*K*nowledge graph-based system f*o*r technical d*o*cument retrieval : a deductive reas*o*ning-f*o*cused expl*o*rati*o*n

Matthias Sesboué

Normandie Université

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité **INFORMATIQUE**

Préparée au sein de l'**INSA Rouen Normandie**

Knowledge Graph-based System for Technical Document Retrieval A deductive reasoning-focused exploration

Présentée et soutenue par

MATTHIAS SESBOUE

Thèse soutenue le 05/09/2024

devant le jury composé de :

MME CECILIA ZANNI-MERK	PROFESSEUR DES UNIVERSITÉS - INSA Rouen Normandie (INSA)	Directeur de thèse
MME FATIMA SOUALMIA	PROFESSEUR DES UNIVERSITÉS - Université de Rouen Normandie (URN)	Président du jury
M. NICOLAS DELESTRE	MAÎTRE DE CONFÉRENCES - INSA de Rouen Normandie	Co-encadrant de thèse
M. JEAN-PHILIPPE KOTOWICZ	MAÎTRE DE CONFÉRENCES - INSA de Rouen Normandie	Co-encadrant de thèse
MME ROSSI SETCHI	PROFESSEUR - University of Cardiff, Royaume-Uni	Membre
M. GRÉGORY ZACHAREWICZ	PROFESSEUR DES UNIVERSITÉS - Ecole de Mines d'Ales	Membre
MME CASSIA TROJAHN	MAITRE DE CONFERENCES DES UNIVERSITES HDR - Université Toulouse II	Rapporteur
MME EDLIRA VAKAJ	ASSOCIATE PROFESSOR - Birmingham City University, Royaume-Uni	Rapporteur

Thèse dirigée par **CECILIA ZANNI-MERK** (LABORATOIRE D'INFORMATIQUE DE TRAITEMENT DE L'INFORMATION ET DES SYSTEMES)

MIIS

litis

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité Informatique

Préparée au sein de l'Institut National des Sciences Appliquées Rouen Normandie

Knowledge Graph-based System for Technical Document Retrieval
A deductive reasoning-focused exploration

Présentée et soutenue par

Matthias Sesboüé

Thèse soutenue publiquement le 5 septembre 2024 devant le jury composé de		
Cassia TROJAHN,	MCF HDR, Univ Toulouse II - IRIT	Rapportrice
Edlira VAKAJ,	Associate Professor, Birmingham City University	Rapportrice
Rossi SETCHI,	Professor, University of Cardiff	Examinatrice
Lina Fatima SOUALMIA,	PU, Université Rouen Normandie - LITIS	Examinatrice
Grégory ZACHAREWICZ,	PU, Ecole de Mines d'Ales - LG2IP	Examinateur
Cecilia ZANNI-MERK,	PU, INSA Rouen Normandie - LITIS	Directrice de thèse
Nicolas DELESTRE,	MCF, INSA Rouen Normandie - LITIS	Co encadrant de thèse
Jean-Philippe KOTOWICZ,	MCF, INSA Rouen Normandie - LITIS	Co-encadrant de thèse
Mickaël PICOT,	Chief Data Officer - TraceParts	Encadrant en entreprise

Thèse dirigée par Cecilia Zanni-Merk

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

MIIS litis ANRT ASSOCIATION NATIONALE RECHERCHE TECHNOLOGIE traceparts Product Content Everywhere

Abstract

These industrial research works explore Knowledge Graph-Based Systems (KGBS) for Information Retrieval (IR). They have been conducted in partnership with the company TraceParts. TraceParts is one of the world's leading Computer-Aided Design (CAD)-content platforms for Engineering, Industrial Equipment, and Machine Design. Hence, our use case considers a technical document corpus composed of Computer-Aided Design (CAD) models and their descriptions. Rather than leveraging the CAD models, we focus on their descriptive texts. Knowledge Graphs (KG) are ubiquitous in today's enterprise information systems and applications. Many academic research fields, such as Information Retrieval (IR), have adopted KGs. These digital knowledge artefacts aggregate heterogeneous data and represent knowledge in a machine-readable format. They are graphs intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities. The Architecture Engineering and Construction projects produce a wealth of technical documents. IR systems are critical to these industries to retrieve their complex, heterogeneous, specialised documents quickly. Healthcare is another similar domain with such a need. Though these industries manage documents with some textual content, such text and the metadata contain domain-specific concepts and vocabularies. Open KGs and the existing ontologies often describe concepts that are too high-level and need more fine-grained knowledge required by IR applications. Hence, companies' IR and knowledge management tools require domain-specific KGs built from scratch or extending existing ones.

Throughout our literature review, we first explore Knowledge Graphs (KG), ontologies, and how they relate to and derive our unifying KG definition. We consider ontologies one component of a KG and take a Semantic Web perspective, proposing illustrative candidate technologies from the World Wide Web Consortium Semantic Web standards. We also explore the theoretical and practical meaning of the term *semantics*. We then explore the literature on IR, focusing on KG-based IR. We break down this review section, first exploring the literature on IR using the term *knowledge graph* and then the one using the term *ontology*. We thereby point out some similarities and distinctions in the KG usages. Our contributions first introduce a KGBS architecture relating knowledge acquisition, modelling, and consumption arranged around the KG. We demonstrate that Semantic Web standards provide an approach for each KGBS component. To organise our work, we follow this system architecture; hence, each of our contributions addresses knowledge acquisition, modelling, and consumption, respectively. For our work, we do not have a pre-built KG or access to domain experts to construct it. Hence, we address knowledge acquisition by designing our Ontology Learning Applied Framework (OLAF) collaboratively with some of our research group members. We use OLAF to build pipelines to automatically learn an ontology from text. We implement our framework as an open-source Python library and build two ontologies to assess the OLAF's pertinence, usability, and modularity. We then focus on knowledge modelling, presenting our IR ontology and demonstrating its usage with an OWL reasoning-powered IR system. While most IR systems leverage reasoning in an offline process, our approach explores OWL reasoning at runtime. While demonstrating our IR ontology, we illustrate a Semantic Web-based implementation of our KG definition by pointing out each KG component in our IR ontology demonstration. Finally, we tackle the CAD model retrieval challenge our industrial partner TraceParts faces by implementing a KG-based approach at scale and using real-world data. We illustrate moving from an

existing text-based technical document retrieval system to a KG-based one. We leverage real-world TraceParts' CAD-content platform user interactions to evaluate our KG-based IR system proposal.

Keywords: *Knowledge Graph, Ontology, Information Retrieval, OWL, Semantic Web technologies*

Résumé

Ces travaux de recherche industrielle explorent les systèmes fondés sur les graphes de connaissances (KGBS) pour la Recherche d'Informations (RI). Ils ont été menés en partenariat avec l'entreprise TraceParts. TraceParts est l'une des principales plateformes de contenu de conception assistée par ordinateur (CAO) pour l'ingénierie, l'équipement industriel et la conception de machines. Ainsi, notre cas d'utilisation considère un corpus de documents techniques composé de modèles CAO et de leurs descriptions. Plutôt que d'exploiter les modèles CAO directement, nous nous concentrons sur leurs textes descriptifs.

Aujourd'hui, les graphes de connaissances (KG) deviennent omniprésents dans les systèmes d'information et les applications des entreprises. De nombreux domaines de recherche, tels que la RI, ont adopté les KG. Ces artefacts numériques agrègent des données hétérogènes et représentent les connaissances dans un format interprétable par nos ordinateurs. Ce sont des graphes destinés à accumuler et à transmettre les connaissances du monde réel, dont les nœuds représentent des entités d'intérêt et les arêtes les relations entre ces entités. Les projets d'ingénierie et de construction produisent une multitude de documents techniques. Les systèmes de RI sont essentiels pour les industries de ces domaines afin de retrouver efficacement leurs documents. Ces derniers sont complexes, hétérogènes et spécialisés. La santé est un autre domaine similaire avec un tel besoin. Bien que ces industries manipulent des documents avec un contenu textuel, ces textes et leurs métadonnées contiennent des concepts et du vocabulaire spécifiques à chaque domaine. Les KG ouverts et les ontologies existantes décrivent des concepts généraux et manquent des connaissances plus fines requises par les applications de RI. Par conséquent, les outils de RI et de gestion des connaissances nécessitent des KG spécifiques à chaque domaine, construits à partir de documents ou étendant des KG existants.

Nous explorons tout d'abord les KG, les ontologies et leur relation. Cette revue de littérature nous amène à proposer notre propre définition de KG. Nous considérons les ontologies comme une composante d'un KG et adoptons une perspective fondée sur le Web Sémantique en proposant des technologies issues des normes du Consortium World Wide Web. Nous explorons également la signification théorique et pratique du terme *sémantique* avant de poursuivre notre revue de la littérature avec la RI, en mettant l'accent sur la RI fondée sur les KG. Cette revue explore d'abord la littérature sur la RI utilisant le terme *graphe de connaissances* puis celle utilisant le terme *ontologie*. Nous mettons ainsi en avant des similitudes et distinctions dans les utilisations des KG. Nos contributions introduisent d'abord une architecture pour les systèmes fondés sur un graphe de connaissances. Cette architecture organise l'acquisition, la modélisation et la consommation des connaissances autour du KG. Nous démontrons que les standards du Web Sémantique fournissent une approche pour chaque composante de notre architecture. Nous utilisons cette dernière pour organiser la présentation de la suite de notre travail. Chacune de nos contributions aborde respectivement l'acquisition, la modélisation et la consommation des connaissances. Pour nos travaux, nous n'avons pas de KG préconstruit ou d'accès à des experts du domaine pour le construire. Par conséquent, nous abordons l'acquisition de connaissances en concevant notre approche d'apprentissage automatique d'ontologies (OLAF). Nous utilisons OLAF pour construire des chaînes de traitements et apprendre automatiquement des ontologies à partir de texte. Nous implémentons notre approche sous forme d'une bibliothèque Python open-source et construisons deux ontologies pour évaluer la pertinence, la facilité d'utilisation et la modularité de notre outil. Nous nous concentrons ensuite sur la modélisation des connaissances, en présen-

tant notre ontologie de RI dont nous démontrons l'utilisation avec un système de RI fondé sur du raisonnement déductif OWL. La plupart des systèmes de RI exploitent le raisonnement dans un processus hors ligne. Notre approche explore le raisonnement OWL en temps réel. La démonstration de notre ontologie de RI illustre par la même occasion une implémentation fondée sur le Web Sémantique de notre définition de KG en alignant chaque composant de notre définition avec un ensemble de triplets RDF. Enfin, nous abordons le défi de la recherche de modèles CAO auquel notre partenaire industriel TraceParts est confronté. Nous mettons en oeuvre à échelle industrielle une approche fondée sur les KG avec des données provenant de la plateforme de contenue CAO *wwww.traceparts.com*. Nous illustrons l'évolution d'un système existant de recherche de documents techniques fondé sur du texte vers un système fondée sur un KG. Pour évaluer notre système de recherche, nous exploitons les interactions des utilisateur·rice·s de la plateforme de contenu CAO.

Mots-clés: *Graphe de connaissances, Ontologie, Recherche d'information, OWL, Technologies du Web sémantique*

Acknowledgements

With these lines, I express my most genuine gratitude to all the great humans who contributed to supporting me in these last 3 years and the journey that led me to where I am today. I will do my best to cite everyone directly. Nevertheless, my gratitude goes to those I might have only indirectly mentioned.

First, I would like to thank all the reviewers for their time and attention to my work. It is truly an honour for me to count you among my reviewers. Your feedback is valuable to me and opens new perspectives. I especially appreciate your direct and genuine feedback, which helps me evolve in the best direction possible.

I want to thank my PhD advisors and the fantastic Litis team, who accompanied me on the twisted path of a PhD, always there to show me the way and help me overcome all the obstacles. Thank you, Cecilia, Nicolas and Jean-Phillippe. A special thanks to Cecilia for making the operational part of the PhD process smooth. Another special thanks to Nicolas' unconditional presence and support, even for the tasks that were the most challenging for you. I can not forget Sandra and Brigitte. Thank you so much for everything you do in the back to help us. You are genuinely an essential part of our PhD journeys. You have spared me much mental hassle in many situations. Thank you so much. Thank you also to the Litis administration members who always supported my small projects, which revolved around the Litis laboratory. I must address a special thank you to Marion, without whom this journey would have been much more lonely and hard. Thank you for all the great work we shared, and keep up your critical mind.

Thank you to Raynald and Gabriel for offering me this opportunity to learn at Traceparts. Thank you to all the great Traceparts team who welcomed me with open arms and always took the time to answer my questions. A particular thanks to Mickaël, who patiently followed my work during these 3 years. You found the perfect way to challenge my naive young perspectives with your experienced feedback without ever preventing me from freely exploring any direction. Your feedback, support, and weekly meetings have been invaluable to me. Thank you so much.

Because every achievement in anyone's life comes with a story and a long journey, I must thank everyone I met along my path. At the very beginning, thank you to my parents and family for always supporting me. I like to say that my "semantic" journey started in India during my first internship at Bosco Soft Technologies. Thank you to the Bosco Soft Technologies team, particularly Fr. Thaddeus Singarayan, who welcomed me and gave me this fantastic opportunity. Then comes M. Guidat. Thank you so much. I have never met you, but you have been a decisive part of my journey, introducing me to Mr McComb and Semantic Arts. A huge thanks to Mr McComb and the Semantic Arts team for opening your doors to me. You have provided me with such a great experience and propelled me into the next adventure at Wrangleworks. Thank you to Eric Hills for allowing me to join your project. You gave me my first startup experience, which I am now pursuing. My time in Austin would never have been the same without Dave. Thank you so much for supporting me since my childhood.

Finally, I would like to thank all my friends. In Particular, thank you, John, for all the weekends at sea liberating my mind. And most importantly, thank you, Evan, for being part of my life. You have been, are, and I am sure will always support me and be part of the equilibrium I need.

Thank you to all the ones I couldn't mention. Nobody builds anything alone. You all have been part of this (Brice, Caroline, Adam, Sean, Simon, Pierre, ...)

Contents

List of Figures

List of Tables

Part I

Synthèse de la thèse en français

Introduction

Les travaux de recherche présentés dans ce manuscrit ont été réalisés en collaboration avec l'entreprise TraceParts. TraceParts est l'une des principales plateformes de contenu CAO (Conception Assistée par Ordinateur) pour l'ingénierie, les équipements industriels et la conception de machines. Elle compte 5,3 millions de membres inscrits représentant 1,3 million d'entreprises qui recherchent activement des informations sur les produits et des données techniques depuis plus de 195 pays différents. Pour TraceParts, le moteur de recherche de leur plateforme de contenu CAO est un enjeu essentiel. La recherche fondée uniquement sur l'alignement de texte montre ses limites avec le contenu multilingue et des vocabulaires techniques spécifiques aux mulitples domaines couverts par les modèles CAO. Les graphes de connaissances (KG) sont des outils numériques puissants qui permettent de représenter de la connaissance dans un langage interprètable par les ordinateurs. En se fondant sur des concepts et leur relations, le processus de Recherche d'Information (RI) est enrichi avec des données structurées.

Les KG sont des structures de données de plus en plus communes dans l'industrie. Ces graphes unifient des données hétérogènes dans un format commun et interprètable par nos ordinateurs numériques. Ce sont des graphes dont l'intention est d'accumuler et de transmettre des connaissances sur le monde réel, dont les noeuds représentent des entités d'intérêt et les arrêtes des relations entre ces entités (traduit de [HBC+21]). Ils sont particulièrement utilisés dans des domaines comme la vision par ordinateur, le Traitement Automatique du Langage (TAL) ou la RI. Pourtant les concepts incarnés derrière ces structures de données et les systèmes associés ne sont pas nouveaux. Ils remontent au début de l'informatique, dans les années 70. Ils ont connu un regain d'intérêt en 2012 avec l'annonce du KG de Google et plus récemment avec les grands modèles de langages pour lesquels les KG sont étudiés comme source de connaissance structurée permettant de les guider. Parmis les applications qui utilisent des KG, la RI est un domaine récurrent. Il est étroitement lié au traitement automatique du langage naturel. Le domaine de la RI a pour but de retrouver au regard d'un besoin utilisateur·rice le contenu non structuré pertinent, dans de larges collections de documents (traduit de [MRS08])

Depuis les premiers ordinateurs, la quantité et la diversité des données stockées n'a cessé d'augmenter. Aujourd'hui la capacité à retrouver rapidement une information dans de grandes collections de documents de formats variés est devenu un enjeu stratégique pour de plus en plus d'industrie. Quelques soit le format des données stockées, audio, vidéo, image, text; elles manquent régulièrement d'information de contexte pour être exploitée dans les meilleures conditions. Pour des applications comme la RI, les données manquent des connaissances provenant du contexte d'exploitation nécessaire pour être efficacement utilisées. C'est en particulier le cas dans les domaines techniques qui nécessitent des connaissances approfondies et un vocabulaire spécifique. La santé, l'architecture et la construction sont des exemples de tels domaines. L'ingénierie en est un autre auquel nous nous intéressons en particulier dans ces travaux. Les modèles 3D et leurs documents techniques associés sont des resources essentielles pour des industries comme la mécanique, l'électronique ou encore la construction. Ces industries produisent de grandes quantités de documents techniques et modèles 3D, aussi appélés modèles CAO. Les moteurs de recherches de modèles CAO sont des outils essentiels à toute conception en ingénierie. Les requêtes acceptées par ces moteurs de recherches peuvent prendre deux formes principales. Un modèle CAO peut être proposé pour que le moteur retrouve des modèles similaires ou en relation. Mais plus régulièrement ce sont des requêtes textuelles qui sont utilisées. La langue naturelle est ambigüe et les domaines techniques utilisent des vocabulaires très spécifiques les rendant difficile à traiter informatiquement. À cela peut s'ajouter la nécessité de prendre en charge plusieurs langues.

Différentes structures de données ont été étudiées pour répondre à ces problèmes en formalisant les contenus non structurés comme le texte. Les KG font partis de ces structures de données. Mais avec le récent engouement pour ces graphes et les technologies qui leurs sont associés, le terme, *knowledge graph* en anglais est utilisé pour faire référence à des concepts diverses, étroitement liés mais bien différents. Cela mène à des définitions variées qui tendent à s'adapter à des

domaines ou des applications particulières. Il existe une confusion entre ce que sont les KG et les ontologies.

Une capacité mise en avant dans l'utilisation des KG et des ontologies est le raisonnement automatique, et en particulier le raisonnement déductif. Malgrès les nombreuses recherches utilisant des KG pour la RI, nous avons trouvé une utilisation limitée du raisonnement par déduction.

Nos travaux explorent les systèmes de RI fondés sur les KG. Notre cas d'étude s'intéresse plus particulièrement à un corpus composé de modèles CAO et de leurs documentations techniques associées. Une partie de la recherche scientifique sur ce sujet considère directement les modèles CAO. Dans notre étude, nous nous concentrons sur les contenus textuels associés à ces modèles comme par exemple les descriptions et labels. Dans un premier temps, notre état de l'art sur les KG nous amène à proposer notre définition inspirée de [HBC+21] qui considère les ontologies comme un élément des KG. Notre état de l'art sur les systèmes de recherches fondés sur les KG met en avant les utilisations différentes des termes *knowledge graph* et *ontology* dans la littérature scientifique.

Ce manuscrit présente nos travaux en partant de la théorie pour aller vers l'application industrielle. Nous présentons d'abord notre architecture pour les systèmes de RI fondés sur les KG. Cette architecture se veut générale et s'abstrait du cas particulier de la RI. Elle définit en détail les éléments d'un tel système articulé autour du KG et montre les intéractions entre l'acquisition, la modélisation et la consommation des connaissances. Nos autres contributions s'appuient respectivement sur ces trois activités. Nous nous intéressons à l'acquisition de connaissances en introduisant notre framework pour l'apprentissage automatique d'ontologies à partir de texte, OLAF (pour *Ontology Learning Applied Framework*). Nous avons implémenté OLAF sous forme d'une bibliothèque logiciel libre Python et construit deux ontologies pour démontrer son utilisation, son adaptabilité et sa pertinence en tant que boîte à outils. Nous considérons ensuite une approche de modélisation de connaissances pour la RI avec notre ontologie OWL pour la RI. Nous illustrons l'utilisation de cette ontologie en implémentant un système de recherche fondé sur le raisonnement déductif. Cette démonstration nous permet également d'illustrer notre définition de KG introduite en conclusion de notre état de l'art sur les KG. Enfin, nous proposons une approche de moteur de recherche fondé sur un KG. Avec notre partenaire industriel TraceParts, nous implémentons le passage d'un système de recherche fondé sur le texte à un système fondé sur les KG. Nous testons et comparons ces systèmes à l'échelle industrielle sur le corpus de modèles CAO de TraceParts. Nous utilisons un jeu de données tiré des recherches utilisateur·rice·s de la plateforme de contenu CAO www.traceparts.com pour évaluer et comparer notre approche.

En résumé, ces travaux présentent les contributions suivantes:

- Une définition de KG intégrant la notion d'ontologie.
- Une architecture pour les systèmes fondés sur un KG.
- Un framework pour l'apprentissage automatique d'ontologie et son implémentation sous forme d'une bibliothèque logicielle libre Python.
- Une ontologie OWL pour la RI et la démonstration de son utilisation pour une système de recherche fondé sur du raisonnement déductif.
- Une comparaison à échelle industrielle d'un système de recherche fondé sur le texte avec un système fondé sur les KG.

État de l'art

Pour nos travaux nous réalisons un état de l'art dans deux domaines de recherche : les KG et la RI. Pour ce dernier domaine, nous nous concentrons en particulier sur les approches de RI fondées sur un KG.

Graphes de connaissances

L'histoire des KG remonte au tout de début de l'informatique en étudiant les évolutions des idées et des technologies [GS21]. Les KG sont le résultat de l'évolution continuelle et de la convergence entre les technologies de traitement et de stockage de données et la représentation des connaissances dans l'informatique. Avec l'évolution des technologies matérielles et logicielles, nous stockons toujours plus de données de format divers. En revanche, il est toujours difficile de conserver le contexte et l'origine et donc une compréhension fine de ces données. Les ontologies, puis plus récemment les KG, sont des technologies qui permettent de représenter le contexte des données, c'est-à-dire les connaissances associées aux données. Un des grands acteurs de la représentation des connaissances est le "*World Wide Web Consortium*" (W3C) qui produit des standards pour le Web Sémantique (SW). Le SW est un web à destination des machines par opposition au web des documents, plus connu, et qui est à destination des humains. Ce dernier est le Web que nous consultons lorsque nous faisons une recherche sur notre moteur de recherche favori. Les machines ont besoin de contexte pour pouvoir manipuler les données. Les standards du W3C tel que RDF [CWL14], RDFS [BG14] et OWL [HKP$^+$12], permettent de représenter les connaissances dans des langages interprétables par nos ordinateurs. Cela permet ensuite de développer des programmes intelligents capables de raisonnements déductifs et explicables. La sémantique, ou l'intelligence dans "*Intelligence Artificielle*" (IA), est incarnée par la capacité des machine à "raisonner" pour déduire des faits par elle même. Aujourd'hui la notion de sémantique en informatique peut être interprétée de deux grandes façons : les raisonnements inductifs se distinguent des raisonnements déductifs. Le raisonnement inductif s'appuie sur les données pour déterminer des schémas. Les réseaux de neurones profonds sont un exemple de système pour du raisonnement inductif. Le raisonnement déductif s'appuie sur des faits explicitement formulés dans un langage machine pour en déduire de nouveaux. Les systèmes experts avec leurs moteurs de règles sont un exemple de système fondé sur du raisonnement déductif.

Il existe plusieurs définitions différentes de KG et d'ontologie [Kee20]. Cela montre une confusion autour de ces notions. Dans notre étude nous adaptons la définition donnée par les auteurs de [HBC$^+$21] qui explore les KG et leurs applications. Les auteurs font d'abord la distinction entre la connaissance que l'on veut représenter, le modèle de graphe et le schéma de ce graphe. La connaissance est par définition tout ce que l'on sait et que l'on souhaite modéliser. Le modèle de graphe est la structure mathématique que l'on choisit pour modéliser la connaissance. Il existe deux grands modèles : le modèle de graphe orienté et labellisé sur les arrêtes (DELG) représenté comme un ensemble de triplet (*sujet, prédicat, objet*) et le modèle de graphe attribué (LPG) dans lequel les noeuds et les arrêtes peuvent avoir n attribus sous la forme d'une liste de clé-valeur. Dans nos travaux, nous utilisons le modèle DELG et son implémentation avec le *Resource Description Framework* (RDF) qui est le standard du W3C à la base du Web Sémantique et de ses technologies. Quant au schéma du graphe, il définit la structure de celui-ci. Le schéma peut définir soit la structure des données, soit la sémantique. Dans le premier type de schéma, c'est la structure des données qui est validée. Un noeud représentant une personne doit avoir un nom, un prénom et est lié à une autre personne par une relation *ami*. Dans le deuxième, le schéma sémantique définit ce que cela signifie d'être un ami, par exemple en définissant la relation *ami* comme symétrique: si Henri est un ami de Martine, alors Martine est aussi l'ami de Henri. Dans ce second type de schéma, on distingue les interprétations en monde fermé et en monde ouvert. Dans l'interprétation en monde fermé, on considère tout ce qui n'est pas défini comme faux. Dans celle en monde ouvert, on ne peut considérer comme faux uniquement ce qui est explicitement défini comme faux. Si rien n'est définit, on ne sait simplement pas et on ne peut donc pas s'appuyer directement sur ce fait pour tirer des conclusions par raisonnement déductif.

Les auteurs de [HBC$^+$21] introduisent également la distinction entre le graphe de données et le graphe de domaine. Le graphe de domaine définit la connaissance et donc la sémantique qui régit l'interprétation du graphe de données. Le graphe des données est l'ensemble des points de données liés entre eux. Ces points de données et leurs liens peuvent avoir leur entité et relation correspondantes dans le graphe de domaine qui définit alors leur sémantique. Dans le graphe de

domaine on parle d'entités et de relations et dans le graphe de données de noeuds et d'arrêtes. Les graphes de domaine et de données sont liés par un alignement défini (le "*mapping*"). Lorsque une ontologie est définie avec le langage OWL, le RDF est utilisé pour la partager. Le RDF permet de représenter une ontologie sous forme de triplets qui définissent un graphe suivant le modèle DELG. Ce graphe correspond au graphe de domaine d'un KG. Les ontologies sont un élément du graphe de domaine et donc d'un KG. Nous réprésentons notre définition de KG dans la figure 1 sur laquelle nous reportons les standards du W3C qui peuvent être utilisés dans l'implémentation de chaque élément.

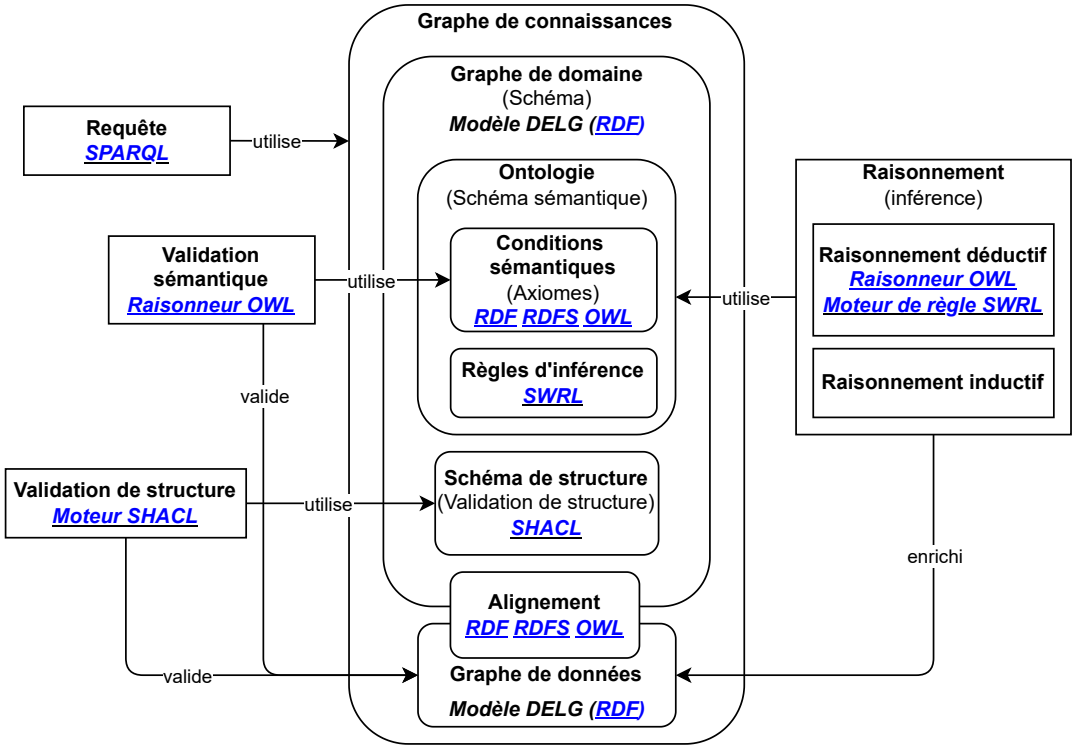


Figure 1: Définition d'un graphe de connaissances avec des propositions de technologies du Web Sémantique pouvant être associées à chaque composant.

Recherche d'information

La RI est une tâche qui est réalisée depuis les premiers documents écrits. Avant les premiers ordinateurs, des inventeurs ont imaginé des machines mécaniques pour retrouver des documents dans des corpus. Avec l'arrivée et la démocratisation des ordinateurs, la recherche s'est intéressée à la manière d'optimiser les moteurs de recherche à l'aide de ces derniers. Les premières approches ont cherché à numériser les méthodes de classification utilisées dans les bibliothèques. Les approches qui ont suivi se sont intéressées à la structure de la langue naturelle. Enfin, l'émergence du Web a conduit à de nouvelles formes de moteurs de recherche fondés sur les données utilisateur·rice·s [SC12, HP23].

Dans un système de RI, on peut distinguer différents objectifs, différents composants et différentes tâches. Parmis les différents moteurs de recherche, on distingue ceux qui ont pour objectif de retrouver des documents entiers, des passages de documents, ou des entités comme des personnes ou des objets. Un système de RI est composé d'un besoin d'information qui est exprimé sous forme d'une requête. Cette requête est soumise au système qui la confronte au corpus de documents. Ce corpus est indexé, c'est-à-dire stocké dans une structure de données qui facilite l'accès aux documents et les calculs de pertinence des documents par rapport à une requête.

Afin de retrouver les documents pertinents au regard d'une requête, et ordonner ces documents sélectionnés pour que le plus pertinent se retrouve au début de la liste, le système de RI peut s'appuyer sur différents éléments. Les plus courants sont un modèle d'extraction de documents, un modèle d'ordonnancement de documents, des resources externes comme un KG, ou encore des resources construites à partir du corpus comme des espaces latents ou un modèle de langage. Un espace latent est un espace mathématique dans lequel il est plus facile de calculer le niveau de pertinence entre un document et une requête [LJLH19]. Un modèle de langage est un modèle statistique construit à partir des contenus textuels du corpus. Un exemple de modèle de langage est un modèle qui cherche à estimer la probabilité qu'un mot apparaisse en même temps qu'un autre mot [JM23].

Le domaine de la RI au sens informatique est connu depuis les premiers ordinateurs. Il existe donc une multitude de modèles d'extraction de documents. On peut les séparer en deux grandes catégories, les modèles dit "*Ad Hoc*" qui utilisent des modèles probabilistes, des espaces latents ou des modèles de langage, et les modèles fondés sur les retours utilisateur·rice·s ou *feedback*. Ces retours peuvent être direct ou indirect. Parmis les modèles Ad Hoc, on peut mentionner BM25 [RZ09] encore très utilisé aujourd'hui. Enfin, dans un système de RI on peut distinguer plusieurs tâches. Du côté du traitement de la requête, on peut procéder à sa transformation pour faciliter l'extraction de document et leur ordonnancement qui peuvent être des tâches disctintes. Du côté du corpus, les documents sont indexés en se fondant sur leur contenu textuel ou même sémantique, c'est-à-dire non plus les mots mais les concepts qu'ils représentent.

Nous avons étudié la littérature sur les systèmes de RI fondés sur les KG en considérant des travaux présentés en utilisant le terme KG (*knowledge graph*) d'une part et ceux présentés avec le terme ontologie (*ontology*) d'autre part. Il en ressort que les travaux utilisant le terme KG tendent à mettre à profit la structure de données graphe, c'est-à-dire le graphe de données, en dérivant les vecteurs représentatifs par exemple. Les travaux utilisant le terme ontologie tendent quant à eux à mettre en avant un vocabulaire structuré avec principalement des relations hiérarchiques. Ces travaux s'intéressent au graphe de domaine de notre définition de KG.

Contributions

Dans nos travaux de recherche nous aboutissons à quatre contributions principales. Nous introduisons d'abord notre architecture pour les systèmes fondés sur un KG. Cette dernière nous permet de mettre en contexte nos autres contributions. Nous nous concentrons ensuite sur l'acquisition de connaissances en présentant notre framework pour l'apprentissage automatique d'ontologies et son implémentation. Nous illustrons ensuite notre ontologie OWL pour la RI dans un exemple de système de recherche propulsé par le raisonnement déductif OWL. Enfin nous discutons avec une analyse d'un système de RI à échelle industrielle, les différences entre un système de RI fondé sur le texte à un système fondé sur un KG.

Architecture pour les systèmes fondés sur un graphe de connaissances

Notre première contribution s'intéresse aux systèmes fondés sur un KG d'un point de vue théorique et s'abstrait du cas d'usage de nos travaux qu'est la RI. Notre objectif ici est de bien comprendre le fonctionnement général d'un tel système avant de plonger dans des applications et implémentations particulières. Cette architecture nous servira ensuite de fil conducteur pour la présentation de nos contributions.

La figure 2 présente une vue d'ensemble détaillée de notre architecture, introduisant chaque composant, leurs imbrications, leurs interactions et des technologies du Web Sémantique qui leurs sont associées. Ces composants sont constitués d'activités (boîtes rectangulaires) et des sources de connaissances ou conteneurs de données (boîtes arrondies). Au plus haut niveau, les sources de connaissances sont utilisées par les processus d'acquisition de connaissances. Les connaissances extraites par ces processus sont ensuite modélisées en fonction des choix de techolo-

gies pour former le KG. Les connaissances sont consommées par des applications : dans nos travaux un moteur de recherche. Enfin, pour assurer la qualité et la maintenance du KG, des processus de validation de connaissances sont nécessaires. Nous mentionnons ces processus pour la complétude de notre architecture bien que nous ne les explorions pas dans nos travaux.

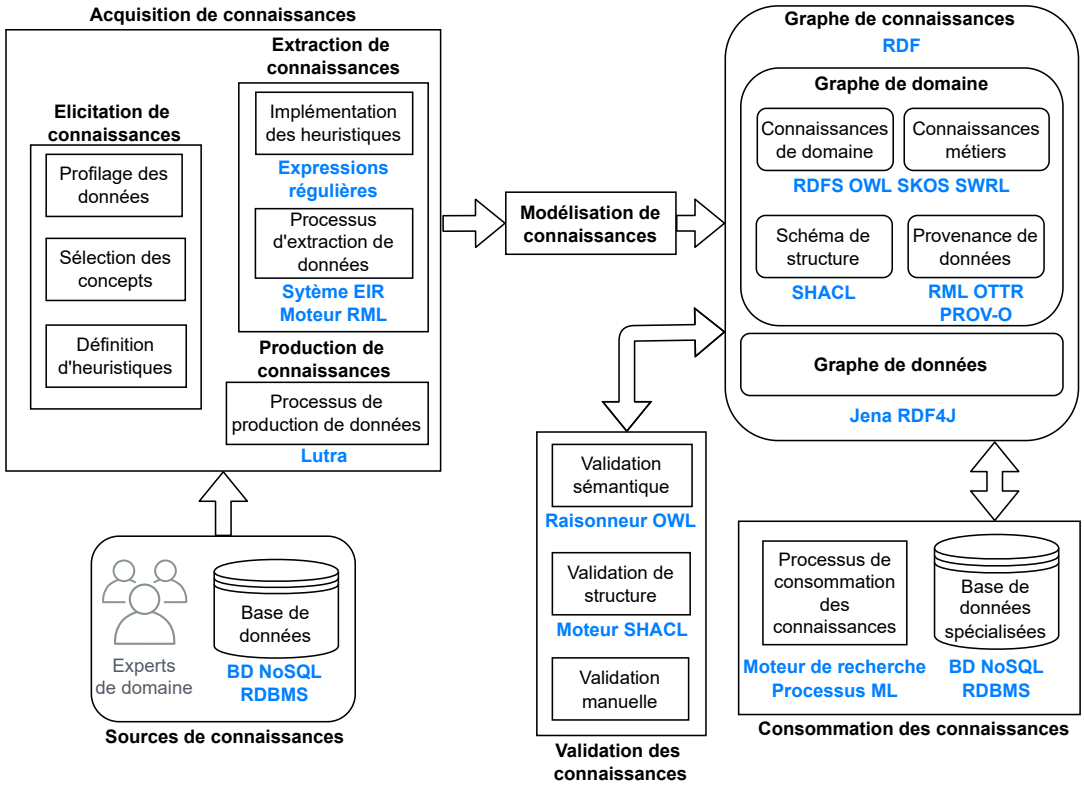


Figure 2: Architecture pour les sytèmes fondés sur un graphe de connaissances. Les composants sont présentés avec des technologies du Web Sémantique associées.

Les boîtes rectangulaires représentent des processus et les boîtes arrondies des conteneurs de données. L'imbrication des boîtes représente des niveaux de granularité. Les flèches montrent des flux de données. Des technologies pour l'implémentation des éléments sont mentionnées en gras et bleu. (SMEs: Subject Matter Experts (Expert de domaine); HITL: Human In The Loop (l'humain dans la boucle))

Les sources de connaissances peuvent être diverses. La source idéale serait les experts de domaine, mais ces experts sont rarement disponibles en pratique et leur temps coûte cher. C'est pourquoi les sources de connaissances les plus courantes sont des corpus de documents ou des schémas de bases de données. Ces sources contiennent régulièrement du contenu textuel que la machine doit interpréter.

Dans nos travaux, nous distingons trois grandes phases dans le processus d'acquisition de connaissances. L'élicitation de connaissances est principalement un processus manuel qui a pour but de définir les contours et les éléments de la connaissance nécessaires à un projet. Dans cette phase, on peut distinguer le profilage des données, c'est-à-dire une cartographie des données que l'on va pouvoir utiliser pour extraire les connaissances et celles que l'on cherche à contextualiser avec cette connaissance. Ce profilage sert également à faire une première sélection des concepts que le KG devra définir. On définit ainsi le périmètre de l'activité d'acquisition de connaissances du KG. Cette exploration nous permet de définir de premières heuristiques pour l'activité suivante d'extraction des connaissances. Bien que cette phase d'élicitation de connaissances soit fondée sur l'humain, elle peut être aidée et optimisée par des outils informatiques.

L'extraction de connaissances implémente les résultats de la phase d'élicitation. Les outils informatique et/ou un paradigme de programmation pour implémenter et exécuter les heuris-

tiques, comme par exemple des expressions régulières, sont choisis. Le profilage des données permet également d'implémenter des processus d'extraction de concepts et de données, par exemple avec des méthodes de TAL comme l'extraction d'entités d'intérêt ou un moteur RML (pour "*Rule Mapping Language*" [DVSC+14]) qui exécute des alignements définis dans le langage RML, entre des schémas de base de données et des concepts et relations.

Enfin l'extraction se distingue de la production de connaissances. Lors de l'extaction, un paradigme de programmation est utilisé, par exemple le paradigme de programmation orienté objet. Mais le langage pour construire le KG n'est pas encore déterminé. Les processus de production de connaissances font le pont entre le format résultant de l'extraction de connaissances et celui du KG. Dans nos travaux nous utilisons un paradigme de programmation objet. Les résultats de l'extraction de connaissances sont donc des classes et leurs propriétés. Nous choisissons de représenter notre KG en RDF avec les langages RDFS et OWL. Le processus de production de connaissances produit donc des triplets RDF qui définissent des classes et des propriétés OWL et RDFS. Un langage intéressant pour cette tâche est OTTR ("*Reasonable Ontology Template*") [SLKF18] et le programme Lutra[1] d'exécution de ces templates de construction de triplets RDF.

Le processus de modélisation de connaissances est le travail principal de l'ingénieur·e de connaissances. Avant de pouvoir automatiser, il faut définir la structure du KG, et donc définir la forme la plus adéquate pour modéliser la connaissance nécessaire à l'application visée. C'est par exemple à cette étape que l'on fait des choix comme ce qui sera une instance ou une classe en OWL. Naturellement, les processus de modélisation et de production de connaissances sont indissociables.

Dans notre état de l'art sur les graphes de connaissances, nous distinguons le graphe de données du graphe de domaine. Dans notre architecture, nous approfondissons le graphe de domaine pour faire une séparation théorique des connaissances que nous modélisons, bien que d'un point de vue implémentation cette séparation n'ait pas forcément de sens. Par exemple en construisant notre graphe de connaissances avec des triplets RDF, tout sera triplet RDF. Cette distinction entre plusieurs sortes de connaissances nous permet une meilleure compréhension théorique de ce qui est modélisé. Dans la littérature sur les ontologies en particulier, les connaissances modélisées sont séparées en connaissances fondationnelles, de domaine, de tâche et d'application. Ces travaux théoriques aident à la compréhension mais les limites entre chaque type de connaissances restent flous en pratique. Dans nos travaux nous choisissons de séparer les connaissances entre celles de domaine, celles métier, celles définissant la structure du KG et celles se concentrant sur la provenance des données et connaissances. Les connaissances de domaine sont des connaissances générales comme des unités de mesures ou les concepts à la base de la physique des matériaux par exemple. Elles correspondent aux ontologies fondationnelles dans la littérature scientifique. Les connaissances métiers sont celles qui viennent étendre les connaissances de domaine pour spécifier ces dernières et les ajuster au métier de(s) application(s) visée(s). Ces deux catégories de connaissances sont typiquement modélisées en RDF avec les langages du SW. Les connaissances de domaines et métiers servent à la validation sémantique. Le schéma structurel spécique à l'implémentation du KG sert à valider la structure du graphe de données. Un langage du SW destiné à cette tâche est SHACL [KK17]. Enfin, pour assurer la qualité et la fiabilité du KG, les informations sur la provenance des données et connaissances sont essentielles. Les standards RML et OTTR sont des exemples de langage qui servent à tracer l'origine des connaissances. PROV-O [LSM+13] est un autre standard du W3C spécifiquement destiné à modéliser la provenance des données. La validation du KG n'est pas approfondie dans nos travaux. On se limite à faire la distinction entre validation structurelle et sémantique, qui ont des langages du SW spécifiques et donc leurs moteurs pour les interpréter. On ajoute que ces tâches de validation, en particulier des connaissances, impliquent régulièrement des experts de domaine et donc une partie de validation manuelle.

Enfin les connaissances modélisées dans le KG sont destinées à être consommées par des applications. Il est essentiel de bien distinguer les systèmes utilisés pour stocker et rendre disponible

[1] https://www.ottr.xyz/#Lutra (Consulté le Thursday 3rd October, 2024)

le KG des systèmes destinés à des applications spécifiques. Ces derniers peuvent extraire une partie du KG pour le stocker dans un autre format plus optimisé pour le besoin de consommation de connaissances d'une application spécifique. C'est un cas très courant qui est nécessaire, mais il est alors essentiel que les nouvelles connaissances potentiellement générées par l'application soient redirigées vers le KG. En d'autres termes, le KG doit rester la source de connaissances de référence.

Nous utilisons notre architecture pour introduire la suite de nos travaux. Ainsi les prochaines contributions s'intéressent respectivement à l'acquisition, la modélisation et la consommation de connaissances.

Framework pour l'apprentissage automatique d'ontologies

Dans nos travaux nous n'avons pas de KG déjà défini, ni accès à des experts de domaine dont le temps est limité et coûteux. Nous nous tournons donc vers les documents textuels dont nous disposons pour extraire les connaissances utiles à notre moteur de recherche. Cette partie de nos travaux a été réalisée en collaboration avec une collègue de l'équipe de recherche dont les travaux portent sur des agents conversationels fondés sur des KG. Nous avions donc tous deux besoin de construire un KG. Pour répondre à ce besoin nous avons exploré les méthodes d'apprentissage automatique d'ontologies à partir de texte (OL) ("Ontology Learning" en anglais).

Dans la littérature l'OL est présentée avec une série de tâches : extraction de termes, puis de synonymes, puis de concepts, hiérarchisation de concepts, extraction, puis hiérarchisation de relations, extraction de schéma d'axiomes puis d'axiomes généraux. Les méthodes de la littérature scientifique s'intéressent à des applications particulières et implémentent les tâches ou combinaisons de tâches qui correspondent aux besoins des cas d'usages considérés pour les démonstrations. Afin de répondre à nos besoins couvrant un plus large spectre, nous nous sommes orientés vers la conception d'un framework d'OL. Ce framework a pour objectif de définir les différents éléments d'une chaîne de traitements pour l'OL et les intéractions entre ces éléments. Ainsi, nous souhaitons développer des méthodes se concentrant sur des aspects particuliers de l'OL tout en permettant la combinaison de ces méthodes pour contruire et évaluer les chaînes de traitements répondant au mieux à nos cas d'usage.

Nos travaux sur l'OL aboutissent à un framework et son implémentation sous la forme d'une bibliothèque logiciel Python open source OLAF pour *Ontology Leanrning Applied Framework*[2]. Cette implémentation fournit les blocs de base nécessaires pour combiner plusieurs méthodes dans une chaîne de traitements d'OL. Nous avons réimplémenté séparément des méthodes combinées dans les approches de la littérature scientifique. Cela nous a permi de réaliser nos expériences.

Dans l'évaluation de notre framework nous cherchons tout d'abord à valider sa pertinence en démontrant son fonctionnement et sa modularité. Nous construisons deux ontologies, une à partir de descriptions de produits électroniques et l'autre à partir de descriptions de pizzas générées à l'aide d'un grand modèle de langage (LLM). Ce second exemple répond à la difficulté d'évaluer les ontologies apprises par nos chaînes de traitements. Nous n'avons pas trouvé de corpus de textes associés à une ontologie de référence. Pour notre première expérience, nous avons sélectionné les corpus et appris une ontologie que nous avons ensuite évaluée manuellement. Pour notre seconde expérience nous avons souhaité utiliser une ontologie de référence. Nous avons donc généré à l'aide d'un LLM des textes descriptifs dans le but d'apprendre l'ontologie. Nos expériences valident la pertinence de OLAF, son fonctionnement et sa modularité. Elles montrent également des résultats encourageants sans pour autant produire d'ontologies directement exploitables.

Modélisation de connaissances pour la recherche d'information

Dans notre revue de la littérature sur les systèmes de RI fondés sur les KG, nous n'avons trouvé aucune ontologie construite pour faire fonctionner directement un système de RI. Les approches

[2]https://wikit-ai.github.io/olaf/ (Consulté le Thursday 3rd October, 2024)

fondées sur du raisonnement OWL n'utilisent pas le raisonnement en temps réel. Seuls les faits inférés par un raisonnement réalisé en amont sont utilisés. Ce chapitre introduit notre ontologie de RI dans un système de RI propulsé par le raisonnement déductif fondé sur les logiques de description. Notre ontologie de RI ainsi que les démonstrations sont accessibles sur notre dépôt GitHub[3].

McComb D. définit un aspect de la conception de l'ontologie Gist[4] qu'il nomme *Cbox*. Gist est une ontologie minimale OWL sur le domaine de l'entreprise. Dans la modélisation de connaissances avec un langage fondé sur les logiques des descriptions, les termes *Tbox*, *Abox*, *Rbox* sont couramment utilisés pour séparer différents types d'axiomes. Ces termes regroupent respectivement les axiomes définissant les concepts d'une ontologie, les faits et les relations. La *Cbox* définie par McComb regroupe les concepts de catégories et leurs instances. Du point de vue de la gestion des connaissances en entreprise, il s'agit d'une question d'échelle selon Dave McComb. Une ontologie devrait contenir au maximum quelques centaines de concepts de base. De plus, elle devrait être gérée par une équipe d'ontologistes. Les catégories sont de l'ordre de quelques milliers. Elles sont gérées par des experts du domaine responsables de la maintenance de leur ensemble de catégories ou de classifications. Enfin, les données, c'est-à-dire les instances organisées par catégories, sont de l'ordre de plusieurs millions. Une telle échelle nécessite des processus automatisés.

Le cas d'usage à partir duquel nous avons construit l'ontologie de RI a pour objectif de faire fonctionner la navigation dans un ensemble de classifications. La recherche textuelle initiale de l'utilisateur·rice permet au système de proposer un ensemble de catégories. Ces catégories servent ensuite de points d'entrée au processus de navigation dans les classifications. Chaque catégorie est liée à ses sous-catégories par une relation hiérarchique et les catégories catégorisent les documents. L'ontologie RI permet donc de déterminer les documents correspondants à une recherche utilisateur·rice exprimée comme un ensemble de catégories. Ces documents sont appelés documents candidats, et les catégories de la recherche utilisateur·rice sont les catégories sélectionnées. La sélection par l'utilisateur·rice du premier ensemble de catégories initie un échange entre le système et l'utilisateur·rice. Une fois qu'un utilisateur·rice sélectionne ses catégories, l'ontologie doit permettre au système de connaître les catégories disponibles pour affiner sa recherche ainsi que les documents candidats. Nous présentons notre ontologie de RI en définissant les questions de compétence [EFK19] auxquelles elle doit pouvoir répondre :

CQ1 Quelles sont les catégories de la recherche en cours ?

CQ2 Quels sont les documents pertinents pour une recherche ?

CQ3 Quelles catégories permettent de raffiner la recherche ?

Dans cette partie de nos travaux, nous considérons l'ontologie RI et laissons de côté le prétraitement de la recherche utilisateur·rice permettant d'aboutir à un ensemble de catégories. La recherche utilisateur·rice correspond ici à des triplets définissant une instance d'une classe que nous nommons "*Search context*". CQ1 garantit que les catégories incluses dans une recherche sont considérées comme sélectionnées. CQ2 découle directement de notre cas d'usage. À tout moment, l'ontologie doit pouvoir être questionnée pour obtenir les documents pertinents au regard d'une recherche. D'un point de vue pratique, CQ2 est l'une des deux principales requêtes impliquées dans le va-et-vient entre le système de RI et l'ontologie. CQ3 permet une recherche par navigation dans les classifications.

L'ontologie de RI définit 7 classes, 6 relations et un total de 34 axiomes. Les 7 classes permettent de représenter :

- un document,

[3] https://github.com/msesboue/ir-ontology (Consulté le Thursday 3rd October, 2024)

[4] https://www.semanticarts.com/gist/ (Consulté le Thursday 3rd October, 2024)

- un document candidat,
- une catégorie,
- une catégorie sélectionnée,
- une catégorie disponible,
- une recherche,
- une recherche en cours.

Les 6 relations permettent de représenter :

- la relation de catégorisation d'un document et son inverse,
- la relation entre une recherche et ses categories,
- la relation entre une catégorie et celles qu'elle rend disponibles une fois sélectionnée,
- les relations de hiérarchie directes et indirectes.

Les classes sont définies comme sous-classes d'une restriction universelle sur une relation et sa classe objet. Cette structure utilisée dans plusieurs définitions de classe permet les raisonnements attendus.

Notre démonstration pas à pas d'une utilisation de l'ontologie RI montre le fonctionnement d'un système de RI fondée sur celle-ci, ainsi qu'un exemple des ensembles de triplets RDF associés à notre définition de KG. Ces recherches nécessitent d'être approfondies pour évaluer la pertinence d'un tel système de RI dans un contexte industriel. L'ontologie RI peut aussi être étendue pour inclure la distinction entre une recherche pour laquelle il n'existe pas de document pertinent connu, d'une recherche pour laquelle il ne peut pas exister de document pertinent. Ce dernier type de recherche correspond à une recherche incohérente comme une pizza végétarienne avec de la viande.

Expérimentations industrielles

La première version de la plateforme de contenu CAO de TraceParts a été lancée en juillet 2001. Il s'agit d'une plateforme web unique offrant des ressources techniques gratuites pour les concepteurs et les ingénieurs dans 25 langues. Parmi ces données figurent plus de 120 millions de fichiers 3D provenant de plus de 1800 catalogues, permettant aux utilisateur·rice·s de logiciels CAO de gagner un temps précieux. Le service est financé par les fabricants et distributeurs de composants qui répertorient leurs produits et par la publicité. Aujourd'hui, le portail compte plus de 5,3 millions d'abonnés et génère plus de 7 millions de fichiers CAO par mois. Les types de données traitées par la plateforme sont très hétérogènes. De nombreuses pièces sont personnalisables, avec différentes structures d'information et types de valeurs, ce qui entraîne un volume de données important et en constante évolution.

Les utilisateur·rice·s peuvent rechercher des pièces de fabricants dans tous les catalogues du site. Une pièce est identifiée par son numéro de pièce fabricant et peut être contenue dans plusieurs classifications. Afin de regrouper et de classer efficacement les pièces des fabricants, TraceParts a créé sa propre classification sous la forme d'un catalogue supplémentaire. Pour leur recherche, les utilisateur·rice·s ont accès un champ de recherche textuelle, la classification de TraceParts ou une liste de catalogues organisés par ordre alphabétique. La liste principale des résultats contient une entrée distincte pour chaque pièce, même si elles correspondent à des variantes d'une même famille de pièces. Ces familles rassemblent toutes les configurations possibles pour une même pièce identifiées par des numéros distincts. Enfin, l'application web permet des recherches multilingues.

Comme mentionné précédemment, le corpus est composé de contenu CAO. Ce contenu englobe à la fois les modèles CAO et leurs données descriptives. Ces dernières se présentent sous diverses formes. Quelques exemples incluent les fiches techniques des modèles CAO, les dessins 2D et les descriptions textuelles de différentes longueurs. De nombreux travaux se concentrent sur les modèles CAO avec des cas d'usage spécifiques, tels que la recherche de conceptions similaires ou équivalentes [QGY+16], la recherche d'assemblages ou la recherche de composants en fonction de la compatibilité et des fonctions mécaniques [LPMG19]. D'autres travaux considèrent les métadonnées associées aux modèles CAO, en particulier les métadonnées textuelles [LRR08]. Nos travaux se concentrent sur ce dernier cas.

Dans les expériences suivantes, nous considérerons deux corpus différents. Le premier corpus considère chaque pièce comme un document. Chaque configuration possible de modèle CAO est un document. C'est le corpus actuel sur *www.traceparts.com*. Le deuxième corpus considère un document par famille de pièces, c'est-à-dire un document par composant comprenant toutes ses configurations possibles. Cela implique naturellement un corpus beaucoup plus petit. Début 2024, le corpus des familles de pièces contenait 1110738 documents, et le corpus des numéros de pièces était plus de cent fois plus vaste, avec 127802485 documents. Nous considérons les mêmes champs de texte indépendamment du choix du corpus. Certains sont courts et peuvent être considérés comme des tags, par exemple une norme telle que "*DIN 912*", un nom de catégorie, de constructeur ou de catalogue. D'autres sont plus longs comme des descriptions. Cependant, la majorité des champs de texte sont courts avec au maximum 2 phrases et utilise du langage technique. La majorité des descriptions ne constituent pas des phrases grammaticalement correctes, comme par exemple "*P01-P02-P04-P06-P08*". Pour les deux corpus, tout le contenu des documents est accessible sur *www.traceparts.com*. Voici un résumé quantitatif des corpus :

- 127802485 pièces (premier corpus);
- 1110738 familles de pièces (deuxième corpus);
- 25 langues;
- Les textes font en moyenne 50 caractères;
- Les textes font en moyenne 7 mots;
- 2556 constructeurs;
- 1913 catalogues;
- 208466 catégories.

Les utilisateur·rice·s de la plateforme effectuent des recherches depuis le monde entier. Leurs recherches sont plus courtes que les champs de texte des documents et peuvent être rédigées dans n'importe quelle langue. En moyenne, un texte de recherche utilisateur·rice contient 13 caractères répartis en 2 mots. Aucun des textes de recherche utilisateur·rice n'est grammaticalement correct. Ce sont tous des mots-clés spécifiques au domaine, des notations, des identifiants et des acronymes.

Notre objectif est de passer d'une recherche fondée sur le texte à une recherche fondée sur les concepts. Dans la littérature scientifique, ce changement de paradigme de recherche est appelé recherche sémantique ou indexation sémantique. Par rapport à l'approche fondée sur le texte, notre approche fondée sur les concepts ajoute une étape dans le processus de recherche de documents. Le texte de la recherche utilisateur·rice est d'abord utilisé pour extraire des concepts qui sont enrichis avant de rechercher des documents. Dans cette approche, plusieurs recherches comme par exemple une recherche en français et la même en anglais, peuvent aboutir à un même ensemble de documents pertinents. La recherche fondée sur le texte aboutie quant à elle à une très grande diversité de résultats et les recherches dans une langue n'aboutiront pas à des documents dans une autre langue.

Nous construisons nos expériences de manière itérative. Tout d'abord, nous implémentons notre référence avec une réplique du moteur de recherche actuel fondé sur le texte. Nous définissons ensuite quelques concepts et implémentons un système de recherche de documents fondé sur les concepts. Dans ce système, nous utilisons les concepts comme un vocabulaire contrôlé et n'exploitons pas encore les relations entre les concepts. Nous utilisons donc les relations entre concepts dans un troisième système pour enrichir la recherche fondée sur les concepts. Nous appelons ce dernier système le système de recherche de documents fondé sur le KG. Enfin, nous expérimentons l'inclusion de l'historique des recherches utilisateur·rice·s de la plateforme comme une étape d'ordonnancement des résultats. Cette historique constitue une source de connaissances implicites.

L'évaluation de nos expériences nécessitent un ensemble d'exemples de recherches utilisateur·rice·s dont nous connaissons les résultats attendus. L'objectif de TraceParts est de réduire le nombre moyen d'itérations qu'un utilisateur·rice effectue avant de télécharger un modèle CAO. Nous considérons donc les modèles CAO téléchargés après une recherche textuelle comme un résultat de recherche positif et construisons un jeux de données à partir des recherches textuelles effectuées sur la plateforme. Enfin pour comparer nos différentes approches nous calculons les scores suivants pour les k égal 5, 25, 50, 100, et 350 premiers résultats dans l'ordre de leur pertinence :

- Le rang moyen du premier résultat positif dans les k premiers documents (*Mean Reciprocal Rank at k*)
- Une précision moyenne glissante sur les k premiers documents (*Mean Average Precision at k*)
- Une moyenne sur l'ensemble des exemples, de la présence ou non d'un résultat positif dans les k premiers documents (*Binary Mean at k*).

Les résultats des expériences montrent que le système de recherche actuel fondé sur le texte a un bon rappel mais une précision très faible. Dans la pratique, quand une famille de pièces est jugée pertinente, beaucoup de pièces de cette même famille sont retournées avec une pertinence élevée. Cela implique beaucoup de documents sélectionnée parmis lesquels se trouve des documents positifs mais ces derniers sont alors mal ordonnés. On observe aussi très peu de diversité dans les résultats de recherche.

Quand on analyse le corpus des familles de pièces, les résultats de recherche sont naturellement meilleur, d'une part parce qu'il y a beaucoup moins de document dans le corpus, et d'autre part parce qu'il y a plus de diversité dans les documents sélectionnés. Le passage à la recherche fondée sur les concepts montre de meilleurs résultats qui peuvent s'expliquer par la réduction de la taille du vocabulaire utilisé pour aligner les documents avec la recherche utilisateur. En revanche, le vocabulaire initial correspondant aux mots du langage naturel est toujours nécessaire pour aligner la recherche utilisateur·rice avec les concepts avant de pouvoir utiliser ceux-ci dans la recherche de documents pertinents.

La recherche fondée sur le graphe de connaissances enrichie les concepts trouvés dans la recherche utilisateur·rice en parcourant le graphe. Cela abouti à plus de documents pertinents sélectionnés mais qui sont dans un premier temps mal ordonnés. On obtient donc une moins bonne précision. En revanche, l'ajout de connaissances implicites provenant de l'historique des recherches utilisateur·rice·s vient améliorer l'ordonnancement des documents pertinents pour atteindre la meilleure précision. Le système de recherche de modèle CAO final permet de passer de 16% des recherches avec un document pertinent dans les 100 premiers résultats à 62% des recherches.

Conclusion et perspectives

Nos travaux de recherche combinent des aspects théoriques et des mises en œuvre pratiques. Notre définition de KG et l'architecture KGBS alignent les travaux existants avec notre vision et notre cas d'usage. Elle réconcilie l'utilisation confuse des termes *graphe de connaissances* et *ontologie* dans la littérature, posant les bases pour comprendre nos travaux de recherche. L'architecture KGBS place cette définition de KG dans un contexte de système d'information avant de plonger dans les méthodes pour aborder des parties spécifiques de l'architecture. Bien que nos travaux proposent des approches pour les composants de l'architecture KGBS, nous n'avons pas pu tous les aborder. De plus, les directions que nous avons explorées nécessitent encore d'être approfondies pour traiter correctement les processus d'extraction et de modélisation des connaissances. Pour la consommation de connaissances, nos travaux considèrent uniquement le cas d'usage de la RI et laissent à des travaux futurs les composants de validation des connaissances. Les parties que nous n'avons pas abordées dans ces travaux sont représentées par des couleurs plus opaques dans la figure 6.9 récapitulative.

Les approches que nous présentons traitent séparément l'extraction, la modélisation et la consommation des connaissances. Les objectifs à long terme de ces travaux sont de mettre en œuvre notre architecture KGBS dans sa globalité sur un seul cas d'usage pour évaluer sa pertinence. Puis d'autres travaux mettant en œuvre notre architecture sur d'autres cas d'usage devraient la faire évoluer. De tels travaux futurs doivent avoir pour objectif d'enrichir notre architecture KGBS pour atteindre un niveau de détail permettant l'implémentation de composants individuels tout en assurant leur intégration dans le système global. Une difficulté est de trouver un corpus simple associé à une ontologie de référence pour évaluer le processus complet. Ce corpus doit également être annoté pour la ou les tâches à évaluer.

Une fois chaque composant de l'architecture mis en œuvre et intégré, deux directions potentielles existent. La première consiste à implémenter d'autres cas d'usage de la consommation des connaissances pour démontrer la viabilité de notre architecture KGBS. La deuxième direction est d'évaluer l'architecture à grande échelle avec des corpus de tailles représentatives de cas d'usages réels. Ces évaluations de mise à l'échelle doivent aussi être réalisées pour les approches que nous proposons. En effet, nous n'avons testé à grande échelle que notre système de RI fondé sur les KG abordant le cas d'usage de notre partenaire industriel.

L'implémentation de notre architecture nécessite cependant des étapes plus petites. L'architecture KGBS aborde un large éventail de domaines directement liés à nos travaux et pour lesquels nous avons proposé des approches mais également des domaines connexes que nous n'avons pas pu traiter. Nous abordons d'abord des directions concernant les domaines connexes avant de discuter celles spécifiques aux approches que nous proposons.

La tâche la plus critique liée aux KG est la liaison d'entités (EL pour "*Entity Linking*"). L'EL a pour objectif de lier les entités d'un KG avec leurs occurrences dans les documents d'un corpus. Cette tâche est essentielle pour exploiter un KG supportant tout cas d'usage. Dans nos travaux nous avons contourné la complexité de l'EL en considérant cette tâche comme une tâche de RI distincte, que nous avons abordée avec une approche BM25 traditionnelle. Nous avons exploré certaines méthodes EL existantes mais avons constaté que les méthodes de l'état de l'art reposent sur un grand nombre d'exemples annotés pour entraîner des réseaux de neurones profonds. Ces exigences en matière de données annotées rendent complexe l'adaptation de ces solutions à notre cas d'usage.

Bien que des exemples annotés soient essentiels pour atteindre les performances de méthodes fondées sur les réseaux de neurones profonds, nous avons constaté que ces méthodes négligent le processus itératif pratique nécessaire pour atteindre un tel niveau. De plus, les approches fondées sur les réseaux de neurones nécessitent un contexte que l'on ne trouve que dans de grands documents tels que des articles de presse mais pas dans nos courts textes descriptifs ou requêtes utilisateur·rice·s formées de mots-clés. Un objectif à court terme est donc de mettre en œuvre des méthodes d'EL utilisant le contenu linguistique directement disponible dans le KG. Nous avons

commencé à suivre cette approche avec notre collègue de l'équipe de recherche. Nous avons lancé un projet open-source parallèle nommé *Buzz-EL*[5].

Une autre tâche liée à l'OL pour laquelle on trouve peu de méthodes dans la littérature est l'extraction d'axiomes. Nos travaux abordent l'extraction d'axiomes en se concentrant sur les axiomes exprimés en OWL et en exploitant les résultats des schémas ontologiques (ODP pour "*Ontology Design Patterns*") [GPSS09]. Cependant, les ODPs conduisent le processus d'OL à générer des constructions OWL particulières à partir des concepts et relations extraits. Ils doivent également encore être sélectionnées manuellement. Une direction de recherche est de définir comment les cas d'usages ciblés lors du démarrage d'un projet de KG pourraient affecter les types d'axiomes appris. Nous pensons que traiter cette question de recherche serait une étape significative vers un OL allant au-delà des outils d'aide à la conception d'ontologie en apprenant des ontologies directement exploitable dans une application.

Au cours de nos travaux, nous avons observé la montée en popularité des LLM et leurs nombreuses applications, telles que la RAG (Retrieval-Augmented Generation), étroitement liées à la RI. Les LLM sont des modèles de langage pré-entraînés sur des tâches génériques dans le but de maîtriser la langue naturelle. Ils sont ensuite affiné avec un second apprentissage pour aborder des tâches particulières, entre autre la génération de texte en réponse à une instruction. Les LLM ont ainsi récemment montré des résultats prometteurs sur tâches spécifiques, grâce à leurs capacités de génération de text en réponse à des instructions. Les modèles de RAG combinent une mémoire paramétrique et non paramétrique pré-entraînée pour la génération de langage (traduit de [LPP+20]). Ici, la mémoire paramétrique sont les poids internes du LLM, et la mémoire non paramétrique est typiquement un grand corpus de documents textuels. Avec l'intérêt récent pour les LLM, la communauté des KG a exploré les KG comme mémoire non paramétrique pour les LLM. Les LLM et le RAG devraient être explorés dans des travaux futurs. Ces travaux devraient explorer RAG comme un cas d'usage de consommation de connaissances dans notre architecture KGBS et étudier l'utilisation des LLM pour chacun de nos composants OLAF. La collègue avec qui nous avons développé OLAF a déjà commencé des travaux dans ce sens[6].

Nous discutons maintenant les directions de recherche spécifiques aux approches que nous avons explorées dans nos travaux de recherche. OLAF est implémenté comme une boîte à outils afin que chacun puisse combiner ses approches pour chaque sous-tâche du processus d'OL. Les travaux futurs devraient, à court terme, rendre cette bibliothèque plus accessible dans le but de former une communauté pour enrichir OLAF de nouvelles approches et créer des corpus et ontologies de références. Ces derniers sont essentiels pour comparer et évaluer les méthodes. Certains composants de OLAF nécessitent également une exploration plus approndie. Nous avons déjà mentionné l'extraction de relations et d'axiomes. Les travaux futurs devraient également étudier des méthodes automatiques pour évaluer les ontologies apprises.

Nous avons abordé le cas d'usage de RI de notre partenaire industriel en proposant une approche pratique pour passer d'un système de RI fondé sur le texte à un système fondé sur un KG. Dans nos implémentations nous avons simplifié l'extraction de concepts en la traitant comme une tâche de RI. Nous pourrions explorer de nombreuses approches différentes d'EL. Cependant, les travaux futurs devraient d'abord exploiter des suggestions de complétion de recherche. En effet, suggérer des complétions est un moyen transparent d'avoir des retours explicites des utilisateurs. Nos résultats expérimentaux démontrent le gain de performance obtenu en ajoutant une phase d'extraction de concepts et en passant à une recherche fondée sur un KG. Cependant, l'implémentation actuelle de la structure du KG est limitée à 4 groupes différents de concepts. Nous pouvons utiliser ces instances de concepts pour implémenter des suggestions de complétion de recherche. Cependant, une étude approfondie des recherches utilisateur·rice·s et de leurs concepts pourrait améliorer le contenu du KG.

Dans nos modélisations de connaissances nous structurons le KG comme un ensemble de taxonomies organisant les concepts de manière hiérarchique. Les concepts sont liés entre les tax-

[5] https://github.com/schmarion/buzz-el (Consulté le Thursday 3rd October, 2024)

[6] https://github.com/wikit-ai/olaf-llm-eswc2024 (Consulté le Thursday 3rd October, 2024)

onomies par des relations transversales. Cette structure de KG est celle que nous utilisons pour démontrer la pertinence de notre ontologie de RI. Elle est dérivée de [LRR08]. Les travaux futurs devraient explorer l'amélioration de nos expériences industrielles avec notre ontologie de RI.

Pour conclure, ces travaux ont exploré la complexité d'un système fondé sur un KG et ont implémenté certaines parties d'un tel système avant de les appliquer à la RI. Nous avons exploré des approches prometteuses pour les différents composants de l'architecture KGBS. Les différentes parties du système peuvent être optimisées individuellement tout en analysant l'impact sur l'ensemble du système. Chaque méthode proposée contient de nombreuses variables qui peuvent être ajustées en utilisant des méthodes d'apprentissage supervisées par exemple.

Part II

Introduction

Context

These industrial research works have been conducted in partnership with the company TraceParts[7]. TraceParts is one of the world's leading Computer-Aided Design (CAD)-content platforms for Engineering, Industrial Equipment, and Machine Design. It has over 5 million registered members from 1.3 million companies and actively sources product information and technical data from over 195 different countries. Available free of charge to millions of Engineers and Designers worldwide, the TraceParts CAD-content platform, *www.traceparts.com*, provides access to over 1,880 supplier-certified product catalogues and billions of 2D drawings and 3D CAD models and product datasheets that perfectly match the digitalisation needs of Design, Purchasing, Manufacturing and Maintenance processes and operations, in virtually any industrial sector. End users search the platform content via a full-text search field, the TraceParts' classification or an alphabetically organised list of catalogues, each defining its specialised classification. The use case we have worked on considers a technical document corpus composed of CAD models and their descriptions. It addresses the challenge of bringing structured knowledge to support the TraceParts CAD-content platform model retrieval process.

The characteristics of the CAD-model descriptive content corpus we are considering naturally lead to exploring Knowledge Graphs (KG) as structured knowledge to support our CAD-model retrieval use case. Indeed, TraceParts content is composed of highly technical and domain-specific vocabulary in 25 different languages. Moreover, the CAD models cover a wide range of technical domains such as mechanical components, pneumatics, sensors and measurement systems. Traditional text-based approaches to IR fall short when dealing with multilingual vocabularies, where words can have different meanings depending on the context. Hence, we need to integrate some computer-processable knowledge, such as KGs, to extract the relevant information from user queries and documents and make sense of them by considering their technical context.

KGs aggregate heterogeneous data and represent knowledge in a machine-readable format. They are graphs intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities [HBC+21]. They are knowledge digital artefacts widely used in IR use cases. Besides providing computer-processable knowledge, such artefacts enable reasoning over the modelled knowledge to derive new facts and draw conclusions.

The term *Knowledge Graph* employed with a meaning close to today's one has been used at least since 1972 [Sch73]. Over the years, the technology has gained attention supported by different events such as the announcement of the Google KG in a 2012 blog post *Introducing the Knowledge Graph: Things, not strings*[8]. Most recently, KGs have been identified as candidates to support generative Large Language Models (LLM) by bringing explicit knowledge into their generation process. However, with KGs' rapid adoption in enterprises, many systems have been labelled as KGs, bringing much confusion about what a KG is.

Amongst the many applications KGs support, IR is a recurring one. IR is about finding material of an unstructured nature that satisfies an information need from within extensive collections [MRS08]. Though nowadays the term *information retrieval* denotes the computing notion, the process dates back to before the invention of computers and takes its roots in library science [MRS08]. Humans produce knowledge they convey historically in written documents. Naturally, IR as a research field has first studied methods to quickly and effectively retrieve particular pieces of information in large corpora of books. Before the first digital computers in the 1950s, humans had structured knowledge with hierarchically organised categories. Computers are expert machines to process structured content but are challenged when processing unstructured content such as text. Hence, once computers were identified in the early 1960s as essential to perform IR tasks, the IR-related research fields shifted to designing methods digitally structuring texts corpora for search. The most recent research explores LLMs and KGs.

[7] https://info.traceparts.com/about-traceparts/

[8] https://blog.google/products/search/introducing-knowledge-graph-things-not/ (Accessed on Thursday 3rd October, 2024)

Motivations

Since the beginning of the digital age in the 1950s, the amount of data we store and their diversity have steadily grown. As a result, finding the specific pieces of information we need in this vast amount of data has become critical. The emergence of the World Wide Web in the 1990s has contributed to a new step in data production and sharing, intensifying the critical need for effective IR methods. The popularity of the web and companies' search engines illustrates such a need. More recently, technological progress, particularly the processing power, eased the production and sharing of much more complex and diverse data, such as CAD models. While historically, searching for information was done using text, most recent applications enable searching using multi-modal queries such as using images or audio as input queries, e.g., Google Lens[9], and Shazam[10].

In engineering, many technical documents come with 3D models. Those 3D models, called CAD models, are generally considered central representations used to convey knowledge and information along the product design process [LPMG19]. In recent years, the amount of data associated with product design has steadily increased. Companies have become aware of the strategic importance of sharing the knowledge accumulated in those component designs. Hence, many research works address the CAD model retrieval challenges focusing on the CAD models with particular retrieval use cases, such as similar or equivalent design retrieval [QGY+16], CAD assembly retrieval or compatibility and function-oriented retrieval [LPMG19]. While some CAD model retrieval works focus on the CAD models, others consider the associated metadata, particularly textual data [LRR08]. Our works consider the latter case, leveraging a KG describing the concepts and relations used in our CAD model corpus descriptions.

Our digital computers are machines designed to process structured information. However, a wealth of unstructured text content exists as humans have always used spoken and written languages to communicate. Text is also the primary medium through which users interact with computers. Language, i.e., words and sentences, is how we are the most used to express our request to a search engine. However, natural language is very versatile. Many words and sequences can be used to express the same request, e.g., synonyms, even within the same language. Different communities use different words to express the same real-world entity and the same words to reference different real-world ones, i.e., synonyms and homonyms, respectively. For instance, the word *apple* can refer to a fruit or a company, and *avocat* in French to a lawyer or a fruit. Domain experts organise text corpora in related categories to help computers cope with unstructured natural language text. Different knowledge structures exist, such as thesaurus, taxonomies and ontologies. A significant part of their objectives is to align natural language terms with concepts, i.e. formalising the natural language content to ease search. Particularly relevant to our works are ontologies. Though many definitions for ontology in the computational sense exist [Kee20], a relevant one is "in the context of computing, an ontology is a concrete, formal representation of what terms mean within the scope in which they are used (e.g., a given domain)" [HBC+21]. The most recent incarnation of such a knowledge digital artefact is KGs.

KGs have been identified as a promising approach to modelling knowledge, particularly in domains requiring highly technical and detailed concepts and relations. Representing knowledge formally in a computer-processable language lets us more efficiently organise content. The better-structured documents are, the easier they are to search through. Hence, KGs are explored and used in many knowledge-intensive applications, such as search engines and technical domains, such as healthcare and Architecture Engineering and Construction (A/E/C). More recently, KGs have been explored as candidates to make explicit the knowledge implicitly hidden in Large Language Models[11].

However, the terms *knowledge graph* and *ontology* are used broadly to reference data artefacts and implementations. Though many works attempt to provide definitions for these terms, e.g.,

[9] https://lens.google/

[10] https://www.shazam.com/

[11] https://arxiv.org/html/2306.08302v3 (Accessed on Thursday 3rd October, 2024)

towardKGdef2016, there is still much confusion around what those terms exactly mean and how they relate to each other [CBC+22]. Hence, before using KGs, we need to define our notions of KG and ontology.

Finally, the scientific literature exploring methods leveraging KGs for IR concentrates on specific use cases and KG implementations. Methods discussions need a broader study of how KGs can integrate an information system, regardless of the particular use case.

Contributions

These industrial research works explore Knowledge Graph-Based Systems (KGBS) for Information Retrieval (IR). Our use case considers a technical document corpus composed of Computer-Aided Design (CAD) models and their descriptions. Rather than leveraging the CAD models, we focus on their descriptive texts.

In this manuscript, we adopt a top-down approach to describe our works. We begin with two chapters of literature review. We first explore KGs and ontologies and how they relate. Our historical review and thorough study of the different definitions in the literature for ontologies and KGs leads us to derive our unifying KG definition. We consider ontologies a component of KGs. The Semantic Web standards influence our works, particularly on the implementations. Hence, we also map Semantic Web standards with our KG definition. We then explore the literature on IR, focusing on KG-based IR.

We found a need for more work exploring KGs as part of an information system. The methods presented in the scientific literature focus on the KG and how it supports the use cases considered for demonstration. Hence, after our literature review of KGs and their usage in IR systems, our contribution part first explores KGs as a component of an information system. For this chapter, we put aside our IR use case and introduce a KGBS architecture relating knowledge acquisition, modelling, and consumption arranged around the KG. We follow this system architecture to organise the presentation of our remaining contributions. Our contributions address knowledge acquisition, modelling, and consumption.

For our work, we do not have a pre-built KG or access to domain experts to construct it. Domain experts are rarely available, and their time is too expensive for a long and tedious task such as extensive knowledge sharing for ontology engineering. Hence, we study Ontology Learning (OL) and KG Construction (KGC), the research fields focusing on methods to automatically build KGs from diverse sources of knowledge. We address the knowledge acquisition component of our KGBS architecture by designing our Ontology Learning Applied Framework (OLAF) collaboratively with some of our research group members. We implement our framework as an open-source Python library to explore and evaluate different combinations of methods addressing each OL subtask. We use OLAF to learn ontologies from text automatically and build two ontologies to assess OLAF's pertinence, usability, and modularity.

We then focus on knowledge modelling for IR. In the literature, we found that methods leveraging OWL ontologies for IR use limited OWL reasoning. In particular, we found reasoning usages primarily focused on hierarchical relations and used offline to complete the KGs before using them. Hence, we introduce our IR ontology and demonstrate its usage with an OWL reasoning-powered IR system. Our approach explores OWL reasoning at runtime. While demonstrating our IR ontology, we illustrate an RDFS and OWL-focused implementation of our KG definition by pointing out each component.

Through our industrial experiments, our last contribution chapter explores an example of a knowledge consumption use case, IR and, more specifically, CAD model retrieval. We tackle the CAD model retrieval challenge our industrial partner TraceParts faces by implementing a KG-based approach at scale and using real-world data. We move from an existing text-based technical document retrieval system to a KG-based one. We leverage real-world TraceParts' CAD-content platform user interactions to evaluate our KG-based IR system proposal.

Figure 3 overviews our main contributions. In summary, these works present the following ones:

- A unifying definition of KG and an example of implementation based on the Semantic Web standards.
- An architecture for KG-Based Systems: KGBS architecture.
- A Framework for Ontology Learning and its implementation as an open-source Python library[12].
- An OWL IR ontology and its usage demonstration.
- A study of a text-based IR system compared to a KG-based one evaluated with an industry use case corpus and user feedback.

Thesis Road-map

Chapter 1 This manuscript's first chapter addresses the challenge of defining Knowledge Graphs and ontologies. We also explore the relation between KG and ontologies and with other common terms often appearing in the literature about KGs and ontologies. To provide context into our KG definition, we first look at how knowledge and data have evolved since the first computers. We then define our notion of KG before focusing on ontologies. We consider ontologies one component of a KG and take a Semantic Web perspective when defining them. We also explore the theoretical and practical meaning of the term *semantics*. Before concluding, we briefly review the literature focusing on automatic approaches to construct KGs and ontologies, i.e., Knowledge Graph Construction and Ontology Learning.

Chapter 2 These research works are about KG and IR. Hence, the second state-of-the-art review chapter concentrates on IR, particularly KG-based IR. As for the first chapter, it is essential to consider a historical perspective before diving into the most recent approaches. Hence, the first section of the second chapter introduces the key historical evolutions of IR, which started before the first computers. We then set the stage by introducing the key concepts to understand the IR literature. Chapter 2's last section is a state-of-the-art review focusing on KG-based IR. We break down this review section into two, first exploring the literature on IR using the term *knowledge graph* and then the one using the term *ontology*. We thereby point out some similarities and distinctions in the KG usages.

Chapter 3 This chapter introduces our first contribution, a KG-Based System (KGBS) architecture. This manuscript follows a top-down approach and starts with a broad KG-based system overview. We propose and describe our KGBS architecture, discussing how each component interacts. As this PhD work focuses on implementations using the Semantic Web standards and their related technologies, we explore such technologies as candidate ones for each component. We demonstrate that Semantic Web standards provide an approach for each KGBS component. Our work also concentrates on IR; hence, we illustrate the KGBS architecture applied to IR before concluding. We use our KGBS architecture as the backbone of this thesis. Each of the following contribution chapters addresses particular parts of our architecture.

Chapter 4 Part of these industrial works have been developed collaboratively with some of our research group members. We had different projects and expertise but shared the need to construct a KG from text without domain experts to support us. This fourth chapter introduces our Ontology Learning Applied Framework (OLAF) and its implementation as an open-source Python

[12] https://wikit-ai.github.io/olaf/ (Accessed on Thursday 3rd October, 2024)

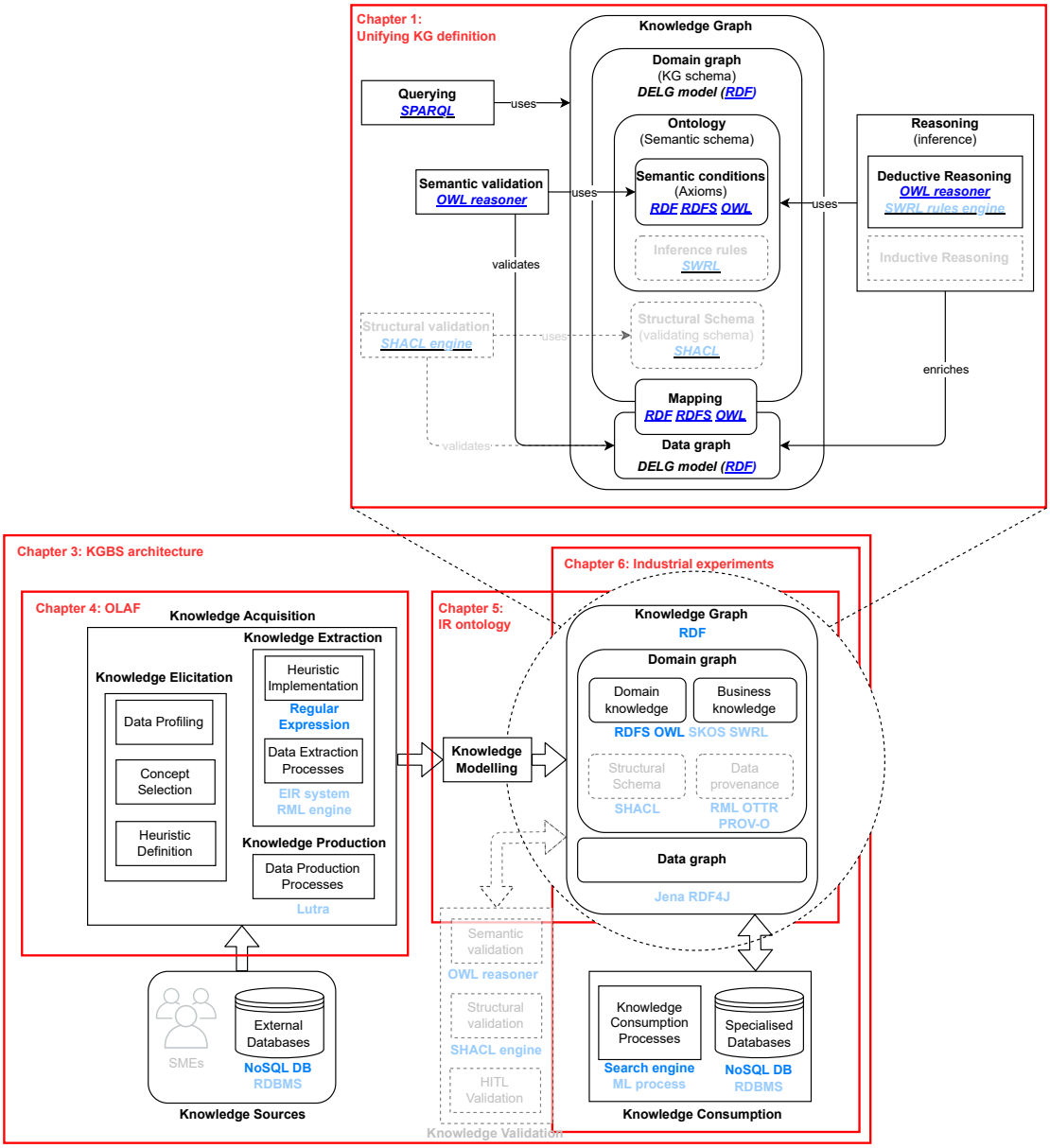


Figure 3: Summary of our industrial research works contributions. It includes our Knowledge Graph-Based System architecture at the bottom and our KG definition at the top right.

Square boxes refer to activities. Round boxes depict containers. The boxes imbrications denote sub-activities and sub-containers, respectively. Large empty arrows illustrate data flows and thin black ones actions. Large red boxes denote the components each contribution addresses. Some candidate technologies for implementing each component are mentioned in bold blue letters. The architecture parts turned into dashed and light colours are the parts left out in these works. (SMEs: Subject Matter Experts; HITL: Human In The Loop)

library. We first detail the framework, discussing each component before describing our experiments. We evaluate our OLAF implementation with two versions. For our first experiment, we relied on ourselves as domain experts. Hence, in our second one, to compare our learned ontology with an existing one, we use an ontology and leverage a generative Large Language Model to generate some corresponding texts.

Chapter 5 The previous chapters progressively bring us to some more applied ones. This fifth chapter introduces our IR Ontology and explores an approach to leverage OWL reasoning for powering an IR system. The literature review on KG-based IR shows that although many approaches leverage ontologies expressed in RDFS and OWL, they limit their use of RDFS and OWL reasoning to either RDFS with hierarchical relations or an offline process to materialise implicit facts. The IR ontology lets us use OWL reasoning at runtime to power an IR system. We first expand on some notions we only briefly explored in the literature review. We then describe our IR ontology before demonstrating its usage. We conclude this chapter by discussing the advantages and limitations of the IR ontology and the possible extensions. This chapter demonstration illustrates an implementation of our KG definition, characterising each component with an example.

Chapter 6 The last chapter dives into our implementations with TraceParts moving from a text-based IR system to a KG-based one. We first introduce the industrial context, presenting the company and the existing IR system. We then specify our IR use case detailing our corpora and user searches. We discuss our results and conclude with some directions for future work.

Finally, in this manuscript's conclusion, we come back to our works' key elements and discuss some future work directions.

Scientific productions

Peer-reviewed international conference papers

- Sesboüé, M., Delestre, N., Kotowicz, J.P., Khudiyev, A., Zanni-Merk, C., 2022. An operational architecture for knowledge graph-based systems. Procedia Computer Science 207, 1667-1676. https://doi.org/10.1016/j.procs.2022.09.224. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 26th International Conference KES2022.
- Schaeffer, M., Sesboüé, M., Kotowicz, J.P., Delestre, N., Zanni-Merk, C., 2023. Olaf: An ontology learning applied framework. Procedia Computer Science 225, 2106-2115. https://doi.org/10.1016/j.procs.2023.10.201, 27th International Conference on Knowledge Based and Intelligent Information and Engineering Sytems (KES 2023)

Open-source software library

- Ontology Learning Applied Framework Python library implementation: https://wikit-ai.github.io/olaf/

Part III

Related works

Chapter 1

Knowledge Graphs

" Knowledge graphs can be considered to achieve an early vision in computing, of creating intelligent systems that integrate knowledge and data on a large scale. "

Sequada J. and Lassila O.

Contents

Technologies and systems labelled with the term *Knowledge Graph* (KG) are becoming ubiquitous in today's enterprise information systems and applications. Many academic research fields, such as Information Retrieval (IR) [RMdR20] or graph machine learning, have adopted KGs. They are data artifacts used as the backbone for various data-savvy systems with applications ranging from question-answering over recommendations to predicting drug-target interaction. As KGs aggregate heterogeneous data and represent knowledge in a machine-readable format, they are a good fit to solve, among others, IR problems, and an essential technology for natural language processing (NLP), computer vision (CV), and commonsense reasoning [NGJ+19].

The term has recently gained interest, particularly in the business world[1], and is used in various contexts as a trendy word. The tech giant Google announcing the Google KG in its famous 2012 blog post *Introducing the Knowledge Graph: things, not strings*[2] is known as the igniter for the resurgence in popularity of KGs. The field has since been actively researched, and we noticed in 2023 a new regain in interest as this computer-processable knowledge representation has been identified as a good candidate to insert explicit — or at least human-understandable — knowledge into Large Language Models (LLM). The term *Knowledge Graph* employed with a meaning close to today's one is not new and has been used at least since 1972, as shown by Schneider's 1973 publication [Sch73], which refers to a graph of knowledge units supporting a computer system for education. And the first systematic study with the term *Knowledge Graph* appeared in 1987 with the Ph.D. thesis of R.R. Bakker [Bak87], *Knowledge Graphs: Representation and Structuring of Scientific Knowledge*. [GS21].

As noted by the authors of [CBC+22], due to the relative ease of creating and visualizing the schema and the availability of built-in analytics operations, KGs are becoming a popular solution for turning data into intelligence in enterprises. With the hype brought by the largest technology giants adopting KGs and similar ideas, many systems have been labelled as KG, bringing much confusion about what a KG is. This section provides an overview of the ideas and systems hidden behind the term *Knowledge Graph*, and introduces the definitions we will use to present our work.

We first briefly explore the historical context surrounding KGs before discussing works tackling the different definitions of KG. Then, a KG is decomposed into components, which are further discussed. Ontology and Semantics are central to our work, hence they deserve their own sections. We conclude with a summary of the definitions used in our work.

1.1 Brief history of Knowledge Graphs

Following the famous quote by George Santayana, "Those who cannot remember the past are condemned to repeat it," some researchers studied the lineage of KGs to understand better the ideas and chain of thoughts that lead to today's KGs[CBC+22, GS21, HBC+21]. We summarise these works here, in particular Gutierrez C. and Sequada J. work[GS21].

There are different approaches to the history of KGs. In [HBC+21] annexes, the authors break down the timeline into two periods, before and after 2012, i.e., the announcement of the Google KG. [GS21] focuses explicitly on the history of KGs, studying the technological and societal changes leading to today's notion of KG. The authors break down their analysis into five periods, starting in the 1950s with the first computers and programming languages. Each period is analysed along three axes: knowledge, data, and the combination of both. Some key references are provided. We now summarise each period using the vocabulary introduced in [GS21].

The advent of the Digital Age, roughly spanning over the 1950s and the 1960s, is marked by the first programming languages. At the time, computers were reserved for a few industrial applications. There arose the awareness of the need to represent knowledge using logical structures and formal languages to reason about data the way humans do. These structures aimed at easing

[1] https://www.gartner.com/en/articles/what-s-new-in-artificial-intelligence-from-the-2023-gartner-hype-cycle (Accessed on Thursday 3rd October, 2024)

[2] https://blog.google/products/search/introducing-knowledge-graph-things-not/ (Accessed on Thursday 3rd October, 2024)

the search in large spaces. This era saw the first methods to retrieve information from unstructured sources and the root languages and principles that later led to the well-known Relational Database Management Systems (RDBMS). However, there were still some heavy limitations, in particular regarding the capabilities and cost of computing hardware. The emergence of graphical representation for knowledge began, but there was still a gap between the visual representation and its encoding for processing by machines.

The Data and Knowledge Foundations era in the 1970s witnessed a much broader adoption of computing in industry, leading to increasing storage, processing power and expertise. The need for methods to process, understand and manage more data became clearer. The challenges were tackled by enhancing the independence of data and software, leading to Database (DB) query languages such as SQL. On the knowledge side, the focus was on the meaning of data. Data organisation as graph structure began with semantic networks and their implementation into real systems. Nevertheless, their weak logical foundations were criticised. First, First-Order Logic (FOL) and next Description Logics (DLs) as a tractable subset of FOL, were identified as reliable logical foundations to implement the meaning of data. It was in the 1970s that data and knowledge started to integrate. Examples of such early integration are the field of logic programming with Prolog and the rise of expert systems. It is also the birth of the knowledge acquisition field, focusing on methods to acquire knowledge from various sources. However, some limitations arose on the data side with the inflexibility of the traditional data structures and on the knowledge side with the weaknesses of the logical foundations.

The Coming-of-Age of Data and Knowledge in the 1980s saw the transition of computing from industry to homes with the emergence of personal computers. The increasing computational power led to new research fields and more complex data artefacts to manage. The need for a data representation independent from the software program was evident, leading to investigations on combining object-oriented programs with DBs. It was the rise of object-oriented DBs. On the knowledge side, this period led to a much better understanding of the trade-off between the expressive power of logic languages and the computational complexity of reasoning tasks, i.e., the cost of deducing new facts. Reasoning at a large scale was yet to be possible, and this would be known as the knowledge acquisition bottleneck. Though some other Logic and non-monotonic reasoning techniques were explored, DLs still stands out. Expert systems were at the centre of the AI hype. They started to show business value, but by the 1990s, they proved hard and expensive to update and maintain. This period also witnessed the Japanese 5th Generation Project, which aimed at adopting logic programming as a basis to combine Logic and data, i.e., creating computers reasoning base on logical deduction. The academic side explored this combination of Logic and data by layering logic programming on top of RDBMS, giving rise to deductive DBs. Datalog, a subset of Prolog for relational data with clean semantics, became the query language for such DBs.

In the 1990s, the *Data, Knowledge, and the Web* era saw first the emergence of the World Wide Web. This global information infrastructure revolutionised traditional data, information, and knowledge practices. Second, every aspect of society has started to be digitalised. The wealth of data generated needed to be analysed, which led to data warehouse systems supporting large-scale and multidimensional data analytics known as On-Line Analytical Processing (OLAP). New areas of research focused on finding patterns in data. The data community moved toward the Web, which triggered the need for distributed self-describing data. Various languages, such as the Resource Description Framework (RDF), were implemented. Ontologies started to be integrated into systems, and the notion was defined as a "shared and formal specification of a conceptualisation" by Gruber[Gru95]. On the research side, the focus was on methodologies to design and maintain ontologies, e.g., METHONOLOGY or CommonKADS. It quickly became apparent that data needed to be connected and processed at web scale. However, the computational power was not enough to handle this new wealth of data. In particular, one challenge was to define how to cope with formal reasoning at such a scale. Various languages were developed to describe and query data on

the Web, and the Semantic Web (SW) project started. The goal was to combine technologies such as ontologies, Logic, DBs, and IR for applications targeted at the Web.

The era of *Data and Knowledge at Large Scale* in the 2000s gave rise to e-commerce and big Web companies, which pushed the data management barriers. The constantly increasing wealth of data, along with new and more easily accessible computation power, enabled new statistical methods and the introduction of deep learning. On the data side, approaches previously hidden by the success of RDBMS regained interest with the NoSQL (Not Only SQL) DBs, which considered other data storage and processing systems based on other models such as the column, document, key-value, and graph data models. On the knowledge side, the DLs community developed new DLs profiles to cope with specific reasoning needs by reducing complexity and implemented reasoning programs into systems. Statistical applications to knowledge via machine learning and neural networks envisioned in the 1960s were now working in practice. In this period, two distinct threads emerged when considering the connection between data and knowledge. Part of the community focused on statistical approaches embodied by neural networks. The other, closely related to the SW project, which by the time was well established, focused on symbolic approaches based on Logic. The 2001 article "The Semantic Web" by Tim Berners-Lee, Jim Hendler, and Ora Lassila [BLHL01] is considered a landmark generating much excitement in academia and industry. Both worked together through the World Wide Web Consortium (W3C) and developed, among others, the RDF, RDFS, OWL, and SPARQL standards. We will further discuss The SW and those standards in section 1.3.1.

There has always been a need to represent, exchange and reason about knowledge. Formal knowledge representations were first studied, and technological innovations brought the concepts to scale with a wealth of data accessible today. Our work is developed in a context where the strengths and weaknesses of two technologies are acknowledged. On the statistical side, inductive approaches embodied by deep learning with Large Language Models (LLMs) are the most recent incarnation of inductive knowledge representation and reasoning. As the knowledge is internally and implicitly encoded, their limitations primarily lie in interpreting their results, which makes their usage uncertain. On the formal deductive knowledge modelling side, KGs are the realisation of old ideas with foundations in Logic modelling knowledge with symbols. Their weaknesses stem from the knowledge acquisition problem, making them hard to build. These approaches represent knowledge implicitly and explicitly, respectively. Since the mid-2010s, the research community turns towards merging both to support one another. As Sequada J. and Lassila O. mentioned [SL21], KGs as we know them today result from the convergence of ideas and technology. Nowadays, various organisations use the Knowledge Graph keyword to refer to data integration, giving rise to entities and relations forming graphs. Academia began to adopt this keyword to loosely designate systems that integrate data with some graph structure, a reincarnation of the Semantic Web and Linked Data [GS21].

This succinct and non-exhaustive history of KGs demonstrates at least two points. First, academic work is tidily coupled and driven by business needs. Second, research leads to technological innovations, inducing societal changes and new needs and technologies over and over again. Most technologies are built on top of old ideas and works which arose too early to be applied in industry and are now possible due to some innovations developed in parallel or at different times. Notice the pace of moving from an idea to a proof of concepts and finally to an industrial application. KGs are undoubtedly the result of many ideas and innovations, and their applications are yet to be discovered. This manuscript explores how some research results can be combined and applied to tackle an industrial need, an application of KGs to IR. The following sections discuss and define the terms used in our work.

1.2 Knowledge Graph definitions

As section 1.1 demonstrates, KGs have been around in various forms for a long time now. Indeed, in the past years, the term *knowledge graph* has gained several meanings across various usage sce-

narios [CBC+22]. With the rising interest in the topic, some works attempted to derive an exhaustive definition from the wealth of publications based on KG. The most prominent and complete work focusing on defining KGs is [HBC+21]. We now explore a selection of KG definitions that will constitute our definition game, a term borrowed from Maria Keet's work [Kee20] on introducing ontology engineering. We first quote each selected definition. Then, study their peculiarities and commonalities before discussing the KG components.

1.2.1 The definition game

This section discusses a non-exhaustive selection of attempts to define a KG. A subsequent section dedicated to defining an ontology summarizes Keet's ontology definition game. In their book focusing on Enterprise KG (EKG) [SL21], Sequada J. and Lassila O. define a KG as:

> *Representing a collection of real-world concepts (i.e., nodes) and relationships (i.e., edges) in the form of a graph used to link and integrate data coming from diverse sources. Knowledge graphs can be considered to achieve an early vision in computing, of creating intelligent systems that integrate knowledge and data on a large scale. In its simplest form, a knowledge graph encodes meaning and data together in the form of a graph:*
>
> - *Knowledge (i.e., meaning): Concepts and the relationships between the concepts are first class citizens, meaning that it encodes knowledge of how the domain users understand the world.*
> - *Graph (i.e., data): A data structure based on nodes and edges that enables integrating data coming from heterogeneous data sources, from unstructured to structured.*
>
> (1)

In their scientific publication "Knowledge graphs: Introduction, history, and perspectives" [CBC+22], Chaudhri et al. introduce a KG as:

> *a directed labelled graph in which domain-specific meanings are associated with nodes and edges.* (2)

Ehrlinger L. and Wöß W. study various KG definitions in [EW16] to derive their own one after proposing a generic architecture and a terminological analysis:

> *A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge.* (3)

On a more industry focus note, Barrasa J. and Webber J. from the graph database vendor Neo4J define in their book [BW23] a KG as:

> *a specific type of graph with an emphasis on contextual understanding. Knowledge graphs are interlinked sets of facts that describe real-world entities, events, or things and their interrelations in a human- and machine-understandable format.* (4)

Finally, in the most exhaustive work on defining what a KG is, the many authors of the book *Knowledge Graphs* [HBC+21] propose to:

> *view a knowledge graph as a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities. The graph of data (aka. data graph) conforms to a graph-based data model, which may be a directed edge-labelled graph, a property graph, etc. By knowledge, we refer to something that is known.* (5)

Definition	Distinctive features
(1)	mentions the integration of knowledge and data at a large scale and makes a clear distinction between knowledge and the graph structure.
(2)	mentions a particular graph model.
(3)	mentions the terms *ontology* and *reasoning*.
(4)	mentions a human- and machine-readable format.
(5)	is the most generic definition, though there is no mention of reasoning or scale.

Table 1.1: Comparison of KG definitions distinctive features.

We now study the definitions by highlighting recurring and distinctive features. Table 1.1 summaries the selected definition's distinctive features. Unsurprisingly, each definition mentions the graph data structure and a reference to knowledge or meaning. However, some are more restrictive and target specific graph data models, such as the Labelled Property Graph (LPG) model or the Directed Edge-Labelled Graph (DELG) model, e.g., definition (2). While the definition (5) provides a generic but loose definition of knowledge, each definition presents knowledge as a set of interlinked concepts or facts.

The idea of integrating data and knowledge, i.e., the meaning of data, in the same place also often comes back implicitly or explicitly, e.g., definitions (1) and (5). Indeed, a KG is always used as a structuring medium to express some knowledge. Note that in the definitions, expressing knowledge often comes down to describing real-world entities and their relationships. However, a KG can semantically describe very abstract things, such as thoughts or intentions. Similarly, integrating different data sources into one physical or virtual source is often mentioned as it is a recurrent use case for KGs, particularly in industry, e.g., EKGs ([SL21]).

The past decade has seen the rise of large Open KGs (OKG), such as DBpedia [LIJ+15], Wikidata [VK14], Yago [SAB+23], or ConceptNet [SCH16], as well as large close KGs such as the Google KG[3], or Microsoft's Bing KG[4]. As such, KGs are becoming popular and ubiquitous in various academic and industry projects, and their large scale often arises in publications mentioning KGs [EW16], e.g., (1).

The definition (3) specifically mentions the term *ontology* along with the process of reasoning. These terms and the term *semantic* often arise in KG works. They most probably bring confusion in the notion and implementations of KGs. Hence, we dedicate a section to each, 1.3 and 1.3.2, respectively. Ontology is associated with formal knowledge modelling, i.e., leveraging symbols to express knowledge, deduce new facts and check consistency using logical deduction processes. While it is one notion of semantics, the term is also used to reference inductive logic-based approaches, for which one of the most prominent examples is embeddings, i.e., knowledge represented as numerical vectors. Though the terms *semantic*, *meaning*, and *knowledge* are recurrent, it is rare for the terms *deductive logic* or *inductive logic* to be mentioned explicitly.

As a means to structure and express knowledge, KG definitions also often reference a machine- and human-readable format. While true, despite its limits, the intuitive graph structure is regularly leveraged for visualisation. In particular, many works confuse RDF with OWL – two technologies we discuss in section 1.3.1 – leading to poor logical definitions only tailored to fit visualisation software, e.g., trying to visualise OWL RDF triples. Visualising a data graph in a visual graph is easy and makes sense. In contrast, it is more challenging and more complex to visualise logical axioms as a graph. Hence some research are specifically tackling the issue, e.g., [PVCFCPGC23].

We selected some attempts to provide a short and exhaustive definition for KGs. However, many works expand their KG definition with either a feature- and use case-focused or a genus differentia description. In [BW23], the definition (4) is extended with references to the KG schema, distinguishing between the graph of data and the graph representing the schema. The part of the latter schema going beyond describing the graph topology is depicted as *organizing principles*

[3]Google blog post: Introducing the Knowledge Graph: things, not strings (Accessed on Thursday 3rd October, 2024)
[4]Microsoft's Bing KG blog post (Accessed on Thursday 3rd October, 2024)

enabling the user or a computer to reason about data. The authors state *The difference between a plain old graph and a knowledge graph is that the interpretation of the information in plain old graphs is encoded into the systems that use the graph rather than being part of the data itself. In other words, the organizing principle is "hidden" in the logic of the queries and programs that consume the data in the graph.* This point is central to our work and drives many ideas developed here.

To better understand the notions hidden behind the term KG and their implementations, it is essential to distinguish three components: the knowledge (data), the graph model and the schema. The following subsections discuss each component and address some closely related terms used in the literature to avoid confusion.

1.2.2 Knowledge

For the purpose of our work, it is not necessary to dive into the philosophical notion of knowledge. However, this section introduces some short philosophical discussions to clarify the terms used. It is intended as a pointer to relevant research. In particular, we discuss two commonly encountered topics when describing KGs: The Knowledge pyramid (DIKW) and the *justified true beliefs* definition of knowledge.

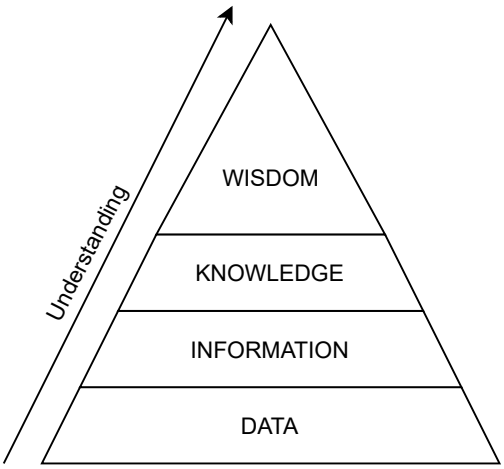


Figure 1.1: The Knowledge pyramid [Row07].

The Knowledge pyramid, reproduced in figure 1.1, is a model often quoted in the information and knowledge literature [Row07] to define and distinguish between Data, Information, Knowledge, and Wisdom (DIKW). The model origin is debatable, though the most referenced article is [Ack89]. Data is raw and can exist in any form, usable or not, e.g. "*8.6*". Information is data that has been given meaning useful or not, e.g., "*8.6 meter*". Knowledge is the appropriate collection of information that intends to be useful, e.g., "*the garage is 8.6 meters long*". Wisdom is the ability to make sound judgments and decisions[5], e.g., "*since the garage is 8.6 meters long I can fit my car and the trailer.*" Understanding is often used to reference the path from data to wisdom. Note that a KG in itself stops at the Knowledge level. Only when using the KG, do we reach the Wisdom level.

The traditional analysis of knowledge defines three components: truths, beliefs, and their intersection, true beliefs. Knowledge is then defined as justified true beliefs, and the following *Tripartite Analysis of Knowledge* is given: S *knows that p if and only if p is true;* S *believes that p;* S *is justified in believing that p*. This analysis of knowledge is the starting point of much of the twentieth-century literature on the analysis of knowledge.[6]

Considering the research questions tackled in this thesis, the short and loose definition of knowledge provided in [HBC+21] is enough: *By knowledge, we refer to something that is known.*

[5] Open HPI 2020 KG course 1.1 (Accessed on Thursday 3rd October, 2024)

[6] The Analysis of Knowledge, Stanford Encyclopedia of Philosophy, 2001. (Accessed on Thursday 3rd October, 2024)

We further point out that something known is always subjective and depends on a person or a group of people's beliefs. Hence, a KG is always a point of view on a particular domain, which can then conflict with another KG, i.e., another point of view.

1.2.3 Graph data model

When discussing KGs, it is essential to distinguish the knowledge it describes and the graph data model chosen to implement it. As stated by the authors of [HBC+21], at the foundation of any knowledge graph is the principle of first applying a graph abstraction to data, resulting in an initial data graph. Two main graph data models are used in practice: the Labelled Property Graph (LPG) model and the Directed Edge Labelled Graph (DELG) model. Some other models are less prominent but exist and are used for specific use cases, e.g., when high-arity edges are convenient, the hypergraph model based on set theory is sometimes used. We focus below on defining the LPG and DELG models as the most prominent ones. Our work leverages the DELG model. The formal definitions are taken from [HBC+21] in which **Con** denotes a countably infinite set of constants.

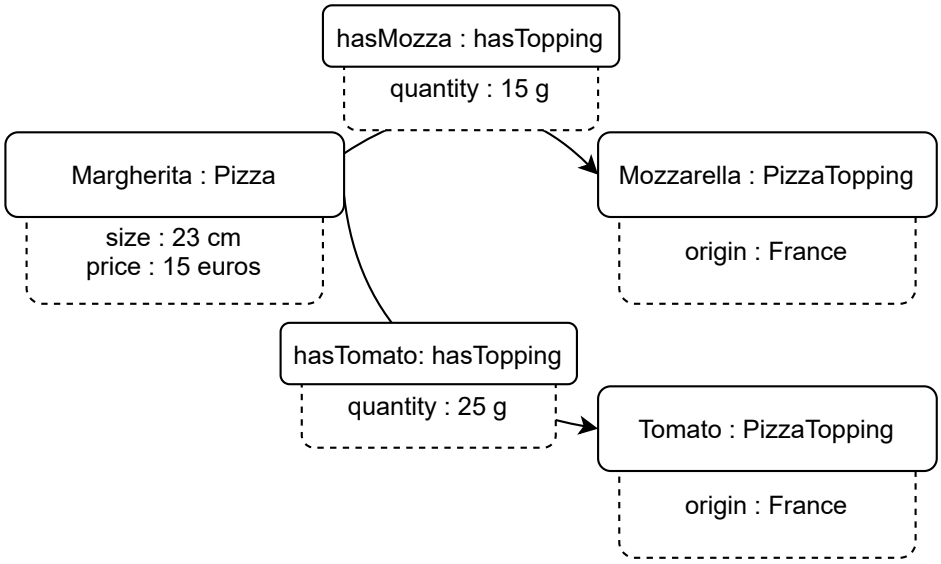


Figure 1.2: Labelled Property Graph model pizza example.

The LPG model is probably the most used graph data model in practice, particularly in industry. Nodes and edges are labelled, and the model allows a set of property-value pairs and a label to be associated with both nodes and edges. It has been designed to allow more flexibility when modelling complex relationships, easing the learning curve and visualisation. [HBC+21] provides the following formal definition we reuse:

Definition 1.2.1 (Labelled Property Graph (LPG))**.** A labelled property graph is a tuple $G = (V, E, L, P, U, e, l, p)$ [HBC+21], where

- $V \subseteq$ **Con** is a set of node ids,
- $E \subseteq$ **Con** is a set of edge ids,
- $L \subseteq$ **Con** is a set of labels,
- $P \subseteq$ **Con** is a set of properties,
- $U \subseteq$ **Con** is a set of values,
- $e : E \rightarrow V \times V$ maps an edge id to a pair of node ids,
- $l : V \cup E \rightarrow 2^L$ maps a node or edge id to a set of labels,

- and $p : V \cup E \rightarrow 2^{P \times U}$ maps a node or edge id to a set of property-value pairs.

Figure 1.2 provides an example of modelling a margherita pizza using the LPG model. Mapping the margherita graph in figure 1.2 with the LPG definition, we have:

- V = {M$argherita$, M$ozzarella$, T$omato$}
- E = {hasM$ozza$, hasT$omato$}
- L = {P$izza$, P$izza$T$opping$, hasT$opping$}
- P = {$size$, $price$, $origin$, $quantity$}
- U = {13cm, 15$euros$, F$rance$, 25g}
- e : {hasM$ozza \rightarrow$ (M$argherita$, M$ozzarella$), hasT$omato \rightarrow$ (M$argherita$, T$omato$)}
- l : {M$argherita \rightarrow$ P$izza$,
  M$ozzarella \rightarrow$ P$izza$T$opping$,
  T$omato \rightarrow$ P$izza$T$opping$,
  hasM$ozza \rightarrow has$T$opping$,
  hasT$omato \rightarrow has$T$opping$}
- {M$argherita \rightarrow$ {($size$, 23cm), ($price$, 15$euros$)},
  M$ozzarella \rightarrow$ ($origin$, F$rance$),
  T$omato \rightarrow$ ($origin$, F$rance$),
  hasM$ozza \rightarrow$ ($quantity$, 15g),
  hasT$omato \rightarrow$ ($quantity$, 25g),}

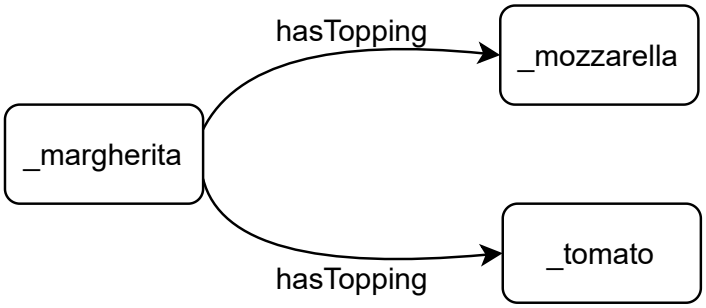


Figure 1.3: Directed Edge Labelled Graph model, pizza example.

Another prominent graph data model is the DELG model, defined as a set of nodes and directed labelled edges between those nodes. This minimal model offers flexibility in use cases like data source integration or metadata modelling. In our work, we use the Resource Description Framework (RDF), which follows the DELG model, has been standardised and is recommended by the World Wide Web Consortium (W3C)[7]. RDF is at the core of the SW, and we further discuss it in section 1.3.1. [HBC+21] provides the following formal definition:

Definition 1.2.2 (Directed Edge-Labelled Graph (DELG))**.** A directed edge-labelled graph is a tuple G = (V, E, L), where V ⊆ **Con** is a set of nodes, L ⊆ **Con** is a set of edge labels, and E ⊆ V × L × V is a set of edges. [HBC+21]

Figure 1.3 provides an example of modelling a margherita pizza using the DELG model. Mapping the margherita graph in figure 1.3 with the DELG definition, we have:

- V = {$\_margherita$, $\_mozzarella$, $\_tomato$}

[7] https://www.w3.org/standards/ (Accessed on Thursday 3rd October, 2024)

- E = {($\_margherita, has\text{T}opping, \_mozzarella$), ($\_margherita, has\text{T}opping, \_tomato$)}
- L = {$has\text{T}opping$}

The LPG model success can be attributed to an easier learning curve and visualisation. Some graph database vendors' heavy marketing also probably helped there. However, the extent to which it is natural to model situations as LPG is debatable. One usual example to motivate using the LPG model is the one of a flight between two destinations, which we would like to model as the node-edge-node *(Paris, flight to, New York)* with extra information about the flight like the date and times. But as Dean Allemang points out in a blog post[8], in our day to day life we speak about a flight to define the statement "my flight from Paris to New York is on Monday". Hence, it would make sense to model a flight as a node attached to other nodes for the starting and destination cities and the dates and times. This process is called reification when modelling using RDF and is often a pain point for new KG modellers. It also demonstrates that data can be converted from one model to another [HBC+21].

The choice of a graph data model is often tied to the originally intended KG usage. From our experience, the LPG model is often chosen when the KG use case requires some graph analytics leveraging graph algorithms. Most LPG-based database comes with built-in graph algorithms. The DELG model, particularly RDF, is chosen when the KG describes the metadata of any content and the use case involves data sharing and interoperability. Most well-known open-access KGs have an RDF representation as the exchange format becomes increasingly popular, though it might not be the original graph model, e.g. Wikidata [VK14].

Remembering the distinction between a KG and the graph model chosen for its implementation is essential. This point will be further illustrated in chapter 3. For instance, we might use RDF to implement the entire KG and export part of the graph into a LPG model for graph analysis. Unless specified, our work uses the DELG model, implemented with RDF. We motivate this choice in section1.3 as we leverage the SW standards.

1.2.4 Knowledge Graph schemata

Many works leveraging KGs briefly introduce them using variants of *KG = Schema + Data*. Though abstract, this definition fits well with most applications in which the schema is a simple typing system. However, the notion of schema is often misunderstood as it can take different forms with different purposes. In particular, the notion of the adjective *semantic* is associated with the KG schema and terms such as *semantic schema*, or *organizing principles* [BW23] are used. Here, we discuss two kinds of KG schemata, the semantic and validating schemata. [HBC+21] further define an emergent schema that we do not discuss here as it is unnecessary to understand our purpose. We refer the reader to the relevant section of [HBC+21] for extensive details.

The validating schema deals with the shape of the data graph. It enforces the completeness and structure of the data graph and enables its structural validation. Such schema enables us to say things like "A person has a name," enforcing each person node in the data graph to be linked to a name node. In contrast, the semantic schema deals with the logical consistency and meaning of the data graph, enabling semantic validation. It also enables reasoning, describing the rules which govern the data graph. Such schema lets us say that "every employee is a person". The semantic schema (aka. organising principles) often distinguishes a plain data graph from a KG [BW23].

To understand the impact and differences between a validating and semantic schema, we must introduce some assumptions made when interpreting a KG schema: the Open World Assumption (OWA) vs Close World Assumption (CWA) and the Unique Name Assumption (UNA) vs No Unique Name Assumption (NUNA). A validating schema considers the data graph complete and deduces that something does not exist if it is not in the data graph. For instance, if no node for employee Mike exists, the validating schema can deduce that no such Mike exists (at least as an employee).

[8] Dean Allemang's blog post: *Why I'm not excited about RDF-Star* (Accessed on Thursday 3rd October, 2024)

By opposition, a semantic schema is typically defined for an incomplete data graph where the absence of an edge between two nodes does not mean that the relation does not hold in the real world [HBC+21]. A semantic schema is typically used in an OWA setting in which for a statement not to hold, it needs to be explicitly stated as such. Reasoning over such semantic schema with the OWA can produce three kinds of response: *True*, *False*, or *We do not know*. Whereas reasoning in a CWA setting always leads to either *True* or *False*. Furthermore, under the NUNA, multiple nodes/edges labels in the graph may refer to the same entity/relation-type. We further detail these assumptions in this chapter following sections.

Semantic schemata are often rightfully associated with an ontology. Ontologies can be expressed in different frameworks and languages and have different levels of complexity. The most simple form of ontology is a simple typing system. As we will see, most ontologies for information retrieval do not contain much more than hierarchical relations. Since ontologies are central to our work, the next section examines their definitions and implementations.

1.3 Ontology

In 1998, Guarino N. pointed out the need to clarify the term "ontology" used in the Artificial intelligence (AI) community. The term is borrowed from philosophy and adapted to AI and its applications in various fields. In [Gua98], he tackles aspects of the application of ontologies in a formal manner. For our purpose, we do not need to grasp an ontology's theoretical and formal definitions fully. We instead focus on the term's meaning in the modern era of KGs in which many approaches are based on the inductive logic-based notion of semantics (see section 1.3.2), e.g., deep learning-based approaches.

Since ontologies have a long history in AI, countless attempts exist to construct a formal definition in the wealth of scientific literature on ontologies and their usage. However, none define ontology perfectly. Keet proposes the *definition game* in her book *An Introduction to Ontology Engineering* [Kee20] and discusses some of these definitions, highlighting their strengths and weaknesses.

As pointed out by Feilmayr and Wöss in [FW16], the various definitions frequently changing may indicate confusion, causing people from various research communities to use the term with different, partly incompatible meanings. The article dates back to 2016. At the time, one of the reasons for misusing the term as a buzzword came from the hype of the Semantic Web, which we discuss in section 1.3.1. We believe the statement is even more true today, notably since the interest regained in KGs. The term *knowledge graph* has become the new buzzword, particularly in industry and academia. This fact is illustrated by the recent academic publications reintroducing the notion of KG [HBC+21, EW16] and recent books stemming from industry on the topic [BW23, Hed22]. In these works, an emphasis is placed on how KG relate to Ontology.

Here, we do not intend to repeat Keet's definition game, and we refer the reader to [Kee20] for a broader discussion. However, we must cite the most quoted definition, i.e., the one Guarino N. provides in his discussion on ontologies [Gua98]:

> *An ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models.* (6)

The definition (6) is abstract, and for our purpose, we do not need to understand the theoretical peculiarities of the notion of ontology. The authors of [HBC+21] provide a much simpler definition that suits our needs:

> *In the context of computing, an ontology is then a concrete, formal representation of what terms mean within the scope in which they are used (e.g., a given domain).* (7)

DLs are a widely used formalism that represents knowledge and constructs ontologies. This formalism distinguishes between two and sometimes even three components of knowledge: terms, assertions or facts, and sometimes roles and relations, to make explicit distinctions between classes and relations amongst terms. The latter knowledge components are grouped into the Terminology box (T-box), the Assertion box (A-box), and the Relation box (R-box) [Kee20].

Though our work will focus on DLs-based ontologies, an ontology is not tied to such a logical framework. Other ones exist, such as First-Order Logic, Datalog, Prolog, and Answer Set Programming [HBC+21]. The authors of [HBC+21] consider all the mentioned technologies as logical frameworks and use the term *ontology* specifically for logical frameworks having a graph representation, e.g., DLs with OWL. In our work, though each technology mentioned above can be used to construct an ontology, we will focus on DL-based ones, particularly those implemented using the Ontology Web Language (OWL). Hence, the restriction made in [HPSv03] considering an ontology equivalent to a DLs Knowledge Base (KB) suits our implementation of ontologies here. The term *knowledge base* can be seen as the precursor of KGs in the 80's and 90's. The graph structure had not yet been identified as a robust data model to represent knowledge. We further define a KB in section 1.3.2.

In practice, an ontology is often confused with one of its simplest forms, which is a taxonomy. Multiple terms related to ontology and coming from information science are in use, such as thesauri or synonym rings. Hedden H. introduces ontology as a kind of controlled vocabulary in her book, *The Accidental Taxonomist* [Hed22]. She provides a table demystifying the various controlled vocabularies used in information science based on their expressive power and complexity. We reproduce Hedden's table in figure 1.4 for the reader to distinguish between the terms and will focus on taxonomies. The distinctive features of each controlled vocabulary are also mentioned. Many controlled vocabularies are often loosely labelled as ontology, particularly taxonomies and thesauri. In figure 1.4, complexity and expressive power rise from left to right. For our work, we need to remember that a taxonomy is a simpler ontology.

Less — Controlled Vocabularies -- Complexity / Expressiveness — More

Term List	Synonym Ring	Name Authority	Taxonomy	Thesaurus	Ontology
Ambiguity control	Synonym control	Ambiguity control Synonym control	Ambiguity control Synonym control Hierachical relationships	Ambiguity control Synonym control Hierachical relationships Associative relationships	Ambiguity control Semantic Relationships Classes Attributes

Figure 1.4: Comparison of controlled vocabulary types (reproduced from [Hed22])

We refer to taxonomy as any hierarchically organised controlled vocabulary. It corresponds to Hedden's Hierarchical Taxonomies [Hed22] and to the definition of taxonomy in the ANSI/NISO standard for controlled vocabularies [ANS10]:

> *A collection of controlled vocabulary terms organized into a hierarchical structure. Each term in a taxonomy is in one or more parent/child (broader/narrower) relationships to other terms in the taxonomy.* (8)

In definition (8), the parent/child relationships can take many forms and are not restricted to sub-typing ones, i.e., *is_a*. In information science, there exist many such parent/child relationships. We refer the reader to [Hed22] for an extensive discussion. For our purpose, we will distinguish between *isa*, *broader/narrower*, and *sub/super-category* relations.

In our work, we will use the term *ontology* to denote the KG schema as soon as the schema goes beyond a minimal typing system for node and edge grouping. In its simplest form, an ontology in

a KG as a typing system, defines at least the structure of the data graph, i.e., terms in the data graph are defined in the ontology. However, the ontology can not easily be used as a validating schema due to the underlying OWA making it harder to conclude an entity or relationship absence. We follow [HBC+21] vocabulary and distinguish between the data graph and the domain graph detailed in section 1.3.2. The ontology is part of the domain graph.

This section and the previous ones introduce some Semantic Web technologies that require clarification. The next section makes those clarifications, providing enough context to understand our work.

1.3.1 The Semantic Web and its standards

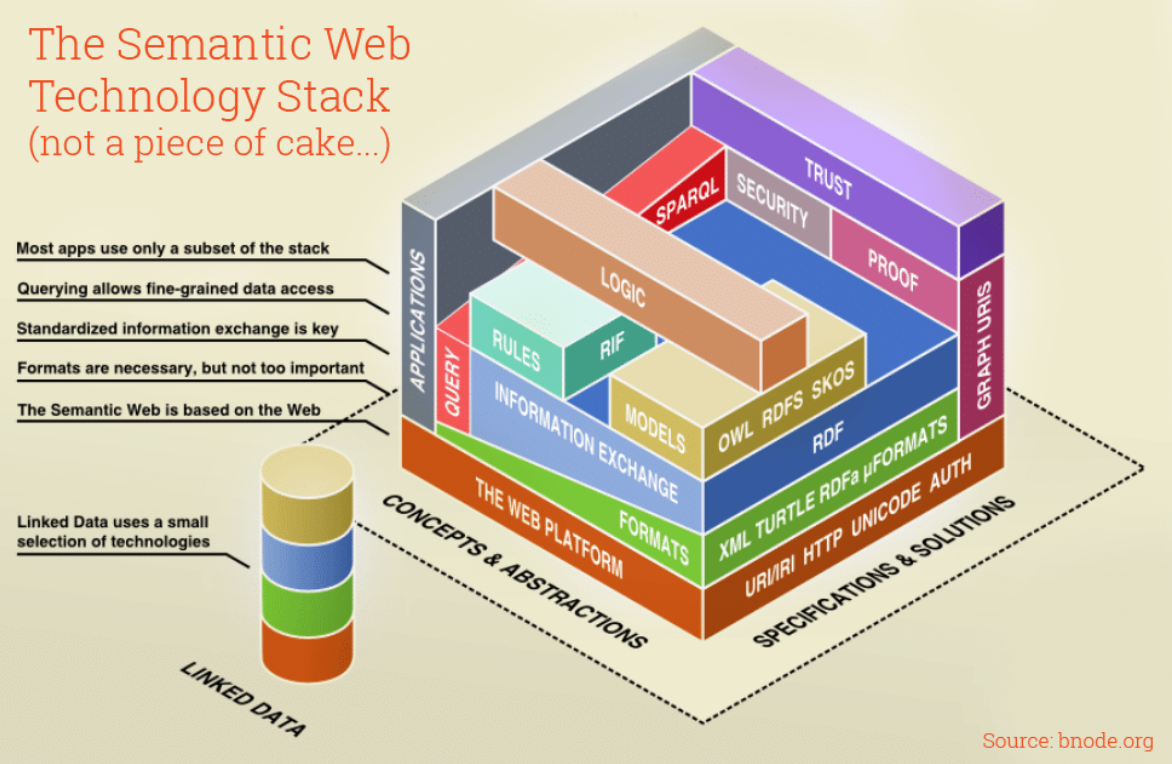


Figure 1.5: The Semantic Web Technology Stack[9]

Our implementations will leverage the standardised Semantic Web (SW) technologies. This section introduces what the SW is, along with the leading technologies.

The SW is a project aiming at enriching the World Wide Web, which in the 1990s and early 2000s had rapidly evolved as a web of documents for people rather than information that can be manipulated automatically. To tackle this issue, the SW augments Web pages with data targeted at computers and by adding documents solely for computers [BLHL01]. This extension of the SW is performed through standards set by the W3C. The SW could be interpreted as the most comprehensive KG[EW16].

The SW standards are built following what is known as the *Semantic Web Technology Stack* initially presented by Tim Berners-Lee in a 2000 talk[10]. Since then, the stack has evolved to include new standards and web technologies. Figure 1.5 presents the most recent SW Technology Stack in a picture borrowed from one of the triple store database vendors blog post[11]. The stack presents many technologies we do not need in our work. We will present RDF, RDFS, OWL, SPARQL, and

[9] https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies/ (Accessed on Thursday 3rd October, 2024)

[10] https://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html (Accessed on Thursday 3rd October, 2024)

[11] Ontotext blog post *What are Ontologies?* (Accessed on Thursday 3rd October, 2024)

some serialisation formats we use to implement our KGs. We refer the reader to the specific standards available on the W3C website for extensive details.

The Resource Description Framework (RDF) is the foundation of the representation languages for the SW. It addresses the issue of managing distributed data [AHG20] and is designed as a data exchange format. RDF was developed on top of the web infrastructure and relies on many of its proven features. Its basic building block is called a *triple* representing a directed edge in a graph with a source node (*subject*), a labelled link (*predicate*), and a target node (*object*). This simple basic structure enables the construction of any graph based on the DELG model and provides the flexibility to merge two RDF graphs easily as the union of two sets of triples. RDF has many other features, and we refer the reader to the W3C standard[CWL14] and the many tutorials available on the Web (of documents)[12] for extensive details. The most fundamental feature of RDF is that it defines different types of nodes, including among others, *Internationalized Resource Identifiers* (IRIs), which allow for global identification of entities on the Web; *literals*, which allow for representing strings and other datatype values such as integers and dates; and *blank nodes*, which are anonymous nodes that are not assigned an identifier [HBC+21] enabling the construction of arbitrary complex structures leveraged by the other RDF-based modelling languages such as OWL.

The W3C standardised language to query RDF data is the SPARQL Protocol And RDF Query Language (denoted using the recursive acronym SPARQL). This query language is closely related to the RDF structure. The query patterns are expressed using a variant of the Terse RDF Triple Language (Turtle), a textual syntax for RDF that allows an RDF graph to be completely written in a compact and natural text form, with abbreviations for common usage patterns and datatypes [BBLPC14]. SPARQL shares many features with other query languages, such as the famous SQL. We refer the reader to the W3C standard[HSP13] and the many tutorials available on the Web (of documents)[13] for extensive details.

RDF provides a vocabulary to define simple constructs such as lists and bags and for typing particular objects. As the standard is designed for the SW, it also states that everything is a resource identified by a Uniform Resource Identifier (URI). The latter is a short string designed to uniquely identify a resource on a network such as the Web. RDF alone enables drawing very limited conclusions based on the modelled data, i.e., to infer new facts. To enable the inference of more useful facts, RDF is extended with some standardised modelling languages. The RDF Schema (RDFS) is the simplest, and a more complex and expressive one is the Web Ontology Language (OWL). RDFS vocabulary provides many constructs enabling only the inference of new facts, i.e., the creation by logical deduction of new triples. One of the most well-known RDFS relationships is *rdfs:subclassOf*, which lets us construct class hierarchies. The RDF Semantics W3C recommendation [HIPSC14] list a set of rules defining what triples can be deduced based on RDF et RDFS triples. However, RDFS alone does not let us build constructs to ensure the inferred triples are consistent with the rest of the ontology. OWL goes beyond RDFS defining a vocabulary for constructs enabling to conclude the logical consistency of an RDF graph. For instance, we can say things like "a person can not be a car" and ensure that if a person is inferred to be a car, a logical inconsistency will be raised.

RDF is about graphs and modelling on the SW in RDFS, and OWL is about sets[AHG20]. The new inferred facts are drawn using DLs reasoning based on set theory. However, being based on DL, OWL is an expressive language. As pointed out in [GS21] and discussed in section 1.1, expressive power comes at the cost of reasoning efficiency. Hence, some subsets of OWL have been developed to enable more efficient reasoning in certain situations. Those OWL variants, based on subsets of DL, are called OWL profiles. An interesting quote from the designers of the SW illustrating such a need for a balance between expressive power and reasoning efficiency is "*The logic must be powerful enough to describe complex properties of objects but not so powerful that agents can be tricked by being asked to consider a paradox. Fortunately, a large majority of the information*

[12] Most of the RDF-based database (aka triple store) vendors have blog posts and tutorials introducing RDF and related technologies

[13] Most of the RDF-based database (aka triple store) vendors have blog posts and tutorials introducing RDF and related technologies

we want to express is along the lines of "a hex-head bolt is a type of machine bolt," which is readily written in existing languages with a little extra vocabulary" [BLHL01].

This section provides enough background information about the SW technologies we will use in our work. When needed, we will extend some matters. However, for the interested reader, we point to:

- [AHG20] for an extensive introduction to the SW and RDF and the built-upon modelling languages;
- [UDG18] for modelling specifically in OWL;
- [DuC13] to learn SPARQL.

Everything we present in our work can be implemented using various other technologies such as RDBMS-based ones. However, we focus on the SW technologies, which provide most of the resources we need. In particular, RDF lets us represent both the data and the schema. Hence, only SPARQL is needed to query the data graph and the ontology. We can query the KG schema the same way we query the data graph, which is impossible in other approaches, such as LPG-based ones. RDFS and OWL let us model the domains at hand and reason about it, generating new parts of the original data graph and checking for logical consistency. Furthermore, as RDF is designed to share data, many OKGs can be downloaded in RDF. Though in our work, we design EKGs – i.e., KG not necessarily intended to be shared outside of a company – we will leverage some OKGs. Using RDF from the start removes one unnecessary translation step. Finally, from an industry point of view, the fewer different technologies we use to design a system, the better it is.

Before concluding, we need to clarify one last topic, *semantic* as an adjective. In particular, what do we refer to in practice when labelling a system as semantic.

1.3.2 Theoretical and pragmatic meaning of semantics

In [NGJ+19], the authors point out that with the increasing adoption and use of KGs in different scenarios and use cases, three contrasting perspectives have emerged: symbolic representation versus vector representation, human curation versus machine curation, and "little semantics" versus "big semantics." These notions are illustrated in [HBC+21] by comparing deductive vs. inductive knowledge. Our work focuses on deductive knowledge. However, we might indirectly use some inductive knowledge, typically in the form of representative vectors. For our purpose, it is enough to consider inductive knowledge as knowledge extracted by generalising patterns automatically found in the data. Many methods exist to mine and abstract such patterns. The most well-known ones are based on deep neural networks that construct vectors representing patterns, i.e. implicit knowledge. Vectors are then much easier for computers to manipulate. However, the latter example is a numeric one. Symbolic approaches to inductively acquire knowledge also exist, e.g., rule mining. In [HBC+21], the authors propose formal definitions of deductive logic based on graphs, which we now summarise using their own words.

Following [HBC+21], we adopt a top-down approach to define deductive logic, adapting our definitions to focus on the particular application based on DLs and its W3C standardised DELG representation, i.e., RDF. We first introduce generic definitions before refining them to fit the peculiarities of DLs and mapping the terms with practical OWL-based meanings. Furthermore, we distinguish the definitions specific to graph representations and those pertaining to logic formulas, which can result in graph expansions. In this manuscript, like the authors of [HBC+21], we focus on ontologies, which constitute a formal representation of knowledge that can be represented as a graph, e.g., an ontology expressed using the OWL language and serialised in RDF. Note that we use the acronym OWL for convenience but refer more specifically to OWL 2[HKP+12].

In [HBC+21], as mentioned in section 1.3, the authors first distinguish between the *domain graph* and the *data graph* before defining *graph interpretations* constituted by a *mapping* and a domain graph. They then define a subset of graph interpretations that are *graph models*. The

domain and data graphs follow the same graph data model, here the DELG model. Let us first introduce the formal definitions for *graph interpretation* and *graph model*. Recall that **Con** denotes a countably infinite set of constants.

Definition 1.3.1 (Graph interpretation)**.** A graph interpretation – or simply interpretation – captures the assumptions under which the semantics of a graph can be defined. A (graph) interpretation I is defined as a pair $\mathrm{I} := (\Gamma, \cdot^{\mathrm{I}})$ where $\Gamma = (\mathrm{V}_\Gamma, \mathrm{E}_\Gamma, \mathrm{L}_\Gamma)$ is a (directed-edge-labelled) graph called the domain graph and $\cdot^{\mathrm{I}} : \mathbf{Con} \to \mathrm{V}_\Gamma \cup \mathrm{L}_\Gamma$ is a partial mapping from constants (in the data graph) to terms in the domain graph [HBC+21].

Let us consider graph illustrated by figure 1.3 as our data graph following the DELG model. We can then define the following domain graph Γ which we describe with the following DLs axiom:

$$\mathrm{M}argherita \sqsubseteq \forall has\mathrm{T}opping.(\mathrm{T}omato \sqcup \mathrm{M}ozzarella).$$

We then consider the RDF graph corresponding to this axiom expressed with the OWL vocabulary as our domain graph. We then have $\Gamma = (\mathrm{V}_\Gamma, \mathrm{E}_\Gamma, \mathrm{L}_\Gamma)$ with[14]:

- $\mathrm{V}_\Gamma = \{\mathrm{M}argherita, \mathrm{T}omato, \mathrm{M}ozzarella, has\mathrm{T}opping, owl:\mathrm{C}lass,$ $owl:\mathrm{O}bject\mathrm{P}roperty, \_:b, owl:\mathrm{R}estriction, \ldots\}$
- $\mathrm{E}_\Gamma = \{(\mathrm{M}argherita, rdf:type, owl:\mathrm{C}lass), (\mathrm{M}argherita, rdfs:sub\mathrm{C}lass\mathrm{O}f\_:b),$ $(\_:b, rdf:type, owl:\mathrm{R}estriction), (\_:b, owl:on\mathrm{P}roperty, has\mathrm{T}opping), \ldots\}$
- $\mathrm{L}_\Gamma = \{rdf:type, rdfs:sub\mathrm{C}lass\mathrm{O}f, owl:on\mathrm{P}roperty, owl:allValues\mathrm{F}rom,$ $owl:union\mathrm{O}f\}$

The partial mapping $\cdot^{\mathrm{I}}$ then aligns $\_margherita$, $\_mozzarella$, $\_tomato$ and $has\mathrm{T}opping$ in the data graph (figure 1.3) with $\mathrm{M}argherita$, $\mathrm{T}omato$, $\mathrm{M}ozzarella$ and $has\mathrm{T}opping$, in the domain graph respectively. In practice such mapping could take the form of the following set of RDF triples: $\{(\_margherita, rdf:type, \mathrm{M}argherita), (\_tomato, rdf:type, \mathrm{T}omato)\}$.

An interpretation can be valid and can satisfy a data graph or not. Some assumptions at the root of an ontology language drive the validity and satisfaction of a data graph. We already discussed the main assumptions in section 1.3, i.e., the UNA vs. NUNA and the CWA vs. OWA. OWL adopts the NUNA and OWA, which is the most general case: multiple nodes/edges in the data graph may refer to the same entity/relation in the domain graph (per the NUNA), and anything not deduced based on an interpretation of the data graph is not assumed to be false as a consequence (per the OWA) [HBC+21].

We denote the domain of the mapping $\cdot^{\mathrm{I}}$ by $dom(\cdot^{\mathrm{I}})$. For interpretations under the UNA, the mapping $\cdot^{\mathrm{I}}$ is required to be injective, while with No UNA (NUNA), no such requirement is necessary. Interpretations that satisfy a graph are then said to be models of that graph.

Definition 1.3.2 (Graph models)**.** Let $\mathrm{G} := (\mathrm{V}, \mathrm{E}, \mathrm{L})$ be a directed edge-labelled graph. An interpretation $\mathrm{I} := (\Gamma, \cdot^{\mathrm{I}})$ satisfies G if and only if the following hold:

- $\mathrm{V} \cup \mathrm{L} \subseteq dom(\cdot^{\mathrm{I}})$;
- for all $v \in \mathrm{V}$, it holds that $v^{\mathrm{I}} \in \mathrm{V}_\Gamma$;
- for all $l \in \mathrm{L}$, it holds that $e^{\mathrm{I}} \in \mathrm{L}_\Gamma$; and
- for all $(u, l, v) \in \mathrm{E}$, it holds that $(u^{\mathrm{I}}, l^{\mathrm{I}}, v^{\mathrm{I}}) \in \mathrm{E}_\Gamma$.

If I satisfies G we call I a (graph) model of G [HBC+21].

[14] For clarity we only write down the main elements of the sets and use the three dots (…) to signify there is more.

To make a clear distinction between indirect references to the domain and data graph, we will use the terms *nodes* and *edges* when referring to elements in the data graph, and the terms *entities* and *relations* when referring to elements in the domain graph. The mapping introduced in 1.3.1 is an alignment between the terms in the data graph, i.e., nodes and edges, and those in the domain graph, i.e., entities and relations.

Notice that thus far, the definitions are generic and focus on the terms' meaning in a graph representation context. Defining such notions becomes useful when we need to define in practice the meaning of entailment, aka reasoning, under semantic conditions that characterise the features of an ontology language. When designing an ontology, we define axioms which enforce some conditions restricting the possible models for a graph. We called such conditions *semantic conditions*.

Definition 1.3.3 (Models under semantic conditions)**.** Let 2^G denote the set of all (directed edge-labelled) graphs. A semantic condition is a mapping $\phi : 2^G \rightarrow \{true, false\}$. An interpretation $I := (\Gamma, \cdot^I)$ is a model of G under ϕ if and only if I is a model of G and $\phi(\Gamma)$ is true. Given a set of semantic conditions Φ, we say that I is a model of G under Φ if and only if I is a model of G and for all $\phi \in \Phi, \phi(\Gamma)$ is true [HBC+21].

In the particular case of OWL, the set of semantic conditions Φ is the set of ontology features defined in the standard [MPSG+12] form the fundamental axioms to built on when modelling in OWL, e.g., assertion, domain and range, or class equivalence.

Definition 1.3.4 (Ontology language feature)**.** Ontology language features are the minimum set of axioms defined by an ontology language.

Ontology language features are the most basic building blocks to construct the axioms forming an ontology. Hence, they also form the minimum set of semantic conditions to build upon when designing an ontology. Concentrating again on OWL, the W3C defines various standards related to OWL [Gro22]. The formal meaning of DLs axioms is given by their semantics[Kee20]. The direct model-theoretic semantics for OWL is defined in [MPSG+12].

The power of formal languages for knowledge representation resides in automating reasoning over such knowledge, i.e., deriving new facts by logical deduction and spotting logical inconsistencies. In practice, the reasoning process can be performed in two ways. Either they are using predefined rules which are applied on the data graph or they leverage the axioms defined in the domain graph. The latter approach comes down to defining whether or not a graph entails another one. Reasoning algorithms try to determine whether the data graph expanded with new facts is still a model of the original data graph under some semantic conditions. We now define graph entailment, rules and their application over a graph.

Remember that **Con** denotes a countably infinite set of constants. For these definitions we introduce **Var**, a countably infinite set of variables ranging over (but disjoint from: $\mathbf{Con} \cap \mathbf{Var} = \emptyset$) the set of constants. We also refer generically to constants and variables as terms, denoted and defined as $Term = \mathbf{Con} \cup \mathbf{Var}$. **Var**(Q) denotes the variables appearing in Q.

Definition 1.3.5 (Graph entailment)**.** Letting G_1 and G_2 denote two (directed edge-labelled) graphs, and Φ a set of semantic conditions, we say that G_1 entails G_2 under Φ – denoted $G_1 \models_\Phi G_2$ – if and only if any model of G_1 under Φ is also a model of G_2 under Φ [HBC+21].

Definition 1.3.6 (Graph pattern (DELG))**.** A graph pattern is a tuple $Q = (V, E, L)$, where $V \subseteq Term$ is a set of node terms, $L \subseteq Term$ is a set of edge terms, and $E \subseteq V \times L \times V$ is a set of edges (triple patterns) [HBC+21].

Definition 1.3.7 (Rule)**.** A rule is a pair $R := (B, H)$ such that B and H are graph patterns and $\mathbf{Var}(H) \subseteq B$. The graph pattern B is called the body of the rule, while H is called the head of the rule [HBC+21].

Definition 1.3.8 (Rule application)**.** Given a rule $R := (B, H)$ and a graph G, we define the application of R over G as the graph $R(G) := \bigcup_{\mu \in B(G)} \mu(H)$ [HBC+21].

To apply reasoning in practice, the research community has developed various reasoning algorithms targeted at ontologies with certain peculiarities. Most tractable algorithms use rules and/or DL. To balance expressivity and algorithmic complexity of reasoning, some algorithms restrict the use of the DLs ontology language features. Such restrictions on the ontology language features define fragments of DL, which are standardised in different W3C standardised OWL profiles [MGH+12] targeted at different reasoning applications [HBC+21]. The W3C also standardise a language to define rules, The SW Rule Language (SWRL) [HPSB+04].

A closely related term often mentioned in KG and Ontology definitions is *Knowledge Base* (KB). KBs have been extensively researched since at least the 1980s. KGs are a logical descendant of those systems. In particular, older works on ontologies expressed using DLs and implemented in OWL often mention KBs. Notice the lack of reference to the graph structure. Indeed, a KB does not enforce such a graph structure. Yet, for OWL KBs, a standard mapping between OWL axioms and their RDF graph representation exists[PSMG+12]. The same holds for SWRL. Hence, the previous definitions apply to KBs defined with such languages.

Definition 1.3.9 (Knowledge Base (KB))**.** A Knowledge Base is a set of axioms and/or rules representing some domain knowledge, and the real world entities composing it.

In [SKA+22], the authors distinguish a KG from a KB based on their T-Box and A-box. KBs are more traditional systems focusing on defining a large T-box. Such systems are also more curated and domain-specific. In contrast, more recent systems with KGs focus on large A-boxes with relatively small and less complex T-boxes. In the latter work of Simsek et al., the authors' notion of an A-box corresponds to our KG data graph. However, in this work, our notion of A-box is tied to the ontology part of the KG. Hence, the A-box corresponds to the entities and relations in our domain graph. The literature, particularly the one focusing on applications of KGs, does not distinguish between the domain and data graph. Hence, these works consider the A-box the data and the T-box the ontology.

1.4 Knowledge Graph Construction and Ontology Learning

Constructing ontologies, and more generically KGs, has always been a challenge. The process is known as knowledge or ontology engineering [EFK19]. Since at least the early 2000s, research communities have explored the automation of knowledge engineering tasks. For ontologies, the processes automating their construction are known as *Ontology Learning* (OL). A closely related term is *ontology population*. For KGs, in recent years, terms such as *KG Construction* (KGC) or *automatic KGC* have emerged. In this work, the acronym KGC is not to be confused with the related term *KG completion*, which refers to a task often part of a KGC process, focusing on completing an existing KG with relations and entities. The latter KG completion task is out of the scope of this manuscript.

Our work concentrates on OL from text sources. Hence, this section introduces OL and related terms and concepts found in the literature. We first explore related tasks and how they differ from OL. We then review existing approaches to OL.

1.4.1 Related tasks

In the scientific literature focusing on automating KG construction processes, we encounter terms such as Knowledge and ontology engineering, automatic KGC, OL and ontology population. Here, we use the term KGC to refer to the tasks aimed at automating KGC tasks, aka automatic KGC. Naturally, KGC focuses on automating tasks to construct a KG, and OL refers to tasks to automate the ontology engineering process. However, though their techniques have many tasks in common, they also differ, particularly in their objective and base assumptions. Let us first discuss the broader tasks of knowledge and ontology engineering.

The term knowledge engineering, similar to the term ontology engineering, dates back to the early days of computers and software engineering. In the 1990s, the first computer processable knowledge models demonstrated their applications in expert systems, spiking interest. At the time, the software engineering field was widely explored. Hence, the scientific communities around knowledge modelling naturally worked on adapting software engineering practices to knowledge modelling. In 1998, Studer et al. defined the goal of the new discipline of Knowledge Engineering (KE) as similar to that of Software Engineering, i.e., turning the process of constructing knowledge base systems from an art into an engineering discipline. This endeavour required analysing the building and maintenance process and designing appropriate methods, languages, and tools specialised for developing such knowledge base systems [SBF98]. Formal knowledge modelling was and is still tied to Logic with logic-based ontology languages such as OWL. Hence, in 1999, John Sowa defined knowledge engineering as the application of Logic and ontology to the task of building computable models of some domain for some purpose [EFK19, Sow99]. More recently, Aussenac-Gilles et al. broadly refer to knowledge engineering as all technical, scientific and social aspects involved in designing, maintaining and using knowledge-based systems [AGCR20].

Ontology Engineering is closely related to knowledge engineering. Both terms are used interchangeably. In [GLÖ09], Gal et al. define ontology engineering as the set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies. Books focusing specifically on ontology engineering, e.g., [Kee20] and [EFK19], define it similarly. However, the term ontology is often tied to computer-processable models, while knowledge engineering tends to be used in a broader sense, including traditional methods that do not necessarily lead to a computer-processable artefact. However, both are traditionally tied to logic-based formal definition of terms and their meaning. They refer to attempts to adapt software engineering methods to knowledge modelling.

KGC is defined as the process of populating a KG with new knowledge elements (e.g., entities, relations, events) [YZCC22]. The authors of [ZWL+23] propose to view KGC as a mapping procedure that maps a data source into a KG. They also point out that KGC usually can only continue with background knowledge provided by pre-designed rules or a language model of representations. Due KGs related literature focusing on large graphs[SKA+22], KGs are often built from unstructured or semi-structured content such as web pages. Hence, all KGC approaches are based on some NLP tasks. The tasks explored in KGC surveys as part of the KGC process assume some background knowledge, such as the kind of entities expected in the KG, e.g., persons or organisations. Other tasks focus on enriching an existing KG. For instance, both surveys [YZCC22] and [ZWL+23] detail NLP tasks such as Named Entity Recognition, Entity Typing, Entity Linking, Event extraction, and Relation extraction. We further discuss these tasks in chapter 2 (section 2.2.4). Nevertheless, these tasks assume a predefined list of types, entities, or kinds of events.

The KGC surveys also mention KG completion, which naturally assumes the existence of a KG. As KGs tend to focus on large amounts of entities and relations rather than on their formal definitions, the KGC research generally do not detail the formal definitions of the entities and relations. They define them through their graph data structure in an inductive fashion, i.e., based on the entities' relations. Notably, KGC surveys often consider the KG lifecycle, i.e., the KG evolution and maintenance.

OL is very similar to KGC. The techniques focus on automating as much as possible knowledge acquisition processes [AWK+18, KAG21]. However, we point out two differences. First, OL techniques do not assume any knowledge or entities that are known beforehand. OL methods include term and then concept extraction, which aim to automatically determine the terms of interest and then define concepts based on them. Second, OL techniques explore axiom extraction tasks, which aim at inferring rules and axioms to describe the concepts and relations. OL methods study the automation of the end-to-end ontology engineering process. In this PhD work, we propose an approach to OL from text. Hence, the following section further discusses OL from text.

Closely related to OL, the task of ontology population aims at learning instances of concepts as well as relations [Cim06]. As such, this task assumes the existence of an ontology. Before reviewing OL from text approaches, we introduce the OL layer cake model.

1.4.2 Ontology Learning layer cake

Cimiano introduces the ontology learning layer cake [Cim06] to organise the various tasks involved in OL. Even though the 8 layers convey a sense of order, it is only sometimes the case. The different layers of the architecture are presented below in the order they [Cim06] and figure 1.6 introduce them:

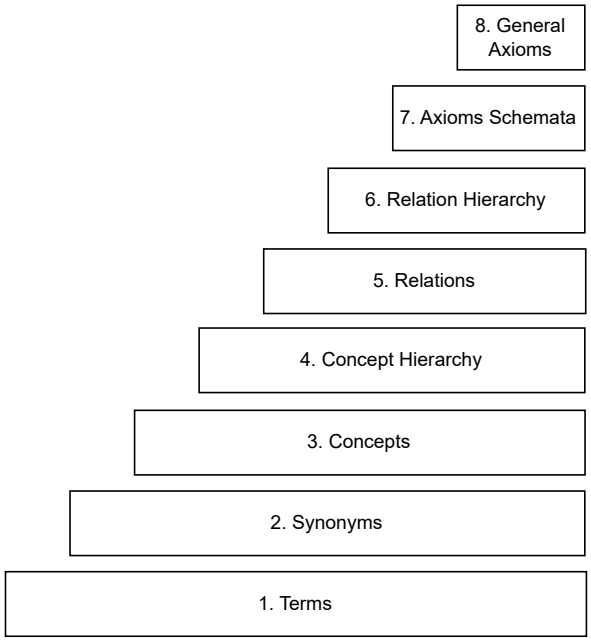


Figure 1.6: Ontology Learning layer cake.

1. Term extraction methods focus on extracting linguistic realisations of domain-specific concepts.
2. The synonym extraction layer aims at acquiring semantic term variants with sense disambiguation and domain-specific synonym identification.
3. Concept extraction processes include extracting informal definitions, that is, the text description of the concept from terms and synonyms.
4. The concept hierarchisation step finds taxonomic relations between concepts.
5. The relation extraction processes focus on discovering non-taxonomic relations.
6. Relation hierarchisation operations intend to order relations potentially linked to each other.
7. Axiom schemata techniques identify generic rules among concepts and relations.
8. General axioms learning processes identify more complex relationships and connections to obtain logical implications.

Most scientific publications dealing with OL apply the OL layer cake model. However, in practice, only some of the tasks are performed. Benchmarks [AWK+18] also use this model to compare OL techniques based on the different sections of the layer cake and the kinds of approaches, i.e. linguistic, statistical or logical. Eventually, we have to point out that the result obtained on the

whole pipeline depends on the ones obtained at each intermediate step of the process and the combination of steps chosen.

1.4.3 Ontology Learning from text

This manuscript OL chapter concentrates on techniques based on raw text sources, i.e., an unstructured source of knowledge. These methods are called ontology learning from text and can be seen as a reverse engineering task [Cim06]. OL from text encompasses multiple sub-tasks. Our work further restricts itself to OL techniques, comprising a complete pipeline from raw text to structured knowledge representation. We leave aside methods focusing on a specific part of the OL process. As most knowledge is embedded in the vast amount of available text content, many automatic knowledge extraction pipelines include and rely on NLP techniques. Other components are external knowledge sources, e.g., Wikidata [VK14], ConceptNet [SCH16], or existing standard and company internal classifications.

Text2Onto [CV05] is a well-known framework for OL. Rather than committing to a particular technology, the approach represents the ontology structure at a metalevel by defining modelling primitives. The Text2Onto system also introduces the idea of *Probabilistic Ontology Models* to consider uncertainty while learning ontology. At the same time, it enables the integration of various modules. The result can be translated into multiple ontology representation languages. A significant advantage of this framework is its graphical interface for users to visualise the knowledge model. Text2Onto is implemented as a plugin for the NeOn project[15] based on GATE [CMBT02]. Unfortunately, the Text2Onto implementation is not maintained anymore.

To our knowledge, Text2Onto is the most flexible system, but other methods have been developed using the principle of division by task with different algorithms. They are not frameworks but specific pipelines. For example, OntoLearn [NVG03] and OntoGain [DZP10] are two statistical-based methods. The first one is primarily based on the WordNet [Mil95] linguistic resource, which is used for terminology extraction and semantic interpretation. The result is a specialised view of WordNet. The second deals with multi-word terms and compares taxonomy construction algorithms and non-taxonomic relation acquisition strategies on two corpora. In [VMUT17, JT10, KDT+21], different techniques are used based on an iterative focused data crawler, a concept-relation-concept tuple extraction and topic modelling, respectively. LExO [VHH08] focuses on expressive ontologies suitable for reasoning. The authors use syntactic and statistical classification methods to identify axioms and evaluate their consistency.

Some approaches restrict themselves to certain relations, algorithms, or tools. For example, ASIUM [FN98] implements a bottom-up clustering approach. Likewise for Mo'K [BNCn00], which additionally provides multiple parameters to tune. OntoLT [BOS04] and SPRAT [MFP09] are tool-specific implementations of methods leveraging patterns and mapping rules. FRED[16] [DGPN13] considers explicitly the use case of converting text into an RDF/OWL ontology within the context of Linked Data[17]. OntoCmaps [ZGH11] differs from other methods as it uses graph theory and provides a visualisation tool.

The majority of OL methods have much in common. The overall process is broken down into several small steps that are often identical from one method to another. For each step, algorithms are identified as more or less relevant and thus popular. The choice of the algorithms largely depends on the corpus and the use case.

1.4.4 Evaluation

Before concluding this chapter, let us briefly discuss methods to evaluate the relevance of an ontology automatically learned from text. Ontology evaluation uses metrics to assess the ontology produced against the intended knowledge model. The result of the evaluation implicitly describes

[15] http://neon-toolkit.org/wiki/Main_Page.html

[16] http://wit.istc.cnr.it/stlab-tools/fred/ (Accessed on Thursday 3rd October, 2024)

[17] https://www.w3.org/wiki/LinkedData (Accessed on Thursday 3rd October, 2024)

the produced ontology [EFK19]. As there are many possible OL techniques, it is essential to have comparison criteria for the knowledge representations learned. It enables us to choose the optimal technique and verify the content of the learned ontology.

The literature describes four evaluation techniques [Wat20, HEBR13, AWK+18]. The golden standard-based evaluation assesses and compares the learned ontology through a reference one. The application-based evaluation is task-oriented, exploiting an ontology explicitly built for an application to perform some task. This evaluation technique assesses the ontology quality based on the application performance, independently of ontology structural properties. The data-driven or corpus-based evaluation uses domain-specific knowledge sources to measure the built ontology coverage of a particular domain. Finally, the human or criteria-based one defines some qualitative or quantitative indicators and assesses the ontology against each to compute numerical scores. The literature uses a set of usual metrics regardless of the evaluation type. In [KAG21], Khadir et al. summarise them as consistency, completeness, conciseness, expandability, and sensitiveness. Each of the four above-described evaluation methods is more or less appropriate for assessing each criterion.

A domain expert makes the evaluation in [NVG03] and most methods studied in [Wat20]. In [DZP10], OntoGain is evaluated using different evaluation techniques, whereas OntoCmaps [ZGH11] evaluation relies on golden standard assessment. The authors implement a data-driven evaluation and a comparison with hand-crafted ontologies. In [VMUT17], authors rely on criteria-based evaluation. Another critical aspect of OL is process automation. Automation of OL tasks can be understood as the degree of human-in-the-loop, i.e., how much human intervention is needed. Several metrics could be used to quantify the latter. We can count the number of steps requiring significant human intervention. The type of human intervention can be qualified to differentiate between long and tedious steps, e.g., labelling, and more straightforward steps, such as verification. Among methods presented in section 1.4.3, the ones presented in [CV05, DZP10, VMUT17, JT10, KDT+21] do not include any task requiring a human implication, although [JT10] and [KDT+21] suggest a manual evaluation of the results. It is also the case in [MFP09], where one step involving a human is needed to choose the relations considered and tag them in the dataset.

1.5 Conclusion

In line with other researchers [EW16, BW23, HBC+21], we argue that KGs are a conceptual framework or a way of approaching knowledge and data modelling that takes various forms depending on the use cases and implementations, rather than a particular technology. In academia, a KG is often introduced as a graph data structure expressing some domain knowledge. Some formal mathematical representations are also provided, e.g., in [HBC+21]. In industry, KGs are often seen as a technology in itself [BW23], referring to the graph data structure and the system to interact with it, e.g., a graph database.

In [BW23], the authors refer to a set of patterns and practices called knowledge graphs emerging to help understand data in context, representing the context as a graph of connected data items. They further add that KGs are agnostic about the physical storage of the underlying data. In [EW16], the authors explicitly state *KG bears more resemblance to an abstract framework than to a mathematical structure.*

As this chapter shows, when discussing KGs and related applications, many similar terms are employed with different meanings referencing very different implementations and applications. Here, we did not intend to redefine those terms but rather clarify what we mean when using each term in the context of our work. In the previous sections, we already defined most of them. However, some terms discussed within paragraphs still need to be introduced appropriately as definitions. Below, we complete our definitions of the terms we use in this manuscript.

We construct our own definitions for KG and ontology using an adapted combination of definitions in [HBC+21].

Definition 1.5.1 (Knowledge Graph)**.** A Knowledge Graph (KG) is a graph intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities. The graph of data (aka data graph) conforms to a graph-based data model. Knowledge refers to something that is known. A KG is a data graph potentially enhanced with representations of schema, context, ontologies and/or rules forming the domain graph. Hence, the KG is composed of a data and domain graph.

Definition 1.5.2 (Data graph)**.** The data graph is the KG component representing the raw data. It organises the world in a graph structure but does not represent knowledge per se. In the data graph, we speak about nodes and edges.

Definition 1.5.3 (Domain graph)**.** The domain graph is the KG component representing the knowledge per se. The knowledge in the domain graph can be expressed following various forms (e.g., rules and axioms) and vocabularies (e.g., OWL and SWRL), which all have a graph representation. We speak about entities and relations in the domain graph to make a clear distinction from the data graph. The domain and data graph follow the same graph data model. The terms in the data graph are also present in the domain graph. In a KG, a mapping $\cdot^{\mathrm{I}}$ exists between the terms in the data graph and the ones in the domain graph.

In this work, we implement KGs using RDF, hence our graph data model is the DELG model. And in practice, in a KG, the domain graph is a model of the data graph.

Definition 1.5.4 (Ontology)**.** In computing, an ontology is a concrete, formal representation of what terms mean within their scope (e.g., a given domain).

In our work, we use the OWL 2 standard to implement ontologies. We also consider potential inference rules part of the ontology and leverage SWRL to implement them if needed.

Definition 1.5.5 (Deductive reasoning)**.** Deductive reasoning is the process deriving new data based on the existing data taken as premises and knowledge explicitly expressed a priori. The latter knowledge takes the form of rules and axioms (i.e., general rules)[HBC+21].

Definition 1.5.6 (Inductive reasoning)**.** In opposition to deductive reasoning, inductive reasoning is the process of deriving new facts based on patterns generalised from data. The new facts are potentially imprecise predictions and often come along with confidence scores.

Definition 1.5.7 (Semantic validation)**.** Semantic validation checks the logical coherence of the data graph. The process leverages the axioms defined in the domain graph and determines whether the domain graph logically entails the data graph.

The semantic validation process is typically carried out after applying the inference rules. Its result depends on the assumptions at the root of the ontology language, here OWL, hence the OWA and NUNA.

Definition 1.5.8 (Structural validation)**.** Structural validation is restricted to verifying the data graph structure. It leverages the structural schema (aka validating schema) part of the domain graph and is typically carried out after applying the inference rules. The result does not say anything about the KG Semantics.

Our work considers the structural validation as future work. However, following the SW standards such future works would explore the Shapes Constraint Language (SHACL) [KK17] to model the validating schema.

Figure 1.7 summaries this chapter. Round boxes denote components, while squared ones denote processes. Bold names in round boxes are the main components' labels. Some alternative labels are added below and in parentheses. The boxes' imbrications denote composition relations. The round box labelled *Mapping* straddles the *Domain graph* and *Data graph* boxes denoting its

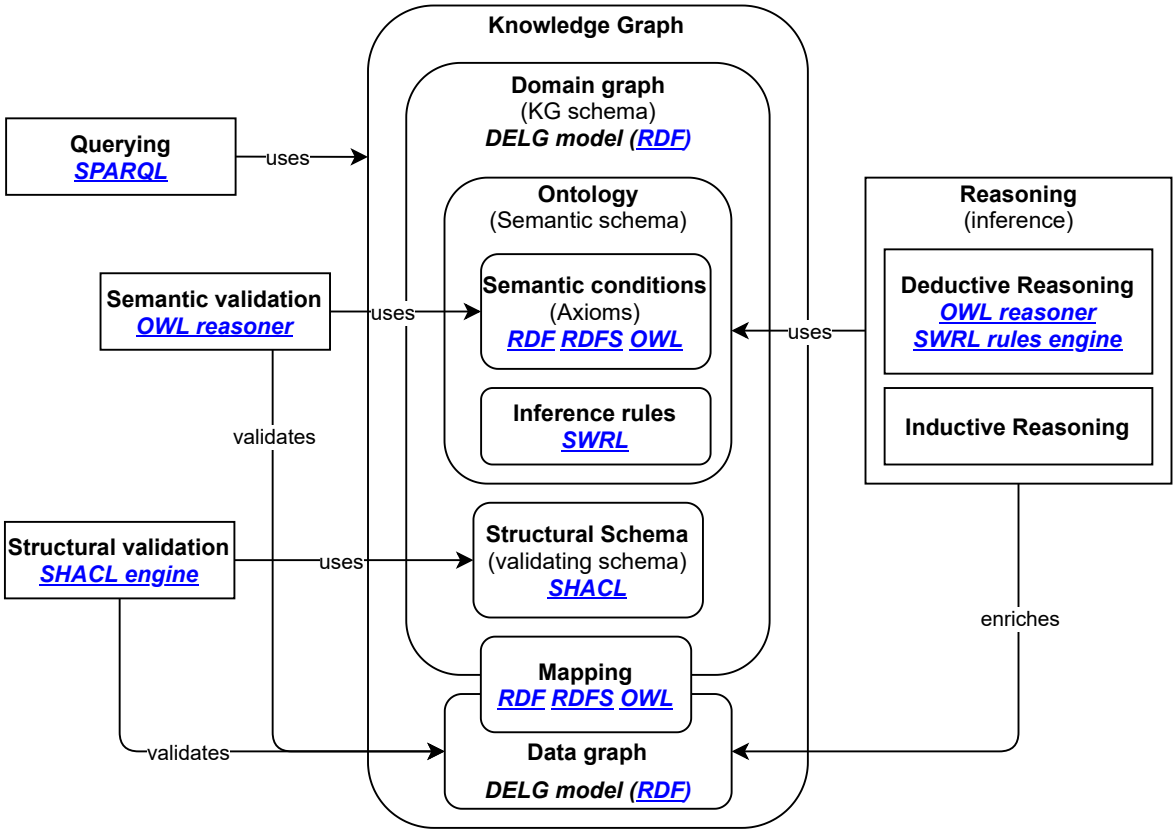


Figure 1.7: Knowledge Graph definition with Semantic Web candidate technologies.

mapping role. Finally, some SW candidate technologies for implementing each component are mentioned in blue, bold italics, and underlined letters.

Before concluding this chapter, we also discuss approaches to automatically construct a KG from semi-structured or unstructured text. In particular, our work includes an OL framework. In the literature, while KGC methods tend to focus on the KG data graph, OL approaches consider the extraction of rules and axioms, therefore contributing to the domain graph.

Chapter 2

Information retrieval

" If humanity were able to obtain the "privilege of forgetting the manifold things he does not need to have immediately at hand, with some assurance that he can find them again if proven important" only then "will mathematics be practically effective in bringing the growing knowledge of atomistic to the useful solution of the advanced problems of chemistry, metallurgy, and biology". "

Wardrip-Fruin, Noah; Montfort, Nick (2003). The New Media Reader

Contents

Information Retrieval (IR) is the process of retrieving information best suiting a user's need from a corpus of documents. Nowadays, the term *information retrieval* denotes the computing notion. Yet, the process dates back way before the invention of computers and takes its roots in library science [MRS08].

After exploring and defining KGs, the present chapter explores IR. Looking for information in documents is a task humans perform since written documents exist. The number of available documents and the technology used to search for information has changed over the years. The latter technologies are also partly responsible for such information scale. Humanity keeps adding new data, information and knowledge [SC12]. It piles up faster than human information processing capabilities evolve. Retrieving past documents and information is becoming increasingly crucial for many businesses. So much so that many businesses base their offering on the sole task of retrieving information. These companies also focus on particular domains. Such specialisation highlights the scale of knowledge available nowadays and the increasingly technical and specialised information needs.

In this chapter, we introduce IR, providing a historical perspective and an overview of the landscape approaches. This first part lets us analyse the ideas' evolutions and the main approaches contemporary methods build on.

In the second part, we focus on IR systems using a KG to support parts of its tasks. Our KG definition separates the data graph from the domain graph. The domain graph is sometimes called an *ontology*. Both terms, *knowledge graphs* and *ontologies*, have existed in the scientific literature for many years with evolving meanings. Recent IR research mentions both terms, leading us to hypothesise about the convergence of scientific communities. Hence, our literature review explores and compares published methods along two axes: the literature using the term *knowledge graph* and the one using the term *ontology*.

2.1 Brief historical perspectives on information retrieval

To anchor the IR component of our KG-focused literature review into a historical perspective, we focus on two IR literature reviews [SC12, HP23] spanning from the first ideas of IR to 2012 and from 2013 to June 2022, respectively. This section outlines the main ideas leading to today's IR research.

The story of IR starts way before the first computers were even a thought in someone's mind. The IR research trying to move away from traditional library-based approaches dates back to the end of the 19th century when we saw the first patents describing mechanical and electrical devices for document retrieval based on punch cards and microfilms. In the 1930s, Vannevar Bush started its works, leading to the Memex proposal in 1945. Then, at the end of the 1940s and the beginning of the 1950s, the Univac was a machine searching for text references associated with subject codes. It was in the 1950s that the term *Information Retrieval* was first coined, and we see the first system able to match substrings in a text. The beginning of the 1950s was a forgotten period, where the question of whether IR should be done with mechanical and electrical machines or with computers was still being determined. However, in the late 50s and early 60s, computers were identified as the best candidate, and the question shifted from whether IR is practical on computers to how to optimise the indexing and retrieval of documents on computers. It is the beginning of IR as a scientific field. By this time, ideas also shifted from indexing based on predefined categories to indexing documents based on the words they contain. The late 1950s saw the first version of the boolean model and the idea of assigning scores to documents regarding a query and ranking them. Term frequency was identified as one efficient scoring feature.

In the 1960s emerged the first attempts to model documents and queries as sparse vectors of size the number of distinct words in the corpus. Relevance feedback-based experiments also appeared, introducing an iterative approach to IR harnessing the user by providing feedback on a first search results set to refine. Other directions explored were documents clustering based on their content, the statistical association of words and query expansion-based approaches to IR. The decade also witnessed the first commercial systems for IR targeted at the industry. However,

there were some striking examples of the need for more communication between industry and academia. One instance is the commercial systems still only implementing boolean model-based IR, although academic research consistently showed how ranked retrieval produced better results. This trend continued at least until the 1990s.

The famous Term Frequency (TF) and Inverse Document Frequency (IDF) ideas arose in the 1970s alongside works focusing on formalising IR approaches. The latter works, such as the one of Salton's group, lead to the Vector Space Model (VSM) [SWY75]. In the same decade, the first studies of probabilistic methods were explored. At the time, those foremost probabilistic approaches considered terms independent to simplify the calculations. By the decade's end, initial studies focused on including term dependency in existing approaches.

In the 1980s and till the mid-1990s, TF and IDF variations caught much attention. Incorporating TF into the probabilistic model led to the first versions of the BM25 scoring method, which is still used in many systems today. On the VSM side, latent space methods aiming at projecting the queries and documents in a smaller vector space, easing the similarity comparisons, were explored. Some well-known examples of the VSM advances are Latent Semantic Analysis and Indexing (LSA, LSI) [DDF$^+$90]. Meanwhile, on the language modelling side, only the idea of word stemming showed improvement. The Text REtrieval Conference (TREC) was founded at the beginning of the 1990s as an annual exercise where research groups gathered to build much larger test collections of documents. This research experiment made the IR community aware that different collections required different approaches. The existing weighting and ranking functions could have been better suited for such data size. The first learning to rank ideas focusing on ranking a pre-selected set of documents emerged. However, such methods became genuinely effective with the rise of the World Wide Web and the wealth of search training data it brought.

By the end of the 1990s, the Web had exploded in popularity. While its logs provided a wealth of example data, they also brought new problems as web contributors discovered how to manipulate the web page's content to push them to the top of the results. It led to the first link analysis-based methods, such as the PageRank algorithm [PBMW99], and searching in anchor text (e.g., titles). The 2000s saw a tighter collaboration between industry and academia. IR techniques were becoming increasingly complex, with many parameters to tune. Though IR methods were still viewed as unsupervised, in practice, the many parameters required adjustments based on examples. The first search personalisation methods were also explored as the community learned that users might use the same search query with different search intent.

In the past decade, conventional approaches have continuously been improved, and new ones now incorporate deep learning techniques enabled by advances in both hardware and algorithms. In particular, with the rise of web searches, results ranking has become an essential aspect of IR. Current IR is commonly viewed as a two-stage system: a retrieval and a ranking stage. The retrieval stage aims to select a ranked list of documents. The ranking stage leverages the initial list of documents and re-ranks them to meet the user's search intent better.

Conventional retrieval methods include query and document augmentation, the latter addressing the risk of query drift or overfitting encountered with query augmentation. The IR research community also considers lexical dependency models aiming at introducing the relations between terms in techniques considering them independent; topic modelling approaches automatically inducing topics in documents; multilingual ones modelling the IR problem as a statistical machine translation problem to overcome the vocabulary mismatch problem between user queries and documents. Sparse vector-based retrieval methods have also been explored to address the retrieval stage. These approaches enhance the vector space model by learning predefined feature weights, typically leveraging deep learning, or by directly learning sparse feature vectors for documents and queries. Naturally, learning dense vectors of different document elements and using various machine learning-based techniques has been explored, as well as combinations of the techniques mentioned.

Part of the research focuses on approaches that solely target the ranking task. The older learning-to-rank techniques have been continuously enhanced. New deep learning-based ranking

models have emerged, forming two main research areas. Representation-based models learn a query and document representations separately to feed a ranking function. Interaction-based models address the risk of missing crucial matching signals. They combine queries and document pairs as input to a model trained to identify essential interactions between them before feeding the ranking function.

In every approach, machine learning components have been incorporated. More recently, with the excellent results of transformer-based methods, techniques leveraging attention-based approaches [VSP+17] and pre-trained language representations have been explored.

2.2 Information Retrieval, setting the stage

The scientific literature on IR can be broken down into various ways. We can consider the different approaches, the various components each method focuses on, the IR objective, or even the kind of documents forming the search space and its size. This section introduces the terms we will usedistinguishing each IR system's objectives, components, and tasks. The following sections will focus on IR systems based explicitly on a KG.

Historically, from library science, IR aimed to find books and publications on library shelves. After finding a book, we could use IR techniques such as table of contents and lexicon to find a particular portion of the book with the content matching our information needs. With the advent of computers, historical processes were quickly automatised and enhanced. The field has evolved with technology, leading to different search needs and corpus kinds. Nowadays, we can search with an information need expressed as natural language text, structured through a query language, or even using an image or audio file. Examples of such IR systems are Google Search[1], SPARQL [HSP13], Google Lens[2], and Shazam[3]. The corpora comprise various documents such as text, images, or audio. The IR system response is an ordered set of documents, a natural language response, or a natural language document set summary. One must distinguish each IR system's objectives, components and tasks to understand our work on KG-based IR.

[MRS08] defines IR as "*finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)*". We commonly categorise documents based on their structure as structured, semi-structured, and unstructured. The boundaries between each category are subjective. However, some globally agreed examples of each category are raw texts such as news articles for unstructured documents, web pages for semi-structured ones, and relational database content for structured ones. As mentioned above, a document can take many forms in modern IR systems. In our work, we focus on text documents. We use the formal definition provided by the authors of [RMdR20] initially for document retrieval:

Definition 2.2.1 (Information Retrieval (IR))**.** Given a query q and a collection of documents D, score and rank each document $d \in$ D based on its relevance to q.

We now detail the manifold objectives of an IR system, specifying the notion of a document.

2.2.1 Information retrieval systems' objectives

The IR literature is closely related to the Recommendation System (RS) one. RS are a particular kind of Information Filtering (IF) system and the most popular. IR systems focus on retrieving content most relevant to a user's need. In contrast, IF systems select items in a content stream that may interest a given user [Val21]. Drawing a clear line between IR and IF systems is often complex. A recent PhD thesis considers the relation between IR and IF, adapting IR methods to IF [Val21].

[1] https://www.google.com/
[2] https://lens.google/
[3] https://www.shazam.com/

RS aim to provide personalised content to users. Hence, such systems have two main elements: the users and the items [Val21]. One finds typical examples of RS in the E-commerce industry, where products are constantly suggested to users. The literature often classifies RS into two main categories. Content-based filtering RS suggest items to users, aligning the items' and users' profiles, i.e., attributes representing them. Collaborative filtering RS rely on feedback from other users' recommendations to make personalised suggestions [Val21]. Naturally, a third category of RS combines both approaches to compensate for the limitation of the one with the other. In this thesis, we focus on IR systems. Hence, we redirect the interested reader to relevant literature, such as [Val21] or [GZQ+22], for further discussion on the IF and RS topics.

Many different but closely related tasks exist in IR. One can consider three main objectives shaping the tasks: document, passage, and entity retrieval [RMdR20]. Document retrieval aims to retrieve all documents relevant to a query in a corpus. Passage retrieval deals with retrieving relevant passages within the corpus' documents. Entity retrieval extracts relevant entities from corpus documents or directly from a KG. Each IR system objective has its task requirements. Some are common to all systems, and others are specific to an objective. Note that most objectives have their RS counterparts.

In the following subsections, we first introduce the IR system's components before discussing the related tasks.

2.2.2 Information retrieval system components

In an IR system, one can distinguish the following components, which we represent in figure 2.1 depicting IR systems main components:

- A corpus (and its index)
- A user need or intent
- A user query
- A retrieval model
- A ranking model

Some other components might be part of the system as well, depending on the use case and the chosen approach:

- A latent space
- A language model
- A KG (or, more generally, a structured external resource)

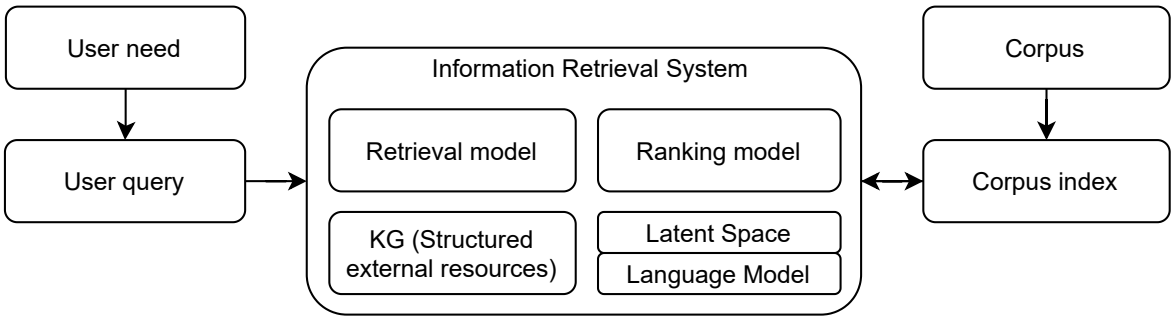


Figure 2.1: Information Retrieval system main components overview.

The corpus of documents constitutes the IR system search space. Here, we use the term *document* in the generic sense. A document comprises the content and all its metadata available, i.e., any other information about the document, such as a date. If the corpus comprises images, we call *document* an image and its related pieces of information, such as a date, a description, and an author. The corpus is the set of all the documents.

Definition 2.2.2 (Document)**.** A document noted $d \in \mathrm{D}$ is the unit the IR system is built over. It contains the IR system-targeted content as well as its associated metadata. The set of all documents constitutes the corpus, noted D, over which the retrieval task is performed.

Definition 2.2.3 (Corpus Index)**.** The corpus index is the data structure used to organise the corpus documents so they are easily accessible through specific criteria. The corpus index typically contains precomputed values, such as document term frequencies.

Though in figure 2.1 we distinguish between the corpus and its corresponding index, the term *corpus* is often used to reference the corpus index. In this work, we are using this misuse of language for simplicity.

Definition 2.2.4 (Query)**.** A query $q \in \mathrm{Q}$ is the user need's representation the IR system leverages to retrieve relevant documents.

Definition 2.2.5 (IR system search space)**.** The IR search space is the set of all possible items the IR system is built on and has to retrieve and rank.

Note that the IR system search space might differ slightly from the corpus in some cases. For instance, the corpus might contain text documents when performing passage retrieval. However, the search space comprises all the passages extracted from the documents' text over which the IR system is built.

Users can express their information needs to the IR system in various ways. There is a distinction between the user's need as it is in the user's mind and the query, representing the user's need in a format ready for the IR system to process. Typically, an IR system will try to pre-process the initial user query to better specify the user's need before running the document retrieval and ranking processes.

A retrieval model deals with selecting the corpus documents relevant to a user's need, i.e., with respect to a user query. Retrieval models are an active research area [Val21, HP23]. The simplest retrieval models, such as the boolean retrieval model, only select documents. They can be viewed as a function taking a document and a query as input and returning a boolean value *True* or *False*. Some others, sometimes called ranked retrieval models, also assign a relevance score to each document. The most well-known historical example is the vector space model [SWY75].

Definition 2.2.6 (Retrieval model)**.** A retrieval model is a scoring function $score : \mathrm{D} \times \mathrm{Q} \to \mathbb{R}$ that assigns a score s to a document d with respect to a user query q: $score(d, q) = s$.

Some approaches leverage extra components, such as a KG or a latent space. We already defined KGs in chapter 1. Let us discuss latent spaces before diving into the different retrieval models. Sometimes, it might be worth leveraging metadata from documents and queries to select so-called features from them. The sets of all possible feature value combinations form a latent space where we can represent documents and queries to ease the similarity measurement process. There are various algorithms to derive latent spaces. However, they all are vector spaces of reduced dimensionality intended to produce more general features that helpfully characterise the input [LJLH19]. In this induced vector space, items resembling each other are positioned closer to one another. In the literature, latent spaces are also known as *embedding space*. Note that sometimes the term *latent* is used as an adjective to denote something implicit, e.g., a latent concept as denoted in [MC07].

Definition 2.2.7 (Latent space)**.** A latent space is a vector space of reduced dimensionality intended to produce more general features that helpfully characterise the input [LJLH19]. In this induced vector space, items resembling each other are positioned closer to one another. The dimensionality of a latent space is typically lower than the feature space from which the data points are drawn.

Retrieval models

Among IR approaches, one can first distinguish two kinds of retrieval: ad hoc and relevance feedback-based. It is the first-level distinction depicted in figure 2.2, providing an overview of IR main approaches organised hierarchically. Ad hoc retrieval provides relevant documents from the corpus, leveraging the user query and the corpus documents' representations. In some situations, we can enhance the search results, i.e., the IR system retrieved documents, by leveraging user feedback on the document's relevance. The later feedback can come in 3 primary flavours. The user can provide explicit feedback by grading or selecting the relevant ones among the retrieved documents. Relevance feedback can be gathered implicitly from user behaviour, such as clicks or web page openings. The most researched relevance feedback is the so-called pseudo-relevance feedback. It automates the explicit user feedback by running an initial retrieval process, selecting the top k retrieved documents and considering them relevant. Further retrieval is then performed on this corpus subset of pseudo-relevant documents.

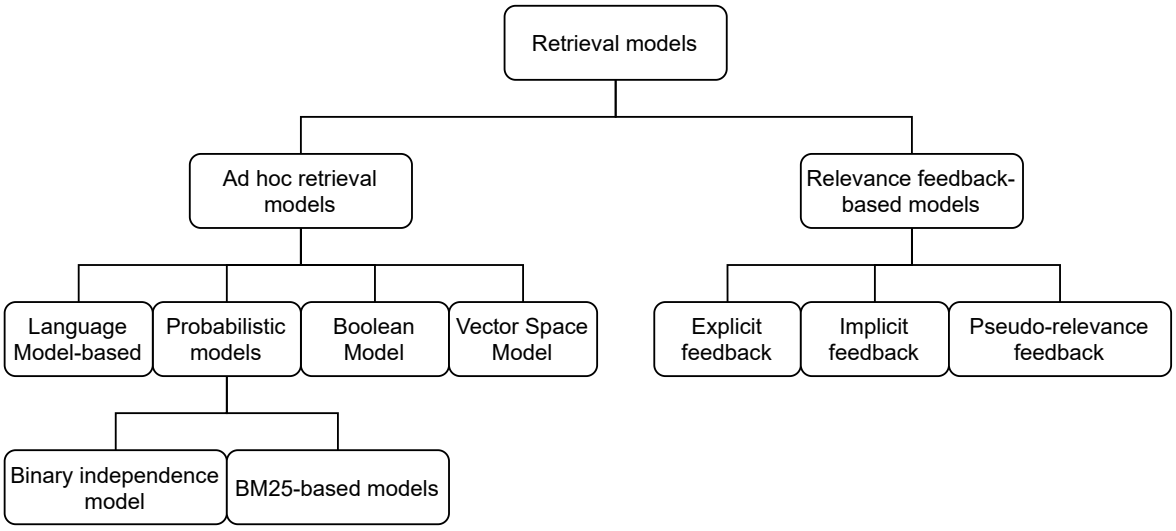


Figure 2.2: Information Retrieval main retrieval models.

Note that the term *retrieval* denotes assigning a score to a document based on its relevance to a query. Hence, retrieval encompasses both selecting and ranking. This point is essential as there is a distinction between retrieval models and a model focusing explicitly on ranking in the IR literature. Indeed, some approaches build ranking models specifically aiming at (re)ranking the initial retrieval results. On another note, document selection is sometimes called document filtering. It is two sides of the same coin depending on whether one sees the corpus as a set or a stream of documents.

Among Ad Hoc retrieval models, the most well-known ones are the boolean model and its extended version [SFW83]; the vector space model [SWY75]; some probabilistic models, such as the binary independence model and the Okapi BM25[RZ09]; and approaches based on language modelling. We now provide short descriptions of each method to better understand the literature review on KG-based IR systems. We also redirect the interested reader to the relevant publications for extensive details.

The Boolean retrieval model initially focused on text documents represented as sets of terms, i.e., a Bag Of Words (BOW). The user query comprises terms combined with operators AND, OR,

and NOT. However, the Boolean model does not consider term weights, leading to either too big or too small result sets. The extended boolean model [SFW83] overcomes this limitation, letting the user express its queries in natural language rather than enforcing the boolean expression. The IR system then decides which terms or conjunction of terms should appear in the retrieved documents based on partial matches, allowing the adjustment of the result set.

The Vector Space Model (VSM) [SWY75] represents the documents and queries as vectors of dimension equal to the number of terms in the corpus. Each vector dimension is assigned a weight, typically based on Term Frequency (TF) and Inverse Document Frequency (IDF). The retrieval task then comes down to computing a similarity between the query and document vectors, most often the cosine similarity measure. One can optimise the VSM by modifying the weights or the similarity measure.

Probabilistic retrieval models aim at deriving a document ranking function by estimating the probability that a document d is relevant to a query q, i.e., modelling the probability $\mathrm{P}(\mathrm{R}=1|q,d)$ where R is a binary variable denoting whether a document d is relevant to a query q. One well-established probabilistic model is the binary independence model, which makes some base assumptions so that estimating the query documents' relevance distribution becomes computationally tractable. In this model, documents and queries are binary vectors representing the absence or presence of terms. The latter terms are considered independently distributed among relevant and irrelevant documents. In many cases, the latter assumptions are considered limitations. Hence, some models try to overcome them. The most well-known one still in use in many systems such as Apache Lucene[4] and Elasticsearch[5] is the Okapi BM25 and its variants [RZ09].

BM25 [RZ09] (BM standing for Best Matching) is a weighting scheme which considers documents and queries as BOWs and is based on a combination of TF, IDF, normalising the score by BOW length, i.e., the document word count. The resulting retrieval function is best known as Okapi BM25, Okapi being the first IR system that implemented the approach at London's City University in the 1990s and 1980s[6]. Many variants modify the parameters and components. Among them, we can emphasise the BM25-F, which considers the document structure, assigning different weighting parameters to the document fields (the F standing for Field). The BM25's popularity comes from the relevance of its search results and the many scoring function components that can be computed offline, making it efficient at query time. One of the most used formulae is:

$$score(\mathrm{D},\mathrm{Q}) = \sum_{i=1}^{n} \mathrm{IDF}(q_i) \times \frac{f(q_i,\mathrm{D}) \times (k_1+1)}{f(q_i,\mathrm{D}) + k_1 \times \left(1 - b + b \times \frac{|\mathrm{D}|}{avgdl}\right)}$$

where $f(q_i,\mathrm{D})$ is the number of times that q_i occurs in the document D, $|\mathrm{D}|$ is the number of words in document D, and $avgdl$ is the average document length in the text collection from which documents are drawn. k_1 and b are free parameters to be optimised. $\mathrm{IDF}(q_i)$ is the inverse document frequency weight of the query term q_i. It is usually computed as

$$\mathrm{IDF}(q_i) = ln\left(\frac{\mathrm{N} - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

where N is the total number of documents in the collection, and $n(q_i)$ is the number of documents containing q_i[7].

Another common approach to retrieval is based on language modelling. A Language Model (LM) is a probability distribution over sequences of words [JM23]. The notion of LM is inherently probabilistic and builds upon the assumption that, in reality, almost no data are indeed "unstructured". It is true of all text data if you count the latent linguistic structure of human languages [MRS08]. The basic LM-based approach to IR builds a LM for each document and then ranks them based on how likely the document LM M_d is to generate the query q, i.e., scoring documents with

[4] https://lucene.apache.org/core/ (Accessed on Thursday 3rd October, 2024)

[5] https://www.elastic.co/ (Accessed on Thursday 3rd October, 2024)

[6] https://smcse.city.ac.uk/doc/cisr/web/okapi/okapi.html (Accessed on Thursday 3rd October, 2024)

[7] https://en.wikipedia.org/wiki/Okapi_BM25 (Accessed on Thursday 3rd October, 2024)

the value of $P(q|M_d)$. The most used approach to estimate the latter probability is using the query likelihood model, i.e., a Maximum Likelihood Estimation (MLE) approach [MRS08]. The most recent research leverages Large Language Models (LLM) and applies them to IR tasks [HP23].

Definition 2.2.8 (Language model)**.** A language model is a function that puts a probability measure over strings drawn from some vocabulary. That is, for a language model M over an alphabet Σ [JM23]:

$$\sum_{s \in \Sigma *} P(s) = 1$$

2.2.3 Information retrieval system tasks

Now that we have defined the main components of an IR system, we can introduce the main processes. Before diving in, we should keep in mind that we present a series of tasks that can be implemented partially. However, some tasks, such as document indexing or document retrieval, are required. Some others, such as query expansion, correspond to a peculiar approach and hence are only sometimes needed. Figure 2.3 introduces an overview of IR main tasks organised hierarchically.

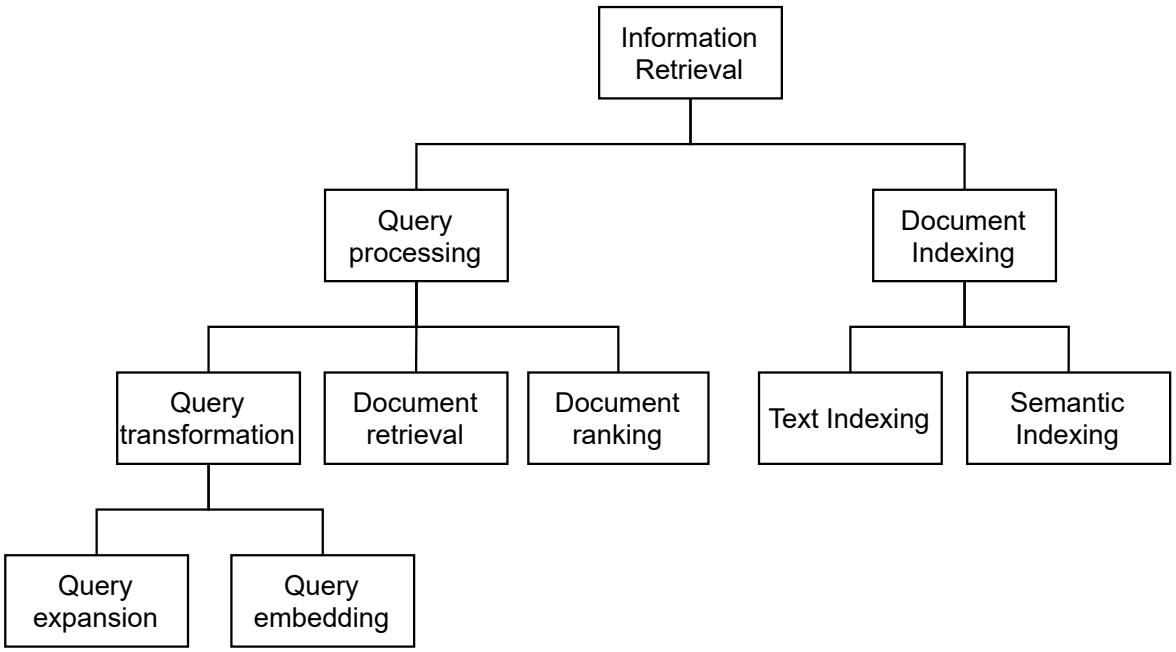


Figure 2.3: Information Retrieval main tasks.

We can consider the document indexing, query processing and retrieval processes at the most abstract level. Under these umbrellas, multiple subtasks exist. We will discuss the main ones in this section. The following section will introduce the minimum set of Natural Language Processing (NLP) concepts and tasks common to many IR processes we will explore.

Let us begin with document indexing. Documents, their content, and metadata need to be preprocessed and organised so that it will be easy to compute their relevance to a query. Such processes are called indexing. They encompass, among other things, extracting metrics about the content and adding metadata such as tags to it. Two common sorts of indexing are text and semantic indexing.

Text indexing corresponds to computing various metrics about the terms contained in the texts. Then, we organise the documents so they can easily be fetched based on the terms they contain and their metadata. Such indexing often results in an inverted index in which keys are the terms and values are the list of documents containing them. Standard term-related metrics are TF and IDF, but other exists. Generally, for an IR system to be efficient at query time, one favours

relevance scores based on as few query-dependent parameters as possible. All non-query dependent parameters can be precomputed and stored offline, making the query document relevance scores fast to compute at query time. Hence, corpus-only dependent metrics such as TF and IDF are convenient.

Semantic indexing can take various forms. It denotes any indexing based on more than just terms. One standard semantic indexing is building an inverted index like the one for text indexing but with entities from a KG as keys instead of terms. Such indexing requires first processing the text to tag entities, an NLP task named Entity Linking (EL) discussed in the next section.

Next, let us discuss the main query processing tasks. This term groups document retrieval and ranking, query expansion, and query embedding.

Document retrieval assigns a relevance score, typically greater than zero, to each document. The latter score enables the selection and ranking of documents based on their relevance to the query using a threshold value. Some approaches stop there. They either take the top k documents, filtering them based on a score threshold, which in its simplest case can be zero, or return all documents ordered by their score. Some other approaches use this first retrieval step as a document selection and build a model specifically tailored to re-rank this subset of selected documents. Building such ranking models is called *learning to rank* or *deep learning to rank* in the literature. This initial document selection is also used in pseudo-relevance feedback-based approaches in which the pre-selected documents are considered relevant and leveraged to enhance retrieval. In most cases, the document scores are used only to compare documents' relevance to the same query. They can not be used for comparison across different retrieval processes. The latter score scope is leveraged to ignore constants in the score and lower the computational cost [MRS08].

We can directly match the query with documents without transformation during query processing. We can also consider enhancing the query with extra information or translating it into another format in which the retrieval process is facilitated. The first approach is commonly called query expansion. It aims to enrich the user query with extra terms, entities, and any feature that better represents the user's need in the search context. This approach adjusts the query to best fit the corpus at hand, i.e., the goal is to align the user vocabulary with the corpus one. A classic example is adding query terms' synonyms so the IR system retrieves documents mentioning them.

An approach we mentioned in section 2.2.2 is leveraging another representation to ease the IR task. In such cases, one needs to construct a vector space called a latent space. In this latent space, similar items are close to one another. Vectors are easier to manipulate in the latent space than in the original document space. Such vector spaces can be used for various subtasks. We can use vectors to compute similarity scores directly or to retrieve related entities and expand the user query before the final document retrieval. The most elaborate approaches build different vector spaces tailored to each subtask. We denote this task *query embedding* in figure 2.3.

As we will see in the following sections, the information retrieval tasks presented here are adapted, combined and arranged in different manners to fit particular use cases. While we focus on textual content, those tasks have variants for any content.

2.2.4 Natural Language Processing

As our work focuses on text, we must introduce some necessary concepts and vocabulary from the Natural Language Processing (NLP) field we will employ. We give an overview and redirect the interested reader to [JM23] for a deeper dive into specific topics.

Almost all NLP pipelines begin with normalising the text. Text normalisation often encompasses at least three tasks: tokenisation, word normalisation, and sentence segmentation [JM23]. While normalising the word forms and segmenting the text into sentences or any other granularity level greatly depends on the kind of text or the global objective, tokenising the text is a mandatory task. Tokenising breaks down text into tokens, which is the smallest unit the NLP pipeline components will work with.

In our work, we distinguish tokens, words and terms. As mentioned, a token is the smallest unit of text the processing components will work with. A token can be anything ranging from

a single dot or any punctuation character to a subword or an entire word like "*elephant*". The most recent tokenisers, such as WordPiece [SSS+20] or SentencePiece [KR18], even consider word parts. A word is a language-dependent meaningful element composed of one or more tokens. "*knowledgeable*" is an example of word. Depending on the tokeniser, this word could be composed of a single token or two, e.g., *knowledge* and *##able*. Punctuation characters like a question mark (*?*) or a hyphen (-) are not words. "*knowledge graph*" is more than just a word. We consider terms an extension of words formed by one or multiple words. Hence, a term is a language-dependent unit composed of one or more words. "*elephant*" is a single-word term, and "*knowledge graph*" is a multi-word term. Categorising one or more words as a term comes with a mark of interest for further processing.

During text processing, various tasks assign a tag, i.e., categorise the tokens, words and terms. Different commonly used tasks assign different sorts of tags. Some examples are:

- Part Of Speech (POS) tagging assigns to words POS tags such as NOUN, DET, or VERB. The set of possible POS tags is typically derived from linguistic studies. The most well-known example is the universal POS tags[8].
- Named Entity Recognition (NER) tags terms, i.e., sequences of one or more words with predefined tags. The latter tags are called *named entities* when extracted from a predefined backed-by-linguistic-studies set of generic term classes, such as *person* or *organisaton*. The tags are called *entities of interest* when they are more subjective and context-dependent. Naturally, such entities of interest often encompass named entities.
- Entity Linking (EL) assigns an entity to a text sequence mentioned in a corpus. Such entities are typically extracted from a KG.

The IR problem we tackle in this work often implicitly depends on some of the mentioned NLP tasks. In particular, as we will discover in the following section, EL is critical. This task also implicitly contains a minimal sequence of NLP tasks applied to text beforehand, such as text tokenisation.

2.3 Knowledge Graph-based Information Retrieval literature review

In this section, we focus our study on the part of the IR literature that leverages a KG. Among the scientific publications on KG-based IR, we considered two main keywords for our paper selection. First, we study the methods described using the term *knowledge graph*. We extract these research works from Reinanda and colleagues' work [RMdR20] focusing on KGs for IR.

Second, we leveraged the scientific literature search engines of ScienceDirect [9], Google Scholar[10] and the ACM digital library [11]. Our paper selection process mainly focused on the ScienceDirect search engine. In a first pass, we searched for combinations of the terms *knowledge graph*, *ontology*, *information retrieval*, *search*, and *semantic* in the titles. To limit the number of publications, we concentrated on the ones treating the Architecture Engineering and Construction (A/E/C) domain. We selected articles from 3 journals: Computer in Industry, Automation in Construction, and Advanced Engineering Informatics. We also discarded the articles that did not consider a textual user query.

We below summarise our KG-based IR literature review studying methods presented using the term *knowledge graph* in section 2.3.1 and the ones described using the term *ontology* in section 2.3.2. We analyse the use of both terms. This chapter will refer to KG-based IR and ontology-based IR for readability reasons.

[8] https://universaldependencies.org/u/pos/ (Accessed on Thursday 3rd October, 2024)

[9] https://www.sciencedirect.com/ (Accessed on Thursday 3rd October, 2024)

[10] https://scholar.google.fr/ (Accessed on Thursday 3rd October, 2024)

[11] https://dl.acm.org/ (Accessed on Thursday 3rd October, 2024)

2.3.1 Knowledge Graph-based Information Retrieval

This section summarises our literature review of KG-based IR approaches introduced using the keyword *knowledge graph*. We first describe each method before discussing some limitations.

In 2014, Dalton et al. [DDA14] proposed an approach called Entity Query Feature Expansion (EQFE). They first index documents based on the entity they contain. The same EL is applied to queries. Feedback signals from an external KG, the corpus itself and a query-dependent entity context based on the initially retrieved documents are used to expand the query with terms. Finally, a learning-to-rank approach is exploited to weigh each feedback. EQFE is a reference approach as demonstrated by the multiple methods proposed in the following years for which the article's state-of-the-art review sections extensively describe EQFE [XC15a, XC15b, LF15, RMdR20].

In the year following EQFE, Xiong et al. compare query-dependent expansion-based approaches leveraging external KGs [XC15b]. They propose retrieving entities from a subset of corpus documents initially retrieved based on the user query and using a web search engine such as Google. Once entities are selected, terms are extracted based on them using scores such as TF-IDF. A supervised approach is also leveraged to mitigate the latter term expansion scores based on their sources. The method depends on some impractical resources for real-world business applications (for technical and financial reasons), such as the Google search engine or an already annotated dataset.

The same year, the same authors also introduced a general method EsdRank [XC15a] that considers features as objects they use to construct a latent space. The latter vector space is built by simultaneously learning the query-object and object-document mappings. The learning process leverages the ListMLE ranking model, introducing a novel approach, latent-ListMLE. The relevance scores are based on the MLE and can easily integrate usual relevance feedback.

In 2015, Liu X. and Fang H. proposed an approach based on a high-dimensional latent space as a bridge to map the query with documents [LF15]. They neither expand the query nor the documents but create entity profiles to project them in the latent space where each dimension corresponds to one entity. The document relevance function comprises one part focusing on the projections' similarity in the latent space and another directly estimating the document relevance to the query. In a supervised fashion, both parts are balanced by a parameter λ learned using a Support Vector Machine (SVM).

The following year, Raviv and colleagues introduced an approach leveraging an entity-based LM [RKC16]. They initially attempt to determine whether the markup of entities in a query and documents is sufficient information for improving retrieval effectiveness. With this purpose in mind, they define LMs based on document tokens' vocabulary enriched with entities considered tokens, including the entity linking uncertainty in the LM construction. They experiment with variations of an MLE-based LM. Their final ranking approach includes one part based on the corpus terms and the other based on the entities. Both parts are balanced with a parameter λ to be defined or learned. Though the authors evaluate their models for direct document retrieval, they demonstrate that the approach could be used effectively for query expansion or clustering-based IR.

Similarly, Ensan et al. construct an entity-based LM, SELM, incorporating the latter with a keyword-based one [EB17]. Their LM is constructed with a generative approach, i.e., optimising how likely the query entities will be generated knowing the document and corpus entities. The model also depends on a semantic analysis system to determine and include relations between entities. The final document query score is a weighted sum of the SELM and keyword scores. The parameters' weights are estimated using the EM algorithm.

More recent approaches incorporate deep learning into their systems. In 2017, Xiong and colleagues proposed the Explicit Semantic Ranking (ESR) approach [XPC17], constructing their KG to learn entity embeddings. They use the latter learned vectors to represent semantic relations between entities tagged in their queries and documents. Queries and documents are represented as bags of entities (BOEs). The entities' relations weights are computed using cosine distances between their embeddings. The resulting query document semantic matching information is sum-

marised in histograms after max and bin pooling. The histograms are used as features to enhance the existing word-level-based ranking function. Both parts of the final ESR ranking model are weighted with values to learn.

In a follow-up article, the same authors propose a word-entity duet representation for document ranking [XCL17]. They consider the representations of both query and documents as both bags of words (BOWs) and bags of entities (BOEs) and then compute four matching scores based on the four possible alignments: query words-document words, query word-document entities, query entities-document words and query entities-document entities. Classic frequency-based methods are used to align BOWs. To match BOW with BOE, the entities' surface forms are used with classical frequency-based methods (for document entities, top k ones are selected). To align BOEs, the authors reuse their previously published ESR method with entities and relations embeddings learned using the TransE model [BUGD+13]. The authors propose to leverage an attention mechanism to mitigate entity linking errors. Four features are extracted for the attention mechanism training. Three focus on the annotation ambiguity based on the entity surface forms ambiguity and popularity. For the last one, an entity word joint embedding model is trained to compute the cosine distance between entities and their words. Finally, a ranking function is learned using the attention features and the BOWs and BOEs alignments by optimising a pairwise hinge loss.

Following the same trend, some of the previous method authors try better to understand the role of KGs in neural IR [LXSL18]. They construct entity embeddings with a linear combination of 3 learned vectors: an entity one, a description words one and a type one. The constructed entity embeddings derive the same entity-word duet translation matrices as in [XCL17], which are then concatenated. The resulting matrix is leveraged in a kernel-based neural ranking model.

In 2018, Shen et al. proposed an unsupervised ranking approach [SXH+18]. They represent documents as BOEs and BOWs and queries as heterogeneous graphs. Two LMs are fitted, one for the document entities and the other for the document words. The queries are represented as token graphs where tokens can be words or entities. The authors then consider two kinds of relations, entity-entity and word-word ones and leverage a type hierarchy to assign relation weights. The ranking model is constructed as a graph covering process rewarding documents capturing inter-entity relations and covering more unique entities. A parameter λ balances the word-based relevance with the entity-based one. Interestingly, since the latter λ has to be learned, the authors define an unsupervised model selection algorithm assuming high-quality ranking models will rank documents based on a similar distribution. In contrast, low-quality ones will rank them in a uniformly random fashion.

Leveraging KGs for IR has been identified as a promising solution to enhance cross-lingual IR [RMdR20]. In 2014, Franco-Salvador et al. proposed an unsupervised document retrieval and categorisation approach based on multilingual document representations [FSRN14]. The latter representations are constructed leveraging the BabelNet KG. The authors first construct document vectors based on the log TF-IDF of terms. The latter TF-IDF values are used to select a set of top K terms per document, which are lemmatised to construct an initial entity graph using BabelNet. The KG is enriched with limited-size paths between lemmas found in BabelNEt. The edges are weighted directly using BabelNet weights, and a topic-sensitive variant of the PageRank algorithm [PBMW99] is used to weight the nodes. The authors then define a graph similarity metric combining a graph structure measure (a sort of weighted Jaccard similarity on nodes and edges sets) and concatenations of the translated document vectors, i.e., vectors constructed based on BabelNet translations of the terms in the initial document vector. The authors test their approach only on document similarity-based tasks.

In [ZRZ16, ZFR16], Zhang et al. summarise multiple research and present their cross-lingual KG-based search engine. In their approach, KG entities are tagged in the documents and queries. A query entity graph is constructed at query time. Furthermore, the top k graph exploration algorithm weighs the nodes and edges. The user interface also enables users to refine their search manually, i.e. providing explicit feedback. The query entity graph and the scores form a query entity vector, which is expanded with related entities. Finally, a standard similarity measure, such

as the cosine distance between the query and document entity vectors, is used to rank the documents. The authors test their system on a news feed in multiple languages.

Article	KG usage	IR system features
[DDA14]	KG as data graph	Semantic indexing, Query expansion, Document Ranking
[XC15b]	KG as data graph	Semantic indexing, Query expansion
[XC15a]	KG as data graph, Ontology as data graph, Ontology as vocabulary	Latent space
[LF15]	KB as KB	Entity-based latent space
[RKC16]	KG as a data graph	Entity and term-based Language Models
[EB17]	KB as KB	Entity based Language Models
[XPC17]	KG as data graph	Entity-based latent space
[XCL17]	KG as data graph, KB as KB	Entity-based latent space
[LXSL18]	KG as data graph	Entity-based latent spaces
[SXH+18]	KB as KB	Entity and term-based Language Models, Graph-base ranking
[FSRN14]	KG as data graph, KB as KB	Term and graph-based similarity scores
[ZFR16]	KB as KB	Semantic indexing, Graph-based similarity, Entity-based latent space

Table 2.1: Summary of the KG-based IR approaches listing the main system features and their KG usage (KB stands for Knowledge Base).

Discussion Table 2.1 summarises the KG-based IR literature review using the term *knowledge graph*. The column labelled *IR system features* lists for each method the main features. The column *KG usage* gives some KG usage details that we will explore at the end of this chapter.

KGs, particularly their entities, enhance IR systems by enabling an improved understanding of the user intent, queries, and documents. KGs enable exploring related entities and explanations, pushing such understanding beyond what we can achieve through only word tokens [RMdR20].

The first approaches leveraging KGs for IR extended the older traditional linguistic-based methods, such as query expansion and LMs approaches, to incorporate KG entities. Latent spaces are also adapted to incorporate KG entities. The recent approaches follow the latent space idea with methods incorporating neural network-based components. Extending such historical approaches comes with its well-known drawbacks and advantages.

While historical research tends to produce unsupervised approaches such as TF-IDF or BM25, the recent ones often include at least one supervised component. It might be more efficient. However, it also requires a wealth of annotated data only sometimes available [XC15a]. Many recent approaches also incorporate an entity-based query document matching component while keeping the word-based matching separate. These methods keep the word-based score to compensate for the potential lack of known entity in both the query and the KG. However, it often results in balancing the final score between the word and entity-based components using a parameter. The latter parameter needs to be learned in a supervised fashion. Only [SXH+18] proposes an unsupervised solution among all the surveyed methods.

Multiple strategies are expansion-based. The query, the document, or both are enriched with terms and entities. In historical methods, adding some terms to the query is a common approach

to deal with the problem of vocabulary mismatch between the user and the documents. However, focusing on expansion-based approaches, one challenge is the high-risk, high reward of such methods [XC15b]. While enriching the document or query with terms may be very beneficial, it may also greatly hinder the retrieval process by producing unrelated terms. There is no known right middle.

For efficiency and scalability reasons, many document-query score computations still heavily rely on component values that can be pre-computed offline. Typical examples are the Term Frequency (TF), Inverse Document Frequency (IDF) and their entity-based siblings. Some approaches explore query-dependent score computation. However, such query-dependent computations have to be performed online at query time. Hence, they must be limited to be considered in a real-life setting.

We also note that the KG used are often OKGs such as Freebase [BEP+08], Yago [SAB+23], DBpedia [LIJ+15], or Wikidata [VK14]. Though quite complete, they may need more detailed knowledge in particular domains, i.e., their content might be too generic, lacking details essential to their effectiveness in domain-specific contexts. It is crucial to evaluate the KG content first with regard to the targeted business use cases. Though OKGs add value in practical applications, they often quickly fall short and must be extended. Such KG extension leads us to the last and most critical limitation. The approaches heavily depend on the Entity Linking (EL) task.

2.3.2 Ontology-based Information Retrieval

We must consider two keywords for our literature review on KG-based IR systems. In the previous section, we studied IR methods presented with the keyword *knowledge graph*. This section studies the literature introducing IR approaches using the keyword *ontology*. Selected articles mainly focus on industrial use cases. Though we also include some literature focusing on use cases from other domains, we can broadly group the ontology-based IR approaches we explore into the Architecture Engineering and Construction domain (A/E/C), with some notable examples focusing on Computer-Aided Design (CAD) and Building Information Model (BIM) model retrieval.

The A/E/C projects produce a lot of technical documents. IR systems are critical to these industries to retrieve their complex, heterogeneous, specialised documents quickly. Healthcare is another similar domain with such a need. Though these industries manage documents with some textual content, such text and the metadata contain domain-specific concepts and vocabularies. OKGs and the existing ontologies often describe too high-level concepts lacking the fine-grained knowledge required by IR applications[CJH19]. Hence, companies' IR and knowledge management tools require domain-specific KGs built from scratch or extending existing ones.

Let us consider a more general review before focusing on particular industry use cases. In [MS18], the authors study the use of ontologies for knowledge modelling and information retrieval. They break down their analysis in two. They first explore ontology-based information retrieval and then focus on database-to-ontology transformations and ontology-to-database mappings. Their study of IR systems with ontologies highlights three use cases: IR systems leveraging ontologies to assist visual and interactive query formulations, systems enhancing keyword search by information linking, and systems refining user queries based on domain ontologies. Most ontology-based visual query formulation systems help construct a query by navigating ontologies designed as concept trees. In query formulations, ontologies enable a translation layer on top of a relational database. In such a use case, the user also browses a domain ontology to formulate the query. A database-to-ontology transformation constructs an ontology which is mapped to the domain ontology. The generated ontology and the mapping form the translation layer, enabling a transparent database query construction from the user query. In query enhancement by information linking use cases, the ontology serves as a translation and knowledge source integration layer, formulating refined user queries adapted to each data source. Finally, in the user query refinement use case, the ontology helps enrich, disambiguate or reformulate a text-based user query leveraging the ontology structured vocabulary.

Much research focuses on the A/E/C domain and structures knowledge in various specialised technical domains to enhance IR capabilities. In a series of articles from 2011 to 2019, Shang-Hsien et al. explore the construction of ontologies in the engineering domain for applications in IR. In [HLC+11], the authors introduce a semi-automatic procedure for constructing a base domain ontology from domain handbooks. The approach relies on digitised books containing at least a table of contents, definitions or descriptions of terms organised in an index and a glossary of relevant terms for each chapter. The procedure leverages the semi-structured content to construct a vocabulary and organise it into a hierarchy with non-hierarchical relations abstracted to "has-a" ones. Then, the approach alternates between domain expert revisions and rule-based refinement. In [LCH12], the authors use the base ontology construction method to propose an ontology-based IR system. Their research proposes OntoPassage, an ontology-based method to partition a corpus into concept-focused passages. OntoPassage constructs document passages based on the domain ontology concepts and the frequencies of their associated terms in the corpus. The authors experiment with three historical approaches to retrieve passages based on a user search. The retrieved passages associated with concepts let the user explore the corpus following the concepts. More recently, in [CJH19], the authors build on the base ontology development method of [HLC+11] to propose an enhanced semi-automated approach to developing a base domain ontology tailored explicitly to supporting IR tasks. Their approach selects the base domain ontology top-level concepts based on the domain expert's main topic of interest, i.e., information need. The ontology construction is based on a top-down iterative pseudo-relevance feedback approach. For each hierarchy level, the concepts are used as query terms to retrieve N documents considered relevant. In the latter top N documents, K terms are selected to become sublevel concepts. The approach selects the K terms based on the Chi-square test, a standard statistical score to evaluate the independence of two variables. The ontology designer chooses a threshold for the Chi-square measure. The process is repeated until no further relevant sub-concepts are found. The concept trees are then merged based on concepts in common to form the ontology. This new approach still abstracts non-hierarchical relations between concepts as "is related to" ones.

Still in the engineering domain, in [HYLS14], Hahm et al. propose an ontology-based approach to personalised engineering document retrieval they expand in a subsequent article the following year [HLS15]. Their method represents documents and user preferences as semantic networks. The latter networks are constructed with the help of a domain ontology. They point out the need for ranking methods to consider the relations between concepts within and between the user intent and the documents. They propose a ranking approach based on bags of relation triples rather than bags of concepts to address this issue. Concepts and their relations are assigned different importance computed with linear combinations of various term frequency-based scores, considering the significance of the source concept relation regarding both the corpus and the user.

The authors of [STS11] focus on image retrieval to support concept designers. They address the challenge of not relying on any annotated example requirements. To do so, the method first extracts from the web texts associated with each image in the corpus. The authors then extract concepts from the texts by aligning them with existing ontologies. The extracted concepts are used to tag images. The image retrieval process relies on the TF-IDF score. The authors consider any taxonomy, thesauri, or classification to be an ontology in their work.

In [KKW+16], the authors stress the importance of provenance visualisation in enterprise search processes. They propose a search system with various visualisation features. The system design eases the integration of multiple ontologies. Ontologies, particularly their term ontology, are used to construct search provenance graphs and enrich user searches. Their document scoring approach relies on bi-party graph matching.

The above studies focus on engineering document retrieval in general. The construction domain leverages Building Information Modelling (BIM) resources for their different activities. BIM technology is capable of restoring both geometric and rich semantic information of building models, as well as their relationships, to support lifecycle data sharing. Hence, some research focuses on BIM resource retrieval. In [GLW+15], Gao et al. introduce the BIMseek search engine.

The approach description focuses on their ontology constructed from the Industry Foundation Classes (IFC), a major standard for BIM, and the query expansion process the ontology assists. In [GLL+17], Gao et al. focus on BIMTag, the semantic annotation tool they use to tag BIM documents for the BIMseek search engine. More recently, in [LLL+20], Li et al. enhanced BIMseek, proposing an approach to measuring similarity between BIM documents based on a combination of the document content and attributes. We summarise this series of articles below.

In [GLW+15], Gao et al. propose a local feedback-based query expansion leveraging both term and concept expansion. The method first enriches query terms with synonyms from WordNet[12]. Then, the synonyms are used to retrieve concepts in the ontology built using the IFC standard. The concepts are expanded based on their distance in the hierarchy tree. The expanded concepts are then pruned using a Local Context Analysis (LCA). The expanded terms and concepts are used to select documents to evaluate concepts' relevance based on the cooccurrence of concepts and terms. Once the concepts are pruned, the method selects documents and ranks them based on a Vector Space Model (VSM) where the vector is composed of weights assigned to terms and concepts. The authors also detail the ontology construction process. The latter ontology is an almost direct translation from the IFC standard industry classification into an OWL class hierarchy with attributes.

[GLL+17] presents BIMtag, the system used for semantic annotation of BIM documents in the BIMseek search engine document indexing. The annotation operates at the word and document level. The authors use the ontology for word sense disambiguation at the word level, similar to the LCA applied for the BIMseek search. They leverage a LSA approach at the document level. The documents and concepts are projected into a latent space where the dimensions are the terms used in the documents denoting concepts in the ontology. Documents are enriched with concepts similar to the ones they contain. The BIMseek search engine favours the latter document-level concepts.

[LLL+20] enhances the BIMseek system by introducing a new similarity measure between BIM documents. In this extension, the user query is a BIM document, and the search process leverages the large amount of BIM document attributes to compute a similarity with the BIM documents forming the corpus. The similarity score focuses on the geometric and semantic attributes and is based on a combination of Resnik information and Tversky similarity models. Resnik formula quantised the amount of information content for a concept as the probability that the concept appears in a document set. Tversky model computes document similarity based on a document attributes set-based formula.

In engineering, many technical documents come with 3D models. Those 3D models, called Computer-Aided Design (CAD) models, are generally considered central representations used to convey knowledge and information along the product design process [LPMG19]. In recent years, the amount of data associated with product design has steadily increased. Companies have become aware of the strategic importance of sharing the knowledge accumulated in those component designs. Hence, much IR research considers the CAD model retrieval use case. It is the use case we are considering in this PhD.

Since various engineering design requirements exist, CAD model retrieval systems consider queries as text but also in other forms. One recurrent and critical use case is finding similar designs. Many approaches explore using a CAD model as a query in such a context. The IR task then focuses on selecting and ranking similar CAD models. In [QGY+16], Quin et al. concentrate on an ontology-based IR system with a CAD model as user queries. The authors propose building two ontologies. The *common feature ontology* represents a standardised set of geometrical and topological concepts, and the *system feature ontology* is a system-specific ontology serving as a mapping between the system CAD model representation format and the common feature ontology. The latter ontology constructs a semantic descriptor for each CAD model. The authors also leverage some reasoning over the ontologies using subject matter expert-defined rules ex-

[12] https://wordnet.princeton.edu/ (Accessed on Thursday 3rd October, 2024)

pressed in SWRL. They then propose two retrieval processes: a vector space model process with term frequency-based vectors and a tree-based one comparing graph patterns.

In an older, though foundational research [LRR08], Li et al. introduce a CAD model search system based on ontologies constructed as interlinked sets of taxonomies. In their system, user queries are text, and they propose a visual interface to navigate the different taxonomies simultaneously. They construct two ontologies and distinguish the concepts from engineering terms. They break down the engineering domain into themes that define the focus of taxonomies. Concepts are then hierarchically organised in their respective taxonomy. Non-hierarchical relations across taxonomies are also defined. For instance, the authors define a taxonomy of functions and one of the devices. Some devices have some functions, introducing some non-hierarchical relations. The indexing and CAD model retrieval processes are based on terms aligned with the ontology concepts using the ontology of engineering terms.

Article	KG usage	IR system features
[LCH12]	Ontology as vocabulary	Concept-based exploration UI, Semantic indexing
[CJH19]	Ontology as vocabulary	Pseudo relevance feedback concept extraction
[HYLS14, HLS15]	Ontology as data graph, Ontology as vocabulary	Semantic indexing, Ranking model
[STS11]	Ontology as vocabulary	Semantic indexing
[KKW+16]	Ontology as data graph, Ontology as vocabulary	Concept-based exploration UI, Semantic indexing, Query expansion
[GLW+15]	Ontology as data graph, Ontology as vocabulary	Semantic indexing, Query expansion
[GLL+17]	Ontology as vocabulary	Semantic indexing, Latent space
[LLL+20]	Ontology as vocabulary	Semantic indexing
[QGY+16]	Ontology as data graph, Ontology as vocabulary	Semantic indexing
[LRR08]	Ontology as data graph, Ontology as vocabulary	Semantic indexing

Table 2.2: Summary of the ontology-based IR approaches listing the main system features and their ontology usage.

Discussion Table 2.2 summarises the KG-based IR literature review using the term *ontology*. The column labelled *IR system features* lists for each method the main features. The column *KG usage* gives some KG usage details that we will explore in the conclusion of this chapter.

Ontology-based IR methods focus on highly domain-specific use cases for texts with many technical terms. Our review focused on the A/E/C domain, particularly BIM and CAD model retrieval. However, other domains like biology or healthcare explore similar ontology-based IR approaches. The documents in those domains contain many particular technical terms. Some commonly used terms are employed with a domain-specific meaning, requiring word sense disambiguation. The concepts denoted in the documents also require a low-level granularity, and their definitions involve complex constraints and relationships.

Hence, ontologies are expressed in formal logic-backed languages, such as OWL, to define each term's meaning formally. Such formal definitions enable the deduction of new facts based on deductive reasoning. However, ontology-based IR methods that leverage relations amongst concepts often focus on hierarchical relationships, such as those found in class hierarchies or taxonomies. It results in ontologies forming of tree structure sets [MS18].

In our review, only [QGY+16] leverages some reasoning, using rules. Though the ontology is expressed in OWL, the authors define SWRL rules to derive the new facts. We did not find any method utilising an ontology expressed in OWL and leveraging OWL reasoning other than through the class hierarchy. If the ontologies used in the scientific literature define logical axioms other than hierarchical ones, enabling the inference of new facts, they are either not used or not mentioned. One possible reason is that such reasoning is performed offline and not reported in the articles. The methods could implicitly consider the ontologies as containing the inferred facts. Another probable reason for using OWL is its standardised RDF graph representation, which facilitates sharing and adoption. The RDF representation also provides a graph structure that is straightforward to process.

The ontology-based IR literature also heavily relies on terms. Though the terms are associated with concepts and used to match the text documents with their related concepts, some methods consider terms as concepts. Other ones, such as [LRR08], explicitly define a term and a concept ontology separately. The term ontology helps align texts with their embedded concepts.

2.4 Conclusion

After defining Information Retrieval and its main elements, we explored using KGs to support IR in this chapter. IR can also support KGs in some domains [RMdR20].

We can distinguish the user query, the corpus and the IR system in IR. A user utilises an IR system to find documents in a corpus. An IR system aims to align the user information need with the documents in the corpus. The user information need is expressed with a query that can take many forms, e.g., keywords, a natural language question, or an image. It is essential to notice that the user query is already a representation of the user's need. Hence, some information is lost, and the IR system must operate with limited information. In each user query, some implicit knowledge is hidden.

An IR system has to recover the implicit or missing knowledge in a user query. It can act on the user query, the corpus, or a subset of candidate documents pre-selected using the user query. One of the main tasks is to align the user query vocabulary with the corpus one. To address this task, methods enrich with various metadata the user query, the documents, or both. Among the various metadata, some are sometimes called features and used to build a latent space, i.e., a built vector space in which it is easier to align the user query with the documents. Some metadata come from already structured external resources or are computed directly from the corpus content. To operate in a real-world application, as much metadata as possible must be computed offline so that as little computation as possible is needed at query time. Hence, in an offline process, the metadata are computed, and the documents are stored in a data structure, easing their retrieval based on those computed metadata. The documents are indexed. Figure 2.4 summarises IR systems, gathering their main components and tasks in one figure.

When introducing the KG-based IR literature review section, to ease the readability, we started using the terms KG-based IR and ontology-based IR to refer to the KG-based IR methods explained using the terms *knowledge graph* and *ontology*, respectively. We now refer to the term KG as we defined it in chapter 1, i.e. encompassing the ontology. Though the methods share a lot, there are some differences worth noticing.

The literature considers KG structured content, which reduces the search space. The concepts or entities defined in the KG have fewer variations than the terms denoting them in documents. The methods also emphasise the graph data structure, enabling the definition of complex relationships between concepts and entities, i.e., the nodes in the data graph and their formal definitions.

However, different communities proposed the scientific work we reviewed. Hence, some differences arise, mainly in the use cases they consider. The methods introduced using the term *knowledge graph* focus on leveraging the graph data structure beyond the hierarchies often leveraged in ontology-based approaches.

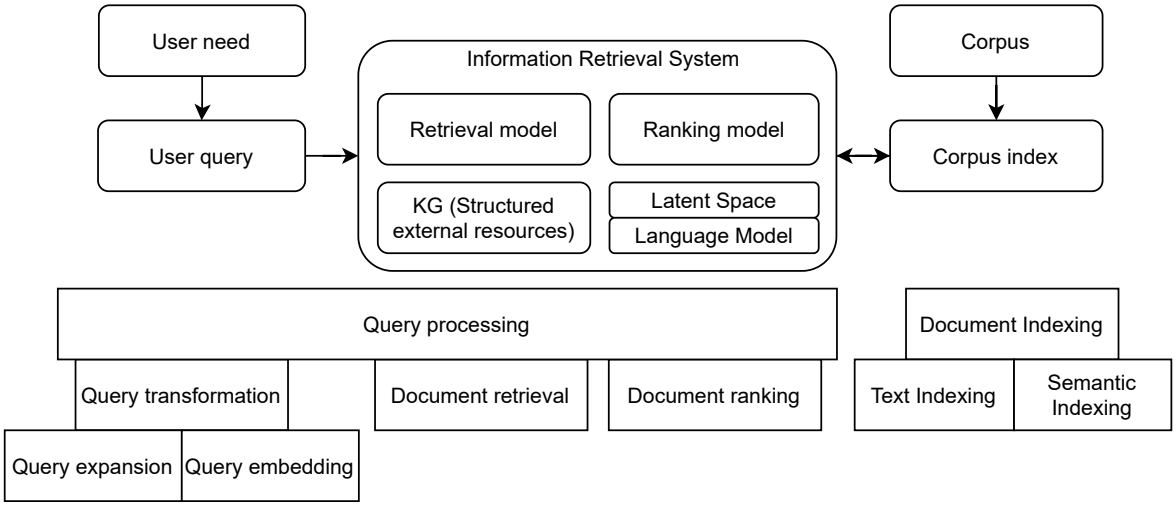


Figure 2.4: Information Retrieval system overview.

Round boxes denote system components. Square boxes at the bottom denote the main tasks. The latter tasks are placed below the components they act on.

The latter ontology-based approaches focus on more domain-specific and technical use cases than knowledge graph ones, which often use large OKGs spanning many domains. It implies a notion of scale. Where methods presented with the term Knowledge Graph use extensive graphs, methods introduced using the term ontology focus on smaller graphs often constructed for the use case. The domain-specific context often requires creating a specialised KG. Hence, methods described with the term ontology develop a methodology to construct the KG from the corpus or domain experts. Due to the technical terms used in the domains considered by ontology methods, the KG is often considered a structured vocabulary.

The large KG scale considered in the literature using the term knowledge graph enables the application of inductive reasoning methods such as deep neural networks-based techniques. Meanwhile, the smaller KGs involved in the literature described with the term ontology are often implemented using OWL and therefore utilising deductive reasoning approaches.

The IR literature review shows that KGs applied to support an IR system act as external structured content. It is a data artefact containing curated and structured content that is easier to manipulate than the existing corpus. Following our definition of KG, we distinguish the following usage of both terms in the IR research we study:

- *KG as data graph*: The term *knowledge graph* denotes a data graph used for its graph data structure. Relations among the entities in the KG and a graph algorithm are leveraged to support one or more IR system tasks.
- *KG as a knowledge base*: The term *knowledge graph* denotes a KB. The KG is used as a set of entities, i.e., elements often associated with some text representations. The relations among entities are not leveraged.
- *Ontology as data graph*: The term *ontology* denotes a data graph. the KG might be expressed in OWL for its RDF-based representation. Hence, it is the graph data structure that is used. The logical axioms are not leveraged. However, concepts similar to the elements of a KB are defined.
- *Ontology as vocabulary*: The term *ontology* denotes a data graph focusing on terms. It is a structured vocabulary with potentially some linguistic relations to link a text representation with the same concept.
- *KB as KB*: Some authors also rightfully employ the term *knowledge base* to denote a set of entities, i.e., a KB.

We report the above term usage in the summary tables 2.1 and 2.2. Most approaches introduced using the terms *ontology* or *knowledge graph* leverage the graph data structure. Only some using the term *ontology* consider the KG as a structured vocabulary. However, we could argue that this ontology usage is similar to the *KG as knowledge base* one.

This chapter details IR systems, focusing on the ones leveraging a KG. Our definition of KG in chapter 1 introduces a complex and versatile data artefact which can represent many forms of information among which knowledge. The literature review demonstrates that methods leveraging KGs to support an IR system focus on the graph data structure and, consequently, the data graph KG component. While there is much to explore focusing on KG data structure, some formal knowledge representation capabilities could be further explored. Hence, the main contribution of this manuscript focuses on the domain graph component of a KG expressed in OWL. While state-of-the-art KG-based IR methods focus on inductive reasoning, we explore the application of deductive reasoning approaches. The other contributions of this manuscript consider the integration of KGs in KG-based systems.

Part IV

Contributions

Chapter 3

An operational Knowledge Graph-Based Systems architecture

Contents

In this manuscript's first part, we explored the KGs and IR state of the art, focusing on approaches leveraging KGs for IR. In chapter 1, we introduced our definition of a KG, clarifying the relation between ontologies and KGs. An ontology is a component of a KG, more specifically, part of the domain graph (Cf section 1.5.3). In chapter 2, we studied the different usages of KGs to support an IR system. We can summarise these KG usages as methods using KGs as a graph data structure and methods using them as a structured vocabulary. Our literature review of KG-based IR approaches shows that methods concentrate on the IR systems but never detail the integration of the KG.

While many applications use KGs, the KG-based IR literature we reviewed focuses on the application of KGs for particular IR purposes. The research community has extensively studied KGs in their various forms. However, we found a need for research exploring KG-Based Systems (KGBS) from a broader system perspective and studying the integration of KGs in an information system abstracting the downstream application. Authors of KG-related literature only mention using a KG to support their use case or focus on its construction process, e.g., [EFK19].

Hence, before focusing on our approach to KG-based IR, we explore the integration of KGs in a system from a theoretical and abstract point of view. This chapter studies KGBSs. We explore how KGs can fit an overall information system regardless of any specific use case. We propose our KGBS architecture to better understand the KG roles within a system and how we can integrate and implement them.

We first introduce our KGBS architecture. We then discuss candidate technologies to implement each architecture component, focusing on open-source Semantic Web-based solutions. We conclude by applying the KGBS architecture to an IR use case, illustrating our argument.

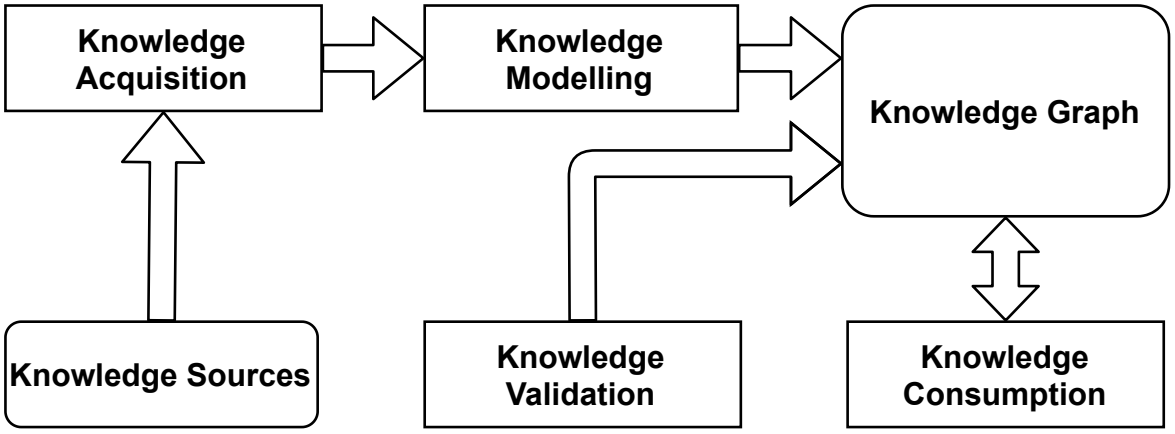


Figure 3.1: Knowledge Graph-Based System architecture overview.

Square boxes refer to activities. Round boxes represent containers. Arrows illustrate data flows between activities and components.

3.1 An architecture for Knowledge Graph-Based Systems

The use of KGs in the industry and the different scientific communities exploring them for decades illustrate the theoretical and practical opportunities brought by modelling knowledge in a machine-interpretable format. Though challenging and time-consuming to produce, explicit and structured knowledge is more straightforward to manipulate than implicit knowledge. KGs are the best-known candidates to implement such explicit knowledge in a machine-readable and interpretable format. However, the main challenges arise when implementing and using KGBSs. This section discusses a general architecture for a KGBS, regardless of the downstream application. Figure 3.1 gives a high-level overview of the components in our architecture. We read the figure from left to right, though the components' positions in figure 3.2 do not intend to reflect any order in the activities. Indeed, KG construction is a cyclic iterative process. Knowledge is extracted from knowledge

sources via tasks grouped under the label *knowledge acquisition*. *Knowledge modelling* structures the extracted knowledge in a machine-readable and interpretable format to form the KG. Building a KG requires a significant effort; hence, such endeavours pursue supporting at least one application and having a data structure that is versatile enough to support more applications in the future. We gather these applications under the name *knowledge consumption*. Finally, though not explored in this manuscript, we mention for completeness the processes involved in validating and maintaining the KG under the name *knowledge validation*.

We now dive into the details of each architecture component and give some examples. Figure 3.2 presents our operational architecture for KGBSs detailing each component. While we can easily translate some activities into a computer process, some also denote a human thought process.

3.1.1 Knowledge acquisition

The literature often refers to knowledge acquisition as one activity, e.g., in [MF97]. Our architecture divides the latter process into knowledge elicitation, extraction, and consumption. Following [EFK19], before modelling any knowledge, we need to scope the project and define the concepts and relations of interest. Knowledge elicitation aims at extracting the concepts required to answer the business questions, i.e., the Competency Questions (CQs) [EFK19]. The latter CQs are derived from a business need and can not generally be fully automated. However, some tasks can help precisely define the business questions and discover the concepts and relations needed to answer them.

Considering IR use cases, for example, concepts are hidden in a corpus of documents and user queries. In other use cases like data integration, the concepts and relations might be hidden in a database structure, queries and code logic. Data profiling computes and displays various statistics about these contents. The task is best driven by the exploration of the knowledge scientist as defined by Fletcher et al. [FGS20]. The latter analysis lets us select the concepts needed to answer our CQs.

After selecting a concept, we define heuristics to extract instances from documents or enterprise resources. An example of a heuristic is that phone numbers (in France) are sequences of ten digits with potentially an optional double zero or plus sign and a two-digit country code before. In the knowledge extraction activities, we extract concept instances after implementing the heuristic, e.g., with a regular expression.

In the knowledge acquisition part of our KGBS architecture, we distinguish between the latter extraction process and the production one, which applies transformations for the data to fit the chosen KG technology. For instance, as part of our work we use an Object-Oriented Paradigm (OOP) for knowledge extraction and decide to represent our KG with a Directed Edge Labelled Graph (DELG) model. Hence, the knowledge extraction phase will construct objects, i.e., OOP class instances describing all the extracted pieces of information. Such OOP objects are typically collections of concepts and relations. The knowledge production process will build the DELG based on the objects created in the previous phase. Knowledge production is performed in collaboration with the knowledge modelling one we now introduce.

3.1.2 Knowledge modelling and the KG

The main container in a KGBS is the KG. It has the critical role of holding a single source of knowledge. The sole goal of the knowledge modelling process is to organise the knowledge acquisition production into a meaningful computer processable data structure, i.e., to model concepts and their relations forming the domain graph. Knowledge production also aligns the concepts and relations with the data graph, i.e., it defines the mapping in our KG definition 1.5. In chapter 1, we focused on defining a KG. In the KG component we depict in figure 3.2, we focus on the different kinds of knowledge often modelled in a KG. We break down this knowledge into four subcomponents: domain, business, structural, and data provenance knowledge.

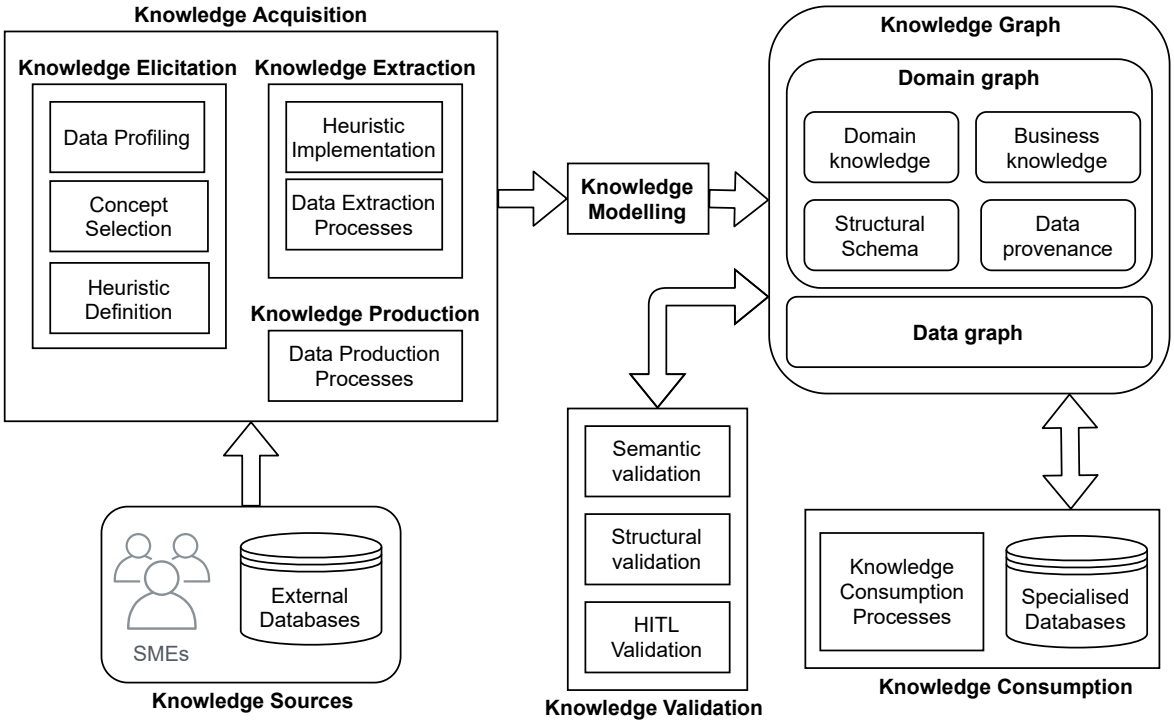


Figure 3.2: Knowledge Graph-Based System architecture with detailed system component overview.

Square boxes refer to activities. Round boxes depict containers. The boxes imbrications denote sub-activities and sub-containers, respectively. Arrows illustrate data flows. (SMEs: Subject Matter Experts; HITL: Human In The Loop)

The domain knowledge represents concepts and relations describing the domain of interest, e.g., mechanics, medicine, biology. For instance, domain knowledge could be standard notations, specific dimensions and terms in a mechanical engineering application. The most common examples are modelling units, dimensions, time and domain-specific terms.

Some foundational work proposes to break down this domain knowledge into categories such as top-level, domain, task and application knowledge [Gua98]. Those classifications of knowledge might be pertinent, particularly for documentation and ease of understanding. In this manuscript, we propose our categorisation. However, in practice, while some concepts are straightforward to categorise, separating the categories is subjective and often confusing. What remains essential is to share a common understanding of the knowledge within an organisation.

The business knowledge includes all business- and application-specific knowledge. It can range from company employee details for the HR department to marketing documents for potential customers. The concept's definitions and the terms used to denote them are specific to a company's point of view. While domain knowledge denotes a point of view shared across an entire domain, business knowledge is specific to an organisation. For instance, in the financial domain, we might define a contract for a loan. However, from a business perspective, this loan is a liability for one party and an asset for the other. While the domain knowledge defines the loan contract concept, its definition might differ, and the business knowledge might refine it.

The other parts of the KG are knowledge necessary for lineage and maintenance. They are paramount to ensuring the KG's organic evolution and trustworthiness. Nowadays, data provenance knowledge is the knowledge we still need to include the most. We store loads of data and information, producing even more from them, but we need metadata to keep track of their meaning and origin. The domain knowledge adds context and meaning to business knowledge. However, to trust the knowledge we extract from the latter and make informed deductions, we need to know precisely where each piece of knowledge comes from and what processes produce it. Keeping track of such data provenance knowledge is known as data lineage. The structural schema defines the expected shape of data, e.g., an employee must have a social security number and a

ten-digit phone number. Such structural metadata is necessary to ensure the quality of stored knowledge over time and ease its maintenance and consumption. It is not to be confused with domain knowledge, which describes data semantics, e.g., a person employed by a company is an employee of the latter, and a social security number is an identifier that uniquely identifies her or him. The domain knowledge and structural schema are leveraged by two separate validation processes we explore next.

3.1.3 Knowledge validation and consumption

In the previous section, we introduce metadata for knowledge validation. They are pieces of knowledge encoded using specific vocabularies. We discuss some of them in a subsequent section when introducing candidate technologies. Knowledge validation processes are systems designed to understand the latter vocabularies and process them on the KG to validate its semantic consistency and the shape of the data graph. As we discussed in section 1.2.4, it is essential to distinguish between structural validation and semantic validation. The latter concentrates on logical consistency and is more challenging, often requiring domain experts to check the encoded logic. However, once the domain knowledge, business knowledge and structural schema are adequately defined, the knowledge validation processes automatically validate the domain and data graph. Nevertheless, validation includes some Humans In The Loop (HITL) processes. Involving humans, such as Subject Matter Experts (SMEs), effectively is often challenging but critical.

In figure 3.2, we abstract any specific use cases in the knowledge consumption activity. Some inductive inference processes typically infer new knowledge based on the data graph topology, which we then add to the KG. Alternatively, some parts of the graph are loaded in a dedicated database for performance reasons when an application needs to consume the knowledge. The KG needs to be the single source of knowledge. As such, application databases are responsible for staying in sync and up to date with it. In particular, if any application gathers or derives any new knowledge, it should implement some processes to communicate this knowledge back into the KG so that the other applications can benefit from it. It should not be the other way around, i.e., creating knowledge locked in siloed as it is too often the case[1] [Mcc18].

3.1.4 Knowledge sources

Knowledge acquisition methods consider an ideal setting with access to SMEs and enough time, e.g., [EFK19]. Domain experts are rarely available in practice. Indeed, it would be ideal to have unlimited access to a group of domain experts and translate the knowledge they are sharing into a machine-interpretable model. Unfortunately, very few companies have the resources to invest enough to fully allocate SMEs to a knowledge-sharing activity. The lack of access to domain experts is one of the main challenges for knowledge acquisition.

Knowledge sources include SMEs for completeness in figure 3.2. However, besides domain experts, in practice, most knowledge is encoded in companies' databases, schemas, documents and code bases. Each employee is also an expert in one aspect of the company's activity. They encode their knowledge in various forms. For example, the developers introduce knowledge in the code they write. The challenge is not the access to knowledge but rather the multiple forms it has and the vast amounts we need to process.

3.2 Discussing some candidate technologies

In the previous section, we propose an architecture for KGBSs depicted in figure 3.2. We discuss each system element from an abstract point of view without considering any implementation. This section discusses candidate technologies focusing on open-source solutions and semantic web standards. We enrich figure 3.2 with the candidate technologies we discuss in figure 3.3.

[1] http://datacentricmanifesto.org/ (Accessed on Thursday 3rd October, 2024)

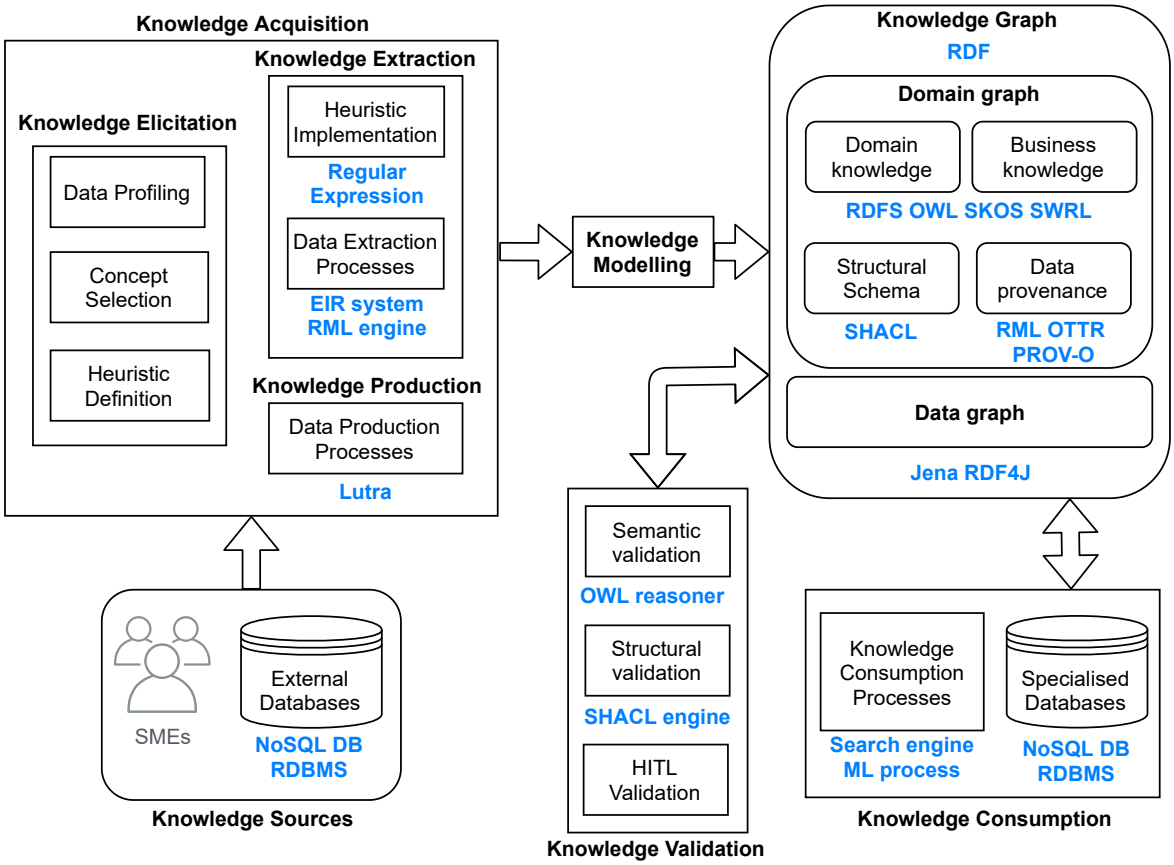


Figure 3.3: Knowledge Graph-Based System architecture with detailed system component overview and candidate technologies for their implementation.

Square boxes refer to activities. Round boxes depict containers. The boxes imbrications denote sub-activities and sub-containers, respectively. Arrows illustrate data flows. Some candidate technologies for implementing each component is mentioned in bold blue letters. (SMEs: Subject Matter Experts; HITL: Human In The Loop)

3.2.1 Different Knowledge Graph paradigms

KGs play a role in a wide range of applications. Depending upon the use case, its implementation varies. In section 1.2.3, we introduce the distinction between the knowledge modelled through the KG and its graph data structure. In particular, we mentioned two main graph data models used in practice: the Directed Edge Labelled Graph (DELG) and the Labelled Property Graph (LPG).

We can roughly distinguish between two main kinds of KG usage in the industry. When the problem is a typical graph one or considered a graph problem for solving purposes, i.e., it involves some graph algorithms, such as a centrality or shortest path one, a graph database stores and lets us manipulate the data. This graph database is said to store the KG. However, the stored graph is often only a portion of the original KG extracted for a dedicated application. In our architecture, this KG usage would correspond to a knowledge consumption activity discussed in section 3.1.3. Typical examples are any analytics performed on networks of any kind. While in this usage, the graph might be called a KG, it is usually a minimal form of KG regarding our KG definition (Cf section 1.5). Such systems solely leverage the graph data structure and often miss the processes feeding back the derived knowledge into the original KG. In this paragraph, we generalise our comment based on what we have seen in the broader literature. However, focusing on IR, our literature review in section 2.3.1 presents some methods considering the KG as a data graph and leveraging the graph data structure. This KG usage is typically implemented with an LPG model.

The second use of KGs is for modelling knowledge as metadata. This KG usage corresponds to a data management solution. The emphasis is put on the flexibility of the graph data structure, enabling the representation of any knowledge and the possibility for continuous extension. The knowledge-sharing feature is also essential. As such, the SW technologies based on RDF suit KG approaches with the need to model metadata describing content. As introduced in chapter 1, the DELG model can be implemented using the RDF.

We advocate for the latter framework since the approach to IR we focus on in our works leverages knowledge modelled with formal Logic. We consider KG to be modelling metadata. The W3C semantic web standards were designed for modelling and exchanging metadata. Many technologies have been developed on top of RDF and implemented in large-scale projects. Hence, we do not wish to reinvent the wheel. This framework provides all the required languages, and the vast community of users have developed tools to manipulate them. RDF also comes built-in with a unique identifiers scheme. It allows us to exploit identifiers globally, internally and externally, to explore and integrate external knowledge bases such as the DBpedia [LIJ+15] databases or Wikidata [VK14], which come with an RDF serialisation.

3.2.2 Knowledge Graphs implementations

Many commercial solutions exist to store KGs represented with the DELG or the LPG models. The graph database market is still in its early days. There is a constantly changing landscape of newcomers [SL21]. Our work focuses on open-source solutions to store KGs expressed with RDF triples. Such databases are called triple stores.

There exist two main open-source frameworks for triple storage and manipulation, Eclipse RDF4J[2] and Apache Jena[3]. They are complete frameworks with built-in components for storing, querying, reasoning, loading, and exporting RDF triples. They are also flexible enough to integrate other solutions, such as commercial ones, to replace particular elements. These open-source frameworks illustrate the advantage of using a standardised representation such as RDF. We are only committing to the standard and not to any software vendor. For instance, the storage layer could be any database, ranging from RDBMS to document-oriented ones and passing by native graph storage.

[2] https://rdf4j.org/
[3] https://jena.apache.org/

3.2.3 Knowledge acquisition

In section 3.1.1, we break down the knowledge acquisition process into knowledge elicitation, extraction and production. Knowledge elicitation can involve technological tools, though it is primarily human work. Fletcher et al. define the role of the knowledge scientist in [FGS20]. Using their own words, knowledge science is technical, and people work. Tools can help but not replace the knowledge scientist. In particular, technological tools could help engage the domain experts in the knowledge acquisition process.

As mentioned in section 3.1.4, having sufficient access to domain experts is demanding and unrealistic for many businesses. However, most knowledge is hidden in various formats, and much is embedded in text. Hence, though out of the scope of our research, topics related to Natural Language Processing (NLP) are of interest. We do not focus on advancing the NLP research; instead, we use some results in the following chapters. Python frameworks such as spaCy[4] lets us quickly implement Named Entity (NER) or Entity of Interest Recognition (EIR) systems (Cf. 2.2.4) that can also implement efficient regular expressions.

Figure 3.3 mentions the *Rule Mapping Language (RML) engine* and *Lutra*[5] as candidate technologies for knowledge extraction and production, respectively. An *RML* engine is software designed to process the *RML* language. Furthermore, *Lutra* is a software to process the *Reasonable Ontology templates*. We discuss these technologies in section 3.2.5 focusing on knowledge provenance.

3.2.4 Knowledge modelling

In section 3.1.2, we break down the KG's domain graph into four components but do not mention any standard languages. We wish to model domain knowledge in OWL eventually. However, we recognise that OWL's complexity and logical foundations make it hard to adopt from an industry point of view. The open-world assumption is often problematic to grasp, and most employees are familiar with object-oriented programming principles, which leads to lousy modelling practices and misunderstanding of the inferences.

We already noticed in the KG-based IR literature review, in chapter 2.3.1, that most KG projects leveraging inference are limited to the hierarchical ones, i.e., the transitivity of hierarchical relations. RDFS or SKOS languages are more accessible and often sufficient to initiate a knowledge modelling project. Within a company, there often already exists, in various forms, some business concept classifications. The first step is to standardise them before enriching them. When the need for more complex descriptions arises, we can move from SKOS to OWL. To this purpose, the SKOS standard discusses the link between SKOS concepts and OWL classes [IS09]. When OWL expressivity is insufficient to model the required knowledge, we can leverage SWRL to define rules.

Not all concepts modelled in SKOS have to become an OWL class, i.e., not all concepts need an advanced description in OWL. Both modelling languages can work alongside. RDFS, SKOS and OWL enable us to model domain and business knowledge. It is often tempting to model an entire document as precisely as possible. We might need such a level of detail in the future, but most certainly not at the beginning of a KG project. First, having a URI uniquely identifying a physical business document is enough to build its related knowledge in an iterative process. We further explore and illustrate this modelling approach based on multiple classifications in chapter 5.

In section 3.1.2, we mentioned the domain knowledge most common examples of modelling units, dimensions, time and domain-specific terms. For the latter examples, some working groups have been developing standard ontologies defined in OWL that can be reused. For instance, the OWL-Time ontology is an OWL-2 DL ontology of temporal concepts for describing the temporal properties of resources in the world or on Web pages. It provides a vocabulary for expressing facts about topological (ordering) relations among instants and intervals, information about durations, and temporal position, including date-time information [CL22]. Furthermore, the Quan-

[4] https://spacy.io/ (Accessed on Thursday 3rd October, 2024)

[5] https://www.ottr.xyz/#Lutra (Accessed on Thursday 3rd October, 2024)

tities, Units, Dimensions and Types (QUDT) Schema defines the base classes properties and restrictions for modelling physical quantities, units of measure, and their dimensions in various measurement systems [HMP+24]. Other examples are the Financial Industry Business Ontology (FIBO)[6] in the financial domain and the collection of ontologies known as the Open Biological and Biomedical Ontology (OBO)[7] in the biology domain.

3.2.5 Knowledge provenance

The W3C (World Wide Web Consortium) Provenance Working Group's definition of provenance is a record that describes the people, institutions, entities, and activities involved in producing, influencing or delivering a piece of data or a thing [LM13].

The PROV standard includes PROV-O [LSM+13], an OWL2 ontology with all the building blocks to describe provenance information. Among the concepts is the notion of activity, which concerns any processes involved in data production, such as extracted data from a Relational DataBase Management System (RDBMS) or generating triples based on data extracted from text.

To be as complete as possible in the expression of data provenance, we consider other languages with mapping to the RDF data model. RML[8] [DVSC+14] and its standardized extension for RDBMS, R2RML [DSC12], let us state explicitly mappings between data stored in any database and the corresponding RDF triples to generate. For triples directly generated from text analysis processes, we are exploring a recent effort to formalize Ontology Design Patterns [GPSS09] using RDF, namely Reasonable Ontology Templates (OTTR) [SLKF18]. The latter template language lets us explicitly describe macros for triple creation. On the knowledge acquisition side, we find the corresponding systems built specifically for processing these languages: an RML engine for RML or R2RML mappings and *Lutra* for OTTR templates.

3.2.6 Knowledge validation

KGBS maintenance, particularly the KG, requires constant updates and validation. The processes involved in tackling such tasks are very similar to techniques we use in software development when considering software as a product. Hence, in the data management industry, some advocate for treating data as a product [Pat12] and work on adapting continuous integration and development ideas to data. We see industry communities terms such as *dataOps*[9] emerging on the data analytics side. On the knowledge representation communities, the term *SemOps*[10], the contraction of *Semantics* and *DevOps*, is emerging.

OTTR templates contribute to the KG quality by consistently generating triples. To apply a continuous development and integration approach to data production, we need to validate the data structure generated by diverse processes. It is also critical for data consumption as applications rely on this latter structure to function. The industry has been working on the latter issue, and a recent W3C effort led to a standard, the SHape Constraint Language (SHACL) [KK17]. We can also mention the Shape Expression (ShEx) language[11], which has a goal similar to SHACL. However, both technologies solve the problem from different perspectives and formalisms [GPB17].

It would be optimistic to believe that the knowledge validation processes can all be performed automatically. It is critical to involve and engage human experts to ensure quality maintenance. The study of solutions to validate RDF data structures involving humans is out of the scope of our works. However, some related topics are ontology verbalization, Natural Language Generation (NLG) and chatbots. We could imagine generating natural language sentences from KG content,

[6] https://github.com/edmcouncil/fibo (Accessed on Thursday 3rd October, 2024)

[7] https://obofoundry.org/ (Accessed on Thursday 3rd October, 2024)

[8] https://rml.io/specs/rml/ (Accessed on Thursday 3rd October, 2024)

[9] https://medium.com/data-ops/dataops-is-not-just-devops-for-data-6e03083157b7 (Accessed on Thursday 3rd October, 2024)

[10] https://www.semanticarts.com/the-data-centric-revolution-the-role-of-semops-part-1/ (Accessed on Thursday 3rd October, 2024)

[11] https://shex.io/ (Accessed on Thursday 3rd October, 2024)

e.g., combining techniques employed in ontology verbalization and NLG and using the result to engage experts in the validation process through a chatbot conversation.

Section 3.1.3 mentions the difference between data and semantic validation, which we also discuss in section 1.2.4. We address the former with either SHACL or ShEx and a suitable processor. We consider the latter by using the OWL2 languages alongside a reasoner to spot any logical inconsistencies. RDFS is a simple ontology language and does not contain the logical notion of negation. As such, RDFS reasoning cannot create or reveal inconsistencies. Semantic validation requires the expressivity of OWL. However, some of this validation can also be performed by employing rules encoded in SWRL or SPARQL queries.

3.3 Knowledge Graph-Based System architecture for Information Retrieval

To illustrate our argument, we consider a KG-based approach to IR. Figure 3.4 presents an overview of a KG-based IR system's main components and processes. The KG is the central component around which the other system components revolve. The knowledge acquired and modelled from various knowledge sources feeds the KG. Users express their information needs with a query that can take various forms, as discussed in chapter 2. The IR system processes the latter query, possibly applying multiple transformation tasks. The resulting processed query is used to retrieve and rank documents. The latter documents are indexed based on the KG concepts, i.e., semantically indexed.

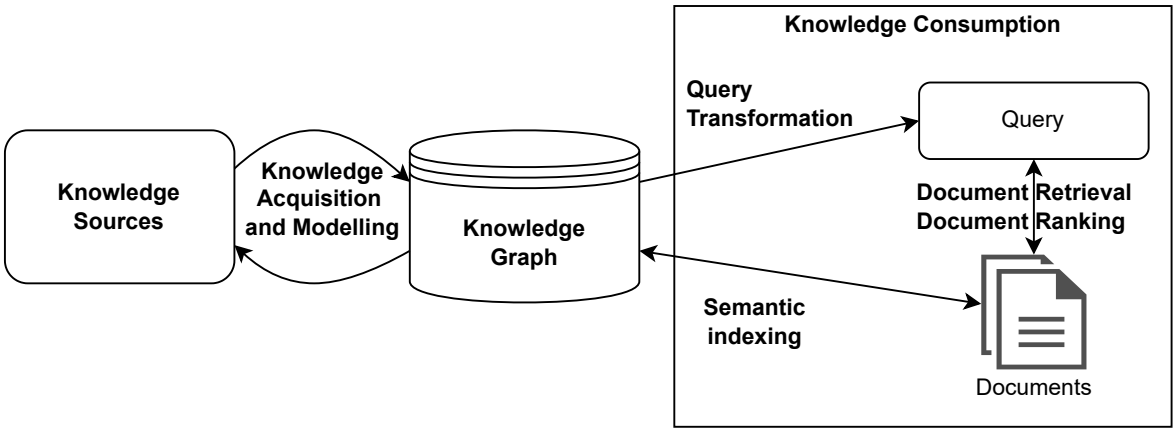


Figure 3.4: Knowledge Graph-Based System architecture applied to an IR systems.

Square boxes and bold labels including a verb refer to activities. Round boxes and the symbols such as the database and documents ones, depict containers. Arrows illustrate data flows.

In figure 3.2, we abstract the specific IR use case in the knowledge consumption component presented in section 3.1.3. For instance, we might extract the domain and business knowledge parts of the KG's domain graph and store them in a specialised database to index documents based on them. Such a system could be a NoSQL (Not Only SQL) database such as Elasticsearch[12], which integrates document indexing, retrieval, and ranking. This engine also includes solutions for textual query processing. We can also leverage an EIR system to extract the concepts expressed in the text. Such an EIR system can benefit query transformation and document indexing tasks. It enables us to structure the query and documents by introducing the KG concepts and entities they contain. We can then use a SPARQL engine to query over the domain knowledge and enrich the query and documents. The KG must be as complete and clean as possible to ensure the enrich-

[12]https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html (Accessed on Thursday 3rd October, 2024)

ment task is the most accurate possible. The KG validation processes introduced in section 3.1.3 and the processes and technologies mentioned in section 3.2.6 are critical in this endeavour.

Knowledge is not static; neither should the KG. The knowledge acquisition processes depicted in section 3.1.1 and their candidate implementations technologies proposed in section 3.2.3 are essential to any KGBS, regardless of the knowledge consumption use case. In figure 3.4, the knowledge sources are the same as discussed in sections 3.1.4 and 3.2.3. They could be anything ranging from the outcome of brainstorming sessions with domain experts to a piece of code encoding a business logic. Each source might require its specific knowledge acquisition approach.

3.4 Conclusion

This first contribution chapter proposes an architecture for KG-based systems to understand better the KG's role and integration within an information system. Understanding such a broader KG integration scope is essential before diving into the specifics of some use cases. The chapter introduces the architecture and its components. Though explored from an abstract point of view, we discuss the latter components in great detail. In particular, we break down the knowledge acquisition phase, often referred to as one activity in the literature, into knowledge elicitation, extraction and production. We also further explore the KG. Chapter 1 defines KGs from a structural perspective. This chapter additionally explores the KG from a content perspective, breaking it down into the domain and business knowledge, structural schema, and data provenance.

This chapter commences with a discussion of theoretical principles, presenting the architecture from an abstract point of view. We then offer practical solutions, advocating for candidate technologies for our architecture components. We endorse the W3C standards and demonstrate that there is an RDF-based standard for each architecture component. Furthermore, we delve into the associated tools for each language, highlighting their practicality and applicability.

Our work provides an approach to building a KG-based system. Existing research should also discuss applying its methods to an industrial environment. Hence, we explore candidate solutions considering business constraints. The KG-based system architecture combined with our definition of a KG is another step toward better understanding KGs' versatile industrial usages and applications. However, we recognise the knowledge acquisition bottleneck before using a KG and investigate solutions in the following chapter.

Finally, figure 3.5 reuses the KGBS architecture figure to introduce the following chapters. Large red boxes denote the components each following chapter addresses. The architecture parts turned into dashed and light grey or blue are the parts this manuscript is leaving as future work.

We follow figure 3.5 to present our remaining contributions. Chapter 4 addresses the knowledge acquisition bottleneck by proposing and discussing our ontology learning framework and its implementation as an open-source Python library. Chapter 5 gently dives into applying KGs to IR by designing an OWL-powered IR system. The latter IR system demonstration also illustrates an implementation of the KG definition introduced in chapter 1. Before concluding this manuscript, chapter 6 discusses the practical industrial experiments we have implemented as part of this thesis partnership with TraceParts. In these experiments, we aim to update an existing text-based search engine to a KG-based one.

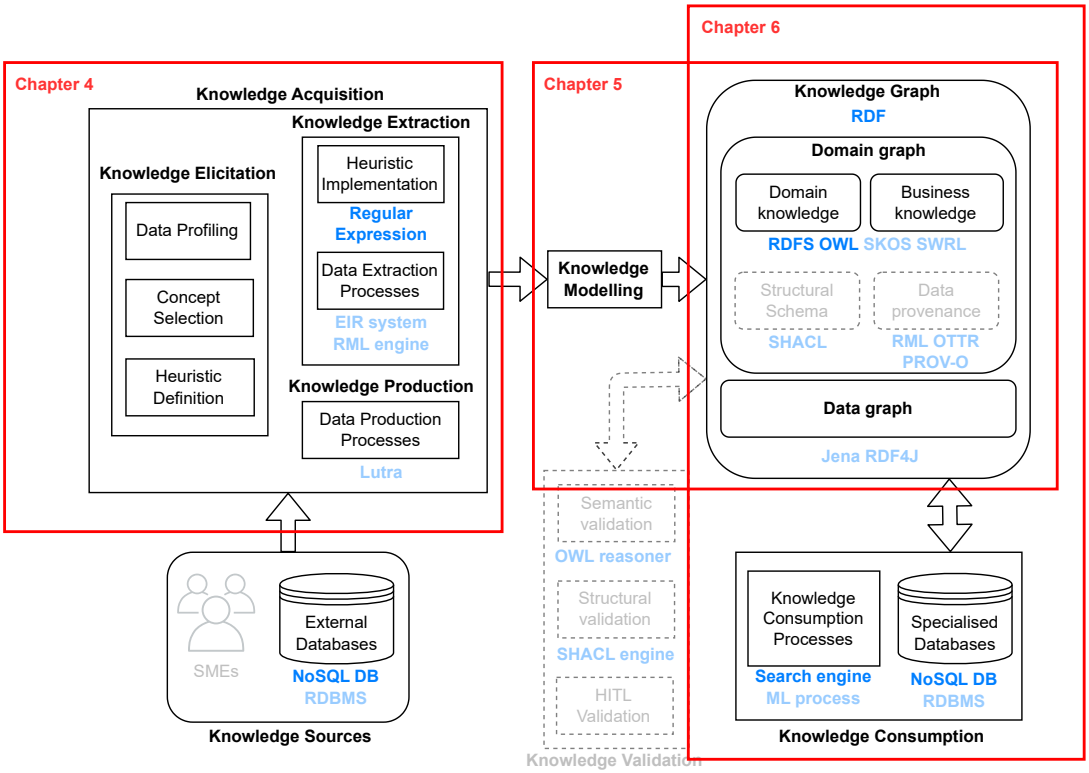


Figure 3.5: Knowledge Graph-Based System architecture introducing the following chapters focus.

Square boxes refer to activities. Round boxes depict containers. The boxes imbrications denote sub-activities and sub-containers, respectively. Arrows illustrate data flows. Large red boxes denote the components each following chapter addresses. Some candidate technologies for implementing each component are mentioned in bold blue letters. The architecture parts turned into dashed and light grey or blue are the parts this manuscript is leaving as future work. (SMEs: Subject Matter Experts; HITL: Human In The Loop)

Chapter 4

Ontology Learning Applied Framework

Contents

In the previous chapter, we have introduced an operational architecture for KG-Based Systems (KGBSs). We structure our work around this architecture. This chapter focuses on the knowledge acquisition challenges as depicted in figure 4.1 reproducing our KGBS architecture.

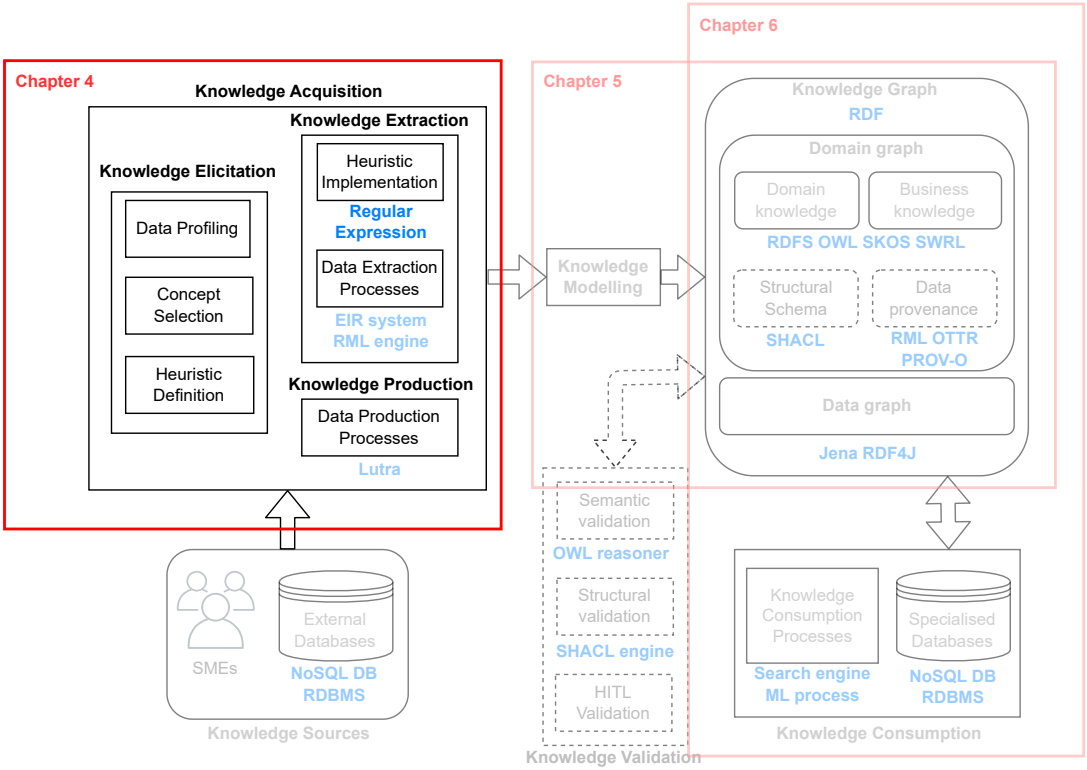


Figure 4.1: Knowledge Graph-Based System architecture. Focus on chapter 4.

Square boxes refer to activities. Round boxes depict containers. The boxes imbrications denote sub-activities and sub-containers, respectively. Arrows illustrate data flows. Large red boxes denote the components each chapter addresses. Some candidate technologies for implementing each component are mentioned in bold blue letters. The architecture parts turned into dashed and light color are the parts left out in this chapter. (SMEs: Subject Matter Experts; HITL: Human In The Loop)

We developed the work we introduce in this chapter in collaboration with a PhD student colleague of the LITIS laboratory MIND team. Both she and us work on KGBSs with different applications. Her use case is about conversational systems, and ours is about IR. We both work with text content and need to construct a KG, specifically, an ontology. However, we do not have access to Subject Matter Experts (SMEs) to help us design our respective domain graphs. Hence, we joined forces to work on Ontology Learning (OL) from text. Her expertise is in NLP and Large Language Models[1] (LLMs). Ours is toward knowledge engineering. She developed all the LLM-based components. I focused on developing the axiom extraction components and all the serialisation aspects to extract the learned ontologies using the Semantic Web technologies. We combined our implementations and worked together on designing a complete OL framework to integrate our approaches.

[1]LLMs are outside the scope of this work. Hence, we stay voluntarily vague. However, we define LLMs as a language model notable for its large parameter number and its ability to achieve general-purpose language generation and other natural language processing tasks (https://en.wikipedia.org/wiki/Large_language_model). When we mention LLMs in this work unless specified we are more specifically considering generative LLMs such as GPT-4 (https://openai.com/research/gpt-4).

Our collaboration led to the implementation of an OL from text framework. We propose the Ontology Learning Applied Framework (OLAF) and implement it in the form of an open-source Python library[2]text.

The literature review in section 1.4.3 presents multiple OL techniques. Each author establishes a different combination of tasks leading to the creation of an ontology. However, OL is a task that still presents substantial challenges [KAG21].

Automation is a limitation as it often affects the quality of the learned ontology. The lack of acceptance for uncertainty can be the reason for retaining human involvement in OL tasks. Our work focuses on OL approaches with a minimum degree of Human-In-The-Loop (HITL). We aim to fully automatically construct an ontology that is good enough to support an existing system. We call such an ontology a Minimal Viable Ontology (MVO). However, this MVO needs enhancements in future project iterations. OLAF considers techniques spanning from data pre-processing to axiom definition. We allow for more noise to justify a higher level of automation.

Another critical limitation is that existing OL approaches focus more on the ontology than its application use case. However, the usage intent influences the ontology structure and must, therefore, be taken into account during the OL process. In our work, we consistently keep in mind the KG's targetted main application, even though a KG is a central knowledge artefact that should be able to serve many different applications.

As mentioned in [ZGH11], it is essential to check that the proposed framework is domain-independent. The fact that an ontology is influenced by the system in which it is integrated should not prevent the development of a generic creation method. Most existing OL approaches impose non-iterative rigid step orders and a restrictive combination of algorithms. Therefore, we built OLAF for different types of applications, combining sub-tasks modularly.

In the remaining sections of this chapter, we introduce our framework and its architecture, discussing each component and task. We then explore two experiments we ran to evaluate OLAF and conclude by exploring future works considering OLAF within a broader KGBS context.

4.1 Ontology Learning Applied Framework

This section details the structure of our framework OLAF. Before diving in, we specify some vocabulary we use. We then explore OLAF components with their possible approaches and the data containers we define to store each knowledge extraction step outcome.

4.1.1 Terminology

The literature on OL defines synonyms as words or terms denoting the same concept. The associated terms are often found in external data sources, e.g., WordNet [Mil95] or ConceptNet [SCH16]. We found the name *synonym* misleading as it tends to convey the idea that those words should have precisely the same meaning. However, in practice, systems use synonyms to match a concept in text content. We call linguistic realisations such textual representations of a concept. We could link a concept to multiple linguistic realisations that are close in meaning depending on the use case and its knowledge granularity requirement. Hence, we use the name *term enrichment*. For example, "*ontology*", "*taxonomy*" and "*thesaurus*" could be added as synonymous linguistic realisations for the concept "*knowledge graph*". However, some applications might require a refined granularity and consider "*knowledge graph*" and "*ontology*" as synonyms for a KG concept, separately from "*taxonomy*" and "*thesaurus*" that could be associated with a *controlled vocabulary* concept.

The definition of a concept is controversial. Thus, the concept extraction process differs from one method to another, depending on the definition used and the use case envisioned for the ontology. We consider concepts a notion of a chosen granularity with related terms as linguistic

[2]https://github.com/wikit-ai/olaf (Accessed on Thursday 3rd October, 2024)

realisations. For instance, the concept *knowledge graph* might have "*knowledge graph*" and "*ontology*" as terms, i.e., linguistic realisations.

Relations enable us to define and make explicit links between known concepts. Whereas the literature often distinguishes taxonomic and non-taxonomic relations, we introduce relations and metarelations. In our framework, relations are transversal connections whose labels are candidate terms extracted from the source text corpus. Metarelations are deduced from linguistic or statistical information whose labels are manually predefined. Some common examples of such manually labelled relations are generalisation relations such as *broader/narrower*, generic association relations such as *related to*, and mereological relations such as *part of*. The automatic extraction of transversal relations is largely underdeveloped in the OL literature, which mainly focuses on metarelations.

Finally, we distinguish between the ontology resulting from the OL processes and the language employed to represent it, e.g., OWL. Thus, we will denote the OL process result as the Knowledge Representation (KR). The KR is a set of concepts, relations, and metarelations. Now that we have introduced our vocabulary let us dive into the framework architecture.

4.1.2 Framework architecture

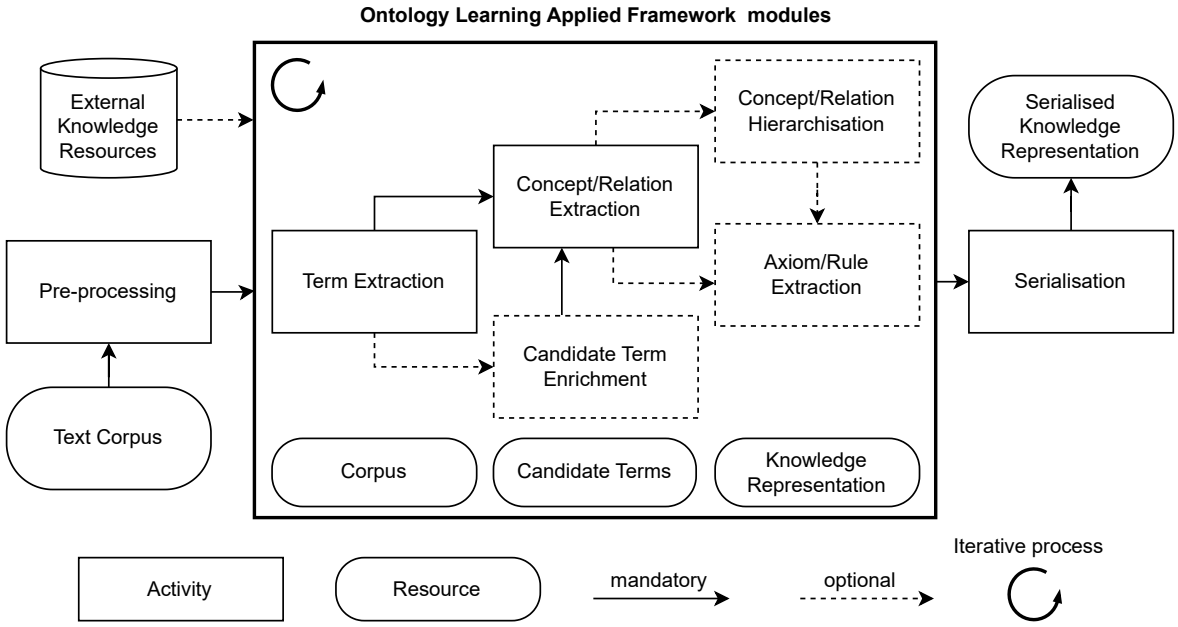


Figure 4.2: Ontology Learning Applied Framework components.

Square boxes refer to activities. Round boxes represent containers. Arrows illustrate data flows between activities and components.

Figure 4.2 presents the architecture of our framework OLAF. To ease the user learning curve, the framework components follow the OL layer cake (1.4.2). However, we break down the procedure into different steps, enable an iterative process, and adapt the names as presented in the previous section. Unlike the meaning conveyed by the ontology learning layer cake, the processes are neither all mandatory nor should they be executed in a determined order. However, the framework implicitly defines some constraints.

Overview

Reading figure 4.2 from left to right, the OL process begins with a corpus of text documents first handled employing various NLP methods. The pre-processing module is necessarily the first one in an OL pipeline. Its methods start from a raw set of texts and turn them into structured content.

The corpus is the data container holding such structured content, i.e., any information related to the corpus data.

The central box labelled *Ontology Learning Applied Framework modules* is the framework heart. The modules use the corpus to learn the final KR depicted on the right side of the figure (*Serialised Knowledge Representation*). The term extraction module extracts candidate terms before concepts and relations since the concept and relation extraction module is based on them. Candidate term enrichment is an optional step that adds information fetched from external knowledge resources, such as synonyms and other linguistically linked texts. The external resources are depicted at the top left of the figure. Any OL module methods might use them. The concept and relation extraction module feeds the KR with concepts or relations inferred from the candidate terms. The hierarchisation and axiom extraction modules can be used to enrich the KR. Hierarchisation enables ordering existing concepts and/or relations to learn taxonomies. Axiom extraction infers rules encoding the concepts and relations' meaning. Each process introduced in the OLAF modules of figure 4.2 can be repeated indefinitely. Knowledge constantly evolves, and so should its computed representations.

Data containers

We distinguish between two worlds living in parallel: the conceptual one and the linguistic one. In the conceptual one, candidate terms are the terms of interest extracted from the text content. They are used to extract concepts and relations. In the linguistic one, we create linguistic realisation from candidate terms during concepts and relations extraction. Ultimately, the OL objective is to reconcile and align the linguistic world with the conceptual one. Hence, the resulting KR is the data container for the data constituting the knowledge. Let us now review each component.

Corpus The corpus is a data structure holding all the corpus text information. Examples of such information are the words in the text and all the tags categorising them. Such tags can categorise single words or groups of words, e.g., Part-Of-Speech (POS) tags or named entity ones. They can also qualify the relation between two words, such as the subject, the verb or the adjectives in a sentence. As we focus on OL from text, the corpus results from the initial text analysis performed by various NLP techniques. The corpus typically structures the text content in a non-destructive manner, i.e., we can always reconstruct the original documents from the corpus. The corpus is critical since all processes rely directly or indirectly on its structure. The better the text is analysed and structured, the better the knowledge extraction.

Candidate Term The candidate term container holds metadata about the corpus elements of interest. Candidate terms are the first building blocks for extracting concepts and relations at the linguistic level. In the next OL steps, they will become linguistic realisations of the concepts and relations.

Linguistic Realisation As mentioned in this section's introduction, we distinguish the linguistic world from the conceptual one. Linguistic realisations are the links between the corpus texts and the knowledge representation's concepts and relations. They are the final form of the candidate terms and hold any linguistic information, such as the tense. Linguistic realisations are all the forms a concept or relation can take within a text. They can come directly from the corpus, i.e., the candidate terms, and from external resources such as a controlled vocabulary.

Concept Concepts result from candidate terms and are the first pieces of formalised knowledge. They are entities in the KG domain graph (Cf section 1.5.3) or classes in an ontology. Linguistic realisations materialise concepts in the corpus.

Relation From a practical point of view, relations are the links in our KG. In OLAF, we distinguish relations from metarelations. We will explore metarelations next. In contrast to metarelations, relations can be singletons, pairs, or triples, i.e., a relation optionally has a source and/or a destination concept and must have a relation kind. We already discussed concepts. Source and destination concepts come from the corpus. In the case of a relation, its kind directly stems from the corpus. As such, it is like a concept. The relations are derived from candidate terms, which become the relation's linguistic realisations. Finally, a relation can be drawn from some candidate terms without any relation to some concepts, i.e., it is a particular case of a concept. Singleton relations typically occur when extracting them based on POS tags. A verb becomes a relation even though the system has not been able to establish the subject, for instance.

Metarelation While relations are drawn from the corpus, metarelations are predefined, linking two concepts without necessarily having a relation's linguistic realisation. The metarelation kind is manually defined and is not drawn from the corpus. Examples of such metarelations are hierarchical, association and mereological ones. Hierarchical metarelations organise concepts and relations hierarchically in a taxonomical manner. A typical example is an RDFS class hierarchy using the *rdfs:subclassOf* property [BG14]. More abstract hierarchical metarelations not involving explicit typing are the ones defined by the SKOS standard [IS09], *skos:broader* and *skos:narrower*. Association metarelations are abstract, only specifying an unknown or not explicit relation between two concepts, e.g., *skos:related* in SKOS. Finally, mereological relations, a.k.a. parthood relations, describe relations of part to whole and the relations of part to part within a whole[3].

Knowledge Representation The KR is the data structure grouping the concepts, relations, metarelations, and other information to describe the learned ontology. It is typically empty at the beginning of the OL pipeline and is filled by the various processes. However, we can begin the OL process with a seed ontology to enrich and specify. Then, the initial KR container holds the seed ontology.

Knowledge Representation serialisation The KR must be converted into a form interpretable by dedicated systems to be used in an application as an ontology. This is the role of the serialisation module. The KR is an abstract representation specific to OLAF. The knowledge it represents can be serialised in any representation language and model. In our work, we serialise the KR to an OWL ontology expressed as RDF triples. However, we can also serialise the KR to an LPG-based modelling language.

External knowledge resources External knowledge resources can take various forms. The most prominent ones are linguistic resources and OKGs. Linguistic resources can be explicit, such as WordNet [Mil95], or implicit, such as LLMs and their contextualised word vector representations. We can leverage OKGs such as Wikidata [VK14] and ConceptNet [SCH16].

Modules methods

We have expanded the discussion on OLAF data containers. Let us now discuss the methods we can implement to tackle each module. Many of them are listed in [AWK+18], in which the authors present possible methods for each OL task, distinguishing them between linguistic and statistical approaches. Rather than reproducing the list, we discuss below the methods we propose so far in our OLAF implementation[4]

[3] https://plato.stanford.edu/entries/mereology/ (Accessed on Thursday 3rd October, 2024)

[4] https://github.com/wikit-ai/olaf (Accessed on Thursday 3rd October, 2024)

To implement our framework, we use Python 3 and base the corpus pre-processing on spaCy[5]. We choose Python as it eases access to the vast Python community and its library ecosystem, particularly NLP tools and numerous machine learning libraries.

Pre-processing We have based our OLAF implementation on Python 3 and spaCy, so we leverage the spaCy design for data pre-processing. This choice lets anyone implement their text processing pipeline based on spaCy and plug it into OLAF. We base the implementation of the methods for OLAF modules on the spaCy data structure. Hence, our corpus data container implementation is a list of spaCy documents.

Term extraction For term extraction, we implement pattern-matching approaches from POS tags and scored-based methods based on occurrences, the C-value algorithm [FAM00], and TF-IDF.

Candidate Term enrichment Candidate terms are the points of interest we extract from our corpus. Before deriving concepts and relations from them, we can enrich them with external knowledge resources. Those external knowledge resources are typically OKGs or manually curated linguistic resources. We match the candidate term in the external resources and fetch the valuable resource content for the OL use case.

Concept and Relation extraction Concept and relation extraction is typically rule-based. We can rely solely on the information derived from the corpus or trust external resources. For instance, we can use rules such as considering a concept every candidate term tagged as a noun and a relation each one tagged as a verb. Alternatively, we can trust an OKG such as ConceptNet and consider that a candidate term having a match in ConceptNet is a concept or a relation. We also extract concepts by grouping candidate terms based on the extracted linguistic relations. For metarelations specifically, we also implement extraction based on term cooccurrences.

Concept and Relation hierarchisation To organise concepts and relations hierarchically, various statistical methods based on term cooccurrences and forms have been explored in the literature. We implement term subsumption [FG04] and hierarchical clustering[6] approaches.

Axiom and rule extraction The extraction of rules or axioms provides meaning to the KR at a higher level of abstraction. This task depends on the level of axiomatisation the ontology application requires. Axiom extraction remains challenging and little addressed in the literature. Methods are often rule-based, and only Inductive Logic Programming approaches show potential. These approaches inductively infer axioms based on positive and negative examples. The axioms are optimised to satisfy as many positive examples as possible while not satisfying negative ones.

However, axioms truly distinguish a graph of data from a KG. We extended OLAF with a rule-based axiom extraction approach based on Ontology Design Patterns (OPDs) [GPSS09]. We assume an ontology is constructed with a purpose, directly impacting the kind of axioms to define. For our OLAF implementation, we decided to rely on the OWL language. The OLAF OWL axiom extraction component constructs an OWL RDF graph from the previously extracted concepts and relations. The user provides functions generating OWL RDF triples. Some straightforward examples of such OWL axiom generator functions are creating each KR concept as an OWL class and each KR relation as an object property. With the same spirit as the ODPs, we can create particular OWL constructs based on the KR.

The OLAF OWL axiom extractor component also checks for the generated OWL ontology's logical consistency. The process generates the full OWL RDF graph and runs a reasoner. It stops if

[5] https://spacy.io/ (Accessed on Thursday 3rd October, 2024)

[6] https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering (Accessed on Thursday 3rd October, 2024)

the reasoner finds no logical inconsistency or unsatisfiable classes. Based on whether a logical inconsistency or an unsatisfiable class is found, the OLAF OWL axiom extraction process iteratively prunes axioms until a consistent ontology is reached.

Serialisation As mentioned when introducing the KR serialisation, KR is an abstract representation specific to OLAF that needs to be serialised into a dedicated format using a particular language. This rule-based process maps OLAF data containers with the serialisation language and possible structures. It is worth noting that not all languages will allow the same kind of expressivity. For instance, we can express things in OWL that we can not express with RDFS. The KR serialisation can also be merged with the axiom extraction as the latter requires the use of a knowledge modelling language.

4.2 Experiments and results

We tested our OLAF implementation with two versions. We first learn an ontology from a corpus of Schneider Electric product descriptions. Based on this experiment, we updated the framework and its implementation. At the beginning of this chapter, we described the most recent version of our framework. More recently, with our research team colleague, we explored using LLMs for OL. We implement a demonstration with the second version of OLAF, which we augment with an LLM-based version of each component.

Here, we do not discuss the pertinence of LLMs for OL, as LLMs are outside the scope of our work. Instead, we present and discuss the experiments in the order in which we implement them.

4.2.1 Schneider Electric experiment

This section introduces and discusses our first experiment with OLAF. Our objective was naturally to learn an ontology from text. Nevertheless, we also aimed to test the functioning of our OLAF implementation and its modularity. Though we conduct this experiment with an older version of our OLAF implementation, the code is still available on our GitLab[7]

Corpus

For the Schneider Electric experiment, we use a collection of 10,000 Schneider Electric product descriptions from the Schneider Electric website[8] (Accessed on Thursday 3rd October, 2024). This corpus is a small extract of the data we are working with in this PhD IR use case. We aim to learn an ontology to support the KG-based IR system we introduce in chapter 6.

Ontology Learning pipeline

Several iterations, optimising the OLAF pipeline, lead to the approaches for each OLAF component presented in figure 4.3. We use the *C-value* algorithm [FAM00] for term extraction. We then enrich the candidate terms using WordNet [Mil95] with a filter on the domains under interest using the work of the *Fondazione Bruno Kessler*[9]. We fetch the WordNet domains alignments from the Argilla[10] project *spacy-wordnet*[11]. To infer concepts, we group the candidate terms based on their synonyms. To organise concepts, we build hierarchies using a Subsumption approach [FG04] and extract relations based on concept cooccurrences. We construct axioms following the approach described in section 4.1.2.

[7] https://gitlab.insa-rouen.fr/msesboue/ontology-learning/-/tree/fois_2023_paper_code/demonstrators/fois_2023/schneider_electric_pipeline (Accessed on Thursday 3rd October, 2024)

[8] https://www.se.com/ww/en/work/products/master-ranges/

[9] https://wndomains.fbk.eu/ (Accessed on Thursday 3rd October, 2024)

[10] https://argilla.io/

[11] https://github.com/argilla-io/spacy-wordnet (Accessed on Thursday 3rd October, 2024)

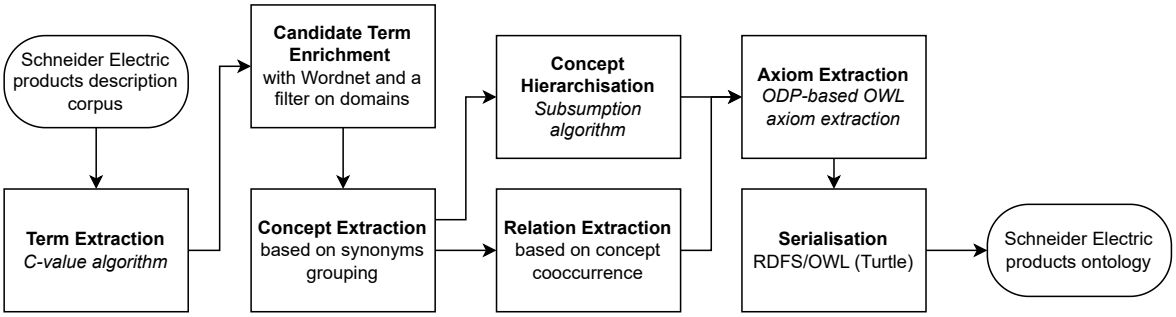


Figure 4.3: Ontology Learning pipeline for the Schneider Electric products corpus.

Square boxes refer to activities. Round boxes represent containers. Arrows illustrate data flows between activities and components.

Evaluation methodology

We do not have access to an existing Schneider Electric products ontology, so we primarily rely on ourselves as domain experts for the ontology evaluation. To perform an evaluation as qualitatively as possible, we follow and apply the CQ approach. As described in [EFK19], CQs validate whether an ontology defines enough knowledge to answer questions on the relevant application domain. In this experiment, we restrict our evaluation to a sample of the CQs we now present.

By default, we expect to find the main concepts mentioned in the one-sentence descriptions of each product range, e.g., pushbutton, relay, and tower light for the Harmony Schneider product range. Besides the latter obvious concepts, we define below a sample of the CQs:

Q1 Which are the Tesys products?

Q2 Which are the different kinds of control units?

Q3 What concepts are related to a specific product range?

Q4 Are protection devices part of the Easy9 product range?

Q5 Is a handle part of a switch?

Results

The learned ontology contains 68 concepts and 495 metarelations. However, it can only partially answer questions Q1 and Q3. The learned ontology contains relevant concepts but drowns in noise. Different cases cause this situation. Too many relations or metarelations can create noise, especially hierarchical and association (*related*) ones, which are based on concepts' cooccurrences and are threshold-dependent. This is the case for Q3, where the ontology defines multiple associations. Though there are many noisy relations, the learned model shows concepts such as *residual current* and *miniature breaker* related to *Easy9*, which is a product range for residential consumer unit and circuit protection[12]. Regarding Q1, a concept exists for *Tesys*, which is related to concepts denoting devices and having a linguistic realisation with the term "*Tesys*". Cleaning up the noisy relations would require human intervention and can hardly be automated. The noisy relations case corresponds to high recall and low precision.

The learned ontology can answer Q2. However, we can not ensure completeness for this kind of open question. It is the opposite of the previous noisy relation case, i.e., low recall and high precision.

The learned ontology needs to define more concepts to answer Q4 and Q5. This lack of defined knowledge often originates in term or concept extraction. For instance, regarding Q4, the learned ontology does not define any *protection device* concept. The extraction or enrichment steps did

[12]https://www.se.com/uk/en/product-range/63125-easy9- (Accessed on Thursday 3rd October, 2024)

not extract such candidate terms. Such a case can occur when the term extraction algorithm does not score the terms well or when the terms do not appear explicitly in the text. The latter case is a major challenge. Traditional statistical-based approaches rely on the concept explicitly appearing in the corpus. Such a challenge is one potential reason for missing knowledge to answer Q5.

This first experiment with OLAF shows the usability of our framework and its implementation. However, at the time, we needed to implement more methods. This experiment also underlines the importance of corpus preprocessing. Indeed, though other approaches might better handle missing concepts and noisy relations, structuring the corpus is still essential. Most methods will rely on some NLP task results such as POS tagging and Named Entity Recognition (Cf section 2.2.4). In our case, technical product descriptions are too different from classical news text. Hence, the out-of-the-box NLP systems did not perform well on our corpus.

4.2.2 Pizza Ontology experiment

Our first experiment lacks a proper reference ontology to compare our learned one. In practice, though it is easy to find a corpus or an ontology, it is challenging to find both a corpus of text and its corresponding manually engineered ontology. We want to compare our results with a reference ontology for our second experiment. Hence, we decide to use the Pizza Ontology[13] and leverage an LLM to generate example texts from it. All the codes for the experiment we describe below and the ones with LLMs are available on GitHub[14].

Corpus and reference ontology

We consider the Pizza Ontology to be our reference ontology. It has been created for an OWL tutorial with the Protégé software[15] and introduces basic pizza concepts such as ingredients, pizza categories, and the most popular pizzas. We use an LLM to generate descriptive text about the Pizza Ontology.

We extract the Pizza Ontology's RDFS labels as a list of strings, removing the OWL constructs. Removing such OWL constructs is essential to avoid bias in the LLM-generated texts. The RDFS labels list feeds the prompt context to generate the textual descriptions. The prompt instructs the LLM to generate a text describing pizzas containing all the labels. We use the *gpt4* OpenAI model[16] and set the temperature to 0. LLMs are not deterministic, so we performed several runs with the same prompt. The text obtained was identical for 5 successive generations. Each paragraph is a document from the corpus. We present below the corpus which comprises 10 documents with an average of 48 words each.

> Pizza is a popular dish of Italian origin, consisting of a usually round, flattened base of leavened wheat-based dough topped with tomatoes, cheese, and often various other ingredients, which is then baked at a high temperature, traditionally in a wood-fired oven.
>
> There are two main types of pizza bases: the Deep Pan Base, which is thick and doughy, and the Thin And Crispy Base, which, as the name suggests, is thin and crispy. The type of base used can significantly alter the pizza's overall taste and texture.
>
> Pizza toppings are incredibly diverse and can be categorized into several types. Cheese Topping is a staple on most pizzas, with Mozzarella Topping being the most common. The Four Cheeses Topping, used in Quattro Formaggi Pizza, typically includes mozzarella, gorgonzola, parmesan, and a fourth cheese like fontina or ricotta.
>
> Meat Topping is popular on Non Vegetarian Pizza, with options ranging from Chicken Topping to Parma Ham Topping and Peperoni Sausage Topping. The Meaty Pizza is a favorite among meat lovers, often loaded with various meats.

[13] https://github.com/owlcs/pizza-ontology/ (Accessed on Thursday 3rd October, 2024)

[14] https://github.com/wikit-ai/olaf-llm-eswc2024 (Accessed on Thursday 3rd October, 2024)

[15] https://protege.stanford.edu/ (Accessed on Thursday 3rd October, 2024)

[16] https://openai.com/research/gpt-4 (Accessed on Thursday 3rd October, 2024)

Vegetable Topping is a staple on Vegetarian Pizza. Options include Artichoke Topping, Asparagus Topping, Green Pepper Topping, Leek Topping, Mushroom Topping, Olive Topping, Onion Topping, and Spinach Topping. The Giardiniera Pizza is a well-known vegetarian pizza loaded with Cheesy Vegetable Topping.

Seafood Topping is another category, with options like Prawns Topping and Mixed Seafood Topping. The Frutti Di Mare Pizza is a well-known seafood pizza.

There are also pizzas known for their spiciness, like the American Hot Pizza and the Cajun Pizza, which feature Hot Green Pepper Topping, Hot Spiced Beef Topping, Jalapeno Pepper Topping, and Cajun Spice Topping.

Some pizzas are known for their unique toppings. The Capricciosa Pizza, for example, features a mix of ham, mushrooms, artichokes, and olives. The Napoletana Pizza is topped with anchovies, capers, and olives. The Margherita Pizza is a simple yet beloved pizza topped with tomatoes, mozzarella, and basil.

There are also pizzas with unique combinations of toppings, like the Pollo Ad Astra Pizza, which features chicken, sweet peppers, and red onions, or the Sloppy Giuseppe Pizza, which is topped with hot spiced beef, green peppers, and red onions.

In conclusion, pizza is a versatile dish with a wide variety of bases and toppings to cater to every palate. Whether you prefer a cheesy, meaty, vegetarian, or spicy pizza, there's a pizza out there for everyone.

Ontology Learning pipeline

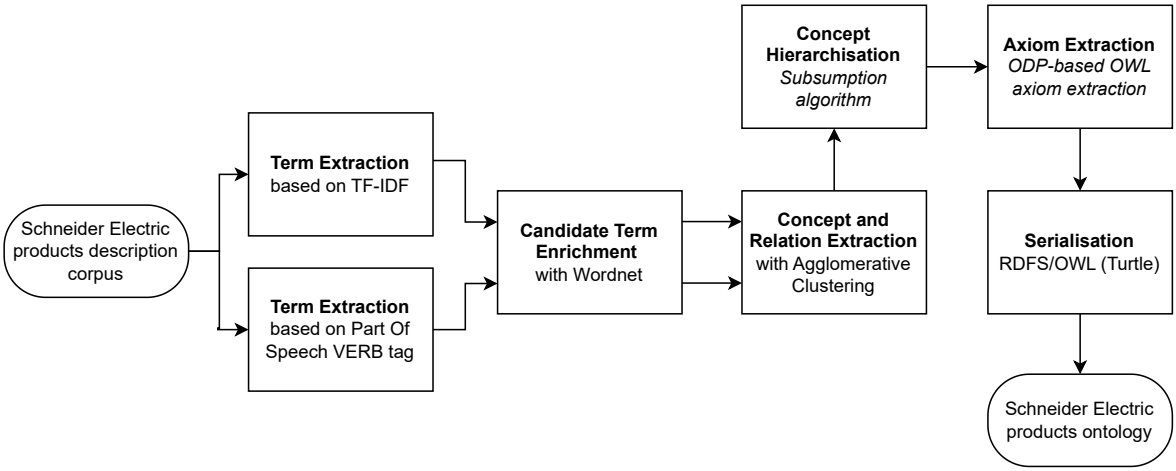


Figure 4.4: Ontology Learning pipeline for the Pizza Ontology corpus.

Square boxes refer to activities. Round boxes represent containers. Arrows illustrate data flows between activities and components.

This experiment is part of a larger one to evaluate the pertinence of LLMs for OL. In this broader work, we also compare the LLM-based pipeline with a no-LLM one. As LLMs are out of the scope of our work, we only explore the OLAF no-LLM pipeline presented in figure 4.4. The candidate terms extraction uses TF-IDF [Jon21] scores with a threshold. We enrich these candidate terms using WordNet [Mil95]. They are then grouped as concepts with Agglomerative Clustering (AC) [ZMRA13] and the sentence-t5-base Sentence Transformer embedding model [NHAC+22]. We compute hierarchies based on the Subsumption algorithm [FG04]. The relation candidate terms are extracted from VERB POS tags, enriched using WordNet, and grouped into relations with AC. OWL axioms are extracted with the approach described in section 4.1.2.

Evaluation methodology

For this second experiment, we evaluate the ontology manually by comparing it with the original Pizza ontology. We first characterise the learned ontology with a quantitative analysis. We then manually align the ontologies and compute a precision, recall and F1-score for the classes, individuals, object properties and *rdfs:subclassOf* relations. Choosing whether a concept should be an OWL class or a named individual is often tied to design choices related to the ontology usage. Hence, we also combine the classes and individuals to compute precision, recall, and F1-score without distinguishing between classes and individuals.

Results

Table 4.1 presents the learned ontology OWL axiom counts. The learned classes and properties counts are close to those of the Pizza Ontology. However, the learned ontology creates many named individuals and *rdfs:subclassOf* relations.

Counts	Pizza Ontology	OLAF no LLM
OWL named classes	97	111
OWL object properties	8	22
OWL named individuals	5	343
RDFS *subClassOf* tuples	141	390

Table 4.1: Ontologies' OWL axioms counts grouped by kinds.

Many learned named individuals are sensible since our OLAF pipeline creates them based on linguistic realisations. The term extraction and enrichment processes might extract too many terms. However, the number of OWL classes close to the Pizza ontology one shows that the AC approach for concept extraction performs well. Indeed, our pipeline constructs OWL classes from concepts and makes their linguistic realisations instances of the concept class. The good named individuals recall in table 4.2 suggests that relevant individuals are learnt, though many noisy ones are also. However, this is not true in practice since table 4.1 the Pizza Ontology contains only 5 named individuals. Hence, it is easy to reach a high recall value.

Metrics	OLAF no LLM
Classes precision	0.387
Classes recall	0.453
Classes f1-score	0.417
Individuals precision	0.006
Individuals recall	0.400
Individuals f1-score	0.012
Classes and individuals precision	0.130
Classes and individuals recall	0.415
Classes and individuals f1-score	0.198
Object properties precision	0.136
Object properties recall	0.375
Object properties f1-score	0.200
SubClassOf pairs precision	0.012
SubClassOf pairs recall	0.023
SubClassOf pairs f1-score	0.015

Table 4.2: Results of aligning the Pizza Ontology with the learned one.

We can similarly explain the large number of *rdfs:subClassOf* relations by the amount of extracted terms. Indeed, the term Subsumption algorithm [FG04] uses concept cooccurrences in

the corpus to infer subsumption relations and the concepts are matched in the corpus by their linguistic realisations. Hence, the large number of individuals suggests many concept corpus occurrences. Since we have a small corpus, there is also limited examples for the algorithm to generalise.

The number of classes and object properties in table 4.1 are sensible with the values of table 4.2. The precision values suggest noisy concepts and relations. However, the higher recall values show that the OL process extracts most of the expected ones.

4.3 Conclusion and future works

This chapter introduces OLAF, our approach to OL from text. We developed our framework with a research team colleague and implemented our version as an open-source Python library. The literature review in section 1.4.3 displays the limitations of previous attempts. In addition to getting older, many approaches were developed as tools to help knowledge engineers in their work. As such, the human expert is a central part of the system. Hence, we build OLAF with full automation in mind and consider the targetted application, i.e., the KGBS. OLAF addresses the knowledge acquisition component of our KGBS architecture introduced in chapter 3. Moreover, state-of-the-art OL approaches consider only one task or particular task sequences. Hence, our framework is modular and follows the OL layer cake (Cf section 1.4.2). We develop our OLAF Python implementation as an open-source project so anyone can add and compare their approaches. We aim to gather feedback and grow a community to develop and test multiple algorithms. Various satellite tools could also be developed to enhance the implementation of the framework.

The above sections discuss each OLAF component and task before discussing two experiments. The first experiment primarily aimed at demonstrating the usability of our OL framework. We relied on ourselves as domain experts for the learned ontology evaluation. The second experiment is part of a larger one investigating using LLMs for OL. Though we do not explore OLAF LLM implementation, our second experiment uses an existing ontology as a reference for our evaluation. Together, the two experiments demonstrate OLAF usability and modularity. However, they point out the limitations of the approaches we implement so far. Hence, one area of focus for future work is implementing other methods for each OL task. In particular, we explore the pertinence of LLMs for each OL task. The experiments also emphasise how critical the corpus preprocessing quality is for the OL result quality.

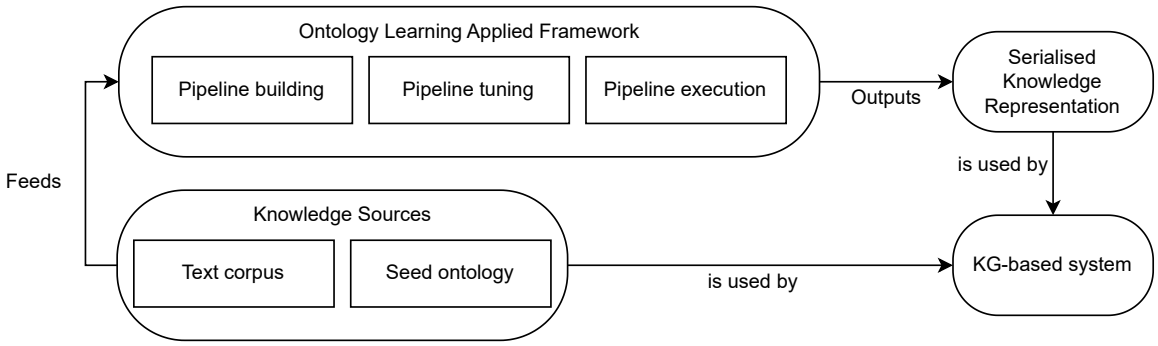


Figure 4.5: Ontology Learning Applied Framework within a broader Knowledge Graph-based system.

Square boxes refer to activities. Round boxes represent containers. Arrows illustrate data flows between components.

Before closing this chapter, consider how OLAF could fit into a larger system. Figure 4.5 presents an example of how OLAF can interact with other components of a KGBS. We consider the processes of building, fine-tuning, and executing an OL pipeline part of OLAF. Such processes correspond to our experiments in sections 4.2.1 and 4.2.2. In our OLAF implementation, though we can fully automatically perform the pipeline execution, the pipeline construction and tuning still require some human effort. Providing we have some example data for each task, we could imagine

automatically tuning the selected methods for each OL task by adjusting their parameters. Going one step further, we could even test each available approach for each task and select the best one according to our example data. However, finding example data for each OL task is challenging.

In figure 4.5, we begin with a text corpus at the bottom left that feeds the OL pipeline. The pipeline constructs a MVO which supports a KGBS. As this system provides feedback on the MVO and the KGBS use case evolves, the MVO can serve as a seed ontology to feed the OL pipeline along with the updated text corpus. Thus, we ideally need to have a self-enhancing OL process. Moreover, an ontology should be flexible enough to address different applications. However, a new KGBS leveraging an existing learned ontology might require some knowledge missing in the existing ontology. Then, the ontology can serve as seed ontology to be extended using OLAF with a different corpus.

Though we aim to minimise human involvement requirements in designing OLAF, humans can intervene at any step of the OL process. We can implement OLAF with Human-in-the-Loop processes (HITL) at any stage. As we discussed in this chapter introduction, the learned MVO aims to be good enough to begin supporting the targetted KGBS.

Chapter 5

Knowledge modelling for Information Retrieval

" In the database management system world, the schema language and the data language are often separated. We have a Data Definition Language and the Data Management Language. Weirdly, they are separate but inseparable. The data can not exist without its schema. In Semantics, both are more similar than different. It is all triples. But they are separable. One can apply the different points of view, i.e., Tbox, over the same set of instances, i.e., Abox. "

McComb D.

Contents

The literature review on KG-based IR in section 2.3 introduces multiple examples of ontologies and whole KGs used to support IR systems by providing specific vocabularies and structured content. However, we did not find any ontology constructed to power the IR system operations. OWL reasoning-based systems rarely rely on a real-time reasoning requirement. Often, only the reasoning process result, i.e., the inferred facts, is used online. This chapter introduces the Information Retrieval ontology (IR ontology), an attempt to define a Description Logics-powered IR system. Hence, this chapter focuses on the knowledge modelling part of our KGBS architecture (chapter 3) as depicted in figure 5.1.

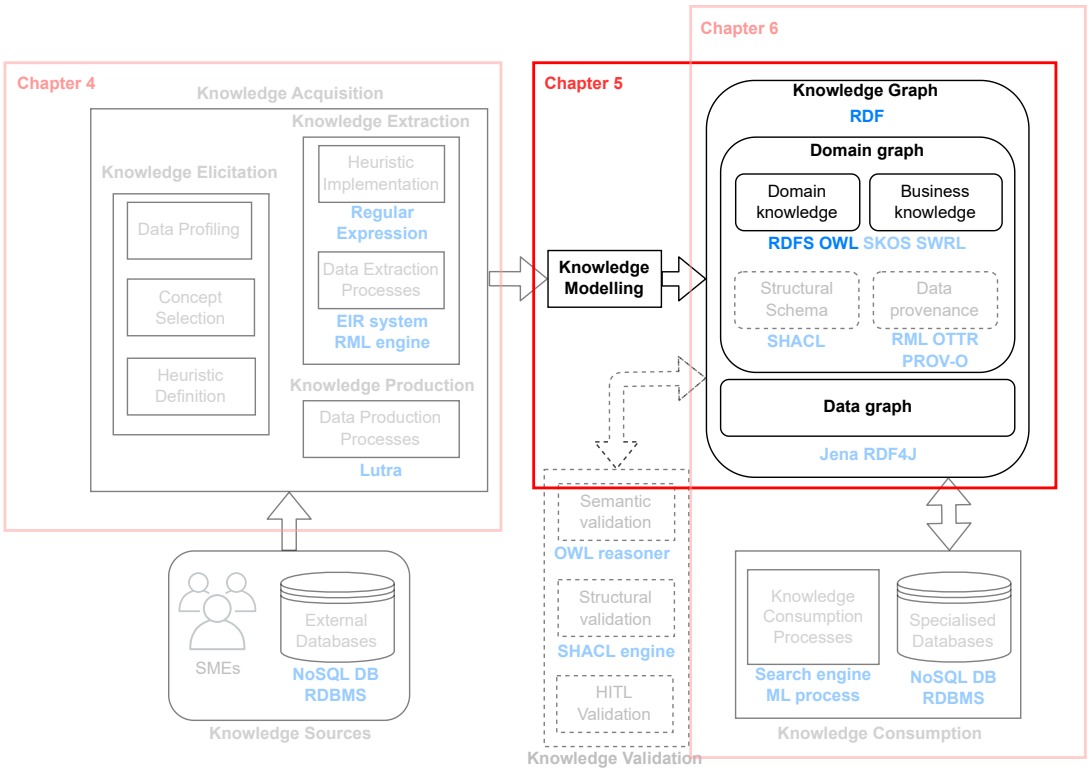


Figure 5.1: Knowledge Graph-Based System architecture. Focus on chapter 5.

Square boxes refer to activities. Round boxes depict containers. The boxes imbrications denote sub-activities and sub-containers, respectively. Arrows illustrate data flows. Large red boxes denote the components each chapter addresses. Some candidate technologies for implementing each component are mentioned in bold blue letters. The architecture parts turned into dashed and light color are the parts left out in this chapter. (SMEs: Subject Matter Experts; HITL: Human In The Loop)

Before diving into the ontology description, we introduce some background concepts tracing the lineage of the ideas we are extending and applying. We then review the ontology engineering process's main components and describe our IR ontology. Putting the theory into practice, we propose a step-by-step demonstration of how to use the ontology. The demonstration also proposes an implementation of the figure 1.7 KG definition. Finally, this chapter explores the advantages and limitations of our IR ontology before concluding with an extension of the ontology.

5.1 Background and literature review

The IR ontology results from different theoretical and practical knowledge management and ontology engineering ideas, combined to meet an IR purpose. We first explore an ontology design practice introduced in the context of OWL modelling as the *Cbox*. We then discuss an ontology

modelling example presented in [AHG20] to demonstrate the power of OWL reasoning. The IR ontology is directly derived from this example.

5.1.1 The *Cbox* modelling approach

In the Gist Council of March 2019[1], Dave McComb explores an aspect of the Gist ontology[2] design he coined as the *Cbox*. Gist is an OWL minimal ontology focusing on the enterprise domain. The video introduces and showcases the Category box (*Cbox*) in relation to the well-known Assertions box (*Abox*) and Terminology box (*Tbox*) (sometimes even the *Rbox* for Relations or Roles box). This section is a summary of the core ideas presented in this video.

When modelling knowledge with a Description Logics (DLs) language, the terms *Tbox*, *Abox*, *Rbox* are commonly used to separate different kinds of axioms. DLs are a structured fragment of First Order Logic. The *Tbox* groups the axioms defining the terms, i.e., the concepts in an ontology and their relations. The *Abox* groups axioms asserting facts. It is often simplified as the set of OWL named individuals. The *Rbox* considers axioms defining the relations and how they relate. The *Rbox* is often considered part of the *Tbox*.

From a DLs perspective, the *Cbox* corresponds to a portion of the ontology spanning over both a part of the *Tbox* and one of the *Abox*. The *Cbox* regroups the concepts of categories and their instances. Hence, the *Cbox* top concept could be a named class *Category*. In the Gist ontology, it is the named class *gist:Category*. A category is a concept or label used to categorise other instances informally. Things that can be thought of as types are usually Categories[3]. One could have a category *Gender* (an OWL named class) with at least two instances that are *_men* and *_women*. The *Cbox* is then composed of the *Tbox* formed by the class *Category* and its subclass *Gender*, and the *Gender* class instances *_men* and *_women*.

From an enterprise knowledge management perspective, according to Dave McComb, it is a scale issue. An ontology should contain, at most, a few hundred core concepts. Using his own words, "it is typically something you can take home with you over the weekend and learn by heart". Moreover, it should be managed by a team of ontologists. The categories are in the amount of a few thousand. They are managed by domain experts (SMEs) responsible for maintaining their sets of categories or classifications. Finally, the data, i.e., instances organised by categories are in the amount of millions. Such scale requires automated processes. Data product owners should manage them, a concept aligned with the data mesh idea not explored in this work. The interested reader can read the book of the same title by Zhamak Dehghani[Deh22]. Katariina Kari, lead ontologist at Inter IKEA Systems, supports this enterprise knowledge management perspective. She discusses the topic in several interviews and in a blog post *IKEA's Knowledge Graph and Why It Has Three Layers*[4]. In the latter blog post, she summarises the *Cbox* idea and its application in a diagram we reproduce in figure 5.2 and adjust to our KG definition.

In her figure, Katariina Kari uses the term KG, while in figure 5.2, we specifically mention the domain graph. Indeed, categories and the ontology are part of the domain graph. The data layer might be thought of as the data graph. However, our domain graph definition 1.5.3 mentions: "The terms in the data graph are also present in the domain graph. In a KG, a mapping $\cdot^{\mathcal{I}}$ exists between the terms in the data graph and the ones in the domain graph." Hence, the data layer depicted in figure 5.2 is part of the domain graph.

The *Cbox* is the home of taxonomies, thesauri, and any classification system we defined in chapter 1.2. They are typically built and managed by software tools different from ontology ones. For Dave McComb, this modelling approach addresses two common pitfalls in ontology engineering. He names the first one the taxonomy first design, which corresponds to building a giant tree

[1] The Gist Council is a monthly meeting discussing the Gist ontology: https://www.youtube.com/watch?v=0-j9nWFVoYc (March 2019 recording accessed on Thursday 3rd October, 2024)

[2] https://www.semanticarts.com/gist/ (Accessed on Thursday 3rd October, 2024)

[3] *skos:definition* of *gist:Category* in Gist 12.0.1.

[4] https://medium.com/flat-pack-tech/ikeas-knowledge-graph-and-why-it-has-three-layers-a38fca436349 (Accessed on Thursday 3rd October, 2024)

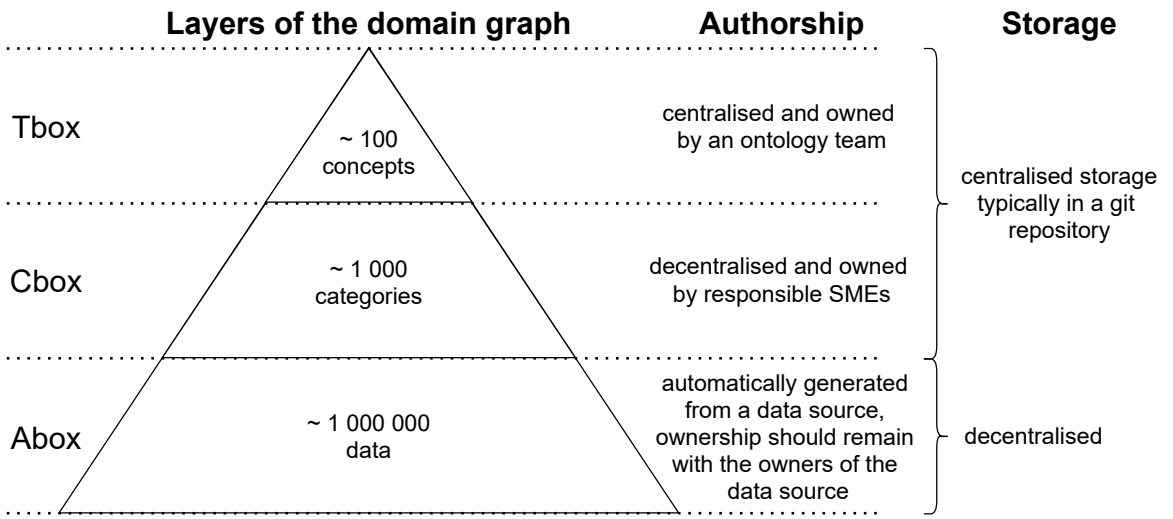


Figure 5.2: The domain graph broken down into the *Tbox*, *Cbox* and *Abox* layers.

in which everything in the considered domain should fit somewhere. The second pitfall is the traditional object-oriented design applied to ontology engineering. Ontology engineering beginners tend to create a new class each time they learn about new distinctions between things in the domain. In a subsequent section introducing the reasoning patterns used in the IR ontology, we will explore Dave McComb's approach to coping with such pitfalls. However, let us first present the OWL modelling example from which we derive the IR ontology.

5.1.2 An OWL modelling example

In the third edition of the book *Semantic Web for the Working Ontologist* [AHG20], Allemang et al. present a running example about managing questions and answers in a questionnaire solely powered by OWL. This example largely inspires the IR ontology reasoning constructs. Here, we describe the questionnaire structure and will dedicate a subsequent section to the reasoning patterns applied to the IR ontology.

The example Allemang et al. use is about a series of questions asked as part of the screening for the helpdesk of a cable television and internet provider. There is no correct answer but rather a question path constructed based on previous questions' answers. Some questions should be enabled only if specific answers have previously been selected.

This screening questionnaire example presents OWL constructs enabling inferences of the kinds:

- If a question is linked to a selected answer, then the question is an answered question.
- Question 1 is enabled only if answer A or B have been selected.

We could ensure these inferences by leveraging SPARQL CONSTRUCT queries and SWRL rules. However, the example's idea is that explicitly describing the domain knowledge is enough to meet questionnaire logic requirements. While using SPARQL or SWRL is arguably a more straightforward way to address the problem, it is using a shortcut and not exploiting the source modelling language, i.e., OWL, to its maximum potential. It also requires learning another language and maintaining a rule set on top of the ontology. The IR ontology leverages the OWL knowledge modelling language as much as possible to meet its requirements.

Now that we have introduced enough background information let us dive into the IR ontology description.

5.2 Information Retrieval ontology

Before we delve into the detailed definitions of the IR ontology concepts, let us introduce our vocabulary.

First, we make the distinction between a concept and a category. In [Mur10], Gregory L. Murphy distinguishes between concepts and categories with the following definition: "A concept is a mental representation of a class (e.g., skunks, liberals), which includes what we know about such things. A category is the set of examples picked out by a concept." In the IR ontology, a category is a concept. The categories classifying documents are the instances of this category concept. Among all the categories, some are selected, and others are enabled. In natural language, the selected categories should represent the search. The enabled categories are the categories available to refine the search, i.e., we can select an enabled category to refine the search.

Documents in the IR ontology are defined similarly as in definition 2.2.2. They are the things forming the search space. Categories categorise documents. Some of those documents might suit the search intent. They are candidate documents.

In the IR ontology, we also distinguish between searches we know of and the ones we are currently considering. The former searches are searches the system is aware of, but does not consider at the moment. The latter search is called the search context and should start the reasoning process.

We now introduce the IR ontology. Building upon the previous sections, we design this ontology to move as much knowledge as possible closer to the data. Following the ontology engineering practices presented in [EFK19], we first introduce some targeted use cases and define the competency questions. We then dive into the classes and relations defined in the IR ontology. We conclude with a detailed demonstration.

5.2.1 Use cases

The IR ontology aims to power a search engine. Various practical ways to search for documents in a corpus exist. The most straightforward way is to use free text. Nowadays, it is often the entry point into a search process. Another approach is to browse the corpus by navigating one or more classifications.

The initial search lets the system propose a set of categories corresponding to the user's search intent. The latter categories serve as entry points to the classification browsing process. Each category is linked to its children by a hierarchical relation. Those categories categorise documents. Therefore, the first IR ontology use case determines the documents corresponding to a user search expressed as a set of categories. Those documents are called candidate documents, and the user search categories are selected categories.

Expanding on the first use case, we consider each category linked to others by hierarchical and transversal relations. We can use these relations to refine the user search. Here, we refer to transversal relations in opposition to hierarchical ones. The user selecting the first set of categories initiates a back-and-forth between the system and himself. It is the classification browsing process, in some particular cases, called faceted search. A second use case for the IR ontology is powering a faceted search. Once a user selects some categories, the ontology should let the system know the categories enabled to refine the user search.

5.2.2 Competency questions

The knowledge representation community defines Competency Questions (CQ) as the set of questions that a KG must be able to answer correctly. The latter definition comes from [EFK19] we expand to KGs. In [EFK19], they describe competency questions, mentioning both the questions and their answers. CQs' goal is to ensure the ontology has enough background knowledge to fit the intended use cases. Here, we are considering a core ontology that is generic enough to fit any

IR retrieval system. We should extend such core ontology for custom applications. It encapsulates the core logic to use in any peculiar IR system. Hence, we can not pre-define some specific expected answers for the CQs. However, we can expect a kind of answer derived from expected behaviours.

In this section, we are focusing on an IR ontology. Hence, we restrict the competency questions to validating the domain graph. We define the following 3 competency questions:

CQ1 What are the categories in the user search?

CQ2 What are the documents relevant to a search?

CQ3 What categories are enabled to refine the search?

Here, we consider the IR ontology and leave out the user search preprocessing that might be involved when integrated into a production system, e.g., an NLP process, such as entity linking to tag categories mentioned in the search. Hence, we consider the user search processed and inserted as named instances triples. We will see some examples in section 5.3. CQ1 ensures that categories included in a search are considered selected.

CQ2 directly stems from the first use case. At any given time, the system should be able to query the ontology for the documents relevant to a user search. From a practical standpoint, CQ2 is one of the two main queries involved in the back-and-forth between the IR system and the ontology.

CQ3 is then the second such query. It stems from the second use case and enables a proper logic-based faceted search or classifications browsing process.

As mentioned, we can not define specific expected answers to these CQs since we focus on a core ontology. However, we can expect some behaviours we will express as if/then rules.

Regarding CQ1, we expect the following:

- If a category is used in a search, it should be selected.
- If a category is selected, then all its subcategories should also be selected.

We refer to documents relevant to a search as candidate documents. Hence, regarding CQ2, we expect the following:

- If a category is selected, all the documents categorised by this category are candidate documents.
- If a category is selected, all the documents categorised by any of the subcategories of this selected category are candidate documents.

Finally, regarding CQ3, when looking for categories to refine the search, we expect the following:

- If a category is selected, all the categories related to this selected category by an enabling relation are enabled categories.

Above, we express the expected behaviours of the IR ontology as if/then rules. Hence, it might make sense to use a rule language such as SWRL to implement them. However, we can translate those requirements using a knowledge modelling language such as OWL. The following section introduces the reasoning patterns used to realise such an endeavour.

5.2.3 Reasoning Patterns

Reasoning patterns are very similar to Ontology Design Patterns (ODPs). An ODP is a modelling solution to solve a recurrent ontology design problem. They do not depend on any specific representation language [GPSS09]. However, our notion of reasoning patterns is modelling language-dependent. Our patterns are OWL-based and focus on the reasoning results they should trigger. Hence, among the different ODPs the literature identifies, they should match the *Logical* or *Reasoning* OPDs. The debate of which ODP kind the reasoning patterns we present here belongs to is out of this work scope. Conforming the IR ontology to an ontology engineering methodology such as EXtreme Design [PDGB09] is left out as future work. Hence, we use the term *reasoning pattern* in this work.

This section describes the reasoning patterns implemented in the IR ontology and the purpose they pursue. However, before focusing on the ontology, let us introduce the reasoning pattern Dave McComb uses to motivate the *Cbox* ontology modelling approach.

Motivating the *Cbox* modelling approach

In section 5.1.1 introducing the *Cbox* idea, we mention some modelling pitfalls presented in Dave McComb's Gist council presentation. His presentation explores the implication of creating either a category or a class when encountering new distinctions between domain individuals. The conclusion is that turning the class into a category instance involves much refactoring if we create a class. Meanwhile, creating an instance of a category and then deciding to make it a class can be handled gently by creating the category class twin with a clever definition to leverage the OWL reasoning power.

The modelling methodology described by Dave McComb is as follows. When encountering a new distinction, first ask:

- Will the system need to infer membership in this class based on some knowledge about an instance?
- Does the system need to enforce different properties about this class?

If the answers to these questions are yes, then we should create a class with a suitable definition. However, if the answers are no or not immediately yes, we should create a category instance.

The point of this approach is that we can change our mind later. If it turns out we need to say more about a category, enforcing some logical constraints, then we can do the following:

1. Create a class twin of the category with the needed definition.
2. Make this class equivalent to anything categorised by the category instance.

In OWL terms, it would mean creating the following we express using the Manchester syntax [HPS12]:

```
Class: MyCategoryClass EquivalentTo(categorisedBy value _myCategory)
```

In natural language, we define the category twin class as equivalent to anything categorised by the category. That way, there is no need for any refactoring. The OWL reasoner will infer all instances related to the category by a relation *categorised by* of the category class twin type.

Information Retrieval ontology reasoning patterns

We design the IR ontology around one main reasoning pattern applied to the different top classes. This reasoning pattern powers the IR process by being applied in a sequence. [AHG20] describes the pattern with much detail and illustrates it with the questionnaire management example. Here,

we describe the reasoning pattern with generic names and illustrate it in the next section with the definitions of the IR ontology classes. We encourage the interested reader to study [AHG20] for extensive and illustrated examples. In this work, we will refer to this reasoning pattern as the *role-based range* pattern since the idea is to infer class membership of a relation range based on the role.

First, we create some OWL classes, say classes *A* and *B*, and an object property, say *A2B*, without defining any domain and range. Then, we also create a subclass of *B*, say *C*. The heart of the reasoning pattern is hidden in the definition of class *A*. Expressed as a rule, what we want the reasoning pattern to achieve is: "if an instance of class *A* is related to an instance of class *B* by the property *A2B*, then the instance of class *B* is an instance of subclass *C*." Figure 5.3 depicts our definition of the role-based range pattern based on [AHG20].

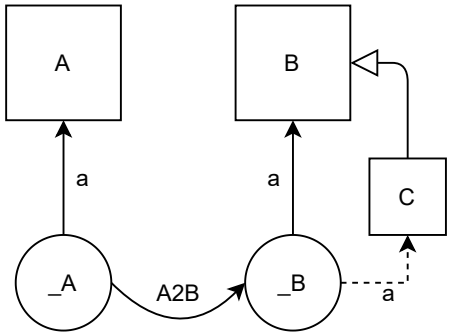


Figure 5.3: The role-based range reasoning pattern.

Squares are classes; circles denote instances, solid lines represent asserted facts, and the dashed arrow is the expected inference. We use the Turtle syntax convention "a" to denote *rdf:type*. The empty head arrow corresponds to a subclass relationship.

We also leverage one of the simple part-whole patterns presented in [RWNW05], the first representation pattern "Representing part-whole for individuals". However, we do not apply the pattern in a part-whole relation setting but rather in a hierarchical one. Nonetheless, the objective is the same: we want to ensure the transitivity of the primary relation while still being able to distinguish the other parts from the ones directly related to the considered part.

The role-based range and part-whole reasoning patterns are enough to understand the IR ontology. Let us now dive into the formal definitions of its classes and object properties.

5.2.4 Formal definitions

So far, we have described the IR ontology rather informally. In this section, we dive into the formal definitions of each class and object properties. We first introduce the object properties before detailing the classes that are the reasoning power's heart. We conclude with an overview of the reasoning process powering a search engine. The following section illustrates the IR ontology usage.

The IR ontology is small enough to copy its turtle serialisation here and directly study it. It defines 34 axioms, among which 7 classes and 6 object properties. To simplify the visualisation, let us introduce the namespace definitions here.

```
@prefix : <http://www.msesboue.org/o/ir-ontology#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

We will use the above namespaces to present the IR ontology in the next subsections. The default namespace is *http://www.msesboue.org/o/ir-ontology#*. It means that a term *:Category* effectively corresponds to *http://www.msesboue.org/o/ir-ontology#Category*. The prefix *rdf* is to be

expended to *http://www.w3.org/1999/02/22-rdf-syntax-ns#*. So *rdf:type* is effectively *http://www.w3.org/1999/02/22-rdf-syntax-ns#type*.

Object properties

The IR ontology defines the following object properties:

```
:categorisedBy rdf:type owl:ObjectProperty ;
               owl:inverseOf :categorises ;
               rdfs:range :Category .

:categorises rdf:type owl:ObjectProperty .

:hasSearchCategory rdf:type owl:ObjectProperty ;
                   rdfs:range :Category .

:enablesCategory rdf:type owl:ObjectProperty ;
                 rdfs:domain :Category ;
                 rdfs:range :Category .

:hasSubcategory rdf:type owl:ObjectProperty ;
                rdfs:subPropertyOf :enablesCategory ;
                rdf:type owl:TransitiveProperty .

:hasDirectSubcategory rdf:type owl:ObjectProperty ;
                      rdfs:subPropertyOf :hasSubcategory .
```

The *:categorisedBy* and *:categorises* object properties relate documents to their categories. When extending the IR ontology with domain-specific data, we should ensure any relation used to categorise the documents in our domain is defined as a subproperty of either *:categorisedBy* or *:categorises*. For instance, if we organise some products based on their shape, then the hypothetical relation *hasShape* linking a product to its shape should be defined as a subproperty of *categorisedBy*.

The IR ontology defines the *:hasSearchCategory* object property to link a search instance with its categories. When defining classes, we leverage this property to drive the entailment: if a category is one of a search, then it is a selected one.

The *:enablesCategory* object property links categories together, indicating that the target category is enabled if the source category is selected. We specify the latter entailment when defining classes. It might make sense for some systems to declare the *:enablesCategory* property symmetric. However, in the IR ontology, we include the hierarchical relation *:hasSubcategory* defined as a subproperty of *:enablesCategory*. In practice, it ensures that its subcategories are enabled when a category is selected. However, if we were to make the *hasSubcategory* symmetric, it would entail that if a category is selected, its parent category would be enabled. It is not a desired behaviour for the IR ontology.

When specifying the ontology, we should extend the *:enablesCategory* property with any relations between categories in our domain denoting a dependence. For instance, in the ingredient classification search we will explore in our demonstration, if we select the cheese category, we should expect the goat cheese and the vegetarian categories to be enabled. Hence, we should assert the hierarchical relation between the cheese and goat cheese categories either as equivalent to the IR ontology *:hasSubcategory* property or as a subproperty of either *:hasSubcategory* or *:enablesCategory*. We should also assert a transversal relation between the cheese and vegetarian categories as a subproperty of *:enablesCategory*.

We just introduced the *:hasSubcategory* object property defined as a subproperty of *:enablesCategory*. It means that if a property *:hasSubcategory* between instances *a* and *b* is asserted, then the property *:enablesCategory* will be inferred.

:hasSubcategory is constructed using the part-whole reasoning pattern. The property is made transitive, i.e., if some properties *:hasSubcategory* are asserted between instances *a* and *b* and between *b* and *c*, then the same property between *a* and *c* will be inferred. In practice, it ensures the reasoning process links all the direct or indirect subcategories to its parent category. However, such inference also loses the information about the direct subcategories. Hence, following the part-whole reasoning pattern, the IR ontology defines a non-transitive property *:hasDirectSubcategory* as a subproperty of *:hasSubcategory*. In practice, while the reasoner will infer all the *:hasSubcategory* relations, the system can still query for the direct subcategory using the *:hasDirectSubcategory* property. For instance, it could let a program traverse the taxonomy following the category levels one by one while leveraging some inferred facts.

This section introduced the object properties the IR ontology defines. They enable defining the logic of the classes driving the reasoning-powered IR system. Let us now define the latter classes.

Classes

The IR ontology defines the following classes:

```
:Document rdf:type owl:Class .

:CandidateDocument rdf:type owl:Class ;
                rdfs:subClassOf :Document .

:Category rdf:type owl:Class .

:EnabledCategory rdf:type owl:Class ;
                 rdfs:subClassOf :Category .

:SelectedCategory rdf:type owl:Class ;
        rdfs:subClassOf :Category ,
                        [ rdf:type owl:Restriction ;
                            owl:onProperty :categorises ;
                            owl:allValuesFrom :CandidateDocument
                        ] ,
                        [ rdf:type owl:Restriction ;
                            owl:onProperty :enablesCategory ;
                            owl:allValuesFrom :EnabledCategory
                        ] ,
                        [ rdf:type owl:Restriction ;
                            owl:onProperty :hasSubcategory ;
                            owl:allValuesFrom :SelectedCategory
                        ] .

:Search rdf:type owl:Class .

:SearchContext rdf:type owl:Class ;
            rdfs:subClassOf :Search ,
                            [ rdf:type owl:Restriction ;
                                owl:onProperty :hasSearchCategory ;
                                owl:allValuesFrom :SelectedCategory
                            ] .
```

We construct the IR ontology to have a reasoner perform a classification task so that an IR system can leverage the result. In practice, the latter system queries only for instances of some classes. Hence, the ontology defines some top-level classes. Initially, all the domain instances are instances of those top-level classes. The IR ontology defines some subclasses of these classes. The IR system leveraging the IR ontology focuses on those subclasses. The reasoning process aims to specialise some top-level instances by inferring them of type those subclasses. We will illustrate this process in the next section. However, let us first introduce the formal definitions of the IR ontology classes.

At the top level, the IR ontology defines 3 classes: *:Document*, *:Category* and *:Search*. They are, respectively, the things the IR system lets us look for, the things organising the search space, i.e., classifying documents and the search queries' representations. Without any inference, all instances are asserted in one of those classes. The following definitions sound similar to the vocabulary introduction of section 5.2. They are the DLs' definitions of the terms introduced in this section.

Some of the instances of class *:Document* are candidate documents. Hence, a subclass of *Document* is *:CandidateDocument*. The ontology-based IR system typically queries for instances of *:CandidateDocument* to fetch the user search results.

Instances of the *:Category* class are broken down into 2 subclasses. Some instances are *:EnabledCategory*. The latter class groups instances of *:Category* that we can use to refine the search. The ontology-based IR system typically queries for those instances to power the facetted search user interface. Among the other instances, some are *:SelectedCategory*.

The IR ontology defines class *:SelectedCategory* using a combination of the role-based range pattern. It defines a big part of the IR ontology functioning and aims at ensuring the behaviours we previously defined in section 5.2.2 with if/then rules as:

A: If a category is selected, all the documents categorised by this selected category are candidate documents.

B: If a category is selected, all the categories related to this selected category by an enabling relation are enabled categories.

C: If a category is selected, all subcategories are also selected.

The following anonymous classes represent the above rules:

A: `Class: SelectedCategory SubclassOf(categorises only CandidateDocument)`

B: `Class: SelectedCategory SubclassOf(enablesCategory only EnabledCategory)`

C: `Class: SelectedCategory SubclassOf(hasSubcategory only SelectedCategory)`

The ontology defines the *:SelectedCategory* class as a subclass of the intersection of all those 3 anonymous classes that are object property restrictions and the *:Category* class. In plain English, we define the selected categories as the categories that are:

- related with the *:categorises* object property to only candidate documents and;
- related with the *:enablesCategory* object property to only enabled categories and;
- related with the *:hasSubcategory* object property to only selected categories.

In practice, it implies that when the reasoner infers an instance of *:Category* to be an instance of *:SelectedCategory*, then all the instances of *:Document* this category categorises are inferred to be instances of *:CandidateDocument*. We apply the same process to other instances of *:Category* related to this selected category with the properties *:enablesCategory* and *:hasSubcategory*. The reasoner infers them as instances of *:EnabledCategory* and *:SelectedCategory*, respectively.

Notice the chain reaction that this reasoning process implies. When one category is inferred as an instance of *:SelectedCategory*, the reasoning process potentially infers other categories as instances of *:SelectedCategory*. The latter inferences let the reasoner infer new candidate documents and potentially other selected categories.

Notice also that the combination of the *:SelectedCategory* class definition and the *:hasSubcategory* object property transitivity entails that all the subcategories of a selected category are also selected.

We discuss instances of *:SelectedCategory* as being inferred. In practice, this is the case. The IR system, leveraging the IR ontology, inserts instances of the class *:Search* and asserts one of those instances as being an instance of the *:Search* subclass *:SearchContext*. The latter subclass is defined using the role-based range pattern:

```
Class: SearchContext SubclassOf(hasSearchCategory only SelectedCategory)
```

Such a definition lets us insert many searches in our domain graph while considering only one at a time, triggering cascading inferences. Let us now illustrate how an IR system could leverage the IR ontology.

5.3 Demonstration

The purpose of the following demonstration is twofold. First, it aims to illustrate how the IR ontology can be leveraged. We also illustrate the KG definition discussed in chapter 1 by designing our example KG, separately constructing the data, domain and mapping graph. That way, we propose a concrete implementation of the KG definition summarised by figure 1.7. The demonstration we detail below is available on GitHub[5] as a Python notebook along with the data files.

We implement our example KG in RDF. Hence, it is effectively a set of files corresponding to the Turtle serialisation of RDF triples. Our KG follows the DELG model. The data graph follows the latter graph data model, and the domain graph leverages the RDF, RDFS and OWL languages to model knowledge.

5.3.1 Setting the stage: a pizzeria use case

To demonstrate the potential of the IR ontology, we build an example around pizzas. This choice stems from the famous pizza ontology tutorial and its many variants[6].

For this demonstration, we consider a pizza and technology lover who owns a restaurant. Our protagonist has been making pizzas since childhood. The restaurant business is going well, and he has decided to wisely allocate time and money to leverage the knowledge he has acquired from long years of experience as best as possible. He recently heard about this fascinating technology that are KGs. Their representation of Semantics enables machines to look even brighter than before. So he decides to make a KG of his restaurant pizzas. In practice, he is creating the data graph in our KG definition. To make the example feel natural (and also because of our lack of imagination), we will use the pizzas proposed on the menu of the Bisou, a great pizzeria located in Rouen, France[7].

To illustrate the different RDF triple files representing the graphs we create for this demonstration, we will show some selected pieces of the raw Turtle files. To ease the visualisation, let us introduce here all the namespaces we will use (See 5.2.4 for an explanation of how to interpret such namespaces):

[5] https://github.com/msesboue/ir-ontology (Accessed on Thursday 3rd October, 2024)

[6] Our inspiration comes from the pizza tutorial of Michael DeBellis: https://www.michaeldebellis.com/post/new-protege-pizza-tutorial (Accessed on Thursday 3rd October, 2024)

[7] You can find the menu here: https://www.bisourouen.fr/#la-carte (Accessed on Thursday 3rd October, 2024)

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ir-onto: <http://www.msesboue.org/o/ir-ontology#> .
@prefix pizza: <http://www.msesboue.org/o/pizza-data-demo/bisou#> .
```

5.3.2 The data graph

Our example data graph is composed of two parts. The first part models the pizzas, and the second organises ingredients and pizza components in different taxonomies.

The pizza graph

To represent our pizzas, we use the following edges:

- *pizza:has_topping* edge links a pizza node to a topping ingredient one.
- *pizza:has_kind* edge links a pizza node to a kind of pizza, e.g., the vegetarian pizza kind node.
- *pizza:has_base* edge links a pizza node to its base node, e.g., the tomato or cream base nodes.
- *pizza:has_spiciness* edge links a pizza node to a spiciness level one, e.g., medium or mild.

In the above, we pay attention to using the terms *node* and *edge* since we are describing the data graph. However, once linked to the domain graph using the mapping, the nodes and edges will be referred to by their domain graph counterparts *entity* and *relation*. In this work, we focus on exploiting the domain graph. However, if we were to exploit the data graph, e.g., running some graph algorithm to extract new information, we would continue to talk about nodes and edges.

According to the menu[8], we have 12 pizzas in our data graph. Here is an excerpt of the triples representing the pizzas *God Save the King* and the *Hot Stuff 4.0*. Figure 5.4 depicts the corresponding graph representation.

```
pizza:_godSaveTheKing pizza:has_base pizza:_tomatoBase ;
pizza:has_kind pizza:_porkPizza ;
pizza:has_topping pizza:_carpaccioChampignonDeParis,
    pizza:_jambonDeParisWithHerbs,
    pizza:_mozzaFiorDiLatte,
    pizza:_olive .

pizza:_hotStuff40 pizza:has_base pizza:_tomatoSauceAlaNdujaBase ;
    pizza:has_kind pizza:_porkPizza ;
    pizza:has_spiciness pizza:_medium ;
    pizza:has_topping pizza:_blackPepper,
        pizza:_mozzaFiorDiLatte,
        pizza:_olive,
        pizza:_oliveOil,
        pizza:_spianataPiccante,
        pizza:_sweetPepperDrop,
        pizza:_tarragon .
```

[8] https://www.bisourouen.fr/#la-carte (Accessed on Thursday 3rd October, 2024)

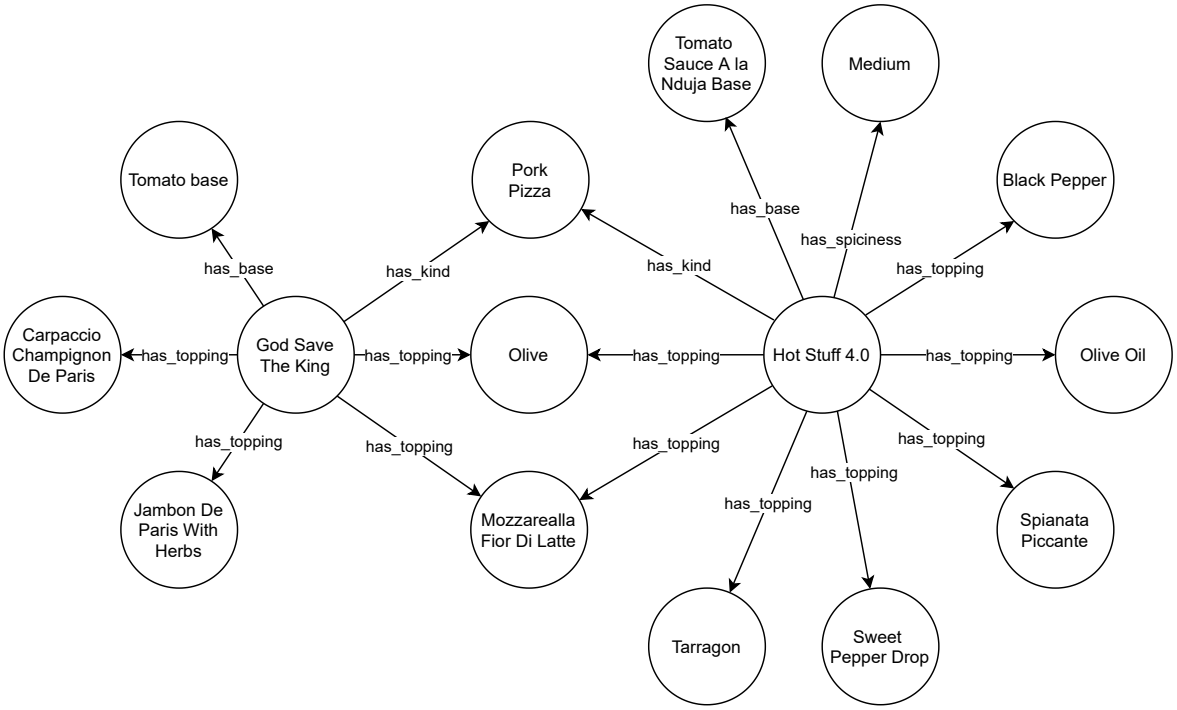


Figure 5.4: Graph representation of the *God Save the King* and the *Hot Stuff 4.0* pizzas.

The taxonomies graph

We extend our data graph with a taxonomies data graph. This graph represents 4 different taxonomies: a taxonomy of pizza bases, one of pizza kinds, one of pizza toppings and one for the spiciness levels. The category nodes in the taxonomies data graph are linked together by *pizza:has_subcategory* edges. This edge is distinct from the IR ontology *ir-onto:hasSubcategory* relation.

Here is an excerpt of the triples representing the pizza bases taxonomy, and the corresponding graph representation is depicted in figure 5.5.

```
pizza:_pizzaBase pizza:has_subcategory pizza:_creamBase,
    pizza:_tomatoBase .

pizza:_creamBase pizza:has_subcategory pizza:_blackTruffleCreamBase,
    pizza:_mustardCreamBase,
    pizza:_oignonCreamBase,
    pizza:_onionSquashCreamBase,
    pizza:_ricottaCreamBase .
```

In our demonstration, the taxonomies are separated. No transversal relations exist between the categories of one taxonomy and the ones of another. However, we could easily imagine other examples of use cases where it could be the case. In an industrial component classification context, we could imagine a taxonomy of functions and one of shapes. A specific function could imply a particular shape.

Our data graph now comprises the union of the pizza and taxonomies graphs. It contains 80 different nodes, among which 12 nodes denote pizzas and 68 denote categories, i.e., nodes of the taxonomy graph. The data graph in itself is meaningless. It does not self-contain any semantic definitions of its nodes and edges. We can only consider running some graph algorithm on top of it to extract topographical-based information that we could interpret based on implicit knowledge about this data graph. For instance, we could derive the most used pizza base. However, it would

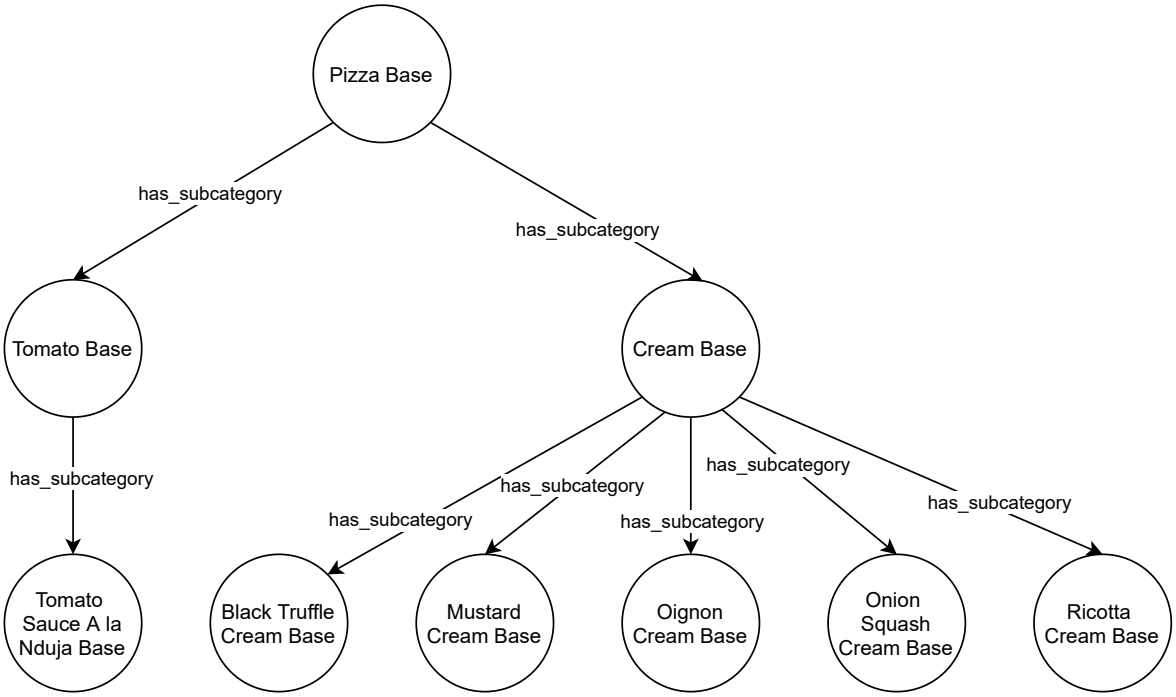


Figure 5.5: Graph visualisation of the pizza bases taxonomy.

imply some knowledge about the nodes, such as that some nodes denote pizzas and others their base. This knowledge is implicit at the moment. The goal of the domain graph is to make it explicit.

5.3.3 The domain graph

Our domain graph is also composed of two parts. We start from a core ontology, the IR ontology, that we extend with our domain knowledge about pizzas.

We already presented the IR ontology. We extend it with our pizza knowledge, which implies expressing such knowledge with OWL. We first define some OWL classes:

- The *pizza:Pizza* class denotes the concept of a pizza.
- The *pizza:PizzaTopping* class denotes the concept of a pizza toppings.
- The *pizza:Spiciness* class denotes the concept of a spiciness.
- The *pizza:PizzaKind* class denotes the concept of a pizza kinds.

We give some human-readable and understandable URIs to our pizza domain classes. Hence, using the mapping, one might envision all pizza nodes in our data graph as instances of the *pizza:Pizza* class. However, the domain graph will be used by a computer for which the human-readable URIs are plain old meaningless identifiers. The only understanding the computer can grasp is the logic defined by the OWL language derived from Descriptions Logics. Asserting the classes *pizza:Pizza* and the others as OWL classes gives the latter meaning. Which, in practice, comes down to giving an OWL interpreter, i.e., a computer program, a file containing the triples `pizza:Pizza rdf:type owl:Class` and the same for each class. The OWL interpreter should also know how to read the chosen serialisation syntax, here the Turtle syntax.

We only need to say this little bit about our pizza domain knowledge. However, at the moment, the IR ontology and the pizza domain knowledge are separate. None of them knows about the existence of the other. We need to align the pizza domain knowledge with the IR ontology. To do so, we add the following knowledge:

```
pizza:Pizza rdfs:subClassOf ir-onto:Document .

pizza:PizzaBase rdfs:subClassOf ir-onto:Category .
pizza:PizzaKind rdfs:subClassOf ir-onto:Category .
pizza:PizzaTopping rdfs:subClassOf ir-onto:Category .
pizza:Spiciness rdfs:subClassOf ir-onto:Category .
```

The above 5 triples define what we consider to be categories and documents in our pizza domain. Pizzas are the thing our IR system lets us search. They are categorised by their base, kinds, toppings and spiciness.

Our domain graph comprises the IR ontology, 34 axioms, among which 8 classes and 6 object properties extended with 5 the classes of our pizza domain knowledge. It forms a total of 44 axioms.

At the moment, the data and domain graphs are entirely separate. Hence, we can not interpret the data graph leveraging the Logic defined in the domain graph. We must construct the mapping between the data and domain graphs to complete our pizza KG.

5.3.4 The mapping graph

In our example, we define our mapping as a graph as well. We can break down the mapping into different parts. As our domain domain graph is expressed in OWL, we should first map the data graph into OWL. Then, we can focus on mapping the data graph specifically into our pizza and IR ontologies.

Mapping the data graph into OWL corresponds to asserting all nodes as a *owl:NamedIndividual* and all edges as an *owl:ObjectProperty*. Stating such a thing is already mapping our data into an interpretation framework. We can automatically perform this task by applying the following SPARQL CONSTRUCT query over the data graph:

```
CONSTRUCT {
    ?s   rdf:type    owl:NamedIndividual .
    ?p   rdf:type    owl:ObjectProperty .
    ?o   rdf:type    owl:NamedIndividual .
} WHERE {
    ?s  ?p  ?o .
}
```

To map the data graph specifically into our domain graph, we need to specify the type of each node, i.e., asserting each node as being of type either *pizza:Pizza*, *pizza:PizzaTopping*, *pizza:PizzaKind*, or *pizza:Spiciness*. Knowing the structure of our data graph, we can also perform this task with some SPARQL CONSTRUCT queries. We detail each query in the companion Python notebook. We give here as an example the one asserting pizza toppings of type *pizza:PizzaTopping*:

```
CONSTRUCT {
    ?topping   rdf:type    pizza:PizzaTopping .
} WHERE {
    pizza:_pizzaToppings  pizza:has_subcategory*  ?topping .
}
```

The latter SPARQL CONSTRUCT query uses a property path, `pizza:has_subcategory*`, to navigate all the pizza toppings taxonomy from the top category down to the leaf ones. We can use similar queries to assert the other category types. Another query asserts pizzas of type *pizza:Pizza*.

Finally, we must map the new object properties with the IR ontology. At this stage, we make critical decisions concerning the expected inferences. We need to specify the relations from the

data graph used to categorise our documents, i.e., the pizzas. We also link the relation *pizza:has_subcategory* coming from the data graph with one defined in the domain graph, specifically in the IR ontology. We add the following triples:

```
pizza:has_spiciness rdfs:subPropertyOf ir-onto:categorisedBy .
pizza:has_kind rdfs:subPropertyOf ir-onto:categorisedBy .
pizza:has_topping rdfs:subPropertyOf ir-onto:categorisedBy .
pizza:has_base rdfs:subPropertyOf ir-onto:categorisedBy .

pizza:has_subcategory owl:equivalentProperty ir-onto:hasDirectSubcategory .
```

The first four triples define the relations *pizza:has_spiciness*, *pizza:has_kind*, *pizza:has_topping* and *pizza:has_base*, as relations categorising the pizzas. The last triple defines the relation *pizza:has_subcategory* as equivalent to the IR ontology *ir-onto:hasDirectSubcategory* object property. We could have made the latter pizza data graph relation equivalent to the IR ontology one *ir-onto:hasSubcategory*. However, doing so would lose the information about direct subcategories as *pizza:has_subcategory* would have been inferred transitive. We would not have leveraged the part-whole reasoning pattern.

Our knowledge graph is the union of the data, mapping and domain graph. It is ready to process user searches.

5.3.5 Putting it all together: search examples

At this stage, we put ourselves in the shoes of an IR system. The latter system processes the user searches and transforms them into triples representing the search. It then inserts these search data graphs into the KG and maps them to the domain graph by asserting the search of type *:Search*.

The IR system can insert as many searches as needed. However, the reasoning process is ignited only when the system asserts a search as of type *:SearchContext*. Hence, it must add one more triple to leverage the domain graph logic. The IR system can query for the candidate documents and the enabled categories. Finally, it iterates over this process as the user interacts with the system. The process is illustrated in figure 5.6.

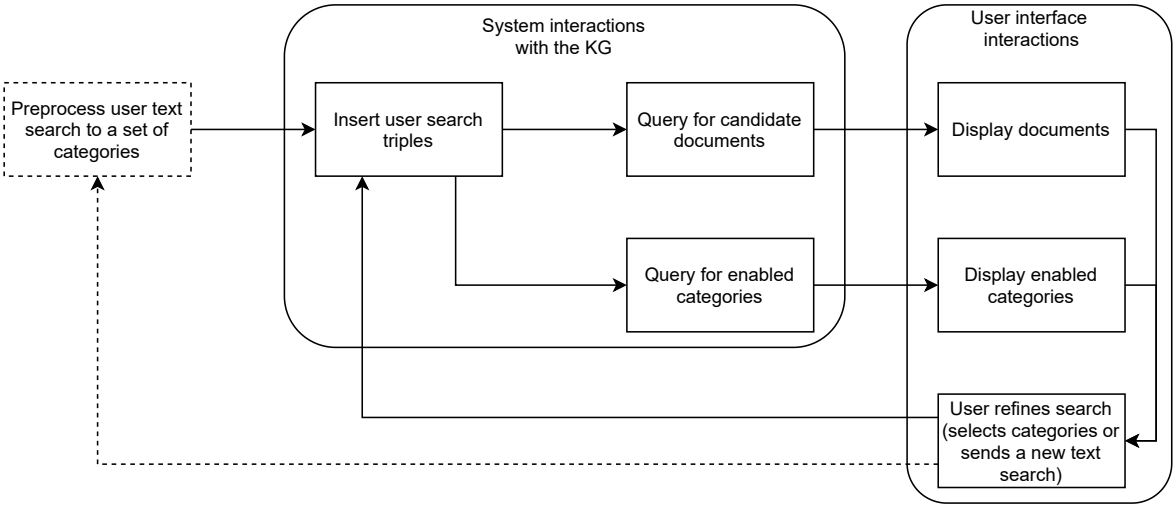


Figure 5.6: Example of the process an IR system based on the IR ontology can follow.

The solid arrows show a typical flow of the system stages. The dashed arrows indicate some optional paths. Square boxes are tasks. Round boxes group system tasks together.

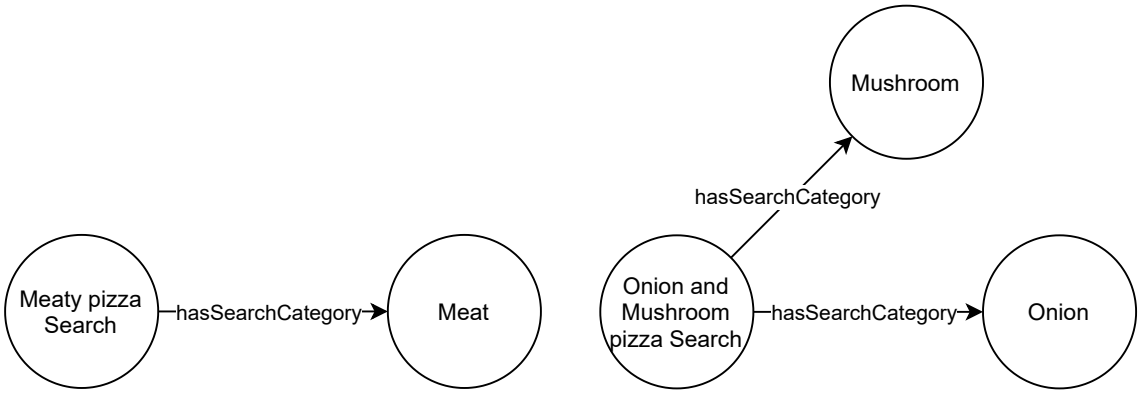
Let us illustrate the process with some examples of user searches. The following triples represent the search data graphs and their mapping into the domain graph. Figure 5.7 illustrates the corresponding data graphs.

The following search looks for pizzas with meat toppings and is illustrated in figure 5.7a. The triples contain both the data graph and its mapping into the domain graph.

```
pizza:_meatyToppingSearch rdf:type ir-onto:SearchContext .
pizza:_meatyToppingSearch ir-onto:hasSearchCategory pizza:_meat .
```

The following search looks for pizzas with some onions or mushrooms and is illustrated in figure 5.7b. The triples contain both the data graph and its mapping into the domain graph. As we will see when discussing the IR ontology limitations in 5.4.1, here we intentionally say pizzas with onions *or* mushrooms, and not *and*.

```
pizza:_onionMushroomToppingSearch rdf:type ir-onto:SearchContext .
pizza:_onionMushroomToppingSearch ir-onto:hasSearchCategory pizza:_onion .
pizza:_onionMushroomToppingSearch ir-onto:hasSearchCategory pizza:_mushroom .
```



(a) Data graph visualisation of the meaty pizza search.

(b) Data graph visualisation of the onions mushrooms pizzas search.

Figure 5.7: Data graph visualisation of the meaty and onions mushrooms searches.

The IR system only needs to implement 3 different SPARQL queries. One is to insert the search triples, and the two other ones are to fetch the candidate documents and enabled categories. The latter two queries correspond to the competency questions CQ2 and CQ3 we introduced in section 5.2.2. We present the 3 SPARQL queries below.

The following SPARQL INSERT query lets the IR system insert a user search in the KG, triggering the reasoning process. *search_uri* and *search_data_graph* are placeholders to replace by the search URI and the above-shown data graphs, respectively.

```
INSERT DATA {
    <search_uri> a ir-onto:SearchContext .
    search_data_graph .
}
```

The following two SPARQL queries let the IR system query for the candidate documents and enabled categories, respectively.

```
SELECT ?enabled_cate WHERE {
    ?enabled_cate rdf:type ir-onto:EnabledCategory .
}

SELECT ?candidate_doc WHERE {
    ?candidate_doc rdf:type ir-onto:CandidateDocument .
}
```

When we use the above 3 SPARQL queries with the search data graph for meaty toppings pizzas, the SPARQL query engine returns the following enabled categories:

```
pizza:_rostelloHamWithHerbs
pizza:_parmaHam
pizza:_jambonDeParisWithHerbs
pizza:_spianataPiccante
pizza:_speck
pizza:_bresaola
pizza:_jambonDeParisWithBlackTruffle
pizza:_ham
pizza:_mortadella
```

When used with the candidate documents query, the engine returns the following:

```
pizza:_bambino
pizza:_burraTadah
pizza:_godSaveTheKing
pizza:_hotStuff40
pizza:_laVieEnRose
pizza:_leonardo
pizza:_tartufo
pizza:_zucchero
pizza:_mortadella
```

Looking at the menu, the Bambino, Burra'Tadah!, God Save The King, Hot Stuff 4.0, La Vie en Rose, Leonardo, Tartufo, Zucchero and Mortadella pizzas all contain some meat. If we had defined the IR ontology subcategory relation as non-transitive, no pizza would have been found since no pizza is categorised by the category *meat*. Since our taxonomies graph does not assert any enabling transversal relations between categories, the enabled categories are all the subcategories of *meat*.

We discuss the onions and mushrooms pizzas search results in the next section.

In this demonstration, we illustrated how to use the IR ontology in an IR system. We constructed the domain graph by extending the IR ontology, the data graph and the mapping. Each corresponds to one or more RDF triple files. Constructing the graphs separately illustrates an implementation of the KG definition components. Figure 5.8 illustrates the pizza IR KG we use in this demonstration following the KG definition schema in figure 1.7. In the next section, we discuss the advantages and limitations of the IR ontology.

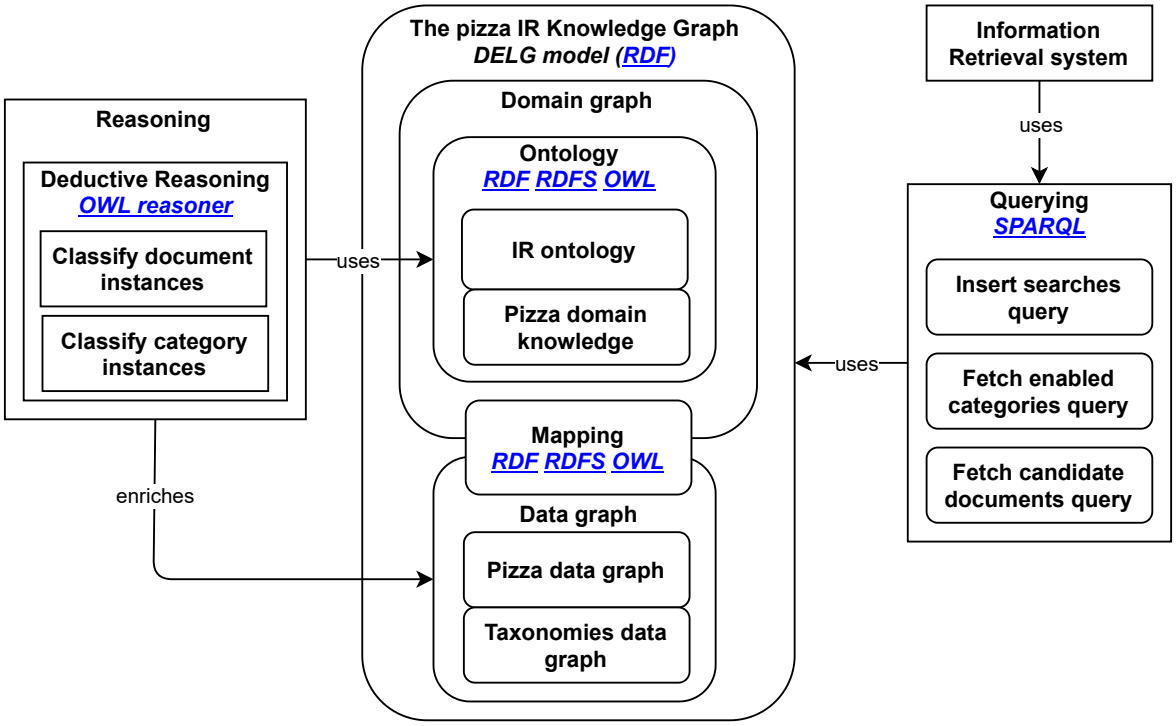


Figure 5.8: The pizza Information Retrieval Knowledge Graph illustrated following the knowledge graph definition given in 1.7.

Round boxes denote RDF files, while squared ones denote processes. Bold names in round boxes are the components' labels. The boxes' imbrications denote composition relations. The round box labelled *Mapping* straddles the *Domain graph* and *Data graph* boxes denoting its mapping role. Finally, the SW technologies to implement each component are mentioned in blue bold italic and underlined letters.

5.4 Conclusion and future works

This chapter presents the IR ontology, a core ontology designed to power an IR system. With an example based on the famous pizza OWL tutorial, we demonstrate how to use the IR ontology in practice. The latter demonstration also showcases an implementation of the KG definition summarised in figure 1.7. To conclude, we first discuss our IR ontology advantages and limitations with some potential tracks to explore. We then propose an extension to our ontology.

5.4.1 Information Retrieval ontology advantages and limitations

One common goal of knowledge modelling projects is to make explicit, implicit knowledge hidden in code bases, database schemas, and designers' heads. One approach to knowledge modelling is implementing the W3C Semantic Web standards, such as the modelling languages RDF, RDFS and OWL. One motivation for choosing such an option is not only to take knowledge out of the data silos but also to move it closer to the data itself. With KGs implemented using RDF, this data and knowledge gathering is achieved by representing both using the same format, i.e., using RDF triples in our case.

We achieve this data and knowledge gathering when building a KG extending the IR ontology, expressed in OWL. However, like any choice, modelling knowledge in OWL also comes with some theoretical and practical limitations.

The demonstration relies on a corpus of documents categorised by categories to leverage the IR ontology. The categories can be organised in different classification systems linked together by transversal relations. From a graph topology perspective, we expect a graph of interlinked tree structures. Having data structured in such a manner might be a limiting factor, but those requirements are fairly easy to meet in practice. Indeed, companies almost always organise their content

using some tags or categories. Moreover, different people in the same company often organise the same documents differently, i.e., using different classification systems. Meeting such data format requirements only involves identifying the classification systems and turning them into an RDF data graph. The IR ontology then enables the integration of any point of view, i.e., classification systems. It aligns with the World Wide Web *AAA* slogan, "Anyone can say Anything about Any topic" [AHG20].

In practice, reasoning over an OWL ontology is often performed offline. A system might leverage newly inferred facts online. However, in production systems, real-time reasoning is rarely required. Indeed, inferring new facts and validating the semantic coherence of a KG at runtime, at scale and with today's KG size is resource-intensive and, therefore, challenging. With our approach, it would be even more challenging since we require intensive writing, reading and reasoning operations at runtime. A production system leveraging the IR ontology to power a faceted search would need extreme parallelism to cope with multiple users simultaneously. It would require one ontology instance for each user search. Indeed, to function correctly, the reasoning can consider only one search at a time. Otherwise, the selected categories, the candidate documents and the enabled categories of one search would pollute the others. We could address this issue by considering named graphs, e.g., having one named graph for each search. We would get inferences in the search context by querying in the scope of one named graph. However, more is needed to solve the computational resource requirements.

We can use the IR ontology to power a faceted or classification browsing search. However, another limit is that we add new categories to the search as the user refines it without removing the broader ones. Hence, the candidate documents set will only be refined if we remove the parent categories when adding child ones to the search. We designed the ontology so that if a category is selected, all its subcategories are also selected. Hence, if we do not remove the parent categories from the search when adding the refined ones, the candidate documents inferred because of the parent category will still be inferred as candidate documents.

OWL reasoning only creates new triples while ensuring the inferred triples stay consistent, i.e., the inferred triples do not create logical inconsistencies. OWL reasoning does not update any ground asserted triples from which it draws the inferences. One way of solving the issue of updating the candidate document sets as the user refines the search would be by updating the search asserted triples. That is to say, when asserting the triples corresponding to the user-refined search, if a selected category is a subcategory of a previously selected one, we remove the latter parent category for the refined search asserted triples. In other words, we programmatically ensure the OWL ontology always contains only the most restrictive search. In the exploration process, the user could also back up to a parent class and re-broaden the search. Then, there is no need to remove the subcategory asserted triples.

The limit just described, though programmatically circumventable, shows OWL reasoning limitations forcing part of the logic corresponding to our IR domain knowledge to stay in the system code base. We must leverage rules to push this logic closer to the database, e.g., using SPARQL UPDATE queries.

So far, we have discussed rather practical application limits. However, one principal limit in our use cases comes from OWL foundations. Since OWL reasoning is based on the OWA, we can not easily use a conjunction of categories to refine a search. The reasoning process will always infer as candidate documents any pizzas containing one of the selected categories since each candidate document inference is independent of another.

Coming back to the onions and mushrooms pizza search. The results of the SPARQL query fetching candidate documents are the following:

```
pizza:_godSaveTheKing
pizza:_laVieEnRose
pizza:_tartufo
```

The candidate pizzas are the *God Save the King*, *La Vie en Rose* and *Tartufo*. The Bisou pizzeria menu states that the *God Save the King* and *Tartufo* pizzas only have some mushrooms among

their toppings. Moreover, the pizza *La Vie en Rose* only has some onion among its ingredients. None of the pizzas have both onions and mushrooms.

Since the inference is made by logical deduction, the reasoners often let us inspect the process and explain each inference. One workaround to the category conjunction issue could be to count and inspect the number of possible explanations. We could then keep only the candidate documents having at least as many explanations as the search categories. We would also need to study the explanations and determine the selected categories responsible for the candidate document inference.

Restricting ourselves to OWL, we can infer some candidate documents only if a conjunction of categories is selected. It is, however, different from inferring as candidate documents only the documents categorised by the conjunction of all selected categories. The approach requires counting the selected categories on top of identifying them. Due to OWL being based on the OWA and the NUNA, counting things in OWL requires workarounds to close the world partially. In particular, we must explicitly state that two things are different to conclude that there are at least two things. The NUNA would otherwise leave the possibility for two instances to be the same and, therefore, counted as one. The particulars of counting things in OWL are out of the scope of our work. We encourage the interested reader to study [AHG20] thoroughly.

Many faceted search systems derive the enabled categories based on existing documents. Such systems are operating in a CWA. However, we could apply the OWA approach in the IR context. It might be interesting to distinguish between searches with candidate documents, the ones that do not have any known candidate documents, and those that can not have any candidate documents. The latter searches would be semantically inconsistent with our view of the world. That is the idea we explore in the next section as an extension of the IR ontology and conclusion.

5.4.2 Extending the Information Retrieval ontology

Another use case for the IR ontology could be identifying logically inconsistent searches. It would correspond to the competency question:

CQ4 What are the semantically incoherent searches?

In this use case, we consider the world of IR searches composed of 3 kinds. The most apparent searches are the ones having some known candidate documents. Our knowledge lets us find some documents fitting the search intent. The second obvious kind of searches is the one considering searches for which we do not have candidate documents, i.e., our knowledge does not let us infer any document fitting the search intent. However, following the OWA, we can consider two cases among the latter searches. Either our knowledge is too limited. Hence, some documents might exist, but we are unaware of them. Alternatively, the search does not make sense, i.e., the search is logically inconsistent. We could infer that no document will ever fit its intent in this case.

In practice, OWL reasoners are solvers. They do not detect logical inconsistencies per se. Instead, they represent unsatisfiable classes by inferring them as equivalent to the class *owl:Nothing*. The OWL standard defines the class *owl:Nothing* and *owl:Thing*. From a set perspective, they correspond to the empty set and the set of all individuals, respectively. Hence, asserting something equivalent to the class *owl:Nothing*, though often interpreted as a logical inconsistency, does not make an ontology inconsistent. It simply defines an unsatisfiable class, i.e., a class that can not have any instances. However, if an instance of such an unsatisfiable class is asserted, then there is a true inconsistency. In such a case, the reasoner stops and can not provide any insights as the ontology is logically false.Similarly, we can emphasise tautologies when the reasoner infers classes equivalent to *owl:Thing*.

To address the semantically incoherent searches use case, we add a new subclass of the class *ir-onto:Search*, *ir-onto:IncoherentSearch*. In contrast with many logical consistency validation use cases, we do not intend to infer anything equivalent to *owl:Nothing*. Instead, we apply a similar approach restricted to searches. We aim at asserting incoherent searches as instances of our class *ir-*

onto:IncoherentSearch. To achieve this reasoning objective, we define the *ir-onto:IncoherentSearch* equivalent to a search that has search categories the incompatible combination of categories.

Extending our pizza example, we could define a search for a vegetarian pizza with ham. This search is incoherent. Hence, we define the class *ir-onto:IncoherentSearch* as equivalent to an instance having search categories both *pizza:_vegetarianPizza* and *pizza:_ham* categories. It corresponds to the following Manchester syntax definition:

```
Class: IncoherentSearch EquivalentTo(
        (hasSearchCategory value pizza:_vegetarianPizza)
        and
        (hasSearchCategory value pizza:_ham)
    )
```

We could envision the same approach for semantically incoherent documents. However, the IR ontology objective is not to semantically validate a data graph. Validating the semantic coherence of the documents should be done with a particular domain graph. We could apply the pizza ontology to our pizza graph. It corresponds to applying a different domain graph, i.e., a different point of view on the same data. That is possible with semantic Web technologies since they are built on top of the Web foundations, which follow the *AAA* slogan. Modelling incoherent searches this way also has severe limitations. It requires explicitly expressing all the combinations of categories that can not exist.

Chapter 6

Industrial experiments

Contents

This manuscript describes works done in the context of an industrial PhD (CIFRE[1] in France) in partnership with the company TraceParts. From the TraceParts's perspective, this PhD work aims at enhancing their Computer-Aided Design (CAD) content platform[2] search engine. The practical aim is to enhance the user experience on www.traceparts.com, specifically by reducing the number of iterations a user goes through before downloading a CAD model. This user action is a key metric for TraceParts' business success.

The PhD research focused on optimising the TraceParts search engine, employing a formal knowledge modelling approach. The preceding chapters introduced approaches we explored to implement this knowledge-based system. However, we have yet to delve into the practical applications and solutions we studied in the industrial context of TraceParts. This chapter focuses on the knowledge consumption part of our KGBS architecture (chapter 3) as depicted in figure 6.1.

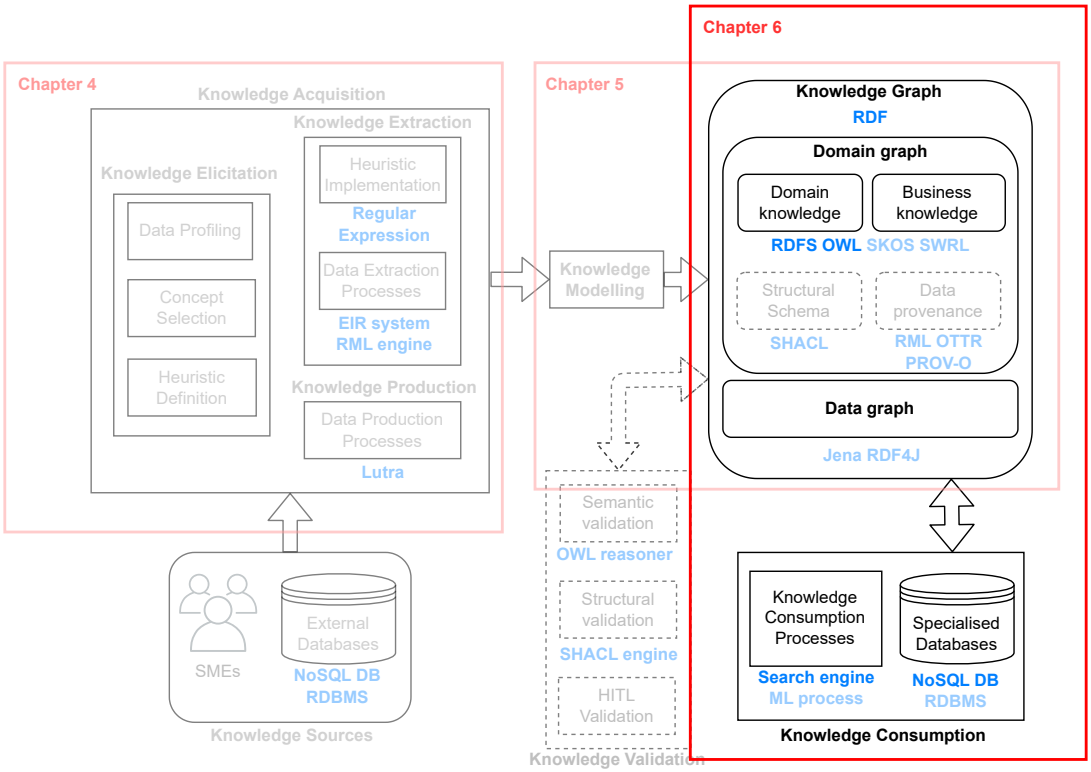


Figure 6.1: Knowledge Graph-Based System architecture. Focus on chapter 6.

Square boxes refer to activities. Round boxes depict containers. The boxes imbrications denote sub-activities and sub-containers, respectively. Arrows illustrate data flows. Large red boxes denote the components each chapter addresses. Some candidate technologies for implementing each component are mentioned in bold blue letters. The architecture parts turned into dashed and light color are the parts left out in this chapter. (SMEs: Subject Matter Experts; HITL: Human In The Loop)

This chapter introduces and discusses our experiments iteratively implementing knowledge-based approaches to TraceParts's Information Retrieval (IR) use case. We first define the application context, presenting the company TraceParts and then focusing on their current search engine. We then dive into the details of the IR use case, describing our corpus, the user queries and the challenges we face. This second section provides the necessary context for the experiments we then describe. Before presenting and discussing the experiment results, we detail the experiment setting. We conclude this chapter by discussing the experiments, their advantages, limitations and potential for future work.

[1] https://www.anrt.asso.fr/fr/le-dispositif-cifre-7844
[2] www.traceparts.com

6.1 Industrial context

This section presents TraceParts and its CAD-content platform. The latter platform's search engine is this chapter's main subject. We introduce the platform, and then focus on the current search engine.

6.1.1 TraceParts CAD-content platform

The first version of the TraceParts' CAD-content platform was launched in July 2001. It is a unique web platform offering free technical resources for designers and engineers in 25 languages. Among these data are over 120 million 3D files from more than 1,800 catalogues, enabling CAD software users to save precious time by not having to redraw "off-the-shelf" industrial components for their machine, tool or assembly design work.

The service for downloading 3D CAD files from www.traceparts.com is financed by the component manufacturers and distributors who list their products and by advertising. Today, the portal has over 5.3 million subscribers and generates over 7 million CAD files monthly, making it one of the world's leading 3D content websites for the industry.

Catalogue data is structured around open, future-proof formats. Finally, centralised data indexing enables robust, fast, high-performance catalogue search and navigation. The type of data processed through the platform is highly heterogeneous. Many parts are custom-configurable, with different information structures and value types, resulting in a large and constantly changing volume of data.

6.1.2 Search on TraceParts CAD-content platform

The CAD-content platform uses Elacticsearch[3] as its search engine. Every item extracted from the supplier catalogues is stored in Elacticsearch. The end users can search for manufacturers' parts in all the catalogues on the website. A part is identified by its manufacturer Part Number and may be contained in several catalogues. In order to group and classify manufacturer parts efficiently, TraceParts has created its own classification in the form of an additional catalogue.

End users search the platform content via a full-text search field, TraceParts' classification or an alphabetically organised list of catalogues. The results returned are then displayed on the main results page according to the classification proposed by default by Elacticsearch.

Users can use the "search" page to refine the results list using a faceted search. When a search does not identify the desired catalogue, facets are based on TraceParts categories. The category tree displayed is calculated in real time by compiling the search results and the category hierarchies stored separately.

The main results list contains a separate entry for each item, even if they correspond to variants of the same item group. We call the latter item group a Part Family, which gathers all the possible configurations for the same item identified by separate Part Numbers. Finally, the web application allows multi-language searches. Initially, all languages were available. However, for performance reasons, the current implementation uses the language chosen by the user and the default language, English.

6.2 The Information Retrieval use case

We have introduced TraceParts and its CAD-content platform, which shapes the industrial context surrounding our IR use case. Let us now dive into the details of this use case. We first introduce the corpus and then the user searches. We conclude this section with the challenges induced by such documents and searches.

[3] https://www.elastic.co/elasticsearch

6.2.1 Corpora and Documents

As already mentioned, the corpus is composed of CAD content. This content encompasses both the 3D CAD models and their descriptive data, i.e., their metadata. The latter metadata comes in various forms. Some examples are the CAD model datasheets, 2D drawings, and textual descriptions of various lengths.

Many works addressing the CAD model retrieval challenges focus on the CAD models with particular retrieval use cases, such as similar or equivalent design retrieval [QGY+16], CAD assembly retrieval or compatibility and function-oriented retrieval [LPMG19]. While some CAD model retrieval works focus on the CAD models, others consider the associated metadata, particularly textual data [LRR08]. This work focuses on the latter case.

In the following experiments, we will consider two levels of documents, i.e., two different corpora. The first corpus considers a Part Number as one document. Each possible CAD model configuration is a document. It is the current setting on www.traceparts.com. As Part Numbers are constructed by the combination of component parameters, they are theoretically infinite. In practice, though considering one document per Part Number implies a large corpus, the number of documents is still limited. Indeed, it is impossible to compute an infinity of component configurations. Moreover, in practice, the components' parameters are constrained by each other, the physics law, and the catalogue provider, which only sells a limited number of configurations.

The second corpus considers one document per Part Family, i.e., one document per component comprising all its possible configurations. This naturally implies a significantly smaller corpus. In early 2024, the Part Family corpus contained 1,110,738 documents, and the Part Number corpus was more than a hundred times more extensive, with 127,802,485 documents.

Searching on the Part Number or Part Family corpus has some practical implications, particularly on the user experience. Though each Part Number has distinct characteristics, their 3D CAD model preview looks very similar at first sight. Hence, from a user experience point of view, searching on the Part Family corpus enables better visual diversity in the search results page. Indeed, some commonly searched Part Families have millions of Part Numbers, leading to a search results page full of similar-looking results. Moreover, when the users click on a search result, they are redirected to a product page with a table containing all the enabled Part Numbers of the corresponding Part Family. This latter product page can be seen as a user search refinement.

We consider the same text fields regardless of the corpus choice. Some are short and can be seen as a label, e.g., a standard such as "*DIN 912*". Others are longer like descriptions, e.g., "*The P01 to P08 pumps are designed to pump lubricating fluids (oil, diesel oil, etc.). Their flow rate is from 1 to 24 L / min; maximum working pressure 10 bar.*"[4]. However, compared to a blog post or a tweet, for instance, most text fields are concise. There are at most 2 phrases. The text is highly technical. It contains many metrics, domain-specific notations, and abbreviations.

Part Family and Part Number documents contain sections in their metadata referring to particular objects. Each document is categorised into at least two categories, one from the TraceParts classification and one from the data provider classification. The latter data provider is the company providing the CAD content. They are suppliers and manufacturers for which TraceParts make available their catalogue content on www.traceparts.com. Each data provider can provide one or more catalogues with a classification. Data providers, catalogues, and categories have textual metadata, which is part of the metadata of the Part Number and Part Family documents.

The texts are also available in 25 languages. It means that, though not all documents have text in all 25 languages, each language can be found across the entire corpus. To those 25 languages, we can add all the words that are language-independent due to the text's technical nature. These latter words could be considered as part of a 26th language. Finally, though the content is available in many languages, most of it does not constitute grammatically correct sentences as they are technical titles. An example of such content is the Part Number title "*P01-P02-P04-P06-P08*".

[4] https://www.traceparts.com/en/product/pollard-pumps-pump-p01?Product=90-06052020-036848&PartNumber=P01 (Accessed on Thursday 3rd October, 2024)

Whether considering the Part Number or the Part Family corpus, all the documents' content is accessible on www.traceparts.com. Here is a numerical summary of the corpora:

- 1,110,738 Part Families
- 127,802,485 Part Numbers
- 25 languages
- Texts are on average 50 characters
- Texts are on average 7 words
- 2,556 data providers
- 1,913 catalogues
- 208,466 categories

We directly extract the Part Family, Part Number, language, data provider, catalogue, and category counts from our source database. Regarding the average text length values, we extract each Part Family's main default title field, which is also used as a short description, and do the mean over those texts.

6.2.2 User searches

TraceParts CAD-content platform users are searching from all over the world. Their search texts are smaller than the documents' text fields and can come in any language. On average, a user search text contains 13 characters separated into 2 words. There is no grammatically correct text among the user searches. They are all domain-specific keywords, notations, identifiers, and acronyms. However, those domain-specific tokens are aligned with the documents' content. Though the user search diversity is naturally huge, some searches are common. Those searches are even more common if we consider some spelling variations as the same search. Table 6.1 presents the top 20 searches leading to a CAD content download.

6.2.3 Search engine challenges

We decided to focus on textual CAD model metadata in this work. Here, we consider textual metadata, both descriptive texts and shorter texts, that can also be considered tags, e.g., a reference to a standard such as ISO or DIN. Aside from the corpus size and the financial investment available, which constrain possible approaches, we can identify three main challenges for the TraceParts CAD-content platform search engine. The latter challenges directly originate from the sort of text to process.

The most challenging characteristic is the 25 languages available on the platform, which must be indexed for search. This implies a much bigger vocabulary to align with the user's search and much more variety in the possible texts that reference the same document or concept. One example issue is that a user searching in Chinese for a document will never be suggested relevant documents lacking a Chinese description.

A typical approach to address the multi-language issue is to break the corpus into one corpus per language. We can then redirect the user search to the corresponding corpus. This approach partially corresponds to the current search engine implementation as the text fields are separated by language, and the search is directed to the default English language and the user search language fields. However, this approach does not alleviate the lack of language-specific texts for some documents, which implies that the language-specific corpora will have different documents. Second, this approach heavily relies on correctly detecting the user search language.

Query text	Nb search
motor	540
din 912	445
ball valve	443
valve	374
din 933	331
din 125	326
vis	323
skf	301
bearing	284
din 471	284
wheel	283
gear	270
screw	267
bolt	262
din 934	245
din 931	237
nut	237
din912	235
item	226
coupling	224

Table 6.1: The top 20 user searches leading to a CAD content download extracted from a dataset of 190,080 searches.

The challenge of user search language detection is a persistent one. User searches are typically short, domain-specific, and rarely form a grammatically correct sentence. This lack of contextual information makes it difficult to reliably infer a search language based solely on the search text. The current implementation relies on the user's selected language, which is manually set or inferred from the browser setting. However, this approach has limitations, given the common occurrence of users searching in different languages. When a search in a particular language fails, it is common for users to try the same search in English, for instance.

Though most texts are grammatically unstructured in the sense of natural language, they might be structured from a domain-specific point of view. Hence, we could tackle the language diversity challenge with a purely linguistic approach. However, the third challenge is that the CAD content covers many domains with too many vocabulary and language structures to handle them manually.

6.3 Our approach

In the previous sections, we have introduced the industrial context and the information retrieval use case, and described in depth the corpus and user searches we are working with. This section introduces our approach to improving the existing information retrieval system. We first present our objective and then our approach to reach it.

6.3.1 From text-based to concept-based search

Our objective is to improve the TraceParts CAD-content platform search engine. To do so, we leverage formal document representations and focus on textual content. In previous chapters, we presented some theoretical systems we could implement based on formal document representations. However, in practice, we need to lower our formal document representation expectations.

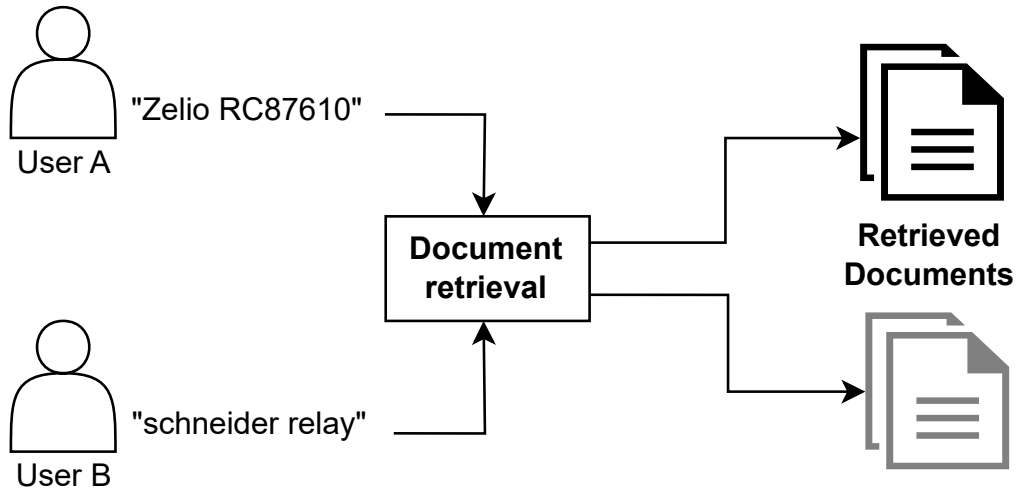


Figure 6.2: Example of a text-based document retrieval.

Square boxes refer to activities.

We can summarise our practical objective as moving from a text-based search to a concept-based one. In the scientific literature, such search paradigm shift is called semantic search or semantic indexing. For our implementations, it means moving from search results documents explicitly containing the query text to search results documents containing the query concepts or related ones.

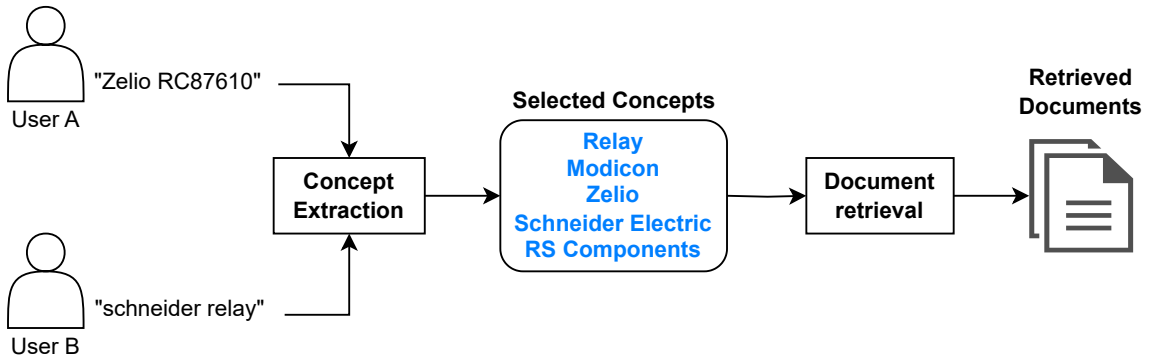


Figure 6.3: Example of a concept-based document retrieval.

Square boxes refer to activities. Round boxes depict containers.

Figures 6.2 and 6.3 depict from an abstract point of view a text-based document retrieval system vs a concept-based, respectively. Compared to the text-based approach, the concept-based one adds a step in the document retrieval process. The user query text is first used to extract concepts that are optionally enriched before retrieving documents. As depicted in the figures, whereas in a text-based system (figure 6.2), the retrieved documents most certainly will be different for two different text queries, in the concept-based system (figure 6.3), two different queries might lead to retrieving the same set of documents. Indeed, different texts might refer to the same concepts. A straightforward example of two such different text searches that should lead to the same retrieved documents are two same searches expressed in two languages, e.g., “vis DIN 912” in French and “DIN 912 screw” in English.

6.3.2 An iterative process

We build our experiments iteratively. First, we implement our baseline, a replica of the current CAD-content platform search engine. We then define some concepts and implement a concept-based document retrieval system. In the latter system, we use the concepts as a controlled vocabulary and do not yet leverage the relations between concepts. Hence, we use the concepts' relations in a third system to enrich the concept-based query. We call the latter system the KG-based document retrieval system. Finally, we experiment with including the platform user search history as a separate ranking step to influence the different system results lists.

6.4 Experimental protocol

We have introduced our approach to enhance the TraceParts CAD-content platform search engine with KG-based methods. Let us now introduce our experimental setting. Figure 6.4 gives an overview of the technical framework we implement to evaluate each document retrieval system prototype and compare their performance. We first present how we construct our dataset of ground truth examples and our corpora before discussing the metrics we select to compare our different approaches.

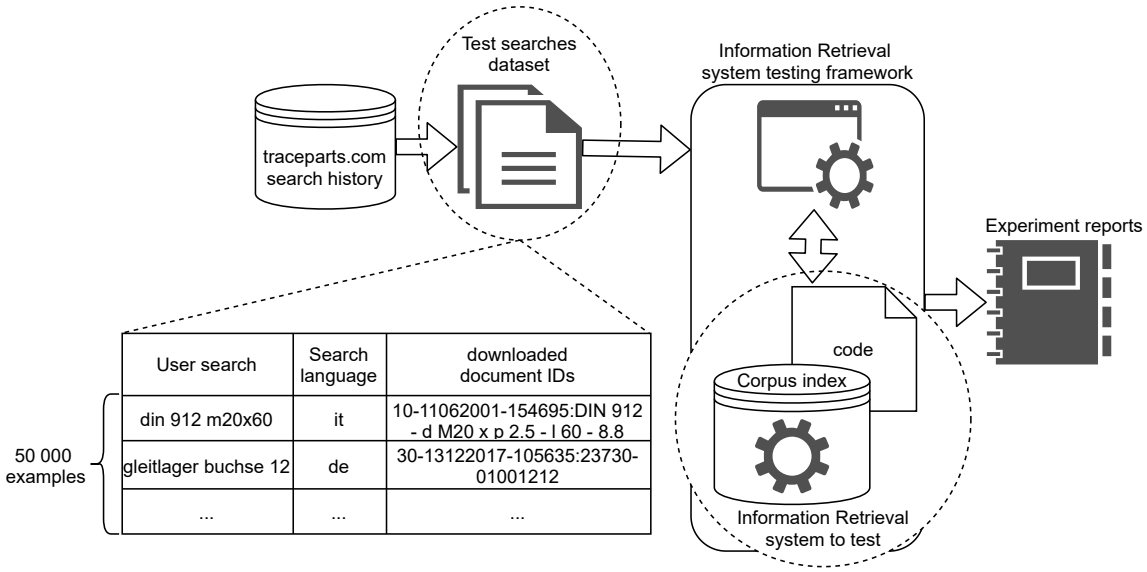


Figure 6.4: System to test the search systems.

Arrows denote data flows.

6.4.1 Evaluation dataset and corpora

Evaluating and comparing our experiments requires a set of example user searches with expected results. To define such expected search results, we must first define what makes a search successful.

From TraceParts' point of view, this PhD project aims to reduce the average number of iterations a user goes through before downloading a CAD model. Hence, to evaluate our search approaches, we consider the CAD models downloaded after a textual search to be an expected search result. We isolate in TraceParts' web application logs user sessions containing a text search leading to a download and associate the search and the downloaded document ID, i.e., the CAD model. We also retrieve the search languages necessary to reproduce the current search query.

The experiment results in the following sections are based on an evaluation dataset of example searches extracted from logs from October to December 2023. Among the about 160,000 examples, we selected a random sample of 50,000 for our tests. We ran some experiments on the full

dataset to verify that they do not show significant performance differences with the 50,000 sampled examples. Table 6.2 presents the Binary Mean at k (BM@k) values for both corpora and with the full dataset of example searches and the sample one. We can see the negligible differences between the entire dataset and the sample one. To run the multiple experiments in a manageable time, we continue with the sample of 50,000 example searches.

	Text-based system (baseline) BM@k			
Corpus → **@k ↓**	**PN corpus all examples**	**PN corpus 50K sample**	**PF corpus all examples**	**PF corpus 50K sample**
@5	**0.084**	**0.082**	0.115	0.114
@25	**0.129**	**0.129**	0.149	0.148
@50	**0.142**	**0.142**	0.157	0.157
@100	**0.152**	**0.151**	0.161	0.161
@350	**0.162**	**0.161**	0.164	0.164

Table 6.2: Text-based search results comparison for the full dataset of example search vs the 50,000 sample, for Part Family (PN) and Part Family (PF) corpora.

As mentioned in 6.2.3, the Part Family and Part Number corpora contain 1,110,738 and 127,802,485 documents, respectively. While we work with the original Part Family corpus for our experiments, we must sample the Part Number corpus. We extract a random subset of 1,674,603 Part Number documents. Likewise, for the concept-based approaches, we use the whole data providers (2,556), catalogues (1,913) and categories (208,466) datasets.

6.4.2 Evaluation metrics

We evaluate our experiments with 3 search results quality metrics: the Mean Average Precision at k (MAP@k), the Mean Reciprocal Rank at k (MMR@k), and the Binary Mean at K (BM@k). The first and second metrics take into account the search results positions, while the BM@k does not. For each metric, the closer the value is to one, the better the document retrieval system.

The Precision at k (P@k) is a measure taking into account the amount of relevant documents in the k first results:

$$\mathrm{P@}k = \frac{|\mathrm{R}es[1..k] \cap \mathrm{R}el|}{k}$$

where $|\mathrm{R}es[1..k] \cap \mathrm{R}el|$ is the number of relevant documents amongst the k first results. The Average Precision at k (AP@k) is the average over P@k for $k \in [1..k]$:

$$\mathrm{AP@}k = \frac{1}{|\mathrm{R}el|} \cdot \sum_{i=1}^{k} relevant(i) \cdot \mathrm{P@}i$$

where $|\mathrm{R}el|$ is the number of relevant documents in the corpus and $relevant(i)$ is the function returning 1 if the i^{th} document in the results is relevant, else 0. The MAP@k is then the AP@k mean over the set of example queries Q:

$$\mathrm{MAP@}k = \frac{1}{\mathrm{Q}} \cdot \sum_{n=1}^{\mathrm{Q}} \mathrm{AP@}k_n$$

The Reciprocal Rank at k (RR@k) monitors the position of the first relevant document in the k first results:

$$\mathrm{RR@}k = \frac{1}{min\{k|\mathrm{R}es[k] \in \mathrm{R}el\}}$$

where $min\{k|\text{Res}[k] \in \text{Rel}\}$ is the minimum position k amongst the results positive documents. The MMR@k is then the RR@k mean over the set of example queries Q:

$$\text{MRR@}k = \frac{1}{\text{Q}} \cdot \sum_{n=1}^{\text{Q}} \text{RR@}k_n$$

Finally, the BM@k considers whether there exists a positive document in the k first results and performs the mean over the set of example queries Q:

$$\text{BM@}k = \frac{1}{\text{Q}} \cdot \sum_{n=1}^{\text{Q}} relevant \in \text{Res}$$

We also monitor the number of searches leading to 0 and less than 400 selected documents. The latter values give us a better intuition for the precision of the potential overall search results.

The search results quality metrics are all precision-focused. Indeed, we could not compute any recall-based indicators as it would require exhaustively knowing the search examples' negative results. While defining a download as a positive document for a user search is straightforward, finding negative documents is harder and would require some manual annotations by domain experts. Even if we had access to such domain experts time, it would still be hard, if possible, to define an exhaustive set of negative documents. The notion of a negative document for a search is subjective in our context. Moreover, in the particular case of search engines, the recall value has become less indicative. Indeed, as corpora grew in size, it became less insightful to monitor whether or not all the relevant documents were in the search results. The recall value is even less informative when looking at only the k first search results, as the users often look for only one relevant result. Hence, precision is more indicative [BCC16].

6.4.3 Search engines

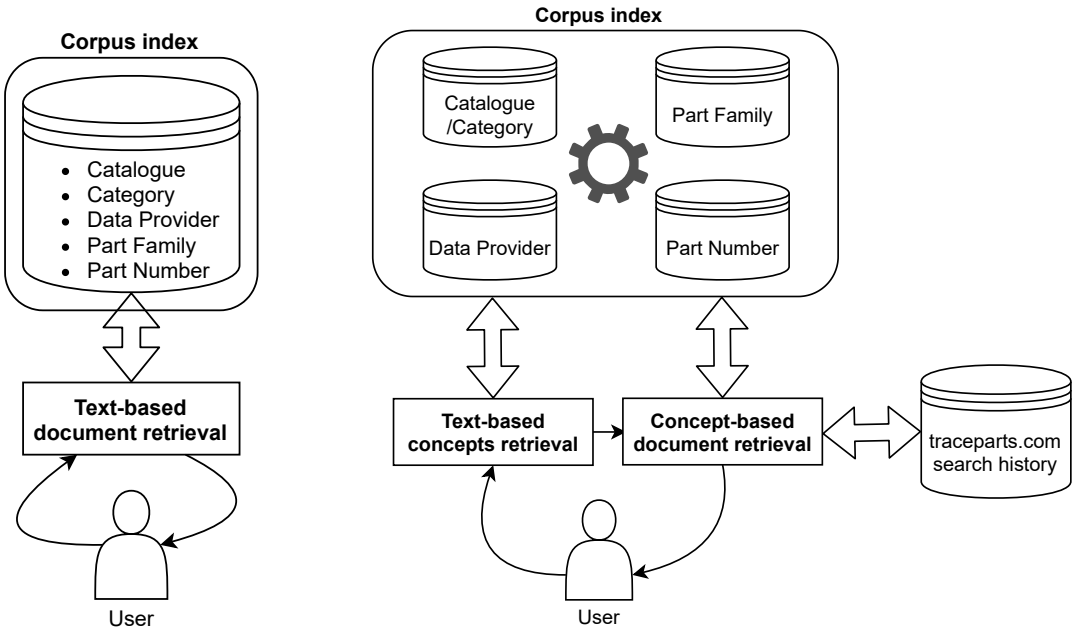
We built and tested 6 document retrieval systems following our iterative approach. The first is our baseline and reproduces the current text-based system. The second move from text-based to concept-based search but does not yet leverage the relations between concepts. Hence, a third KG-based system leverages concept relations. The fourth, fifth and sixth systems experiment integrating some implicit knowledge to the text, concept and KG-based systems. In our experiments, the implicit knowledge comes from the search history. This section provides in depth details on the search engines implementations before presenting the experiment results in the next section.

Each document retrieval system is implemented with the Elasticsearch system's implementation of the BM25 retrieval score. Elasticsearch is the technology in place at TraceParts, and we thought to exploit it as much as possible. This choice makes system implementation easier, which lets us focus on other aspects of the system. It is also a choice made to ease TraceParts' adoption of the solutions we propose. Indeed, they are already used to Elasticsearch and have the skills in their teams.

Text-based CAD model retrieval

We refer to the current TraceParts CAD-content platform search system as text-based. It considers each document as self-containing all their metadata. Hence, each document indexed contains a set of metadata that we can break down into four sections: the metadata about the document's catalogue and categories, the data provider ones, and the Part Family and Part Number ones. Each document metadata section contains, at minimum, a field with a one-sentence length description that looks much like a title. All the corpus document's fields are accessible from any www.traceparts.com product page[5]. Figure 6.5a depicts the text-based system next to the concept-based one (figure 6.5b).

[5]An example of product page is https://www.traceparts.com/fr/product/din-brides-din-6314-plates?Product=32-24012013-084776 (Accessed on Thursday 3rd October, 2024)



(a) Text-based document retrieval system.

(b) Concept-based document retrieval system.

Figure 6.5: Text-based vs concept-based document retrieval system.

The empty arrows denote data flows. The thin black arrows illustrate the back and forth between the user and the document retrieval system. Squared boxes are processes and database symbols depicts indexes.

Concept-based CAD model retrieval

The concepts-based system depicted in figure 6.5b separates the notion of a document from the concepts describing it. Our experiments consider Part Families, data providers, categories and catalogues as concepts. The catalogues are classification trees; the top category is the catalogue name. The data providers are the manufacturers and suppliers providing the catalogue content. Finally, the Part Families are the parts. Part Numbers are Part Families' specific configurations, i.e., their instances. Ultimately, our concept set contains about 1.3 million distinct concepts. Note that with the latter point of view, the concept set contains the Part Family corpus documents.

We design the concept-based CAD model retrieval system using a two-step approach. The first step extracts concepts represented by the user text search. In theory, this step should be performed by an entity extraction system. However, we needed more time and resources to develop such a system. Therefore, we view this concept extraction task as an IR one and leverage the BM25 score to match user queries with their related concepts. In practice, we create 4 indices, one for each concept kind. The system queries each concept kind's index for the 10 top matching concepts, which act as our simplified user query concept extraction.

The documents, the Part Families or Part Numbers, depending on the experiment, are also indexed based on their representative concept. Hence, a second query fetches the ranked list of relevant documents using the previously selected concepts.

Knowledge Graph-based CAD model retrieval

The concept-based CAD model retrieval approach introduces a concept extraction step before querying for the corpus documents. However, the document retrieval step uses the concepts as it is. The KG-based CAD model retrieval approach depicted in 6.7 adds a third step leveraging the relations amongst concepts to enrich the concept-based document retrieval.

We use the initially extracted concepts as entry points into the KG and perform a rule-based graph traversal to fetch complementary concepts. In our experiments, we defined two rules:

- for category concepts, we fetch the subcategories that categorise some part families.

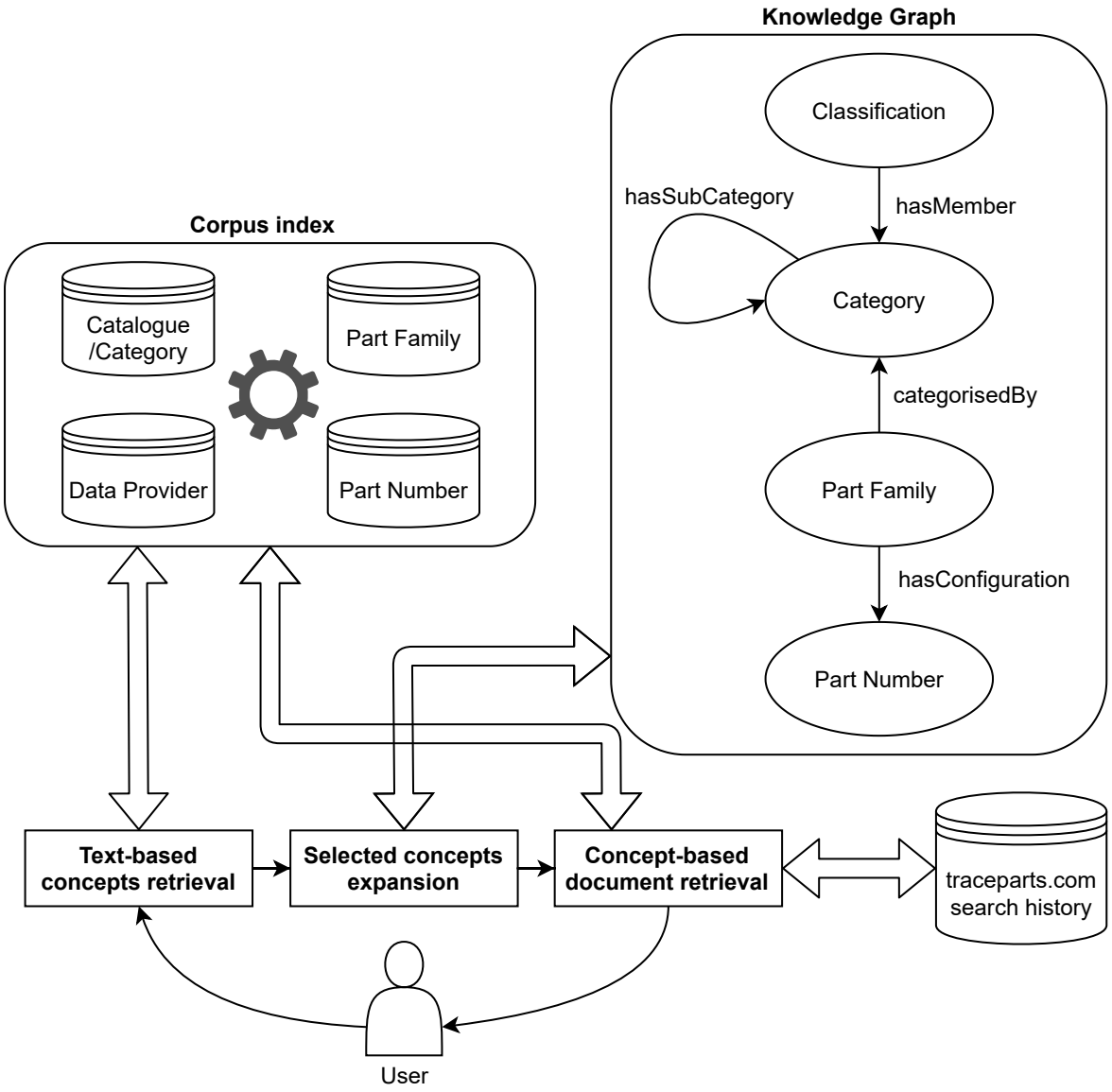


Figure 6.7: KG-based document retrieval system with user search history.

The empty arrows denote data flows. The thin black arrows illustrate the back and forth between the user and the document retrieval system. Squared boxes are processes and database symbols depicts indexes.

- for part numbers, we fetch their part family.

The enriched concepts are used for the final document retrieval step, similar to the concept-based approach. For implementation reasons, we could only evaluate the KG-based approach on the Part Family corpus in our experiments.

Adding implicit knowledge

In the last system, we experiment with integrating implicit knowledge into the retrieval process. We add the influence of the search history as a reranking step.

We extract an older version of the test examples dataset we use for evaluation and process it to associate search tokens with the downloaded documents. We extract the documents and the number of times a search containing the token leads to the document download. In the retrieval process, we rerank the search results by better ranking the search results documents that were downloaded previously with the same search query tokens. The rank-boosting applied directly depends on the document count associated with the token and the document.

Though the implicit knowledge comes from the same data source, in our experiments, we carefully select a set of search examples distinct from the one we use for evaluation. We experimented with adding such implicit knowledge to the text, concept and KG-based approaches. In figures 6.5b and 6.7, the search history is depicted with the database symbol at the bottom right.

6.5 Results

In this section, we present the experiment results. First, we compare the Part Number and Part Family corpora by analysing the quality of the results from the text-based and concept-based systems. We then focus on the Part Family corpus to compare the different approaches.

We evaluate the different CAD model retrieval systems with 5 k values. We consider the top 5, 25, 50, 100 and 350 search results. 5 is approximately the number of search results immediately visible on the user screen. 25 is approximately the number of results a user sees when shortly scrolling down the list. On www.traceparts.com, the search results pagination is almost transparent since the user can continuously scroll down. However, 50 is the number of search results per page, i.e., the system issues a new request to Elasticsearch to fetch documents 51 to 100. Hence, 100 corresponds to 2 search results pages. Finally, in a previous internal analysis, it has been discovered that, on average, a user goes through 7 search results pages. Hence, 350 corresponds to 7 pages of 50 documents each.

6.5.1 Part numbers vs part families corpus

Table 6.3 compares the text-based system performance on the Part Number and Part Family corpora. The table shows the values for each metric (MAP@k, MRR@k, and BM@k) horizontally and vertically for each k value.

Looking vertically at the values for different top k results, we notice that they do not vary much for the MAP@k and MRR@k metrics. However, they significantly increase for the BM@k one. It suggests that the positive documents are present in the result set but must be better ranked. These numbers support the intuition that TraceParts CAD-content platform search engine selects the proper documents but does not rank them well. In other words, the text-based system probably has a good recall but a low precision.

As we move down the results list, the AP@k value for positive documents decreases. This implies that when k increases, the new positive documents considered have less impact on the MAP@k value reported in the tables. The same trend can be observed with the MRR@k values. For instance, if the highest ranked document for a search is at the 10^{th} position, its RR@k value is $\frac{1}{10}$. Consider another search for which the top-ranked positive document is at the 30^{th} position. For the MMR@25 value, this search counts as a 0 in the mean. However, for the MRR@50, it counts as

Text-based search						
	MAP@k		MRR@k		BM@k	
Corpus → / @k ↓	PN	PF	PN	PF	PN	PF
@5	0.041	0.061	0.054	0.082	**0.082**	0.114
@25	0.044	0.064	0.059	0.085	**0.129**	0.148
@50	0.045	0.064	0.059	0.085	**0.142**	0.157
@100	0.045	0.064	0.059	0.085	**0.151**	0.161
@350	0.045	0.064	0.059	0.085	**0.161**	0.164

Table 6.3: Text-based search comparison of Part Number (PN) and Part Family (PF) results levels.

a $\frac{1}{30}$, which has a much lower impact on the mean than $\frac{1}{10}$. This trend indicates that as the k value increases, the MAP@k and the MMR@k values do not increase significantly. Therefore, for higher top k values, it is more informative to consider the BM@k values, where each positive document has the same impact in the mean.

The values are consistently higher for the Part Family corpus than the Part Number one. This is unsurprising, considering the Part Number corpus has fewer potential matching documents than the Part Family one. There could be thousands in the Part Family one when there is precisely one positive document in the Part Number corpus. This is because a Part Family can have many distinct configurations, i.e., Part Numbers. Simply put, there is a 1 to n relation from a Part Family to a Part Number and a 1 to 1 relation from the Part Number to the Part Family, i.e. a Part Number has only one corresponding Part Family.

Table 6.4 is the same as table 6.3 for the concept-based CAD model retrieval approach. We observe the same behaviours as the text-based approach when looking vertically at the different top k values. Nevertheless, we notice a more significant increase in the BM@k values from k equal 5 to 25. It suggests that a significant portion of positive documents are in the top 25 results for the concept-based system.

Concept-based search						
	MAP@k		MRR@k		BM@k	
Corpus → / @k ↓	PN	PF	PN	PF	PN	PF
@5	0.107	0.152	0.132	0.184	**0.182**	0.243
@25	0.114	0.159	0.140	0.192	**0.271**	0.334
@50	0.115	0.160	0.142	0.194	**0.308**	0.371
@100	0.116	0.161	0.142	0.194	**0.345**	0.403
@350	0.116	0.161	0.142	0.194	**0.406**	0.429

Table 6.4: Concept-based search comparison of Part Number (PN) and Part Family (PF) results levels.

Unsurprisingly, the Part Family corpus values are much higher than those of the Part Number one. However, the absolute difference is more significant for the concept-based approach than the text-based one. For the MAP@k, it suggests that the concept-based approach ranks higher more diverse documents that are also pertinent.

Comparing both tables, the performance of the concept-based approach in table 6.4 is generally better by a factor of more than 2 than the values for the text-based system in table 6.3.

The following results focus on the Part Family corpus and compare the different approaches. We do not discuss the results on the Part Number corpus as they demonstrate behaviours similar to the experiments on the Part Family corpus.

6.5.2 Comparing search engines

Tables 6.5 and 6.6 present the MAP@k, MRR@k, and BM@k values for different systems first without and then with leveraging the search history implicit knowledge. The metric values are given for each top k value. In this section, we discuss the latter search results quality measures and support our analysis with the search results lists quantities presented in table 6.7.

Text-based search

Looking at the tables, we first note that all tested approaches perform better than the original text-based system. Focusing more specifically on table 6.7, the text-based baseline is restrictive during the document selection phase. Indeed, almost two-thirds of the tested searches did not retrieve documents. Among the third example searches leading to some retrieved documents, many searches led to an extensive result list. It is a potential indicator for noisy documents. Considering the BM@k values for the text-based system and the third of the searches leading to more than 400 selected documents, we infer that the ranking is reasonably good for only one-sixth of the example searches.

	Text-based system (baseline)			**Concept-based system**			**KG-based system**		
@k ↓	**MAP@k**	**MRR@k**	**BM@k**	**MAP@k**	**MRR@k**	**BM@k**	**MAP@k**	**MRR@k**	**BM@k**
@5	0.061	0.082	0.114	**0.152**	**0.184**	**0.243**	0.115	0.142	0.202
@25	0.064	0.085	0.148	**0.159**	**0.192**	**0.334**	0.122	0.151	0.290
@50	0.064	0.085	0.157	**0.160**	**0.194**	**0.371**	0.123	0.151	0.319
@100	0.064	0.085	0.161	**0.161**	**0.194**	**0.403**	0.123	0.152	0.343
@350	0.064	0.085	0.164	**0.161**	**0.194**	**0.429**	0.124	0.152	0.364

Table 6.5: Comparing text, concept, and KG-based systems on the Part Family (PF) corpus for different k values.

By integrating the implicit knowledge from the search history into the text-based search (table 6.7), we observe a significant improvement in the search results quality, which is at least double. This integration, implemented as a reranking step, also acts as a retrieval one. For each search text token, the documents matching in the search history are included in the selected documents, even though the document might not have been selected. Then, the new document list is ranked. The search results quality values suggest that a substantial part of the performance enhancement comes from the search history. It also indicates that users often express similar search intent with various search variations.

The original text-based search system enforces that all the search query tokens are present in all the document text fields combined for a document to be selected. This explains the high number of searches without any document retrieved. It also suggests that many of those searches without any retrieved documents have positive documents that can be retrieved from some search tokens. One way to augment the number of searches with retrieved documents is to ease the constraint of having all the search tokens present to having at least one. However, such an approach leads to the retrieval of many noisy documents, requiring a performant ranking. More retrieved documents also imply higher resource consumption.

Concept-based search

When the search history knowledge is not integrated, the concept-based search outperforms the text and KG-based approaches. In table 6.7, we observe that first fetching concepts to use them in the document retrieval step significantly reduces the number of searches without any retrieved documents. Interestingly, the selected document lists are not much larger than for the other approaches, indicating that the concept selection step does not necessarily lead to selecting many

noisy documents at the retrieval stage. Most of the retrieved document lists for the concept-based system are smaller than 400 (or 7 pages).

Integrating the search history knowledge into the concept-based system (table 6.6) brings more positive documents in the results, according to the better BM@k values. However, these documents are not well-ranked. We can assume that a significant part of the relevant documents in the latter are from the search history. They are, however, not ranked higher in the results.

	Text-based system with search history			**Concept-based system with search history**			**KG-based system with search history**		
@k ↓	**MAP@k**	**MRR@k**	**BM@k**	**MAP@k**	**MRR@k**	**BM@k**	**MAP@k**	**MRR@k**	**BM@k**
@5	0.142	0.169	0.251	0.129	0.153	0.223	**0.170**	**0.202**	**0.291**
@25	0.157	0.185	0.417	0.143	0.167	0.381	**0.186**	**0.218**	**0.471**
@50	0.160	0.187	0.493	0.146	0.169	0.452	**0.189**	**0.221**	**0.552**
@100	0.162	0.188	0.562	0.147	0.170	0.517	**0.191**	**0.222**	**0.624**
@350	0.163	0.188	0.652	0.148	0.170	0.600	**0.192**	**0.222**	**0.715**

Table 6.6: Comparing text, concept, and KG-based systems with search history knowledge on the Part Family (PF) corpus for different k values.

KG-based search

	No results	**Less than 400 results (non empty)**
Text-based system (baseline)	64.48%	35.44%
Concept-based system	11.43%	**88.36%**
KG-based system	9.15%	86.09%
Text-based system with search history	18.38%	46.14%
Concept-based system with search history	22.35%	42.97%
KG-based system with search history	**8.10%**	51.59%

Table 6.7: Comparing all search systems results set on the Part Family (PF) corpus.

The KG-based search system introduces a concept enrichment step before retrieving the documents. It is similar to performing a query expansion. As for the concept-based approach, it could likely imply selecting more documents. Table 6.7 shows that even fewer searches are left without search results than for the concept-based search. However, the table also shows that most search results lists are smaller than 400 documents.

Without the search history knowledge, the KG-based approach is slightly worse than the concept-based one. However, when including the latter knowledge, the KG-based approach becomes the best-performing system by far. Again, the search history knowledge document retrieval part plays a significant role. However, the KG-based approach handles document ranking better.

Search history implicit knowledge

As we have already discussed in the previous sections, the search history knowledge integration is implemented as both a retrieval and reranking phase. Adding such knowledge generally augments the search results quality. It seems that a significant part of the increase in results quality comes from the documents retrieved through the search history knowledge.

In table 6.7, adding the search history drastically increases the number of selected documents. However, based on the approach we chose to retrieve tables 6.7 values and the search history integration, we can infer that the selected documents lists with more than 400 documents do not con-

tain a lot more than 400 documents. Indeed, the document lists that were smaller than 400 documents in the experiments without the search history only have the documents from the search history as new ones in the second experiment set. It also suggests that the document lists are close to 400.

6.6 Conclusion and future works

This PhD work has been conducted in collaboration with TraceParts. The industry use case is enhancing the TraceParts CAD-content platform search engine. From a business standpoint, we aim to reduce the average number of user iterations before reaching a download.

In this chapter, we introduced the different systems we proposed to tackle TraceParts search engine challenges. Our knowledge-based approaches aim to bring formal knowledge representation to support the retrieval of CAD models. We decided to concentrate on the textual content. Hence, our general approach can be summarised as moving from an existing text-based search to a concept-based one.

One of the main challenges for TraceParts' CAD model retrieval from text searches is the 25 different languages the platform supports, combined with the different domain-specific vocabularies and text structures. In our approaches, we tackle these issues by formalising the user searches. That is to say, we first extract the search concepts to then retrieve documents based on them. In practice, this means better structuring both the corpus documents and the queries. However, in our implementations, we did not optimise the concept extraction.

In practice, we moved the text variety issue from the document retrieval phase to the concept extraction phase. To focus on KG-based IR, we implemented the concept extraction task as an IR problem, retrieving concepts in a text-based IR fashion from the user search text. Hence, one area of focus for future work is to enhance the extraction of user search concepts.

The literature refers to extracting concepts from the text as Entity Linking. Much research exists on the topic, and many approaches could be explored. However, before complexing the concept extraction, we should experiment using search completion suggestions. Indeed, suggesting completion as the user types their search is a transparent way to leverage user explicit feedback.

The experiment results demonstrate the performance gain of adding a concept extraction phase and moving to a KG-based search. However, the current implementation of the KG structure is limited to 4 different kinds of concepts: catalogues, categories, data providers, and part families. Those can be used to implement initial user search completion suggestions. However, an extended study of the user searches and their concepts could enhance the KG content and, therefore, the KG-based search. Concepts like industrial standards, domain-specific notations, functions, materials, and so forth could be added.

Figure 6.8 illustrates an example of a KG used to extract concepts from a user search and contextualise it with related concepts. The user search "M10" is identified as a notation *M* followed by a number *10*. From the notation *M*, leveraging the KG, the system can infer that the user is looking for a screw with a particular thread diameter, and the number *10* denotes a thread diameter expressed in millimetres. The KG is structured as a set of taxonomies organising concepts hierarchically. The concepts are linked across taxonomies by non-transversal relations. We already mentioned this structure in chapter 5, which is derived from [LRR08]. Such KG structure is what we envision for future work expanding the KG used in our experiments.

The experiment results seem to suggest that the IR process should heavily rely on the search history implicit knowledge. However, from a business point of view, it is an approach that must be considered with great caution. Indeed, letting the search history knowledge influence the document retrieval might easily result in proposing always the same set of documents. While, from a user perspective, it might be a good approach, from a business point of view, it means not putting forward new catalogues and, therefore, implicitly discarding new TraceParts customers. To address such a lack of diversity risk in search results, we can design a search history document boosting based on a dataset updated regularly with a sliding window to define.

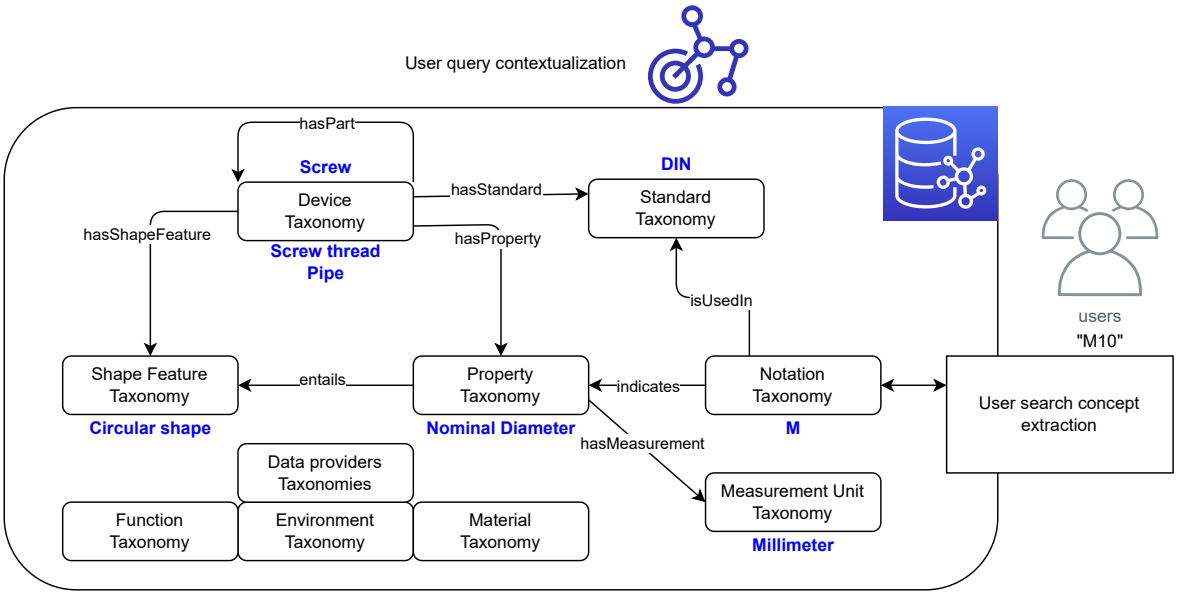


Figure 6.8: Example of a KG used to extract concept from a user search and contextualise it with related concepts.

The arrows illustrates non taxonomic relations between taxonomy concepts. Round boxes denote the taxonomies grouping concepts and organising them hierarchically. Squared boxes are processes.

Finally, each proposed system contains many variables acting on each component. These variables could be optimised using a supervised approach.

Part V

Conclusion and future works

We explored Knowledge Graph-Based Systems (KGBS) for Information Retrieval (IR) in these industrial research works. Our use case considered a technical document corpus composed of Computer-Aided Design (CAD) models and their descriptions. The use case and corpus come from our industrial partner TraceParts and concentrate on their CAD-content platform search engine[6]. We focus on their descriptive texts rather than directly leveraging the CAD models. In this concluding chapter, we summarise these research works and their findings before exploring potential directions for future work. Figure 6.9 summarises our works, highlighting the main contributions in red boxes.

Conclusion

We adopted a top-down approach to describe our works, beginning with two literature review chapters. We first explored KGs and ontologies and how they relate. Our historical review and thorough study of the different definitions in the literature for ontologies and KGs highlighted a need for clarity regarding the exact definition of a KG. Hence, we derived our unifying KG definition, which considers ontologies a component of KGs. Our KG definition illustrated in the top-right part of figure 6.9 extends [HBC+21] authors' one, distinguishing between the data graph and the domain graph. We define a KG as is a graph intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities. The graph of data (aka data graph) conforms to a graph-based data model. Knowledge refers to something that is known. A KG is a data graph potentially enhanced with representations of schema, context, ontologies and/or rules forming the domain graph. Hence, the KG is composed of a data and domain graph.

The Semantic Web standards influenced our works, particularly for implementations. Therefore, we also mapped Semantic Web standards with our KG definition. Our KG definition helps us better understand and navigate the complex merging research domains of KGs and ontologies. In our contribution chapter introducing our IR ontology, we also illustrated our definition with an RDF-based example. Our KG definition fits these works' KG and ontology use case and the ones we have encountered in our reviewed literature. However, further exploration, particularly outside of our IR-focused use case, is required to determine the robustness of our KG definition.

We then explored the literature on IR, focusing on KG-based IR. After a historical perspective and an introduction of the necessary IR components, we explored state-of-the-art methods leveraging KGs for IR. In particular, we explored the scientific literature using the term *knowledge graph* and using the term *ontology* separately. This exploration showed that KG-focused literature tends to concentrate on the data graph of our KG definition, i.e. the graph data structure, overlooking the domain graph. On the other hand, the ontology-focused one tends to leverage the domain graph defining structured vocabularies. Though the latter ontology literature aims at leveraging deductive reasoning, we highlighted that such reasoning is often limited to hierarchical relations. In summary, state-of-the-art KG-based methods for IR concentrate on the data graph in our KG definition with limited use of the domain graph. This a limitation we addressed with our IR ontology.

In our first contribution (chapter 3), we motivated a need for more work exploring KGs as part of an information system. We argued that the methods presented in the scientific literature focus on how the KG supports the considered use cases overlooking the KG itself. Hence, we first explored KGs as a component of an information system. We put aside our IR use case and introduced a KGBS architecture relating knowledge acquisition, modelling, and consumption arranged around the KG. The KGBS architecture helps visualise the components gravitating around a KG in an information system regardless of the practical use case. The architecture thereby helps better understand the role of KGs within a system. Our other contributions followed this system architecture, addressing knowledge acquisition, modelling, and consumption. In these works, we leave aside the knowledge validation, which we will consider in future work directions.

[6] www.traceparts.com (Accessed on Thursday 3rd October, 2024)

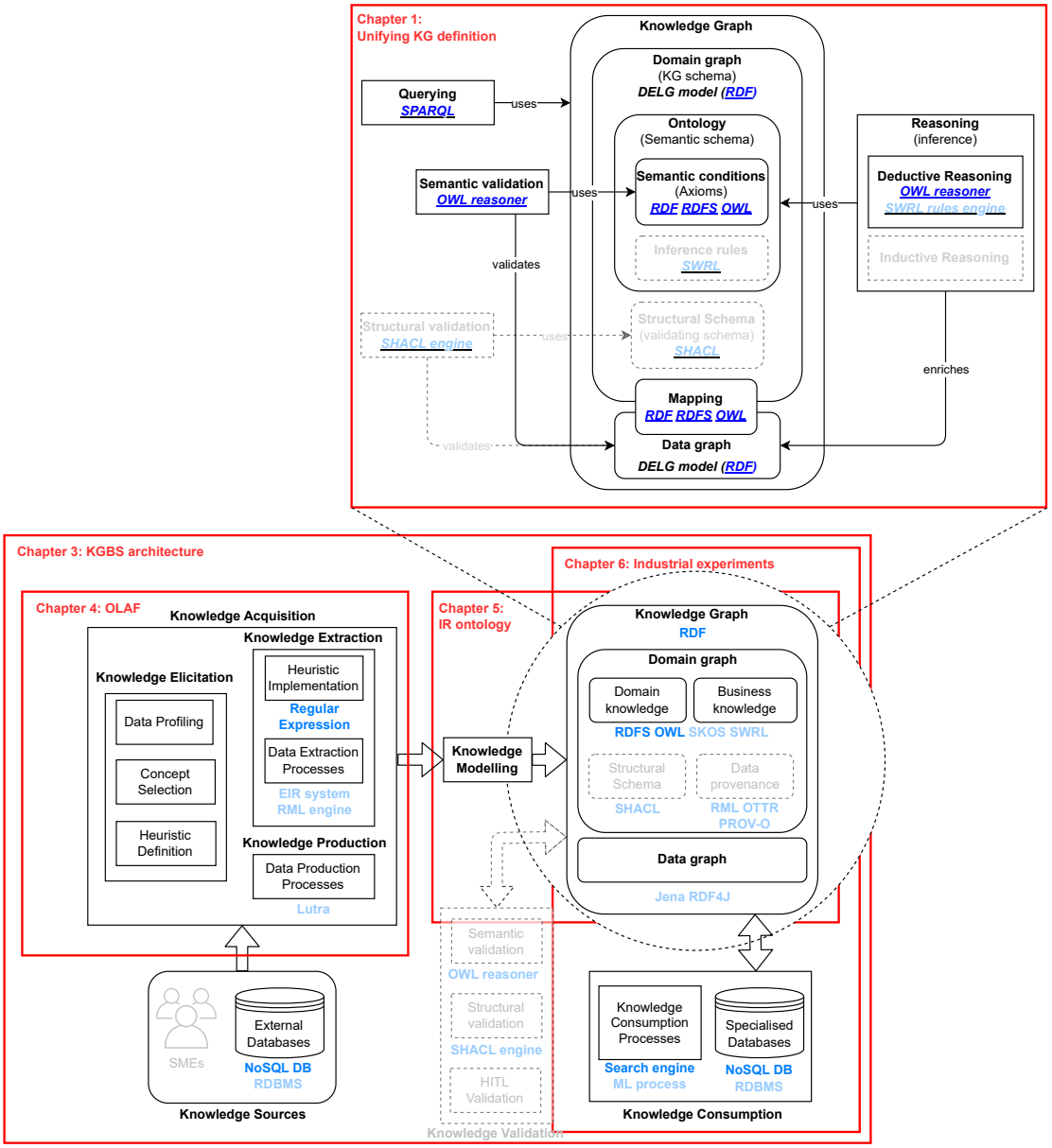


Figure 6.9: Summary of our industrial research works contributions. It includes our Knowledge Graph-Based System architecture at the bottom and our KG definition at the top right.

Square boxes refer to activities. Round boxes depict containers. The boxes imbrications denote sub-activities and sub-containers, respectively. Large empty arrows illustrate data flows and thin black ones actions. Large red boxes denote the components each contribution addresses. Some candidate technologies for implementing each component are mentioned in bold blue letters. The architecture parts turned into dashed and light colours are the parts left out in these works. (SMEs: Subject Matter Experts; HITL: Human In The Loop)

We did not have a pre-built KG or access to domain experts to construct it. Open KGs are too broad or do not perfectly match our needs, and domain experts' time is expensive. Hence, we addressed our KGBS architecture's knowledge acquisition component by designing our Ontology Learning Applied Framework (OLAF) collaboratively with a research group member. We implemented our framework as an open-source Python library to explore and evaluate different combinations of methods addressing each OL subtask. We used OLAF to learn ontologies from text automatically and built two ontologies to assess the OLAF's pertinence, usability, and modularity. Though our experiments demonstrated our OLAF implementation functioning and modularity, they considered small ontologies and corpora. The framework is a great toolbox and would benefit from further method implementations and experiments on larger corpora and targeted ontology.

We then focused on knowledge modelling for IR. As mentioned, we found a limited use of deductive reasoning in the literature focusing on methods leveraging ontologies for IR. Hence, we implemented our IR ontology with OWL and demonstrated its usage with an OWL reasoning-powered IR system. While state-of-the-art approaches leverage reasoning offline, our approach explored OWL reasoning at runtime for IR. While demonstrating our IR ontology, we illustrated an RDF, RDFS and OWL-focused implementation of our KG definition by aligning each set of RDF triples in our experiment with our KG definition's components. We discussed some potential limitations, particularly regarding the runtime reasoning at scale, which requires further exploration.

Finally, through our industrial experiments, we explored an example of a knowledge consumption use case, IR and, more specifically, technical document retrieval. We tackled the CAD-model retrieval challenge our industrial partner TraceParts faces by implementing a KG-based approach at scale and using real-world data. Our experiments illustrated moving from an existing text-based technical document retrieval system to a KG-based one. We leveraged real-world TraceParts' CAD-content platform user interactions to evaluate our KG-based IR system proposal. The evaluation shows better quality results for the KG-based approach, which combines expert knowledge and implicit knowledge from user feedback.

Future works

Our research works combine theoretical works framing long-term visions and practical implementations, paving the path for short and mid-term realisations. Our KG definition and KGBS architecture align existing theoretical works with our long-term visions and short-term use case. Our KG definition reconciles the literature's confusing usage of the terms *knowledge graph* and *ontology*, laying out the foundations to understand our research works. The KGBS architecture puts this KG definition in an information system context before diving into methods to address particular architecture parts. While our works propose some solutions for the KGBS architecture components, we could not address each. Moreover, the directions we investigated still require further exploration to properly tackle the knowledge extraction and modelling processes. Our works only consider the IR use case for knowledge consumption and leave out the knowledge validation components. The pieces we did not address in these works are shown in the summary figure 6.9 by the lighter colours.

Our approaches tackle knowledge extraction, modelling, and consumption separately. The long-term objectives future works are to implement our KGBS architecture fully and on a single use case to evaluate its pertinence. The works implementing our architecture should also make it evolve. Such future works should enrich our KGBS architecture to reach a level of detail enabling the implementation of particular components individually while ensuring easy integration within the broader system. The first step is to implement the end-to-end architecture on a simple use case with a small-size corpus to avoid scaling issues. One challenge to finding such a corpus is that it should be annotated for the task to evaluate. It would be even better if annotation existed at the architecture component level so they could be evaluated independently and together. However, we are not aware of such an ideal annotated corpus. Nevertheless, IR is an interesting knowledge consumption use case with some existing corpora. One promising work in this direction is Chi et

al. one [CJH19] in which the authors develop a base domain ontology from the NCREE (National Center for Research on Earthquake Engineering) collection to tackle an IR use case. According to the authors, the NCREE collection contains 225 technical reports annotated with some concepts and a topic. The collection is, however, not directly linked by the authors.

Once each architecture component is implemented and integrated, two potential directions exist. The first is implementing other knowledge consumption use cases to demonstrate the viability of our KGBS architecture. The second direction is evaluating the architecture at scale with much larger corpora. For each of our proposed approaches, future works should implement some experiments at scale. Indeed, we only tested at scale our KG-based IR system tackling our industrial partner's CAD-model retrieval use case. Our OL framework and our IR ontology experiments still need some large-scale evaluations.

However, reaching such a long-term objective requires smaller steps. The KGBS architecture addresses a broad scope. Hence, it encompasses many different domain areas that are direct or satellite elements of our works. In our works, we proposed some approaches for the components directly part of our research focus. We later discussed specific future directions for those components. First, let us discuss aspects we voluntarily overlooked but are critical.

The most critical task related to KGs is Entity Liking (EL). EL aims to link entities in a KG with their occurrences in corpus documents or any content considered for the use case. This task is essential to leverage a KG supporting any use case. EL was needed in our IR task to semantically index our corpus and link entities in the user's natural language queries. We focused on building the KG and implementing a KG IR system. Hence, we worked around the EL complexity by viewing this task as a separate IR task, which we tackled with a traditional BM25 approach. We explored some existing EL methods but found that state-of-the-art ones leveraged a wealth of annotated examples to train neural networks. Such annotated data requirements made it complex for us to adapt these solutions to our use case.

While annotated examples are critical to reaching such state-of-the-art methods performances, we found that the latter methods overlooked the practical iterative process required to reach such a level. Moreover, neural networks-based approaches require context only found in large documents such as news articles-size content. Such context was lacking in our short descriptions and keyword user queries. Hence, a short-term objective is to implement and evaluate base EL solutions using the linguistic content directly available in the KG. We started to follow this approach with our research team colleague when implementing our OLAF framework. We began a side open-source project named *Buzz-EL*[7]. While constructing an ontology from text using OLAF, we keep the link between the learned concept and its origins in the text. We thought we could leverage such links between learned concepts and their origin linguistic realisation to create base entity linkers for the learned ontology. Here, *base entity linker* refers to an entity linker based on text matching. Such a tool would benefit methods building more performant entity linkers with better generalisation capabilities by providing the base training examples. We can also automate the construction of such base entity linker from any KG.

A long-term objective with this *Buzz-EL* project is to initiate a self-enhancing KG and ontology learning process. While a base entity linker could be used directly in an application, we could also use it to facilitate the OL concept and relation extraction tasks. Regarding relation extraction, while hierarchical relations are well-studied in the literature, non-hierarchical relation extraction methods are limited to particular use cases such as mereological relations.

Another OL task largely overlooked in the literature is axiom extraction. The authors of [AWK+18] identified Inductive Logic Programming (ILP) as a promising approach to axiom extraction. ILP leverages a set of positive and negative examples to search for the axioms satisfying all positive examples while not satisfying negative ones. Our works address axiom extraction by focusing on OWL-expressed axioms and leveraging Ontology Design Patterns (ODPs) results [GPSS09]. However, OPDs lead to the OL process generating particular OWL constructs from the extracted concepts, and relations still need to be manually selected. A mid-term research direc-

[7] https://github.com/schmarion/buzz-el (Accessed on Thursday 3rd October, 2024)

tion is to define how the use case applications targeted when starting a KG project could affect the kinds of learned axioms. We believe addressing this question would be a significant step towards OL going beyond tools to help the ontology engineer in its knowledge modelling task and truly learning minimum viable ontologies.

Let us touch on one last future work direction for research areas closely related to our works before diving into directions specific to our proposals. During the works we presented, we saw the recent rise of generative Large Language Models (LLMs) and their many applications, such as Retrieval Augmented Generation (RAG), which is closely related to our IR focus. LLMs are large pre-trained language models initially trained on generic tasks, so they learn how to manipulate a language. They then have been fine-tuned to address specific tasks, amongst which responding to prompts. More recently, they have shown promising results based solely on their language generation capabilities in response to prompts. These prompt-based generation capabilities have been cleverly repurposed to address specific tasks. RAG models combine pre-trained parametric and non-parametric memory for language generation [LPP+20]. Here, the parametric memory is the internal LLM weights, and the non-parametric memory is typically a large corpus of textual documents. With the recent interest in LLMs, the KG community started exploring KGs as the non-parametric memory for LLMs. In our research works, we purposely avoided LLMs and RAG to stay focused as our works were already too advanced to change the orientation. However, LLMs and RAGs should be explored for future work. Future works should explore RAG as a knowledge consumption use case in our KGBS architecture and investigate the use of LLMs for each of our OLAF components. The colleague with whom we developed OLAF already began to explore such generative LLM applications to OLAF components[8] and our OLAF implementation contains specific interfaces to integrate generative LLMs.

Let us now dive into directions developing specifically the approaches we explored in our research works. We implemented our framework OLAF as an open-source Python library to serve as a toolbox for anyone to combine their approaches to each subtask of the OL process. Hence, future works should, in the short term, prioritise making this open-source library more accessible. Though we have already put great efforts into documenting, testing and demonstrating our code, we should continue in this direction. Providing more accessibility aims to gather a community around our OLAF implementation to create benchmarks. These benchmarks are essential to compare and evaluate methods. Hence, they should be a mid-term focus for future work. While prominent in other communities, such as NLP or computer vision, the OL community introduces approaches evaluated on custom ontologies rather than community benchmarks. Our experiments with OLAF are limited as they consider toy ontologies and small corpora. However, they demonstrated the modularity of OLAF implementation and its pertinence as a toolbox. Hence, new methods for each component should be implemented in the future to test new combinations, i.e., pipelines fitting particular use cases. Existing methods would also benefit from some optimisations to enable better scaling.

We could not address some OLAF components properly during our work. Hence, they require further exploration. We already mentioned relation and axiom extraction. However, future works should also investigate automatic methods to evaluate the learned ontologies. Ontology evaluation can consider different aspects of the ontology, such as the consistent URI construction following rules and avoiding ontology constructs known as bad practices. An interesting research to explore are the ones around the OntOlogy Pitfall Scanner [PVGPSF14] and the corresponding OWL ontology evaluation tool OOPS![9]. On the operational aspects, some closely related works are the *SemOps* practices currently explored in the industry[10]. *SemOps* is the contraction of *Semantic* and *DevOps*. The term refers to approaches aiming at adapting the software industry practices known as *DevOps* to ontology engineering. These methods should eventually be integrated into the KGBS

[8] https://github.com/wikit-ai/olaf-llm-eswc2024 (Accessed on Thursday 3rd October, 2024)

[9] https://oops.linkeddata.es/

[10] https://www.semanticarts.com/the-data-centric-revolution-the-role-of-semops-part-1/ (Accessed on Thursday 3rd October, 2024)

architecture future works directions mentioned above with Semantic Web languages such as the SHApes Constraint Language (SHACL) [KK17].

In our last contribution (chapter 6), we tackled our industrial partner IR use case by proposing our practical approach to moving from a text-based IR system to a concept-based one. We use the term concept to highlight the difference between text and concept matching. However, the final system showing the best performances is a KG-based one. One of the main challenges for TraceParts' CAD model retrieval from text searches is the 25 different languages the platform supports, combined with the different domain-specific vocabularies and text structures. We tackled these issues by extracting the search concepts and retrieving documents based on them. However, as we already mentioned, we did not optimise the concept extraction in our implementations. We implemented it as a simple IR task leveraging classic BM25-based retrieval. We could explore many different EL approaches. However, future works should in the short-term first experiment using search completion suggestions. Indeed, suggesting completion as the user type is a transparent way to leverage explicit user feedback.

The experiment results demonstrate the performance gain of adding a concept extraction phase and moving to a KG-based search. However, the current implementation of the KG structure is limited to 4 different kinds of concepts. We can use those concepts instances to implement initial user search completion suggestions. However, in a mid-term vision, an extended study of the user searches and their concepts could enhance the KG content and, therefore, the KG-based search. For instance, in our particular CAD model retrieval use case, we should add concepts like industrial standards, domain-specific notations, functions, and materials as illustrated in figure 6.8 introduced in the conclusion of chapter 6.

In figure 6.8, the KG is structured as a set of taxonomies organising concepts hierarchically. The concepts are linked across taxonomies by non-transversal relations. This KG structure is the one we use in chapter 5 to demonstrate our IR ontology. The structure is derived from [LRR08] and has already proven relevant to an IR use case. We envision such a KG structure for future works, expanding the KG used in our experiments. Such KG structure is the one we rely on when introducing our IR ontology to implement an OWL-reasoning-powered classification search at runtime. A direction to explore is to combine the IR ontology with an existing one. The IR ontology should easily merge with any ontology to add the capability of browsing the ontology categories. It should specifically be valid for the KG structure we just mentioned, i.e., a set of interlinked concept hierarchies. Hence, we should explore enhancing our industrial experiments with the IR ontology. Such IR ontology integration would also evaluate its scalability, which is a concern when considering reasoning at runtime.

In the last section of chapter 5, we discuss a direction to extend our IR ontology that semantically expresses incoherent searches. We should explore distinguishing a user search for which the IR system has no document from one without any existing document. In the former case, a document might exist, but the system is unaware of it. In the latter case, the search is inconsistent; hence, the system knows no document corresponding to the search can exist. It corresponds to applying the open-world assumption directly to IR.

To conclude on a general remark, these works explored the complexity of a KG-based system and implemented some parts of such a system before applying them to an IR use case. While we explored promising approaches to the different KGBS architecture components, some are very early-stage works demonstrated with toy examples. The various moving parts of the system can be optimised individually while analysing the whole system's impact. Each proposed method contains many variables acting on each component. We should explore fine-tuning these variables using supervised methods.

Bibliography

[Ack89] Russell L Ackoff. From data to wisdom. *Journal of applied systems analysis*, 16(1):3–9, 1989. 35

[AGCR20] Nathalie Aussenac-Gilles, Jean Charlet, and Chantal Reynaud. *Knowledge Engineering*, pages 733–768. Springer International Publishing, Cham, 2020. 47

[AHG20] Dean Allemang, Jim Hendler, and Fabien Gandon. *Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL*, volume 33. Association for Computing Machinery, New York, NY, USA, 3 edition, 2020. 42, 43, 105, 106, 109, 110, 123, 124

[ANS10] Guidelines for the Construction, Format, and Management of Monolingual Controlled Vocabularies. Standard, National Information Standards Organization, May 2010. 40

[AWK+18] Muhammad Nabeel Asim, Muhammad Wasim, Muhammad Usman Ghani Khan, Waqar Mahmood, and Hafiza Mahnoor Abbasi. A survey of ontology learning techniques and applications. *Database*, 2018:bay101, 10 2018. 47, 48, 50, 94, 150

[Bak87] René Ronald Bakker. *Knowledge Graphs: representation and structuring of scientific knowledge*. 1987. 30

[BBLPC14] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. RDF 1.1 Turtle, Terse RDF Triple Language. W3c recommendation, w3c, february 2014. 42

[BCC16] Stefan. Büttcher, Charles L. A. Clarke, and Gordon V. Cormack. *Information Retrieval, Implementing and Evaluating Search Engines*. The MIT Press, 2016. 136

[BEP+08] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. Association for Computing Machinery. 67

[BG14] Dan Brickley and R.V. Guha. RDF Schema 1.1. W3c recommendation, w3c, february 2014. 5, 94

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *scientam*, 284(5):34–43, May 2001. 32, 41, 43

[BNCn00] Gilles Bisson, Claire Nédellec, and Dolores Cañamero. Designing clustering methods for ontology building: The mo'k workbench. In *Proceedings of the First International Conference on Ontology Learning - Volume 31*, OL'00, page 13–28, Aachen, DEU, 2000. CEUR-WS.org. 49

[BOS04] Paul Buitelaar, Daniel Olejnik, and Michael Sintek. A protégé plug-in for ontology extraction from text based on linguistic analysis. In Christoph J. Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *The Semantic Web: Research and Applications*, pages 31–44, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. 49

[BUGD+13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 2787–2795, Red Hook, NY, USA, 2013. Curran Associates Inc. 65

[BW23] Jesús. Barrasa and Jim. Webber. *Building Knowledge Graphs: A Practitioner's Guide*. O'Reilly, 2023. 33, 34, 38, 39, 50

[CBC+22] Vinay K. Chaudhri, Chaitanya Baru, Naren Chittar, Xin Luna Dong, Michael Genesereth, James Hendler, Aditya Kalyanpur, Douglas B. Lenat, Juan Sequeda, Denny Vrandečić, and Kuansan Wang. Knowledge graphs: Introduction, history, and perspectives. *AI Magazine*, 43(1):17–29, 2022. 23, 30, 33

[Cim06] *Ontology Learning from Text*, pages 19–34. Springer US, Boston, MA, 2006. 48, 49

[CJH19] Nai-Wen Chi, Yu-Huei Jin, and Shang-Hsien Hsieh. Developing base domain ontology from a reference collection to aid information retrieval. *Automation in Construction*, 100:180–189, 2019. 67, 68, 70, 150

[CL22] Simon Cox and Chris Little. Time Ontology in OWL. W3c recommendation draft, w3c, November 2022. 84

[CMBT02] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. A framework and graphical development environment for robust nlp tools and applications. In *Annual Meeting of the Association for Computational Linguistics*, 2002. 49

[CV05] Philipp Cimiano and Johanna Völker. Text2onto. In Andrés Montoyo, Rafael Muńoz, and Elisabeth Métais, editors, *Natural Language Processing and Information Systems*, pages 227–238, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 49, 50

[CWL14] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3c recommendation, w3c, february 2014. 5, 42

[DDA14] Jeffrey Dalton, Laura Dietz, and James Allan. Entity query feature expansion using knowledge base links. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 365–374, New York, NY, USA, 2014. Association for Computing Machinery. 64, 66

[DDF+90] Scott C. Deerwester, Susan T. Dumais, George W. Furnas, T. K. Landauer, and Richard A. Harshman. Indexing by latent semantic indexing analysis. *Journal of the Association for Information Science and Technology*, 1990. 55

[Deh22] Zhamak Dehghani. *Data Mesh*. O'Reilly Media, Inc., 2022. 105

[DGPN13] Francesco Draicchio, Aldo Gangemi, Valentina Presutti, and Andrea Giovanni Nuzzolese. Fred: From natural language text to rdf and owl in one click. In Philipp Cimiano, Miriam Fernández, Vanessa Lopez, Stefan Schlobach, and Johanna Völker, editors, *The Semantic Web: ESWC 2013 Satellite Events*, pages 263–267, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 49

[DSC12] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. W3c recommendation, w3c, September 2012. 85

[DuC13] Bob DuCharme. *Learning SPARQL, 2nd Edition*. O'Reilly Media, Inc., 2 edition, 2013. 43

[DVSC+14] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: a generic language for integrated RDF mappings of heterogeneous data. In Christian Bizer, Tom Heath, Sören Auer, and Tim Berners-Lee, editors, *Proceedings of the 7th Workshop on Linked Data on the Web*, volume 1184 of *CEUR Workshop Proceedings*, April 2014. 9, 85

[DZP10] Euthymios Drymonas, Kalliopi Zervanou, and Euripides G. M. Petrakis. Unsupervised ontology acquisition from plain texts: The ontogain system. In Christina J. Hopfe, Yacine Rezgui, Elisabeth Métais, Alun Preece, and Haijiang Li, editors, *Natural Language Processing and Information Systems*, pages 277–287, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 49, 50

[EB17] Faezeh Ensan and Ebrahim Bagheri. Document retrieval model through semantic linking. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 181–190, New York, NY, USA, 2017. Association for Computing Machinery. 64, 66

[EFK19] Deborah L. McGuinness Elisa F. Kendall. *Ontology Engineering*. Synthesis Lectures on Data, Semantics, and Knowledge. Springer, 2019. 11, 46, 47, 50, 78, 79, 81, 97, 107

[EW16] Lisa Ehrlinger and Wolfram Wöß. Towards a Definition of Knowledge Graphs. 2016. 33, 34, 39, 41, 50

[FAM00] Katerina Frantzi, Sophia Ananiadou, and Hideki Mima. Automatic recognition of multi-word terms: the c-value/nc-value method. *International Journal on Digital Libraries*, 3:115–130, 2000. 95, 96

[FG04] Hermine Njike Fotzo and Patrick Gallinari. Learning "generalization/specialization" relations between concepts: Application for automatically building thematic document hierarchies. In *Coupling Approaches, Coupling Media and Coupling Languages for Information Retrieval*, RIAO '04, pages 143–155, Paris, FRA, 2004. 95, 96, 99, 100

[FGS20] George Fletcher, Paul Groth, and Juan Sequeda. Knowledge scientists: Unlocking the data-driven organization, 2020. 79, 84

[FN98] David Faure and Claire Nédellec. A corpus-based conceptual clustering method for verb frames and ontology acquisition. In Paola Velardi, editor, *LREC workshop on Adapting lexical and corpus ressources to sublanguages and applications*, pages 5–12, Granada, Spain, Mai 1998. 49

[FSRN14] Marc Franco-Salvador, Paolo Rosso, and Roberto Navigli. A knowledge-based representation for cross-language document retrieval and categorization. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 414–423, Gothenburg, Sweden, april 2014. Association for Computational Linguistics. 65, 66

[FW16] Christina Feilmayr and Wolfram Wöß. An analysis of ontologies and their success factors for application to business. *Data & Knowledge Engineering*, 101:1–23, 2016. 39

[GLL+17] Ge Gao, Yu-Shen Liu, Pengpeng Lin, Meng Wang, Ming Gu, and Jun-Hai Yong. Bimtag: Concept-based automatic semantic annotation of online bim product resources. *Advanced Engineering Informatics*, 31:48–61, 2017. Towards a new generation of the smart built environment. 69, 70

[GLÖ09] Avigdor Gal, LING LIU, and M. TAMER ÖZSU. *Ontology Engineering*, pages 1972–1973. Springer US, Boston, MA, 2009. 47

[GLW+15] Ge Gao, Yu-Shen Liu, Meng Wang, Ming Gu, and Jun-Hai Yong. A query expansion method for retrieving online bim resources based on industry foundation classes. *Automation in Construction*, 56:14–25, 2015. 68, 69, 70

[GPB17] Jose Emilio Labra Gayo, Eric Prud'hommeaux, and Iovka Boneva. *Validating RDF Data*. Morgan & Claypool Publishers, September 2017. 85

[GPSS09] Aldo Gangemi, Valentina Presutti, Steffen Staab, and Rudi Studer. *Ontology Design Patterns*, pages 221–243. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 16, 85, 95, 109, 150

[Gro22] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview. W3c recommendation, December 11 20122. 45

[Gru95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5):907–928, 1995. 31

[GS21] Claudio Gutierrez and Juan F. Sequeda. Knowledge graphs. *Commun. ACM*, 64(3):96–104, feb 2021. 5, 30, 32, 42

[Gua98] Nicola Guarino. Formal ontology and information systems. 1998. 39, 80

[GZQ+22] Qingyu Guo, Fuzhen Zhuang, Chuan Qin, Hengshu Zhu, Xing Xie, Hui Xiong, and Qing He. A survey on knowledge graph-based recommender systems. *IEEE Transactions on Knowledge and Data Engineering*, 34(8):3549–3568, 2022. 57

[HBC+21] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. *Knowledge Graphs*. Number 22 in Synthesis Lectures on Data, Semantics, and Knowledge. Springer, 2021. 3, 4, 5, 21, 22, 30, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 50, 51, 147

[HEBR13] Maryam Hazman, Samhaa El-Beltagy, and Ahmed Rafea. A survey of ontology learning approaches. *International Journal of Computer Applications*, 22:36–43, 09 2013. 50

[Hed22] Heather Hedden. *The Accidental Taxonomist, 3rd edition*. 2022. 13, 39, 40

[HIPSC14] Patrick J. Hayes, Florida IHMC, Peter F. Patel-Schneider, and Nuance Communications. RDF 1.1 Semantics. W3c recommendation, w3c, february 2014. 42

[HKP+12] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer (Second Edition). W3c recommendation, December 11 2012. 5, 43

[HLC+11] Shang-Hsien Hsieh, Hsien-Tang Lin, Nai-Wen Chi, Kuang-Wu Chou, and Ken-Yu Lin. Enabling the development of base domain ontology through extraction of knowledge from engineering domain handbooks. *Advanced Engineering Informatics*, 25(2):288–296, 2011. Information mining and retrieval in design. 68

[HLS15] Gyeong June Hahm, Jae Hyun Lee, and Hyo Won Suh. Semantic relation based personalized ranking approach for engineering document retrieval. *Advanced Engineering Informatics*, 29(3):366–379, 2015. 68, 70

[HMP+24] Ralph Hodgson, Daniel Mekonnen, David Price, Jack Hodges, James E. Masters, Simon J D Cox, and Steve Ray. Quantities, Units, Dimensions and Types (QUDT) Schema - Version 2.1.35. Technical report, QUDT.org, Janvier 2024. 85

[HP23] Kailash A. Hambarde and Hugo Proença. Information retrieval: Recent advances and beyond. *IEEE Access*, 11:76581–76604, 2023. 6, 54, 58, 61

[HPS12] Matthew Horridge and Peter F. Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax. W3c working group note, w3c, december 2012. 109

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3c member submission, May 21 2004. 46

[HPSv03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From shiq and rdf to owl: the making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003. 40

[HSP13] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 Query Language. W3c recommendation, w3c, march 2013. 42, 56

[HYLS14] Gyeong June Hahm, Mun Yong Yi, Jae Hyun Lee, and Hyo Won Suh. A personalized query expansion approach for engineering document retrieval. *Advanced Engineering Informatics*, 28(4):344–359, 2014. 68, 70

[IS09] Antoine Isaac and Ed Summers. SKOS Simple Knowledge Organization System Primer. W3c working group note, w3c, August 2009. 84, 94

[JM23] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3rd edition, 2023. 7, 60, 61, 62

[Jon21] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *J. Documentation*, 60:493–502, 2021. 99

[JT10] Xing Jiang and Ah-Hwee Tan. Crctol: A semantic-based domain ontology learning system. *J. Assoc. Inf. Sci. Technol.*, 61:150–168, 2010. 49, 50

[KAG21] Ahlem Chérifa Khadir, Hassina Aliane, and Ahmed Guessoum. Ontology learning: Grand tour and challenges. *Computer Science Review*, 39:100339, 2021. 47, 50, 91

[KDT+21] Ahmed Khemiri, Amani Drissi, Anis Tissaoui, Salma Ben Sassi, and Richard Chbeir. Learn2construct: An automatic ontology construction based on lda from texual data. *Proceedings of the 13th International Conference on Management of Digital EcoSystems*, 2021. 49, 50

[Kee20] C. Maria Keet. *An Introduction to Ontology Engineering*. https://people.cs.uct.ac.za/mkeet/OEbook, 2020. 5, 22, 33, 39, 40, 45, 47

[KK17] Holger Knublauch and Dimitris Kontokostas. Shapes Constraint Language (SHACL). W3c recommendation, June 20 2017. 9, 51, 85, 152

[KKW+16] Saiful Khan, Urszula Kanturska, Tom Waters, James Eaton, René Bañares-Alcántara, and Min Chen. Ontology-assisted provenance visualization for supporting enterprise search of engineering and business files. *Advanced Engineering Informatics*, 30(2):244–257, 2016. 68, 70

[KR18] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In Eduardo Blanco and Wei Lu, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. 63

[LCH12] Hsien-Tang Lin, Nai-Wen Chi, and Shang-Hsien Hsieh. A concept-based information retrieval approach for engineering domain-specific technical documents. *Advanced Engineering Informatics*, 26(2):349–360, 2012. Knowledge based engineering to support complex product design. 68, 70

[LF15] Xitong Liu and Hui Fang. Latent entity space: a novel retrieval approach for entity-bearing queries. *Information Retrieval Journal*, 18:473–503, 2015. 64, 66

[LIJ+15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, S. Auer, and Christian Bizer. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6:167–195, 2015. 34, 67, 83

[LJLH19] Yang Liu, Eunice Jun, Qisheng Li, and Jeffrey Heer. Latent space cartography: Visual analysis of vector space embeddings. *Computer Graphics Forum*, 38(3):67–78, 2019. 7, 58, 59

[LLL+20] Nanxing Li, Qian Li, Yu-Shen Liu, Wenlong Lu, and Wanqi Wang. Bimseek++: Retrieving bim components using similarity measurement of attributes. *Computers in Industry*, 116:103186, 2020. 69, 70

[LM13] Paul Groth Luc Moreau. *Provenance*. Morgan & Claypool Publishers, September 2013. 85

[LPMG19] Katia Lupinetti, Jean-Philippe Pernot, Marina Monti, and Franca Giannini. Content-based cad assembly model retrieval: Survey and future challenges. *Computer-Aided Design*, 113:62–81, 2019. 13, 22, 69, 130

[LPP+20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020. 16, 151

[LRR08] Zhanjun Li, Victor Raskin, and Karthik Ramani. Developing Engineering Ontology for Information Retrieval. *Journal of Computing and Information Science in Engineering*, 8(1):011003, 02 2008. 13, 17, 22, 70, 71, 130, 143, 152

[LSM+13] Timothy Lebo, Satya Sahoo, Deborah McGuinness, Khalid v, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. PROV-O: The PROV Ontology. W3c recommendation, w3c, April 2013. 9, 85

[LXSL18] Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. Entity-duet neural ranking: Understanding the role of knowledge graph semantics in neural information retrieval. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2395–2405, Melbourne, Australia, july 2018. Association for Computational Linguistics. 65, 66

[MC07] Donald Metzler and W. Bruce Croft. Latent concept expansion using markov random fields. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 311–318. Association for Computing Machinery, 2007. 58

[Mcc18] Dave Mccomb. *Software Wasteland: How the Application-Centric Mindset is Hobbling our Enterprises*. TECHNICS PUBN LLC, February 2018. 81

[MF97] Natalia Juristo Mariano Ferndndez, Asuncion Gomez-Perez. Methontology: From ontological art towards ontological engineering. *AAAI Technical Report SS-97-06*, 1997. 79

[MFP09] Diana Maynard, Adam Funk, and Wim Peters. Sprat: a tool for automatic semantic pattern-based ontology population. 2009. 49, 50

[MGH+12] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles (Second Edition). W3c recommendation, December 11 2012. 46

[Mil95] George A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, nov 1995. 49, 91, 94, 96, 99

[MPSG+12] Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Uli Sattler. OWL 2 Web Ontology Language Direct Semantics (Second Edition). W3c recommendation, December 11 2012. 45

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. 3, 21, 54, 56, 60, 61, 62

[MS18] Kamran Munir and M. Sheraz Anjum. The use of ontologies for effective knowledge modelling and information retrieval. *Applied Computing and Informatics*, 14(2):116–126, 2018. 67, 70

[Mur10] Gregory L. Murphy. 11What are categories and concepts? In *The Making of Human Concepts*. Oxford University Press, 01 2010. 107

[NGJ+19] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs: Lessons and challenges. *Commun. ACM*, 62(8):36–43, jul 2019. 30, 43

[NHAC+22] Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1864–1874, Dublin, Ireland, May 2022. Association for Computational Linguistics. 99

[NVG03] Roberto Navigli, Paola Velardi, and Aldo Gangemi. Ontology learning and its application to automated terminology translation. *IEEE Intell. Syst.*, 18:22–31, 2003. 49, 50

[Pat12] D. J. Patil. *Data jujitsu : the art of turning data into product.* O'Reilly, Sebastopol, CA, 2012. 85

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking : Bringing order to the web. In *The Web Conference*, 1999. 55, 65

[PDGB09] Valentina Presutti, Enrico Daga, Aldo Gangemi, and Eva Blomqvist. Extreme design with content ontology design patterns. In *Proceedings of the 2009 International Conference on Ontology Patterns - Volume 516*, WOP'09, page 83–97, Aachen, DEU, 2009. CEUR-WS.org. 109

[PSMG+12] Peter F. Patel-Schneider, Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, Alan Ruttenberg, and Michael Schneider. OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition). W3c recommendation, December 11 2012. 46

[PVCFCPGC23] María Poveda-Villalón, Serge Chávez-Feria, Sergio Carulli-Pérez, and Raúl García-Castro. Towards a uml-based notation for owl ontologies. In *Proceedings of the 8th International Workshop on the Visualization and Interaction for Ontologies, Linked Data and Knowledge Graphs*, pages 18–27, Athens, Greece, november 2023. CEUR Workshop Proceedings. 34

[PVGPSF14] María Poveda-Villalón, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. Oops! (ontology pitfall scanner!): An on-line tool for ontology evaluation. *Int. J. Semant. Web Inf. Syst.*, 10(2):7–34, apr 2014. 151

[QGY+16] Feiwei Qin, Shuming Gao, Xiaoling Yang, Ming Li, and Jing Bai. An ontology-based semantic retrieval approach for heterogeneous 3d cad models. *Advanced Engineering Informatics*, 30(4):751–768, 2016. 13, 22, 69, 70, 71, 130

[RKC16] Hadas Raviv, Oren Kurland, and David Carmel. Document retrieval using entity-based language models. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 65–74, New York, NY, USA, 2016. Association for Computing Machinery. 64, 66

[RMdR20] Ridho Reinanda, Edgar Meij, and Maarten de Rijke. Knowledge graphs: An information retrieval perspective. *Foundations and Trends® in Information Retrieval*, 14(4):289–444, 2020. 30, 56, 57, 63, 64, 65, 66, 71

[Row07] Jennifer Rowley. The wisdom hierarchy: Representations of the dikw hierarchy. *J. Inf. Sci.*, 33(2):163–180, apr 2007. 13, 35

[RWNW05] Alan Recto, Chris Welty, Natasha Noy, and Evan Wallace. Simple part-whole relations in OWL Ontologies. W3c editor's draft, w3c, august 2005. 110

[RZ09] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009. 7, 59, 60

[SAB+23] Fabian Suchanek, Mehwish Alam, Thomas Bonald, Pierre-Henri Paris, and Jules Soria. Integrating the wikidata taxonomy into yago. 2023. 34, 67

[SBF98] Rudi Studer, V.Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1):161–197, 1998. 47

[SC12] Mark Sanderson and W. Bruce Croft. The history of information retrieval research. *Proceedings of the IEEE*, 100(Special Centennial Issue):1444–1451, 2012. 6, 54

[Sch73] E. W. Schneider. Course Modularization Applied: The Interface System and Its Implications For Sequence Control and Data Analysis. Technical Report HumRRO-PP-10-73, 300, North Washington Street Alexandria, Virginia 22314, 1973. Paper presented at the Association for the Development of Instructional Systems (Chicago, Illinois, April 1972). 21, 30

[SCH16] Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. *arXiv.org*, 2016. 34, 49, 91, 94

[SFW83] Gerard Salton, Edward A. Fox, and Harry Wu. Extended boolean information retrieval. *Commun. ACM*, 26(11):1022–1036, nov 1983. 59, 60

[SKA+22] Umutcan Simsek, Elias Kärle, Kevin Angele, Elwin Huaman, Juliette Opdenplatz, Dennis Sommer, Jürgen Umbrich, and Dieter Fensel. A knowledge graph perspective on knowledge engineering. *SN Computer Science*, 4(1):16, Oct 2022. 46, 47

[SL21] J. Sequeda and O. Lassila. *Designing and Building Enterprise Knowledge Graphs*. Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool Publishers, 2021. 32, 33, 34, 83

[SLKF18] Martin G. Skjæveland, Daniel P. Lupp, Leif Harald Karlsen, and Henrik Forssell. Practical ontology pattern instantiation, discovery, and maintenance with reasonable ontology templates. In *Lecture Notes in Computer Science*, pages 477–494. Springer International Publishing, 2018. 9, 85

[Sow99] John F. Sowa. *Knowledge representation: logical, philosophical and computational foundations*. Brooks/Cole Publishing Co., USA, 1999. 47

[SSS+20] Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. Linear-time wordpiece tokenization. 2020. 63

[STS11] Rossitza Setchi, Qiao Tang, and Ivan Stankov. Semantic-based information retrieval in support of concept design. *Advanced Engineering Informatics*, 25(2):131–146, 2011. Information mining and retrieval in design. 68, 70

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, nov 1975. 55, 58, 59, 60

[SXH+18] Jiaming Shen, Jinfeng Xiao, Xinwei He, Jingbo Shang, Saurabh Sinha, and Jiawei Han. Entity set search of scientific literature: An unsupervised ranking approach. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, pages 565–574, New York, NY, USA, 2018. Association for Computing Machinery. 65, 66

[UDG18] Michael Uschold, Ying Ding, and Paul Groth. *Demystifying Owl for the Enterprise*. Morgan & Claypool Publishers, 2018. 43

[Val21] Daniel Valcarce. Information retrieval models for recommender systems. *SIGIR Forum*, 53(1):44–45, mar 2021. 56, 57, 58

[VHH08] Johanna Völker, Peter Haase, and Pascal Hitzler. Learning expressive ontologies. In *Ontology Learning and Population*, 2008. 49

[VK14] Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, September 2014. 34, 38, 49, 67, 83, 94

[VMUT17] Sree Harissh Venu, Vignesh Mohan, Kodaikkaavirinaadan Urkalan, and Geetha Tv. Unsupervised domain ontology learning from text. In *Mining Intelligence and Knowledge Exploration*, pages 132–143, 04 2017. 49, 50

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017. 56

[Wat20] Jarosław Watróbski. Ontology learning methods from text - an extensive knowledge-based approach. *Procedia Computer Science*, 176:3356–3368, 2020. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES2020. 50

[XC15a] Chenyan Xiong and Jamie Callan. Esdrank: Connecting query and documents through external semi-structured data. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 951–960, New York, NY, USA, 2015. Association for Computing Machinery. 64, 66

[XC15b] Chenyan Xiong and Jamie Callan. Query expansion with freebase. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, ICTIR '15, pages 111–120, New York, NY, USA, 2015. Association for Computing Machinery. 64, 66, 67

[XCL17] Chenyan Xiong, Jamie Callan, and Tie-Yan Liu. Word-entity duet representations for document ranking. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 763–772, New York, NY, USA, 2017. Association for Computing Machinery. 65, 66

[XPC17] Chenyan Xiong, Russell Power, and Jamie Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1271–1279, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. 64, 66

[YZCC22] Hongbin Ye, Ningyu Zhang, Hui Chen, and Huajun Chen. Generative knowledge graph construction: A review. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1–17, Abu Dhabi, United Arab Emirates, dec 2022. Association for Computational Linguistics. 47

[ZFR16] Lei Zhang, Michael Färber, and Achim Rettinger. Xknowsearch! exploiting knowledge bases for entity-based cross-lingual information retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 2425–2428, New York, NY, USA, 2016. Association for Computing Machinery. 65, 66

[ZGH11] Amal Zouaq, Dragan Gasevic, and Marek Hatala. Towards open ontology learning and filtering. *Inf. Syst.*, 36(7):1064–1081, nov 2011. 49, 50, 91

[ZMRA13] Marie Lisandra Zepeda-Mendoza and Osbaldo Resendis-Antonio. *Hierarchical Agglomerative Clustering*, pages 886–887. Springer New York, New York, NY, 2013. 99

[ZRZ16] Lei Zhang, Achim Rettinger, and Ji Zhang. A knowledge base approach to cross-lingual keyword query interpretation. In *The Semantic Web – ISWC 2016*, pages 615–631. Springer International Publishing, 2016. 65

[ZWL+23] Lingfeng Zhong, Jia Wu, Qian Li, Hao Peng, and Xindong Wu. A comprehensive survey on automatic knowledge graph construction. *ACM Comput. Surv.*, 56(4), nov 2023. 47