform the end-to-end predictions of RGAT on MUTAG, BGS, and AM. The apparent advantage of end-to-end RGAT on AIFB is not statistically meaningful, as ten of the 36 test nodes are topologically indistinguishable. Due to LoRE's neighborhood-invariance, these nodes receive identical embeddings, yielding the theoretical maximum accuracy of 91.67% by predicting the majority class. This behavior highlights that LoRE consistently maps structurally equivalent nodes to identical representations, thereby fulfilling the structural invariance requirement R5.1 from Section 10.1.

Dataset	RESCAL	TransR	RDF2Vec	KGCL	RGAT	LoRE
AIFB	84.10 ±1.28	87.22 ±1.63	88.89 ±1.77	91.30 ±1.22	**92.22** ±1.53	91.67 ±**0.00**
MUTAG	70.50 ±1.52	75.00 ±1.81	73.68 ±1.83	75.29 ±1.48	83.97 ±1.08	**85.29** ±**0.91**
BGS	67.80 ±1.67	69.31 ±1.89	68.28 ±1.94	72.41 ±1.39	73.45 ±1.11	**74.14** ±**1.03**
AM	71.25 ±1.63	74.75 ±1.79	87.47 ±1.68	91.26 ±1.26	92.42 ±**0.88**	**92.93** ±0.96

Table 10.4: Classification accuracies (± std) on directed and relational benchmarks.

10.3.3 Adaptability to Graph Updates

To evaluate dynamic adaptability (R5.4), we use the same benchmarks and compare LoRE with KGCL [243], RGAT [41], NodePiece [75], and NAVI [118]. We consider two complementary dynamic scenarios. We first simulate incremental updates under a fixed set of nodes and relation types. In this setting, graph dynamics arise solely from changes in the edge set. Concretely, we randomly remove half of each test node's edges before training while ensuring that at least one edge remains. This controlled perturbation reflects common real-world scenarios where graphs are incomplete and new edges are discovered incrementally. Formally, we consider a graph trajectory $(\mathcal{G}_t)_{t\in\mathbb{T}}$ and an initial snapshot $\mathcal{G}_t$, which is used to optimize the base embeddings and retrieve the constructor $\mathcal{K}_{\theta^*}(\psi_{\theta^*}, \mathcal{G}_t)$. In the following, we denote this by $\mathcal{K}_t$ and ψ_t. Accordingly, we compute

$$\phi_t = \mathcal{K}_t(\psi_t, \mathcal{G}_t)$$

as input to downstream classifiers, yielding performance scores at time t. At a later time $t' > t$, we consider the updated graph

$$\mathcal{G}_{t'} \supseteq \mathcal{G}_t,$$

where the previously removed edges are reinserted. We then construct updated embeddings

$$\phi_{t'} = \mathcal{K}_t(\psi_t, \mathcal{G}_{t'}),$$

i.e., the base embeddings and message passing parameters are reused and embeddings are recomputed without retraining the classifiers. Each experiment is repeated ten times with identical perturbations across models to ensure strict comparability. For NAVI, we use pretrained RDF2Vec embeddings as base embeddings. The results in Table 10.5 show that the predictive performance of KGCL, RGAT, and NodePiece typically degrades after updates, whereas NAVI and LoRE consistently improve. While end-to-end RGAT typically performs best on the incomplete graph, LoRE consistently achieves the strongest performance after edge reinsertions. This behavior highlights a key advantage of reconstruction-based embeddings: both approaches are able to incorporate newly observed graph facts directly through neighborhood reconstruction, without retraining the embedding model. NAVI already demonstrates this capability using fixed RDF2Vec base embeddings. However, LoRE extends this idea by learning base embeddings end-to-end, yielding the strongest post-update performance across all datasets.

Dataset	Phase	KGCL	RGAT	NodePiece	NAVI	LORE
AIFB	Start	69.10 ± 4.87	74.27 ± 4.08	68.76 ± 5.32	72.57 ± **3.54**	**76.03** ± 3.96
	End	65.44 ± 3.63	73.86 ± 4.72	64.08 ± 4.66	75.17 ± **1.17**	**85.20** ± 1.33
MUTAG	Start	61.59 ± 5.26	**65.17** ± 4.67	55.40 ± 7.86	60.44 ± **3.22**	63.60 ± 3.48
	End	57.94 ± 4.98	66.02 ± 4.92	50.29 ± 5.63	65.04 ± **0.89**	**66.18** ± 1.14
BGS	Start	52.30 ± 5.81	**69.40** ± 5.03	56.45 ± 6.33	69.03 ± 3.85	68.20 ± **2.26**
	End	49.75 ± 5.39	63.14 ± 5.58	57.78 ± 4.00	71.58 ± 1.21	**72.36** ± **0.97**
AM	Start	61.01 ± 4.33	**62.18** ± 4.41	61.06 ± 4.97	55.23 ± **3.37**	60.79 ± 3.59
	End	60.80 ± 4.44	59.96 ± 4.18	60.34 ± 6.98	58.50 ± **1.24**	**65.19** ± 1.19

Table 10.5: Classification accuracies (± std) before and after edge-level perturbations and embedding updates via neighborhood reconstruction.

To further analyze the sensitivity to localized structural changes, we consider a second setting in which, for each test node, at most one neighboring node is excluded from its local neighborhood. Test nodes are processed in random order, and for each node, a neighbor is selected that has not been excluded previously and is not shared with neighborhoods affected by earlier perturbations, if such a neighbor exists. This ensures that perturbations remain localized and non-overlapping across test nodes. Analogous to the previous setting, we again consider a graph trajectory $(\mathcal{G}_t)_{t\in\mathbb{T}}$. However, instead of removing edges, we construct an initial snapshot $\mathcal{G}_t$ in which selected neighboring nodes are absent. Consequently, we obtain $\mathcal{V}_{t'} \setminus \mathcal{V}_t \neq \emptyset$, i.e., nodes that are not present at time t appear at a later time $t' > t$ together with their incident edges in the updated graph $\mathcal{G}_{t'} \supseteq \mathcal{G}_t$. As before, we obtain $\phi_t = \mathcal{K}_t(\psi_t, \mathcal{G}_t)$ for training downstream classifiers. In contrast to the previous setting, base embeddings for newly appearing nodes are first constructed and ℓ^2-normalized, yielding $\psi_{t'} \in (\mathbb{R}^d)^{\mathcal{V}_{t'}}$ with

$$\psi_{t'}(v) = \psi_t(v) \quad \text{for} \quad v \in \mathcal{V}_t.$$

We then compute updated embeddings as

$$\phi_{t'} = \mathcal{K}_{t'}(\psi_{t'}, \mathcal{G}_{t'}),$$

without retraining the model, where $\mathcal{K}_{t'}$ shares the same parametrization as $\mathcal{K}_t$ and differs only with respect to the input embedding size. This results in a minimal but structurally meaningful perturbation, as entire local neighborhoods are partially removed rather than individual edges. Importantly, the nodes in $\mathcal{V}_{t'} \setminus \mathcal{V}_t$ are not prediction targets but constitute structural context that must be incorporated when computing updated embeddings.

The results in Table 10.6 further confirm that LORE consistently matches or outperforms the considered baselines after structural updates. In contrast to the edge-deletion experiments, baseline performances again mostly drop after updates, although the degradation is less pronounced due to the more localized nature of the perturbation. Notably, both LORE and NAVI show robust or improved performance, highlighting their ability to incorporate newly available structural information without retraining. For the AIFB dataset, performance differences between models are comparatively small. This can likely be attributed to the higher graph density, where nodes exhibit an average degree of approximately 10, compared to average degrees of roughly 3–5

in the other datasets. In such dense settings, the removal or addition of a single neighboring node has a less pronounced impact on the overall neighborhood structure, thereby reducing the relative advantage of reconstruction-based approaches.

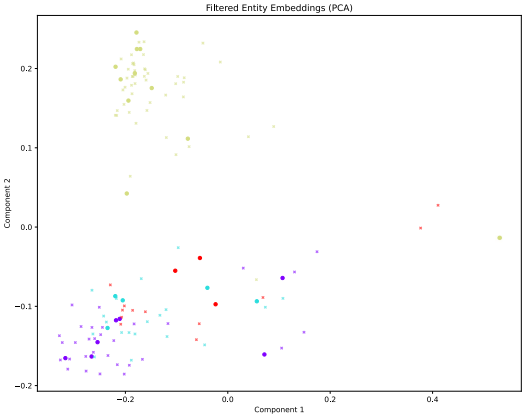
A key observation across all datasets is that, in contrast to the considered baselines, both LoRE and NAVI require only a single GNN layer. This highlights the efficiency of reconstruction-based approaches, as they achieve competitive or superior performance without relying on deeper architectures. The ability to effectively propagate local structural information within a single layer further underlines the suitability of these models for dynamic and inductive graph scenarios.

Dataset	Phase	KGCL	RGAT	NodePiece	NAVI	LoRE
AIFB	Start	84.32 ± 2.10	**88.28** ± 2.64	83.45 ± **2.10**	82.83 ± 2.31	86.92 ± 2.16
	End	83.73 ± 2.69	85.64 ± 2.80	84.13 ± 2.76	83.66 ± **2.83**	**87.99** ± **1.87**
MUTAG	Start	75.34 ± 4.77	**79.66** ± 3.52	72.23 ± 3.20	74.58 ± 3.66	77.81 ± **2.85**
	End	71.10 ± 5.21	75.08 ± 4.13	70.46 ± 3.71	76.24 ± 3.25	**80.78** ± **2.93**
BGS	Start	66.90 ± 4.96	**68.73** ± 3.57	63.84 ± 3.40	66.59 ± 3.25	68.44 ± **3.09**
	End	62.34 ± 4.70	68.14 ± 4.27	57.78 ± 5.63	65.13 ± 2.86	**68.94** ± **2.55**
AM	Start	74.08 ± 5.83	**77.39** ± 4.01	74.49 ± 5.29	73.01 ± 4.88	75.41 ± **3.68**
	End	73.75 ± 4.79	**77.98** ± 3.56	72.70 ± 4.89	74.77 ± 3.33	76.54 ± **2.49**

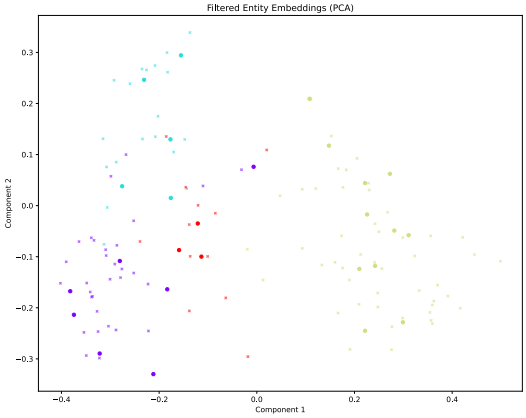
Table 10.6: Classification accuracies (± std) before and after node-level perturbations and embedding updates via neighborhood reconstruction.

10.3.4 Separability in the Embedding Space

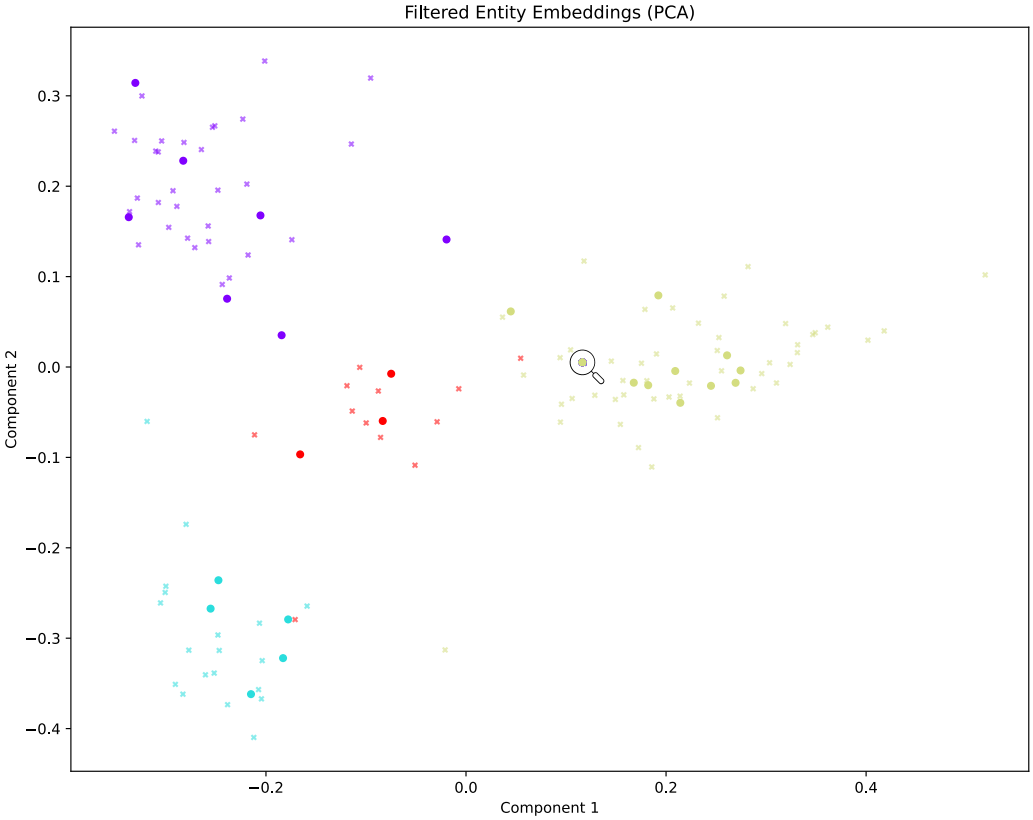
Figure 10.3 shows Principal Component Analysis (PCA) projections of embeddings for the AIFB dataset. Compared to KGCL and RGAT, LoRE produces more compact clusters and clearer class separation. Similar patterns were observed across other datasets, although the effect is most pronounced on AIFB. The magnifying glass highlights structurally equivalent nodes belonging to different classes. Due to the neighborhood-invariance of LoRE (R5.1), such nodes receive highly similar embeddings, as their local graph context is indistinguishable, resulting in embeddings that lie within or near the boundary of the majority class cluster rather than forming separate clusters. This behavior reflects a fundamental trade-off between structural consistency and discriminative separability: while identical neighborhoods induce similar representations, class-specific information introduces slight deviations that preserve separability. Notably, although cluster separation was less visually pronounced in some datasets than in the AIFB case, this boundary effect was consistently observed, with structurally identical nodes of different types located within or close to dominant clusters. This provides an initial insight into how reconstruction-based embeddings organize structural and semantic information, and suggests promising directions for future research, particularly in explicitly controlling the separation of structurally equivalent nodes with differing labels.



(A) KGCL embeddings.



(B) RGAT embeddings.



(C) LoRE embeddings.

Figure 10.3: PCA projections of AIFB embeddings for KGCL, RGAT, and LoRE. The magnified region shows neighborhood-invariance via local self-reconstruction: structurally identical nodes share embeddings and lie in the majority class cluster near the decision boundary.

10.4 Discussion

ML on graph data provides a key feedback mechanism from semantic digital shadows to the represented system. In this context, graph embeddings serve as the interface between symbolic knowledge representations and data-driven prediction models. The results of this chapter demonstrate that reconstruction-based embeddings, in particular LoRE, enable this feedback loop in a manner that is consistent with the structural and semantic properties of real-world graphs. By deriving supervision from local neighborhood context rather than global edge prediction, LoRE aligns naturally with the open-world assumption and avoids the need for artificially generated negative samples. At the same time, its reconstruction-based formulation preserves structural consistency while maintaining sensitivity to local variations, thereby supporting both interpretability and discriminative performance.

Analogous to the LRP results presented in Chapter 9, these findings also highlight substantial opportunities for future work. Most notably, the current experiments are based on simulated perturbations and comparatively small node classification benchmarks. Extending this evaluation to large-scale graph trajectories such as DBpedia [17] or Wikidata [227] constitutes a natural next step. Furthermore, our analysis primarily focuses on single-layer architectures, leaving deeper and more expressive variants largely unexplored. Moreover, additional insights could be gained through systematic ablation studies. For instance, preliminary experiments indicate that incorporating δ_{LoRE} into existing graph embedding models can already yield performance improvements. Another promising direction is a more detailed investigation of the observed improvements in embedding space separability. Taken together, these findings suggest that self-reconstruction objectives not only provide a robust foundation for graph representation learning, but also open multiple avenues for future research.

Most importantly, the ability to incorporate graph updates without retraining allows embeddings to evolve together with the underlying knowledge graph, which is essential for dynamic environments such as Industry 5.0 systems. These properties make reconstruction-based graph embeddings particularly well suited for semantic predictive loops, where predictions derived from graph representations are continuously fed back into the system. In such settings, LoRE provides a robust and adaptive representation layer that enables consistent and interpretable integration of newly observed knowledge into downstream learning processes. Overall, this establishes self-reconstructive learning as a principled and practically viable paradigm for graph representation learning in real-world environments.

PART V

Conclusion and Outlook

Ich habe fertig.

Giovanni Trapattoni

Summary

This thesis investigated how the principles of Industry 5.0, i.e., human-centeredness, resilience, and sustainability, can be supported through integrated knowledge modeling approaches for collaborative industrial systems. Central to this vision is the orchestration of human-centered workflows via process modeling and execution, enabling structured coordination between human actors and AI components. We showed how process execution traces can act as semantic connectors between otherwise isolated domain knowledge, linking heterogeneous information through the processes in which they participate. This gives rise to semantically enriched representations of collaborative system behavior in the form of semantic digital shadows, effectively unifying process-centric and semantic perspectives within a common modeling framework. Building on this foundation, the results demonstrate how process-driven coordination, semantic knowledge representation, and graph-based learning can be systematically combined to model and analyze collaborative industrial systems. By linking dynamic process executions with formal domain semantics and augmenting them with interpretable and adaptive graph learning mechanisms, the presented approach enables a transition from purely descriptive digital representations towards bidirectional semantic digital representations. In this way, it contributes to the realization of human-centered, resilient, and semantically grounded industrial systems through *process-driven RDF knowledge graphs as semantic digital twins for Industry 5.0*.

Knowledge modeling requirements and architecture (C1). Building on empirical insights from the Teaming.AI project, the thesis derived five key requirements for knowledge modeling in Industry 5.0, capturing the need for explicit orchestration, temporal traceability, semantic grounding, integration of process and domain knowledge, and adaptive feedback capabilities (Part II). These requirements motivate a layered architecture that extends conventional digital representations towards semantic digital representations. In particular, the architecture provides a structured integration of process-driven dynamics and semantic domain knowledge, enabling consistent and interpretable representations across system boundaries.

Semantic integration via process-aware knowledge graphs (C2). Existing domain knowledge encoded as RDF knowledge graphs and corresponding ontologies was integrated with BPMN-based process models and execution traces (Part III), resulting in semantically grounded, time-resolved knowledge graph trajectories that constitute semantic digital shadows of collaborative industrial processes. This integration is realized through the sBPMN framework, which transforms process models and executions into RDF-based knowledge representations and establishes a semantic bridge between process-centric and ontology-centric knowledge. The framework was deployed in a real-world use case in the context of the Teaming.AI project.

Explainable graph representation learning (C3). To support transparent decision-making, the thesis introduced explainable graph representation learning on knowledge graphs (Part IV, Chapter 9). In particular, relational GNNs were combined with LRP as an established explainable AI technique to attribute predictions to relevant graph structures, enabling interpretable and trustworthy reasoning in knowledge graph-based systems. This approach establishes a direct link between learned representations and their underlying semantic structures, allowing predictions to be traced back to meaningful components of the knowledge graph.

Adaptive graph embeddings via local self-reconstruction (C4). To enable effective learning under dynamic real-world conditions, the thesis introduced local self-reconstruction as a novel GNN-specific graph embedding paradigm for dynamic knowledge graphs, implemented in the models NAVI and LORE (Part IV, Chapter 10). These approaches support robust learning for evolving graph structures, improving both flexibility and predictive performance compared to conventional GNNs, while maintaining efficiency in the presence of new data. Beyond adaptivity, they fulfill additional key requirements of incomplete and evolving domains that rely on consistent latent graph representations, which are particularly relevant in industrial contexts.

To enable both the derivation of the main contributions C1–C4 of this thesis and to provide a foundation for future research, Part I establishes the formal and conceptual groundwork. It introduces Industry 5.0 objectives and formalizes BPMN and RDF as complementary formalisms for modeling and instantiating domain knowledge. Knowledge graphs are defined as ABox components of RDF knowledge bases within a generalized graph framework, distinguishing between graphs as timeless mathematical objects and graph trajectories as their time-aware counterparts. Furthermore, the thesis reviews graph representation learning with a focus on node embeddings, differentiating between self-contained and constructive models. For the latter, a generalized formulation of GNNs is introduced, which itself constitutes a conceptual contribution of this work. Together, these foundations enable the integration of concepts developed in this thesis and provide a basis for future research beyond its scope.

Limitations and Future Work

While this thesis demonstrates how semantic digital shadows and digital twins can be realized through the integration of process modeling, semantic technologies, and graph-based learning, several limitations remain and open directions for future work emerge.

First, within the Teaming.AI project, the proposed approach was realized up to Requirement R4 by means of the sBPMN framework, enabling the transformation of workflow orchestrations into dynamic knowledge graph trajectories. Although these trajectories were successfully enriched with semantic domain knowledge, their integration with existing domain knowledge bases remains limited so far. Future work should therefore focus on tighter and more scalable integration mechanisms. In addition, the current system architecture could be extended towards more scalable triplestore solutions beyond Apache Jena TDB and recent versions of Camunda, facilitating broader industrial adoption of BPMN-based semantic digital shadows and digital twins.

Second, this work focuses on explicit representations of collaborative workflows using BPMN. While this provides a well-structured and executable formalism, the underlying principles are not limited to BPMN. Future work could investigate the transfer to alternative process modeling languages or even to less structured event streams, where process traces are reconstructed retrospectively. Moreover, the proposed approach is not restricted to manufacturing scenarios, but can be transferred to other process-driven domains such as healthcare, logistics, or business process management in general, where similar requirements for coordination, traceability, and semantic integration arise.

Third, the explainability of RGNN predictions via LRP was evaluated primarily through quantitative perturbation-based analyses. While these results provide initial evidence of meaningful explanations, future work should incorporate domain experts to assess explanations qualitatively, enabling a more comprehensive evaluation of their practical usefulness in real-world scenarios.

Fourth, the proposed graph embedding models NAVI and LORE were evaluated on general-purpose benchmark datasets of moderate size and primarily using single-layer forward passes. Future work should extend these evaluations to larger and more complex datasets, as well as deeper architectures. In addition, further ablation studies on individual model components and a more detailed analysis of embedding space properties, such as separability and structural consistency, would provide deeper insights into the behavior of the proposed methods. Furthermore, it would be of interest to investigate the incorporation of literal values into reconstruction processes, as well as to explore self-reconstruction as a regularization mechanism when training models for the prediction of such literals.

Finally, an additional promising direction concerns the data-driven identification of semantic drift within evolving graph embeddings. Preliminary work in this direction has already been carried out, but remained out of scope of this thesis [125]. Future work could investigate how such drift detection mechanisms can be integrated into dynamic knowledge graph representations, enabling the automatic identification of anomalies or structural changes that are not captured by symbolic reasoning or graph-based ML predictions. This would introduce an additional feedback mechanism and further strengthen the capabilities of semantic digital twins.

Bibliography

[1] Sunitha Abburu. A Survey on Ontology Reasoners and Comparison. *International Journal of Computer Applications*, 57(17):33–39, 2012.

[2] Francisco Antonio Lloret Abrisqueta, Antonio Guerrero González, and Roberto Zapata Martinez. Redefining Human-Machine Collaboration: Industry 5.0 to Improve Safety and Efficiency. *IEEE Latin America Transactions*, 23(8):729–735, 2025.

[3] Giovanni Acampora, Autilia Vitiello, Bruno Di Stefano, Wil Aalst, Christian Gunther, and Eric Verbeek. IEEE 1849TM: The XES standard. *IEEE Computational Intelligence Magazine*, pages 4–8, 2017.

[4] Activiti. Activiti BPM Platform. `https://www.activiti.org/`.

[5] Amr Adel. Future of industry 5.0 in society: human-centric solutions, challenges and prospective research areas. *Journal of Cloud Computing*, 11(1):40, 2022.

[6] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. Distributed Large-scale Natural Graph Factorization. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 37–48. ACM, 2013.

[7] Christos Alexakos, Manos Georgoudakis, Athanasios P Kalogeras, and Spiridon L Likothanassis. Ontology-Based Flexible Multi Agent Systems Design and Deployment for Vertical Enterprise Integration. In *IFIP International Conference on Advances in Production Management Systems*, pages 80–87. Springer, 2012.

[8] Thomas Allweyer. *BPMN 2.0: Introduction to the Standard for Business Process Modeling*. BoD–Books on Demand, 2016.

[9] Ephrem Ryan Alphonsus and Mohammad Omar Abdullah. A review on the applications of programmable logic controllers (PLCs). *Renewable and Sustainable Energy Reviews*, 60:1185–1205, 2016.

[10] Fatima Zahra Amara, Mounir Hemam, Tawous Amara, and Meriem Djezzar. Ontological modeling of ERP for Industry 4.0. In *KGSWC Workshops*, pages 161–170. CEUR-WS.org, 2022.

[11] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 1–13. ACM, 2019.

[12] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, 2008.

[13] Amina Annane, Nathalie Aussenac-Gilles, and Mouna Kamel. BBO: BPMN 2.0 based ontology for business process representation. In *20th European Conference on Knowledge Management (ECKM)*, volume 1, pages 49–59. ACI, 2019.

[14] Apache Software Foundation. Apache Kafka. `https://kafka.apache.org/`.

[15] Ali Assi, Hamid Mcheick, and Wajdi Dhifli. Data linking over RDF knowledge graphs: A survey. *Concurrency and Computation: Practice and Experience*, 32(19):e5746, 2020.

[16] Colin Atkinson and Thomas Kühne. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, 20(5):36–41, 2003.

[17] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International The Semantic Web Conference*, pages 722–735. Springer, 2007.

[18] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLOS ONE*, 10(7):e0130140, 2015.

[19] Stefan Bachhofner, Elmar Kiesling, Kabul Kurniawan, Emanuel Sallinger, and Philipp Waibel. Knowledge Graph Modularization for Cyber-Physical Production Systems. In *Proceedings of the ISWC 2021 Posters, Demos and Industry Tracks*. CEUR Workshop Proceedings, 2021.

[20] Stefan Bachhofner, Elmar Kiesling, Philipp Waibel, and Axel Polleres. Automated Process Knowledge Graph Construction from BPMN Models. In *Database and Expert Systems Applications*, pages 32–47. Springer, 2022.

[21] Stefan Bachhofner, Kabul Kurniawan, Elmar Kiesling, Kate Revoredo, and Dina Bayomie. Knowledge Graph Supported Machine Parameterization for the Injection Moulding Industry. In *Knowledge Graphs and Semantic Web*, pages 106–120. Springer, 2022.

[22] Sebastian Bader, Irlan Grangel-Gonzalez, Priyanka Nanjappa, Maria-Esther Vidal, and Maria Maleshkova. A Knowledge Graph for Industry 4.0. In *The Semantic Web*, pages 465–480. Springer, 2020.

[23] Gagan Bansal, Besmira Nushi, Ece Kamar, Walter S Lasecki, Daniel S Weld, and Eric Horvitz. Beyond Accuracy: The Role of Mental Models in Human-AI Team Performance. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 7, pages 2–11, 2019.

[24] Richa Bansal and Anil Kumar Dubey. Communication protocols used for industrial automation. In *Computational Intelligence in the Industry 4.0*, pages 73–94. CRC Press, 2024.

[25] João Barata and Ina Kayser. How will the digital twin shape the future of industry 5.0? *Technovation*, 134:103025, 2024.

[26] Alessio Baratta, Antonio Cimino, Maria Grazia Gnoni, and Francesco Longo. Human Robot Collaboration in Industry 4.0: a literature review. *Procedia Computer Science*, 217:1887–1895, 2023.

[27] Claudio D. T. Barros, Matheus R. F. Mendonça, Alex B. Vieira, and Artur Ziviani. A Survey on Embedding Dynamic Graphs. *ACM Computing Surveys (CSUR)*, 55(1):1–37, November 2021.

[28] Aneta Becker and Jarosław Becker. Dataset shift assessment measures in monitoring predictive models. *Procedia Computer Science*, 192:3391–3402, 2021.

[29] Amin Beheshti, Boualem Benatallah, Hamid Reza Motahari-Nezhad, Samira Ghodratnama, and Farhad Amouzgar. BP-SPARQL: A Query Language for Summarizing and Analyzing Big Process Data. In *Process Querying Methods*, pages 21–48. Springer, 2021.

[30] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Proceedings of the 15th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 585–591. MIT Press, 2001.

[31] Thomas Bergs, Sascha Gierlings, Thomas Auerbach, Andreas Klink, Daniel Schraknepper, and Thorsten Augspurger. The Concept of Digital Twin and Digital Shadow in Manufacturing. *Procedia CIRP*, 101:81–84, 2021.

[32] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax. Technical Report RFC 3986, Internet Engineering Task Force, 2005. `https://www.rfc-editor.org/rfc/rfc3986`.

[33] Adrien Bibal, Michael Lognoul, Alexandre de Streel, and Benoît Frénay. Legal requirements on explainability in machine learning. *Artificial Intelligence and Law*, 29(2):149–169, 2021.

[34] Russa Biswas, Lucie-Aimée Kaffee, Michael Cochez, Stefania Dumbrava, Theis E Jendal, Matteo Lissandrini, Vanessa Lopez, Eneldo Loza Mencía, Heiko Paulheim, Harald Sack, Edlira Kalemi Vakaj, and Gerard de Melo. Knowledge Graph Embeddings: Open

Challenges and Opportunities. *Transactions on Graph Data and Knowledge*, 1(1):1–32, 2023.

[35] Antoine Bordes, Sumit Chopra, and Jason Weston. Question Answering with Subgraph Embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 615–620, 2014.

[36] Antoine Bordes, Nicolas Usunier, Alberto Garcia, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, pages 2787–2795. Curran Associates, 2013.

[37] Stuart A. Boyer. *Scada: Supervisory Control And Data Acquisition*. John Wiley & Sons, 2004.

[38] Maija Breque, Lars de Nul, and Athanasios Petridis. Industry 5.0: Towards a Sustainable, Human-Centric and Resilient European Industry. EU research and innovation paper series KI-BD-20-021-EN-N, Directorate General for Research and Innovation (DG RTD) of the European Commission, 2021.

[39] Dan Brickley and Ramanathan V. Guha. RDF Schema 1.1. W3C Recommendation, World Wide Web Consortium (W3C), 2014. `https://www.w3.org/TR/rdf-schema/`.

[40] Georg Buchgeher, David Gabauer, Jorge Martinez-Gil, and Lisa Ehrlinger. Knowledge Graphs in Manufacturing and Production: A Systematic Literature Review. *IEEE Access*, 9:55537–55554, 2021.

[41] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y. Hammerla. Relational Graph Attention Networks. *arXiv preprint arXiv:1904.05811*, 2019.

[42] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[43] Diego Calvanese, Tahir Emre Kalayci, Marco Montali, and Ario Santoso. OBDA for Log Extraction in Process Mining. In *Reasoning Web. Semantic Interoperability on the Web: 13th International Summer School, Tutorial Lectures*, pages 292–345. Springer, 2017.

[44] Camunda. Camunda Platform. `https://camunda.com/`.

[45] Irene Celino, Valentina A. Carriero, Antonia Azzini, Ilaria Baroni, and Mario Scrocca. Procedural knowledge management in Industry 5.0: Challenges and opportunities for knowledge graphs. *Web Semantics: Science, Services and Agents on the World Wide Web*, 84(C):1–8, 2025.

[46] Jiaoyan Chen, Olga Mashkova, Fernando Zhapa-Camacho, Robert Hoehndorf, Yuan He, and Ian Horrocks. Ontology Embedding: A Survey of Methods, Applications and Resources. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[47] Miyoung Cho, Hanil Kim, and Pankoo Kim. A New Method for Ontology Merging based on Concept using WordNet. In *8th International Conference Advanced Communication Technology*, volume 3, pages 1573–1576, 2006.

[48] Carl Corea, Michael Fellmann, and Patrick Delfmann. Ontology-Based Process Modelling - Will We Live to See It? In *Conceptual Modeling*, pages 36–46. Springer, 2021.

[49] Simon J. Cox and Chris Little. Time Ontology in OWL. W3C Recommendation, World Wide Web Consortium (W3C), 2022. `https://www.w3.org/TR/owl-time/`.

[50] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, World Wide Web Consortium (W3C), 2014. `https://www.w3.org/TR/rdf11-concepts/`.

[51] Yuanfei Dai, Shiping Wang, Neal N. Xiong, and Wenzhong Guo. A Survey on Knowledge Graph Embedding: Approaches, Applications and Benchmarks. *Electronics*, 9(5), 2020.

[52] Dusanka Dakic, Darko Stefanovic, Ilija Cosic, Teodora Lolic, and Milovan Medojevic. Business Process Mining Application: A Literature Review. *Annals of DAAAM & Proceedings*, 29, 2018.

[53] Ayushi Dalmia, Ganesh Jawahar, and Manish Gupta. Towards Interpretation of Node Embeddings. In *The 2018 Web Conference Companion*, pages 945–952, 2018.

[54] Pietro De Giovanni. Sustainability of the Metaverse: A Transition to Industry 5.0. *Sustainability*, 15(7):6079, 2023.

[55] Antonio De Nicola and Michele Missikoff. A Lightweight Methodology for Rapid Ontology Engineering. *Communications of the ACM*, 59(3):79–86, 2016.

[56] Stefan Decker, Sergey Melnik, Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing*, 4(5):63–73, 2000.

[57] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2D Knowledge Graph Embeddings. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, 2018.

[58] Beniamino Di Martino, Antonio Esposito, Stefania Nacchia, and Salvatore Augusto Maisto. Semantic annotation of BPMN: current approaches and new methodologies. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*, pages 1–5, 2015.

[59] Mahmoud Dinar and David W Rosen. A Design for Additive Manufacturing Ontology. *Journal of Computing and Information Science in Engineering*, 17(2):021013, 2017.

[60] Xiaowen Dong, Pascal Frossard, Pierre Vandergheynst, and Nikolai Nefedov. Clustering With Multi-Layer Graphs: A Spectral Perspective. *IEEE Transactions on Signal Processing*, 60(11):5820–5831, 2012.

[61] Paul Doran, Valentina Tamma, and Luigi Iannone. Ontology Module Extraction for Ontology Reuse: An Ontology Engineering Perspective. In *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*, CIKM '07, pages 61–70. ACM, 2007.

[62] Dublin Core Metadata Initiative. DCMI Metadata Terms. `https://www.dublincore.org/specifications/dublin-core/dcmi-terms/`, 2012.

[63] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2 edition, 2018.

[64] Michel Dumontier, Pierre-Yves Vandenbussche, Ghislain A. Atemezing, and María Poveda-Villalón. Linked Open Vocabularies (LOV): A gateway to reusable semantic vocabularies on the Web. *Semantic Web*, 8(3):437–452, 2017.

[65] Martin Dürst and Michel Suignard. Internationalized Resource Identifiers (IRIs). Technical Report RFC 3987, Internet Engineering Task Force, 2005. `https://www.rfc-editor.org/rfc/rfc3987`.

[66] Lisa Ehrlinger and Wolfram Wöß, Wolfram. Towards a Definition of Knowledge Graphs. In *SEMANTiCS (Posters, Demos, SuCCESS)*, volume 1695 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[67] Orri Erling and Ivan Mikhailov. RDF Support in the Virtuoso DBMS. In *Networked Knowledge - Networked Media: Integrating Knowledge Management, New Media Technologies and Semantic Systems*, pages 7–24. Springer, 2009.

[68] EUROMAP. EUROMAP 77: Data Exchange between Injection Moulding Machines and MES, 2018. `https://www.euromap.org/euromap77`.

[69] European Parliament and Council of the European Union. Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). Official Journal of the European Union, L 2024/1689, 2024. `https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng`.

[70] Yassine Feddoul, Nicolas Ragot, Fabrice Duval, Vincent Havard, David Baudry, and Ahlem Assila. Exploring human-machine collaboration in industry: A systematic literature review of digital twin and robotics interfaced with extended reality technologies. *The International Journal of Advanced Manufacturing Technology*, 129(5):1917–1932, 2023.

[71] Michael Fellmann, Hogrebe Frank, Oliver Thomas, and Markus Nüttgens. Checking the Semantic Correctness of Process Models: An Ontology-driven Approach Using Domain Knowledge and Rules. *Enterprise Modelling and Information Systems Architectures (EMISA)*, 6:25–35, 2011.

[72] Joel E. Fischer, Chris Greenhalgh, Wenchao Jiang, Sarvapali D. Ramchurn, Feng Wu, and Tom Rodden. In-the-loop or on-the-loop? Interactional arrangements to support team coordination with a planning agent. *Concurrency and Computation: Practice and Experience*, 33(8):e4082, 2021.

[73] Flowable. Flowable Open Source BPM Platform. `https://flowable.com/`.

[74] Alejandro Germán Frank, Lucas Santos Dalenogare, and Néstor Fabián Ayala. Industry 4.0 technologies: Implementation patterns in manufacturing companies. *International Journal of Production Economics*, 210:15–26, 2019.

[75] Mikhail Galkin, Etienne Denis, Jiapeng Wu, and William L. Hamilton. NodePiece: Compositional and Parameter-Efficient Representations of Large Knowledge Graphs. In *International Conference on Learning Representations*, 2022.

[76] Antonio García-Domínguez, Mariano Marcos Bárcena, and Ivan Medina. A Comparison of BPMN 2.0 with other Notations for Manufacturing Processes. In *Advances in Manufacturing Systems (MESIC)*, volume 502 of *Key Engineering Materials*, pages 1–6. Trans Tech Publications Ltd, 2012.

[77] Julia Gastinger, Christian Meilicke, Federico Errica, Timo Sztyler, Anett Schülke, and Heiner Stuckenschmidt. History Repeats Itself: A Baseline for Temporal Knowledge Graph Forecasting. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4016–4024, 2024.

[78] Matthias Geiger, Simon Harrer, Jörg Lenhard, and Guido Wirtz. BPMN 2.0: The state of support and implementation. *Future Generation Computer Systems*, 80:250–262, 2018.

[79] Genet Asefa Gesese, Russa Biswas, Mehwish Alam, and Harald Sack. A Survey on Knowledge Graph Embeddings with Literals: Which model links better Literal-ly? *Semantic Web*, 12(4):617–647, 2021.

[80] Chiara Ghidini, Marco Rospocher, and Luciano Serafini. A formalisation of BPMN in Description Logics. *arXiv preprint arXiv:2109.10716*, 2021.

[81] Giuseppe De Giacomo and Maurizio Lenzerini. TBox and ABox reasoning in expressive description logics. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 316–327. Morgan Kaufmann Publishers Inc., 1996.

[82] Isaías González, Antonio J. Calderón, João Figueiredo, and João M. C. Sousa. A Literature Survey on Open Platform Communications (OPC) Applied to Advanced Industrial Environments. *Electronics*, 8(5), 2019.

[83] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[84] Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[85] Irlán Grangel-González. *A Knowledge Graph Based Integration Approach for Industry 4.0*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2019.

[86] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[87] Ramanathan V. Guha, Dan Brickley, and Steve Macbeth. Schema.org: Evolution of Structured Data on the Web. *Communications of the ACM*, 59(2):44–51, 2016.

[88] Shurui Gui, Hao Yuan, Jie Wang, Qicheng Lao, Kang Li, and Shuiwang Ji. FlowX: Towards Explainable Graph Neural Networks via Message Flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(7):4567–4578, 2023.

[89] Anna Lisa Guido, Pandurino Andrea, and Roberto Paiano. An Ontological Meta-model for Business Process Model and Notation (BPMN). *International Journal of Business Research and Management (IJBRM)*, 7(3):40–52, 2016.

[90] Philipp Hagedorn and Markus König. BPMN-related Ontology for Modeling the Construction Information Delivery of Linked Building Data. In *Proceedings of the 9th Linked Data in Architecture and Construction Workshop*, volume 3081, pages 91–102. CEUR-WS.org, 2021.

[91] Sang Hyu Hahn and Heiko Paulheim. RDF2vec Embeddings for Updateable Knowledge Graphs – Reuse, Don't Retrain! In *The Semantic Web: ESWC*, pages 217–222, 2024.

[92] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.*, 40:52–74, 2017.

[93] Martin Hankel and Bosch Rexroth. Reference Architectural Model Industrie 4.0 (RAMI 4.0). *ZVEI*, 2(2):4–9, 2015.

[94] Olaf Hartig. Foundations of RDF⋆ and SPARQL⋆ (An Alternative Approach to Statement-Level Metadata in RDF). *In: Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web*, 2017.

[95] Frederick Hayes-Roth, Donald A. Waterman, and Douglas B. Lenat. *Building Expert Systems*. Teknowledge Series. Addison-Wesley Publishing Company, 1983.

[96] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language: Primer. W3C Recommendation, World Wide Web Consortium (W3C), 2012. `https://www.w3.org/TR/owl2-primer/`.

[97] Thomas Hoch, Bernhard Heinzl, Gerald Czech, Maqbool Khan, Philipp Waibel, Stefan Bachhofner, Elmar Kiesling, and Bernhard Moser. Teaming.AI: enabling human-AI teaming intelligence in manufacturing. In *Proceedings of Interoperability for Enterprise Systems and Applications Workshops co-located with 11th International Conference on Interoperability for Enterprise Systems and Applications*. CEUR-WS.org, 2022.

[98] Aidan Hogan. Skolemising Blank Nodes while Preserving Isomorphism. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 430–440. International World Wide Web Conferences Steering Committee, 2015.

[99] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge Graphs. *ACM Computing Surveys (CSUR)*, 54(4), 2021.

[100] Danny Holten and Jarke J. van Wijk. A User Study on Visualizing Directed Edges in Graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2299–2308. ACM, 2009.

[101] Ian Horrocks and Peter Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.

[102] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 2004. `https://www.w3.org/Submission/SWRL/`.

[103] Yining Huang, Saadia Dhouib, Luis Palacios Medinacelli, and Jacques Malenfant. Enabling Semantic Interoperability of Asset Administration Shells Through an Ontology-based Modeling Method. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '22, pages 497–502. ACM, 2022.

[104] International Society of Automation. ANSI/ISA-95 Enterprise-Control System Integration Standard, 2010. `https://www.isa.org/standards-and-publications/isa-standards/isa-95-standard`, Parts 1–5.

[105] Eeva Järvenpää, Niko Siltala, Otto Hylli, and Minna Lanz. The development of an ontology for describing the capabilities of manufacturing resources. *Journal of Intelligent Manufacturing*, 30(2):959–978, 2019.

[106] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge Graph Embedding via Dynamic Mapping Matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 687–696, 2015.

[107] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, 2022.

[108] Catholijn M. Jonker, M. Birna van Riemsdijk, and Bas Vermeulen. Shared Mental Models. In *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, pages 132–151. Springer, 2011.

[109] Gregor Jošt, Jernej Huber, Marjan Heričko, and Gregor Polančič. An empirical investigation of intuitive understandability of process diagrams. *Computer Standards and Interfaces*, 48:90–111, 2016.

[110] Wei Ju, Yifan Wang, Yifang Qin, Zhengyang Mao, Zhiping Xiao, Junyu Luo, Junwei Yang, Yiyang Gu, Dongjie Wang, Qingqing Long, Siyu Yi, Xiao Luo, and Ming Zhang. Towards Graph Contrastive Learning: A Survey and Beyond. *arXiv preprint arXiv:2405.11868*, 2024.

[111] Henning Kagermann and Wolfgang Wahlster. Ten Years of Industrie 4.0. *Sci*, 4(3):26, 2022.

[112] Rui Kang, Junxiang Zhu, Stephen Green, and Ioannis Brilakis. Knowledge graphs for operational decision-making in industrial maintenance: A systematic review. *Expert Systems With Applications*, 299:130172, 2026.

[113] Shima Khoshraftar and Aijun An. A Survey on Graph Representation Learning Methods. *ACM Transactions on Intelligent Systems and Technology*, 15(1):1–55, 2024.

[114] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[115] Holger Knublauch and Dimitris Kontokostas. Shapes Constraint Language (SHACL). W3C Recommendation, 2017. `https://www.w3.org/TR/shacl/`.

[116] Ronald A Kohser, Peter C Collins, and J Temple Black. *DeGarmo's materials and processes in manufacturing*. John Wiley & Sons, 2024.

[117] Nikolaos Konstantinou, Dimitrios-Emmanuel Spanos, and Nikolas Mitrou. Ontology and Database Mapping: A Survey of Current Implementations and Future Directions. *Journal of Web Engineering*, pages 1–24, 2008.

[118] Franz Krause. Dynamic Knowledge Graph Embeddings via Local Embedding Reconstructions. In *The Semantic Web: ESWC Satellite Events*, pages 215–223. Springer, 2022.

[119] Franz Krause, Kabul Kurniawan, Elmar Kiesling, Heiko Paulheim, and Axel Polleres. On the Representation of Dynamic BPMN Process Executions in Knowledge Graphs. In *Knowledge Graphs and Semantic Web: 5th Iberoamerican Conference and 4th Indo-American Conference*, pages 97–105. Springer, 2023.

[120] Franz Krause and Heiko Paulheim. Explaining RGNN Predictions on Knowledge Graphs via LRP. In *Proceedings of the 22nd International Conference on Semantic Systems (SEMANTiCS)*, pages 178–195. IOS Press, 2026.

[121] Franz Krause and Heiko Paulheim. LoRE: Adaptive and Robust Graph Embeddings via Local Self-Reconstruction. In *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 650–666. Springer Nature, 2026.

[122] Franz Krause, Heiko Paulheim, Elmar Kiesling, Kabul Kurniawan, Maria Chiara Leva, Hector Diego Estrada-Lugo, Gernot Stübl, Nazim Kemal Üre, Javier Dominguez-Ledo, Pedro Demolder, Hans Gaux, Bernhard Heinzl, Thomas Hoch, Jorge Martinez-Gil, Agastya Silvina, and Bernhard A. Moser. Managing human-AI collaborations within Industry 5.0 scenarios via knowledge graphs: key challenges and lessons learned. *Frontiers in Artificial Intelligence*, 7:1247712, 2024.

[123] Franz Krause, Heiko Paulheim, Amparo Vázquez, and Alejandro Espert. sBPMN: A Framework for Capturing Business Process Semantics in Dynamic Knowledge Graphs. In *Proceedings of the 13th Knowledge Capture Conference 2025*, K-CAP '25, pages 156–164. ACM, 2025.

[124] Franz Krause, Tobias Weller, and Heiko Paulheim. On a Generalized Framework for Time-Aware Knowledge Graphs. In *Proceedings of the 18th International Conference on Semantic Systems (SEMANTiCS)*, volume 55, pages 69–74, 2022.

[125] Franz Krause, Xi Yan, Baptiste Darnala, and Michel Dumontier. On the Combination of Event Calculus and Empirical Semantic Drifts. In *SEMMES 2023-1st International Workshop on Semantic Methods for Events and Stories*, volume 3443, 2023.

[126] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018.

[127] Ryoji Kubo and Djellel Difallah. RAW-Explainer: Post-hoc Explanations of Graph Neural Networks on Knowledge Graphs. *arXiv preprint arXiv:2506.12558*, 2025.

[128] Vitus SW Lam. A Precise Execution Semantics for BPMN. *IAENG International Journal of Computer Science*, 39(1):20–33, 2012.

[129] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, 2014.

[130] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[131] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[132] Stefan-Helmut Leitner and Wolfgang Mahnke. OPC UA - Service-oriented Architecture for Industrial Applications. *Softwaretechnik-Trends*, 26(4), 2006.

[133] Severin Lemaignan, Ali Siadat, J-Y Dantan, and Anatoli Semenenko. MASON: A Proposal For An Ontology Of Manufacturing Domain. In *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06)*, pages 195–200. IEEE, 2006.

[134] Christos Liakos, Aimilios Christos Panagopoulos, and Panagiotis Karkazis. Smart Business Processing in Industry 4.0. In *Proceedings of the 26th Pan-Hellenic Conference on Informatics*, PCI '22, pages 32–37. ACM, 2023.

[135] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2181–2187, 2015.

[136] Bo Liu, Thomas Glock, Victor Pazmino Betancourt, Matthias Kern, Eric Sax, and Jürgen Becker. Model Driven Development Process for a Service-oriented Industry 4.0 System. In *9th International Conference on Industrial Technology and Management (ICITM)*, pages 78–83. IEEE, 2020.

[137] Ninghao Liu, Qizhang Feng, and Xia Hu. Interpretability in Graph Neural Networks. In *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 121–147. Springer, 2022.

[138] Shimin Liu, Yuqian Lu, Xingwang Shen, and Jinsong Bao. A digital thread-driven distributed collaboration mechanism between digital twin manufacturing units. *Journal of Manufacturing Systems*, 68:145–159, 2023.

[139] Yang Lu. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6:1–10, 2017.

[140] Scott M. Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems*, 30, 2017.

[141] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized Explainer for Graph Neural Network. In *Advances in Neural Information Processing Systems*, volume 33, pages 19620–19631. Curran Associates, 2020.

[142] Shuai Ma, Ying Lei, Xinru Wang, Chengbo Zheng, Chuhan Shi, Ming Yin, and Xiaojuan Ma. Who Should I Trust: AI or Myself? Leveraging Human and AI Correctness Likelihood to Promote Appropriate Trust in AI-Assisted Decision-Making. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23. ACM, 2023.

[143] Aleksander Malinowski and Hao Yu. Comparison of embedded system design for industrial applications. *IEEE Transactions on Industrial Informatics*, 7(2):244–254, 2011.

[144] Soujanya Mantravadi and Charles Møller. An Overview of Next-generation Manufacturing Execution Systems: How important is MES for Industry 4.0? *Procedia Manufacturing*, 30:588–595, 2019.

[145] Heidy M. Marin-Castro and Edgar Tello-Leal. Event Log Preprocessing for Process Mining: A Review. *Applied Sciences*, 11(22), 2021.

[146] Jorge Martinez-Gil, Thomas Hoch, Mario Pichler, Bernhard Heinzl, Bernhard Moser, Kabul Kurniawan, Elmar Kiesling, and Franz Krause. Examining the Adoption of Knowledge Graphs in the Manufacturing Industry: A Comprehensive Review. In *Artificial Intelligence in Manufacturing*, pages 55–70. Springer, 2024.

[147] César Martínez-Olvera. The role of manufacturing efficiency in the achievement of sustainable mass customization 4.0. *Production & Manufacturing Research*, 10(1):132–159, 2022.

[148] Aeshita Mathur, Ameesha Dabas, and Nikhil Sharma. Evolution From Industry 1.0 to Industry 5.0. In *4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, pages 1390–1394, 2022.

[149] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119. Curran Associates, 2013.

[150] László Monostori. Cyber-physical Production Systems: Roots, Expectations and R&D Challenges. *Procedia CIRP*, 17:9–13, 2014.

[151] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise Relevance Propagation (LRP): An Overview. *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 193–209, 2019.

[152] Bernhard Mörzinger. *Accessing Manufacturing Data through Virtual Knowledge Graphs: On the Value and Semantic Peculiarities of Time Series Data*. PhD thesis, Technische Universität Wien, 2019.

[153] Sergi Nadal, Oscar Romero, Alberto Abelló, Panos Vassiliadis, and Stijn Vansummeren. An Integration-Oriented Ontology to Govern Evolution in Big Data Ecosystems. *Information Systems*, 79:3–19, 2019.

[154] Saeid Nahavandi. Trusted Autonomy Between Humans and Robots: Toward Human-on-the-Loop in Robotics and Autonomous Systems. *IEEE Systems, Man, and Cybernetics Magazine*, 3(1):10–17, 2017.

[155] Saeid Nahavandi. Industry 5.0 - A human-centric solution. *Sustainability*, 11(16):4371, 2019.

[156] National Academies of Sciences, Engineering, and Medicine. *Human-AI Teaming: State-of-the-art and Research Needs*. Consensus study report. National Academies Press, 2022.

[157] Christine Natschläger. Towards a BPMN 2.0 Ontology. In *Business Process Model and Notation*, pages 1–15. Springer, 2011.

[158] Patrick W. Neumann, Sven Winkelhaus, Eric H. Grosse, and Christoph H. Glock. Industry 4.0 and the human factor – A systems framework and analysis methodology for successful development. *International Journal of Production Economics*, 233:107992, 2021.

[159] Mark Newman. *Networks*. Oxford University Press, 2018.

[160] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[161] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A Three-Way Model for Collective Learning on Multi-Relational Data. In *Proceedings of the 28th International Conference on Machine Learning*, pages 809–816. Omnipress, 2011.

[162] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale Knowledge Graphs: Lessons and Challenges. *Communications of the ACM*, 62(8):36–43, 2019.

[163] Oyejide Timothy Odofin, Abraham Ayodeji Abayomi, Abel Chukwuemeke Uzoka, Bolaji Iyanu Adekunle, Oluwademilade Aderemi Agboola, and Samuel Owoade. Designing Event-Driven Architecture for Financial Systems Using Kafka, Camunda BPM, and Process Engines. *International Journal of Scientific Research in Science, Engineering and Technology*, 11(4):427–441, 2024.

[164] OMG. *Business Process Model and Notation (BPMN), Version 2.0*. Object Management Group, January 2011. `http://www.omg.org/spec/BPMN/2.0`.

[165] Ontotext. GraphDB. `https://www.ontotext.com/products/graphdb/`.

[166] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1105–1114, 2016.

[167] Florian Patzer, Friedrich Volz, Thomas Usländer, Immanuel Blöcher, and Jürgen Beyerer. The industrie 4.0 asset administration shell as information source for security analysis. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 420–427. IEEE, 2019.

[168] Heiko Paulheim. Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic Web*, 8(3):489–508, 2017.

[169] Ricardo Silva Peres, Xiaodong Jia, Jay Lee, Keyi Sun, Armando Walter Colombo, and Jose Barata. Industrial Artificial Intelligence in Industry 4.0 - Systematic Review, Challenges and Outlook. *IEEE Access*, 8:220121–220139, 2020.

[170] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710. ACM, 2014.

[171] Roland Petrasch and Roman Hentschke. Process Modeling for Industry 4.0 Applications: Towards an Industry 4.0 Process Modeling Language and Method. In *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–5, 2016.

[172] Simone Piaggesi, Megha Khosla, André Panisson, and Avishek Anand. DINE: Dimensional Interpretability of Node Embeddings. *IEEE Transactions on Knowledge and Data Engineering*, 36(12):7986–7997, 2024.

[173] Simone Piaggesi, André Panisson, and Megha Khosla. Disentangled and Self-Explainable Node Representation Learning. *arXiv preprint arXiv:2410.21043*, 2024.

[174] Christoph Piller. Describing and Analyzing AI Agents with the Tools We Already Trust: A Comparative Study of PASS and BPMN. In *International Conference on Subject-Oriented Business Process Management*, pages 115–136. Springer, 2025.

[175] Axel Polleres, Aidan Hogan, Renaud Delbru, and Jürgen Umbrich. RDFS and OWL Reasoning for Linked Data. In *Reasoning Web. Semantic Technologies for Intelligent Data (9th International Summer School)*, pages 91–149. Springer, 2013.

[176] Jan Portisch, Nicolas Heist, and Heiko Paulheim. Knowledge graph embedding for data mining vs. knowledge graph embedding for link prediction – two sides of the same coin? *Semantic Web*, 13(3):399–422, 2022.

[177] Leo Poss and Stefan Schönig. Location-aware business process modeling and execution: Location-aware business process modeling and execution. *Softw. Syst. Model.*, 24(1):37–67, 2024.

[178] Eric Prud'hommeaux and Gavin Carothers. RDF 1.1 Turtle: Terse RDF Triple Language. W3C Recommendation, World Wide Web Consortium (W3C), 2014. `https://www.w3.org/TR/turtle/`.

[179] Eric Prud'hommeaux, Steve Harris, and Andy Seaborne. SPARQL 1.1 Query Language. Technical report, W3C, 2013. `http://www.w3.org/TR/sparql11-query`.

[180] Theofanis P. Raptis, Andrea Passarella, and Marco Conti. Data Management in Industry 4.0: State of the Art and Open Challenges. *IEEE Access*, 7:97052–97093, 2019.

[181] Red Hat. jBPM: Open Source Business Automation Platform. `https://www.jbpm.org/`.

[182] Zobia Rehman, Claudiu Vasile Kifor, Farhana Jabeen, Sheneela Naz, and Muhammad Waqar. Automatic Acquisition of Failure Mode and Effect Analysis Ontology for Sustainable Risk Management. *Sustainability*, 12(23), 2020.

[183] Marco S Reis and Geert Gins. Industrial Process Monitoring in the Big Data/Industry 4.0 Era: from Detection, to Diagnosis, to Prognosis. *Processes*, 5(3), 2017.

[184] Raven T Reisch, Tobias Hauser, Benjamin Lutz, Alexandros Tsakpinis, Dominik Winter, Tobias Kamps, and Alois Knoll. Context awareness in process monitoring of additive manufacturing using a digital twin. *The International Journal of Advanced Manufacturing Technology*, 119(5):3483–3500, 2022.

[185] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

[186] Dennis M. Riehle, Sven Jannaber, Patrick Delfmann, Oliver Thomas, and Jörg Becker. Automatically Annotating Business Process Models with Ontology Concepts at Design-Time. In *International Conference on Conceptual Modeling*, pages 177–186. Springer, 2017.

[187] Petar Ristoski, Gerben Klaas Dirk de Vries, and Heiko Paulheim. A Collection of Benchmark Datasets for Systematic Evaluations of Machine Learning on the Semantic Web. In *The Semantic Web – ISWC*, pages 186–194, 2016.

[188] Petar Ristoski and Heiko Paulheim. RDF2Vec: RDF Graph Embeddings for Data Mining. In *The Semantic Web – ISWC*, pages 498–514, 2016.

[189] Blaž Rodič. Industry 4.0 and the New Simulation Modelling Paradigm. *Organizacija*, 50(3):193–207, 2017.

[190] Marco Rospocher, Chiara Ghidini, and Luciano Serafini. An Ontology for the Business Process Modelling Notation. In *Formal Ontology in Information Systems*, volume 267 of *Frontiers in Artificial Intelligence and Applications*, pages 133–146. IOS Press, 2014.

[191] Alberto Rossi, Matteo Tiezzi, Giovanna Maria Dimitri, Monica Bianchini, Marco Maggini, and Franco Scarselli. Inductive–Transductive Learning with Graph Neural Networks. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 201–212. Springer, 2018.

[192] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge Graph Embedding for Link Prediction: A Comparative Analysis. *ACM Trans. Knowl. Discov. Data*, 15(2):1–49, 2021.

[193] Edna Ruckhaus, John Toledo, Edgar Alexis Martínez Sarmiento, and Oscar Corcho. Lessons Learned from the Combined Development of OWL and SHACL. In *Proceedings of the 13th Knowledge Capture Conference 2025*, K-CAP '25, pages 1–4. ACM, 2025.

[194] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[195] Klaus-Dieter Schewe. Horizontal and Vertical Business Process Model Integration. In *Database and Expert Systems Applications*, pages 1–3. Springer, 2013.

[196] Gunar Schirner, Deniz Erdogmus, Kaushik Chowdhury, and Taskin Padir. The Future of Human-in-the-Loop Cyber-Physical Systems. *Computer*, 46(1):36–45, 2013.

[197] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In *The Semantic Web: 15th International Conference*, pages 593–607. Springer, 2018.

[198] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, Kristof T. Schutt, Klaus-Robert Muller, and Gregoire Montavon. Higher-Order Explanations of Graph Neural Networks via Relevant Walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7581–7596, 2022.

[199] Zohair Shafi, Ayan Chatterjee, and Tina Eliassi-Rad. Generating Human Understandable Explanations for Node Embeddings. *arXiv preprint arXiv:2406.07642*, 2024.

[200] Robert D. C. Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC)*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[201] Cogan Shimizu, Karl Hammar, and Pascal Hitzler. Modular Ontology Modeling. *Semantic Web*, 14(3):459–489, 2023.

[202] Cogan Shimizu, Quinn Hirt, and Pascal Hitzler. MODL: A Modular Ontology Design Library. In *Proceedings of the 10th Workshop on Ontology Design and Patterns (WOP 2019) co-located with 18th International Semantic Web Conference (ISWC 2019)*, volume 2459 of *CEUR Workshop Proceedings*, pages 47–58. CEUR-WS.org, 2019.

[203] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A Practical OWL-DL Reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[204] Tarek Skouti, Ronny Seiger, Frank J. Furrer, and Susanne Strahringer. RBPMN: the value of roles for business process modeling. *Software and Systems Modeling*, 23(6):1375–1406, 2024.

[205] Dipa Soni and Ashwin Makwana. A Survey on MQTT: A Protocol of Internet of Things. In *International Conference on Telecommunication, Power Analysis and Computing Techniques*, volume 20, 2017.

[206] Saul Stahl. The embeddings of a graph — A survey. *Journal of Graph Theory*, 2(4):275–298, 1978.

[207] Masashi Sugiyama and Motoaki Kawanabe. Introduction and Problem Formulation. In *Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation*. The MIT Press, 2012.

[208] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *7th International Conference on Learning Representations*, 2019.

[209] York Sure, Steffen Staab, and Rudi Studer. On-To-Knowledge Methodology (OTKM). In *Handbook on Ontologies*, pages 117–132. Springer, 2004.

[210] Ruben Taelman, Joachim Van Herwegen, Miel Vander Sande, and Ruben Verborgh. Comunica: A Modular SPARQL Query Engine for the Web. In *The Semantic Web – ISWC*, pages 239–255. Springer, 2018.

[211] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[212] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Non-Parametric Estimation of Multiple Embeddings for Link Prediction on Dynamic Knowledge Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1):1243–1249, 2017.

[213] The Apache Software Foundation. Apache Jena. `https://jena.apache.org/`, 2025.

[214] Christoph S. Thomalla. Interoperability in manufacturing by semantic integration. In *International Conference on System Science, Engineering Design and Manufacturing Informatization*, volume 2, pages 146–149, 2011.

[215] Oliver Thomas and Michael Fellmann. Semantic Process Modeling: Design and Implementation of an Ontology-based Representation of Business Processes. *Business & Information Systems Engineering*, 1:438–451, 2009.

[216] Basant Tomar and Narendra Kumar. PLC and SCADA based industrial automated system. In *IEEE International Conference for Innovation in Technology (INOCON)*, pages 1–5. IEEE, 2020.

[217] Attila Tóth, László Nagy, Roderick Kennedy, Belej Bohuš, János Abonyi, and Tamás Ruppert. The human-centric Industry 5.0 collaboration architecture. *MethodsX*, 11:102260, 2023.

[218] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66. Association for Computational Linguistics, 2015.

[219] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. Complex Embeddings for Simple Link Prediction. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 2071–2080, 2016.

[220] Oktay Turetken, Tessa Rompen, Irene Vanderfeesten, Ahmet Dikici, and Jan van Moll. The Effect of Modularity Representation and Presentation Medium on the Understandability of Business Process Models in BPMN. In *International Conference on Business Process Management*, pages 289–307. Springer, 2016.

[221] Thomas H-J Uhlemann, Christian Lehmann, and Rolf Steinhilper. The Digital Twin: Realizing the Cyber-Physical Production System for Industry 4.0. *Procedia CIRP*, 61:335–340, 2017.

[222] Zahid Usman, Robert IM Young, Nitishal Chungoora, Claire Palmer, Keith Case, and Jennifer A Harding. Towards a formal manufacturing reference ontology. *International Journal of Production Research*, 51(22):6553–6572, 2013.

[223] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Pratim Talukdar. Composition-based Multi-Relational Graph Convolutional Networks. *arXiv preprint arXiv:1911.03082*, 2019.

[224] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.

[225] Deepak Verma. Industry 5.0: A Human-Centric and Sustainable Approach to Industrial Development. *International Journal of Social Relevance & Concern*, 12(5):17–21, 2024.

[226] Valeria Villani, Marco Picone, Marco Mamei, and Lorenzo Sabattini. A Digital Twin Driven Human-Centric Ecosystem for Industry 5.0. *IEEE Transactions on Automation Science and Engineering*, 22:11291–11303, 2024.

[227] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[228] Lidong Wang and Guanghui Wang. Big Data in Cyber-Physical Systems, Digital Manufacturing and Industry 4.0. *International Journal of Engineering and Manufacturing (IJEM)*, 6(4):1–8, 2016.

[229] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[230] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge Graph Embedding by Translating on Hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1112–1119. AAAI Press, 2014.

[231] Douglas West. *Introduction to Graph Theory*. Prentice Hall, 1996.

[232] Bob J. Wielinga and Guus Schreiber. Conceptual modelling of large reusable knowledge bases. In *Management and Processing of Complex Data Structures*, pages 181–200. Springer, 1994.

[233] Tianxing Wu, Arijit Khan, Melvin Yong, Guilin Qi, and Meng Wang. Efficiently embedding dynamic knowledge graphs. *Knowledge-Based Systems*, 250:109–124, 2022.

[234] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

[235] Guohui Xiao, Davide Lanti, Roman Kontchakov, Sarah Komla-Ebri, Elem Güzel-Kalaycı, Linfang Ding, Julien Corman, Benjamin Cogrel, Diego Calvanese, and Elena Botoeva. The Virtual Knowledge Graph System Ontop. In *The Semantic Web – ISWC*, pages 259–277. Springer, 2020.

[236] Wenhao Xie, Shuxin Wang, Yanzhi Wei, Yonglin Zhao, and Xianghua Fu. Dynamic Knowledge Graph Completion with Jointly Structural and Textual Dependency. In *Algorithms and Architectures for Parallel Processing*, pages 432–448, 2020.

[237] Chengjin Xu, Yung-Yu Chen, Mojtaba Nayyeri, and Jens Lehmann. Temporal Knowledge Graph Completion using a Linear Temporal Regularizer and Multivector Embeddings. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2569–2578. Association for Computational Linguistics, 2021.

[238] Xun Xu. From cloud computing to cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, 28(1):75–86, 2012.

[239] Bingcong Xue and Lei Zou. Knowledge Graph Quality Management: A Comprehensive Survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(5):4969–4988, 2022.

[240] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *3rd International Conference on Learning Representations, ICLR*, 2015.

[241] Jaewon Yang and Jure Leskovec. Defining and Evaluating Network Communitiesbased on Ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, pages 1–8, 2012.

[242] Leshanshui Yang, Clément Chatelain, and Sébastien Adam. Dynamic Graph Representation Learning With Neural Networks: A Survey. *IEEE Access*, 12:43460–43484, 2024.

[243] Yuhao Yang, Chao Huang, Lianghao Xia, and Chenliang Li. Knowledge Graph Contrastive Learning for Recommendation. In *Proceedings of the 45th International SIGIR Conference on Research and Development in Information Retrieval*, pages 1434–1443. ACM, 2022.

[244] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding Negative Sampling in Graph Representation Learning. In *Proceedings of the*

26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '20, pages 1666–1676. ACM, 2020.

[245] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 40–48, 2016.

[246] Zi Ye, Yogan Jaya Kumar, Goh Ong Sing, Fengyan Song, and Junsong Wang. A Comprehensive Survey of Graph Neural Networks for Knowledge Graphs. *IEEE Access*, 10:75729–75741, 2022.

[247] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. GNNExplainer: Generating Explanations for Graph Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, 2019.

[248] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph Contrastive Learning with Augmentations. *Advances in Neural Information Processing Systems*, 33:5812–5823, 2020.

[249] Chenhan Yuan and Hoda Eldardiry. GradXKG: A Universal Explain-per-use Temporal Knowledge Graph Explainer. *arXiv preprint arXiv:2310.04889*, 2023.

[250] Gongsheng Yuan, Jiaheng Lu, Zhengtong Yan, and Sai Wu. A Survey on Mapping Semi-Structured Data and Graph Data to Relational Data. *ACM Computing*, 55(10):1–38, 2023.

[251] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. XGNN: Towards Model-Level Explanations of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 430–438, 2020.

[252] Amrapali Zaveri, Dimitris Kontokostas, Sebastian Hellmann, Jürgen Umbrich, Michael Färber, Frederic Bartscherer, Carsten Menne, Achim Rettinger, Amrapali Zaveri, Dimitris Kontokostas, Sebastian Hellmann, and Jürgen Umbrich. Linked data quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO. *Semantic Web*, 9(1):77–129, 2018.

[253] Yue Zhang, David Defazio, and Arti Ramesh. RelEx: A Model-Agnostic Relational Model Explainer. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 1042–1049, 2021.

[254] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[255] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep Graph Contrastive Representation Learning. *arXiv preprint arXiv:2006.04131*, 2020.

[256] Marina C. Zizic, Marko Mladineo, Nikola Gjeldum, and Luka Celent. From industry 4.0 towards industry 5.0: A review and analysis of paradigm shift for the people, organization and technology. *Energies*, 15(14):5221, 2022.