Semantic-enhanced Programmable Knowledge Graph (SPG)

White paper（v1.0）

——The new generation industrial-grade knowledge semantic framework and engine

Utilize SPG's domain type constraints and the fusion representation of facts and logic to automatically complete and increase the semantic relations between knowledge elements, promoting the explicit densification of sparse relationships between knowledge elements. Using the SPG framework can accelerate the knowledge integration of massive enterprise data, and seamlessly connect AI applications through its knowledge symbolic representation and programmability capabilities.

Ant Group × OpenKG co-produced

August 2023

Copyright Notice

Writing instructions

Lead Writing Unit: Ant Technology Group Co., Ltd.

Participating Writing Units: Tongji University, Tianjin University, Hundsun Electronics Co., Ltd., Zhejiang Chuanglin Technology Co., Ltd., Daguan Data Co., Ltd., Haiyizhi Information Technology (Nanjing) Co., Ltd., Zhejiang University, Zhijiang Laboratory, Institute of Computing Technology, Chinese Academy of Sciences.

Writing team member

Ant Technology Group Co., Ltd.: Lei Liang, Zhiqiang Zhang, Jin Peng, Peilong Zhao, Zhihui Guo, Yuxiao He, Lin Yuan

Tongji University: Haofen Wang

Tianjin University: Xin Wang, Xiang Wang

Hundsun Electronics Co., Ltd.: Shuo Bai, Jiao Chen

Zhejiang Chuanglin Technology Co., Ltd.: Yan Zhou, Chen Zhang

Daguan Data Co., Ltd.: Wenguang Wang, Mengjie He

Haiyizhi Information Technology (Nanjing) Co., Ltd.: Fanghuai Hu, Jun Ding

Zhejiang University: Huajun Chen, Wen Zhang

Zhijiang Laboratory: Heng Zhang

Institute of Computing Technology, Chinese Academy of Sciences: Long Bai

Recommendations

The knowledge graph is an extension of early expert systems and semantic web technology. Since Google applied it to the search recommendation field in 2012, knowledge graph technology has been widely adopted in various domains. However, the semantic representation and technical framework of knowledge graphs have not made significant progress for a long time, leading to increased costs and complexities in constructing knowledge graphs across different fields. I am pleased to learn about the collaboration between Ant Group and OpenKG, which leverages Ant Group's extensive industrial experience in knowledge graphs to propose a knowledge semantic framework called SPG. SPG is compatible with big data systems and AI technology systems, and it offers programmability, framework characteristics, and strong cross-scenario migration capabilities. This accelerates the industrialization of knowledge graphs and represents a breakthrough in the knowledge graph technology framework. Since the end of 2022, large language models (LLMs) such as ChatGPT and GPT4 have triggered a new wave of artificial intelligence. However, current LLMs still face challenges such as knowledge illusion, complex reasoning fallacies, and high computational costs. As a complement to LLMs, the technical system of symbolic knowledge graphs enables controlled content understanding and generation. It provides support for accurate domain knowledge and complex reasoning capabilities, facilitating the implementation of LLMs across different industries. We anticipate that SPG will become an important technology in the field of knowledge graphs. Through Ant Group's continuous refinement across diverse scenarios and its collaboration with the OpenKG community, it will drive industry development in the field of knowledge graphs, promote knowledge interconnection across different domains, and enable the controlled and low-cost implementation of LLMs and knowledge graph technology.

——Juanzi Li, Director and professor of the Knowledge Intelligence Research Center of the Institute of Artificial Intelligence of Tsinghua University

As a symbolic knowledge representation system, the knowledge graph possesses capabilities such as high-order semantics, rigorous structure, and complex reasoning. In the era of rapid development of large language models (LLMs), there is a rich interactive relationship between knowledge graphs and LLMs. On one hand, LLMs provide a powerful tool for constructing large-scale knowledge graphs at a low cost. Whether leveraging LLMs to build a world knowledge graph beyond the existing scale by 1-2 orders of magnitude has become an intriguing research question. On the other hand, the knowledge graph, with its high-quality and interpretable knowledge representation and reasoning capabilities, offers a potential exploration direction for addressing the idealistic challenges of LLMs.

Traditional knowledge semantic frameworks like RDF/OWL and LPG have significant limitations in knowledge management and struggle to support the construction and application of knowledge graphs in the

era of LLMs. SPG, derived from the extensive business practices of the Ant Knowledge Graph team, effectively addresses the deficiencies of RDF/OWL and LPG in knowledge management. It represents a new generation of knowledge semantic framework that builds upon the engine architecture by leveraging SPG's semantic specifications and programmable paradigms. SPG facilitates efficient graph construction in various domains and enables semantic alignment of knowledge across different fields.

The future development of the knowledge graph relies heavily on an active community. Ant Group will continue collaborating with the OpenKG community in areas such as SPG, the construction and evolution of the world knowledge graph, to accelerate its technological maturity and industrial implementation. We also welcome colleagues from industry and academia to actively participate in co-creation, jointly promoting the maturity and progress of knowledge graph technology, facilitating knowledge exchange and circulation between different fields, and building a new generation of AI technology systems driven by the controllable implementation of knowledge graph + LLMs.

—— Wenguang Chen, President of Ant Group Technology Research Institute

Ant Group possesses diversified business scenarios and massive amount of domain data. The SPG framework has been developed based on the extensive practical experience of Ant Group in knowledge graphs. The characteristics of Ant Group's business data, such as multi-source heterogeneity, temporal dynamics, and complex correlations, provide an excellent environment for constructing large-scale knowledge graphs. The SPG framework, by abstractly addressing multi-business and multi-scenario challenges, defines a new generation enterprise-level knowledge management paradigm with strong adaptability for enterprise-level applications. Through data intellectualization, the SPG framework transforms massive data into knowledge and solves high-dimensional business problems through methods like complex pattern calculation and graph learning reasoning. The SPG framework presents innovative possibilities for efficient domain knowledge graph construction and cross-domain knowledge graph semantic alignment. Furthermore, in the era of large language models (LLMs), the SPG framework, along with the domain knowledge graph built upon it, enables controlled implementation of LLMs in various business fields such as security risk control, micro credit, and digital finance. Through collaboration with OpenKG, we aim to accelerate the enhancement of the SPG framework by harnessing the power of the community and industry, promote the maturity of knowledge graph technology, and advance industry development. Throughout this journey, we welcome the active participation of all colleagues in co-creation, jointly driving the development and innovation of knowledge graph technology, and realizing controllable AI driven by both LLMs and knowledge graphs, ultimately expediting industry implementation.

—— Jun Zhou, Head of Machine Intelligence Department and Researcher at Ant Group.

Preface

As a method of modeling and managing data, knowledge graph has played a crucial role in the digitalization of enterprises. However, with the increasing demand for knowledge graph, traditional knowledge graph technology is encountering several challenges. Through extensive research and practical experience, Ant Group has identified limitations in traditional knowledge graph technology when dealing with complex business scenarios and large-scale data. For instance, the construction of knowledge graphs requires a unified industrial-level knowledge modeling framework that can adapt to diverse fields. The reasoning capabilities of knowledge graphs need to be more efficient and interpretable. Furthermore, the construction and reasoning processes of knowledge graph require enhanced programmability and cross-scenario transferability.

Lei Liang, as the Head of Ant Group's Knowledge Engine, led the team in developing an industrial-level knowledge graph semantic framework called SPG (Semantic-enhanced Programmable Graph). During his initial introduction of the idea and SPG to me, I was pleasantly surprised to find that we were solving similar challenges at the same time. What was initially planned as a one-hour meeting gradually evolved into a morning of in-depth discussions. Subsequently, I felt increasingly compelled to integrate our efforts in expanding SPG to address new opportunities and needs in the era of large language models (LLMs), while also open-sourcing this comprehensive and innovative knowledge graph platform to the entire community. When I shared this idea with Lei Liang, both he and Ant Group provided strong support. We actively promoted collaboration between the various R&D teams of OpenKG and the Ant Knowledge Graph team, ultimately forming a virtual team to facilitate bi-weekly communication, design planning, and research and development work.

The SPG framework is based on the property graph, combining the semantic nature of RDF/OWL and the structural nature of LPG. It offers the advantages of semantic simplicity and compatibility with big data. Through the SPG framework, we can achieve automatic layering of knowledge from dynamic to static, ensure the uniqueness of knowledge within domains, and define dependencies between knowledge. Furthermore, the SPG framework provides a programmable paradigm, supporting the rapid construction of new domain knowledge graph and cross-scenario migration. It has wide-ranging applications in solving typical problems and scenarios. In the context of risk mining knowledge graph and enterprise causal knowledge graph, the SPG framework can assist enterprises in identifying and addressing illicit activities, enhancing risk prevention and control capabilities. In terms of knowledge reasoning and intelligent question answering, the SPG framework can provide more accurate and interpretable inference results, improving user experience and decision-making effectiveness.

In this whitepaper, we will provide a detailed introduction to the design principles, technical modules, and application cases of the SPG framework. We hope that through this whitepaper, readers will have a comprehensive understanding of the SPG framework and be inspired to engage in further discussion and collaboration. We believe that the SPG framework will provide stronger and more flexible support for

enterprise digitalization, driving the development and application of knowledge graph technology. Lastly, we would like to express our gratitude for your attention and support of this whitepaper. If you have any questions or suggestions regarding the SPG framework or knowledge graph technology, please feel free to contact us. Let's work together to create a future for the next generation of industrial-level knowledge graph!

Thank you!

——Haofen Wang, Lei Liang, and the SPG team

Contents

Chapter 1 From Data-Driven to Knowledge-Driven: Enterprises Deepen Competitive Advantages with Evolving Knowledge Graph Technology

In the process of digitalization, enterprises have accumulated massive amounts of data. This includes both unstructured and semi-structured data such as text, images, videos, and audio, as well as structured data such as user behavior, product orders, services, and merchant profiles. Additionally, there are professional knowledge bases and industry data obtained from external channels to support business development. Faced with this vast amount of data, enterprises need to continuously create value for users while ensuring efficient management and risk control. This places high demands on the digital infrastructure of enterprises and provides diverse scenarios for AI technologies such as Knowledge Graphs (KGs) and Large Language Models (LLMs). It also brings new opportunities and challenges. AI technologies can help enterprises quickly discover patterns, analyze trends, and predict the future from massive amounts of data. This enables enterprises to better understand customer needs, optimize product design, and improve production efficiency. AI can also assist in intelligent risk management and anti-fraud detection. However, enterprises often face challenges such as data silos, data consistency conflicts, and data duplication due to business development and departmental differences. To improve data utilization efficiency, it is necessary to strengthen data management and application, and increase the utilization and value of data. Enterprises need to establish user-friendly management paradigms, define data structures based on business models, clarify semantics, eliminate ambiguities, and identify errors. They also strive to establish mechanisms for connecting data silos, enabling cross-system and cross-department data sharing and collaborative utilization. In addition, enterprises need to establish standardized data and service agreements to achieve efficient data collaboration, expert experience collaboration, and human-machine collaboration. Efficient data management mechanisms, standardized data modeling, ambiguity elimination to enhance consistency, and data silo connection are key issues faced by enterprises in their digitalization journey. More efficient organization and management of enterprise data and the utilization of AI technologies to fully explore data value have become the core driving forces for future enterprise growth.

1.1 The Expectations of Knowledge Graph as the Next-generation Enterprise Knowledge Management Paradigm

As an important branch of AI technology, Knowledge Graph has gained increasing popularity due to its ability to help enterprises organize and manage knowledge data more effectively. By semantically modeling and constructing a knowledge graph, enterprises can gain a better understanding of the relationships between data, uncover hidden values, and make informed decisions. In fact, Gartner predicted in 2021 that Data Fabric, based on Knowledge Graph technology, would become the next-generation data architecture. Neo4j and Cambridge Semantic have also released whitepapers introducing a new generation

of knowledge management paradigms based on Knowledge Graph. Neo4j considers a Knowledge Graph as a semantically enhanced graph, leveraging certain paradigms to semantically enhance the graph and discover more implicit clues from multidimensional relations. Cambridge Semantic believes that Knowledge Graph is a killer application for Data Fabric. It models entities, facts, concepts, and their relations in the real world, providing consistent modeling capabilities for different roles. It enables more accurate representation of organizational data and effectively connects data sources, graph storage, and downstream AI/BI tasks, breaking down data silos and enabling on-demand integration, loading, and seamless connection. Since 2018, Knowledge Graph applications in enterprise digitalization have been widely adopted in various vertical domains such as finance, healthcare, public security, and energy [1, 2, 3]. According to a report [4], the market size of Knowledge Graph in China is expected to reach 29 billion RMB by 2026, with finance and public security being the main driving forces. In the context of enterprise digitalization, the application of Knowledge Graph, such as merchant knowledge graph for merchant risk control, requires a deeper understanding of knowledge, particularly the need for in-depth context (i.e., Deep Context) perception for profiling and risk insights on thin data customer groups such as small and medium-sized businesses, new users, and dormant users [1]. Enterprise-level knowledge management is undergoing a transition from binary static models to dynamic multidimensional models.

1.2 Transition from Binary Static to Multidimensional Dynamic: Shift in Knowledge Management Paradigms

Knowledge Graph is a method of modeling and managing data that utilizes graph structure, knowledge semantics, and logical dependencies to provide capabilities for storing, reasoning, and querying factual knowledge. In its early applications, Knowledge Graph mainly involved extracting <s, p, o> triplets from public corpora to construct static knowledge graphs, aiming to improve search and recommendation efficiency and user experience. As Knowledge Graph applications have shifted from consumer-oriented applications like search and recommendation to enterprise-level applications in risk control and business management, as mentioned earlier, there is a growing demand for profiling and risk insights on long-tail sparse customer groups. This necessitates domain-specific graphs that possess comprehensiveness, accuracy, and interpretability. Moreover, the data sources for knowledge graphs have expanded beyond textual corpora to include diverse and heterogeneous enterprise data. These data sources include unstructured or semi-structured User-Generated Content (UGC) or Professionally-Generated Content (PGC), structured profiles derived from business operations, transactional data, log records, as well as domain-specific business expert knowledge. To support growth management and risk control, it is crucial to build comprehensive profiles of customers, materials, channels, and other dimensions. Taking merchants as an example, Figure 1 illustrates the process of constructing such multidimensional profiles.

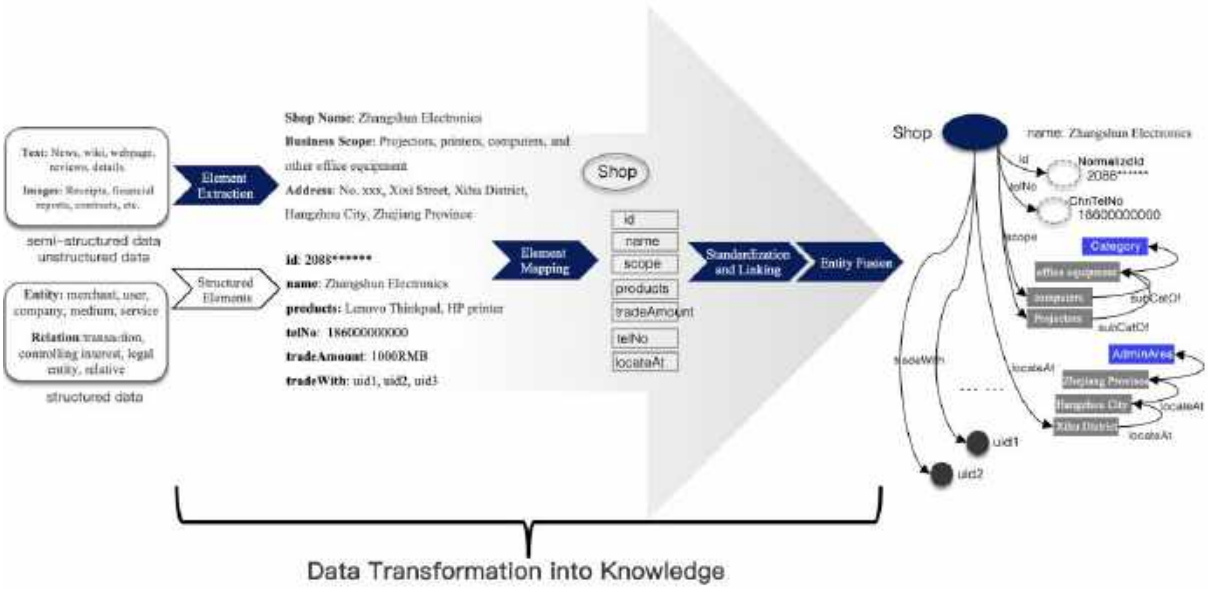

Figure 1: Constructing Merchant Entities

Merchants have surpassed the limitations of static physical stores, as anyone can become a merchant through payment codes. However, this also increases the difficulty of risk control. It is meaningless to rely solely on textual concept tags for risk control, and adding actual factual relations such as transactions and social connections is far from sufficient. As shown in Figure 2, deep collaborative information from multiple aspects of entities is needed to discover more effective associations. The requirements for Knowledge Graph construction have shifted from static common knowledge to dynamic Deep Context in temporal and spatial dimensions. This requires relation propagation based on media (such as Wi-Fi, phone, email) and boundary-based aggregation associations in continuous spatial dimensions [5,6]. It also involves tracking the multidimensional propagation context of events at different levels (micro, macro, and meso), achieving dense representation of sparse semantic relations between entities that are interpretable.

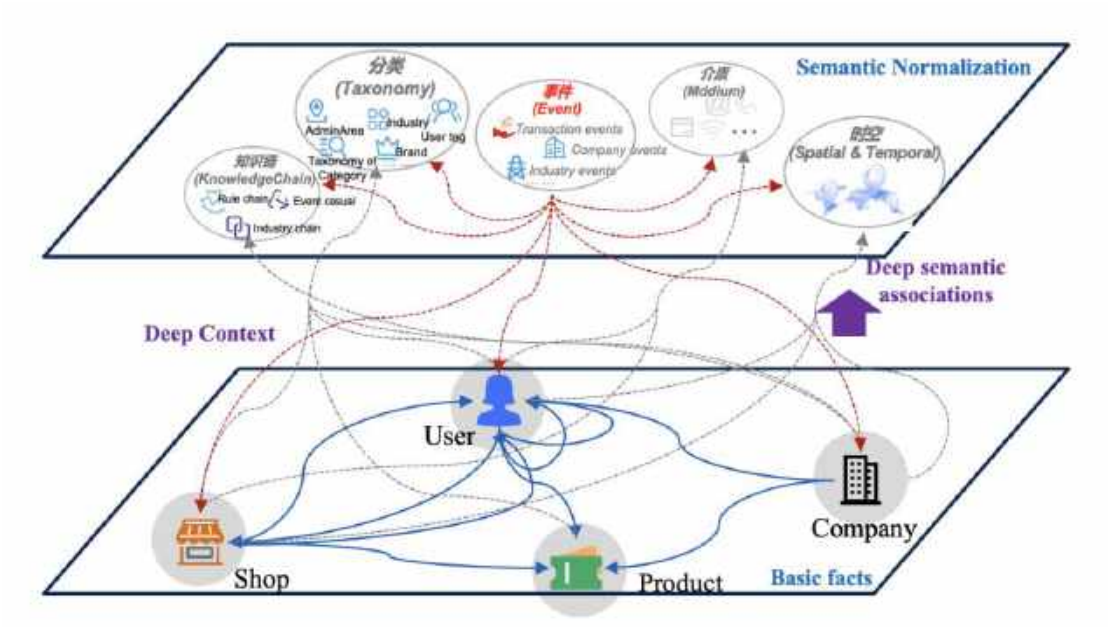

Figure 2: Deep Context Semantic Expansion of Foundational Fact Knowledge Graph

In terms of business applications, Knowledge Graphs can be used to construct knowledge reasoning tasks, such as: 1) Product recommendations: By leveraging semantic connections like category, intent, and temporal information, the semantic associations between people-products, people-merchants, and products-channels can be established, enabling semantic recall of products and representation transfer. 2) eKYB (Electronic Know Your Business): By leveraging media associations, behavioral events, and temporal aggregation, identification of shared merchants or individuals can be achieved, enabling effective profile completion and risk insights. In addition, Knowledge Graphs can also facilitate structured-aware text generation [7], such as: 1) Anti-money laundering intelligent adjudication and qualitative report generation: By combining Deep Context to predict risk behavior and detect criminal networks, the structure of networks and anomalies can be reconstructed through financial chains, temporal aggregation, and device associations, and then transformed into interpretable reports through knowledge graph to text conversion. 2) AI phone call victim alert: Suspicious devices, phishing domains/AppIDs, and criminal networks can be associated with transactional users, generating scripted conversations to alert users and intercept risks. These applications aim to achieve more intelligent and precise risk control and business inference, enhancing the efficiency and value of business operations.

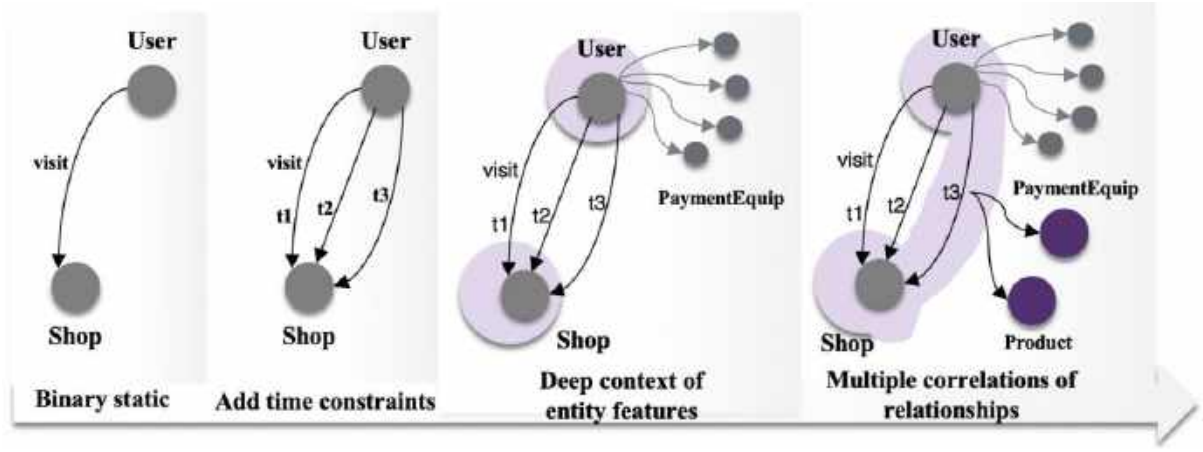

Figure 3: Evolution of Knowledge Representation from Binary to Multivariate

In the case of merchant management and risk control, knowledge management requires strong contextual awareness. Common knowledge graphs, which lack the ability to perceive contextual information and temporal associations, often suffer a significant decrease in effectiveness when applied to scenarios with diversified or intertwined argument elements, as they are unable to perceive individual differences and rely solely on concept-level induction for reasoning [8]. Similar challenges arise in fields such as anti-fraud in public security, insurance claims, medical consultations, and corporate credit assessment. As a result, there has been a significant shift in the expectations of vertical industries towards knowledge graph. Knowledge representation has evolved from the binary static structure depicted in Figure 3 to multidimensional dynamic associations in temporal and spatial dimensions, better aligning with the requirements of real-world applications.

1.3 Integrating Domain Knowledge Provides New Approaches for AI Implementation

According to Yunhe Pan, a member of the Chinese Academy of Engineering, data and knowledge are the two most important elements in the development of AI 2.0. Dealing with big data and multiple knowledge domains forms the core technologies for AI development, as knowledge can effectively assist in AI cognition, decision-making, and learning.

During the process of digitalization, a large amount of domain-specific knowledge, such as factual knowledge, expert experience, and operational procedures, can be accumulated through the extraction of massive data or business practices. This knowledge, existing in various industries and difficult to obtain publicly, holds immense value. By effectively integrating industry expert knowledge with AI, issues related to controllability, safety, and interpretability in AI applications can be addressed. By the end of 2022, ChatGPT had gained global popularity, followed by a surge of similar models in the domestic market. However, as Large Language Models (LLMs) are black-box probabilistic models [9], they struggle to capture and acquire factual knowledge, resulting in illusions and logical errors [10]. Meanwhile, Knowledge Graph (KG) provide factual accuracy, timeliness, and logical rigor, making them an excellent complement to LLMs. The application paradigm of LLM+KG, where Knowledge Graph serve as constraints and a source of complex reasoning capabilities, has attracted widespread attention and sparked numerous application explorations and research studies [9,10,11].

Table 1: Applications of LLMs and KGs in different digital enterprise scenarios.

Scenarios and applications		LLM only	KG enhanced LLM	LLM augmented KG	KG only
Business Growth	Interactive applications	Chat, write poems and songs	Knowledge Q&A, service retrieval, report analysis, etc.	-	Marketing recommendation, event context, marketing decision-making, etc.
	Marketing Recommendation	-	Data report query, crowd label selection, intelligent copywriting, etc.	-	Event analysis, materials analysis, crowd analysis, etc.
Risk Control	Risk forecasting and control	-	Explanatory message generation, waking up the robot, etc.	-	Clues tracking, events transmission, rule based claims, corporate credit, ultimate beneficiaries, equity penetration, etc.
Knowledge Construction	Knowledge extraction	-	-	Document element extraction, event extraction, entity linking, etc.	Knowledge construction based on structured business data
	Knowledge completion	-	-	Obtain the entity LLM embedding representation, extract and supplement the missing knowledge in the knowledge graph from the LLM	Relationships mining, properties prediction, groups mining, rules mining, etc.

In various application scenarios, taking merchant management and risk control as an example, the algorithm tasks can be categorized into the following five aspects: (1) Interactive Applications: including displaying products/services on the consumer end (C) and onboarding services/merchants on the business end (B). (2) Business Management: necessary business analysis and material management for enterprise and merchant operations. (3) Risk Control: combating illicit activities is an ongoing challenge for businesses, requiring enhanced awareness of thin data customer groups and rapid identification of new risk patterns. (4) Knowledge Construction: transforming external unstructured/semi-structured and structured data into

domain knowledge. (5) Knowledge Mining: continuous improvement of coverage for key elements and cross-entity relations to facilitate business growth and risk control. Table 1 lists potential applications of LLMs, KG and the mutual enhancement of LLMs and KG across different categories. These applications can help enterprises achieve better results and outcomes in the field of merchant management and risk control.

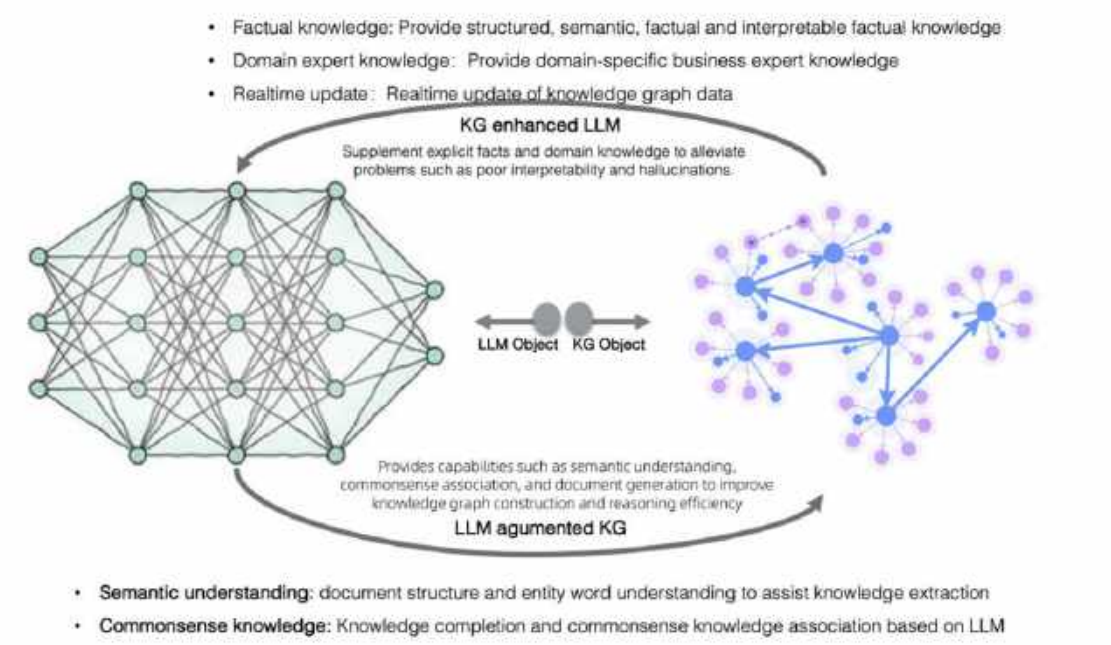

Figure 4: Mutual Drive between Large Models and Knowledge Graph

In general, taking the scenario of merchant management and risk control as an example, the algorithm tasks for LLM and KG applications can be categorized into three types: (1) LLM only: Due to the requirements for domain expertise and factual accuracy, there are currently no clearly applicable scenarios for LLM in the field of merchant management and risk control. (2) LLM + KG dual drive: This is mainly reflected in user interaction scenarios such as knowledge question-answering and report generation, as mentioned earlier, such as AI-powered phone call victim awakening and anti-money laundering intelligent trial report generation. Additionally, there are knowledge element extraction, entity linking, and other knowledge construction scenarios. The detailed description of the dual-drive of LLM and KG is presented in reference [10], including KG-enhanced LLM, LLM-enhanced KG, and the collaborative LLM+KG framework, as shown in Figure 4. (3) KG only: In decision-making, analysis, querying and knowledge mining scenarios that do not require complex language interaction and intent understanding, knowledge graph-based structured knowledge can be directly used for graph representation learning, rule reasoning, knowledge querying, and other tasks. By implementing the collaborative framework of LLM and KG, support is provided for cross-modal knowledge alignment, logic-guided knowledge reasoning, natural language knowledge querying, and more. This presents higher demands for unified representation of KG knowledge semantics and cross-scenario transferability of engine frameworks.

1.4 The Development of Knowledge Graph Technology System Needs to Keep Pace with the Times

The development of the knowledge graph technology does not completely match the expectations of its application in the new paradigm of managing new knowledge data and the dual drive of Large Language Models. The development of knowledge graph technology also needs to keep pace with the times. Specifically, the following issues exist:

Firstly, there is a lack of an industrial-level unified knowledge modeling framework. Despite the development of semantic-rich and loosely structured technologies such as Resource Description Framework (RDF) and Web Ontology Language (OWL) for many years, successful enterprise-level/commercial applications have not emerged. Instead, property graph with strong structure and weak semantics, such as Labeled Property Graph (LPG), has become the preferred choice for enterprise applications.

Secondly, there is a lack of a unified technical framework [2], resulting in poor cross-domain transferability. Due to the variety of tools and complex links, knowledge construction in each domain needs to start from scratch. In addition, there are also significant technical challenges in other aspects, as listed in Table 2.

Table 2: Technical Challenges Faced by Knowledge Graph in the New Paradigm.

Classification	Challenges	Description
1. Overall Framework	1.1 Industrial-Usable Knowledge Semantic Framework	Connecting big data with AI technology system, supporting the knowledge semantic framework that integrates factual and logical representations.
	1.2 Transferable Knowledge Graph Engine across Scenarios	The knowledge graph engine has transferability across scenarios, supporting the rapid incubation of new domain knowledge graphs.
2. Knowledge Graph Construction	2.1 Unified Knowledge Extraction Framework	Unified knowledge extraction based on unstructured/semi-structured data, ensuring throughput and performance under large-scale data.
	2.2 Unified Entity Linking Framework	Unified entity linking/standardization framework for knowledge elements, ensuring throughput, performance, and consistency under large-scale data.
3. Knowledge Graph Reasoning	3.1 Expert Rule Knowledge Representation	Constructing layered representation of logical dependencies between decision rules in the end-to-end business system.
	3.2 Rule-Guided Explainable Reasoning	Implementing effective rule injection and rule constraints, generating explainable inference results [7].

The goal of knowledge graph is to construct a machine-understandable and machine-reasonable digital world, achieving unified representation of knowledge semantics and hierarchical capability. This enables rapid construction of domain-specific knowledge graph and cross-scenario transferability, which is a fundamental core issue that must be addressed in the accelerated industrialization of knowledge graph.

1.5 Industrial Knowledge Graph Engine Based on SPG

The Knowledge Graph Platform of Ant Group, supported by years of experience in the financial industry, has developed a semantic framework based on property graph called Semantic-enhanced Programmable Graph (SPG). It creatively integrates the structural nature of Labeled Property Graph (LPG) with the semantic nature of RDF, overcoming the challenges of industrial implementation faced by

RDF/OWL's semantic complexity while inheriting the advantages of the simplicity of LPG's structure and compatibility with big data systems.

Firstly, SPG provides a clear definition of knowledge in the digital world. Knowledge is the accumulation of human exploration in the material and spiritual world, but how should machines perceive knowledge in the digital world? SPG defines the concept of knowledge in the digital world through formal description and objective facts. In conjunction with Figure 5, SPG provides a formal definition from three dimensions:

(1) Domain Type Structure Constraint: In the objective world, every entity (Thing) belongs to at least one type (Class), and the digital world follows the same principle. Based on SPG, the Domain Model Constrained (SPG DC) provides a constraint on the domain structure type, enabling automatic classification of knowledge subjects and hierarchical organization from dynamic spatiotemporal to static common knowledge.

(2) Uniqueness of Instances within a Domain: In the objective world, there are no two identical entities, and the digital world should be the same. However, due to issues such as data duplication in the digital world, caused by multiple sources and data copying, data redundancy and repetition are common. SPG Evolving utilizes the capabilities of entity linking, concept standardization, and entity resolution provided by the SPG Programming (Knowledge Construction SDK) framework. It combines natural language processing (NLP) and deep learning algorithms to enhance the uniqueness level of different instances within a single type (Class), supporting continuous iteration and evolution of the domain knowledge graph.

(3) Logical Dependencies between Knowledge: In the objective world, everything is connected to other things, and there are no isolated entities, which holds true in the digital world as well. SPG Reasoning utilizes predicate semantics and logical rules to define dependencies and transitivity between knowledge, providing a programmable symbolic representation to facilitate machine understanding.

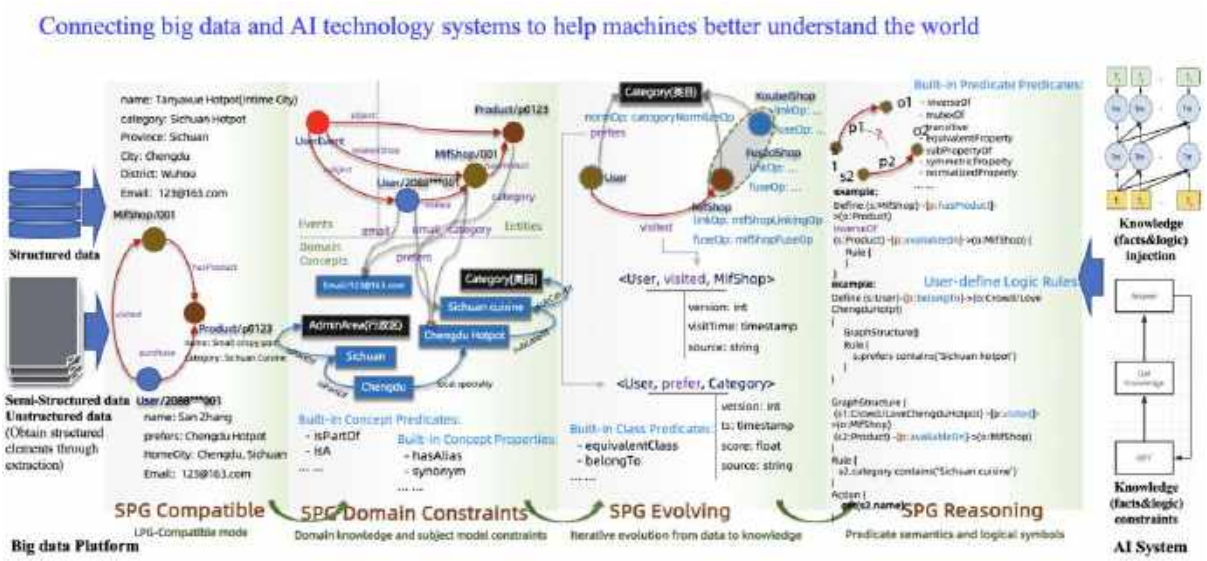

Figure 5: SPG Knowledge Semantic Framework

SPG fully integrates the advantages of LPG and is compatible with big data systems. The knowledge engine built on SPG seamlessly connects with the big data architecture during the knowledge construction phase, providing a framework of knowledge construction operator to facilitate the transformation from data to knowledge. During the storage phase, it can adapt to property graph to fully leverage their storage and computational capabilities. During the reasoning phase, it is formalized as KGDSL (Knowledge Graph Domain Specific Language), which provides a machine-understandable symbolic representation to support downstream rule reasoning, neural/symbolic fusion learning, KG2Prompt collaborates with LLM for knowledge extraction/reasoning, and more. Additionally, through a layered architecture, the construction of a new domain knowledge graph only requires defining the schema, preparing the data, and developing construction/reasoning operators.

The knowledge graph technology is still in a period of rapid development and at a critical turning point in terms of technology. A unified technical framework can significantly lower the application threshold and promote the prosperity of the ecosystem. Therefore, this whitepaper focuses on the fundamental issue of enterprise-level knowledge management and deduces the full lifecycle of knowledge management, knowledge construction, and knowledge reasoning. The goal is to achieve an industrial-level, portable knowledge representation and engine framework. As mentioned earlier, Labeled Property Graph (LPG) has become the preferred choice for most enterprise knowledge modeling due to their unique compatibility with big data architectures. This whitepaper also derives the semantic capabilities required for enterprise-level knowledge management from practical business issues.

Chapter 2 Challenges of Knowledge Management base on Labeled Property Graph

In enterprise-level knowledge graph applications, as discussed in Chapter 1, Labeled Property Graph (LPG) is preferred for domain knowledge graph modeling due to its efficiency and compatibility with large data systems. it enables the rapid realization of business value. While the LPG-based knowledge construction has lower initial costs, as business rapidly grows and the volume of knowledge increases, the shortcomings of LPG become increasingly evident due to the lack of knowledge semantics and management capabilities. Firstly, the evolution of knowledge models becomes increasingly challenging, with schemas becoming more complex. Secondly, the flexibility of the node/edge model leads to redundant type creations and repetitive data preparation, making it difficult to maintain consistency and rationality among different relations / properties. Thirdly, the naive property/relation model is insufficient to depict the intrinsic semantics of entities and the semantic dependencies between them, resulting in significant obstacles to the continuous iteration and upgrading of knowledge graph projects. When the scale becomes unmanageable, new projects have to be created to rebuild schemas and graph data. Additionally, a large amount of hard-coding is required during the business application phase to achieve semantic parsing and alignment. This chapter combines two business cases, namely, risk mining knowledge graph and enterprise causal knowledge graph, to introduce the background and main pain points of business application and summarize the related issues. Next, in Chapters 3/4/5/6/7, we will attempt to propose solutions. Finally, in Chapter 9, we will provide two complete SPG-based solutions, aiming to leverage the advantages of LPG while avoiding its drawbacks, and providing efficient semantic modeling and knowledge management tools for enterprise-level knowledge graph applications.

2.1 Typical Case 1: Risk Mining Knowledge Graph

To achieve the primary business objectives of the risk mining knowledge graph, we aim to construct user-related risk profiles and associated networks for devices, media, transactions, and other relevant factors. Based on explicit or implicit associations, we identify individuals involved in the risky activities and implement risk control measures. Taking the app network risk prevention and control as an example, a particular app is found to be involved in risky activities such as gambling, pornography, and fraud. The following two objectives are expected to be achieved through the associated network of this risky app: (1) Identify the individuals behind the risks and apply corresponding risk control strategies based on the mining clues. (2) Discover other undetected risky apps and prevent the spread of the risks.

However, in practice, individuals involved in risky activities often disguise or hide their behaviors. They may use a large number of virtual devices, virtual IPs, or virtual identities, which are concealed among normal users. Therefore, this chapter will provide examples using a portion of the data listed in Tables 3, 4, 5, and 6 to illustrate the problems encountered in current LPG-based knowledge management. In the given example, the app (denoted as “** Entertainment”) should be identified as a gambling app developed by

"Wang Wu". "Li Si" should be recognized as the boss of the gambling company B, and "Zhang San" and "Li Si" are the same individual.

Table 3: Basic Information of User Entity

User Id	User Name	Phone Num	List of MAC addresses for owned devices	Owned certificate	Entity type
1	Wang Wu	154xxxx3456	06:8A:5F:2E:AB:85 06:8A:5F:2E:AB:86	Certificate 1	Person
2	Li Si	135xxxx5532	06:8A:5F:2E:A1:85		Person
3	Zhang San	135xxxx5532	06:8A:5F:2E:A1:85		Person
4	Company B	131xxxx3456		Certificate 2	Company
...	...	...	...	...	...

Table 4: Basic Information of the Shareholding Relation

Shareholder's Name	Company Being Held	Shareholding Percentage
Zhang San	Company A	100%
Company A	Company B	100%
...	...	...

Table 5: Basic Information of the Application Entity

App Id	App Name	List of MAC addresses of installed devices	Owned certificate	Is it a gambling application (based on user complaint labeling)
1	**Entertainment	06:8A:5F:2F:AB:85, 06:8A:5F:2E:AB:86	Certificate 1	yes
2	Fishing Master	06:8A:5F:2E:AB:85	Certificate 2	unknown
...	...	...	...	

Table 6: Transfer Relation

Transferring User	Receiving User	Transfer Amount
Li Si	Wang Wu	10000
...	...	....

There is a significant gap between the expression of data and the business expectations. This is manifested in the following ways:

- Difficulty in representing deep-level associations between different entities: It is challenging to directly derive the relations between applications and users, as well as the associations between different applications, from the data structure.
- Alignment of different characterizations of the same entity: Natural persons and users cannot be directly equated. For example, in this case, the users "Zhang San" and the one labeled as a gambling boss may refer to the same individual.

In business practice, although there may not be direct associations between applications and users, or between different applications, indirect associations can often be discovered through intermediaries such as devices or certificates. Similarly, relations between users can be explored through methods like shared

phones or devices. To cope with such complex network relations, the knowledge graph typically evolves as follows:

Step 1: Transforming tabular data into property graph representation.

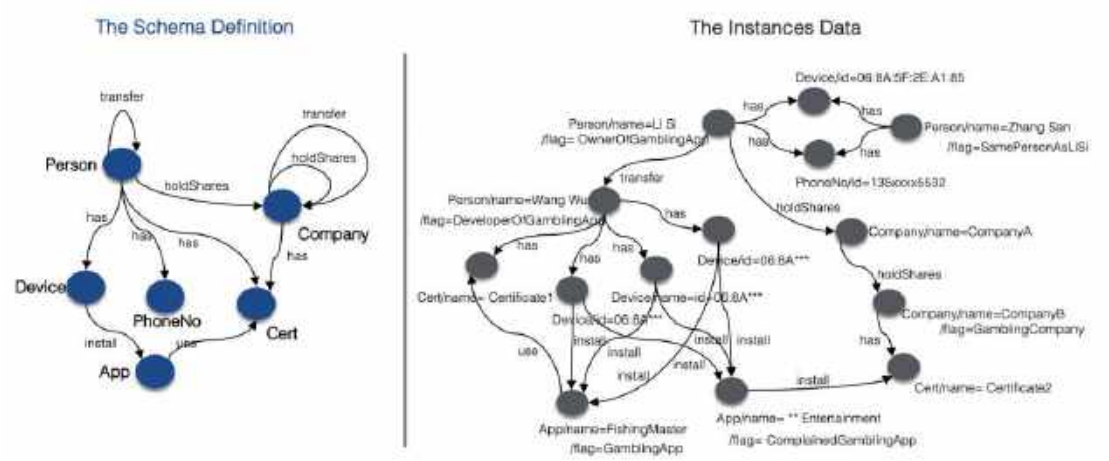

Figure 6: Building the Knowledge Graph by Directly Mapping Tabular Data to Property Graph

Figure 6 illustrates the mapping of tabular data to the data structure of the knowledge graph. At this stage, it is possible to derive the relations between risky applications and risky individuals based on multi-hop relations. However, further analysis and judgment by business experts are still required to directly achieve the business objectives, as depicted in Figure 7. The textual structure of entity instances in both Figure 6 and Figure 7 is as follows: Type / PropertyName = PropertyValue.

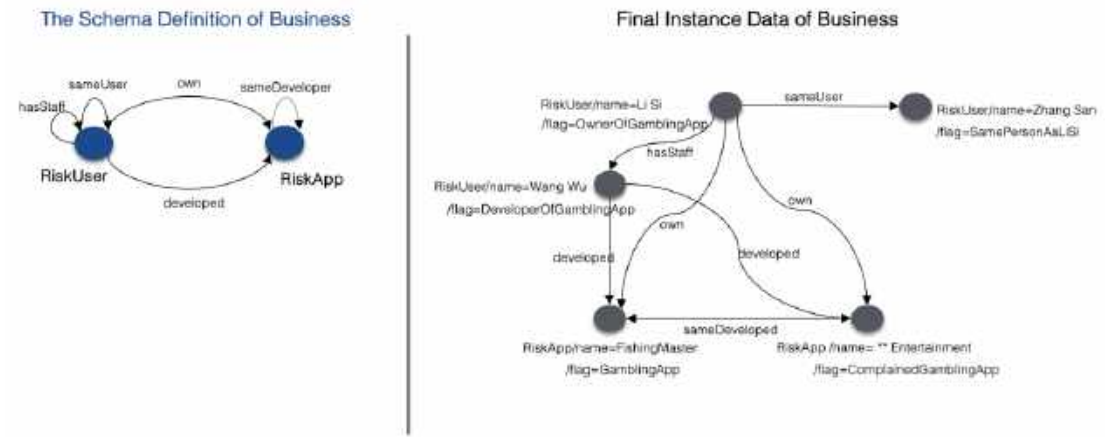

Figure 7: Knowledge Graph Structure Derived through Implicit Inference to Meet Business Expectations

The data structure of the knowledge graph required by the business is typically different from the original graph data. The original graph data represents objective basic data, while the data required by the business is based on the mined associations derived from the objective data. These associations need to be re-integrated into the original data. To uncover these implicit relations, business experts formulate a series of rules, such as rules for determining the same user, rules for user-app ownership, and rules for app developer relations. For example, if two users use the same phone number or device, they are considered to have a “same phone” or “same device” relation. If a user has a controlling relation with a legal entity, the app released by that legal entity is considered to be owned by that user. If a user has multiple devices that

have the same app installed, the user is considered to be the developer of that app. By applying these rules and performing rule calculations using external big data systems, the required data structure for the knowledge graph is obtained, including the addition of new types and relations. The original basic information definitions are also retained to support better decision-making and risk control in the business domain.

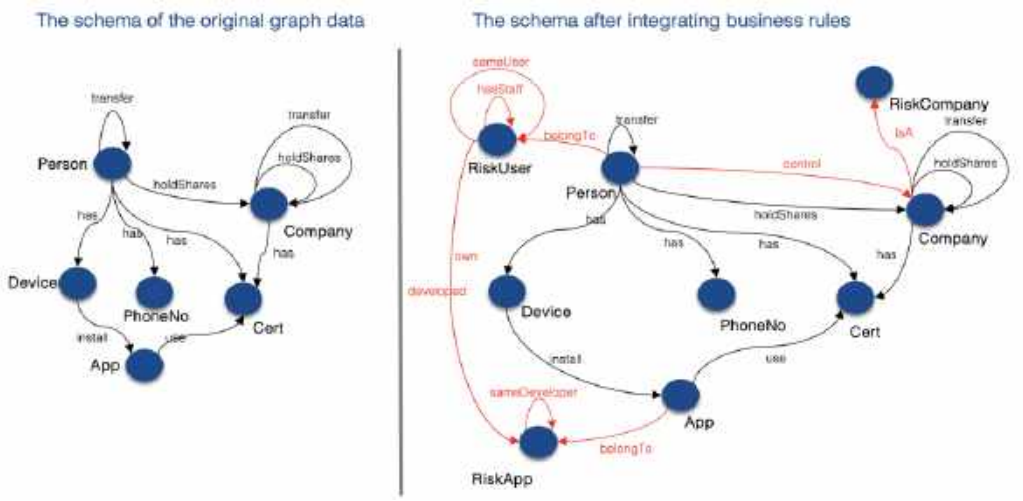

Figure 8: Schema Differences after Incorporating Business Rules

The above example demonstrates a portion of the redundant creations that occur during the business decision-making process. In knowledge graph management, extracting complex implicit associations from basic facts is a fundamental requirement. However, we need to address how to avoid continuous schema expansion caused by the refinement of business objectives and ensure the logical consistency between rule calculations and the underlying facts. These are fundamental issues that knowledge management must address.

2.2 Challenges in Applying LPG to the Risk Mining Knowledge Graph

- **The independent data preparation for nodes and edges significantly increases the construction cost of the knowledge graph.** To construct the required entities and relations for the risk mining knowledge graph, data preparation for nodes and edges is required, which is much larger than the original four tables.
- **The difficulty in directly reusing different knowledge graphs leads to redundant data preparation.** In this business case, it is necessary to construct knowledge graph data for fund transfers and equity structures. Typically, these data already exist as foundational data in other knowledge graphs.
- **Inconsistencies caused by logical dependencies between entities and elements.** In the business modeling, the new types and relations in Figures 7 and 8 are derived from the existing data in Figure 6. When the underlying data changes, such derived data must be synchronized, or else inconsistencies in the knowledge graph data will occur.

- **Continuous expansion of the knowledge graph structure due to the migration and changes of business objectives.** In this case, implicit associations through intermediaries are used to identify risky users behind the applications. However, as risky activities evolve rapidly, there will be frequent updates and creation of different entity and relation types. The size of the knowledge graph schema and instances will continue to expand, making it difficult to manage.

Therefore, when constructing a knowledge graph, these challenges need to be considered, and appropriate measures should be taken to optimize the data transformation process, improve the reusability of knowledge graph, and design the schema to support the expression of logical relations between knowledge, thereby enhancing the efficiency of business semantic migration. This helps us build a more efficient, reliable, and maintainable knowledge graph system.

2.3 Typical Case 2: Enterprise Causal Knowledge Graph

The knowledge management of an enterprise causal knowledge graph focuses more on depicting the logical relations of causality, conditionality, hierarchy, and sequentiality between events. Therefore, the foundation of an enterprise causal knowledge graph is events. In practical applications, it generally evolves from the application of events and graphs to the causal knowledge graph, this evolves capturing production and operational events related to the enterprises, extracting key elements of events, establishing the linkage between event elements and internal enterprise/industry chain knowledge graph, and constructing causal logical chains between risk events and enterprise/industry chain knowledge graph. This allows for quick linkage to internal warnings or risk management when external risk events are identified. When a financial event occurs, we need to infer the event based on basic facts in order to try to obtain answers to the following questions:

- The nature and impact of the event.
- The entities involved and the impact on related entities.
- Whether the associated entities will further generate other impacts and how they will be affected.

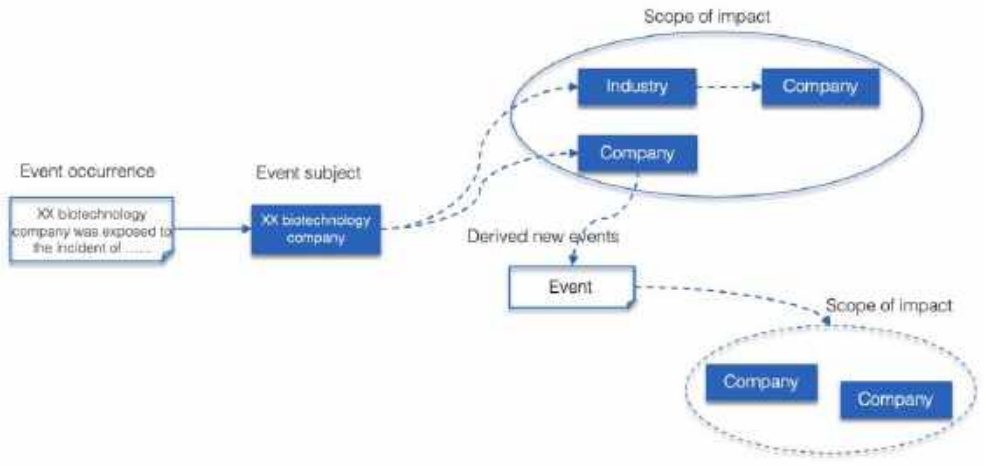

Figure 9: Illustration of Event Impact Propagatio

For example, let's consider an event where a biotech company is exposed for producing fertilizers with severe heavy metal contamination. The extent of the event's significance needs to be further analyzed. When analyzing a specific event, analysts need to repeatedly query and gain insights from basic factual knowledge, based on their understanding of the event and combining it with common knowledge to draw conclusions about the event's impacts. However, various reasoning logic and data are often fragmented and dispersed, requiring effective integration and connection. Therefore, there are many unresolved issues in the application of knowledge graphs in understanding causality. In the case of this event, it is necessary to analyze the impacts on which entities in the enterprise network, the paths and degrees of impact, and whether the impact on other entities will give rise to new events, thus further expanding the scope of influence.

Problem 1: Complex and Diverse Event Classification, Predefined Event Taxonomies Cannot Fully Cover Real-World Application Scenarios.

The traditional approach is to define multi-level event types and construct an event type tree through the expertise of business professionals. This involves defining, describing, and classifying events based on the understanding of equity markets, fixed income markets, and macroeconomic changes. Different events are delineated by their boundaries. Additionally, events can be defined as “changes” in the financial market, typically associated with a series of financial indicators. A set of predefined labels in the form of an “event tree” is created by business experts, and different financial event propagation networks are built based on this tree and historical data. However, such an approach often fails to meet the needs of real-world financial markets, primarily due to the following reasons:

1. Different interpretations of event types. Due to different backgrounds, business experts may have diverse understandings of event trees, leading to inconsistencies and variations in their definitions of the same events. The boundaries between different event types and events may not be clear.
2. Static event trees cannot accommodate the dynamic development of the financial market and the emergence of new financial event types. Especially after the 2008 financial crisis, the global economy entered a new normal, and the domestic economy has shown new features in recent years. For example, the COVID-19 pandemic has had a significant impact on the global economy and various industries. However, in the existing event trees, it is generally classified under the category of “major health security” events, and many business analyses often compare it to the SARS outbreak in 2003 to predict future impacts. However, although both are “major health security” events, they differ significantly in terms of impact time, scope, and other aspects.

In conclusion, due to the complexity of financial events, relying solely on a group of business experts to predefine events cannot fully cover real-world application scenarios. We need a system to dynamically generate derived financial event taxonomies.

Problem 2: Multiple Interrelationships Exist Between Events, Such as Causality and Sequence, which often Require Dynamic Connection through Entity Networks, Demanding Strong Descriptive Capability.

Due to the complexity of the financial event network, the impact to other events after the occurrence of a particular event may vary. This often depends on the differences in entities and relations behind the different events, which determine the different directions of impact for each event.

For example, if Company A's stock price rises due to its expansion of production capacity, and the capital market has a positive outlook on its future development, whether the stock price of its competitor, Company B, will rise or fall will often depend on various factors, such as market demand, the scale of capacity expansion, and the relative market share between the companies and their competitors. Suppose Company A is a semiconductor manufacturer and decides to expand its production capacity. For its competitor, Company B, this may be good news. If there is strong global demand for semiconductors and a tight supply, A's capacity expansion may help alleviate this supply-demand imbalance and stabilize the entire market. In this case, as the market environment improves, the competitor B may also benefit from it. The logic in this case is: if the demand for the entire industry exceeds supply, any action that increases supply may have a positive impact on the entire industry as it helps maintain market stability and prevent price surges or other factors that may lead to market instability. On the other hand, if Company A is an automobile manufacturer and decides to expand its production capacity, it may have a negative impact on its competitor, Company B. In this case, if market demand does not grow, A's capacity expansion may lead to market oversupply, triggering price competition. Therefore, for competitor B, this may result in lower sales volume and profits, making it a negative news. The logic in this case is: if the supply growth in an industry exceeds demand, it will lead to oversupply, potentially triggering price competition, which in turn affects the profit levels of all firms.

In conclusion, due to the complexity of the financial event network, when describing the transmission relations between different events, it is necessary to dynamically link them with the relevant entity network and build a strong descriptive capability based on it.

Problem 3: How to better describe and analyze the propagation of event impacts?

Due to the complexity of financial event inference, it is necessary to analyze the propagation effects of events from two perspectives: the propagation in entity networks and the propagation in event networks. Taking the example of “Company A announces bankruptcy/bond default”, we can analyze the event's propagation effects from these two angles:

1. Propagation in entity networks: Company A's bankruptcy will directly impact its shareholders, especially major shareholders, whose financial conditions may be affected, thus further influencing their investments in other companies. Additionally, Company A's competitors may benefit from its bankruptcy, potentially gaining market share. Similarly, suppliers and creditors of

Company A may suffer economic losses due to the bankruptcy. These impacts will propagate in the entity network, affecting other relevant entities.

2. Propagation in event networks: Company A's bankruptcy may serve as a cautionary example for other companies, preventing similar occurrences. For example, it may enhance risk awareness in related industries or markets, prompting companies with issues in financial management and risk control to learn from it and make necessary improvements. The impact of this event will propagate in the event network, forming new events and affecting other entities.

These two propagation processes are not isolated but intertwined. For example, Company A's bankruptcy may draw the attention of its competitors and influence their decision-making, thereby triggering new events in the entity network. Simultaneously, this new event may also become a new node in the event network, further influencing the behavior of other companies.

Problem 4: The process of financial event inference is not sufficient solely based on relation network, it often requires the use of extensive external data for analysis.

In 2019, a dam collapse incident occurred at the Vale of Brazil, resulting in an increase in iron ore prices, which in turn led to a rise in steel production costs. Within the entire chain of event impacts, some companies involved in industry competition benefited from this incident, as their profits rising. However, it also had a negative impact on the downstream of the industry chain, as rising costs led to a decrease in profits.

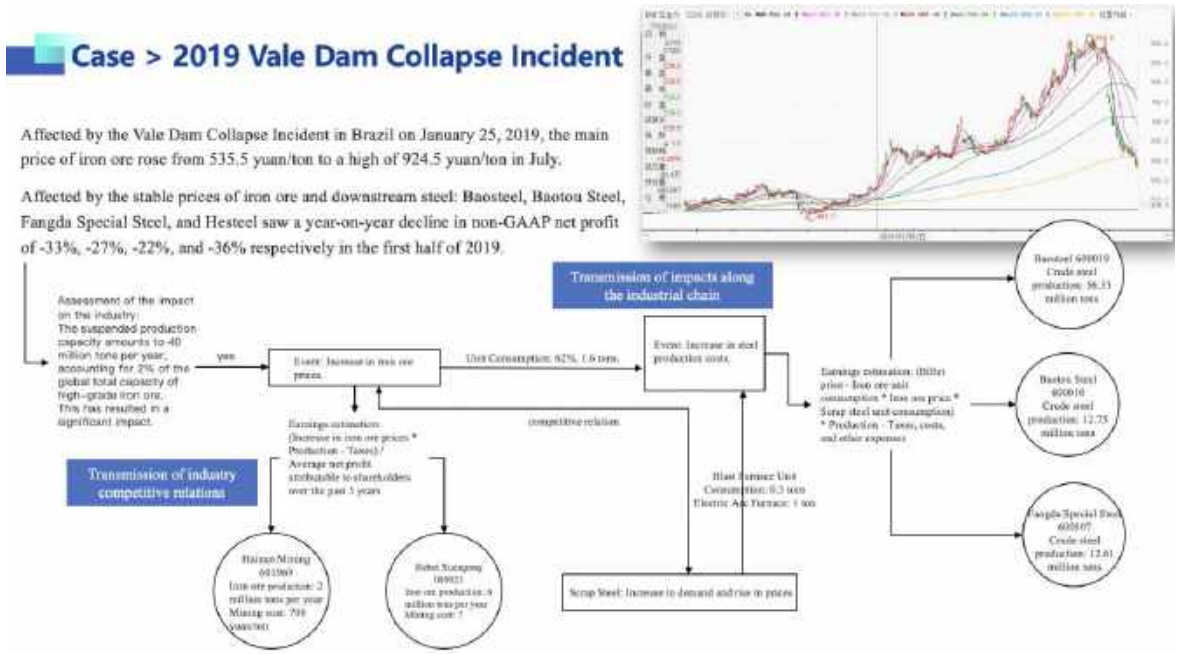

Figure 10: Transmission Diagram of the Impact Chain of the Vale Dam Collapse Incident

The entire iron ore industry chain starts with iron ore extraction, and Vale S.A. is an important participant in the global mining industry, with its operations significantly impacting global iron ore supply and prices. It is precisely because of the company's importance in the global iron ore industry chain that the dam collapse incident led to a global increase in iron ore prices.

China, as a major infrastructure country and the largest consumer of iron ore globally, heavily relies on the global iron ore market. Therefore, a major incident in a Brazilian company can lead to an increase in raw material prices in the iron ore industry chain and successfully transmit the event to the domestic capital market. On the other hand, after the import of iron ore, it goes through the process of smelting, refining in a converter, and casting to produce pig iron. Pig iron is then further processed into various steel products, such as long products (rebars, wire rods) and flat products (hot-rolled coils, cold-rolled coils), which are used in industries such as automobiles, appliances, and shipbuilding. Additionally, there are also pipe products (seamless steel pipes, welded steel pipes), special steel, high-strength steel, and other different products. Various Chinese listed companies are involved in these upstream and downstream segments of the industry chain, such as Baosteel, Baotou Steel, and Fangda Special Steel. The specific transmission logic and impact need to be analyzed in conjunction with the details of the arguments, including the following aspects:

1. Whether the company engages in hedging in the derivatives market, and the value of hedging transactions.
2. The market share of the company's products and the competition landscape in the segmented industry. Generally, the competition landscape for special steel is considered better than ordinary steel.
3. Whether the company has the ability to transfer upstream production pressures to downstream, and whether there are upstream alternatives domestically.

Only by dissecting the above arguments into finer granularity and introducing relevant external data can a complete transmission network be constructed.

2.4 Challenges in Applying LPG to the Enterprise Causal Knowledge Graph

In general, the analysis of event impacts is based on the analyst's understanding of the event, repeatedly querying and gaining insights from basic factual knowledge, and combining it with common-sense knowledge to draw conclusions about the event's impact. It can be seen that the entire process of event inference is outside of the knowledge graph, as basic factual knowledge lacks common-sense and reasoning logic, and a pure event knowledge graph cannot express the context of the event. In practical applications, in order to complete event inference, various logics have to be scattered in various places outside of the knowledge graph, and reasoning is performed through various external plugins. Such methods inevitably bring many application issues to enterprise causal knowledge graph:

- **Contradiction between the static nature of predefined schemas and the dynamic nature of events.** Events are often complex and diverse, and if a strongly schema-constrained property graph is used, it is generally impossible to predefine events. It can only be tailored to specific scenarios. If a schema-free property graph is used, the overly lenient mode will result in increasing data management and usage costs.

- **Inability to express the entire event transmission context.** Since the knowledge graph only contains basic facts without the definition and transmission relations of the events, it is impossible to establish expert rules for event analysis, let alone represent the entire context of event propagation. To express the event context, it is necessary not only to illustrate the evolutionary process of events over time but also to combine abstract entities to express the relevance of events within the event domain through abstract levels.
- **Separation of the knowledge graph and reasoning logic makes it difficult to evaluate the correctness of the reasoning logic and hinders the reuse of reasoning logic.** Due to the separation of schema and reasoning logic, when maintaining basic factual data, it is impossible to assess the impact on the correctness of external reasoning logic. For example, changes in data such as property names or deleted relations may cause the failure of reasoning logic that exists outside of the knowledge graph. Such problems are unavoidable in traditional event knowledge graph. Furthermore, reasoning logic may consist of a combination of query statements and scripts, and these contents may be managed in analysts' local storage, making it difficult to reuse reasoning logic that is highly generic.
- **Poor interpretability of the conclusions derived from event propagation reasoning.** Since the external reasoning logic may be a combination of multiple query statements and scripts, it is not possible to visually observe the deductive process from the cause to the result when the entities affected by the event are calculated. In this case, interpretability becomes a black box, and understanding the query statements and scripts is necessary to comprehend the reasoning logic.

2.5 Complexity and heterogeneity caused by the coupling of structural definition and semantic representation in knowledge modeling

RDF/OWL is a syntax-level representation framework, leading to a higher learning cost. In traditional ontology modeling in knowledge engineering, a classification system needs to be defined through description logic syntax. The labeled property graph (LPG), has simple syntax elements but only represents the data structure. None of the above methods address the problem of “design patterns” themselves. In the process of practical business implementation, the coupling of data structure definition and knowledge semantic ontology design in the modeling process leads to difficulties in decision-making. The schema design of domain knowledge graph is subjective, where entities of the same type are defined differently due to naming and granularity differences. Heterogeneity issues caused by different schema definitions are prevalent, hindering the dissemination and reuse of knowledge, and further exacerbating knowledge inconsistency.

2.5.1 Repetitive construction issue caused by differences in entity type granularity due to different business goals

In the application of the risk mining knowledge graph, there is a need to classify the "Person" entity and determine whether they are involved in risky activities. The risky personnel can be further divided into categories such as gambling individuals, bookmakers, money launderers, and so on, as shown in Figure 11.

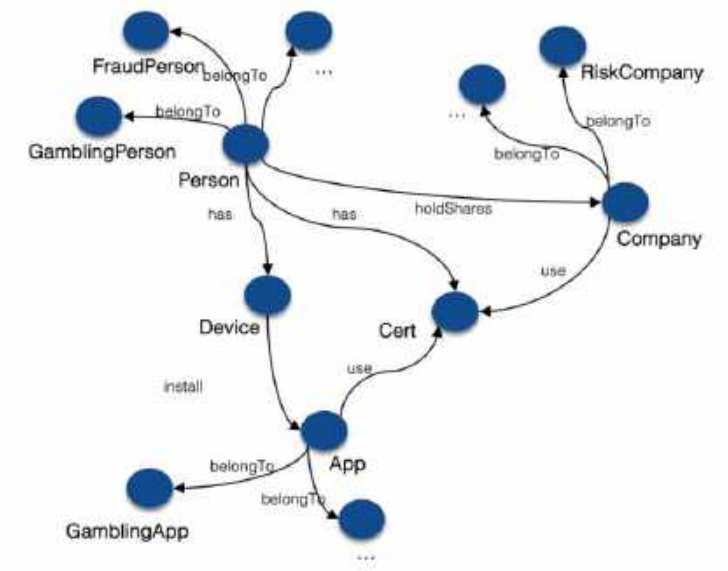

Figure 11: Entity granularity expansion in the modeling process of the risk mining knowledge graph

Even within the same knowledge graph project, different internal demands may lead to the creation of new entity types. The requirement for an entity to have multiple types is often resolved by creating redundant new types. This results in increasingly complex schemas. At a certain stage of business evolution, there may be a need to start from scratch and redesign the knowledge graph. Taking the risk mining knowledge graph as an example:

- **Demand for analyzing different apps:** Black industry groups produce a large number of apps through batch repackaging, similar to an app factory. It is necessary to classify and refine the types of apps to address different risk control strategies.
- **Demand for on-demand refinement of entity types:** When investigating black industry groups, some individuals may be associated with bookmakers, fraud activities, and other specific roles. This leads to the creation of additional types such as "GamblingPerson" and "FraudPerson". As the business evolves, the classification of entities continues to become more refined.

These issues are often strongly correlated with specific business scenarios, and change as the business evolves, and different scenarios arise. From the data management perspective, these apps or persons may use the same or similar data structures. However, from the business logic perspective, there is a need for semantic-level type differentiation. The mixture of schema/ontology modeling from different perspectives

leads to continuously increasing costs for user understanding and maintenance. The redundant construction of entity types also increases the preparation and maintenance costs of data tables.

2.5.2 Different knowledge graph defining the same entity differently

Taking fund flow as an example (cross knowledge graph), as shown in Figure 12.

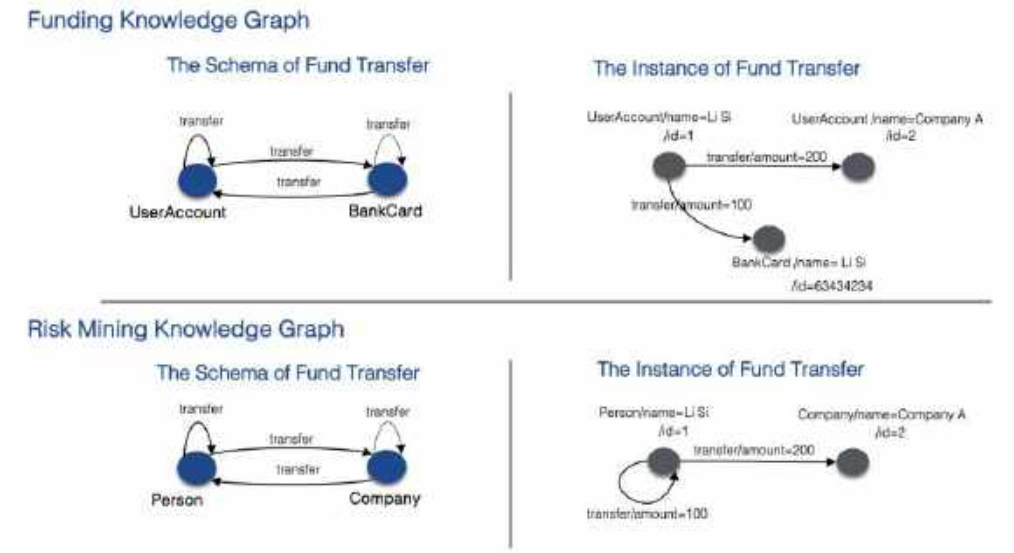

Figure 12: Illustration of cross knowledge graph fusion

In the risk mining knowledge graph, we focus on the transactional relations between users and companies to identify the masterminds behind illegal activities. In the fund knowledge graph, our focus is on analyzing the flow of funds. Therefore, we deploy tracking and control strategies for the involved financial products and treat them as more granular entity types. In both of these scenarios, we deal with transactional relations to meet our respective business requirements. However, there are two problems:

- Different businesses handling the same data in a similar way result in the inability to consolidate common requirements and share accumulated business experiences. Each newly business scenario needs to start from scratch to prepare the data, increasing the threshold for business usage.
- Knowledge sharing across knowledge graphs. For example, the “BankCard” entity exists in the fund knowledge graph. However, it cannot be securely used for business needs such as anti-money laundering or anti-fraud.

2.5.3 Difficulties of making the choice between defining as properties or relations due to construction costs

In the labeled property graph model, each entity and relation type require independent data preparation. With M types of entities and N types of relations, due to differences in property quantities, M + N message structures or data tables need to be prepared to complete the knowledge graph construction. Leading to high data preparation costs. As the demand for knowledge graph-based analysis increases, this cost continues to

escalate. All entities and relations defined in the labeled property graph require separate data preparation, forcing businesses to balance between current simple applications and future scalability. When properties are directly constructed as relations, it increases the complexity of simple application usage, such as the lack of property filtering.

Consider a simple question: connecting devices that use the same Wi-Fi. As shown in Figure 13, our usual approach is as follows: (1) Construct entity types for Device and Wi-Fi, and a relation type "Device -[useWifi]-> Wi-Fi". (2) Prepare data separately for the entities and relations mentioned above, and generate unique entity IDs for Wi-Fi.

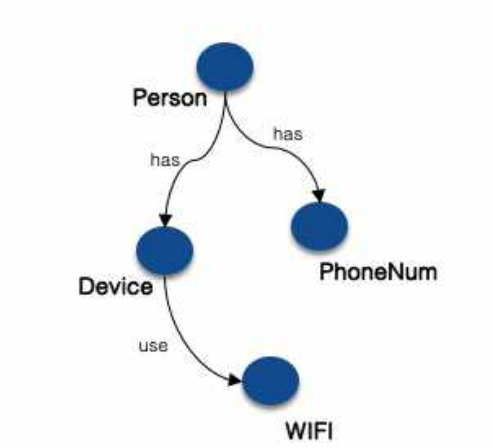

Figure 13: Connecting devices using the same Wi-Fi

This significantly increases the complexity of data preparation, as each entity and relation type require separate data preparation. In extreme cases, if each type requires its own data table, the number of tables increases from 2 to 6. This results in a larger workload for data cleansing. Assuming there are "m" entity tables, with an average of "n" property columns per table that need to be transformed into relation, we would need to generate a total of "m*(2*n+1)" tables. This raises the threshold for user usage.

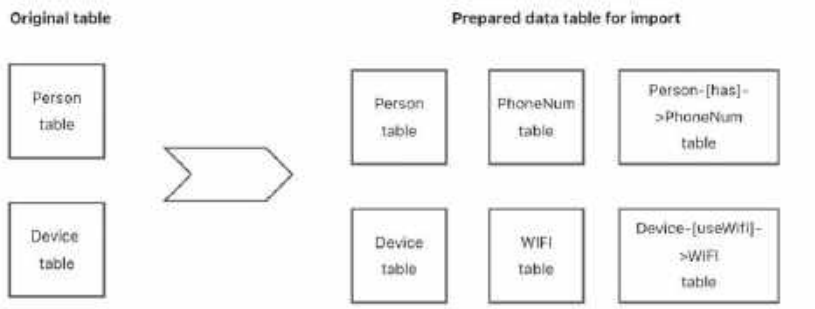

Figure 14: Cost escalation due to entity/relation data preparation

2.6 Insufficient expressive power for representing diverse and heterogeneous domain knowledge

During the implementation of knowledge graph in the financial domain, there is a need for heterogeneous representations of temporal and spatial aspects, such as user behavior, industry events, macro

events, and so on. For example, the enterprise causal knowledge graph needs to express the temporal and spatial relations of individual events as well as model simple or complex logical relations such as causality, succession, co-occurrence, and structure. It is difficult to achieve lossless expression using RDF/OWL, and although the introduction of the HyperGraph [12] can alleviate some of these issues, it does not integrate well with the RDF/OWL system, thereby increasing the cost of user application and understanding.

2.6.1 Representation issues of temporal and spatial structures in events

Representing the multi-element structure of events is also a problem of lossless representation, similar to the HyperGraph. It expresses the temporal relations of various elements in a multi-element structure, where events are temporary associations formed by these elements due to certain behaviors. Once the behavior ends, the association disappears, as shown in Figure 15.

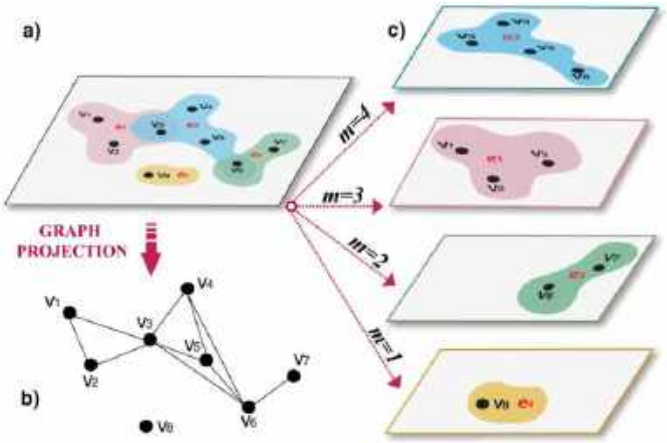

Figure 15: Representation of the HyperGraph [13]

The representation method of RDF-Star [14] extends the modeling capabilities of RDF for such scenarios, and in 2022, the W3C established the RDF-Star Working Group to further enhance RDF. Taking the application of the enterprise causal knowledge graph as an example, the simple structure of a safety production event in a company is represented as shown in Figure 16.

Figure 16: Extension of multi-element relations based on RDF-Star triples

In the representation format of "<s, p, o>" triples, the first step is to extend it with a time element to "<s, p, t, o>" in order to further represent temporal constraints, as shown in the example "<Company, Occurrence, OccurrenceTime, SafetyRiskEvent>". However, event associations are often complex combinations of multiple elements. Breaking down the different aspects of an event into independent elements is necessary, as shown in Figure 16. In the construction of domain knowledge graph based on labeled property graph, which has been developed for many years, there is no solution for how RDF-Star can be applied. We need the representation capabilities of a spatiotemporal event hypergraph based on labeled property graph in order to build the event representation and reasoning capabilities required for enterprise causal knowledge graph.

2.6.2 Problems with causal succession, composition, structure, and logical dependencies

The enterprise causal knowledge graph have an ontology layer, which means that there are not only horizontal associations between events and entities, but also vertical associations from specific to general or from general to specific. Horizontal associations involve roaming, association, and analogy, while vertical associations involve induction, deduction, and evolution. Therefore, the corresponding architecture should carefully consider these situations when making decisions. In terms of the definition and instantiation layer, there are four components: abstract entities, concrete entities, abstract events, and concrete events. They are physically connected as a single graph, but logically can be divided into entity domain and event domain horizontally, and ontology domain and instance domain vertically, forming a so-called four-quadrant architecture [15], as shown in Figure 17.

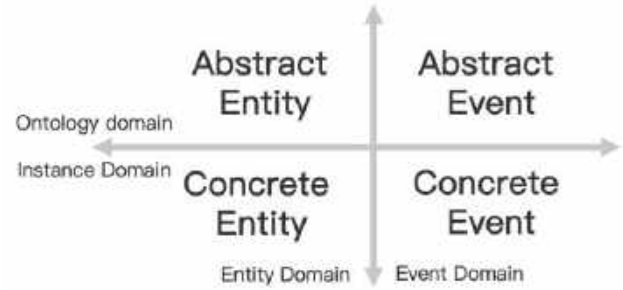

Figure 17: Four-quadrant architecture for enterprise causal knowledge graph

The main challenge is the coexistence of event models and causal models. Common event graphs only represent the relational connections between bare events without their arguments. However, in enterprise-level applications, event instances contain richer information. For example, in an enterprise risk event, it may include information about the entities involved, the industry involved, and whether the production is halted. These additional details can complement the bare events, and both aspects are mutually beneficial. We need the coexistence of event models and causal models. The event model represents a spatiotemporal multi-element hypergraph structure, while the causal layer involves reasoning about causality, succession, and logical combinations. For example, when land prices increase, it leads to an increase in fiscal revenue. The combination of "Land prices in Province A increase" is a binary relation between an administrative entity and an abstract event, and should also lead to the deduction of "Fiscal revenue in Province A increases".

The increase in fiscal revenue then cascades down the impact chain. Similarly, there are expressions of hierarchy between arguments, such as “Interest rate increase” and “Yen interest rate increase”. Ultimately, this can form a specific path of “Event → Abstract entity (superior) → Abstract entity (inferior) / Specific entity → Event”.

2.7 Consistency and propagative reasoning issues caused by logical dependencies between knowledge

In the domain knowledge graph, there are implicit logical dependencies between different properties and relations. Applications in financial risk control, for example, require the establishment of logical dependencies between property elements to construct the automatic propagation capability of risks. In the LPG model, it is necessary to prepare all relations and properties. However, inconsistencies may arise due to factors such as computational timeliness and logical correctness. These issues become more apparent when dealing with logical dependencies between multiple elements, increasing the complexity of pre-computation/construction.

2.7.1 Inconsistency and redundancy construction issues caused by logical dependencies in data

Figure 18 provides a simple example of the issue of properties errors caused by implicit logical linkage across entities in the risk mining knowledge graph.

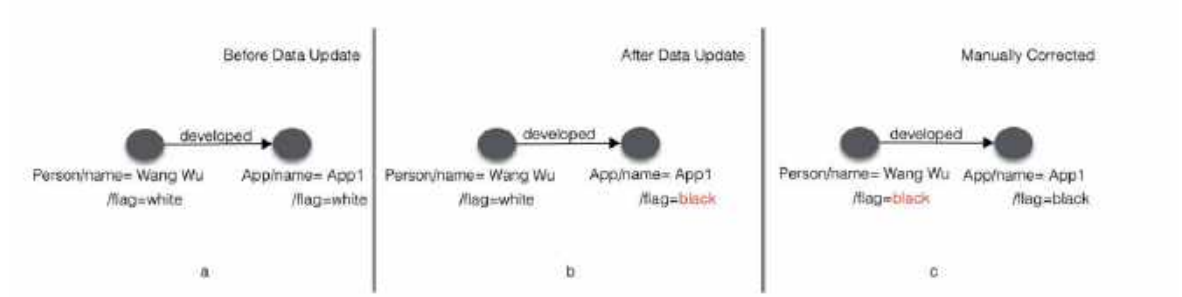

Figure 18: Implicit logical association

Looking from the perspective of discovering risky activities, a rule is defined: “When an app released by Company A is flagged as black, Company A should also be flagged as black”, as shown in Figure 18b. Both the company and the app have a “flag” property. However, when App A is reported and identified as black, Company A is still marked as white. In this case, data inconsistency occurs. It requires waiting for the completion of external system calculations before updating, as shown in Figure 18c, or manual intervention to address the issue. Incorrect data or delayed updates can result in incorrect conclusions and the knowledge graph being unavailable during the correction period.

2.7.2 Problem of obstructed risk propagation/transmission due to logical dependency transfer

In Section 2.3, the dam collapse incident at Vale in 2019 resulted in a rise in raw material prices, subsequently causing an increase in production costs for downstream companies, ultimately leading to a decline in their profits. From a causal perspective, this event originated from a production accident at a company and propagated through the industry chain, triggering financial risks for downstream companies. During the propagation process, it is not a simple diffusion of relations but rather a causal transmission with logical dependencies. Moreover, each instance in the propagation chain still retains the key elements of the initial event. These complexities are challenging to capture in an event knowledge graph based on foundational facts, as the presence of logical dependencies can hinder the propagation of events. To construct the propagation of event risks, it is necessary to consider the triggering mechanisms, the transmission of event impacts, and the rules of transmission.

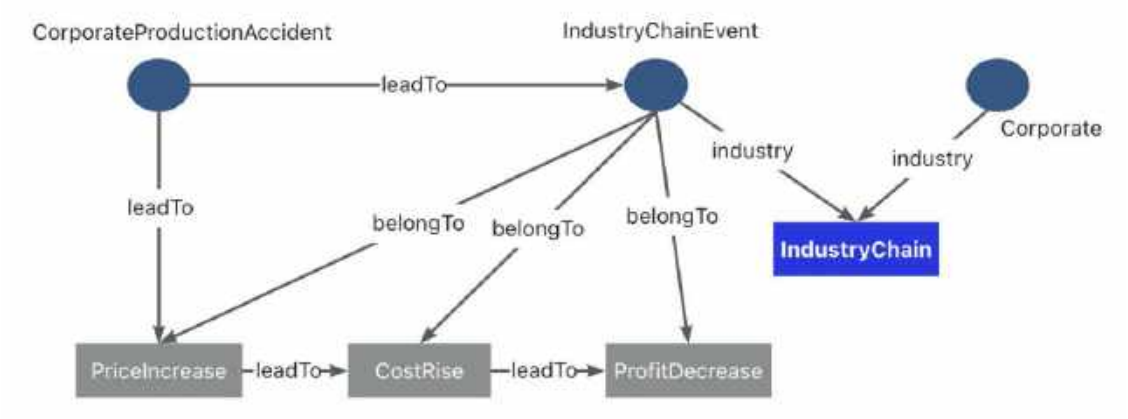

Figure 19: Impact propagation of events between instances

- **Event triggering mechanism:** Structured event elements are obtained based on the extraction of external relevant information or monitoring of the key data changes, resulting in event instances. Based on specific event instances, corresponding event propagation rules are triggered.
- **Event impact propagation:** Event impacts are propagated directly along relations. For example, in the enterprise causal knowledge graph, the impact of a company's safety production accident is propagated to the industry which it belongs to, based on the industrial characteristics of the occurrence subject in the event instance. The relation transmission of events can express which conditions allow an event to be transmitted to another target event. These conditions can utilize various properties of entities and relations obtained through query of associated subgraphs in the transmission path, such as determining whether the occurrence subject is a listed company, the industry of the company, and downstream industries.

During the propagation process, the logical judgment can reference the relevant properties of all preceding entities/relations in the current judgment condition's position. As shown in Figure 20, the “Price increase” event needs to reference the industry property of the subject in the “Vale dam collapse incident”,

the “Cost increase” event needs to reference the downstream industry property of the “Price increase” event, and the “Profit decrease” event needs to reference the industry property of the “Cost increase” event.

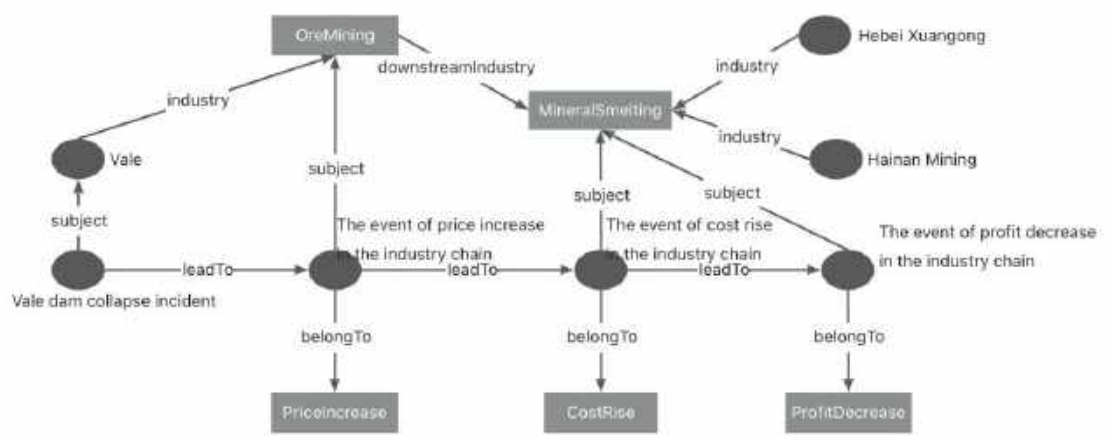

Figure 20: Concept induction and impact propagation based on rules

2.8 Graph Construction and Evolution Problems for Incomplete Data Sets

The construction of enterprise-level knowledge graph is often based on incomplete data sets, with constantly changing sources and construction strategies. Continuous iterations are needed to improve coverage, accuracy, and reduce conflicts and errors. This incompleteness typically includes two aspects: the heterogeneity of data sources and the heterogeneity across multiple knowledge graphs. The heterogeneity of data sources manifests as different instances and properties of the same entity type coming from different data sources. It requires addressing disambiguation, alignment, and fusion of different data sources, as well as evaluating and selecting data sources based on different confidence strategies to achieve traceability and quantifiability. The heterogeneity across multiple knowledge graphs arises from the presence of duplicate definitions of the same entity type in different domain knowledge graphs, which need to be merged and linked across knowledge graph based on business domain requirements and data differences.

2.8.1 Reliable Fusion and Trustable Traceability of Heterogeneous Data from Multiple Sources in Graph Construction

In enterprise knowledge graph applications, different properties and relations of the same entity type may come from different data sources. The common practice to construct entities based on heterogeneous data sources is entity linking and entity resolution. Entity linking involves finding an accurate and unique entity ID for each data update, while entity resolution merges the updated properties and achieves the consolidation of the properties and relations. As shown in Figure 21, in the enterprise causal knowledge graph, the construction process of the enterprise entities, involves the merging of various data sources, such as company announcement extraction, basic business information, and court announcements.

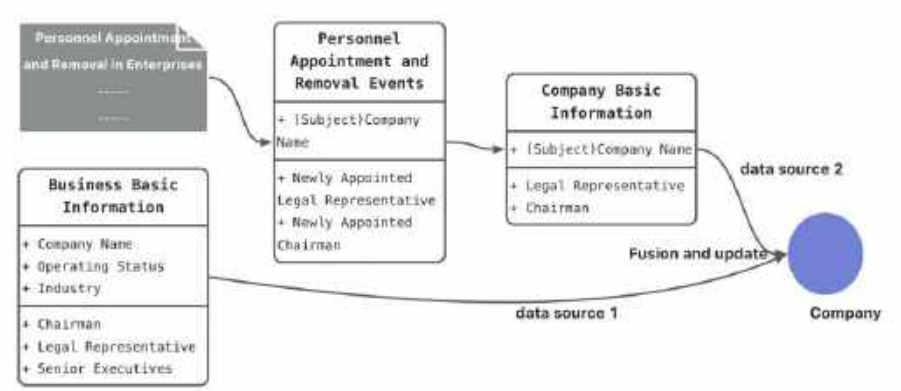

Figure 21: Entity updates based on heterogeneous data sources

The definition of an enterprise entity is generally represented as "<Company, legalPerson, String>". In practical applications, when conflicts occur in property values, decisions on how to retain or update them need to be made based on dimensions such as source type (sourceType) and algorithm prediction scores (score). For example, the confidence level of basic business information is the highest, so it needs to be unconditionally overridden. However, the timeliness of updates for business information and company announcements may not be consistent. There may be cases where company announcements have been captured but the business information has not been synchronized. In such cases, secondary descriptive information needs to be preserved on property elements. This can be formally represented as: "<Company, legalPerson, String>" as p, with the addition of "p.sourceType", "p.score", and the recording of the coverage rules for p in the schema. For example, p.fuseRule = "sourceType == 'business information'; score > p.score".

2.8.2 Entity Alignment, Real-time Updates, and Fusion/Traceability Problems in Cross-Graph Fusion

The problem of cross-graph fusion is similar to 2.5.2. When merging user entities from the risk mining knowledge graph and the fund knowledge graph, it is necessary to determine how to preserve the properties and relations in the new "FuseEntityType".

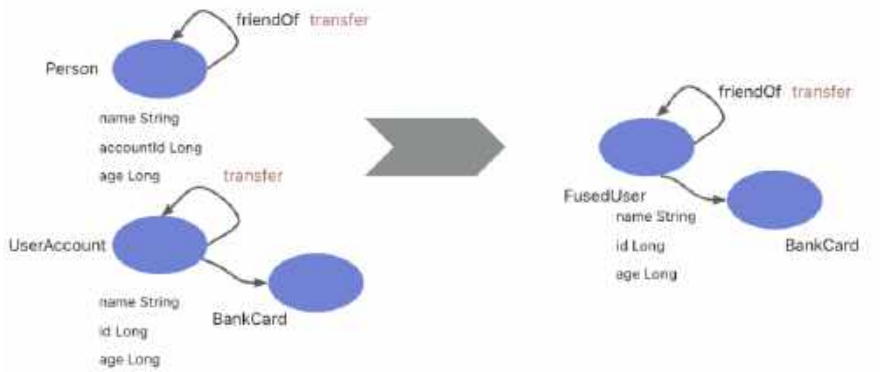

Figure 22: Stable fusion and traceable update problems in cross-graph fusion

In order to ensure that updates to properties/relations of "Person" and "UserAccount" can trigger timely updates to "FusedUser", while forwardly ensuring the stability of the results and backwardly supporting interpretability and traceability of the results, we need to extend properties and relations, and record supplementary properties for fusion and update strategies. These strategies are then executed during the entity update stage. For the iterative evolution of knowledge graph based on incomplete data sets, the knowledge modeling framework needs to address the following problems:

- The properties/relations can carry supplementary properties: These supplementary properties are used to describe the source, confidence, relevance, author, and other relevant asset information of the properties/relations.
- The properties/relations can define update strategies: Support for executable rule expressions is needed to define selection and prioritization strategies for the properties/relations. This ensures the stability of results even when data from the different sources arrive randomly.
- The entity types can be bound to entity linking operators: In industrial-level applications, many data sources do not provide standardized IDs. Therefore, we need to use entity linking strategies such as text matching and spatiotemporal clustering to find the target entity ID. Support for binding entity linking operators to target entity types is needed to ensure the execution of the same entity linking operator when different source data updates occur, thus ensuring the result is stability.

2.9 Summary of Problems with Semantic-less, Non-programmable Labeled Property Graph

Firstly, knowledge management is associated with the entire lifecycle of the business, requiring the ability to evolve iteratively and support continuous business iteration while effectively avoiding combinatorial explosion and duplicate construction. Secondly, knowledge management faces the complex problem of modeling knowledge from incomplete data sets, heterogeneous data sources, and multiple business expert perspectives. This requires the ability to implement differentiated perspectives and lightweight alignment of heterogeneous data sources through programmable paradigms, thereby reducing system complexity. Lastly, knowledge management needs to establish necessary knowledge hierarchies and classification systems to achieve effective linkage, induction, and deduction between different levels. This enables automatic extraction of static common knowledge to support efficient cross-business reuse and effective accumulation of core assets. The following chapter 3 and 4 will provide detailed explanations of the semantics-enhanced programmable framework (SPG).

Chapter 3 Semantic Enhancement Programmable Framework (SPG)

To address the issues mentioned in Chapter 2, we have developed the semantic representation framework (SPG) based on property graph, taking into account the characteristics of enterprise-level business scenarios. This framework defines and represents knowledge semantics from three aspects. Firstly, SPG provides a formalized representation and programmable framework for "knowledge" to be defined, programmed, and understood by machines. Secondly, SPG enables compatibility and progression between knowledge hierarchies, supporting the construction and continuous iterative evolution of the knowledge graph in industrial scenarios with incomplete data. Lastly, SPG effectively bridges the gap between big data and AI technology systems, enabling efficient knowledge transformation of massive data to enhance data value and application value. With the SPG framework, we can construct and manage knowledge more efficiently, while better supporting business needs and application scenarios. Due to its scalability and flexibility, the SPG framework allows for quick construction of domain models and solutions for new business scenarios by extending domain knowledge models and developing new operators.

3.1 The semantic model of SPG

The overall semantic model of SPG is illustrated in Figure 5 of the Chapter 1, and briefly described in Chapter 1 as well. Firstly, SPG formally defines knowledge from the following three dimensions:

1) **Domain Type and Structure Constraint:** In the objective world, there are no things without domain types. However, in the digital world, there are numerous text/numeric representations without domain types. SPG DC requires that everything must have a distinct domain type (Class) and the domain type must have its own inherent structural representation, including properties, relations, etc., which are associated with other things through relations. Additionally, based on the principles of dynamic to static, specific to general, and instance to concept in domain knowledge, SPG DC classifies domain types into Event HyperGraph, Entity, and Concept. This facilitates efficient knowledge classification and reuse in business and achieves automatic hierarchical separation of knowledge from dynamic to static. For detailed information, please refer to the descriptions in sections 4.1.1 and 4.2.

2) **Unique Instance within the Domain:** In the objective world, there are no two things that are exactly the same. However, in the digital world, there are numerous instances of the same thing due to data copying, different descriptive perspectives, and multiple heterogeneous sources. To ensure consistent representation between the digital world and the objective world, SPG requires that every instance within a domain type must be unique to guarantee the accuracy and consistency of the knowledge. To achieve this, SPG Evolving provides programmable capabilities for entity linking, property standardization, and entity resolution through SPG-Programming. Users can use built-in or self-developed algorithms (operators) to improve the uniqueness of instances. More

detailed descriptions are expected to be released in the SPG White Paper 2.0, and relevant information can be found in Chapter 7.2: SPG-Programming.

3) **Logical Dependency between Knowledge:** In the objective world, there are no things that are not related to other things. We often understand things through their connections with other things. These connections represent the intrinsic characteristics of things as well as the logical/physical relations with other things. They encompass both the general commonality of inductive significance and the specific uniqueness at the instance level. SPG defines dependencies between knowledge through the SPG Reasoning predicate/logic system, including logical dependencies and inference between properties, relations, types, etc. Additionally, SPG defines basic predicate primitives through the predicate system to support knowledge reasoning and inference. This allows for better handling the associations and dependencies between knowledge, and supports modeling and analysis of the complex business scenarios. Detailed descriptions can be found in sections 4.3 and 4.4.

Furthermore, the SPG framework achieves compatibility and progression between knowledge hierarchies to adapt to industrial-level knowledge graph. In practical applications, businesses often face the objective reality of incomplete datasets, incomplete expert experiences, and incomplete understanding of the knowledge graph. On one hand, businesses expect to quickly realize business value through the knowledge graph. On the other hand, the coverage of business data and the experience of the knowledge graph are also incomplete, requiring continuous business iterations to gradually deepen the understanding and application of the knowledge graph. However, RDF/OWL requires complete knowledge exchange, which is inconsistent with the objective reality of practical application scenarios. To address this issue, SPG requires compatibility and progression from left to right when defining knowledge representation. Users can choose the simplest SPG Compatible mode to directly construct the representation of property graph from the big data system, or they can enhance the semantic clarity of the subject model by adding SPG DC domain model constraints. Additionally, users can continuously improve the uniqueness of subjects and the semantic associations between subjects by adding entity linking and entity resolution operators through SPG Evolving. Finally, a symbolic representation of knowledge is constructed through complex predicate and logic systems. Through the layered compatibility and progression of SPG, the cost of implementing knowledge graph business can be greatly reduced. In the process of knowledge graph application, users can gradually improve and optimize the domain knowledge graph by selecting different modes and operators based on their own needs and data conditions.

In conclusion, the SPG framework effectively bridges the gap between big data architecture and knowledge systems, enabling the automatic construction of knowledge systems from big data systems. Specifically, by employing the ER2SPG approach to transform data from the big data system into the SPG knowledge graph representation, seamless integration of data and knowledge can be achieved. Furthermore, the SPG-Reasoning component enables the construction of a machine-understandable symbolic system, which facilitates the linkage with deep learning models through knowledge constraints, logical symbols, etc.,

thereby providing more possibilities for knowledge graph applications. Additionally, the SPG framework aims to establish a symbolic linkage with large language models (LLMs) through SPG-Reasoning. By mapping the output of LLMs into the symbolic representations and inputting the symbolic representation of the knowledge graph into the LLMs, better integration and collaboration between knowledge and models can be achieved, enabling efficient interaction and co-evolution between knowledge and models. This is of great significance for achieving more intelligent application scenarios.

In summary, the SPG framework enables the automatic transformation and application of data into knowledge by bridging the big data architecture and constructing a machine-understandable symbolic system. In the future, the SPG framework will continue to leverage its advantages, explore more application scenarios, and establish closer linkage with LLMs, bringing more possibilities to knowledge graph applications.

3.2 SPG Layered Architecture

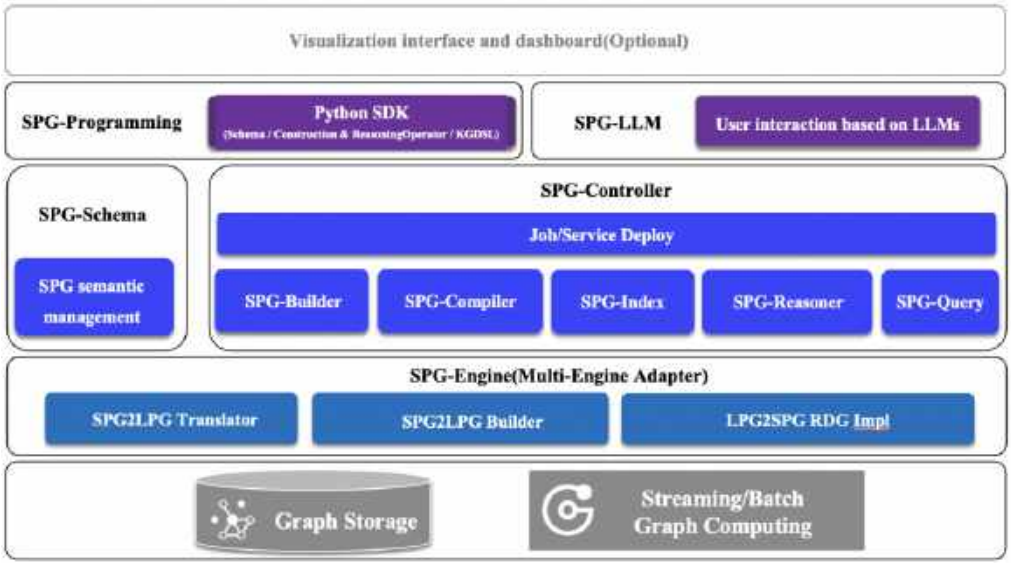

Figure 23: Overall Architecture of the Knowledge Engine based on SPG

The core objective of SPG is to build a standardized knowledge engine architecture based on SPG, providing clear semantic representation, logical rule definition, operator framework (construction, reasoning), etc., for the domain knowledge graph construction. It supports pluggable adaptation of the basic engines, algorithm services, and the solution construction by various vendors. This section provides a brief overview of the overall framework.

- SPG-LLM: Responsible for the interaction subsystem with LLMs (Large Language Models), such as natural language understanding (NL), user instructions, queries, etc. See Chapter 8 for more details.
- SPG-Schema: Responsible for the design of the Schema framework that enhances the semantic understanding of the property graph, including subject models, evolution models, predicate models, etc. See Chapter 4 for more details.

- SPG-Controller: Responsible for the design of the control center subsystem, including control framework, command distribution, plugin integration, etc. See Chapter 6 for more details.
- SPG-Programming: A programmable framework subsystem responsible for the design of the SDK framework and compilation submodules, such as knowledge construction, knowledge evolution, expert experience projection, knowledge graph reasoning, etc. See Chapter 7 for more details.
- SPG-Engine: Knowledge graph engine subsystem responsible for the design of the integration/adaptation layer for multiple engines, such as reasoning engine, query engine, etc. See Chapter 5 for more details.

3.3 The Objectives of SPG

We aim to build a next-generation cognitive engine infrastructure based on SPG, as shown in Figure 24, which represents the overall capabilities. The legend in the figure also indicates the coverage of this whitepaper.

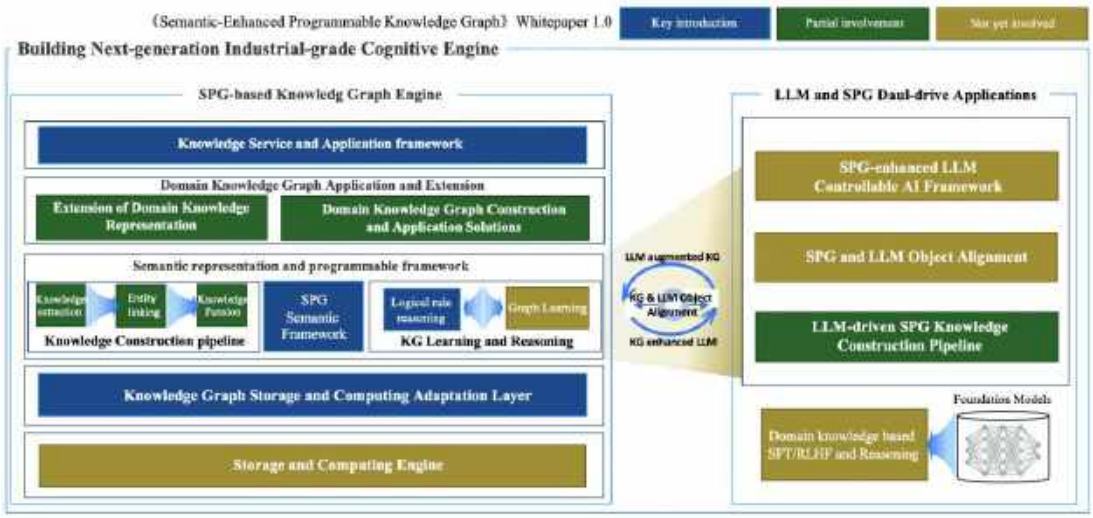

Figure 24: Target Architecture of SPG and LLMs Bidi-Driven (Draft)

This whitepaper, titled "Semantic-Enhanced Programmable Knowledge Graph (SPG) 1.0", is the initial release. It discusses the current pain points and possible solutions in the development of knowledge graph, as well as the proposed approach, core capabilities, and overall framework of SPG, as described in Chapter 2. In the future, SPG will continue to improve the content of the whitepaper, including domain model extensions, programmable framework, knowledge construction engine, knowledge reasoning engine, and the bidirectional interaction between LLMs and KG. Additionally, SPG will accelerate the open-source development of semantic and basic engine frameworks, promoting the industrial implementation of knowledge graph. The 1.0 version of the whitepaper will focus on the following topics:

- **SPG Semantic Foundation Framework:** Introduces the background of SPG, the core problems it addresses, and presents the semantic framework and schema model of SPG through two business cases.

- **SPG Logical Rule Framework:** Introduces the logical rule system of SPG and how it organically integrates logical rules with factual knowledge based on SPG.
- **SPG Multi-Engine Adaptation Layer:** Provides a detailed introduction to the capabilities of the adaptation layer, incorporating the adaptation abstractions of SPG2LPG and LPG2SPG, to facilitate the efficient integration of the graph storage and the graph computing engines developed by various vendors.

We will continue to update the whitepaper, including versions 2.0 and 3.0. In this release, certain topics, such as the programmable framework and knowledge reasoning, have only been briefly introduced. In the future, we will focus on these topics in separate discussions. We will also continue to explore and make breakthroughs in the bidirectional interaction between SPG and LLMs. The release plan for the future of SPG is outlined in the Chapter 11.

Chapter 4 SPG-Schema Layer

4.1 Overall Architecture of the SPG-Schema

The core objective of SPG is to leverage the advantages of the property graph compatibility with the big data architecture, and address practical problems in industrial practice to achieve semantic enhancement and build a comprehensive semantic system. This chapter provides a detailed explanation of two aspects: the extension of the SPG DC subject classification model and the extension of semantic predicates in SPG Reasoning. First, extending the subject model based on the definition of schemas or fields in the big data tables is the most direct and flexible approach. It involves mapping the columns or fields of the table model to the types, properties, and relations of the SPG subject model. This mapping allows for the integration of data from multiple heterogeneous sources into an incomplete subject structure. Next, the iterative evolution of the incomplete subject structure is carried out to achieve the extension of the logical predicate semantics. In this process, SPG draws inspiration from the minimal usable set of ρdf and the logical predicate capabilities of OWL. It defines the minimal semantic units of the SPG subject model and expands the expression of SPG in terms of the predicate semantics and the logical rules.

4.1.1 Extension of the Subject Classification Model

To enhance the semantic expression of the node types in LPG, SPG extends and introduces additional subject classification models on the node types and edge types in LPG. This expansion aims to accommodate a more diverse representation of knowledge. The expanded subject types include standard types, concept types, entity types, event types, and more. The domain classification model of SPG is shown in Figure 25.

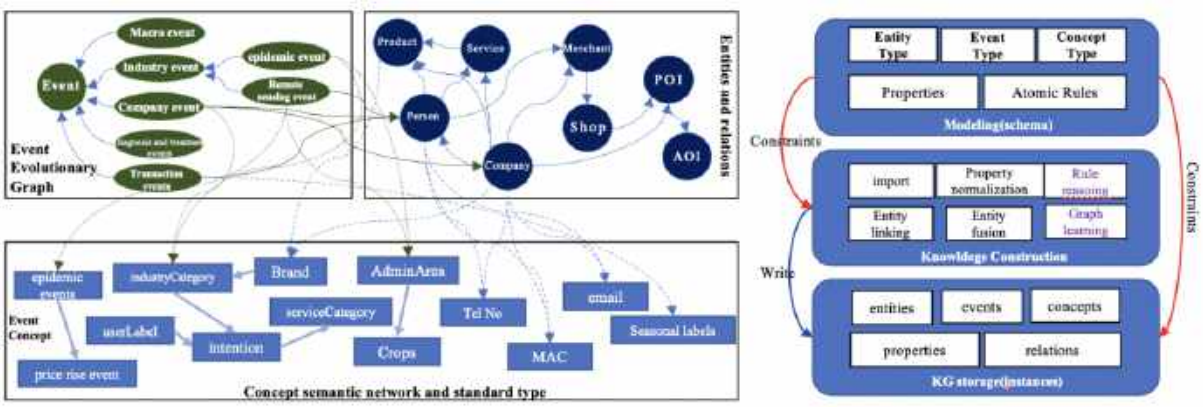

Figure 25: SPG Domain Classification Model

A brief explanation of the SPG subject classification model is as follows:

- **Entity:** Objective objects with strong business relevance, represented by a complex structure characterized by multiple properties and relations, such as users, companies, merchants, etc.
- **Concept:** Abstractions from concrete entities to general ones, representing a group of entity instances or event instances, forming a classification system. Concepts are relatively static and

represent common knowledge with strong reusability, such as audience labels, event classifications, administrative divisions, etc. To simplify the enterprise applications, standard types are also included in the concept category.

- **Event:** Temporal and spatial multi-dimensional types with constraints (such as time and space). For example, the industry events, company events, medical events extracted through NLP, CV, or the user behavior events generated from actions like purchasing, redeeming, registering, etc.
- **Property:** Properties are the components of the entities, events, concepts, etc., used to describe the individual elements of a complex structure. Each property element can be associated with a specific simple or complex structure, such as base types, standard types, concept types, etc.
- **Relation:** Relations are defined similarly to the properties, and express the association between a complex object and the other objects. The difference between the relations and the properties is that relations involve entity types as the associated objects.

1. Entity Types

Entity types are the basic unit of types in SPG. They are composite data types composed of multiple properties and relations, and are directly extended from the Node types in LPG. In the Chapter 2, we conducted an in-depth analysis of the challenges currently faced in knowledge management with LPG. To address the high data preparation costs and the lack of semantic capabilities in property type in LPG, SPG-Schema extends the expression of property value types on the LPG Node types. The type of the value can be standard type, concept type, entity type, etc. In order to achieve inheritance and reuse of the entity types, we draw inspiration from and extend the semantic of the "subClassOf" predicate to enable subclass inheritance of properties and relations from the parent classes. To address the issue of inconsistent naming of the same entity type due to heterogeneity, we support the fusion of the entity types and align the logical alignment of the entity types in different knowledge graphs through entity linking and entity resolution operators. In the future, we will focus on releasing the operator binding section in the SPG Whitepaper 2.0.

2. Concept types and Event types

In the schema of the domain knowledge graph, there is a subjective design issue due to different internal demands in different business domains. This leads to the existence of multiple similar types for the same entity, as different businesses may have different naming and granularity requirements for these types. However, the data for these similar types all come from the same source, which severely affects and hinders knowledge dissemination and causes inconsistencies between knowledge and data. To avoid such inconsistencies, SPG-Schema introduces concept types to classify the similar types and resolves knowledge heterogeneity by linking concepts with basic entity types.

In addition, the event model in the enterprise causal knowledge graph involves multiple dimensions of time and space. When modeling the causality layer, it is necessary to associate simple or complex logics such as causality, sequence, co-occurrence, and structure. To address the inability of LPG to perfectly

express these requirements, SPG-Schema introduces the concept of events to extend the classification model and better express the horizontal and vertical associations between the events, entities, and concepts.

The Concept-Event Quadrant Diagram, as shown in Figure 26, describes the associations between the entity types, concepts, and events based on the principles of domain knowledge transitioning from dynamic to static, from specific to general, and from instances to concepts. At a more specific level of definition and instantiation, the Concept-Event Quadrant Diagram can be divided into four components: abstract entities, concrete entities, abstract events, and concrete events. Physically, they are interconnected and form a unified graph.

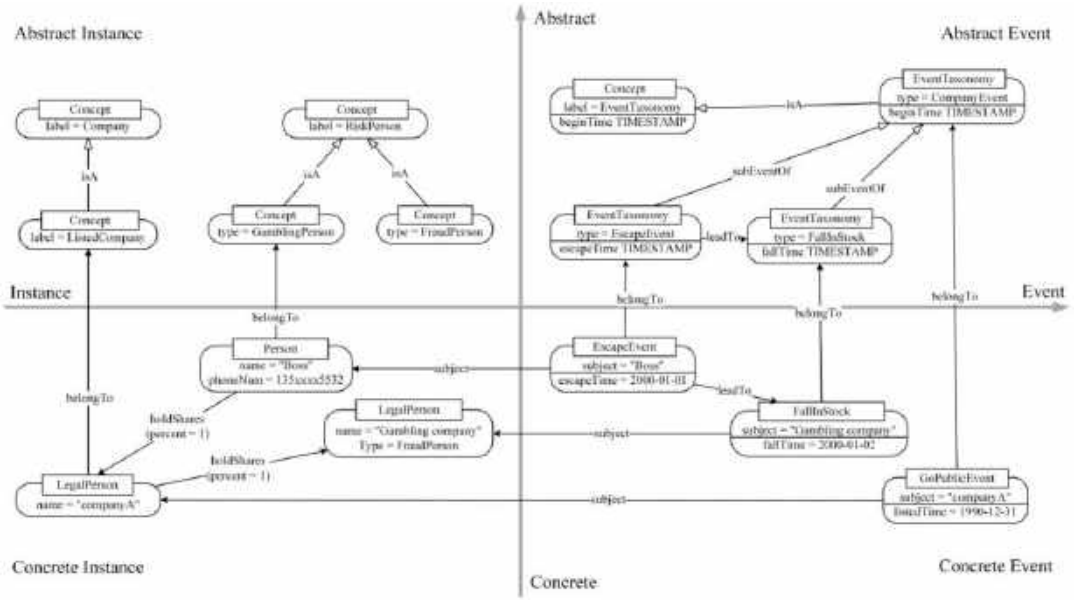

Figure 26: The Concept-Event Quadrant Diagram

Specifically, the quadrants are divided horizontally into the entity domain and the event domain, and vertically into the abstract domain and the concrete domain. The vertical division can also be referred to as the concept domain and the instance domain. The abstract entity type represents the abstract concept of the concrete entity types, while the concrete entity refers to the specific instantiation of the entity types. The abstract event represents the abstract concept of the events to express causality and sequence between the events, while the concrete event corresponds to the specific instantiation of the event types.

3. Standard Properties

In the RDF/OWL model, each entity, relation, and property needs to be modeled independently. While the syntax elements of the property graph are simpler compared to RDF, they are merely declarations of data structures and cannot effectively utilize property knowledge for knowledge dissemination. In the actual business implementation process, it is often necessary to use the properties for knowledge dissemination and analysis. However, in LPG, the properties without explicit domain type constraints are simply literal texts or numbers. This not only fails to ensure the integrity and correctness of the property, but also makes it difficult to implement effective queries and propagation based on the structure. Leads to the additional modeling for the properties and significant data preparation costs due to the variations in property scales. As

the demand for knowledge graph-based association analysis increases, these costs are expected to rise even further.

To balance the tradeoff between RDF and LPG regarding properties and effectively reduce data preparation costs, SPG-Schema introduces standard property types to simplify data dependencies. The application of standard properties can automatically materialize textual properties into relations, increasing the ability to propagate knowledge and implicit associations. Since the standard properties are used instead of the relation modeling, explicit definition of the relations is not required. The relation propagation between entity types is achieved through the semantic propagation of the standard properties.

4.1.2 Expansion of the Semantic and Rule Reasoning Capability

In the general modeling process, LPG only consists of two elements: Node and Edge. However, the properties of these elements are often in the form of text/strings, which can lead to various issues in practical business scenarios. In order to achieve more efficient knowledge propagation and inference on top of the expanded 5-category classification model, SPG-Schema introduces a series of semantic predicates to constrain LPG, enabling more syntax and semantics. The specific semantic syntax hierarchy diagram can be seen in Figure 27.

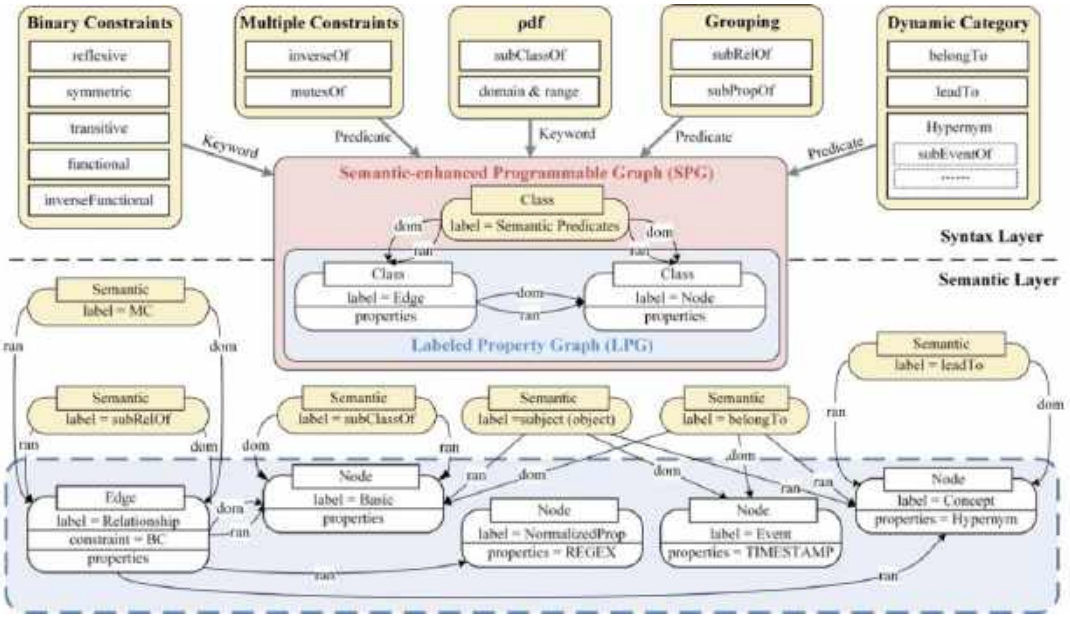

Figure 27: SPG-Schema Syntax and Semantic Hierarchy Diagram

- Syntax Layer: In this layer, the syntax content of SPG-Schema is defined. The related syntax used for semantic reasoning in SPG-Schema can be divided into five categories: minimal constraint set (ρdf), binary constraints (BC), multiple constraints (MC), relation group constraints, and dynamic types. These can be applied to SPG-Schema in the form of Keywords and built-in Predicates from the standard namespace std, depending on the usage scenario.

- Semantic Layer: In this layer, the specific domain (dom) and range (ran) of the semantic reasoning capabilities in SPG-Schema are defined. This reflects the association between the built-in predicates of the reasoning rules and the refined entity classification model.

4.1.3 The Four-Layer Architecture of SPG-Schema

For the overall framework of SPG-Schema, starting from four basic contents, the requirements are gradually expanded and decomposed to determine the content included in SPG-Schema Core based on the semantic completeness and the practical industrial needs. Lightweight syntax is used as much as possible to avoid high complexity, ensuring that the complexity of SPG-Schema does not exceed PTIME and guaranteeing efficiency in industrial-level implementation. The balance between semantic complexity and business application cost is achieved.

The MOF architecture is a layered metadata architecture, and based on this structure, we can also divide the overall modeling hierarchy of SPG-Schema into four layers, as shown in Figure 28.

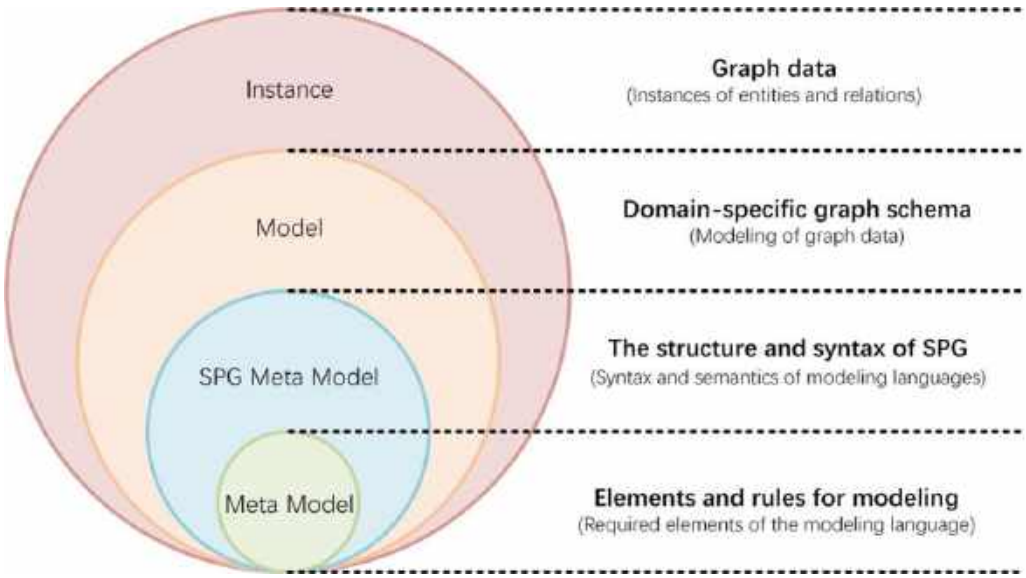

Figure 28: The Four-Layer Architecture of SPG-Schema

Based on the above summary, combined with the risk mining knowledge graph and the enterprise causal knowledge graph as the basic scenarios, the overall four-layer architecture of SPG-Schema is shown in Figure 29.

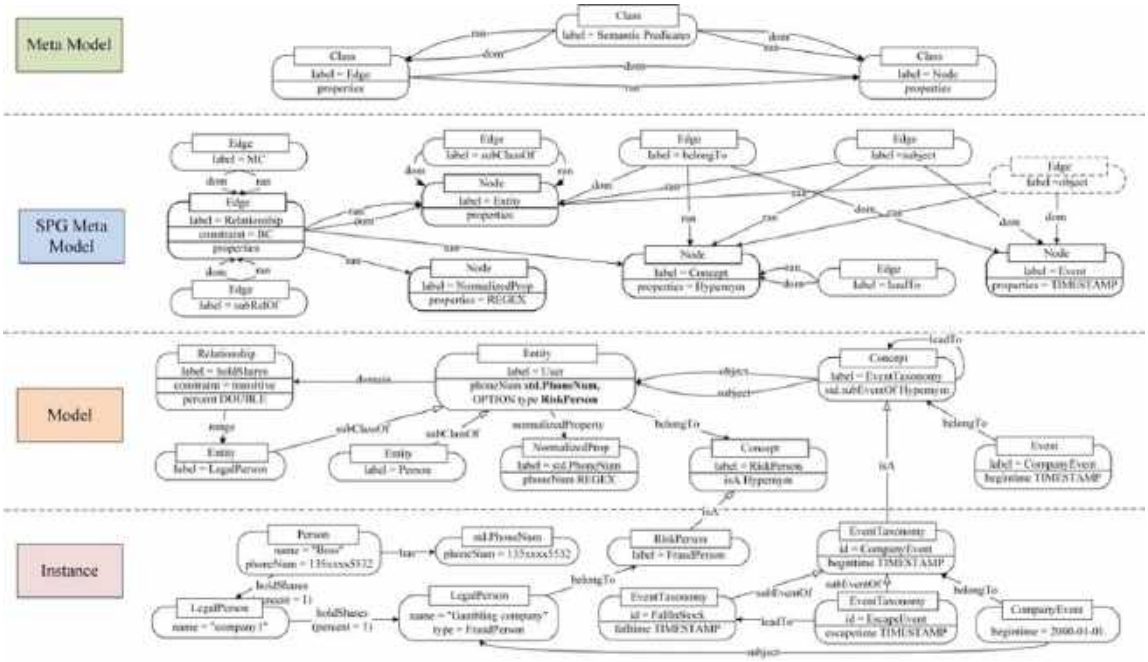

Figure 29: SPG-Schema Four-Layer Architecture Diagram

The explanation of the four-layer architecture is as follows:

- Meta Model: This layer provides an overall definition of the SPG Meta Model paradigm. It defines the elements needed for modeling with SPG-Schema, but it is transparent to general users.
- SPG Meta Model: This layer instantiates the Meta Model layer and provides a more detailed classification of types and predicates. It defines the structure and syntax of the modeling language, adds more semantics to the modeling in the Model layer.
- Model: This layer defines a specific system model. Users can define the classes, properties, relations, and their semantics that can be perceived by the Instance layer using the syntax. The data stored in the Model layer represents the modeling in the Instance layer, where the data is the instance of the model in the Model layer.
- Instance: SPG-Schema adopts the Instance-Class separation paradigm, where each Instance must strictly adhere to the constraints defined in the Schema of the Model layer. This layer is the largest and most concrete layer. The contents of the Instance layer are specific instances.

4.2 Semantic Enhancement of Nodes and Edges

The main syntax design of the SPG-Schema extension is achieved by introducing additional keywords based on the PG-TYPES [16]. Therefore, the main semantics of SPG-Schema are also divided into three categories: NodeType, EdgeType, and GraphType, which correspond to the definition of nodes (Node), edges (Edge), and views in LPG. The basic definition of the semantics is located in the SPG Meta Model layer, and it adds more related semantics to the Meta Model layer by classifying the LPG node types more finely.

4.2.1 Syntax and Semantics for the NodeType

Based on the definition in section 4.1.1, we have classified the commonly used node types in the knowledge graph into entity type, standard type, concept type, and event type. we will introduce the basic syntax and semantics for these four types.

1. Entity Type

Using the syntax "CREATE ENTITY TYPE", you can define a node type (Class) and the labels and property types that appear in the node type. For example, you can create a "User" class using the following syntax:

```
// Definition of the User entity
CREATE ENTITY TYPE ( User {
  phoneNum std.PhoneNum,
  OPTIONAL taxonomy RiskPerson,
  OPEN
});
```

In the above example, the User type is defined with two properties: "phoneNum" and "taxonomy". The type of "phoneNum" is "std.PhoneNum", and the type of "taxonomy" is "RiskPerson". The OPTIONAL keyword indicates that the "taxonomy" property is optional. The OPEN keyword indicates that when defining the model and populating instance data, additional property fields can be added.

In some cases, it may also be necessary to declare a type as an abstract type, which means that it cannot be directly instantiated. In the example above, the "User" type can be annotated as an abstract entity type using the ABSTRACT keyword. Therefore, when populating instance data, it cannot be directly populated under this type, but rather in the subtypes that inherit from this abstract parent class. This type may not be very useful, but it can be helpful for reusing shared properties among multiple subclasses.

In addition to the above three keywords, we can add more semantic information and constraints to the node definitions by referring to the keyword constraints in PG-Keys [17]. The keywords EXCLUSIVE, MANDATORY, and SINGLETON are used to represent unique, at least, and at most constraints, respectively. This further enhances the semantic constraints on entity properties in SPG-Schema. In the following example, we will modify the "User" type to be an abstract type and apply the three property constraint keywords.

```
// Extend the definition of the User type as an abstract type
CREATE ENTITY TYPE ABSTRACT ( User {
  EXCLUSIVE idcard STRING,                    // Each instance should have a unique ID card number
  MANDATORY name STRING,                      // Each instance should have at least one name
  SINGLETON birthday DATE,                    // Each instance can have at most one birthday
  OPTIONAL phoneNum std.PhoneNum,             // Optional field to add phone number with standard property
  OPTIONAL SINGLETON taxonomy RiskPerson,     // Optional concept classification property
  OPEN
});
```

When the three keywords (EXCLUSIVE, MANDATORY, and SINGLETON) are used together with the OPTIONAL keyword in type constraints, the former is used to constrain the data filling in the instance layer, while the latter corresponds to whether the instance can use that property. They are not conflict with each other. For example, “OPTIONAL SINGLETON type RiskPerson” can be used to indicate that the data may not include the concept classification property “taxonomy”, but once this property is added, the instance can have only one classification property.

2. Standard Types

To distinguish the standard types from the user-defined types, the concept of namespaces is introduced, with the “std” namespace representing the standard types. All the standard types are defined within the “std” namespace and created using regular expressions. Since standard types are more associated with property standardization and are derived from property, they typically only have one property. After defining a standard property using the syntax “CREATE NORMALIZED TYPE”, you can use it directly or use the “SET PROP” statement and “NORMALIZED” to normalize the properties. It is important to note that standard properties should be used in conjunction with regular expressions guided by the “REGEX” keyword to constrain the pattern/format of the property values.

```
// Definition of the standard type Email
CREATE NOMALIZED TYPE (std.Email {
  value STRING REGEX '[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,3}'
});
// Definition of standard type phoneNum
CREATE NOMALIZED TYPE (std.PhoneNum {
  value STRING REGEX '/^(13[0-9]|14[01456879]|15[0-35-9]|16[2567]|17[0-8]|18[0-9]|19[0-35-9])\d{8}$/'
});
// Modify the phoneNum property in the User class to be a standard property
SET PROP (User.phoneNum) NORMALIZED std.PhoneNum;
```

Standard types are also one of the most important aspects of property standardization in SPG-Schema. Unlike regular LPG properties, when the property in SPG is defined as a standard type, and the semantic of that standard type is propagable, the property will be automatically converted into a relation, and the value of the property will be treated as an instance of the standard type. This generates more meaningful information for semantic reasoning. In this example, we modify the “phoneNum” property of the “User” type to be a pre-defined standard property “std.PhoneNum”. Additionally, users can directly specify properties as standard types when creating node types.

3. Concept Types

When creating a concept type, you can use the “OPTIONAL” keyword to mark certain properties as optional, while the remaining properties are considered mandatory by default. By using this type, you can associate entitys, properties, and other elements with relevant business conceptual domains, enabling further business inference.

```
// Definition of the concept of company classification
CREATE CONCEPT TYPE (CompanyTaxonomy {
```

```
  isA std.Hypernym,
  OPTIONAL beginTime TIMESTAMP,
  OPEN
});
```

In the example above, a concept type called “CompanyTaxonomy” is created as a conceptual domain. When populating concept instances, they will categorized to this concept type. The property “isA”, of type “std.Hypernym”, is a mandatory property within this conceptual domain, representing the hierarchical relation between concept instances in the domain. The field “begintime” is an optional property. More details about the “std.Hypernym” type will be discussed in section 4.3.5 with further elaboration.

4. Event Types

When creating an event type, there are mandatory and optional requirements for time, subject, and object. The “REQUIRED” keyword is used to indicate mandatory requirements for the fields, while the remaining fields are considered optional. It is important to note that event types must have a timestamp property and a specified subject type by default. Therefore, we can define an event using the following syntax.

```
// Definition of the company operation event
CREATE EVENT TYPE (CompanyEvent {
  {REQUIRED occurrenceTime TIMESTAMP, OPEN}
  REQUIRED SUBJECT (Company | Person),
  OBJECT (Company | Person)
});
```

In the above example, we define a company operation event. It includes a mandatory timestamp property called “occurrenceTime”, and a mandatory subject type of either “Company” or “Person”. An event should be treated as a graph structure. For example, in the defined company operation event called “CompanyEvent”, it automatically uses the built-in predicate “std.subject”, indicated by the “SUBJECT” keyword, to point to the Company and Person types as the possible subjects of the event. This is a mandatory property, while the “OBJECT” can be empty.

4.2.2 Syntax and Semantics of the EdgeType

The semantics of EdgeType specify the labels, properties, and types of the property values that appear in an edge type. It also specifies the allowed source and target entity types. When using the “CREATE EDGE TYPE” statement to create a relation, it is required that both the source and target entity types have been defined. Otherwise, there will be a dangling mount situation, resulting in a relation error, and the creation will not be allowed.

```
// Definition of the holdShares relation
CREATE EDGE TYPE
(Person)-[holdShares {percent DOUBLE}]->(LegalPerson);
```

In many cases, users may not want to use the triple representation when using the relation. Therefore, we can set relation aliases to directly refer to a triple relation. There are two ways to set aliases: direct

definition and later modification. For the newly defined “holdShares” relation, the former sets an alias directly in the definition using the AS “< >” statement, while the latter uses the “ALTER” keyword to add an alias to the relation that has been forgotten to be set.

```
// Specify the alias "holdSharesType" when defining the relation
CREATE EDGE TYPE
(Person)-[holdShares {percent DOUBLE}]->(LegalPerson)
AS <holdSharesType>;

// Use ALTER to set an alias for the above holdShares relation
ALTER EDGE TYPE
(Person)-[holdShares {percent DOUBLE}]->(LegalPerson)
AS <holdSharesType>;
```

In the definition of an edge type, the “< >” notation is used for the first time to represent the alias of the edge. In SPG-Schema, we use “< >” to quickly refer to a graph. For relation, it essentially represents a graph type composed of node-edge-node relation. In addition to the basic definition, similar to the property constraints mentioned above, the “EXCLUSIVE” keyword can also be applied to relation constraints, which we refer to as cross-type constraints. Since there has a unique source node and target node for each relation instance by default, the “MANDATORY” and “SINGLETON” keywords are not used to constrain relations.

```
// Applied to the source node => an entity cannot simultaneously have multiple outgoing edges of the same relation type.
CREATE EDGE TYPE (EXCLUSIVE Class1)-[ Type { propClause }]->(Class2);

// Applied to the target node => an entity cannot simultaneously have multiple incoming edges of the same relation type.
CREATE EDGE TYPE (Class1)-[ Type { propClause }]->(EXCLUSIVE Class2);
```

By constraining the source and target nodes of an edge, we can easily achieve relation constraints similar to those in relational databases.

```
// One-to-one
CREATE EDGE TYPE (EXCLUSIVE Class1)-[ Type { propClause }]->(EXCLUSIVE Class2);

// one-to-many,
CREATE EDGE TYPE (EXCLUSIVE Class1)-[ Type { propClause }]->(Class2);

// many-to-many
CREATE EDGE TYPE (Class1)-[ Type { propClause }]->(Class2);
```

In addition, we have also added a specific type of constraint called Binary Constraints (BC) to the relation type. These constraints specify the characteristics shared by all individual instances of a specific relation type. We will focus on introducing these constraints in Chapter 4.3.2.

4.3 Semantic Enhancement of the Predicates and Constraints

In order to manage the minimal predicate set and expand the set of built-in predicates in SPG-Schema, a namespace mechanism is used, similar to RDF. This mechanism has already been used when using standard types. By introducing the “std” namespace, built-in predicates can be categorized, ensuring that the existing

basic predicates form the minimal set. When building schemas for different domain knowledge graphs in the future, specific namespaces can be added for different domains. In addition, by defining namespaces at different levels, the semantic and capabilities of the Schema Core can be enriched and improved. This will provide greater flexibility and utility for applications in different industrial environments.

The Model layer defines the perceptible class, property, and relation and their semantics for users. As the abstraction layer of the Model layer, the SPG Meta Model layer needs to define schema structures through a series of constraints. In order to express the semantics of entities, starting from the minimal constraint set (ρdf), in section 4.2, we introduced the core semantics of SPG-Schema. In this section, we will focus on the additional built-in predicates and constraint semantics that we have added to the Meta Model level of SPG Meta Model. The constraints that appear at the Model level are categorized into single binary relation constraints, multiple binary relation constraints, grouping rules, and dynamic types. Currently, the predicates and constraints we are discussing are all stored under the standard namespace "std".

4.3.1 The Minimal Constraint Set ---- ρdf

The concept of ρdf [18] is derived from the minimal set of predicates in RDF. In order to better meet the needs of SPG, adjustments have been made to retain subClassOf, domain, and range as the minimal constraint set of the SPG-Schema.

1. Entity Type Hierarchy (subClassOf)

The "subClassOf" predicate is used to supplement the semantics of type inheritance by defining hierarchical relations between node types. When defining a node type, inheritance can be achieved by specifying the name of the type preceded by the "SUBCLASSOF" keyword.

```
// Definition of two subclasses of the User
CREATE ENTITY TYPE (Person {age INT, OPTIONAL father Person}) SUBCLASSOF (User);
CREATE ENTITY TYPE (LegalPerson {amount INT, legalId STRING}) SUBCLASSOF (User);
```

By inheriting from the abstract node type (User), both of these subclasses will automatically include the properties and constraints defined in the "User" class. In addition, the "Person" type will include an additional "age" property and an optional "father" property. The "LegalPerson" type will include an additional "amount" property indicating the number of shares held, and a "legalId" property indicating the legal identification number.

It is important to note that when using the "subClassOf" keyword, it is necessary to ensure that the subclass node does not include properties with the same name as those in the superclass node. Otherwise, there will be issues with overwriting and overriding. This applies to both cases: when the name and type are the same, and when the name is the same but the type is different. We consider both of these cases to be errors and they will not be processed.

2. Domain and Range of the Relations

When defining relations, both the source node type and target node type need to be specified. In order to reduce the redundancy and computational costs associated with entity conversion, modifications to the DOMAIN and RANGE should follow the principle of "addition without modification". For example, in the previous example, a relation called "holdShares" was created, which included a property called "percent" representing the percentage of shares held. However, due to an initial improper definition, it was discovered that there is an additional requirement for the relations of the form "(LegalPerson) -[holdShares] -> (LegalPerson)". To meet this requirement, the value domain of the relation needs to be modified to include "LegalPerson".

```
// The defined holding relation mentioned above
(Person)-[holdShares {percent DOUBLE}]->(LegalPerson)
// Adding the domain LegalPerson for it
ALTER <hold_share> DOMAIN (LegalPerson);
// The actual form of the relation after the operation should be
(Person | LegalPerson)-[holdShares {percent DOUBLE}]->(LegalPerson)
```

4.3.2 Single Binary Relation Constraints (Binary Constraints, BC)

In SPG-Schema, the BC features are defined as follows: "Binary constraints (BC), i.e., defined to be (ir)reflexive, (in)transitive, (a)cyclic, (a/anti)symmetric, etc".

These constraints focus more on the properties of a binary relation and are usually defined together when defining the relation. Therefore, we define these constraints as keywords. After defining BC constraints for a relation at the Model layer, all instances of that relation at the Instance layer should comply with these constraints. According to the above definition, BC constraints in SPG-Schema should have the basic properties of reflexivity, symmetry, and transitivity.

Let's assume that several node types, Class1 and Class2, have already been defined. We can create reflexive, symmetric, and transitive relations using the keywords "REFLEXIVE", "SYMMETRIC", and "TRANSITIVE" respectively. The semantic deductions of these definitions are also provided.

```
// Definition of reflexive relation edgeA.
CREATE EDGE TYPE REFLEXIVE (Class1)-[ edgeA { prop STRING }]->(Class1);
// This means that
(a:Class1)-[p:edgeA]->(a:Class1)
---------------------------------
// Definition of symmetric relation edgeB
CREATE EDGE TYPE SYMMETRIC (Class1)-[ edgeB { prop STRING }]->(Class2);
// This means that
(a:Class1)-[p:edgeB]->(b:Class2),
------------------------------
(b:Class2)-[p:edgeB]->(a:Class1)
// Definition of transitive relation edgeC
CREATE EDGE TYPE TRANSITIVE (Class1)-[ edgeC { prop STRING }]->(Class1);
// This means that
(a:Class1)-[p1:edgeC]->(b:Class1),
(b:Class1)-[p2:edgeC]->(c:Class1)
```

```
--------------------------------
(a:Class1)-[p3:edgeC]->(c:Class1)
```

In addition to the three basic constraints mentioned above (reflexivity, symmetry, and transitivity), in order to achieve semantic completeness, we have also added functional and inverse functional relations based on the OWL syntax. These relations are defined using the keywords “FUNCTIONAL” and “INVERSE_FUNCTIONAL” respectively. The definitions are as follows:

$$FunctionalProperty:\ \top \sqsubseteq (\leqslant 1\, r.\top)\ \ (e.g., \top \sqsubseteq (\leqslant 1\, hasMother))$$
$$InverseFuntionalProperty:\ \top \sqsubseteq (\leqslant 1\, Inv(r).\top)\ \ (e.g., \top \sqsubseteq (\leqslant 1\, isMotherOf^{-}))$$

```
// Definition of functional relation edgeD
CREATE EDGE TYPE FUNCTIONAL (Class1)-[ edgeD { prop STRING }]->(Class2);
// This means that
(a:Class1)-[p:edgeD]->(b:Class2),
(a:Class1)-[p:edgeD]->(c:Class2)
--------------------------------
  (b:Class2) = (c:Class2)

// Definition of inverse functional relation edgeE.
CREATE EDGE TYPE INVERSE_FUNCTIONAL (Class1)-[ edgeE { prop STRING }]->(Class2);
//  This means that
(b:Class2)-[p:edgeE]->(a:Class1),
(c:Class2)-[p:edgeE]->(a:Class1)
--------------------------------
  (b:Class2) = (c:Class2)
```

4.3.3 Multiple Binary Relation Constraints (Multiple Constraints, MC)

To enhance the semantic reasoning capabilities of SPG-Schema, additional support for MC predicates has been added. These predicates are more focused on the relation inference between two binary relations. They are set using the “SET REL” syntax, where the predicate section uses built-in predicates under the std namespace.

Let's assume that two relations have already been defined with the aliases “<Pred1>” and “<Pred2>”. We can define relation inversions and mutual exclusions using the built-in predicates “std.inverseOf” and “std.mutexOf” respectively. The semantic deductions of these definitions are also provided.

```
// The two relations are mutually inverse
SET REL <Pred1>-[std.inverseOf]-<Pred2>;
```

The “inverseOf” predicate is used to define an inverse relation. If two relations are defined as inverseOf each other, it essentially means that they are a pair of equivalent inverse relations. For example, the “superior” relation can be defined as the inverseOf the “subordinate” relation. During reasoning, it is possible to utilize the "superior" relation to automatically infer the “subordinate” relation, thereby solving industrial problems arising from complex semantic reasoning. In the example mentioned, “(Class1)-[Pred1]->(Class2)” and “(Class2)-[Pred2]->(Class1)” form a pair of inverse relations.

```
//The two relations are mutually exclusive
SET REL <Pred1>-[std.mutexOf]-<Pred2>;
```

The “mutexOf” predicate is used to define a mutual exclusion relation, where an instance relation can only be chosen from the two defined relation types. In the example mentioned, for the same relation instance ‘s’, it is not possible to have both the “Pred1” and “Pred2” relations simultaneously. In other words, “(Class1)-[Pred1]->(Class2)” and “(Class1)-[Pred2]->(Class3)” can only be mutually exclusive, allowing for a binary choice.

In the MC predicates, the appearance of angle brackets “< > -[ ]- < >” is used for a more user-friendly representation, which aligns with the original usage habits of Cypher users. In this context, the angle brackets “< >” are used as aliases when defining relations. For example, “<Pred1>” can essentially be seen as the triplet relation “(Class1)-[Pred1]->(Class2)”. This greatly simplifies the complexity of expressing relation between multiple binary relations, making the overall syntax more concise and easy to understand.

4.3.4 Relation Grouping

In real-world scenarios, it is often necessary to query a class of similar relations through an abstracted relation. Therefore, it is important to introduce relation grouping predicates to assist users in constructing groups and reduce query costs. The relation grouping can be divided into two parts: relation grouping and property grouping. Since properties can be considered as a kind of relation to some extent, the “SET REL” syntax is uniformly adopted to define relation grouping.

1. Relation Grouping

The first application scenario of classification is relation grouping, which is primarily defined using the built-in predicate “std.subRelOf”. When constructing a group, it is important to ensure the existence of a top-level relation. This top-level relation should be marked with the “ABSTRACT” keyword when creating it using the “CREATE EDGE TYPE” statement. This indicates that no instances can be associated with the abstract grouping relation. Any loaded entity relations should belong to specific relations within this group.

Relation grouping can also be considered as a relation between two binary relations, but it is different from the MC predicates mentioned above. Relation grouping predicates have a distinction in syntax from relation type definitions. While the MC predicates define an equivalence relation between two binary relations, in relation grouping, there is a clear hierarchy between the two relations. Therefore, the syntax for defining relation grouping is in the form of a triplet with arrows: “< >-[ ]-> < >”.

```
// Creating a grouping of family relations using the ABSTRACT keyword for annotation
CREATE EDGE TYPE ABSTRACT (Person)-[kinship]->(Person) AS <kinship>;

// Definition of three family relations: isFatherOf, isMotherOf, and conjugality
CREATE EDGE TYPE (Person)-[isFatherOf]->(Person) AS <father>;
CREATE EDGE TYPE (Person)-[isMatherOf]->(Person) AS <mother>;
CREATE EDGE TYPE (Person)-[conjugality]->(Person) AS <conjugality>;

// Definition of relation grouping
```

```
SET REL <father>-[std.subRelOf]-><kinship>;
SET REL <mother>-[std.subRelOf]-><kinship>;
SET REL <conjugality>-[std.subRelOf]-><kinship>;
```

Based on the above definitions, we consider the parent-child relation, mother-child relation, and marital relation as specific relations within the family relation grouping. These three relations can be directly obtained through the "kinship" relation. However, when loading instances, there will not be any triplets belonging to the family relation group. Instead, they should belong to one of the specific relations: parent-child, mother-child, or marital relation.

2. Property Grouping

After property normalization, properties can also be considered as specific relations and can be grouped using the "subPropOf" predicate. However, this requirement does not align with a complete and comprehensive semantic definition. There may still be a need for property grouping even for non-standardized properties.

Therefore, the syntax for property grouping is similar to "subRelOf", as it involves grouping properties. The difference lies in the fact that the top-level grouped property may have its own instance data, and properties are loaded into entities. Therefore, there is no need to define an abstract top-level property in advance using the "ABSTRACT" keyword. Instead, the (type.attribute) pattern is used to specify the desired property under a specific type.

```
//To group transaction aggregated values within the Person type
SET REL (Person.1_day_complaint_rate)-[std.subPropOf]->(Person.day_complaint_rate);
SET REL (Person.7_day_complaint_rate)-[std.subPropOf]->(Person.day_complaint_rate)
```

4.3.5 Dynamic Types

1. The Hypernym Predicate for Concept Hierarchy

Due to the diversity of conceptual domains, different domains may use different hypernyms to express hierarchical relations. Therefore, we support the use of the "Hypernym" predicate to express a class of hypernyms while defining events. This allows for the hierarchical classification of the events in different conceptual domains. This predicate is defined simultaneously when defining the concept type. For example, in the concept of risk personnel classification, we can set the hypernym as "isA", while in the concept of city classification (CityTaxonomy), we can set the hypernym as "locateAt".

```
// Definition of Risk Personnel Classification Concept:
CREATE CONCEPT TYPE (RiskPerson {
  isA std.Hypernym,
  OPEN
});

// Definition of City Classification Concept
CREATE CONCEPT TYPE (CityTaxonomy {
```

```
  locateAt std.Hypernym
});
```

With this definition, in the risk personnel classification concept, there can be instances such as “gambler” isA “risk personnel”. In the city classification concept, there can be instances such as “Chengdu” locateAt “Sichuan” and “Sichuan” locateAt “China”.

In addition, event types are a special type in the schema. They are essentially a graph and are often associated with the event classification concepts. Therefore, it is necessary to use special predicates to constrain the event types. The “std.subEventOf” is the specific predicate used for the event concept hierarchy within the Hypernym predicate. When defining the event concept types, “std.subEventOf” must be used as the hypernym predicate.

```
// Define the concept of company operation events.
CREATE CONCEPT TYPE (CompanyOperationTaxonomy {
  std.subEventOf std.Hypernym,
  OPTIONAL beginTime TIMESTAMP,
OPEN
 });

// At the instance layer: The concept of executive escape event belongs to the concept of company operation events
<EscacapeEvent:CompanyOperationTaxonomy>-[std.subEventOf]-><CompanyEvent:CompanyOperationTaxonomy>;

// At the instance layer: The instance of stock price fall event belongs to the concept of company operation events
<FallInStock:CompanyOperationTaxonomy>-[std.subEventOf]-><CompanyEvent:CompanyOperationTaxonomy>;
```

The “std.subEventOf” predicate applies to the event concept instances at the instance layer. After the event concept hierarchy is defined, the subject and object of a child event must be subclasses of the subject and object types of the parent event, respectively. The child event can also have additional properties apart from the parent event. For example, in the given example, the “EscapeEvent” concept and the “FallInStock” concept both belong to the “CompanyEvent” concept.

2.The belongTo Predicate for Dynamic Types

Dynamic types refers to the practice of associating a concept instance with an entity instance or an event instance, effectively using the concept instance's name as the type of that instance. We primarily use the “SET REL” syntax and the “belongTo” keyword to specify the specific instances (including the entity instances and event instances) and their belonging to specific concept instances.

```
// Instances of basic entity types belong to the concept of risk personnel classification
SET REL <User>-[std.belongTo]-><RiskPerson>;
```

Firstly, we can associate the entity types with the concepts. In the previous context, we defined the concept of “RiskPerson” and the entity type “User”. We can use the “std.belongTo” keyword to create an association indicating that the “User” entity type belongs to the “RiskPerson” classification concept.

```
// Definition of Company Operation Event
CREATE EVENT TYPE (CompanyEvent {
```

```
  {REQUIRED begintime TIMESTAMP}
  SUBJECT (Company | Person),
  OBJECT (Company | Person)
});
// Instances of company operation events belong to the concept of company operation events.
SET REL <CompanyEvent>-[std.belongTo]-><CompanyOperationTaxonomy>;
```

According to the previously defined concept taxonomy for company operations, “CompanyOperationTaxonomy”, we can associate the newly defined company event type, “CompanyEvent” with it, using the “std.belongTo” keyword. This association allows us to define instances of company events that belong to specific instances of the company operation concept within the company operation event classification concept.

3. The leadTo Predicate for Concept Inference

The “leadTo” predicate, unlike “belongTo”, represents a causal relation at the event concept level. Utilizing this rule allows for the automatic inference of related relations within event types. However, both “leadTo” and “belongTo” use the same syntax. We can further describe how SPG-Schema uses “leadTo” for semantic inference by redefining the event types of executive escape and stock price drop.

```
// Define that the operation event of one company can lead to the operation event of another company
SET REL <CompanyOperationTaxonomy>-[std.leadTo]-><CompanyOperationTaxonomy>;
// If an event instance "a" belongs to the concept of executive escape event, and there is an instance of executive escape
event concept leading to an instance of stock price fall event
<a:EscapeEvent>-[std.belongTo]-><EscapeEvent:CompanyOperationTaxonomy>;
<EscapeEvent:CompanyOperationTaxonomy>-[std.leadTo]-><FallInStock:CompanyOperationTaxonomy>
---------------------------------------------------------------------
// Automatically generate an instance "b" of FallInStock event, associated with the stock price fall event of company A
<b:FallInStock>-[std.belongTo]-><FallInStock:CompanyOperationTaxonomy>;
```

The relation defined by “std.leadTo” represents an association at the modeling level but applies to event concept instances at the instance layer. In the example mentioned above, the instance "ExecutiveEscapeEvent leads to FallInStock" represents a causal relation. When an event ‘A’ of an executive escape occurs and is associated with the executive escape event concept in the domain, a new event instance, “FallInStock of Company A”, can be directly generated.

4.4 Semantic Enhancement through Rule Definitions

To form a more comprehensive semantic framework, in addition to the basic predicates and constraints, it is also necessary to introduce rule definitions for supplementation. However, rule definitions are mostly implemented through programming rather than syntax definition. Therefore, in this section, we will introduce the syntax for rule definitions and provide relevant examples for reference.

4.4.1 The Self-defined Relation/Property Rules

The Self-defined relation/property rules are common requirements in business scenarios, where a specific relation is generated only when certain conditions are met. To accommodate rule definitions, we

can further expand the EdgeType syntax using the “RULE” keyword. Since the rule definitions often apply to the specific instances, we can use the format (instance:Class) to represent the relation triplets. However, creating a relation without a rule block guided by “RULE” will not have any additional effects.

```
// To define the hold share rate on a single chain link (recursively defined)
CREATE EDGE TYPE
(s:Person)-[p:hold_share_rate]->(o:LegalPerson)
RULE {
  STRUCTURE {
    (s)-[p1:hold_share_rate]->(c:LegalPerson),
    (c)-[p2:hold_share_rate]->(o)
  },
  CONSTRAINT {
    real_rate("The actual holding proportion") = p1.real_hold_share_rate*p2.real_hold_share_rate
    p.real_hold_share_rate = real_rate
  }
};
```

The rules can be divided into structural rules and constraint rules. These two kinds of rules are distinguished within rule blocks, guided by the keywords “Structure” and “Constraint” respectively. The former is composed of the triple relations that form the structure of an instance graph, while the latter is a piece of rule code.

4.4.2 Contradictory Rules

```
// Define relation --- Certificate is validly owned by LegalPerson
CREATE EDGE TYPE (Cert)-[effect_owned_by]->(LegalPerson) AS <effect_owned_by>;
// when the certificate is valid, there exists a inverse relation - LegalPerson owns the certificate.
CREATE EDGE TYPE (o:LegalPerson)-[p2:has]->(s:Cert) CONDINVERSEROF <effect_owned_by>
RULE {
  STRUCTURE {
  },
  CONSTRAINT {
    s.is_effect == true
  }
} AS <has>;
```

The conditional inverse relation is introduced using the “COMDINVERSEOF” keyword combined with the “RULE” keyword. It is guided by curly brackets, which enclose several conditional rule statements. Only when the conditional rules are satisfied can the two relations be considered as inverse relations. When the conditional rule is empty, it defaults to being equivalent to the inverse relation. In the example above, a basic relation, “<effect_owned_by>”, is defined. Through inverse relation, it is defined that the “legal person owns certificate” relation will only be established when the condition of the certificate still being valid is met, and it is annotated with the alias “<has>”.

4.5 The Relationship between SPG-Schemas and PG-Schemas

The PG-Schemas [16] aims to address the shortcomings in property graph database management and the lack of schema support in existing systems. It enhances type definitions and improves data integrity constraints to provide more flexible type management, supporting a certain degree of type inheritance and reuse. By formally defining entities, relations, and properties through PG-Keys [17], they can be expressed formally on the property graph. By establishing a flexible and powerful framework for defining key constraints, it enhances the logical connections and consistency among different elements of the schema. However, the complexity of knowledge graph construction and usage remains high, and users still need to prepare a large amount of data work. Additionally, there are still some issues when directly representing SPG using PG-Schemas.

- Lack of business semantic of types: PG-Schemas allow users to dynamically combine node types using the operators "&" (and) and "|" (or), providing richer type expression capabilities. However, it does not effectively capture the business semantics of types, including the internal semantic structure within types and the semantic representation between types. The fuzzy hierarchy relations between class labels also makes it difficult to control the structure in practical business implementation.
- Lack of support for logical dependencies: PG-Keys provide a framework for defining key constraints as global constraints to enhance the data integrity of the property graphs. However, further exploration is needed for the complexity of validation and maintenance in specific query languages, as well as for the problems of implication and reasoning. SPG knowledge management aims to achieve organic integration between logical rules and factual knowledge, depicting logical dependencies between knowledge, and building a hierarchical derivation mechanism for knowledge to reduce ineffective duplicate construction and ensure logical consistency.
- Partial support for feature completeness: The PG-Schemas article mentions that in the current version, binary constraints (BC) and introspection (IS) features are not fully supported. The authors also found through comparative experiments that BC features are lacking in other works such as RDFS, SHACL, and ShEx.

Overall, PG-Schemas primarily focus on enhancing database management with features such as improved type definitions and strengthened key constraints, as described in Chapters 1, 2, and 3. On the other hand, SPG is designed to provide knowledge management capabilities in terms of logical dependencies, knowledge hierarchy, knowledge construction, and programmability. These are two different perspectives, where PG-Schemas enhance capabilities for databases, while SPG aims to lower the barrier of entry for users to use knowledge graph and reduce their involvement in knowledge construction. PG-Schemas can provide global consistency validation, which can be applied to the knowledge graph produced by SPG. Combining the official description of PG-Schemas, the relationship between SPG and PG-Schemas can be illustrated as shown in Figure 30.

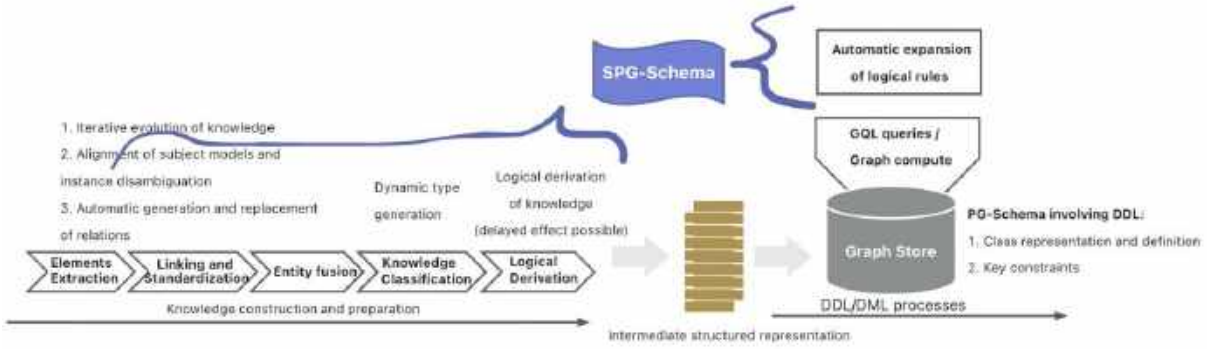

Figure 30: The Relationship between SPG and PG-Schemas

4.6 Summary of SPG-Schema

This chapter provides a detailed description of the overall architecture and the design of syntax and semantics for SPG-Schema. It also explains the extension details of the main model based on the general property graph model. The current version mainly focuses on three main categories: concepts, events, and standard types, which are derived from business requirements in SPG DC. The design and formation of logical syntax and related predicates for SPG Reasoning, which enhances semantic reasoning capabilities, are also described for the expanded classification model. In future versions, there will be a further rigorous proof of the semantic completeness of the current syntax, in order to form a user-friendly and complete syntax and semantic system. More detailed introductions to predicate logic semantics and syntax will be published in the form of serialized articles on the SPG technical community and public platform.

Chapter 5 SPG-Engine Layer

This chapter focuses on the implementation of the actual execution process of SPG syntax, which we refer to as the SPG-Engine layer. The SPG-Engine layer is a module that converts the inference and computation of SPG to be executed in an actual LPG system. The underlying dependencies of SPG typically include basic capabilities such as graph storage, graph querying, and graph computation, which are usually provided by the graph service provider of LPG. This chapter describes the overall architecture of the SPG-Engine layer, dividing it into functional modules such as graph model definition, graph data import, graph querying, and computation, and provides ways to integrate with the underlying LPG processing system.

5.1 The Architecture of SPG-Engine

The SPG-Engine is a module that converts the inference and computation of SPG to be executed in an actual LPG system. The underlying dependencies of SPG typically include basic capabilities such as graph storage, graph querying, and graph computation, which are provided by the graph service provider of LPG. In order to meet the requirements of knowledge graph inference and service capabilities based on SPG, we divide the requirements for engine capabilities into basic capabilities and advanced capabilities.

GQL [19] is the ISO international standard for the property graph query language, which is scheduled to be released in 2024. It defines the specification for querying based on property graphs and is compatible with both weakly-typed labels and strongly-typed types. The SPG solution does not impose any restrictions on whether the underlying graph service uses labels or types. As long as it can implement the interface of the SPG-Engine LPG Adapter, it can be integrated with the SPG engine. The SPG-Engine LPG Adapter provides a way for the third-party property graph systems to connect to the SPG system. It can be implemented using the GQL language or self-defined functions/procedures, and can also be implemented using a single HTAP [20] graph database system or a combination of an OLTP graph database system and an OLAP graph computing system.

In conjunction with the architecture diagram shown in Figure 24, the detailed functions of each module in the SPG Engine are shown in Figure 31.

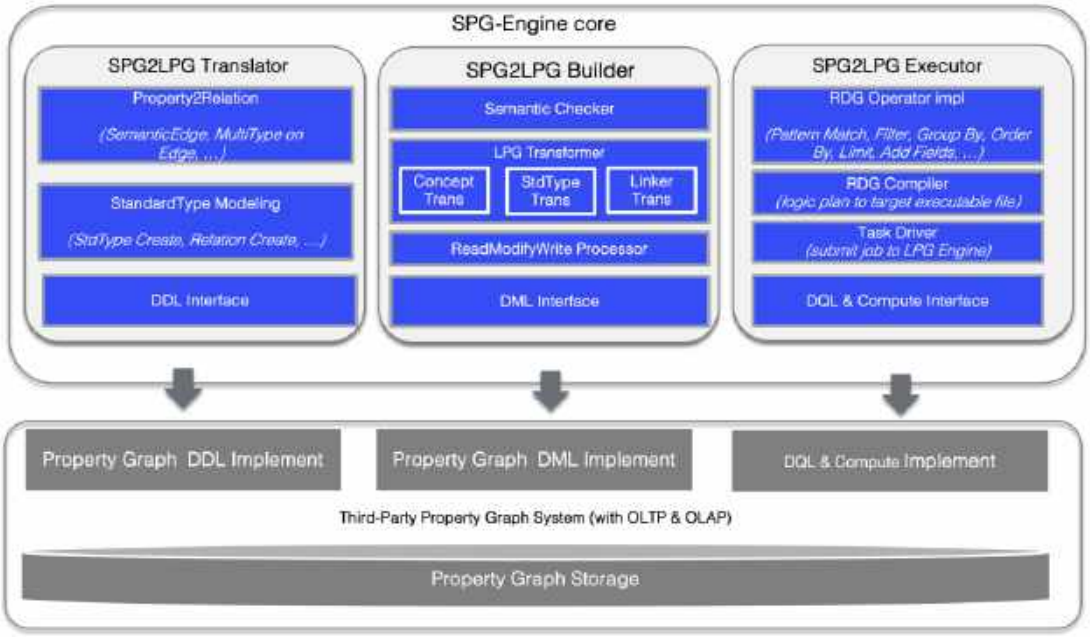

Figure 31: Overall Architecture of SPG-Engine

The SPG-Engine Core layer is a functional module that implements the conversion between SPG and LPG, running as a dependency package within the SPG-Controller process. The third-party property graph system serves as an independent service process, responsible for actual SPG data storage, querying, and computation tasks. It interfaces with the SPG2LPG Translator, SPG2LPG Builder, and SPG2LPG Executor in the SPG-Controller through DDL interfaces, DML interfaces, query interfaces, and computation interfaces. The third-party property graph system needs to meet the basic requirements specified in the SPG-Engine specification and implement the integration interface of the SPG-Engine LPG Adapter. For advanced requirements, they can be described using configuration files, and if not possible to implement, empty interface implementations should be provided.

As a module supporting core functionality, the performance and elastic deployment capability of SPG-Engine are crucial. A high-performance third-party property graph system can not only handle large amounts of data but also ensure system stability and responsiveness. The elastic deployment capability makes the system more flexible in adapting to various application scenarios and changing requirements, improving user satisfaction and business adaptability. We can evaluate performance and elastic deployment capability through methods such as load testing, benchmark testing, and simulation of real-world scenarios. In the current version of SPG 1.0, our focus is on functionality integration and implementation to ensure the complete realization of core functions. In future versions, we will enhance and strengthen the description and implementation of performance and elastic deployment capabilities and conduct more rigorous evaluations to meet user expectations.

5.2 SPG2LPG Translator

The SPG-Schema chapter describes the relationship between SPG-Schema and LPG-Schema, as shown in Figure 29 in Chapter 4. SPG adds semantic predicates, type extensions, logical rules, etc., on top of LPG.

In this chapter, the Semantic Layer's representation of the schema needs to be transformed to the corresponding schema format in the LPG engine. The SPG Meta Model can be transformed back and forth with the Meta Model, following the schema model hierarchy. The SPG2LPG Translator is primarily responsible for converting the SPG schema to the LPG schema format. One of the biggest differences between the SPG Schema and LPG Schema is the properties types. The translation framework needs to convert the semantic property types in SPG to text/numeric data types in LPG and generate the corresponding relations. Additionally, semantic constraints, such as inheritance, dynamic types, and sub-properties, also need to be translated. Furthermore, the built-in standard property types in the SPG Schema need to be converted into separate entity types with constraints and generate the corresponding relations. The SPG2LPG Translator, based on the mapping relationship from the SPG Meta Model to the Meta Model, can be divided into three layers:

- Property2Relation layer: converts properties into edges in the property graph, including standard properties, concepts, and events.
- StandardType Modeling layer: transforms standard properties, concepts, and events into corresponding vertex models.
- DDL interface layer: maps the conversion content from layers 1 and 2 to the DDL interface, requiring underlying property graph support for this capability.

Table 7 shows the type/relation mappings required to complete the translation from SPG Meta Model to LPG Meta Model.

Table 7: Translation Mapping from SPG Meta Model to LPG Meta Model

SPG Meta Model	Function	LPG Meta Model
Class	Entity Type	Node
Concept	Concept	Node
NormalizedProp	Standard Properties	Node
Event	Event	Node
leadTo	Causal Predicates between Concepts	Edge
hypernym	Hierarchical Predicates between Concepts	Edge
object	Object of an Event	Edge
subject	Subject of an Event	Edge
subClassOf	Hierarchy Relation of Entity	Edge
RelationShip	Relation Type	Edge
subRelOf	Hierarchical Relationship between Relations	Edge
MC	Multivariate Constraints of Relations	Edge

In SPG Schema, entity types/concept types support inheritance, and relations support reverse edges. These definitions need to be translated. Examples are provided in Figure 33 and Figure 34.

1. Semantic Translation of Entity Types

For entity types defined using the subClassOf (inheritance) predicate, the properties of the parent class need to be read first and then merged with the properties of the subclass to form the property set of the subclass. It is important to note that the property names of the parent class and subclass should not overlap, as this is a constraint of subClassOf. Additionally, all top-level parent classes of entity types inherit from the root type “Thing”. The “Thing” type includes three basic properties: primary key ID, entity name, and description.

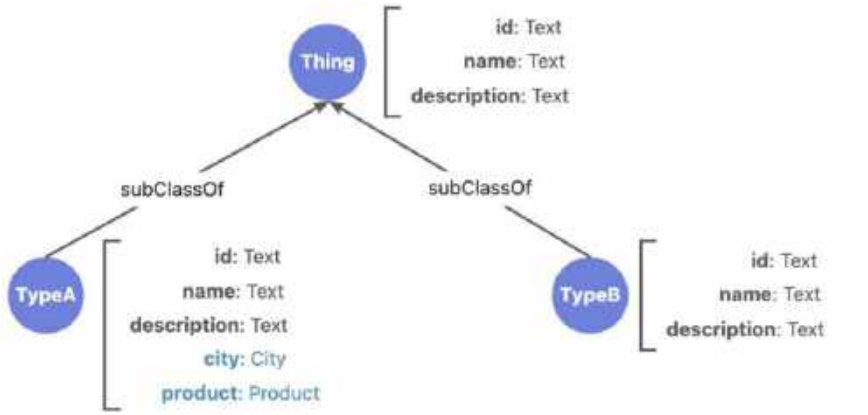

Figure 32: Illustration of the subClassOf Semantic

2. Conversion of Property Types to Entity Types/Concept Types

When the property in the SPG Schema is of type entity or concept, the following conversion actions need to be taken:

- Translate the property type to a text type.
- Add a text type property named “rawOf + propertyName” to store the original value of the property.
- Add a relation from the current entity type to the target entity type or concept type, with the relation name matching the property name.
- If there are sub-properties on the property, these need to be synchronized and created as properties of the relation.

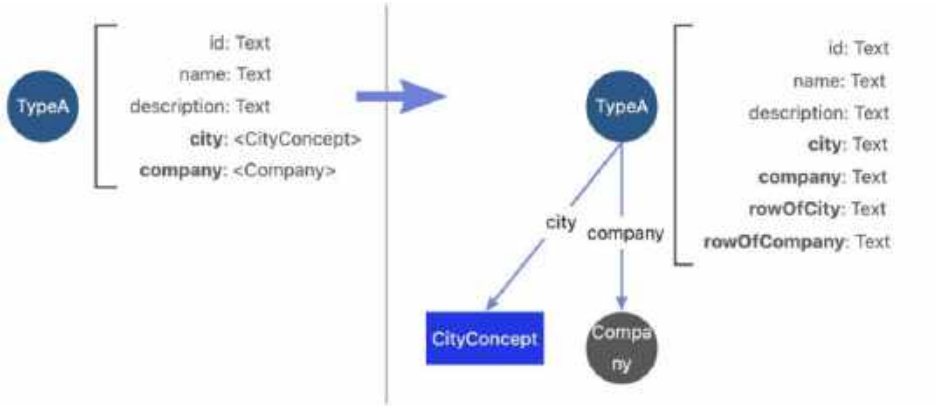

Figure 33: Lossless Redundant Adaptation Process from SPG Semantic Properties to LPG

5.3 SPG2LPG Builder

This module primarily addresses the handling of one or more entity/relation additions/deletions that may occur during the conversion of SPG-formatted data to LPG-formatted data. Data changes can include importing entities, deleting entities, importing relations, deleting relations, importing concepts, and deleting concepts. The submodules for transforming the SPG Meta Model into the LPG Model are shown in Figure 34, and they consist of three parts:

- Semantic Checker: This layer is responsible for semantic checks, ensuring that the input content complies with the constraints defined in SPG.
- LPG Transformer: This layer maps the SPG data to the actual property graph storage model. Detailed transformations will be described below.
- ReadModifyWriter Processor: This layer ensures consistency in read and write operations.

The conversion and adaptation process from SPG to LPG involves entity conversion, relation conversion, concept conversion, etc. For entity conversion, the following DML operations are required: UpsertNode/UpsertVertex for adding or updating nodes, DeleteNode/DeleteVertex for deleting nodes, AddEdge for adding relations, DeleteEdge for deleting relations, GetEdge for querying relations, etc. Firstly, the process checks if the entity property values comply with the corresponding type definitions. Then, queries the relations corresponding to the property names of the current entity and deletes the retrieved relations.

- When the property type is an entity type, the original value is written to the raw property. If the strategy is ID equality, a new relation is directly generated from the current entity to the entity with the ID indicated by the property value. If the strategy is operator, the operator logic is executed, and then a new relation is generated from the current entity to the entity with the result ID indicated by the linking operator.
- When the property type is a concept type, the original value is written to the raw property, and a new relation is generated from the current entity to the concept type specified by the property, with the ID of the property value.
- When the property type is a standard type, the original value is written to the raw property. A new standard type entity with the ID equal to the property value is created, and a new relation is generated from the current entity to the entity generated in the previous step.

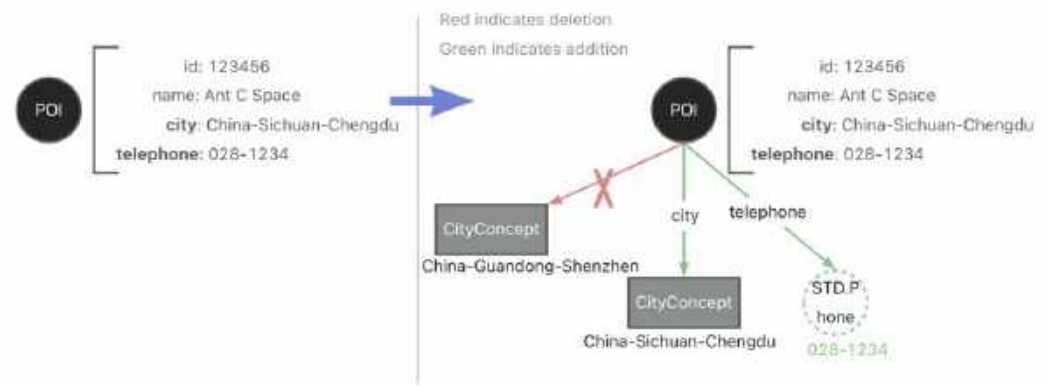

Figure 34: Subgraph Transaction Update Process of SPG Entity Instance converts to LPG

For relation conversion, the following LPG DML operations are required: UpsertNode/UpsertVertex for adding or updating nodes, AddEdge for adding relations, DeleteEdge for deleting relations, GetNode for querying nodes, GetEdge for querying relations, etc. The conversion logic is shown in Figure 37. The process checks if the relation changes comply with the semantic constraints. After adding or deleting a relation, the values on the equivalent properties of the relation need to be updated synchronously.

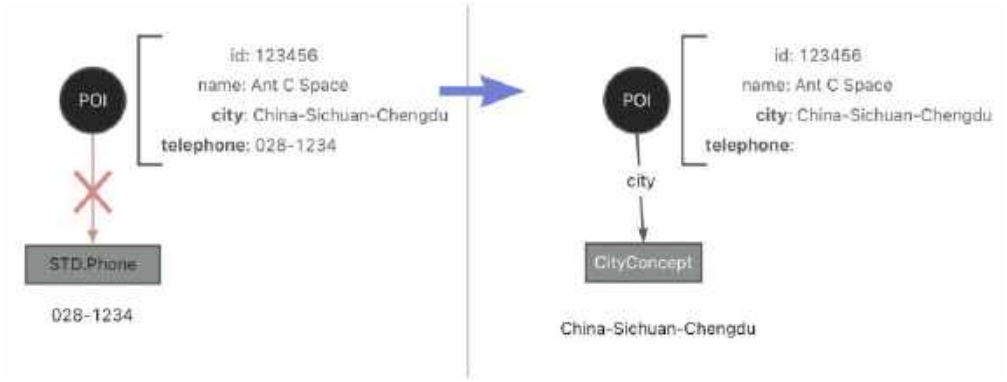

Figure 35: SPG semantic constraint check when updating entity instances

5.4 SPG2LPG Executor

The SPG2LPG Executor is mainly responsible for executing the execution plan composed of RDG operators (Resilient Distributed Graph, RDG) issued by the SPG-Reasoner. The design idea of the RDG model is derived from RDD [21] in Spark. Similar to the RDD approach, RDD simplifies the expression complexity of original MapReduce data operations by abstracted operators such as Map, Filter, and ReduceByKey. The data operation problems also exist in the knowledge graph, so the RDG model is abstracted to transform the required graph operations into operator operations in order to express complex computing processes. The execution plan is organized in a tree structure and the operators on the tree are executed in post-order traversal. To achieve the above goals, the entire SPG2LPG Executor is divided into three parts, each with the following functions:

- RDG Operator Impl: This layer implements the RDG operators based on the underlying LPG engine. It includes the functionality defined by each operator, such as Pattern Match, Filter, etc.

- RDG Compiler: The RDG compiler converts the execution plan issued by SPG-Reasoner into executable binary files that can be executed by the underlying LPG engine.
- Task Driver: This module submits the binary files generated by the RDG Compiler to the LPG Engine for execution. It needs to interface with the specific engine.

1. Execution Plan Generation

The execution plan expresses the process of data processing. Taking the determination of whether a user is a multi-device user as an example, the KGDSL rules are expressed as follows.

```
Define (s:Person)-[p:belongTo]->(o:UserClass/ManyDeviceUser) {
   Structure {
      (s)-[t:has]->(u:Device)
   }
   Constraint {
      has_device_num("Number of devices owned") = group(s).count(u.id)
      R1("owned more than 100 devices"): has_device_num > 100
      R2("Age greater than 18 years old"): s.age > 18
   }
}
```

After being transformed by the SPG-Reasoner, the following operator tree is formed.

```
└──DDL(ddlOp=Set(AddPredicate(PredicateElement(belongTo,p,(s:Person),EntityElement(ManyDeviceUser,UserClass))
   └──Filter(rule=LogicRule(R2,"Age greater than 18 years old",BinaryOpExpr(name=BGreaterThan)))
      └──Filter(rule=LogicRule(R1,"owned more than 100 devices",BinaryOpExpr(name=BGreaterThan)))
         └──GroupByAndAgg(group=Set(NodeVar(s,null))
            └──PatternMatch(pattern=PartialGraphPattern(s,Map(s -> (s:Person), u -> (u:Device)),Map(s -> Set((s)->[t:has]-
```

The operator tree starts with the PatternMatch node and ends with the DDL node. Each node in the tree represents an RDG operator.

2. RDG Operators

In the explanation of the execution plan in section (1), we discussed the order of execution for KGDSL operators. This section primarily introduces the definition of operators in RDG. The table below provides a list of operators.

Table 8: RDG Operator List

ID	Operator	Description
1	patternMatch	Given a subgraph pattern and a starting instance, obtain the subgraph instance (RDG) starting from that instance, returning null if not satisfied.
2	expandInto	Select an entity instance as the starting instance from the already obtained subgraph instances (RDG).
3	filter	Given a condition expression, check if the currently obtained subgraph instance satisfies it, discard the subgraph instance if not.
4	groupByAndAgg	Perform aggregate computation operations on the existing subgraph instances, such as calculating the number of neighbors for a specific point.
5	orderByAndLimit	Sort and truncate operations.
6	limit	Truncate operations without sorting.
7	shuffleAndFilter	Split and aggregate subgraph data based on different perspectives of the entity types.
8	addFields	Store temporary variables computed in the Rule.
9	dropFields	Remove unused temporary variables.
10	join	Merge different subgraph instances (RDG) together.
11	linkedExpand	Invoke a third-party service for link prediction.
12	ddl	Implement the definition of entities and relations.
13	select	Output specified subgraph results.
14	cache	Cache the current RDG for internal computation using the operator.

3. Generating Executable Code

RDG operators represent atomic operations. To convert the RDG operator tree into executable code for the underlying engine, it is necessary to combine it with the execution plan tree generated by the Execution Plan Generator. This process is illustrated in Figure 36.

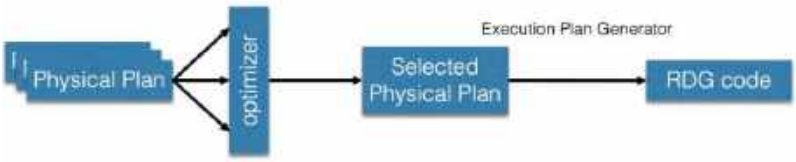

Figure 36: Executable Code Generation Process Flow

The Compiler generates executable code from the Physical Plan and RDG operators. The pseudocode for an Operator is as follows:

```
 abstract class PhysicalOperator[T <: RDG[T]: TypeTag]
extends AbstractTreeNode[PhysicalOperator[T]] {
 /**
  * The context during physical planner executing
  * @return
  */
 implicit def context: PhysicalPlannerContext[T] = children.head.context
 /**
  * The output of the current operator
  * @return
  */
 def RDG: T = children.head.RDG
```

```
  /**
   * The meta of the output of the current output
   * @return
   */
  def meta: List[Var]
}
```

Using the RDG operators as nodes to form a tree-like structure, execute in postorder traversal, for example, the PatternMatch operator.

```
final case class PatternMatch[T <: RDG[T]: TypeTag](
    in: PhysicalOperator[T],
    pattern: Pattern,
    meta: List[Var])
  extends PhysicalOperator[T] {
  override def rdg: T = in.rdg.patternMatch(pattern)
}
```

The input is the RDG of the child nodes. After invoking patternMatch, a new RDG data is returned. Using the example from section 1, the following code (using Neo4j client as an example) would be generated:

```
(new Neo4jRDG(driver)).patternMatch(pattern)
                        .GroupByAndAgg(s, COUNT)
                        .filter(expr("owned more than 100 devices"))
                        .filter("Age greater than 18 years old")
                        .ddl(new AddPredicate("s", "o", "belongTo"))
```

4. Task Execution

Based on different types of LPG engine interfaces, there are two different scenarios:

- Scenario 1: LPG provides query languages such as Cypher [22].
- Scenario 2: LPG provides computational programming frameworks like Spark RDD, allowing users to implement calculations by embedding self-defined operators.

For Scenario 1, the generated executable file is DQL, which communicates with the LPG query engine to perform queries and modifications on the target.

For Scenario 2, the generated code from section 4.3 can be packaged as a plugin and submitted to the LPG engine. This process is illustrated in Figure 37.

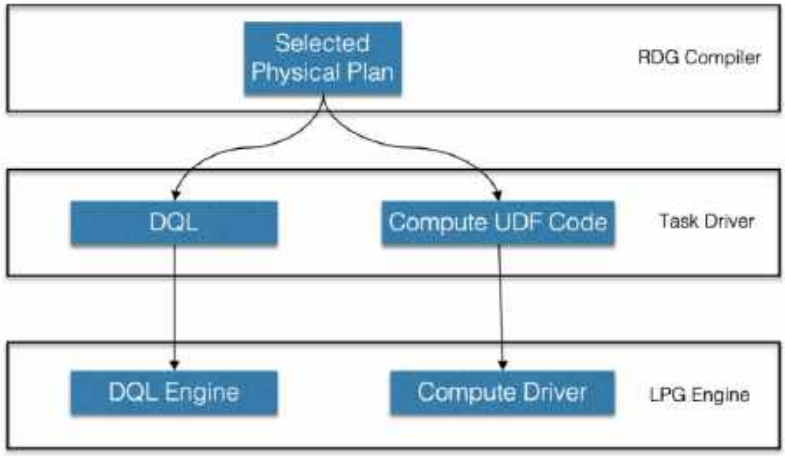

Figure 37: Task Driver Process Diagram

5.5 Basic Requirements for SPG-Engine on the Property Graph Systems

The third-party property graph systems are responsible for storing, querying, and performing calculations on SPG data. These systems use property graphs as the data model, representing and storing data using nodes, edges, and properties. They provide query and computation capabilities based on graph structure semantics. When classifying nodes and edges in the graph, the system can use weakly-typed labels or strongly-typed types, collectively referred to as Labelled Property Graph (LPG) in this context.

The third-party property graph system is an independent service process that should support distributed deployment. It should have independent cluster installation, deployment, management, monitoring, and operation methods, preferably with a web-based UI interface. The graph system interacts with SPG-Controller process through a set of adapter interfaces, which is the SPG-Engine LPG Adapter.

The third-party property graph system needs to include both data storage and querying, as well as analysis and computation on the graph data. Generally, there are two implementation approaches: using a single underlying system with HTAP capabilities, or using different TP and AP systems together to meet the requirements. Regardless of the implementation approach, the system should fulfill the basic requirements for SPG integration with third-party property graph systems, and ideally, fulfill the advanced requirements as well. The implementation should conform to the interface specifications described in SPG-Engine Core.

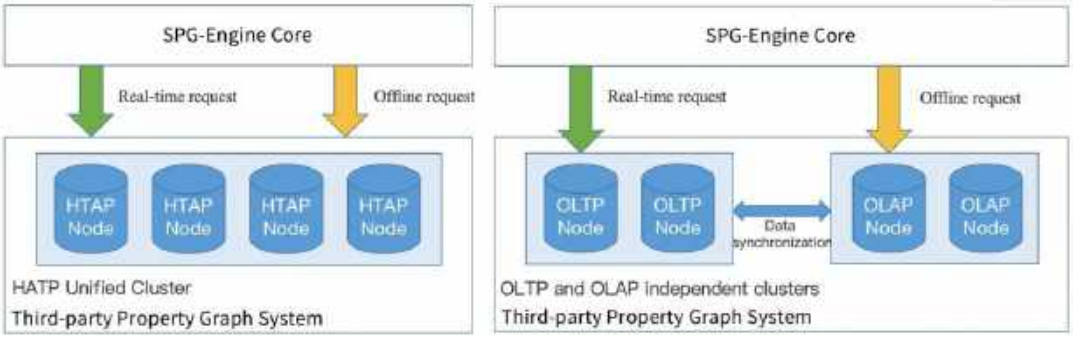

Figure 38: Architecture Diagram for a Single HTAP System and Different TP/AP Systems Integration

The basic requirements capabilities are summarized in Table 9.

Table 9: Basic Functional Requirements for Third-Party Property Graph Systems

Capability	Subcapability	Detailed Description
Graph Model	Supports Property Graph Model	Provides storage and querying of vertices, edges, and properties. Vertices and edges have types or labels.
	Supports Edge Properties	Supports directed and undirected edges. Allows multiple edges of the same type between two vertices.
	Supports Property Types	Supports basic property types: integer, float, string, etc. Supports complex property types: BigDecimal, DateTime, GeographicPoint, etc. Supports container property types: list, set, map, property group, etc.
	Supports Graph Isolation Requirements	Supports storage, querying, and computation of multiple graphs within the same graph service.
Graph Data Import	Data Import Methods	Supports batch import of full or incremental data. Supports streaming import.
	Data Source Support	Supports importing data from existing relational databases using JDBC connections with custom SELECT statements. Supports common file data sources like CSV and JSON formats. Supports popular big data storage file formats like Parquet files on HDFS.
	Mapping Methods	Supports mapping multiple entities and relations from a single source table.
	Import Constraints	Batch import can be performed without stopping the graph engine service.
Property Indexing	Supports indexing for common data types	Indexes can be created on integer, float, string, timestamp, etc.
	Supports composite indexes	Supports composite indexes on multiple property fields.
	Supports fuzzy indexing	Supports fuzzy indexing for full-text search on string properties.
Transaction	Supports ACID properties of transactions.	Provides at least Read Committed isolation level, and allows application-level explicit locking to achieve Serializable isolation level.
Graph Query Language	Supports DQL	Supports standardized query language like GQL or provides DQL query capabilities.
Deployment	Supports flexible deployment modes.	For smaller data volumes, supports master-slave deployment. For larger data volumes, supports distributed sharding deployment.

5.6 Advanced Requirements for SPG-Engine on the Property Graph Systems

5.6.1 System Capabilities

- Support for triggers: Triggers are a special type of stored procedure that sets up an automated event response mechanism in the database. When the specific database operations occur (such as insertions, updates, or deletions), the triggers automatically execute the predefined code or GQL / Cypher statements.
- Support for user-defined functions/procedures/algorithms: User-defined functions / procedures / algorithms provide a mechanism to extend graph queries, allowing users to customize the functionality of the database and access the internal API directly.
- Support for storage and querying of time-series data: Determine the validity of nodes or edges based on a timestamp field, as well as determine the validity of property values based on the timestamps.
- Capability for multi-level graphs: Mapping multiple logical graphs to one or more physical graphs based on business logic. Support mapping from entity to entity, from property to entity, from property to property, and allow querying directly on logical graphs using GQL/Cypher.
- Mapping and conversion of entity/relation types across graphs: Support querying across multiple graphs (including physical and logical graphs).

5.6.2 Semantic Functionality

- Conversion of specific properties to edges: Automatically insert and update edges through triggers and specify the storage method (in-memory cache or physical disk).

Table 10: Capability for Conversion of Properties to Edges

Requirement	SPG-Engine Capability	Description
Standard Property	Property convert to Relation	When creating specific properties, they are converted to relations
Conceptual Property	Property convert to Multi-hop Relations	When converting specific properties, they are converted to paths of multi-hop relations

```
// Entity type with properties support
create (n:Device {id:0, name:"devid****001"})

// When creating a standard property, the Trigger will execute the statement and convert it into an relation
match (m:wifi {id:"TP_LIN*****0011"}), (n: Device {name:"A"}) create (n)-[:use_wifi]->(m)

// When creating a conceptual property, the Trigger will execute the statement and convert it into multiple hop relations
match (m1:Country {name:"China"})<--(m2: Province {name:"SiChuan"})<--(m3:City {name:"ChengDu"}), (n:Person {name:"Zhang San"}) create (n)-[:livi_in]->(m3)
```

- Support for knowledge hierarchy: Automatically insert and update based on self-defined logical rules through triggers.

Table 11: Capability for SPG Type Classification

Requirement	SPG-Engine Capability	Description
Entity Type	Type constraint	When creating a entity and relation, a entity that meets the constraint rules will be created with an relation connected to the labeled entity.
Relation Type	Type constraint	When creating a entity and relation, a entity that meets the constraint rules will be created with an relation connected to the labeled entity.
Property Type	Property constraint	When modifying properties, the properties of entities that meet the constraint rules will be modified accordingly.

```
// Define entity type and add type constraint, check if the constraints are met when creating Person type, create belongTo relation if they are met
// Create Person entity
create (n:Person {id:"2088****0001"})
// Trigger server-side constraint checking, create corresponding relation if constraints are met
match (n:Person {id:" 2088****0001"})-[:has]->(D:Device)-[:has_wifi]->(W:WIFI)<-[:has_wifi]-(D2:Device)<-[:has]-(s)-[p:belongTo]->(o:Fraudster) create (n:Person {id:" 2088****0001"})-[p:belongTo]->(o:Fraudster)

// Define relation type and add type constraint, check if the constraints are met when creating Device entity, create same_wifi relation if they are met
create (n:Device {id:" devid****001"})
// Trigger server-side constraint checking, create corresponding relation if constraints are met
match (n:Device {id:" devid****001"})--[:has_wifi]->(W:WIFI)<-[:has_wifi]-(o:Device) create (n)-[:same_wifi]->(o)


// Define property type and add property constraint, when modify property of App, modify corresponding property of instances that meet the constraint rules
match (n:App {id:"appid*****0012"}) set n.mark = "black"
// Trigger server-side constraint checking, modify corresponding property of instances if constraints are met
match (n:App {id:" appid*****0012"})<-[:release]-(m:Company) set m.mark = "black"
```

- Support for built-in predicates: The built-in predicates require multiple advanced capabilities of LPG, as shown in Table 12.

Table 12: Advanced Capability Requirements for LPG

Requirement Content	Advanced LPG Capability	Introduction
Transitivite	Condition Matching for Paths	Input points, edges, and path conditions to perform path matching.
SameAs	Graph Mapping (see 1.4)	Match subgraphs based on conditions, perform data computation and aggregation, and use the results to create new subgraphs in the current graph.
ConditionInverseOf	Predicate Matching	The predicate expresses a condition of inverse relation.
InverseOf	Predicate Matching	The predicate expresses an inverse relation.
SubPropertyOf	Support for Property Group Value Types	Property groups can have multiple sub-properties, but sub-properties cannot be of Property Group type.
NormalizedProperty	See "Self-defined Logical Rules"	
EquivalentProperty	Property Mapping (see 1.4)	Equality constraints on multiple properties on a entity.
MutexOf	Primary Key Constraint	Unique primary key constraint for the entity of a certain type.

```
// Transitivity, requires support for path condition matching
// User profile aggregation, gathering people with the same taste as Zhang San
match (n:Person {name:"Zhang San"})-[:Hobbies|isA*3]->(m:Taste)<-[:Hobbies |isA*3]-(m2:Person) return m as Taste, collect(m2) as Crowd
// Ultimate controller exploration
match (n:Company {id:" 4201151234****ABC" ) <-[r:Contorl *1..5]-(m:Person)
return m as Controller, sum(reduce(total = 1, h IN r | total * h.holdingRatio / 100.0 ) ) as holdingRatio order by  holdingRatio

// Contradictory conditions, predicate matching
// Find fully-owned parent companies of Company A
match (A:Company {id:" 4201151234****ABC"})-[: SubsidiaryCompany {holding_rate:1}]->(B) return B
// Find fully-owned subsidiaries of Company B
match (B:Company {id:" 4201151234****ABC"})<-[: SubsidiaryCompany {holding_rate:1}]-(A) return A
```

5.7 Summary

This chapter introduced the architecture and implementation of the SPG-Engine layer. The SPG-Engine layer consists of the SPG-Engine Core and the third-party property graph systems. The SPG-Engine Core provides modules for graph modeling, data import, querying, and computation under the SPG semantics, and invokes the interfaces provided by the third-party property graph systems for execution. The third-party property graph systems are provided by LPG graph service vendors, and can be a unified cluster with HTAP capabilities or a combination of separate clusters for OLTP graph databases and OLAP graph processing. This chapter also described the integration with underlying LPG processing systems and the required functionalities of the third-party property graph systems, enabling SPG to run on various commonly used property graph systems. It also provided optimization directions for the property graph vendors to enhance the performance of SPG. More details and implementation examples of RDG will be continuously published in articles on the SPG official account.

Chapter 6 SPG-Controller Layer

The SPG-Controller is the control layer of the SPG framework, responsible for analyzing, invoking, and managing the execution of services and tasks. As the core hub of the SPG framework, it is closely associated with other modules to collectively complete the entire task flow from user input to result. The SPG-Controller receives requests from the SPG-LLM or SPG-Programming, performs parsing and compilation, and generates task planning. It distributes and invokes tasks, selects corresponding capabilities to complete the specific execution process, including choosing the corresponding runtime from registered and deployed SPG-Engine, SPG-Index, or external capabilities. Finally, it returns the task execution results to the caller. This chapter provides an overview of the architecture and workflow of the SPG-Controller, and provides a general description of task compilation planning, task distribution and invoking, as well as knowledge querying, construction, reasoning, and searching services. The detailed description will be gradually unfolded in conjunction with relevant subsystems in the 2.0 and 3.0 white papers.

6.1 The Architecture and Workflow of SPG-Controller

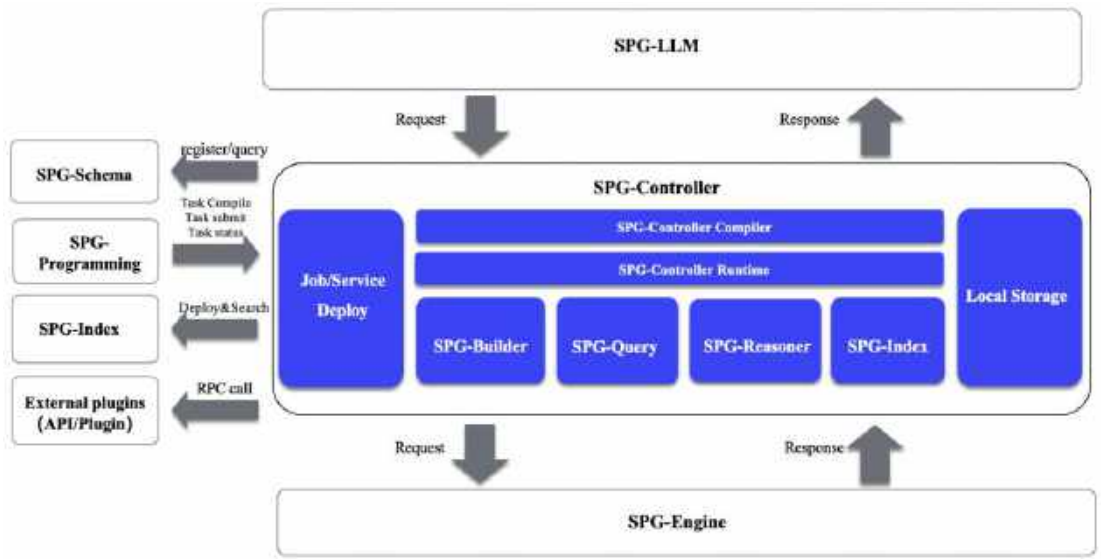

Figure 39: The architecture and workflow of SPG-Controller

The SPG-Controller is the core control hub in the SPG framework, with the following main responsibilities: (1) Parsing, compilation, and task planning: It performs parsing and compilation based on the input from the upper layer, mapping it to the capabilities, commands, and relevant configurations and parameters of each subsystem. Then, based on the results of parsing and compilation, it plans the tasks to achieve the arrangement of task execution mode, process, and cycle. (2) Task distribution and invoking: Based on the results of the task planning, it executes and invokes services and tasks in the corresponding runtime. (3) SPG-Builder: It provides a runtime engine for knowledge construction to transform data into knowledge in SPG. (4) SPG-Query: It provides a runtime engine for knowledge querying, implementing interfaces related to querying and graph-based data mining and analysis in SPG. (5) SPG-Reasoner: It provides a runtime engine for knowledge inference calculation, implementing various inference capabilities

in SPG, such as logical rule inference and neural network inference. (6) SPG-Index: It provides a runtime engine for SPG indexing and searching, implementing the capabilities such as vector search and full-text search in SPG. (7) Job/Service Deploy: It provides the registration and deployment of services or tasks in the corresponding runtime for each SPG Controller, enabling their capabilities to be effectively called.

In the overall system architecture, firstly, the SPG-Controller serves as the runtime engine for the SPG-Schema and provides CRUD Python/Java interfaces. Secondly, it provides task compilation and distribution capabilities for the SPG-Programming. The task compilation includes syntax checking, algorithm/task validity checking and reusing, execution planning, and bytecode generation. Additionally, it accepts pluggable sub-engine modules to be added to the registration information module. Finally, it accepts task requests as input, compiles and distributes tasks, dispatches tasks to sub-engines for computation, receives the results, and assembles them to return to the user.

6.2 Parsing, Compilation, and Task Planning

The input parsing module receives input from SPG-LLM or SPG-Programming and parses it to understand the instructions of the user. This helps identify the task type, execution logic, and process that need to be performed. The input consists of two main parts: task type and task body. The task type determines the type of the task to be executed, while the task body includes the interfaces and their parameters to be invoked. Here are the main task types and their definitions: (1) Schema tasks: Corresponds to basic CRUD interfaces for the knowledge graph schema. The task body includes the interface to be called and its parameters. The interface is defined and implemented in the SPG-Schema module. (2) Query tasks: Corresponds to basic knowledge graph queries, utilizing the graph query capabilities in SPG-Engine. The task body consists of a query language like GQL. (3) Reasoning tasks: Corresponds to reasoning tasks in the knowledge graph, such as rule-based reasoning using KGDSL or neural network-based reasoning tasks. (4) Search tasks: Corresponds to search tasks, such as vector indexing and full-text indexing. This includes tasks such as index creation, tokenization indexing/vector data writing, and text search/vector search. The task body consists of query tasks with conditions. (5) Construction tasks: Corresponds to SPG knowledge graph construction tasks, involving mapping structured data to knowledge, extracting unstructured and multimodal knowledge, and knowledge fusion.

Task planning is based on the results of parsing and compilation to arrange the execution mode, process, and cycle of the tasks.

6.3 Task Distribution and Invocation

Based on the results of task planning, the tasks are distributed and executed in the corresponding runtime, including the registered runtimes for various services and tasks, as well as the pipeline of the task execution. During the task execution process, SPG provides a microservice invocation framework and task scheduling engine to enable service invocation and task execution. Additionally, the SPG-Controller provides storage capabilities to store crucial information during task execution, including log data,

intermediate and final results, execution status, and scheduling data. Storing and tracing this information aids in monitoring and managing the task execution process, ensuring reliability and stability.

6.4 Knowledge Graph Construction

SPG-Builder facilitates the conversion from data into knowledge, including the extraction of knowledge from structured data, semi-structured data, and multi-modal unstructured data. The main capabilities include: (1) ER2SPG: This functionality involves converting data to SPG knowledge, where knowledge is obtained through mapping and transformation of data from a database or big data platform. Based on the conversion of original data to ER (Data to ER, D2R), the input is modified to comply with the SPG specification. (2) Semi-structured knowledge extraction: This functionality is used for extracting knowledge from semi-structured data, and obtain structured elements. This functionality will be added in the second phase. (3) Text knowledge extraction: This functionality will involve utilizing Large Language Models (LLMs) to extract knowledge from text data. This functionality will be added in the second phase. (4) Multi-modal knowledge extraction: This functionality is designed for extracting knowledge from multi-modal unstructured data, such as images, audio, video, and so on. This functionality will be added in the second phase.

With the support of these capabilities, SPG-Builder can extract valuable knowledge from various forms of data, fully harnessing the vast amount of data accumulated in the big data ecosystem. This knowledge is then stored in the SPG repository for future application, analysis, and inference purposes.

6.5 Knowledge Query

SPG-Query provides knowledge graph query and analysis services, including the basic CRUD operations on knowledge graph: graph searching, and graph analysis and mining. Specific query functions include: (2) Basic Query: Supports precise queries on entities, concepts, and properties. For example, querying for the account with the ID “2088****0001” in the risk mining knowledge graph. (2) Advanced Query: Supports fuzzy queries and full-text search on entities, concepts, and properties. For example, performing a fuzzy search on event names or company names in the enterprise causal knowledge graph. (3) Graph Traversal Query: Supports breadth-first and depth-first algorithms for graph traversal queries. For example, querying for accounts that have transaction records in the risk mining knowledge graph. (4) Pattern Matching Query: Supports subgraph queries that satisfy specified patterns. For example, querying for subgraphs in the risk mining knowledge graph that follow the pattern A-B-C-A (transfer of funds).

In addition, SPG-Query provides various graph analysis algorithms, including: (1) Community Detection Algorithms: such as LPA, WCC, SCC, Louvain, etc., which can be used to identify frequent transaction groups in the risk mining knowledge graph. (2) Authority Ranking Algorithms: such as PageRank, HITS, degree centrality, betweenness centrality, closeness centrality, etc., which can be used to calculate the weight of each node in the risk mining knowledge graph. (3) Other Algorithms: such as triangle counting, which can be used to calculate the frequency of a specific transaction subgraph in the risk mining knowledge

graph. These features assist users in gaining a deeper understanding of the knowledge in SPG, enabling them to analyze it more effectively and extract valuable insights.

6.6 Knowledge Graph Reasoning

SPG-Reasoner provides the capability to invoke knowledge graph reasoning, including commonly used knowledge graph reasoning methods. Specific reasoning methods includes: (1) Rule-based Reasoning Algorithms: Used for inferring risk propagation rules in enterprise causal knowledge graph and defining expert rules related to risky activities in the risk mining knowledge graph. (2) Graph Embedding Learning Algorithms: Includes graph neural networks, random walks, translation distance, etc., used for embedding learning and representation learning of the knowledge graph in SPG. (3) Prompt Learning Algorithms: Used for constructing learning algorithms for prompts in SPG-LLM. This functionality will be specifically implemented in future releases.

To support these reasoning capabilities, SPG-Controller provides storage management for registering information on reasoning rules, reasoning algorithms, and configuring algorithm parameters. This aids in monitoring and managing the reasoning process.

6.7 Full-Text Search and Vector Search

SPG-Index provides search services, including conventional search methods such as vector search and full-text search. SPG-Controller interacts with the external SPG-Index plugin, allowing users to easily search and query data. Specific functionalities include: (1) Index Creation and Management: Supports the creation and management of indexes. (2) Index Data Writing and Updating: Supports writing data to the index and performing updates. (3) Vector-based Search: Supports vector-based search, enabling data search and query based on similarity.

6.8 Deployment of the Services and Tasks

“Job/Service Deploy” handles the registration and deployment of services or tasks corresponding to the core modules. Specific functionalities provided include: (1) Registration: Supports the registration of services and tasks, allowing them to be discovered and invoked in SPG-Controller. (2) Management: Supports the management of microservices or tasks, including the oline/offline operations and the configuration management. (3) Execution: Supports the execution of services or tasks in the runtime environment. (4) Monitoring: Supports monitoring the running status, including resource monitoring, to promptly detect and resolve service failures.

In the implementation, SPG-Controller stores the registered and deployed services and tasks, and schedules them during the task execution. To achieve the task scheduling, a task scheduling engine can be used to manage jobs scheduled precisely to the hour, minute, and second, and the concurrency level can be set by dynamically configuring shard parameters. To enable microservice invocation, a microservice

governance framework can be used to manage and invoke microservices, and a microservice gateway can be employed to expose services externally.

6.9 Summary

This chapter provides a summary description of the overall architecture and workflow of the SPG-Controller. In terms of specific tasks, the current version primarily focuses on constructing structured data into a knowledge graph, basic knowledge querying and analysis, knowledge reasoning, and knowledge searching. In future versions, these tasks will be further refined and expanded, such as constructing knowledge graphs from unstructured and multimodal data, representation learning for reasoning, and composite indexing. Additionally, other types of tasks will be integrated, such as integrating external plugin systems and instruction systems. Furthermore, future versions will enhance the management functionalities of SPG-Controller, including unified authentication management and exception handling.

Chapter 7 SPG-Programming Layer

Decoupling the domain model from the underlying engine is a fundamental capability that AI basic engines must possess. By using the programmable SDK and operator frameworks, the business domain model can be separated from the underlying engine, allowing developers to quickly build and deploy self-defined knowledge graph algorithms, schema models, and reasoning capabilities. This helps businesses to rapidly develop and deploy knowledge graph applications, improving efficiency and scalability. The underlying engine also focuses on optimizing general capabilities such as I/O, scheduling, performance, and throughput. The SPG engine is also divided into three layers: the core underlying engine layer, the SDK and operator framework layer, and the business application layer.

Figure 40: SPG-Programming Domain Layering

This chapter provides a brief overview of SPG-Programming. A more comprehensive introduction and the complete syntax representation of KGDSL will be officially released in the Whitepaper 2.0 version. It will also be detailed in a series of articles.

7.1 SPG Semantic Programmable Architecture

The main capability of the programmable sub-module is to provide users with a programmable interactive interface, supporting algorithm/business engineers to model, construct, and reason for the domain knowledge graphs according to the programming framework provided by SPG, and supporting the continuous iteration of the domain knowledge graphs.

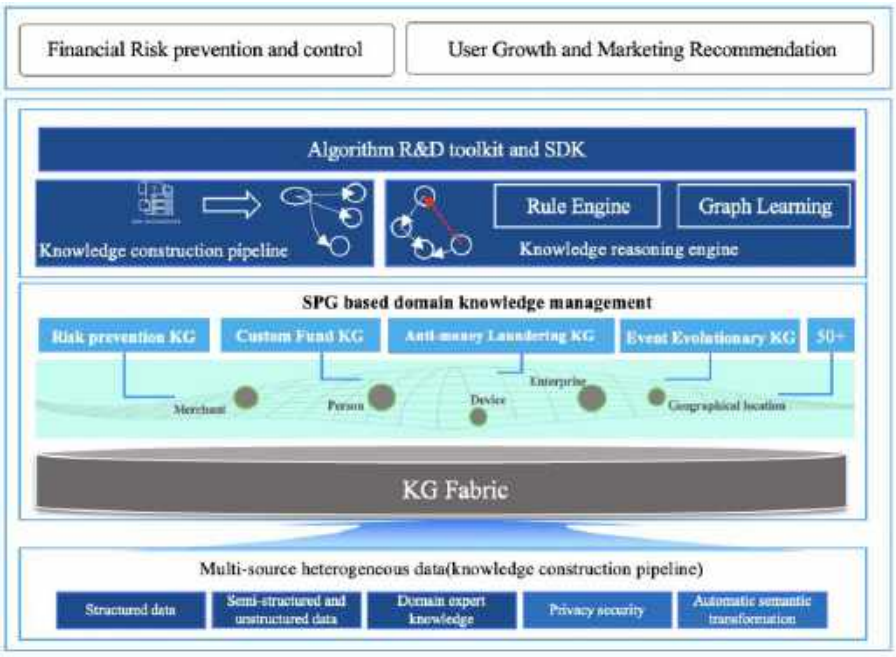

Figure 41: SPG Programmable Framework

In terms of programmability, it includes the following aspects:

- Knowledge Construction Framework: An operator framework for building the domain knowledge graphs based on structured, unstructured, and semi-structured data.
- Logical Rule Programming: Includes logical rule properties/relations, KGDSL rule reasoning, etc.
- Representation Learning and reasoning: Using knowledge graph-based data to implement graph neural networks, logic-guided reasoning, and other knowledge graph-based reasoning methods.

7.2 The Construction and Transformation from Data to Knowledge

As shown in Figure 1 in Chapter 1, with the help of the SPG framework, data can be transformed into knowledge through the programmable operators. These operators include the following categories: (1) Information Extraction Operators: These operators enable the structuring of all unstructured and semi-structured data to obtain knowledge elements. (2) Subject Mounting Operators: These operators involve mounting entities, events, and concept types, as well as mounting entity/event properties. (3) Entity Fusion Operators: These operators address issues related to the construction of entities from multiple heterogeneous sources and the fusion of knowledge across knowledge graphs. (4) Dynamic Classification Operators: These operators support the dynamic definition of business types at a granular level using logical rule-based classification expressions.

The use of standardized operator frameworks and property elements can significantly reduce the cost of preparing raw data. Operators are only related to target entities, concept types, and event types, thus greatly reducing the cost of redundant development. With the introduction of standardized property elements, the types of properties in schema modeling transition from text (Text), integer (Long), and float (Double) to the domain model modeling.

The pseudo-code representation based on LPG modeling is as follows:

```
class User {
  id TEXT(Text);
  name TEXT(Text);
  phoneNo LONG(Integer);
  poc     TEXT(Text);
  homeAddr TEXT(Text);
}
```

Pseudo-code representation based on SPG modeling is as follows:

```
class User {
  id UserNormId(User standard ID);
  name TEXT(Text);
  phoneNo ChnMobilePhone(Mainland China phone number);
  poc     AdminArea/Province/City/District(administrative division);
  homeAddr POI(Standard POI);
}
```

By preparing a User table, users can construct four types of entities and four types of relations, thereby significantly reducing the cost of preparing raw data. With the introduction of knowledge construction operators, the operator framework and operator implementation are separated, and the knowledge construction process is defined as standard components, providing a unified runtime framework for these components. The algorithm developers can quickly implement knowledge construction operators based on the python operator framework and bind them to target types. At the same time, Link Function/Fuse Function or Normalize Function can be specified for each type, facilitating the continuous iteration of the knowledge graph and addressing the issues of continuous construction and evolution of incomplete data and instance resolution in industrial applications. A simple example of the operator definition in pseudo-code is shown below, which can be bound to specific types using “bind_to”:

```
# -*- coding: utf-8 -*-

from knext.api import EntityLinkOp
from knext.api.base import BaseOp
from knext.models.runtime.vertex import Vertex

@BaseOp.register("AdminAreaNormOp", bind_to="AdminArea", is_api_iface=True)
class AdminAreaNormOp(PropertyNormalizeOp):
    def eval(self, property: str, record: Vertex) -> EvalResult[str]
        traces, errors = [], []
        result = ""
        try:
            result = adminNorm(property)
        except Exception as e:
            errors.append(f"property:{property}, error_msg:{e.__repr__()}")
        return EvalResult(result, traces, errors)
```

SPG supports the adaptive propagation of properties and relations during the query phase to minimize user usage costs. When users provide GQL/KGDSL expressions, if propagation is necessary, it will be automatically expanded. If propagation is not required, the standardized property values will be extracted by default. More detailed information about the KGDSL syntax will be provided in future articles.

7.3 Logical Rule Programming

Predicate semantics are the key foundation for implementing the SPG logical rule programming. Through predicate semantics, SPG can be translated into a machine-understandable form and enable machine automatic reasoning capabilities. The definition of these capabilities includes the following layers: (1) **Built-in predicates:** These are predefined basic predicate capabilities. They do not possess any business semantics but can be referenced by higher-level rules. (2) **Logical rule knowledge:** This layer includes logical rules that exist in the form of properties/relations, and enable dynamic classification of entities based on these rules. (3) **Reasoning decision rules:** These rules are derived from subgraphs, structures, paths, and other forms. They support rule-based decisions, knowledge injection, and more. Figure 42 illustrates the overall layered structure. To balance the cost of rule management and computational complexity, it is specified that built-in predicates can only be used to define logical rule knowledge. The application of knowledge reasoning layer relies only on basic factual knowledge and logical rule knowledge.

Figure 42: Dependency of the Reasoning Decision

Defining dependencies between knowledge using predicates and logical rules has been extensively explained in Chapter 4, and it will not be reiterated in this chapter. Logical rule programming mainly consists of two parts: defining knowledge dependencies through logical rules and generating logical derived properties/relations. Additionally, complex end-to-end rule decisions can be defined using DSL/GQL. An example of KGDSL decision is provided below:

```
Structure {
  (s:User)
  (e1:TradeEvent)-[ps1:std.subject]->(su1:User)
  (e1:TradeEvent)-[pp1:std.object]->(sp1:PID)
  (e2:TradeEvent)-[ps2: std.subject]->(su2:User)
  (e2:TradeEvent)-[pp2: std.object]->(sp2:PID)
  (su1)-[has]->(sp2)
```

```
  (su2)-[has]->(sp1)
  (e2)-[pb:belongTo]->(o:/TaxoOfTradeEvent/HighTransactionAmount)
}
Constraint {
  s.id == su1.id
  e1.ts < e2.ts and hour(current_time()) - hour(e1.ts) < 24
  group(s).count() > 10
}
Action {
  createEdgeInstance(
    src=s,
    dst=o:TaxoOfUser/TransactionRisk/MultipleRefundTransactions,
    type="belongTo",
    value= { time=now() }
  )
}
```

7.4 Knowledge Graph Representation Learning

The core problems addressed by the knowledge graph representation learning framework are graph feature extraction and subgraph extraction. It is designed to be compatible with mainstream deep learning frameworks such as TensorFlow/PyTorch and further convert them into tensor structures required by corresponding graph learning algorithms.

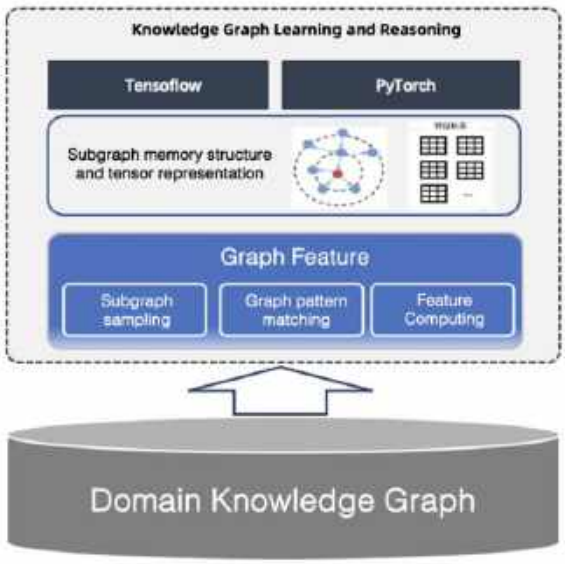

Figure 43: Graph Representation Learning Framework

Knowledge graph representation learning achieves the linkage and decoupling of the graph learning and the graph data through the sampling operator module. The main graph sampling operators currently available include: (1) Subgraph sampling: This is primarily used for multi-hop subgraph sampling in GCN models. It includes positive/negative sample generation and supports weighted sampling and time filtering in the SPG large-scale dynamic heterogeneous paradigm. (2) Structure extraction: This is used for structure-aware reasoning tasks such as symbol rule-guided graph learning and rule mining. (3) Feature computation: This mainly involves extracting subgraph features such as PageRank and degree centrality that can be

computed from the complex graph structure. The following pseudo-code represents the concept of multi-hop subgraph sampling in GCN algorithms. It demonstrates the efficient reading of the knowledge graph data using Python sampling operators:

```
# -*- coding: utf-8 -*-
import libkg_client
from kgrl.conf import KgrlConstants # noqa
from kgrl.data import KGExpression # noqa
from kgrl.data.sampler import KGStateCacheBaseSampler

in_degree = KGExpression.SourceNodeInDegreeKey()
out_degree = KGExpression.SourceNodeOutDegreeKey()
node_version = KGExpression.SourceNodeVersionKey()
edge_version = KGExpression.EdgeVersionKey()
v_begin = 30
v_end    = 40
def get_filters(v_begin, v_end):
   return {
      KgrlConstants.NEIGHBORHOOD_SAMPLING_FILTER_NAME: f"{edge_version}<{v_begin} and
{edge_version}>{v_end}",
      KgrlConstants.NODE_SAMPLING_FILTER_NAME: f"{node_version}==0",
      KgrlConstants.EDGE_SAMPLING_FILTER_NAME: f"{edge_version}<{v_begin} and {edge_version}>{v_end}",
   }
def get_weights(v_begin, v_end):
   return {
      KgrlConstants.NEIGHBORHOOD_SAMPLING_WEIGHT_NAME: f"abs({edge_version}-
{v_begin})*log2({edge_version}+{v_end})",
      KgrlConstants.NODE_SAMPLING_WEIGHT_NAME: f"({out_degree}+{in_degree})",
      KgrlConstants.EDGE_SAMPLING_WEIGHT_NAME: f"abs({edge_version}-
{v_begin})*log2({edge_version}+{v_end})",
   }
sampler_conf = {
    "client_conf": {...},
    "gen_data_conf": {
        "random": True, "fanouts": [50, 20], "buffer_size": 2, "filters": get_filters(10, 20), "weights": get_weights(10, 20),
   },
}
sampler = NodeSubGraphSampler.from_params(sampler_conf)
```

7.5 Summary

This chapter provides a summary introduction to the layered abstraction of the SDK programmable framework. The complete programmable framework is expected to be released in detail in the “SPG Semantic-enhanced Programmable Knowledge Graph Framework” whitepaper 2.0.

Chapter 8 SPG-LLM Layer

At the beginning of 2023, Large Language Models (LLMs) have demonstrated their powerful capabilities, particularly in language understanding and dialogue generation. On the other hand, knowledge graphs excel in addressing factual “illusions” and complex reasoning problems that the LLMs struggle with. By effectively combining the strengths of knowledge graphs and LLMs, we can leverage their respective expertise to provide high-quality AI services and products.

Building upon SPG, the SPG+LLM dual-drive framework is formed by leveraging the structure, semantics, and logical understanding capabilities of LLMs. With SPG's strong schema, logical constraints, and symbolic expression capabilities, the efficiency of the domain knowledge construction and reasoning can be further improved. By integrating the intent understanding, intent diffusion, task construction, and controlled generation based on users' natural language expressions, SPG-LLM enables interactive knowledge querying and reasoning using natural language. This is an ongoing direction of exploration. Base on the strong schema, logical constraints, and symbolic expression capabilities of SPG, further efforts are made to improve the efficiency of the domain knowledge construction and reasoning, accelerate the industrial implementation of knowledge graph, and combine the intent understanding/intent diffusion, task construction, and controlled generation based on users' natural language expressions, to achieve interactive natural language querying and reasoning on the knowledge graph. This chapter provides a brief introduction to the architecture of LLM and SPG for natural language interaction. It also introduces knowledge extraction based on LLM, drawing from the practice of DaGuan Technology.

8.1 SPG-LLM Natural Language Interaction Architecture

Based on the overall architecture defined in Figure 24, the natural language interaction of the large language model (LLM) can be divided into four main parts: LLM Adapter Interface, SPG Constructor for automatic extraction and construction of the knowledge graph, SPG NL Query for natural language querying based on LLM, and SPG NL Reasoner for natural language reasoning based on LLM.

8.2 Automatic Extraction and Automated Construction of Knowledge Graphs

After using LLM, the process of constructing a knowledge graph is as shown in Figure 44.

Figure 44: The Construction Chain of Knowledge Graph

Business Understanding and Schema Design: The design and implementation of the knowledge graph schema require a deep understanding and abstraction of the knowledge in the domain. It also needs to consider the quality and accessibility of the data sources, as well as the requirements and limitations of the

application scenarios. This process is typically a collaborative effort involving multiple parties. The book "Knowledge Graph: Cognitive Intelligence Theory and Practice", provides a series of practical experiences and summarizes them as the "Six Taoist Methods" (as shown in Figure 44). In the schema design process, it is important to make full use of the content of SPG-Schema and further expand it by introducing standardized natural language annotations. The content related to natural language annotations of SPG-Schema will be introduced in future versions of the whitepaper. We can refer to the definitions within Ontology, which allows us to define concepts, properties, relations, constraints, and rules. It also supports inference and validation. Well-known ontology libraries such as schema.org, FIBO, GO, etc., can be referenced or reused to optimize the design of the schema. The goal is to ensure the standardization, consistency, and universality of the designed schema.

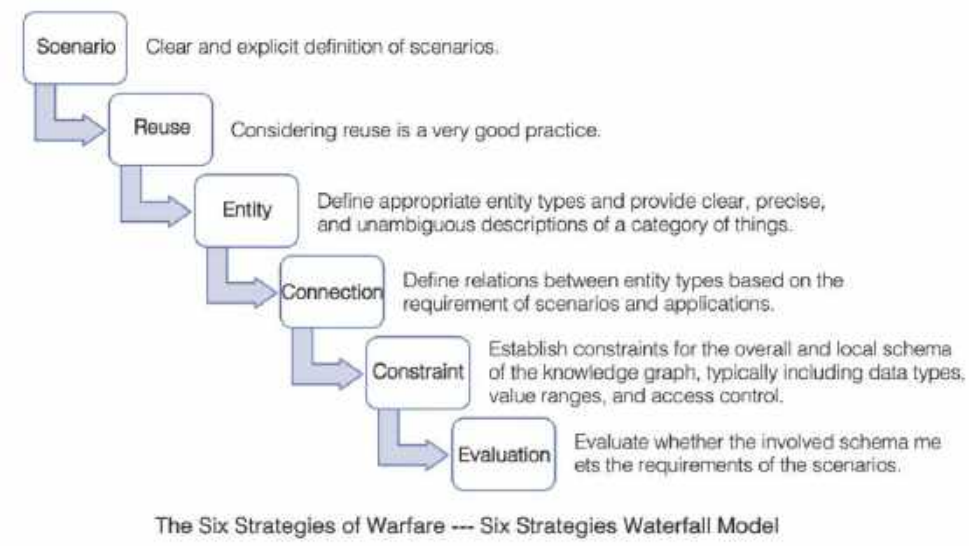

Figure 45: Chapter 2 of the book "Knowledge Graph: Cognitive Intelligence Theory and Practice" [6]

Manually curated Examples and Automatic/Human-Written Prompts: The process of prompt engineering is based on the designed schema to achieve automatic extraction of entities, relations, and properties, thereby constructing a knowledge graph. The engine for automating prompt generation can also be implemented with reference to the reasoning engine in the ontology. The generation of the prompts relies on the natural language annotations in the schema and the manually curated examples. In practice, the manually curated examples or the use of LLM-generated extraction examples can help with few-shot learning to improve the accuracy of LLM extraction.

Figure 46: Example of LLMs Extraction

LLM Extraction and Optional Manual Review: Utilizing LLM to construct a knowledge graph, with the provision of manual review when necessary to ensure the accuracy of the constructed knowledge graph.

Figure 47: Examples of LLMs Extraction and Review

Knowledge Graph Construction: Integrating the results extracted by LLM into an existing knowledge graph, involves using LLM for entity extraction, relation extraction, and other methods to construct the knowledge graph from a large amount of text. The key lies in the specification of the entity types, relation types, property types, and other elements defined in the knowledge graph schema, particularly the relevant natural language annotations. This is strongly related to the specification of the SPG-Schema, which provides natural language annotations in the schema, aiding in their transformation into prompts for LLM extraction and interaction. There should be three levels of content in this regard:

- Natural language annotations for the entity types, relation types, and property types.

- Natural language annotations for the concept hierarchies, semantic correlations, logical rules, and other aspects.
- Standardization of the above two types of annotations, which helps in achieving a common library (engine) for automated prompt generation.

The natural language annotations in the schema serve two purposes: on one hand, they enable the automatic generation of the prompts, and on the other hand, when using LLMs to construct knowledge graphs, they can be used to generate samples for few-shot learning. In practice, few-shot learning is crucial in relation extraction. It is difficult to achieve good performance in zero-shot learning, but few-shot learning can significantly improve the effectiveness of relation extraction.

8.3 Domain Knowledge Completion with LLMs

Using LLMs for knowledge completion can help small and medium-sized institutions acquire richer domain knowledge. Compared to relying solely on internally accumulated knowledge, LLMs leverage vast amounts of text data to obtain comprehensive common knowledge and domain-specific knowledge. By extracting and saving knowledge from LLMs into a knowledge graph using specialized methods, organizations can benefit from more efficient knowledge accumulation and utilization. In contrast to traditional knowledge mining processes that aim to make implicit knowledge explicit in existing knowledge graphs, knowledge completion with LLMs focuses more on extracting domain-specific knowledge from LLMs and integrating them into the knowledge graph, providing knowledge that does not exist in the knowledge graph. This whitepaper only introduces the concept of "knowledge completion" using LLMs, and further versions will provide more implementation methods, examples, significances, and other details.

8.4 Natural Language Knowledge Querying and Intelligent Question Answering

Traditional knowledge graphs have limited natural language understanding capabilities. LLMs can help address this deficiency by leveraging their language understanding and generation capabilities, which are developed through training on billions of parameters. By combining the two, knowledge graphs can understand user queries in natural language and provide accurate answers using their inherent knowledge. The LLM is responsible for semantic analysis, while the knowledge graph provides structured knowledge for answer retrieval, complementing each other. This combination leverages the advantages of structured knowledge in knowledge graphs and the language understanding capabilities of LLMs, thereby providing more user-friendly question answering services. The LLM analyzes the real intent of the query, while the knowledge graph offers rich background knowledge to assist in retrieving more accurate and relevant search results. In dialogue systems, knowledge graphs provide a source of contextual knowledge, making conversations more intelligent and human-like. The LLM handles natural language interaction, while the knowledge graph supplements relevant knowledge, enabling the robot to have stronger contextual awareness.

By combining the powerful capabilities of LLMs and vector search, integrating natural language interaction with knowledge graphs can create controllable, trustworthy, and reliable question answering systems. This approach helps address the "hallucination" problem faced by LLMs and facilitates the implementation of industrial applications, enabling the realization of the "last mile". Please refer to Figure 48 for illustration.

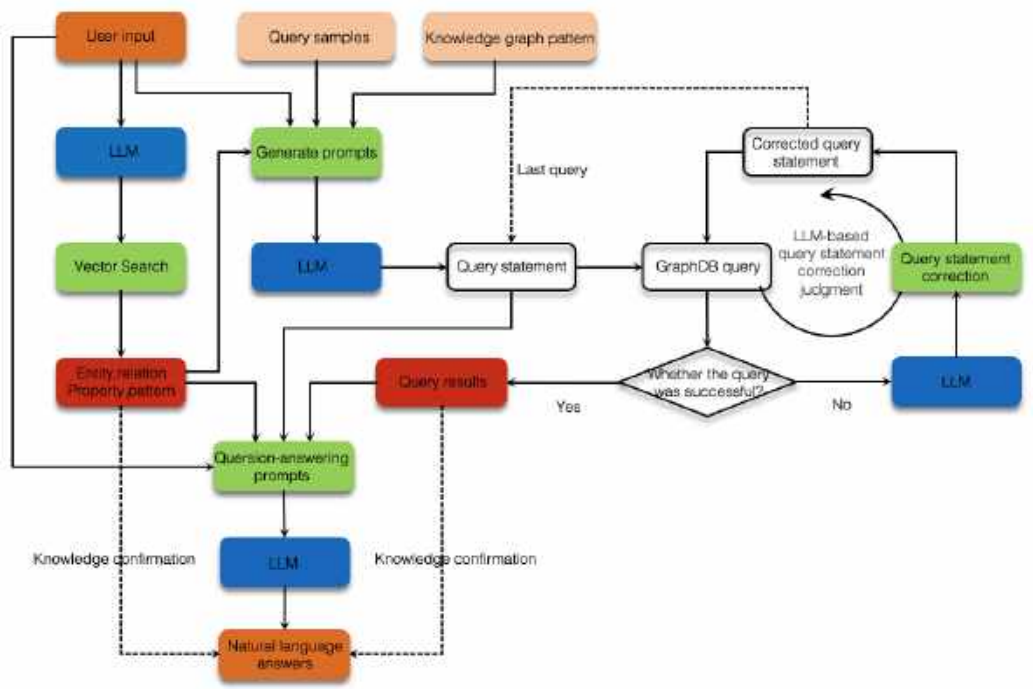

Figure 48: Architecture for controllable, trustworthy, and reliable question answering with the combination of LLMs and knowledge graphs (early draft)

NL2GQL/NL2KGDSL utilizes human-annotated datasets, which contains tens of thousands of natural language - GQL/KGDSL pairs, to perform Semantic Feature-based Training (SFT) on LLM, enabling natural language knowledge querying and intelligent question answering. The focus of this whitepaper release is primarily on theoretical exploration.

In future versions, as GQL or KGDSL mature, related datasets will be released, along with specifications, code repositories, and models for SFT based on open-source large language models. Please stay tuned for further updates.

8.5 Summary

This chapter provides a summary introduction to the basic principles and framework of the SPG-LLM layer. It also introduces the concept of "knowledge completion". The future release of the "Semantic-enhanced Programmable Knowledge Graph Framework (SPG)" whitepaper is expected to focus on providing a comprehensive overview of the SPG-LLM layer.

Chapter 9 New Generation Cognitive Application Cases Driven by SPG

In Chapter 2, we summarized and analyzed the problems in the construction and application of LPG-based knowledge graph in enterprise causal and risk control. This chapter combines the problems raised in Chapter 2 to explain how they are solved based on SPG, and provides an overall solution.

9.1 Enterprise Causal Knowledge Graph Driven by SPG

In this chapter, we take the example of the 2019 dam collapse incident at the Vale of Brazil, mentioned in section 2.3. The incident caused an increase in the price of iron ore, resulting in a rise in steel production costs for downstream companies. In the entire chain of events, companies belonging to the same industry as Vale (i.e., competitors) benefited from the incident and saw an increase in profits. However, this had a negative impact on the downstream of the industry chain, as the rising cost of raw materials led to a decrease in profits for these companies. In response to the issues raised in section 2.3 regarding the application of the enterprise causal knowledge graph based on property graph, we propose a solution using SPG.

1. Addressing the Requirement for Dynamic Event Classification through Derivable Concepts

The enterprise causal knowledge graph involves the evolution and reasoning of events at the conceptual level. Therefore, the first step is to map event instances to their respective event concepts, using the "belongTo" predicate to connect event instances to the corresponding concepts. Since there are numerous event types that are difficult to define in advance, SPG supports the derivation of new concepts through specific combination rules using concepts. This enables dynamic classification of event instances. An example is shown in Figure 49.

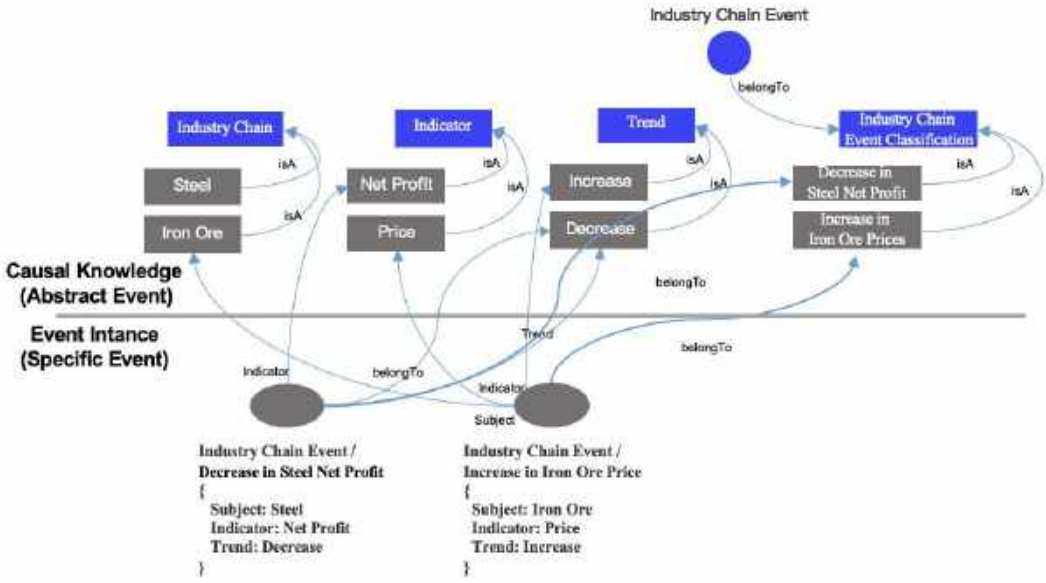

Figure 49: Combination of Concepts

In this example, there are two event instances: “Decrease in Steel Net Profit” and “Increase in Iron Ore Prices”. The former belongs to the event classification of the industry chain, and the concept of “Decrease in Net Profit” can be derived through the combination of the “indicators: Net Profit” and the “trend: Decrease”. Similarly, the latter also belongs to the event classification of the industry chain, and the concept of "Increase in Iron Ore Prices" can be derived through the combination of the “industry chain: Iron Ore”, the “indicator: Prices”, and the “trend: Increase”. It is worth noting that SPG does not immediately derive all possible combinations of concepts but rather derives the corresponding event concepts based on the actual occurrence of event instances, in order to avoid meaningless or logically inconsistent concept systems.

2. Solving the Inability to Express the Entire Event Context through Conceptual Causal Modeling

Building upon the solution to the previous problem, SPG can establish causal relations within the conceptual classification system to express the context of events, as shown in Figure 50. The red dashed lines represent the event propagation automatically generated in the event instance layer after the activation of the “leadTo” predicate in the causal knowledge layer.

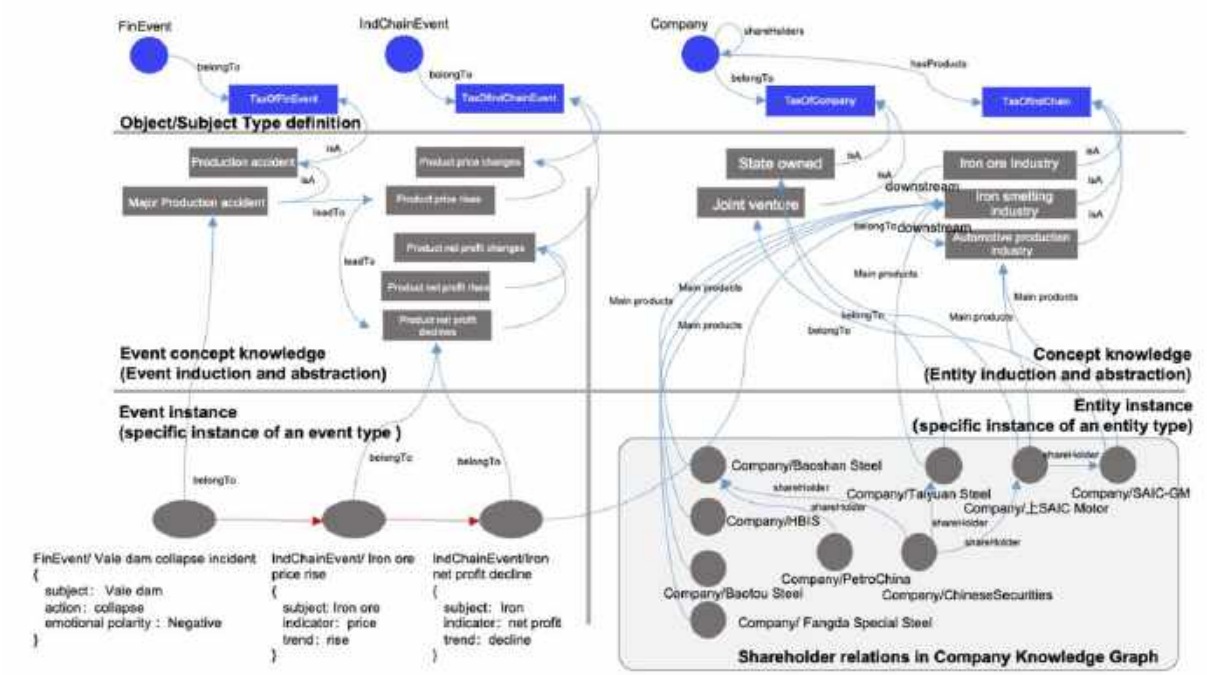

Figure 50: Causal Reasoning

In Figure 50, the diagram is divided into two layers from bottom to top: the event instance layer and the conceptual knowledge layer. The event instance layer represents specific event instances, such as the "Brazilian dam collapse incident," which can be mapped to the conceptual knowledge layer using the method described in 1). The conceptual knowledge layer has the ability to express causal relations at the conceptual level. In this example, the following causal relations have been defined in the conceptual knowledge layer: a “major production accident” in a company leads to either an “increase in iron ore prices” or a “decrease in steel net profit” in the industry chain to which the company belongs. When the Brazilian dam collapse incident occurs and is classified as a “major production accident” concept, the event inference transitions

from the event instance layer to the conceptual knowledge layer. Based on the defined conceptual knowledge, the event propagation mechanism generates a new industry chain event instance at the event instance layer. This new event instance is then classified as the concept of "increase in iron ore prices", making it an "increase in iron ore prices event". The conceptual knowledge can be derived from event induction or generalized from causal patterns, guiding the classification and propagation of specific event instances. For example, a "major production accident" leads to the occurrence of an "industry chain event", which belongs to either the concept of "increase in iron ore prices" or the concept of "decrease in steel net profit". The rule template is as follows:

RULE1: Major Production Accident -leadTo-> Industry Chain Event

RULE2: Industry Chain Event -belongTo-> Increase in Iron Ore Prices

RULE3: Industry Chain Event -belongTo-> Decrease in Steel Net Profit

At this point, events can be classified into derived concepts, allowing the derived concepts that satisfy the rules to form causal relations and ultimately form the causal relations in the conceptual knowledge layer. This expression guides the classification and propagation of the event instances from top to bottom, enabling the expression of the entire event context in the conceptual knowledge layer and the event instance layer. In the example above, the conceptual knowledge layer expresses that accidents lead to a decrease in profits for the downstream companies in the industry to which the entities involved in the accident belong. When concretized at the event instance layer, this means that the "Brazilian dam collapse incident" leds to a decrease in profits for several steel companies.

It is important to note that SPG provides a framework for causal description, and the causal relations between concepts still need to be created by users based on their business characteristics. More detailed examples of concept induction methods will be published in future articles of the SPG, and addressing specific practices.

3. Built-in Logical Expressions to Address Data Unconstrained by Reasoning Logic

SPG expresses conceptual classification logic using the logical constraints. Taking the dam collapse incident as an example, the following rules can be defined to classify this event:

```
Define (s:FinancialEvent)-[p:belongTo]->(o:`FinancialEventTaxonomy/MajorIndustrialAccidents`) {
  Structure {
    (s)-[:std.subject]->(company:Company) //Associated company instances
  }
  Constraint {
    R1("Subject experienced a Industrial accident"): s.behavior == 'Industrial accident'
    R2("Subject company has a market share exceeding x% with significant impact"): company.marketShare > x%
  }
}
```

The above rules can be interpreted as follows: when a safety accident occurs and the market share of the company exceeds x%, then the event is classified as a major production accident in the corresponding

industry. Similarly, the “leadTo” relation between concepts can also be logically expressed. When event “e1” occurs, it will generate and activate another event “e2”.

```
Define (s:`FinancialEventTaxonomy/MajorIndustrialAccidents`)-[p:leadTo]->(o:`IndustryChainEventTaxonomy/
priceIncrease `) {
   Structure {
      (s)-[:std.subject]->(company:Company)-[:industry]->[I:Industry)
   }
   Constraint {
   }
   Action {
      createNode(
         type=IndustryChainEvent
         value={
            subject=I.name
            index='price'
            trend='increase'
         }
      )
   }
}
```

After a new event instance is generated through event propagation, the new event instance can trigger the classification of the event again. The classification of the event can be done using combination concepts. Here is an example that uses the combination concepts of “IndustryChain”, “Index”, and “Trend” to define the classification of the industry chain event mentioned earlier.

```
Define (s: IndustryChainEvent)-[p:belongTo]->(o: `IndustryChainEventTaxonomy`/`IndustryChain` + `Index` + `Trend`) {
   Structure {
   }
   Constraint {
      o = s.subject + s.index + s.trend
   }
}
```

4. Logical Properties and Relations to Address External Dependency on Auxiliary Data

In the example mentioned above, the Company entity has a property called “marketShare”, which represents the market share of the company. In practice, this data may come from other systems and may not exist in the knowledge graph. In such cases, the logical rules can be used to define the source of the data. An example is as follow:

```
Define (s:Compnay)-[p:marketShare]->(o:Float) {
  Structure {
    (s)
  }
 Constraint {
    o = callForMarketShares(s.id, 'marketShare')
  }
}
```

In line 6, the code "callForMarketShares" refers to a user-defined operator that can retrieve the market share information from other systems. Unlike the previous approach of importing all data into the knowledge graph, this method does not require additional copying of data from other systems. It ensures logical consistency of the data. This approach addresses the dependency on external data for decision-making in the context of enterprise causal knowledge graph scenarios.

5. SPG Addresses the Lack of Interpretability in Reasoning Conclusions

In this example, SPG decouples the propagation issue between the conceptual level and the instance level through the definition of event concepts. It also ensures the consistency between the logical rules and data. Following the four-quadrant approach, SPG ensures interpretability in the following four aspects:

- **Interpretability of the generalization and derived logic of the event concept ontology**

At the level of causal modeling, a structured representation scheme is defined for various types of events and entity types. At the same time, a top-down approach is used to define the concepts of entity ontology (such as product classification) and event ontology in a hierarchical system. Each more granular event concept is realized through filling in the event slot values, which concretizes the higher-level event concept (for example, in Figure 50, the "Product Price Change Event" is a specialization concept of the "Industry Chain Event", and fills the slot of "Indicator" with the specific value of "Price". On the other hand, the "Increase in Net Profit" provides further constraints on the concept of "Net Profit Change" in the slot of "Trend". By defining slots and concretizing slot values in a top-down manner, the interpretability of concept semantics is achieved through the combination of slot value properties. Clear generalization and derived logic exist between upper and lower concepts.

- **Interpretability of the logical relations in causal knowledge**

By defining the RULE patterns, the logical relations of causality, succession, and spatiotemporal relations between conceptual events are defined. For example, "an increase in product price" does not necessarily lead to "a decrease in product profit". Through the summary of the domain expert knowledge or analysis of numerous actual cases, rules such as "an increase in the price of upstream raw materials leads to a decrease in net profit of downstream products" can be obtained due to the impact of supply and demand relations in the industry chain. Furthermore, Based on the relations between upstream and downstream in the industry chain, specific causal logic, such as price increase-profit decrease, can be derived and generated in batches. For example, "increase in iron ore price -> decrease in steel profit" and "increase in steel price -> decrease in automobile profit". By defining and applying rules for generating multiple causal knowledge and combining them, an interpretable causal knowledge system can be generated for specific industries and scenarios.

- **Traceability of the factual causality and spatiotemporal succession between event instances**

The event subject, occurrence time, location of each sub-event composing the factual chain, as well as the factual associations, semantic associations, spatiotemporal co-occurrence or succession relations between these elements, provide the basis for the establishment and attribution traceability of the relations

between event instances. In Figure 51, the dam collapse incident at Vale on January 25th, 2019, and the subsequent rise in iron ore prices to a high level in July of the same year, along with the decrease in net profit of Baosteel and Fangda Special Steel, are supported by news reports and financial disclosures. Through the "belongTo" relation between the event instance and the event concept, the "leadTo" relation in the causal knowledge layer, and the industry chain relation between iron ore, steel, and automobiles, it can be explained clearly that these events are not independent but rather form a factual chain that can be explained by the causal relations within the industry chain.

- **Interpretability of the inductive and deduction between factual relations and causal knowledge**

At the level of causal knowledge, the logical relations and common sense relations between abstract and specific ontological concepts are defined. In the level of factual instances, the structured and semantically standardized representation of instance knowledge and the factual associations between events are defined. The representation of factual relations and causal knowledge is decoupled, while using standard predicates in SPG, such as "belongTo", "isA", and "isInstanceOf", providing a unified representation method for the deduction from conceptual events to specific facts and the induction from specific factual relations to causal logic. This association and logical interpretation between abstract concepts and specific facts, can help validate the correctness of causal relations using existing factual relation samples in specific scenarios, and assist in the exploration of hidden causal and succession relations between events using causal logic. For example, by using the generated causal pattern "increase in steel price - decrease in automobile net profit" and integrating the equity penetration relations of the automobile companies in the enterprise knowledge graph, automobile companies with profit decline risks can be discovered.

9.2 Comparison between SPG and LPG in the context of Enterprise Causal Knowledge Graph

Table 13: Comparison of capabilities between SPG and LPG in the context of enterprise causal knowledge graph

Application Phase	Issues in Application	LPG Solutions	SPG Solutions
Causal Definition and Propagation	Causal Definition	Generally replaced by business systems	Supports causal definition based on derived concepts
	Propagation of events between causal layers		Uses specific inference predicates like "leadTo" to describe causal definitions
Event Definition	How events are related to entity graph	There are two main ways: 1) Define events based on subgraphs, which require modeling and data preparation 2) Directly retrieve events from the business system, which requires business system development	Supports native event type definition, simplifying modeling and data preparation without the need for business system development
Event Classification	Single and multiple event classifications	Usually establishing a business system to classify and label events, or not classifying them and directly using analysis tasks instead of classification conclusions.	Supports dynamic classification of events to concepts
	The problem of non-exhaustive event classification		Derives new concepts (abstract events) using combination concepts
Event Propagation	How to describe the impact of an event	Usually customized processing is done in the form of tasks in the business system	Event propagation is achieved based on the linkage between events and concepts.

As shown in the table above, SPG provides a framework for causal expression. In comparison with LPG, it effectively represents the propagation chain of the events, providing a new practical approach for rapid analysis and response to the impacts of financial events.

9.3 SPG-Driven Risk Mining Knowledge Graph

In Chapter 2, the challenges of applying the risk mining knowledge graph were discussed, focusing on the maintenance and management of data as well as the high cost of understanding. The entities and relations mentioned in Chapter 2, can be classified into two categories based on their data generation methods: (1) Basic data, which comes from original table data. Entities such as “Person”, “Phone”, “Cert”, “Device”, “App”, etc., and relations such as “Person-has->Phone” and “Person-has->Cert”, can be directly derived from the original tables. (2) Derived data, which are generated from basic data or other derived data. For example, relations such as “Person-samePhone->Person” and “Person-developed-App” are derived logically. Below, we will discuss in detail how the SPG solution can be applied to the risk mining knowledge graph, and addressing the challenges previously mentioned.

1. To address the issue of data inflation and increased costs after converting raw data into graph data. The original table has significant differences compared to the basic data. For example, the original table only provides the User table and the Application table, without the Device, Certificate, or Phone tables. These pieces of are usually present as fields in the User table and the Application table. Building a knowledge graph using LPG typically requires users to perform additional data transformation or provide mapping operations. SPG, on the other hand, offers standardized property capabilities to simplify user data modeling and reduce data cleaning costs. Here is an example of how to represent phones, devices, and certificates as standard properties:

```
CREATE TYPE (std.Phone {
        value STRING REGEX '^1([38]\d|5[0-35-9]|7[3678])\d{8}$'
});
CREATE TYPE (std.Cert {
        value STRING REGEX '^[a-f0-9]{32}$'
});
CREATE TYPE (std.Device {
        value STRING REGEX '^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$'
});
```

To further define other entities:

```
CREATE NODE ( User {
  id STRING,                    //User primary key
  name STRING,                  //User name
  type STRING,                  //User type, individual or legal entity
  hasPhoneNum std.Phone,        //Using standard property here
  hasCert std.Cert,             //Using standard property here
  hasDevice std.Device          //Using standard property here
});
```

```
CREATE NODE ( App {
  id STRING,
  riskType STRING,              // Risk flag
  hasCert std.Cert,             // Using standard property here
  installDevice std.Device      // Using standard property here
});
```

Due to the use of the standard property to replace relation modeling, there is no explicit definition of relations. Only the import of the user information table and the application information table is required. It can be observed that using the modeling capabilities of SPG, simplifies the modeling cost of entities such as Device and Certificate, and also reduces the cost of the data cleaning.

2. Resolve the issue of duplicate data preparation caused by different business characteristics and support the reuse of knowledge graphs across different businesses. The risk mining knowledge graph requires the use of transfer data and equity data. SPG provides knowledge fusion capabilities, the entities and relations from other knowledge graphs can be referenced and integrated into the current knowledge graph by utilizing self-defined entity resolution operators, to accommodate the specific requirements of the business scenario. These entities and relations can be referenced from the funding knowledge graph and equity knowledge graph. For example, as shown in Figure 51, the fusion of the funding knowledge graph and the risk mining knowledge graph is depicted, where the textual structure of the instance is represented as: Type/PropertyName = PropertyValue.

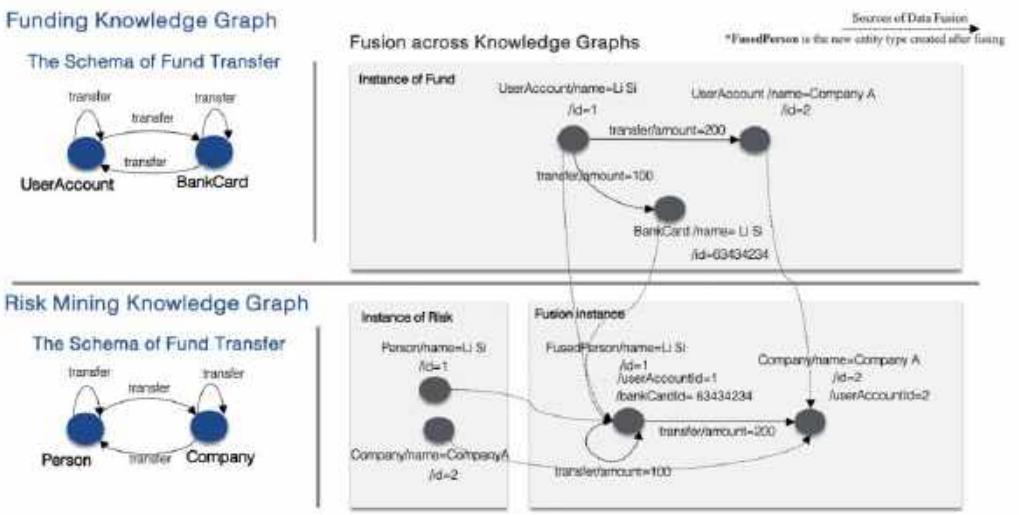

Figure 51: Knowledge Fusion across Knowledge Graphs

"FusedPerson" is derived from the entity linking and resolution operators of "UserAccount" from the funding knowledge graph and "Person" from the risk mining knowledge graph. To achieve this unified relation, two stages need to be defined: entity linking and entity resolution. It is worth noting that "FusedPerson" is only a declaration of the type and the association of the operators, and it does not generate actual fusion instances, which greatly reduces computation and storage costs.

1) Entity linking: This stage primarily defines how the instances of "UserAccount" and "Person" are corresponded using the entity linking operators. It can be a one-to-one correspondence or a many-to-one

correspondence. In this example, it is a many-to-one correspondence. Entity linking across knowledge graphs can be achieved by applying rule-based linking operators. The pseudocode for the operator interface definition is as follows:

```
@BaseOp.register("FusedPersonLinkOp", bind_to="FusedPerson", is_api_iface=True)
class FusedPersonLinkOp(EntityLinkingOp):
  def eval(self, record: Vertex) -> EvalResult[List[Vertex]]:
      pass
```

2) Entity resolution: In this example, entity resolution is achieved using the resolution operators. Based on the conditional expression rules, the properties, relations can be filtered and processed for the successfully mapped instances of “UserAccount” and “Person”. The pseudocode for the operator interface definition is as follows:

```
@BaseOp.register("PersonFuseOp", bind_to="FusedPerson", is_api_iface=True)
class FusedPersonFuseOp(EntityFuseOp):
   def eval(
      self, source_vertex: Vertex, target_vertexes: List[Vertex]
   ) -> EvalResult[List[Vertex]]:
      pass
```

3) Addressing the issue of inconsistency in relation data due to logical dependencies. SPG provides capabilities for derived relations and derived data. Let's take the example of “same phone number” and “same person” as the logical dependencies.

```
Define (s:Person)-[p:samePhone]->(o:Person) {
   Structure {
      (s)-[:hasPhoneNum]->(w:std.Phone]<-[:hasPhoneNum]-(o)
   }
   Constraint {   }
}
```

The pre-defined relations types can be reused in the logical rules of KGDSL. For example, the “same person” relation can be defined as two individuals having the same phone number and the same device simultaneously.

```
Define (s:Person)-[p:sameUser]->(o:Person) {
   Structure {
      (s)-[:hasPhoneNum]->(o),  (s)-[:hasDevice]->(o)
   }
   Constraint {   }
}
```

The complex corporate control relations can also be defined through transitive, as shown in Figure 52, where the textual structure of the instance is represented as: Type/PropertyName = PropertyValue.

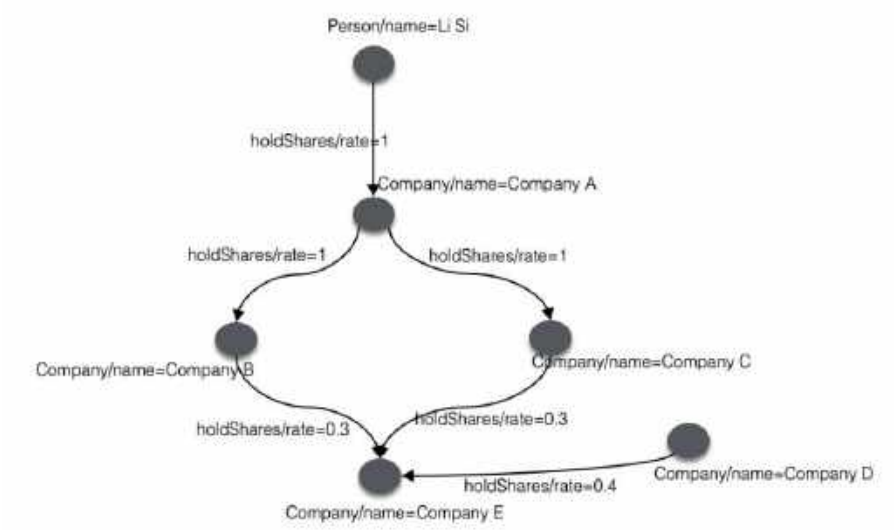

Figure 52: Example of holdShare Relations

```
// Define the holding proportion first, the transitive predicate requires that the types in the GraphStructure must be the same.
Define transitive (s:Company)-[p:holdShares]->(o:Company) {
   Structure {
      // The graph structure must follow the following pattern to express transitivity
      (s)-[p1:holdShares]->(c:Company), (c)-[p2:holdShares]->(o)
   }
   Constraint {
      // Group and aggregate by entity s and o to obtain all actual equity.
      real_rate("The actual holding proportion") = group(s,o).sum(p1.shares*p2.shares)
      p.shares = real_rate  // assigning the actual shares
   }
}
Define (s:Person)-[p: indirectHolding]->(o:Company) {
   Structure {
      (s)-[p1:holdShares]->(c:Company)-[p2:holdShares]->(o)    // Indirect control of company o through company c
   }
   Constraint {
      R1("Direct controlling equity ratio must be greater than 50%"): p1.shares > 0.5
      R2("Indirect controlling equity ratio must be greater than 50%"): p2.shares > 0.5
   }
}
```

By extension, all relations can be completed based on expert rules.

4) Overcoming the problem of continuously expanding and ultimately unmaintainable schema and data in the face of iterative business evolution.

In traditional LPG models, data and schema are tightly coupled. If there are changes in the business, the schema needs to be modified accordingly, resulting in high costs. However, SPG provides dynamic classification capabilities based on concepts, allowing for business extensions at the conceptual level, as shown in Figure 53.

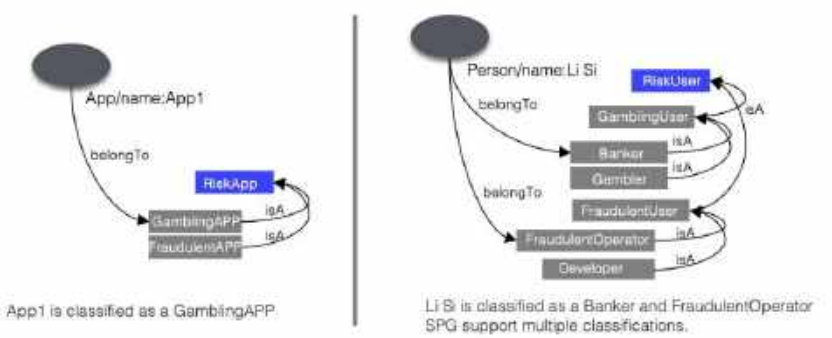

Figure 53: Concept-based dynamic entity classification in the risk mining knowledge graph

We introduce a new reasoning predicate called “belongTo”, which links the entities and concepts that meet the rule requirements. For example, when an App is confirmed as a fraudulent App, its developer can be classified as a suspicious fraudster. Taking "Fraudster" as an example, the rule for dynamic classification can be defined as follows:

```
Define (s:Person)-[p:belongTo]->(o:TaxonomyOfRiskUser/Fraudster) {
  Structure {
    // The person developing fraudulent applications is a fraudster.
    (A:App)-[:developer]->(s)
  }
  Constraint {
    R1("It is a fraudulent application"): A.type == 'fraud'
  }
}
```

The association between concepts and entities can be described using expert rules, which helps address the problem of tight coupling between business and actual data. This approach avoids the need to modify underlying data when there are changes in the business requirements and reduces the direct perception of the changes in underlying data at the business application layer. By strongly binding concepts with the business, concepts can be used as types for the business users. Taking the example mentioned above, let's consider using "Fraudster" as a entity type. Here's an example of how it can be implemented:

```
MATCH
  (u:TaxonomyOfRiskUser/Fraudster)
RETURN
  u
```

Other scenarios for SPG-based risk mining applications.

1. Knowledge Query.

When a certain app is identified as a gambling application (possibly from user complaints or other security events), we can find the organization behind the app through the following query statement.

```
MATCH
   (a:App)-[:developer|boss]->(u:Person)
WHERE
  a.id = 'gambling application 1'
```

```
RETURN
  u
```

2. Fusion learning of neural symbols

The fusion of deep learning and rules has always been a hot and challenging research topic. Deep learning can solve many representation learning problems, such as image classification tasks, while rules (symbolic logic) can handle many explicit reasoning problems. There are two main application directions for existing neural-symbol integration in reasoning.

1) Fusion methods of rules and neural networks.

From the perspective of rule priors, these methods can be classified into two categories:

First category: constraining model structure with rules. Typical methods include DeepProbLog [23], neuro-symbolic forward reasoning (NSFS) [24], Logical Neural Networks (LNN) [25], and LogicMP [26].

Second category: constraining objectives with rules. The rules are treated as prior knowledge and are incorporated into the objective function as a penalty term. Typical methods include SemanticLoss [27] and NCLF [28].

Whether it is the first category or the second category, the first-order predicate forms are required as the input form for rules. Various rules mentioned in this paper can be converted to and from the first-order predicate forms, for example.

```
Define (s:Person)-[p:belongTo]->(o:Fraudster) {
  Structure {
    (A:App)-[:developer]->(s)
  }
  Constraint {
    R1("It is a fraudulent application"): A.type == 'fraud'
  }
}
```

The conversion of first-order predicates is as follows:

```
forall s: exits a:  developer(a, s) & type(a) == 'fraud' -> belongTo(s) == Fraudster
```

In addition, the expertise of business experts, which belongs to hard rules, can easily lead to low recall rates. In LogicMP, specific rule content can be softened to improve rule coverage.

2) Representing relations between symbols as a graph structure and performing reasoning through graph algorithms.

In this example, the graphical form generated by logical rule definitions can be input into the graph algorithms for training. This approach decouples neural and symbolic methods through the form of a graph, ensuring scalability and flexibility.

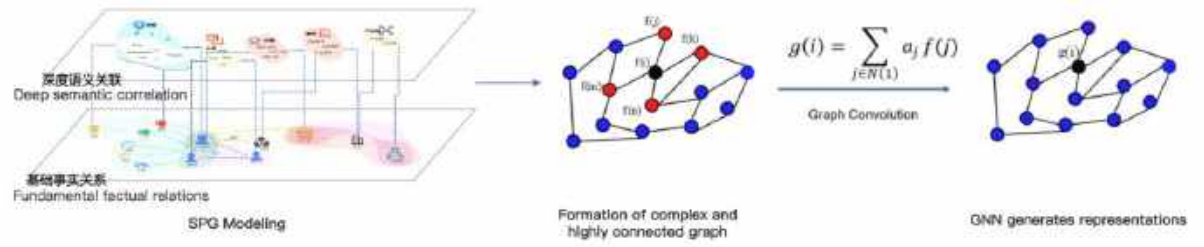

Figure 54: Combine the SPG with the GCN algorithm

9.4 Comparison between SPG and LPG in the Risk Mining Knowledge Graph

From the perspectives of knowledge construction, knowledge application, and knowledge evolution, we can compare the advantages and disadvantages of SPG and LPG.

Table 14: Comparison of abilities between SPG and LPG in the Risk Mining Knowledge Graph

Application Phase	Application Approach	LPG Solution	SPG Solution
Knowledge Construction	Import of vertices, edges, and properties	Provides mapping solutions, where mapping needs to be specified for vertices, edges, and properties	Uses semantic modeling instead of mapping, supports Table2SPG, uses logical properties/relations instead of importing derived data
	Data Consistency Maintenance	Defines schema constraints, typical solution is PG-Key	Maintains data consistency using logical dependencies, effectively reducing the risk of data inconsistency
	Graph Data Fusion	Not supported, can only import one graph or re-import	Supports cross-graph fusion, generates the required data for the current knowledge graph without additional data processing costs
Knowledge Application	Graph Analysis	Provides graph query language, but users need to write analysis language based on the underlying data	Provides graph query language, allows users to use logical relations and properties to shield complex underlying data, users only need to focus on the required information, reducing user learning curve and easier maintenance
Knowledge Evolution	Adding, deleting, or modifying the entity/relation classifications	Requires schema changes and data re-import, which incurs high costs for modifications	Based on the expressive power of concepts, changing concepts can achieve entity classification goals without the need for schema changes and data re-import.

SPG focuses on addressing the cost issue for users of the knowledge graph in the field of risk control. It effectively improves user efficiency and reduces usage costs in all stages of knowledge construction, knowledge application, and knowledge evolution.

9.5 Summary

This chapter primarily discussed the applications of SPG in the enterprise causal knowledge graph and the risk mining knowledge graph. In the enterprise causal knowledge graph, SPG provides an event expression framework that effectively describes the impact propagation relations of events, resulting in timely and effective conclusions, which complements the limitations of LPG in causal knowledge graph applications. In the risk mining knowledge graph, SPG primarily addresses the issues of user usage costs and efficiency, solving problems such as the difficulty of ensuring data consistency, high evolution costs, and high comprehension costs in practical applications of LPG.

Chapter 10 SPG Embracing the New Era of Cognitive Intelligence

The future trend of enterprise digitalization and intelligent upgrading is to build knowledge based on the massive business data accumulated in the enterprise's big data system, promoting data knowledgeization. By integrating business data with AI systems, business intelligence can be achieved. Property graphs have the advantage of compatibility with big data systems, and SPG, based on property graph, aims to accelerate the data knowledgeization and the organic integration of knowledge and AI systems. This chapter combines the core capabilities and two case studies introduced in previous chapters to summarize and analyze the strengths, limitations, opportunities, and challenges of SPG.

10.1 SWOT Analysis of SPG Compared to Property Graphs

Based on the previous descriptions, we can analyze the strengths, weaknesses, opportunities, and threats of SPG from the four quadrants of SWOT.

- **Strengths of SPG.** (1) SPG has a low cost and is compatible with big data architecture, allowing for the rapid construction of domain knowledge graphs based on structured data accumulated in enterprise-level applications. (2) The hierarchical semantic model of SPG supports the continuous evolution of incomplete knowledge graphs, meeting the requirements of rapid business deployment, continuous data accumulation and improvement, and gradual technological application in industrial applications. (3) SPG overcomes the semantic limitations of LPG and effectively connects big data with LPG node/edge structures, enhancing semantic connections and better integrating AI technologies.
- **Weaknesses of SPG.** SPG is still in the growth stage, and there are some compromises and weaknesses in the design of its capabilities, which we need to continuously overcome in the later stage. (1) How does dynamic classification achieve inheritance and extension? Currently, the dynamic classification model effectively addresses the granularity issue of types but has limitations in application, making it difficult to extend properties under new subclasses. (2) Continuous improvement of the built-in semantic structures of the entity is necessary. The current built-in semantic structures in the entity model are not sufficiently enriched, with definitions and constraints limited to event subjects/objects/times and hierarchical concepts. To meet the demands of controlled generation and interpretable reasoning, clear expression of built-in semantic structures within entities requires ongoing improvement in collaboration with downstream applications. (3) Instance-concept linkage reasoning model. While SPG possesses some inductive reasoning capabilities from instances to concepts, there is still significant room for improvement in the co-conduction of concepts and instances and the deductive reasoning from concepts to instances. This requires continuous optimization through the application and refinement of various causal knowledge graphs.

- **Opportunities for SPG.** (1) Filling the gap in semantic frameworks for enterprise knowledge graph applications. RDF/OWL, due to their complexity, have not effectively landed in enterprises. Establishing enterprise-level standardization to facilitate cross-entity knowledge semantic alignment, and to promote the circulation, interoperability, exchange, and sharing of knowledge more conveniently. (2) Driving the development of a universal engine architecture for building knowledge graphs. This promotes the democratization and accessibility of the knowledge graph technology. Large-scale applications in various technical fields rely on standardization and framework development, as seen in search engines, deep learning, and cloud computing. (3) Bridging the gap between knowledge graphs and LLMs. Enterprises can quickly incubate/fine-tune new pre-training models based on Transformer or open-source LLMs. Using the standardized symbols of SPG, efficient knowledge injection, associative prompts, knowledge queries, and other tasks can be achieved during the pre-training, SFT/RLHF, and inference stages, and forming a stable paradigm for the interaction between knowledge graphs and LLMs. Additionally, by data knowledgeization, we aim to construct a symbolic world domain knowledge system that complements and is equivalent to the neural network-based LLM knowledge system.
- **Challenges for SPG.** (1) Performance challenges in scaling applications, particularly during the knowledge construction phase. The performance overhead of extraction models and entity linking can significantly impact the efficiency of large-scale knowledge graph construction. (2) Further refinement of system capabilities. The capabilities of the SPG system need continuous optimization in conjunction with more business scenario and applications. (3) Cultivating semantic understanding in users' mental models. On one hand, there is a need to continuously improve users' understanding of semantics, and on the other hand, efforts should be made to reduce users' perception of semantics.

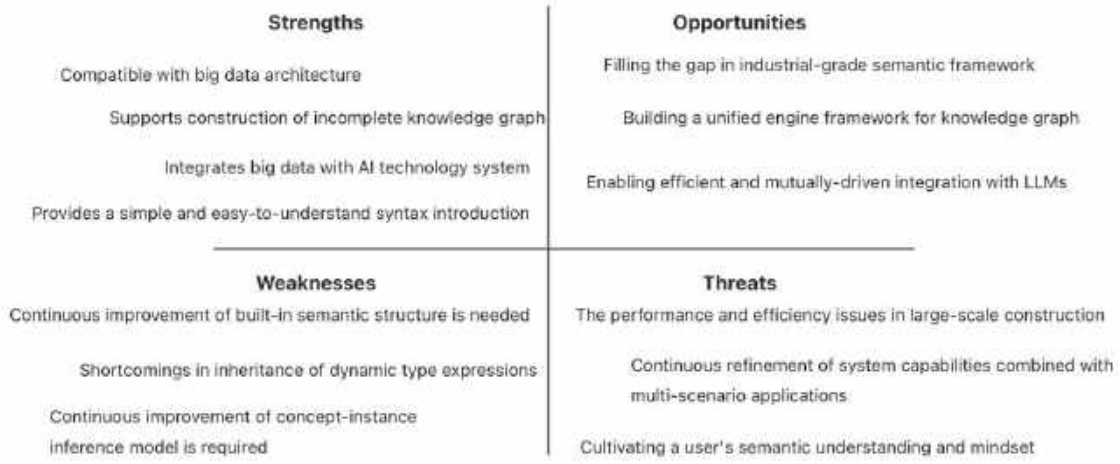

Figure 55: SWOT Analysis of SPG

10.2 Problem Resolution and Outstanding Issues from Chapter 2

Table 15: Fundamental Problems of Knowledge Management Based on LPG and the Resolution Status by SPG

Typical Problems of LPG		SPG Solutions	SPG Status
Type Management	Semantic deficiency in subjects	Event, entity, and concept classification	Supported
	Type granularity issue	Dynamic types	Supported
	Difficulty in property/relation selection	Standard properties, concept types	Supported
	Entities of the same type with different names	Knowledge fusion	Supported
Incompleteness Knowledge Graph Construction	Entity construction from multiple data sources	Entity linking, entity resolution	Partial support
	Knowledge reuse across multiple knowledge graphs	Knowledge fusion	Supported
Logical Dependencies	Semantic deficiency in logical predicates	Semantic predicates with rule conditions	Partial support
	Inconsistency due to separation of definition and logic	Logical Derivation	Supported
	Window-type property/relation explosion	Window standard types, variable parameter properties	Supported
	Inconsistent property logic	Logical properties	Supported
	Inconsistent relation logic	Logical relations	Supported
Logical reasoning of causal	Obstruction in event logic propagation	Activating new event instances during causal deduction	Supported
	Event Classification	Dynamic types	Supported
	Semantic Composition of Concepts	Dynamic composition based on event and entity facts	Supported

It is important to note that Table 15 primarily lists the fundamental problems of knowledge management based on LPG and the resolution status by SPG. It mainly focuses on the semantic aspects of entities and logical predicate semantics. The programmable framework and complex knowledge reasoning are built upon a knowledge management framework in a virtuous cycle. They are not included in the basic capabilities of knowledge management and are not listed in this table. However, they will be further described in the future release plan in Chapter 11.

Chapter 11 Outlook on the Future of SPG

This whitepaper has addressed the challenges faced by enterprise-level knowledge management and discussed the higher requirements for knowledge semantic representation and engine frameworks due to changes in demand paradigms in enterprise-level knowledge graph applications. In Chapter 1, we summarized some of the key problems that still exist in the development of knowledge graph technology:

- Lack of unified semantic representation. Currently, strong semantic knowledge graphs have not achieved industrial implementation based on RDF/OWL, while weak semantic property graphs (LPG) are widely used in industrial-grade knowledge graphs.
- Multiple tools but lack of standardization. The development of customized extraction algorithms/entity linking algorithms for each data set, graph database-backed graph storage, representation learning tools, fuzzy retrieval tools, knowledge query tools, and other tools have led to significant dispersion and inconvenience in the application of knowledge graph technology.

In order to achieve large-scale industrial application of any complex technology, it is necessary to have a unified technical framework that shields complex technical details and supports rapid deployment of new businesses. It also requires a modular architecture that allows for layering and decoupling of domain models and core engines, enabling fast migration to new domains. The same applies to knowledge graphs. The development of knowledge graph technology needs to keep up with the times. SPG defines an industrial-grade, user-friendly knowledge semantic framework for strong semantic knowledge graphs. It helps enterprises accelerate the knowledgeization of massive amounts of data. Through the unified technical framework and engine architecture provided by the SPG knowledge engine, the technology can be standardized, democratized, and made accessible to a wide range of users.

Looking towards the future, knowledge graphs have vast application potential. On one hand, as the best modeling practice for structured data, knowledge graphs can unify data modeling from various perspectives such as machines, algorithms, engineering, business, and operations. They can build next-generation data architectures in line with the concept of data fabric, accelerating the knowledgeization of massive enterprise data, connecting data silos, discovering implicit relations, unlocking the full value of data, and reducing the cost of finding and using data, ultimately bringing greater growth opportunities for businesses. On the other hand, knowledge graphs complement LLMs perfectly. Knowledge graphs have characteristics such as strong facts, weak generalization, strong interpretability, low computational cost, and high construction cost. In contrast, LLMs have weak facts, strong generalization, poor interpretability, high computational cost, and strong semantic understanding. In the future, the goal is to achieve efficient cooperation and complementarity between unified knowledge symbol representation and engine architecture and LLMs. Through the advancement of LLM technology, the cost of knowledge graph construction can be further reduced, accelerating data knowledgeization and providing additional domain knowledge for controlled generation based on LLMs. The construction of massive common-sense domain knowledge bases can also accelerate the progress of general artificial intelligence. Realizing the integration and complementarity of

the knowledge graphs and LLMs strongly relies on a comprehensive knowledge graph and LLM technology stack. Currently, LLM technology has matured, and with the strong semantic knowledge graph framework defined by SPG, it is expected to form a seamless application framework that can be seamlessly integrated with LLMs. This framework will enable industrial-level, generalizable, highly robust, and interpretable comprehensive artificial intelligence technologies based on knowledge graphs and LLMs.

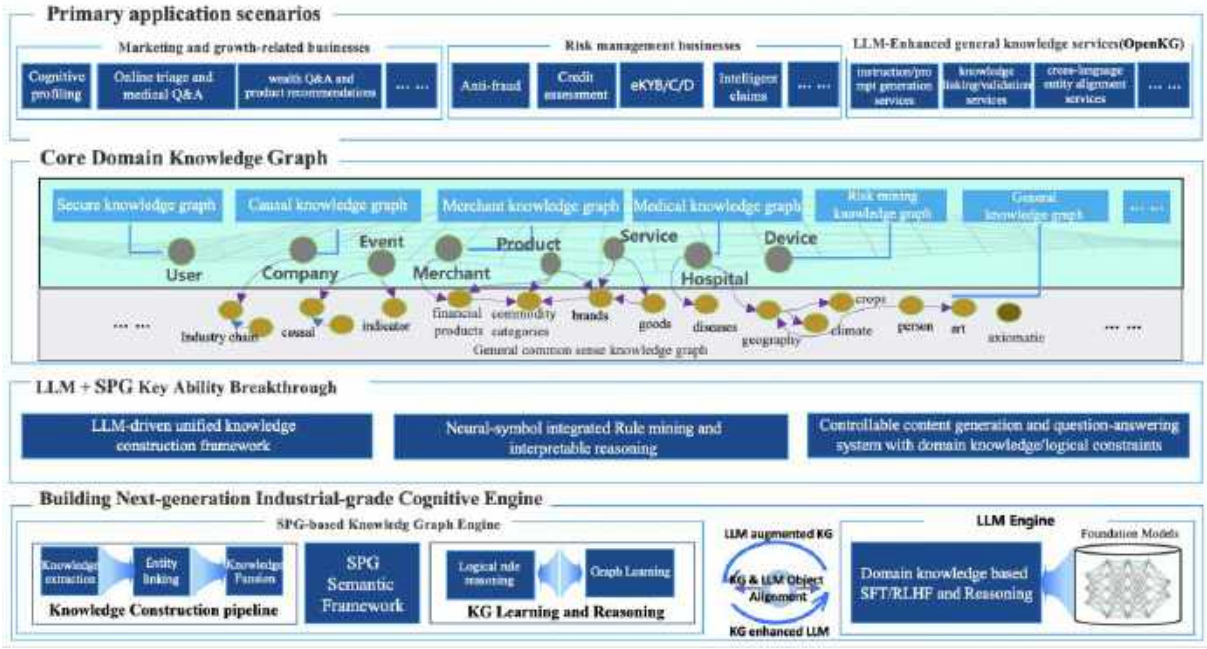

Figure 56: Future Outlook on the Dual-Drive Technical Paradigm of SPG and LLM

The fusion of symbolic logic and neural networks has always been a research hotspot in the industry. One common approach is to use neural networks to learn the rules and relations in symbolic logic, enabling them to better handle complex logical problems. Another approach is to use symbolic logic to guide the learning process of neural networks, improving their accuracy and interpretability. Knowledge graphs, as a typical representative of symbolic logic, have unique advantages in structural representation, semantic characterization, and knowledge association. The unified semantic framework provided by SPG can provide them with stronger vitality. Currently, the fusion of neural networks and symbolic logic mainly occurs in the knowledge reasoning stage. With the emergence of LLMs, new ideas are provided for the integration of symbolic logic and neural networks. On one hand, knowledge graphs, as the underlying support for semantic representation of symbolic logic and knowledge data management, can leverage the powerful semantic understanding capabilities of LLMs and the strong structure and semantics of knowledge graphs to automate prompts and sample construction. This can help the knowledge graphs form a unified knowledge extraction framework and accelerate the knowledgeization of data. On the other hand, in the content generation stage, applying domain knowledge data with strong semantic constraints can effectively avoid the problem of hallucinations and nonsense in LLMs. These issues are expected to be accelerated addressed in the SPG + LLM paradigm. We will continue to improve the expressive capabilities of SPG through industrial practices and enhance LLM through SPG to achieve alignment with objective facts, effectively avoiding/reducing model hallucinations. At the same time, LLM will also enhance SPG to improve the conversion efficiency

of data knowledgeization. We are committed to building a next-generation artificial intelligence engine driven and enhanced by both SPG and LLM.

Table 16: Future Release Plan for SPG

Stage	Release Content	Release Sub-items
Stage 1	"Semantic-Enhanced Programmable Knowledge Graph (SPG) White Paper" V1.0 — Next-generation industrial-grade knowledge semantic framework. Introduce the background, specifications, framework, and case studies of SPG knowledge semantic framework.	1.Overall framework of SPG 2.Two case studies with rich content 3.SPG-Schema Core 4.Basic requirements of SPG-Engine 5.Basic requirements of SPG-controller 6.Basic framework of SPG-Programming
	Establish SPG community and welcome its establishment	1.Establishment of community operation mechanism
	Open-source SPG Engine v1.0	1.Build an open-source engine that meets the requirements of White Paper 1.0 2.Create an open-source community based on the engine 3.Default adaptation to Tugraph technology stack, support vendor extensions
	Recommendation of standardization for SPG [Industry, National, International]	1.Contact various standardization organizations to promote standardization projects
Stage 2	"Semantic-Enhanced Programmable Knowledge Graph (SPG) White Paper" V2.0 — Next-generation industrial-grade knowledge semantic framework. Introduce SPG programmable framework Operator, SDK, case studies, etc., open up a complete knowledge construction framework.	1.Improved SPG-schema 2.Complete definition of operators and framework in SPG programmable framework 3.Support mapping-based [programmable] knowledge graph construction, support search engines, vector retrieval, model services, etc. 4.Basic inference solution, including basic query language for construction, query, and inference 5.Complete storage solution [standalone] 6.Basic requirements of SPG+LLM and exploration in knowledge extraction/NL2KGDSL
	Release a query language that meets semantic and programmable requirements	1.Release GQL or KGDSL that meets the basic requirements of SPG applications 2.CRUD operations in SPG 3.Support basic natural language queries
	Open-source framework V2.0	1.Complete open-source implementation of SPG framework including White Paper 2.0 2.Query language implementation module [GQL or KGDSL] 3.Programmable SDK
	Publish SPG industry and international standards	1.Finalize the standard draft 2.Publicize the standard draft
Stage 3	"Semantic-Enhanced Programmable Knowledge Graph (SPG) White Paper" V3.0 — Next-generation industrial-grade knowledge semantic framework. Introduce SPG inference engine, graph learning, rule-guided interpretable inference, etc., open up complete knowledge inference capabilities.	1.Programmable module to accumulate reusable model hubs, layer hubs, etc., supporting out-of-the-box use 2.SPG industry extensions for domains such as space-time, healthcare, etc. 3.Comprehensive syntax set for rule-based inference engine 4.SPG and AGL linkage to support knowledge inference, interpretable inference, etc.
	Open-source framework V3.0	1.Fully match the final standard draft and SPG V3.0 2.Extension modules
	Knowledge construction and open-source solutions based on common sense knowledge using SPG	Build an open-source, crowdsourced, service community for the healthy growth of common sense knowledge based on OpenKG
Stage 4	"Semantic-Enhanced Programmable Knowledge Graph (SPG) White Paper" V4.0 — Next-generation industrial-grade knowledge semantic framework. Introduce the paradigm of SPG + LLM double-driven and construct a controllable LLM technology system.	1.Complete paradigm and system framework of SPG + LLM 2.Basic natural language interface for construction, query, and inference 3.Natural language interaction module [based on LLM] 4.Supports extractive construction (based on LLM)
	Knowledge services for common sense knowledge based on SPG.	Build an enhanced knowledge services with LLMs, such as instruction/prompt generation services, entity linking/validation services, etc., to promote knowledge element exchange and growth.

In the future, we will continue to upgrade SPG. Table 16 represents our planned release content, and the release timeline will be updated on the SPG official account: "Semantic-Enhanced Programmable Knowledge Graph Framework". We welcome your attention and interaction, and together, we can explore the industrial-grade knowledge graph architecture paradigm.

References

[1] Martin, S., Szekely, B., Allemang, D. (2021). The Rise of the Knowledge Graph. O'Reilly.
[2] 王昊奋, 丁军, 胡芳槐, & 王鑫. (2020). 大规模企业级知识图谱实践综述. 计算机工程, 46(7), 13.
[3] 王昊奋, 漆桂林, 陈华钧 (2019). 知识图谱：方法、实践与应用. 电子工业出版社
[4] 中国知识图谱行业研究报告 [OL]. 艾瑞咨询, 2022.
[5] 陆锋,诸云强,张雪英.时空知识图谱研究进展与展望[J].地球信息科学学报,2023,25(6):1091-1105. [Lu F, Zhu Y Q, Zhang X Y.Spatiotemporal knowledge graph: Advances and perspectives[J]. Journal of Geo-information Science, 2023,25(6):1091-1105.] DOI:10.12082/dqxxkx.2023.230154
[6] 王文广. (2022). 知识图谱：认知智能理论与实战. 电子工业出版社.
[7] Colas, Anthony, M. Alvandipour , and D. Z. Wang . "GAP: A Graph-aware Language Model Framework for Knowledge Graph-to-Text Generation." (2022).
[8] 王昊奋, 王萌. "神经+符号"：从知识图谱角度看认知推理的发展[J].中国计算机学会通讯,2020, 16(8), 52.
[9] Yang, L., Chen, H., Li, Z., Ding, X., & Wu, X. (2023). ChatGPT is not Enough: Enhancing Large Language Models with Knowledge Graphs for Fact-aware Language Modeling. *arXiv preprint arXiv:2306.11489*.
[10] Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., & Wu, X. (2023). Unifying Large Language Models and Knowledge Graphs: A Roadmap. *arXiv preprint arXiv:2306.08302*.
[11] 王文广，王昊奋. 融合大模型的多模态知识图谱及在金融业的应用[j]. 人工智能,2023(02).
[12] Bretto A. Hypergraph theory[J]. An introduction. Mathematical Engineering. Cham: Springer, 2013, 1.
[13] Ferraz de Arruda G, Tizzani M, Moreno Y. Phase transitions and stability of dynamical processes on hypergraphs[J]. Communications Physics, 2021, 4(1): 24.
[14] RDF-star Working Group Charter. https://www.w3.org/2022/08/rdf-star-wg-charter/
[15] 白硕. 事理图谱六问六答 [OL]. 理深科技时评,2019.
[16] RenzoAngles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Bei Li, Jan Hidders, Alastair Green, Leonid Libkin, Victor Marsault,Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savković, Michael Schmidt, Juan Sequeda, Sławek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoč, Mingxi Wu, and Dušan Živković. 2023. PG-Schemas: Schemas for Property Graphs. In Proceedings of the 2023 International Conference on Management of Data (SIGMOD '23), June 18 – 23, 2023, Seattle, USA. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3589778
[17] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savković, Michael Schmidt, Juan Sequeda, Sławek Staworko, and Dominik Tomaszuk. 2021. PG-Keys: Keys for Property Graphs. In Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20 – 25, 2021, Virtual Event, China. ACM, 2423-2436.
[18] Munoz-Venegas S , Perez J ,Gutiérrez, Claudio.Simple and Efficient Minimal Rdfs[J].Social Science Electronic Publishing[2023-08-12].DOI:10.2139/ssrn.3199430.
[19] The GQL Standards Website, https://www.gqlstandards.org/
[20] Jana Giceva and Mohammad Sadoghi. 2019. Hybrid OLTP and OLAP. In Encyclopedia of Big Data Technologies. Springer. https://doi.org/10.1007/ 978-3-319-63962-8_179-1
[21] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing[C]//9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012: 15-28.
[22] Francis N, Green A, Guagliardo P, et al. Cypher: An evolving query language for property graphs[C]//Proceedings of the 2018 international conference on management of data. 2018: 1433-1445.
[23] Manhaeve R , Dumani S , Kimmig A ,et al.Neural probabilistic logic programming in DeepProbLog[J].Artificial Intelligence, 2021:103504.DOI:10.1016/j.artint.2021.103504.
[24] Shindo H, Dhami D S, Kersting K. Neuro-Symbolic Forward Reasoning[J]. 2021.DOI:10.48550/arXiv.2110.09383.
[25] Riegel R, Gray A, Luus F, et al.Logical Neural Networks[J]. 2020.DOI:10.48550/arXiv.2006.13155.
[26] Weidi Xu, Jianshan He, Jingwei Wang, Hongting Zhou, Xiaopei Wan, Taifeng Wang, Ruopeng Li, Wei Chu, An Efficient Mean-field Approach to High-Order Markov Logic. https://openreview.net/forum?id=7UrHaeZ5Ie7
[27] Xu J , Zhang Z , Friedman T ,et al.A Semantic Loss Function for Deep Learning with Symbolic Knowledge[J]. 2017.DOI:10.48550/arXiv.1711.11157.

[28] Zhang Y, Chen X, Yang Y, et al. Efficient Probabilistic Logic Reasoning with Graph Neural Networks[J]. arXiv preprint arXiv:2001.11850, 2020.