Höchstleistungsrechenzentrum
Universität Stuttgart
Prof. Dr.-Ing. Dr. h.c. Dr. h.c. M. Resch
Nobelstraße 19
70569 Stuttgart
Institut für Höchstleistungsrechnen

Ontology Matching in a Distributed Environment

Von der Fakultät Energie-, Verfahrens- und
Biotechnik der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr. - Ing.)
genehmigte Abhandlung

vorgelegt von

Axel Tenschert

aus Herford

Hauptberichter:	Prof. Dr.-Ing. Dr. h.c. Dr. h.c. Michael Resch
Mitberichter:	Prof. Dr.-Ing. Stefan Wesner
Tag der Einreichung:	20. Januar 2016
Tag der mündlichen Prüfung:	16. Dezember 2016

Höchstleistungsrechenzentrum Stuttgart
Universität Stuttgart
Dezember 2016

Danksagung

Der erfolgreiche Abschluss dieser Arbeit wurde ermöglicht durch viele Menschen, die mich immer wieder unterstützten. All Jenen möchte ich an dieser Stelle meinen Dank aussprechen.

Besonderer Dank gilt Professor Michael Resch, meinem Doktorvater, der mich zu meiner Promotion ermutigte und diese ermöglichte. Des Weiteren möchte ich mich bei meinem Mitberichterstatter Professor Stefan Wesner für die dauerhafte Unterstützung sowie die konstruktiven Gespräche bedanken, die mir stets eine große Hilfe waren und mich motivierten meine Arbeit weiter voran zu bringen.

Meinen Arbeitskollegen danke ich für die angenehme Zusammenarbeit. Hierbei möchte ich meinen Dank Lutz Schubert aussprechen, der mir besonders zu Beginn meiner Promotion durch kritische Fragen die Möglichkeit gab, meine Themen ausführlich zu diskutieren, so zum Beispiel im PhD Seminar. Weiterhin bedanke ich mich bei Bastian Koller und Roland Kübert für die stetige Unterstützung während der Zeit meiner Promotion.

Außerdem bedanke ich mich bei meinen Freunden für ihren steten Rückhalt. Sina Rimpf danke ich für das Korrekturlesen. Darüber hinaus gilt mein besonderer Dank meiner Familie, meinen Eltern, die auf meinem gesamten Lebensweg und insbesondere während meiner Promotion immer zu mir standen und Rückhalt gaben.

Stuttgart, im Dezember 2016 Axel Tenschert

Abstract

Nowadays, society, research and industries are faced with a continuously growing amount of information available as structured or unstructured data presenting semantic meanings. In the scope of this work, data structures based on semantic specifications are investigated concluding in an approach for receiving customized semantic data related to a certain domain of interest. As ontologies, originally a philosophy discipline for identifying the fundamental structure of reality, are similar to interlinked semantic data collections extended with rules. They are used for storing, updating and utilizing semantic data. Such ontologies have already reached the barrier of billions of interlinked concepts, causing difficulties in terms of performance when trying to execute queries. Computer sciences define an ontology as a conceptualization of formal representations including interconnected concepts related to a certain domain. Through the matching of several ontologies relevant semantic data containing information are identified or the validity of data structures is proven by matching it to each other. Current ontology matching approaches are based on various matching strategies being the base for several tools and frameworks. Hence, this work is based on such strategies as well as latest ontology matching tools and frameworks.
This work investigated an enhanced ontology matching approach dealing with the continuously growing amount of semantic data for generating facts related to a certain domain of interest. Thus, the processing of semantic data face the challenge of allocating sufficient computing resources as well as for performing a content wise sensible matching of needed data. Current approaches struggle with performance and efficiency issues as soon as the semantic data reaches a high amount of data, e.g. in August 2011 in the scope of the large triple store challenge [1], the loading and querying of about one trillion Resource Description Framework (RDF) triples [2] was processed. The execution took nearly 338 hours by an average rate of 829.556 RDF triples per second running on 80 cores. In addition, the generation of precise matching results needs to be considered. Current risks in the scope of matching semantic data are the generation of false matching results due to an insufficient matching, not accurate enough to receive high quality results, and an afterwards performed reasoning based on the matching results not suitable when thinking of big interlinked data volumes. Additionally, a reasoning based on a corrupt data set generated by an insufficient matching will produce wrong facts. Thus, the main question of this work is how to bridge the gap between current matching approaches and large semantic data stores by applying a semantic matching usable for reasoning on those big data stores. Such a proceeding will manage the continuously growing immense amount of data through a matching strategy and thus, providing an individually customized

knowledge base containing domain specific information relevant for a given use case scenario.
This work makes use of a semantic matching approach usable for ontologies by consideration of similarity values combined with known graph based approaches in a distributed environment. Hence, an ontology matching approach is developed performing a matching by three iterations by executing a lexical, a taxonomy and a non-taxonomy matching.

- Lexical Matching. This is the first matching iteration. It compares the lexical entities of the ontologies.

- Taxonomy Matching. The second matching iteration compares the taxonomies of the ontologies being graphs. Hence, graph paths are compared.

- Non-Taxonomy Matching. The last matching iteration analyses the linking elements between the entities.

For each matching a similarity value is generated, expressing the degree of similarity between ontological structures containing RDF triples. The usage of similarity values together with a graph based approach increases the accuracy of the semantic matching strategy. The calculation of the similarity values is divided into several phases:

- Entity Engineering. Relevant entities of the ontology set are identified.

- Search Step Selection. A defined search space of matches in the graph is derived.

- Similarity Computation. Similarity values for the matches are defined. The three matching iterations (lexical, taxonomy and non-taxonomy) are applied.

- Similarity Aggregation. Several similarity values of a single match (e.g. similarity between properties, relations, etc.) are aggregated to one.

- Interpretation. In the final interpretation step the aggregated similarities are used to derive matches between entities.

The similarity values are generated while the ontology matching is executed. Each matching step generates alignment lists being the result of the matching usable for further reasoning tasks. However, the matching of large data sets leads to a performance problem solved through aligning the developed ontology matching algorithms to a High Performance Computing (HPC) environment. The presented ontology matching approach is performed in a distributed shared memory system on several nodes for increasing the performance. The used HPC system is the NEC NEHALEM cluster hosted at HLRS. The outcome is an ontology matching considering distribution techniques in order to increase the overall performance for matching large data sets and providing results usable for receiving customized data related to a certain domain of

interest. The ontology matching is performed by defining a Priority Ontology (PO) as a source and match it with other ontologies related to the given domain of interest. The result produced by the matching iterations is a single customized ontology providing information of a certain domain for a human being.
In the scope of this work, a prototypical implementation of the matching strategy is developed and presented. Furthermore, the execution of the semantic matching deals with large data stores by performing the matching on allocated HPC resources. The allocation of the HPC resources refers to a Service Level Agreement (SLA) based job scheduling using queues. The presented scheduling approach offers the possibility to schedule computing jobs using defined SLA levels such as gold, silver and bronze. The SLAs are presenting a Quality of Service (QoS) class describing the required resources of an HPC cluster, for instance, the type and amount of nodes as well as the time frame.
The presented work deals with current scalability issues when thinking of matching large scale ontologies. For this, the distributed matching approach makes use of HPC resources. Therefore, this work enhances the efficiency for semantic matching and bridges the gap between current ontology matching scalability issues and the HPC domain. Through the demonstration of an ontology matching approach and a prototypical implementation the foundation for a reasoning task using HPC resources becomes possible.

Zusammenfassung

Das stetig zunehmende Aufkommen von Daten in strukturierter und unstrukturierter Form, stellt Gesellschaft, Forschung und Industrie vor das Problem der angemessenen Datenselektion. Das Einbinden semantischer Relationen unterstützt eine zielgenaue Auswahl aus bekannten Datenbeständen. Daher werden in dieser Arbeit Datenstrukturen basierend auf semantischen Relationen untersucht, um eine effiziente und kontextberücksichtigende Datenselektion zu ermöglichen. Dieser Ansatz sieht die Bereitstellung individueller und domänenspezifischer semantischer Daten vor.
Weiterhin verfolgt der vorgestellte Ansatz die Einbindung von Ontologien, um semantische Daten abzubilden und bearbeiten zu können. Ontologien entstammen den philosophischen Disziplinen und identifizieren und präsentieren grundlegende Strukturen der Wirklichkeit. Sie sind regelbasiert und stehen in Relation zueinander. Die Beschaffenheit von Ontologien ermöglicht die Persistenz, Überarbeitung und Nutzung semantischer Daten. Ein besonderes Augenmerk gilt den Relationen zwischen Ontologien und den darin enthaltenen Entitäten, da allein die Anzahl an Relationen die Milliardengrenze überschritten hat. In Hinblick auf die Datenselektion stellt dies eine enorme Herausforderung für die Performanz dar.
Allgemein definieren Computerwissenschaften Ontologien als Konzeptualisierungen formaler Darstellungen miteinander in Relation stehender domänenspezifischer Konzepte. Dies impliziert, dass mittels der Zusammenführung von mehreren Ontologien relevante semantische Daten identifiziert und die Gültigkeit dieser Datenstrukturen verifiziert werden kann. Gegenwärtige Vergleichsstrategien für Ontologien basieren auf verschiedensten Ansätzen. Daher werden in dieser Arbeit unterschiedlichste Ansätze, Tools und Frameworks zum Vergleich von Ontologien berücksichtigt.
Die in dieser Arbeit entwickelte Strategie zum Ontologievergleich ermöglicht einen verbesserten Ansatz, um stetig wachsende Mengen semantischer Daten individuell miteinander vergleichen zu können. Vor diesem Hintergrund wird die Verarbeitung semantischer Daten untersucht und implementiert. Aufgrund der hohen Anzahl an Relationen zwischen Daten und den darin enthaltenen Entitäten werden ausreichend Rechenressourcen genutzt, wie sie in einem verteilten Computersystem bereitstehen. In dieser Arbeit wird von kontinuierlich wachsenden Datenbeständen ausgegangen, dabei steht die Performanz der verschiedenen Strategien im besonderen Fokus, z.B. wurde im August 2011 im Rahmen der Large Triple Store Challenge [1], das Laden und Abfragen von über einer Billion RDF Triples [2] realisiert. Die Durchführungszeit betrug nahezu 338 Stunden bei einer durchschnittlichen Rate von 829,556 RDF Tripel pro Sekunde, verteilt auf 80 Cores.
Darüber hinaus ist die Erzeugung von genauen Ergebnissen notwendig. Aktuelle

Risiken für Vergleichsstrategien semantischer Daten sind die Erzeugung falscher oder unzureichender Ergebnisse aufgrund zu ungenauer Vergleiche von Ontologien. Zusätzlich ist ein auf diesen Ergebnissen ausgeführtes Schlussfolgern für große miteinander in Relation stehenden Daten nicht geeignet, da durch die Verwendung von unzureichenden oder falschen Ontologien als Datenbasis, falsche Schlussfolgerungen generiert werden. Deswegen ist die zugrundeliegende wissenschaftliche Hauptfrage dieser Arbeit, wie die Hürde zwischen Ansätzen für Ontologievergleiche und sehr großen semantische Datenmengen überwunden und umgesetzt werden kann. Hierbei ist die Berücksichtigung dauerhaft wachsender Datenmengen für Vergleichsstrategien von Ontolgien essentiell. Das Ziel ist es, aus den genannten Daten eine individuell zugeschnittene und domänenabhängige Wissensbasis zu generieren, die für spezifische Anwendungsszenarien zuverlässig verwendbar ist.
Im Rahmen dieser Arbeit wird ein Ansatz zum Vergleich von Ontologien entwickelt und implementiert, der semantische Daten und die zugehörigen Relationen berücksichtigt. Dabei werden Übereinstimmungswerte zwischen Entitäten und Ansätze zur Graphenanalyse berücksichtigt. Um dem Aspekt der dauerhaft wachsenden Datenmengen und der Performanz gerecht zu werden, wird der Ansatz dieser Arbeit für eine verteilte Umgebung entwickelt. Die Qualität der Ergebnisse des Ontologievergleichs wird durch einen dreistufigen Ansatz sichergesellt. Dieser beinhaltet ein lexikalisches, ein taxonomisches und ein nicht-taxonomisches Vergleichsverfahren.

- Lexikalischer Vergleich. In dieser ersten Verlgeichsiteration werden die lexikalischen Entitäten der Ontologien miteinander verglichen.

- Taxonomischer Vergleich. Dies ist die zweite Vergleichsiteration, die Ontologien als Graphen betrachtet und die Pfade der Graphen miteinander vergleicht.

- Nicht-Taxonomischer Vergleich. In der dritten und letzten Vergleichsiteration werden die Relationen der Entitäten der verschiedenen Ontologien analysiert.

Für jedes Matching wird ein Übereinstimmungswert generiert, der den Grad der Ähnlichkeit zwischen den ontologischen Strukturen darstellt. Die Verwendung von Übereinstimmungswerten in Kombination mit Graphen basierten Ansätzen, erhöht die Qualtität der Vergleichsergebnisse durch ein höheres Maß an Genauigkeit. Die Berechnung der Übereinstimmungswerte ist in mehrere Schritte untergliedert:

- Indentifikation der Entitäten. Die relevanten Entitäten der Ontolgien werden identifiziert.

- Festlegung des Suchraums. Der Suchraum für die Vergleichsoperationen im Graph wird definiert.

- Berechnung der Übereinstimmungen. Es werden Ähnlichkeitswerte für die Vergleichsoperationen festgelegt. Die drei Iterationsstufen (lexikalisch, taxonomisch und nicht-taxonomisch) werden ausgeführt.

- Aggregation der Übereinstimmungen. Die berechneten Übereinstimmungswerte der Vergleiche (z.B. Übereinstimmungen zwischen Eigenschaften, Relationen, etc.) werden aggregiert.

- Interpretation. Die Interpretation ist der finale Schritt. Die aggregierten Übereinstimmungen werden verwendet, um Ähnlichkeiten zwischen den Entitäten abzuleiten.

Durch den oben vorgestellten Vergleich der Ontologien, werden Übereinstimmungswerte generiert. Dabei werden bei jeder Vergleichsoperation Abgleichslisten erzeugt, welche als Basis für später durchgeführte Schlussfolgerungsprozesse genutzt werden. Die hohe Anzahl an benötigten Vergleichsoperationen, durchgeführt auf ständig wachsenden Datenmengen, führt jedoch zu Performanzproblemen. Aus diesem Grund berücksichtigt der Ansatz dieser Arbeit eine verteilte Umgebung, wie sie im HPC Umfeld zu finden ist. Hierbei wird auf ein Shared Memory System zurückgegriffen, welches den Vorteil bietet, dass die Vergleichsoperationen auf mehreren Knoten aufgeteilt werden können, mit dem Ziel die Performanz zu verbessern. Das hierbei verwendete HPC System ist das HLRS NEC NEHALEM Cluster. Die Verteilung der Vergleichsoperationen setzt einen Ontologievergleich voraus, der es zulässt Vergleichsmethoden verteilt auszuführen. Der in dieser Arbeit entwickelte ontologische Vergleichsansatz nutzt eine Ontologie als Quelle (Prioritätsontologie) und vergleicht diese mit anderen. Das Ergebnis ist eine einzelne spezifizierte Ontologie, die individuelle Informationen eines Interessenbereichs Menschen zugänglich macht.
Im Rahmen dieser Arbeit wird eine prototypische Implementierung der benötigten Vergleichsstrategie entwickelt. Weiterhin wird die Ausführung des semantischen Vergleichs für sehr große Datenmengen, mittels hierfür allokierter HPC Ressourcen, durchgeführt. Hierbei wird die Reservierung der HPC Ressourcen durch ein SLA basiertes Job Scheduling unter Verwendung von Queues umgesetzt. Der vorgestellte Scheduling Ansatz ermöglicht ein Job Scheduling basierend auf definierten SLA Levels, wie etwa Gold, Silber und Bronze Leveln. Die SLAs repräsentieren QoS Klassen, die die benötigten Ressourcen des HPC Clusters beschreiben, z.B. Typ und Anzahl der Knoten sowie den Zeitrahmen.
Die vorgestellte Arbeit nimmt Bezug auf Performanz und Skalierungsprobleme, die speziell im Bereich der semantischen Vergleiche in Zusammenhang mit großen Datenmengen und Ontologien auftreten. Daher werden HPC Ressourcen verwendet, um hierdurch eine effiziente Implementierung der Vergleichsstrategie für Ontologien zu ermöglichen und somit die Kluft zwischen aktuellen ontologiespezifischen Skalierbarkeitsproblemen und dem HPC Bereich zu überwinden. Durch den in dieser Arbeit vorgestellten und prototypisch implementierten Ansatz für das Vergleichen von Ontologien, wird das Fundament für Reasoning Aufgaben im HPC Bereich geschaffen.

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface . . . 56
AS	Accumulation of Similarities . . . 83
DB	Database . . . 71
DDL	Distributed Description Logic . . . 50
DO	Domain Ontology . . . 61
EC	Entity Count . . . 87
GB	Gigabyte . . . 9
GMM	Generalized Method of Moments . . . 42
GMO	Graph Matcher for Ontologies . . . 55
GUI	Graphical User Interface . . . 45
HPC	High Performance Computing . . . iv
HPC-WSAG	High Performance Computing - Web Service Agreement . . . 60
HR	High Relevance Ontology . . . 71
ID	Identifier . . . 72
IR	Information Retrieval . . . 35
IT	Information Technology . . . 58
JVM	Java Virtual Machine . . . 128
kB	Kilobyte . . . 3
LarKC	The Large Knowledge Collider . . . 57
LLD	Linked Life Data . . . 31
LLOD	Linguistic Linked Open Data . . . 62
LOD	Linked Open Data . . . 31
LR	Low Relevance Ontology . . . 71
LSA	Latent Semantic Analysis . . . 38
LSV	Lexical Similarity Value . . . 79
MB	Megabyte . . . 3
MLN	Markov Logic Network . . . 46
MO	Model Ontology . . . 61
MR	Medium Relevance Ontology . . . 71
NF	Number of Features . . . 83
NGD	Normalize Google Distance . . . 47
NGMF	Next Generation Modeling Framework . . . 58
NTSV	Non-Taxonomy Similarity Value . . . 79
OntoRepDB	Ontology Repository Database . . . 72
OPS	Online Proposal Submission . . . 59
OWL	Web Ontology Language . . . 6

OWL DL	Web Ontology Language Description Logic 11
OWL Full	Web Ontology Language Full 11
OWL Lite	Web Ontology Language Lite 11
OWL 2	OWL 2 Web Ontology Language 11
PBM	Partition Based Block Matcher 55
PC	Path Count 86
plugIT	Plug Your Business Into IT 58
PO	Priority Ontology v
PRP	Probabilistic Ranking Principle 36
QoS	Quality of Service v
RDF	Resource Description Framework iii
RDFS	Resource Description Language Schema 6
SD	Search Depth 86
SLA	Service Level Agreement v
SO	Source Ontology 109
SPARQL	SPARQL Protocol and RDF Query Language 6
TL	Term Length 109
TO	Target Ontology 109
TSV	Taxonomy Similarity Value 79
URI	Uniform Resource Identifier 12
VCD	Validity Check Document 93
WSAG	Web Service Agreement 60
WSMO	Web Service Modeling Ontology 11
WWW	World Wide Web 2
W3C	World Wide Web Consortium 81
XML	Extensible Markup Language 3

Chapter 1

Introduction

1.1 Objectives

Towards the continuously growing amount of data accessible in the world wide web and involving linked data as well as semantic data from a broad set of different media types, the data management has become a key challenge for society, research and industry in the last decades. The importance of handling those data is needed for identifying relevant information from those data sets, making them available and understandable to human beings and machines. The term semantics describes data presenting a certain meaning usable in a computer processable manner. This indicates the need of structuring knowledge by use of formal representations. Within the semantic web activity group [3] ontologies are used to store and manage semantic information as well as relations between the information considering the use of meta data for specifying the presented information more precise.
Ontologies have been introduced by Aristotle ([4], [5]) as formal models describing our awareness of a domain of interest and in addition providing a precise, logical account of certain meanings of terms by structuring entities as real world models. Nowadays, ontologies are used in research and enterprise communities in order to formalize concepts and structure semantic data. Ontologies define the basic terms and relations comprising the vocabulary of a certain individual domain in a formal manner, enabling machines to communicate, understand, analyze and process the ontologically formalized concepts. The information can be used by machines or human beings using the provided terminology of an ontology. Through the use of ontologies as knowledge bases for semantic data, it becomes possible to find semantic correspondences between different information in various ontologies. Su et al. [6] are describing the use of ontologies in the area of the semantic web as a key issue for enabling a semi-automatic matching and reasoning of semantic information:

> In order to understand and process the information in a semantic meaningful way, researchers have created the Semantic Web vision, where data has structure and ontologies describe the semantics of the data. This en-

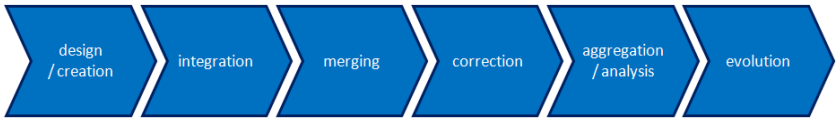

Figure 1.1: Ontology Lifecycle

ables the usage of semi-/automatic reasoning within the data structures such as for software agents or reasoning procedures using ontologies.

The term *Semantic Web vision* was used by Tim Berners-Lee [7] in the context of meaning and understanding for machines in the World Wide Web (WWW). In this context, ontologies are usable for defining a formal representation of real world concepts by defining a terminology of a domain of interest. The ontology life cycle consists of various stages (see figure 1.1) including the *design and creation* of ontologies, the *integration of knowledge data and schema* from various sources to be included in the ontology, the *merging* of ontologies, the *correction of erroneous data* in the ontology, the *aggregation and analysis* of the existing data in the ontology and the *evolution* of ontologies.

The objectives of this work are to provide a scalable, robust and efficient matching strategy for ontologies in order to provide a human being with a tailor-made ontology. Through a matching of several ontologies relevant data are identified being used as an individual knowledge base. This proceeding is an important step for providing necessary data instead of a broad data set. The ontology matching is performed in a HPC environment to deal with scalability issues and time restrictions. Such ontologies contain needed semantic information extracted by reasoning tasks. Moreover, through the use of a distributed architecture the scalability of a useful matching strategy is increased which is an advantage when considering large data sets and, even more important, complex and computational expensive matching processes. Further, a reasoning performed on an ontology needs to deal with time restrictions, large data sets, reliability and so forth for enabling a fast querying on an adequate knowledge base as presented in the next chapter (see 2.2, 2.3 and 2.4). The matching of ontologies includes the *integration of knowledge data and schema*, the *merging* aspect and the *correction of erroneous data* after the merging was performed. These steps are considered to provide an efficient approach for the aggregation and analysis of the ontology data used for providing a specialized knowledge base related to an individual use case for enabling reasoning tasks based on a matched result set.
When thinking of ontologies, especially using them for providing a customized knowledge base, tools and methods from the semantic web need to be considered. The usage of ontologies for receiving relevant knowledge being related to a specific domain of interest and individual tasks is a quite promising approach for making information available through reasoning performed based on the underlying ontologies. Ramesh and Gnanasekaran [8] are describing a single ontology as not sufficient anymore for receiving semantic representations for various context domains:

> A single ontology is no longer enough to support the tasks predicted by a distributed environment like the Semantic Web. Multiple ontologies need to be accessed from several applications. Ontology management is possible through interoperability of semantic data sources.

Through an improved matching strategy, by means of increasing the scalability and overall performance, which is provided with the required computing resources and an exactly adjusted algorithm, the matching by use of several ontologies is performed in order to support reasoning tasks. A semi-/automatic approach for matching ontologies by use of several ontologies which are related to a selected research domain will support an expert who is in the need of receiving additional knowledge. However, Euzenat et al. [9] emphasizes that required knowledge depends on the needs of an actor:

> Different actors have different interests and habits, use different tools and knowledge, and most often, at different levels of detail. These various reasons for heterogeneity lead to diverse forms of heterogeneity, and, therefore, should be carefully taken into consideration.

Hence, this work considers matching approaches such as the matching of concepts, features and relations between entities and neighboring entity pairs of several ontologies in a semi-/automatic manner by consideration of the specific requirements of an actor. In general the process of ontology matching is about finding similarities in terms of semantic correspondences between similar entities in two or more homogeneous ontologies [10]. The result of the matching is an merged alignment list usable for reasoning tasks.
The selection of ontologies is based on a known set of ontologies related to a specific domain. The matching approach will consider the matching of entities in different ontologies in order to calculate a similarity value that expresses the grade of similarity between the entities as proposed by Tenschert [11]. However, the calculation of the similarity value is faced with large data sets. This encounters the challenge of processing an ontology matching also for very large data sets and the allocation required compute resources. However, even large ontologies often only consist of data amounts measurable in Kilobyte (kB) or Megabyte (MB) as presented by the data set of the RDF dump site for linked data sets [12]. For instance, table 1.1 demonstrates data sizes for RDF ontologies considering the RDF triple and the RDF Extensible Markup Language (XML) notation. The RDF triple notation considers the RDF structured data as triples constructed as subjects linked with objects via a predicate. Whereas, the RDF XML notation is a well-formed XML document containing the RDF structured data as a nested hierarchy of entities. Each of the presented sample RDF data are increased by the factor of 10 up to a million.

As presented in table 1.1 the data size increases continuously when increasing the amount of RDF structured data but the amount of data varies between both notations,

Amount of RDF Data	Data Size (KB), RDF Triple Notation	Data Size (KB), RDF XML Notation
100	19	25
1.000	213	233
10.000	2.111	1.930
100.000	18.901	16.852
1.000.000	86.344	79.262

Table 1.1: Amount of RDF Data and Data Size

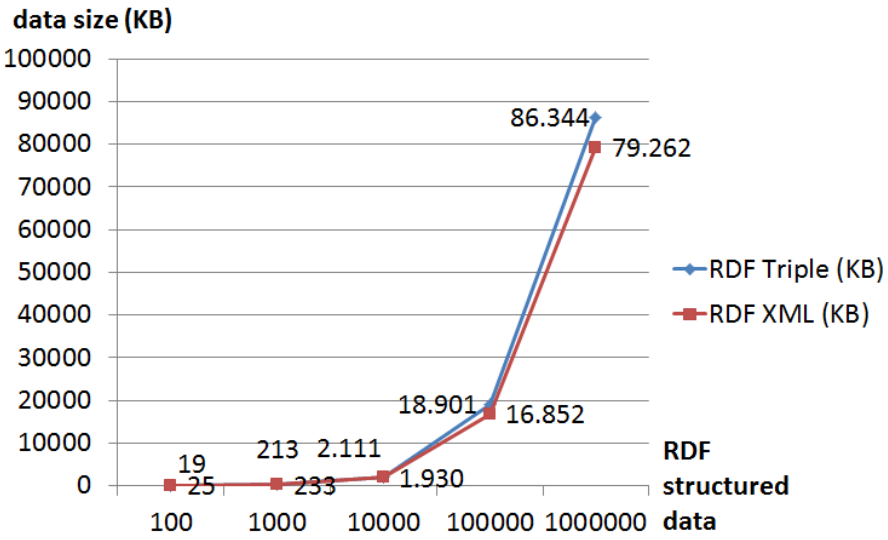

Figure 1.2: Amount of RDF Triples and Data Size

the RDF triple and the RDF XML notation (see figure 1.2). The different data size of both notations is caused through the different type of notation of RDF structured data. While the RDF triple notation annotates the data as simple triples the RDF/XML notation needs to take care of the common nested XML structure of data. Especially, when observing large data sizes, the RDF/XML notation requires more memory than the RDF triple notation. Nevertheless, there is only a slightly different data amount between both notations.

In contrast to the data amount of RDF structured data, the loading and querying of thousands of RDF structured data is a complex and computing resource intensive task being the main challenge when processing those data. As already mentioned in the abstract by referring to the large triple store challenge, this might take hours or even days to generate a useful result. The matching of large scale ontologies implies the problem of handling such large data sets in an efficient manner caused by performing matching processes between millions of entities stored in the ontologies. Further, when thinking of reasoning methods using the matching results as a foundation the need of handling such a matching prompt is a high risk due to very long waiting times

or simply receiving no data for the reasoning due to a data overload.
In order to deal with the research question *How to provide reliable semantic data in an efficient and time saving way for enabling reasoning?*, the main objectives of this work are twofold:

1. Presenting and implementing an ontology matching based on similarity values by consideration of several matching steps.

2. Describing an approach for a distributed ontology matching strategy applicable on HPC resources for enabling urgent reasoning tasks.

The overall goal is to provide a matching enabling a reasoning over a merged alignment list in real-time by adapting improvements of the scalability and performance when executing the matching task. For this, a single ontology being the prioritized one will be the foundation for the matching with target ontologies. The alignment lists contain the finalized matching results convenient for extracting required information through reaoning. In this context, Maedche et al. [13] are evaluating a single core ontology as a beneficial foundation for further methods to measure similarities between ontologies.
The alignment list is used as an information source for specific research domains. Taylor et al. [14] are describing an ontological semantic implementation making use of an ontology as a lexicon but for the purpose to access and represent the meaning of text. However, this work targets the HPC domain and the maintenance of a researcher for specific questions regarding HPC issues but the alignment list will be used as reference book or a specific kind of lexicon as well. Hence, a strategy for generating the result list by matching a priority ontology being the source with target ontologies is presented in this work.

1.2 Chosen Approach

Nowadays, lots of different ontology matching approaches are available. This work considers approaches most beneficial for a semi-/automatic applicability in order to deal with the fact that manual strategies in the research field of matching and improving semantic knowledge structures are time and resource intensive. Furthermore, the ontology matching algorithm needs to be developed for supporting a reasoning strategy on HPC resources. Sahlgren et al. [15] are describing this issue for approaches for construction of a lexicon:

> Manual approaches to lexicon construction vouch for high quality results, but are time- and labor-consuming to build, costly and complex to maintain, and inherently static: tuning an existing lexicon to a new domain is a complex task that risks compromising existing information and

corrupting usefulness for previous application areas.

The fact that a manual execution of semantic approaches leads to a very high effort is taken up by usage of strategies and algorithms for automation of those approaches.
In the scope of this work ontologies are an adequate resource for extracting semantic knowledge out of the given context and relations of the context in the selected ontologies. However, Ramesh and Gnanasekaran [8] considering an ontology in context of the semantic web as available information organized in ontologies. When thinking about knowledge representation and technologies of the semantic web such as Resource Description Language Schema (RDFS) and Web Ontology Language (OWL) the European founded LarKC project [16] offers highly innovative solutions for reasoning in semantic data structures by usage of a platform supporting a parallel execution on distributed compute resources. The general idea of LarKC is to develop a platform which is able to combine and execute several plugins. The plugins are plugged to the platform in order to deploy a workflow consisting of several plugins which ensure the processing of data structures such as with SPARQL Protocol and RDF Query Language (SPARQL) queries. The ability of including plugins allows a wide range of developers to create their own plugins and add them to the platform.
Semantic web technologies and tools are providing methods for explicit knowledge capture and dissemination. Ontologies are describing an accumulation of concepts and the relationships between the concepts and the features of the concepts for providing access to semantic knowledge structures. However, it is necessary to find the access to this semantic knowledge, make it explicit and to search inside the ontologies in order to match concepts of several ontologies to achieve a stable reasoning strategy for the selection of ontologies. Therefore, the syntactic correlations between the ontologies will be exposed to achieve this access. The analysis of the ontologies will take place through a matching method for the ontologies. Nevertheless, before this steps will take place it is essential to evaluate which ontologies are most beneficial for the envisaged purpose of an expert. In addition, an effective reasoning strategy is required in order to extract needed knowledge from the matched and finally merged priority ontology.
A first step for an expert is to evaluate ontologies and store them or the URLs in an ontology repository. There are several aspects that have to be clarified such as the range of specifications and ontology languages that have to be supported in order to integrate the selected set of ontologies 1.3.

Adequate specifications to describe Ontologies are for instance RDFS and OWL while RDF is the base of several ontology description languages. As stated out by Weaver and Hendler [17] when introducing an approach for parallel reasoning on ontologies, the semantic web consists of ever-increasing resource description framework data. RDF is based on the XML standard. In order to use the developed algorithm for ontology matching in a wide range and therefore making it public to the open source community common standards such as RDFS will be applied. Thus, the selected ontologies have to support these specification. Another issue is to evaluate useful ontologies for the described work by considering the used ontology language.

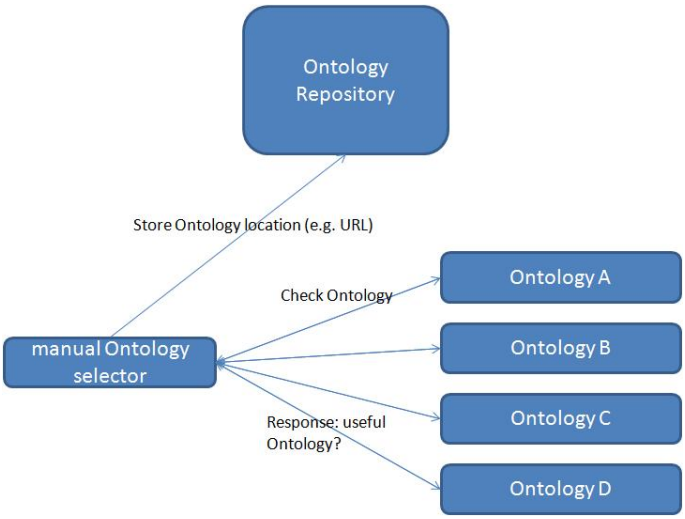

Figure 1.3: Selection of Ontologies

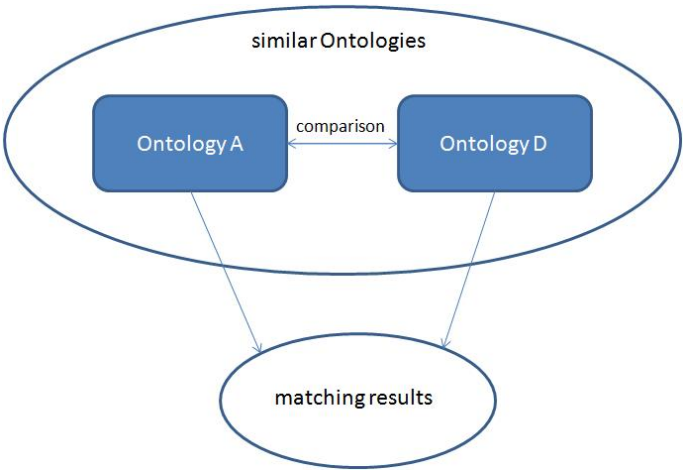

Figure 1.4: Matching between Ontologies

After defining and evaluating a set of ontologies the next step is to explore a matching method for analyzing ontologies which comply with the given requirements. To achieve this method existing ontology analysis will be considered in order to extend already existing methods and enhance them. Through this a new improved analyzing method which is based on proved analyzing strategies is developed. A new ontology matching method which is adapted to the given requirements of an expert is generated 1.4.

Therefore, ontologies must be evaluated considering the requirements of the expert that are focused on a specific research domain. Furthermore, a new matching strategy based on the set of ontologies and existing ontology matching methods will be developed.
In order to ensure a high quality extraction of semantic knowledge it is obvious to

develop a robust and reliable matching algorithm. Therefore, relevant requirements for the selection of ontologies are elaborated. As mentioned before the main goal of the presented work is to ensure a matching of several semantic structures stored in ontologies in order to achieve an individually enhanced knowledge base by means of the alignment list. Ramesh and Gnanasekaran [8] are emphasizing the goal for an ontology merge as the new organization and reusage of existing concepts of source ontologies with the aim to develop a new single terminology. The produced alignment list is based on the previous performed matching between the priority and the target ontologies. It is an ontology as well constructed by such new single terminology.
The ontology matching algorithm refers to the given set of ontologies which is composed by the expert. At this, an expert with background knowledge about his focused work and a little knowledge about ontologies is postulated because of the need to select ontologies fitting to the given use case scenario. The idea is that similar ontologies are selected with the aim of extracting a high number of adequate matching results out of the set of ontologies. Otherwise, the matching of the ontologies will be possible as well but to compare concepts from ontologies which are completely different, e.g. comparing a biomedical ontology with an ontology about engineering, will produce very few or even no useful results for an HPC use case.
A differentiation between the types of matching results is useful for classification. A partial matching improves the classification of types of matching results in a more fine grained fashion and offers the possibility to produce new results. The approach of including partial matches extends the capabilities of matching concepts in a wide range. Nevertheless, to ensure the quality of the extracted results out of the comparisons between the concepts an indicator which presents the degree of accordance is required. For this, a probability indicating the grade of similarity is used to express the level of similarity regarding the matching results. Furthermore, to keep the matching method consistent all three types of matching (no match, partial match and full match) will be described through a probability expressing the grade of similarity between entities 1.5.

A distinction between the degrees of similarity through generation of the similarity value is quite important. Beside the ontology matching approach and the generation of the similarity values and the match and merge based on such, the reasoning processes need to be considered as well. The reasoning tasks are highly dependent on the ontology matching results being the base for the reasoning. Thus, an inadequate, false or no matching will lead to a not feasible reasoning task. The distribution of the ontology matching processes on distributed resources and the allocation of the available computing resources for the matching jobs ensures a high amount of usable computing resources. This proceeding seems to be beneficial and quite promising for enabling an effective ontology matching for enabling reasoning. Tenschert and Cheptsov [18] are describing an effective ontology matching in a HPC environment by considering large scale ontologies needing to be matched in order to provide semantic data in a robust and time saving fashion. Further, Tenschert et al. [19] present an ontology matching approach by focusing on the distribution and parallelization

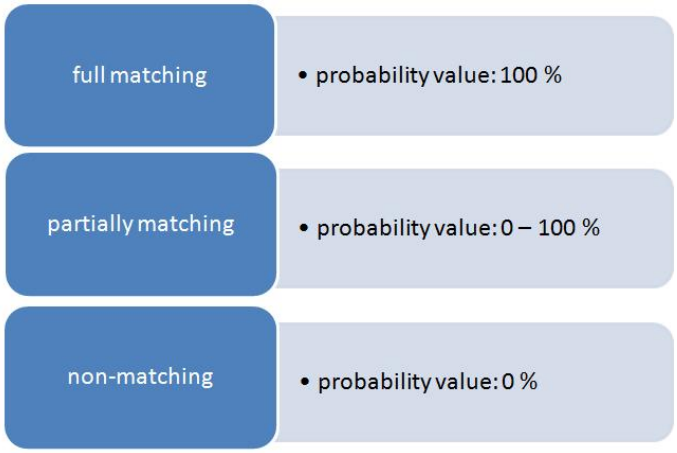

Figure 1.5: Matching Types

techniques. The idea of a distributed ontology matching in a HPC environment is a fruitful approach in order to solve the above mentioned risk and problems when thinking of large scale data.
When thinking of large data sets the one billion triple challenge [20] indicates the amount of data that are used. In the past a typical data set of the challenge includes approximately 1.14 billion statements that have to be processed in a scalable fashion. The zipped size of the dataset is about 17 Gigabyte (GB). However, this size has increased as presented already by the large triple store ([1]) up to trillions of RDF triples. Nevertheless, such a size is processable by traditional computing systems even by consideration of the growing amount of available semantic data in the WWW. Yet, the main concern regarding the data size is the matching process and more important, the reasoning method performed on the data set. This implies the use of a customized knowledge base being adequate to a given use case scenario. Further, such a reasoning task might require several hours or even days (e.g. AllegroGraph [21], OpenLink Virtuoso [22], BigOWLIM [23]). Common reasoning strategies are forward or backward chaining dealing with the use of performing inference rules on an ontology by executing the rules at load time (forward chaining) or when required in case a query need to be answered (backward chaining). Both reasoning approaches need to handle complex reasoning tasks through the execution of the inference rules using the ontologies as knowledge base. Such a reasoning strategy has to be scalable and time efficient even in case a large set of complex inference rules is executed. Especially the forward chaining based reasoning needs to deal scalability issues due to the fact that all inferences are generated at load time. Tenschert and Gilet [24] have discussed the topic of performing an ontology based reasoning on HPC resources by pointing out the need of allocating HPC resources for the ontology matching being the foundation for reasoning tasks. Thus, when thinking of the ontology matching, it need to provide a high quality of matching results in an efficient and fast manner in order to support reasoning tasks. Following this idea, the ontology matching approach is quite crucial for providing an individual and reliable knowledge base enabling experts to identify

urgently required information.

1.3 Research Contribution

The research contribution of this work is a concept for performing a reliable ontology matching in an efficient and scalable manner using HPC resources for creating a customized knowledge base for enabling reasoning. The focus of this thesis is the ontology matching procedure distributed in an HPC environment being the foundation for reasoning in order to provide a relevant set of data, the knowledge base, for individual use case scenarios. Thus, a match of several ontologies for constructing alignment lists as results will be performed. The aim is an updated result ontology supporting experts with required knowledge enabling and supporting reasoning tasks. The distributed ontology matching approach needs to be performed on HPC resources to deal with large and complex semantic rule sets and data sizes even in a restricted time frame.
In summary, the major contributions of this work are:

1. A method for matching ontologies with the aim to construct a customized case specific result ontology by means of alignment lists.

2. An algorithm for executing the ontology matching in a distributed fashion on HPC resources for enabling an efficient matching dealing with time restrictions.

3. Prototypical implementations of the ontology matching strategy to prove the matching concept and enabling the utilization of reasoning tasks using the matching results as an individual knowledge base.

Current matching strategies consider various strategies as it is described in chapter 3. However, this work presents a significant improvement for a semantic matching strategy on HPC resources by consideration of a forward-chaining based reasoning using available data being the knowledge base. Those data are the individual generated result ontology.

1.4 Background

1.4.1 Ontology Matching of Semantic Data in Ontologies

Among others, ontologies contain data usable for work processes, technical configurations or HPC issues. Beside the various contents and topics of ontologies, the format of the ontologies is different which faces the syntactic heterogeneity problem. The simplest syntactic heterogeneity problem takes place through the use of different data formats [10] such as XML, RDF, RDFS, OWL and much more. A brief overview of recommended ontology standards is presented in the following:

- **XML** The eXtensible Markup Language [25].
- **RDF** The Resource Description Framework [26].
- **RDFS** The RDF Schema [27].
- **OWL** The Web Ontology Language [28], [29]. In the scope of the OWL initiative several sub languages of OWL are available such as Web Ontology Language Lite (OWL Lite), Web Ontology Language Description Logic (OWL DL) and Web Ontology Language Full (OWL Full).
- **OWL2** The OWL 2 Web Ontology Language (OWL 2) [30].
- **WSMO** The Web Service Modeling Ontology (WSMO) [31].
- **SPARQL** The SPARQL Protocol And RDF Query Language does not define an ontology but it allows a querying of RDF based ontologies [32].

The RDF format will be mostly relevant for this work because the RDF format is commonly used for semantic data and it is a well known standard as well as the foundation for ontology description languages. A more detailed evaluation of related work is given in 3. Nevertheless, due to the essential nature of RDF and SPARQL both standards will be presented in a brief overview.

RDF RDF is the *R*esource *D*escription **F**ramework used for annotations of semantic correlations between entities. RDF is a common base for a broad range of markup languages used for identifying and describing semantic relations. The RDF was designed for markup of semantics in the WWW but it has become a fundamental basis for the semantic web. The foundation of RDF is the mapping of semantic correlations into three units, a subject, a predicate and an object. The so called RDF triple makes use of

the subject to describe a given resource more precisely described by use of the predicate and finally the subject is linked to another resource described as the object. It is possible that the object is only a literal. RDF provides unique identifiers for resources by assigning URLs to the given resource. The RDF model is a well-formed data model using a formal semantic based on directed graphs. RDF defines resources as triples that can be modeled as mathematical graphs. Powers [33] describes the purpose and benefit of using RDF as follows:

> RDF's purpose is fairly straightforward: it provides a means of recording data in a machine-understandable format, allowing for more efficient and sophisticated data interchange, searching, cataloging, navigation, classification, and so on. It forms the cornerstone of the W3C effort to create the Semantic Web, but its use isn't restricted to this specific effort.

When thinking of common standards in the field of the semantic web well known standards are RDF, RDFS and OWL. In general RDF is based on XML and offers a toolkit for developers for constructing statements, the RDF triples, and as well labeled relations between these statements by use of RDFS. Most semantic standards are based on top of RDF. By use of the RDF structure a simple presentation model is used to apply logic for supporting large-scale information processing for various contexts. Following main features are provided by RDF, nevertheless a broad set of possibilities for constructing semantic statements is supported by RDF structures:

- **Subject** - The subject is a resource that is the source of an arc in a RDF model.
- **Predicate** - The predicate is the property of a RDF triple.
- **Object** - The object of a RDF triple is the value of a statement.
- **Literal** - A literal is a simple string or character that might be the value of a property.
- **Property** - A property is an attribute of a resource such as *¡rdf_prefix¿*.title might be a property, as is rdf.type.
- **Resource** - The resource is the source entry. It can be an abstract or a real entity.
- **Statement** - A statement is an arc in a RDF model that is usually interpreted as a fact.
- **Triple** - A term is another term for a structured statement containing a subject, a predicate and an object.
- **Blank Node** - A blank node represents a resource but without an Uniform Resource Identifier (URI) for the resource. Blank nodes are as existentially qualified

variables constructed by use of first order logic.

- **Dublin Core** - Dublin Core [34] is a standard for meta data web resources.

The RDF structure is a commonly used structure in the semantic web enabling developers to express semantic relations in terms of *subject*, *predicate* and *object*. However, in case the aim is to achieve a higher expressiveness for semantics the RDF standard can be enhanced by use of RDFS. RDFS includes the features of RDF but enables ontology developers to express hierarchies between semantic statements or concepts. By using the additional RDFS elements it becomes possible to construct such a hierarchy similar to a directed graph. Furthermore, the labeled linking between the semantic entities enables the development of an ontology with labeled relations. Table 1.2 describes three RDF triples for the resource *http://www.example.org/materialXY/info.rdf#materialXY101* presenting an arbitrary material. The resource is the subject. The three RDF triples are presented as an directed graph as well in figure 1.6. As presented in this figure, the root node is the subject (resource presented as URL) linked via the predicates (each is a single namespace) to each object. The first two objects are presented as URLs because they are resources and the third object is a literal.

Subject	Predicate	Object
http://www.example.org /materialXY/info.rdf #materialXY101	primarystructure	http://www.ps.materialXY101/
http://www.example.org /materialXY/info.rdf #materialXY101	secundarystructure	http://www.ss.materialXY101/
http://www.example.org /materialXY/info.rdf #materialXY101	name	materialXY101

Table 1.2: Three RDF Triples for a Material

However, expressiveness of RDF can be enhanced, for instance by use of OWL making use of the RDFS elements but offers an additionally set of functionality. One of the main functionality of OWL is the feature to describe classes in a more complex and detailed way. Additionally, different dialects of OWL are available, such as OWL Lite, OWL DL or OWL Full. However, the foundation is RDF and more complex and expressive standards bear the problem of running into scalability and consistency conflicts.

SPARQL When thinking of semantic description languages such as RDF it is crucial to take a closer look on how to use the semantically annotated data. Hence, the *S*PARQL *P*rotocol *A*nd *R*DF *Q*uery *L*anguage abbreviated with the term SPARQL is

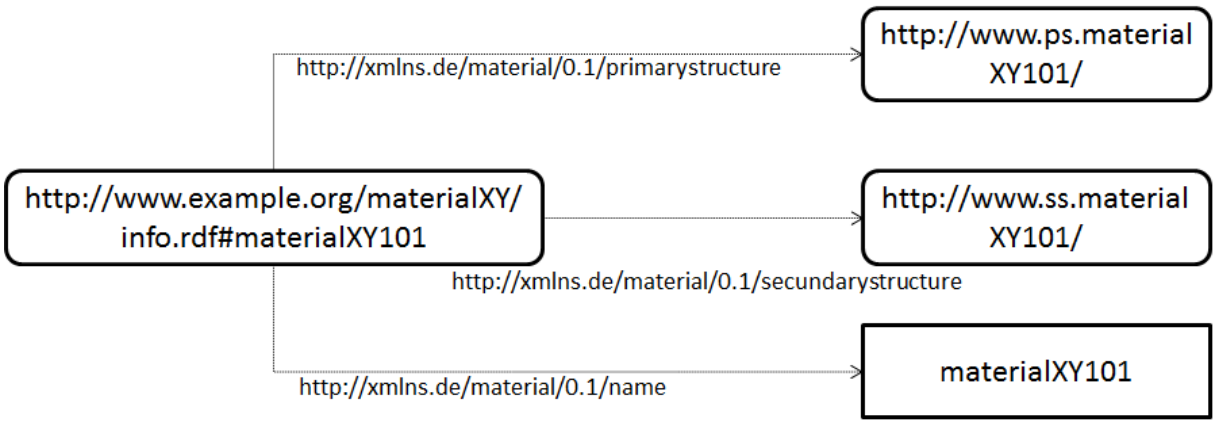

Figure 1.6: Three RDF Triples for a Material as Directed Graph

presented for showing how to make use of semantic data such as RDF data. SPARQL is a graph based query language for RDF. Thus, it is recommended when performing reasoning tasks including RDF queries based on RDF structured data. Additionally, the general idea of the semantic web is to provide data and related semantics. Such semantics need to be described and identified with common standards, for instance with RDF. However, query languages as SPARQL are required to make the semantic data usable for users needing to query the semantically annotated data. A SPARQL query is constructed in a way allowing a user to define parameters of interest and restriction regarding the type of needed semantic data used for the query. A simple SPARQL query can be constructed as follows:

```
1 PREFIX
  SELECT
  WHERE
```

The prefix is similar to a XML namespace, it ensures the use of an identified target namespace by identifying it with an URI. Furthermore, in the given example the parameters of interest are described after the *SELECT* command and the restrictions are described after the *WHERE* command. Besides the presented SPARQL commands the use of SPARQL queries becomes more concrete when presenting the example with parameters referring to an RDF structure. The given RDF structure are the RDF triples from the previously presented RDF triple example. The given RDF triple example is enhanced with a prefix *mp* (material property) for defining the URIs of the predicates. The RDF serialization is a file and contains the following data:

```
  @prefix mp: <http://hlrs.de/ns/materialstructure\#>
2 http://www.example.org/materialXY/info.rdf\#materialXY101 mp:
      primaryStructure http://www.ps.materialXY101/ .
  http://www.example.org/materialXY/info.rdf\#materialXY101 mp:
      secundaryStructure http://www.ss.materialXY101/ .
  http://www.example.org/materialXY/info.rdf\#materialXY101 mp:name
      materialXY101 .
```

The listing above shows three RDF triples consisting of a subject, predicate and an object. A SPARQL query offering the name of the given material by comparison of the primary structure in the RDF file is extracted by use of the following SPARQL query:

```
@prefix mp: <http://hlrs.de/ns/materialstructure\#>
SELECT ?materialName
WHERE
  {
  ?primeStructure mp:primaryStructure "http://www.ps.materialXY101/"
  ?primeStructure mp:name ?materialName
  }
```

Variables are initiated by use of a question mark and the SPARQL query makes use of the prefix as well. The output of the SPARQL query is the name of the material with the fitting primary structure and prefix defined by the URI:

```
materialXY101
```

When thinking of more complex queries a SPARQL query can be extended by adding more parameters in the *SELECT* command or more constraints in the *WHERE* condition. Nevertheless, the example above demonstrates the usability and vital necessity for performing queries on given semantic data sets. The use of a common query language such as SPARQL a simple use of the semantic data becomes possible.

1.4.2 Required Strategy

A strategy handling large scale ontology matching for forward-chaining based reasoning tasks by use of distributed resources is required. Thus, the performance and efficiency problem in the scope of the ontology matching area is solvable. For this, first of all current ontology matching tools and frameworks are elaborated. Furthermore, the current state of ontology matching approaches is analyzed in order to expose the benefits and drawbacks of the different methods. Additionally, the elaborated approach has to deal with distributed computing resources for distributing the matching approach in a HPC environment.
The ontology matching is performed by identifying similarities in different data sets, the ontologies. Thus, a selection of relevant ontologies has to be defined as a foundation for an adequate matching. For ensuring that the matching approach is performed on a beneficial set of ontologies homogeneous ontologies are selected. Nevertheless, it is possible to use heterogeneous ontologies but this might cause a not useful generation of matching results due to the fact that the set of used ontologies is too different in terms of the content. Therefore, the ontology matching strategy need to support a selection of homogeneous ontologies.
Furthermore, the matching strategy needs to clarify which aspects of the ontologies

from the set are taken into account. For instance, an ontology contains concepts, features and values, relations between the concepts and rules. The required strategy has to face the challenge of comparing several aspects of ontologies and the creation of a matching result through a specific measurement defining the grade of similarity between matched entities.
When thinking of the requirements of the ontology matching strategy, the fact that the matching might be performed in a restricted time frame has to be considered. More specifically, the matching strategy takes into account distributed computing resources for dealing with complex and compute intensive matching tasks. The ontology matching needs to deal with large amounts of semantic data and therefore, needs to take into account strategies for increasing the effectiveness and scalability by use of HPC resources for enabling reasoning tasks based on the produced ontology matching results. This work examines this strategy and is made up beside a general introduction to the following chapters:

- **Chapter 2: Background and Requirements.** The research background is presented by identifying requirements of real world scenarios. The requirements are used at a later step for evaluating the results of this work.

- **Chapter 3: State of the Art.** In this chapter the latest strategies and approaches related to ontology matching approaches are examined by consideration of available tools and frameworks. It provides an overview of approaches useful for the approach developed in this work.

- **Chapter 4: Ontology Matching.** This chapter defines the ontology matching approach based on the state of the art analysis. Besides the matching, the complete ontology matching lifecycle is defined.

- **Chapter 5: Distributed Ontology Matching.** Based on the ontology matching approach, this chapter concludes in an improved ontology matching approach usable in a HPC environment. The presented distributed ontology matching approach will enable a forward-chaining based reasoning using the matching results as the needed available data.

- **Chapter 6: Improvements for Use Case Scenarios.** The proposed distributed ontology matching approach is applied to the use case scenarios in order to prove the applicability and usefulness. In this context, the efficiency is measured on a single machine and by use of allocated HPC resources.

- **Chapter 7: Conclusions.** Finally, this work concludes in a summary of reached results, limitations and recommendations for the future.

Through identifying and implementing a required strategy for matching large scale ontologies, a novel concept will be demonstrated being a connecting factor for further

research and developments.

Chapter 2

Background and Requirements

2.1 Background and Requirements

This section covers requirements from real world scenarios by analyzing one general purpose and two specific use cases. They are the foundation for this work and thus, they will be presented in an extensive context considering the given research background. Additionally, the requirements will be evaluated in chapter 6.
The three use case scenarios address the following topics:

- Automotive Engineering (customized UC): Supporting the automotive engineering process postulates a good knowledge regarding development processes and required materials.
- Legal Rule Analysis (customized UC): Comparing legal rules from a complex linked knowledge domain for providing results being additional legal information for a specific case.
- Linked Web Data (general purpose UC): Linked data sets are available for several domains such as weather data collected via sensors.

2.2 Automotive Engineering

2.2.1 Background

Nowadays, large data amounts are usable via the web, databases or further data stores. Lots of them are semantic data expressing certain meanings. When thinking of engineering work processes, we need to distinguish between relevant data from var-

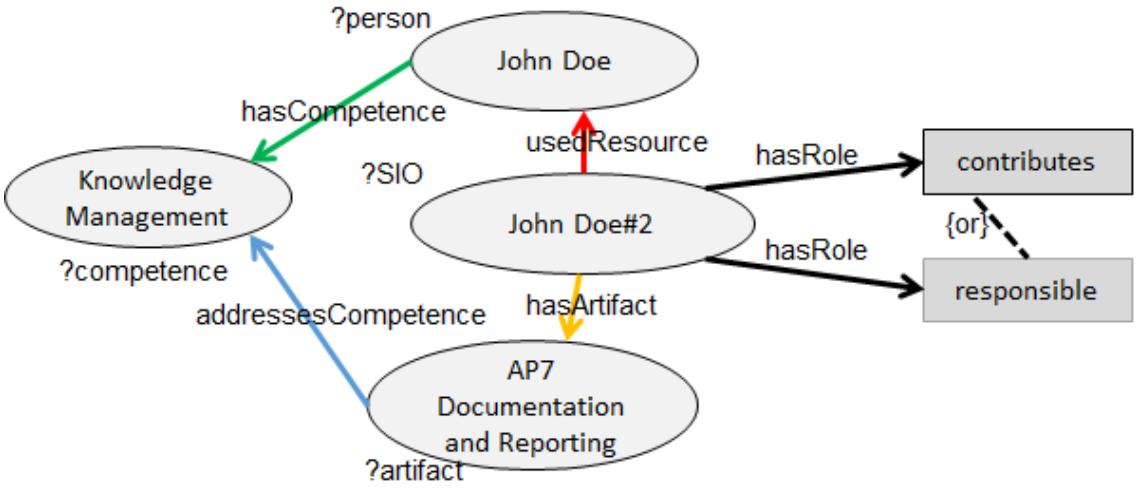

Figure 2.1: Graphical Excerpt of an Ontology for Managing Working Processes in the Automotive Sector

ious data sets. For instance, Syldatke et al. [35] have presented an approach for identifying and processing semantic data stored as ontologies for supporting and accelerating test phases of car components or management structures for work processes. In this sense, ontologies are useful as knowledge base for managing work processes, the nature of specific materials or the definition of certain terms from required domains such as the automotive sector or HPC terminology needed for simulation purposes. A graphical example of a management ontology for work processes from the automotive sector is presented by Schraps et al. [36]. Figure 2.1 presents an excerpt of such an ontology based on Schraps graph based relation model for roles in a work process. Such a relation model is not only used to present management structures but to identify expert knowledge, competences and roles being related to specific work processes.

The ontology presented in this figure presents RDF Triples, described in the following.

```
?person="John Doe" hasCompetence ?competence="Knowledge Management".
?artifact="AP7 Documentation and Reporting" addressesCompetence
?competence="Knowledge Management".
?SIO="John Doe#2" usedResoure ?person="John Doe".
?SIO="John Doe#2" hasArtifact ?artifact="AP7 Documentation and
Reporting".
?SIO="John Doe#2" hasRole "contributes".
?SIO="John Doe#2" hasRole "responsible".
```

Based on the ontology for managing working processes, a structure is composed iden-

tifying the relations and presenting a semantic rule for described competences.

```
<rule name="has competence" type="property">
  ?person[prop#has_competence->competence] :-
    ?person[prop#used_resource->?person]
AND ?SIO[prop#has_artifact->?artifact]
AND (
    ?SIO[prop#has_role->"responsible"_string]
OR ?SIO[prop#has_role->"contributes"_string]
)
AND ?artifact[prop#addresses_competence->?competence].
</rule>
```

The semantic rule is derived from the underlying ontology and enables reasoning tasks for exploring and presenting an excerpt of specific required knowledge, i.e. skills and responsibilities of an employee related to selected work processes. In the given example, a conclusion drawn based on the ontology is, *John Doe* is performing work in a specific artifact *AP7 documentation and reporting* and he has the required competence in *knowledge management*. Further conclusions can be drawn similar, by adapting semantic rules to the given knowledge base. Such a proceeding is called reasoning, it is based on two main pillars, the knowledge base and the semantic rules. This work targets the first pillar, the knowledge base being ontologies, for ensuring a reliable high quality knowledge base being the foundation for customized reasoning tasks.
The amount of diverse ontologies with various structures makes the selection of adequate information very complex. Additionally, the amount of available relevant information in a given research field becomes more sophisticated due to billions of RDF triples. This leads to the question, how to process such amount of information in an efficient and scalable way and how to utilize only necessary information. When thinking of reasoning tasks, the amount, type and structure of information stored in ontologies is relevant for reasoning processes. A reasoning task performed on a well structured data set being closely related to the theme of the query is much more efficient than querying an unknown and large data store. This implies two big problems, the selection and matching of relevant information and the applicability of the provided information by performing reasoning tasks. Nevertheless, the ontology matching is focused in this work but by consideration of the matching results usability for reasoning tasks.

A selection of ontologies from a known set is recommended for concentrating on relevant ones and further, to avoid using competing ontologies with opposing content. In case an ontology has a very low priority for the specific use case scenario, an inclusion in the information selection process will have a negative impact regarding the overall performance and the results quality. In such a case the provided knowledge base would be broad through not required data which would increase the reasoning effort

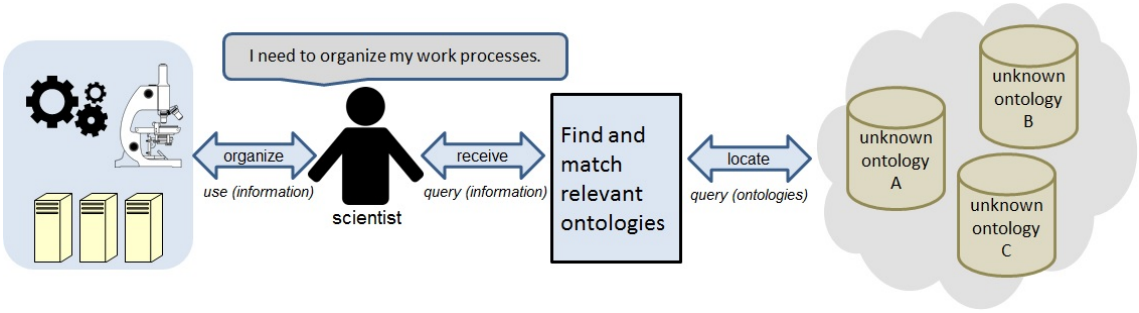

Figure 2.2: User from the Engineering Domain

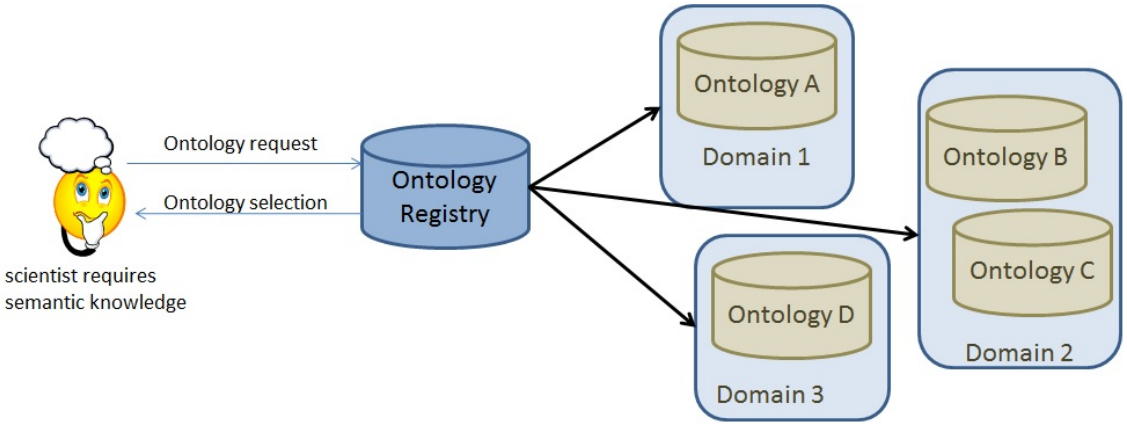

Figure 2.3: Ontology Selection

when performing reasoning tasks. Further, the risk of selecting false data increases due to additionally included irrelevant or not fitting data and thus, the reliability of produced results would decrease.
Generally, the main issue for an expert is to receive information based on the provided knowledge base regarding a specific research issue, presented in figure 2.2. Beyond this, the provided information need to be available allowing an expert to receive required information by submitting a query based on the ontology. In the scope of this work, an ontology matching approach usable in the HPC domain will be elaborated.

2.2.2 The Concrete Scenario

Today lots of semantic data from different domains are distributed all over the WWW. Thus, a researcher needs to know the semantic data locations fitting to the domain of interest. The ontology locations can be stored in an ontology registry, e.g. a meta ontology, a database or a website pointing to the domain specific ontology locations. Thus, first a researcher needs to select an ontology set appropriate for further knowledge capturing processes. The ontology selection phase is presented in figure 2.3.

RDF Triples (NQuad)	Data Size in KB, MB or GB
89	2,5 KB
16.516	293 KB
804.375	7,5 MB
19.655.239	165 MB
80.596.583	1010 MB
198.090.024	4,5 GB
808.977.190	7,1 GB

Table 2.1: Selected RDF Triple Sets of the Billion Triple Challenge

In order to ensure the scalability for reasoning tasks using a wide spread set of ontologies, distribution techniques are relevant for improving the ontology matching in order to provide reliable data. Through distributed matching processes lots of comparisons between ontologies can be processed at same time. When thinking of the automotive scenario this is a quite important issue due to the possibility to match a source ontology containing data regarding skills, roles and competences with target ontologies providing data regarding work processes. Such a matching identifies similarities between the source and the target ontologies and will be able to provide a knowledge base containing a customized set of semantic data fitting to an individually selected work process. Such a proceeding ensures the usage of necessary semantic data instead of a broad data set.
The data sets can include a few up to billions of RDF triples. The data sizes vary from KB up to MB or even GB as already presented by figure 1.2 and table 1.1. The data size of the RDF triple set is easy to handle but the matching processes performed on such data sets are computing intensive. Nevertheless, the billion triple challenge provides large data sets and identifies approaches usable for processing RDF triples up to billions of RDF triples. A set of example data taken from the billion triple challenge is presented in the table 2.1 [37]. The RDF triples are notated in the NQuad notation being RDF triples extended with an extra entity for defining a context.

The Billion-Triple challenge [38] gives an overview of data sizes for semantic data. However, the data size becomes even more computing intensive when the RDF triples are matched. Hence, it need to be distinguished between the data size of the RDF triple stores and the data size when performing comparisons between the RDF triples, the data produced through calculations for assertions. Table 2.1 presents the data size of the RDF triple stores used for performing matching operations. The data set of the billion triple challenge includes approximately 1,14 billion matched statements in 2011. Therefore, it can be assumed that a size of a billion of different semantic statements is a large data set. The matching of such data sets is a time consuming task due to the need of comparing all selected RDF triples with a source for comparison. However, the definition of an acceptable period of time depends on the use case requirements and the instancy of the required results. The given use case expects that the results are required in a short period of time. For this, an effective approach for

matching and querying information in several ontologies combined with the needed computing resources is a useful strategy for supporting a reasoning among large data sets based on several ontologies. When thinking of large data sets execution times for performing a matching vary from milliseconds up to hours or even days. Matching thousands of RDF triples can take several hours or even days depending on the performed matching procedure. Moreover, matching millions up to a billion of RDF triples is a task often not executable on current single computers due to a too long execution time or even a not sufficient computational power. However, the execution time of matching RDF triples is strongly influenced by the selected matching approach. Thus, the main issue of this work is to provide an effective matching approach taking care of execution times.
The presented use case considers data sets used as knowledge base for performing queries. Out of the data sets an expert is enabled to receive new information by matching the already known data in the selected ontologies for generating a customized knowledge base and afterwards, execute a reasoning task by querying the matching results. The new information are stored in an alignment list which is an ontology. The matching is performed by use of a priority ontology as a source for further matching tasks with target ontologies. The priority ontology is selected by an expert from the known set of ontologies. During the matching, the information stored in the selected ontologies are matched with the information of the selected priority ontology. This ontology is updated during the matching by adding new information or updating existing concepts or relations between concepts with new matching results in an alignment list. However, it need to be considered that only matching results with a strong relation to the domain of interest are relevant for the later on performed reasoning. The updated knowledge base, alignment list, is the foundation for the reasoning task.
After updating and enhancing the alignment list, the risk of having produced non coherent results is not resolved yet. This is crucial because the same term in two compared ontologies might have different meanings. Further, a false matching result enables false inferences. However, the risk that different meanings of same terms are not considered by the automatic matching approach needs to be addressed as well as the risk of producing wrong results. To avoid such a mismatch an additional comparison of the terms and relations in the ontologies needs to be performed. A solution for checking the terms and relations of the selected ontologies is highly recommended for enabling an expert to execute the reasoning process based on a reliable knowledge base.
When thinking of an expert needing new information for work processes, the nature of material used for manufacturing or HPC terms relevant ontologies contain information about the organization of specific work processes, material properties or HPC terminology for simulations. To ensure a matching of large data sets as well as the consideration of a high amount of assertional data produced via the matching procedure in an acceptable period of time, the distribution of effective matching procedures in a HPC environment is elaborated.

2.2.3 Identified Requirements for Automotive Engineering

The following main issues regarding this use case are relevant to fit the use case specific requirements.

- Offering an improved base for reasoning through considering various ontologies related to a certain domain, e.g. skills, competences, roles, work processes and material properties.

- The matching of semantic data and even large data sets are the main goal. For this, distribution techniques are used. Large data sets such as billions of statements as identified in the billion triple challenge or the large triple store need to be handled.

- The enabling of an expert to perform reasoning tasks by use of a single priority ontology matched with several target ontologies from a known set. This step produces updated alignment lists used as ontologies by the expert.

- The expert is in the need of receiving information from a certain domain. However, the information is distributed among several ontologies so that the content might overlap within the ontologies or gaps will appear. Through a matching and merging of the selected ontologies in the alignment list a single ontology is available including all relevant data from the selected ontology set usable for reasoning. The expert is enabled to send the request to the produced alignment lists with the aim to receive the required information. The expert has to follow two steps:

  1. select relevant ontologies and define the priority ontology.

  2. execute a reasoning task by use of the produced alignment lists as knowledge base.

  The expert is supported by alignment lists including relevant information regarding the specific needs for the selected research domain. The alignment lists are produced through the ontology matching. At this point an effective matching approach is strongly required in order to provide the alignment lists used as the foundation for enabling reasoning tasks.

Table 2.2 identifies the specific requirements of the ontology matching for this HPC use case scenario.

Req. No.	Definition	Use Case Example

R1	Domain knowledge	Knowledge about useful domain ontologies including information regarding the location, domain, type, structure and data size. A website, meta ontology or database that links to related ontologies (e.g. ontologies about skills, competences, roles, work processes or material properties).
R2	Finding RDF structure	Finding an RDF structure usable for the selected ontologies. Several ontologies with different topics will be used, e.g. departments and experts of a company, production processes and required work materials. The different ontologies need to be compatible in their structure.
R3	Large data sets	Management of large data sets ensuring an effective matching procedure for billions of RDF triples. The billion triple challenge or large triple store.
R4	Validity check	Validity check of matching results ensuring a high quality of produced matching results. The matching results are checked through a validation related to a specific domain.
R5	Maintenance of Result Lists	Maintenance of result lists usable as single ontologies as a foundation for performing reasoning tasks using query languages such as SPARQL. The expert makes use of a single ontology covering questions regarding a certain domain.

R6	Similarity levels	Applicability of a reasoning task through distinguishing different levels of similarity between the matching results for providing best fitting result lists to the expert. The result list is used as a foundation to execute reasoning tasks.
R7	Time efficiency	Clarification of the time when the computing resources have to be available. Usually, the semantic data need to be available on demand.
R8	Information topicality	Life time period for validity of information. An user needs information until a certain point of time. In case of a delayed receipt the information are not valid or no longer required.
R9	Computing resource allocation	The allocation of computing resources is part of the scenario and hence this step is also restricted in time.
R10	Computing resource configuration	The selected HPC infrastructure has to fit to the specific needs of the customer.

Table 2.2: Requirements for Automotive Engineering

2.3 Legal Rule Analysis

2.3.1 Background

Todays legal regulations are defined in huge collections of structured paragraphs distinguishing between several law areas such as the civil code or criminal laws. Especially legislative texts have a very well-defined terminology and a clear systematic. The German civil law manages legal issues between legal persons or between a legal person and an entity. Attempts were performed to digitize such paragraphs in a computer readable way, firstly making legal texts available as online resource and secondly structuring them using standardized formats, e.g. using XML or more with

more expressiveness, RDF data as presented by Schönhof et al. [39]. However, the idea of digitizing and exploiting legal systems is old as the first related papers are dated back to the fifties [40]. Until now, several attempts were performed for describing legal knowledge using semantic web languages [41]; most of them being abstract models not implemented. Since a few years, German law texts are online in the web as texts and structured as XML files [42]. The center of the German law system is the German civil code (BGB) managing, as above mentioned, issues between legal persons and further legal persons or entities. It described the fundamental legal definitions using ongoing numbered paragraphs being subdivided to articles, sub articles and even smaller units of texts.
The amount of legal issues defined and presented in a law framework reaches a vast number of paragraphs and articles. Looking at the paragraphs included in the German BGB, the total number is a few thousands divided by smaller text units as articles, sub articles, half sentences and so forth. Further, due to permanent updates and extensions of a law framework the concrete amount of paragraphs cannot exposed readily. Between 2009 and 2013, Germany resolved 553 federal laws [43] and much more federal state laws. Thus makes it hard to figure out the required paragraphs manually. Further, when thinking of an automated process for finding required information from law related data this process could easily exceed hardware resources of a common PC or cause long waiting times, e.g. a full working day or more.
Regarding the legal rule analysis, the civil code and criminal laws are the knowledge base used for providing relevant information. The continuously growing amount of law texts being related to each other leads to a complex linked data set being expressible as ontologies. The links of a law ontology are given by direct references to law paragraphs, sentences, words and so forth, and indirect through the meaning of entities. Especially, the indirect references given through meaning of entities requires a method for analyzing the meaning of those and match them with further law ontologies. Thus, a matching of law ontologies will examine the given knowledge base and provide useful results as a base for further reasoning processes.

2.3.2 The Concrete Scenario

The concrete scenario aims at an approach to compare law texts by using structured RDF data for supporting users with additional legal information for a specific case. Hence, a brief introduction of law systems based on the German law is given. Further, when comparing law paragraphs relations between the paragraphs are emphasized due to the fact that law texts are not presented as collections of isolated rules but a highly context sensitive network. The complexity of this network is increased through the attempt to reduce repetitions as well as the use of an abstract wording.
Focusing on the German law (BGB), it is divided into five chapters divided according to different aspects of law (*general basic rules*, *obligation rules such as rules for contracts*, *property rules*, *family law* and *succession law*). This brings in another complexity because

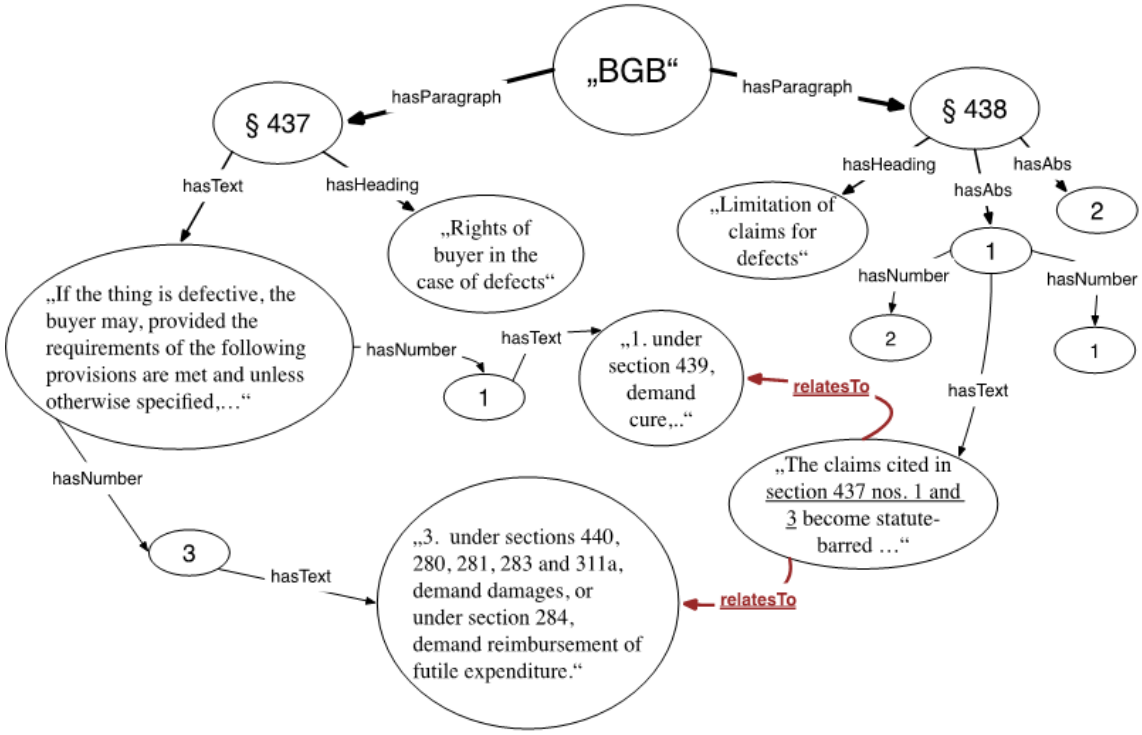

Figure 2.4: Graphical Excerpt of a Law Ontology based on the BGB

a specific case might be regulated by several laws from different law areas. In such a case the more specific law overrules the more general one or a recent law displaces an older one. This implies a permanent domain independent interaction of rules. The relations between laws is compounded by named references or abstract concepts, e.g. specialist terms. Generally, the relation quantity of a legal system is very vast.
Based on the given facts, an automated matching of law texts would support experts with additional information for specific cases. For instance, when searching for law texts related to a given scenario, law texts from other statue books might be relevant. For avoiding a query performed on all available statue books an ontology matching using the law ontologies identifies individually necessary law texts. Those identified law texts will be used as knowledge base for further tasks. In addition, a term described in a law paragraph could be related by its meaning to another term. When thinking of the amount of available law texts and the continuously growing amount of law texts, it becomes more and more difficult to identify related meaning of terms. Experts, being non-jurists and jurists, would benefit from a set of provided case specific law paragraphs offered automatically by an examining and matching procedure. Such a matching between law and general purpose ontologies will examine the meaning of entities used in a law ontology. However, matching procedures for identifying required knowledge are complex calculations and can take several hours. Thus, this process might block progress for a full working day. A graphical excerpt of a law ontology generated from the BGB is presented by Schönhof et al. [44] in figure 2.4.

2.3.3 Identified Requirements for Legal Rule Analysis

The following requirements are identified based on the above scenario.

- An expert requires matching results of several law paragraphs being related to a specific case. This requires the matching of appropriate data. Further, for matching the data in an automated way they need to be structured.

- In addition, due to the complexity of law paragraphs, the matching procedure needs to be able to deal with thousands of law paragraphs. Beyond the data size the complexity of the data is increased by the vast amount of relations between entities. Further, the relations are identified by explicit identifiers or a context sensitive identifier such as specific terms in a text.

- Legal rules are continuously updated and extended thus it has to be assumed that the matching needs to be performed continuously as well. More detailed, a law matching needs to be performed for each given case for ensuring using latest updates. This implies a time dependency.

- The more law paragraphs need to be matched the more crucial becomes the time dependency issue. Further, the required matching time is influenced by the matching configuration, e.g. allocated computing resources, size of data set, matching procedure details, etc.

- The structured data are domain specific in terms of being law paragraphs but it needs to be distinguished between different areas of legal rules. However, the structure of the used data is homogenous.

The above identified use case requirements for the legal rules analysis use case scenario are listed in the table 2.3. For avoiding duplicated requirement items, those being relevant for this use case scenario but already presented in the previous presented automotive engineering use case are not listed in this table. These requirement items are R3 - R9 2.2.

Req. No.	Definition	Use Case Example
R11	Homogenous data structure	In contrast to the automotive engineering use case scenario the data structures are highly homogenous due to similar domains.
R12	Highly linked data	A vast amount of relations between entities.

R13	Context sensitive relations	The relations between entities are labeled through explicit identifiers and context sensitive identifiers resulting from specific terms.
R14	Expert support	An expert being a jurist or a non-jurist needs to be supported because of thousands of available legal regulations.

Table 2.3: Requirements for Legal Rule Analysis

2.4 Linked Web Data

2.4.1 Background

When thinking of data sets available in the WWW such data are often linked data. Linked web data attracted attention due to vast amounts of relations pointing to several entities. Further, data might have a homogenous or heterogeneous structure and additionally domains might vary greatly. Linked web data are referring to Linked Open Data (LOD) being a fundamental part of the semantic web. In addition to linked web data, linked open data are freely available. Such data are available as RDF data sets. For instance, the Linked Life Data (LLD) platform [45] is a data-as-a-service platform providing access to biomedical data structured as RDF data distinguishing between a basic access for freely available use of the linked data and a premium account. Additionally, the linked data web site [46] provides access to linked data in the web and it is related to the linked open data project [47]. Key issue of these initiatives are the idea of linking structured data via URIs or RDF structures across the WWW. For finding alignments between RDF data sets effective comparison methods are crucial for matching RDF data distributed in several domains. Generally, LLD are available for instance for dynamic sensor data (e.g. weather forecast), biomedical data, etc.
Linked web data are semantic data provided as ontologies. Through different domains and focuses of the ontologies as well as the large amount of links a matching of the ontologies in order to provide a specialized knowledge base is a complex task. Thus, matching procedures for ensuring a high quality of results need to be able to handle vast amounts of matching iterations due to the high amount of links.

2.4.2 The Concrete Scenario

In the concrete use case scenario related to LLD, dynamic sensor data for weather forecasts are highly relevant. Those data sets are related to weather focusing on hurricanes or blizzard observations. The complete set of data contains nearly 1.7 million RDF Triples [48], [49] to be taken into account for calculations. A data analysis of those sets requires resources capable for enabling such analysis, e.g. for comparison of collected sensor data, etc. Usually, the LLD are structured as several RDF chunks grouped as RDF triple stores. The RDF chunks are pointing to other RDF chunks included distributed RDF triple store. The LLD comparison enables the identification of sensible segmentation. This scenario leads to questions such as how to distribute the work load when matching several RDF chunks at same time. The links between the RDF chunks are crucial and a matching across the borders of segmented RDF chunks is important for including relevant links and pointers.
In addition, ontologies are developed in different languages and structures. For dealing with ontologies created in different languages, a transformation of the ontologies into the same language is required. This issue counters the fact that a transformation might cause a loss of data and therefore information or data are transformed wrong. Hence, a reliable strategy for the transformation of ontologies is highly recommended. In case of a different structure of ontologies an expert needs to be involved in order to analyze if the selected ontologies are compliant enough for a matching.
The level of involvement of an expert into the ontology matching strategy is another issue to be discussed. The overall idea is to support an user performing an ontology matching using linked web data to enable reasoning task based on the provided matching results being the knowledge base.

2.4.3 Identified Requirements for Linked Web Data

A linked web data scenario usually is in the need of handling results from an ontology match containing several domains of interest. Often the matching methodology is based on big data sets and producing large amounts of assertional data when performing matches. Thus, it is of high importance to provide an effective matching methodology capable to deal with large scale data. In addition, a distributed or high performance computing environment provide such an approach with required computing resources. However, such a proceeding includes scheduling strategies for computing resource allocation. Those resources are the required infrastructure to deal with big data issues.
Besides the already identified requirements the following issues need to be considered when thinking of a linked web data scenario.

- Data sets are linked thus it needs to be ensured that the linked data locations are identified and that the used data fit in terms of used ontology language and

structure.

- The linked data can point to data sets from another domain thus the consistency need to be ensured. This issue might require the transformation of ontologies.
- The matching of used linked data sets is complex and time consuming. A distribution of the matching procedure can increase the overall performance.
- Beyond the technical aspects, the matching procedure needs to be understandable by experts to allow an expert interaction with the matching process. Also, an involvement of an expert into the matching procedure needs to be considered.

Besides the already identified requirement items of the previous sections, additional ones are defined in table 2.4.

Req. No.	Definition	Use Case Example
R15	Identifying RDF chunks	It needs to be elaborated which RDF data from a RDF triple store need to be used for the matching procedure.
R16	Cross-border matching	When matching the LLD chunks the relations and links to other RDF data chunks need to be considered.
R17	Distributed matching	The RDF chunks need to be distributed by consideration of R11, R12 and R13.
R18	Transformation of ontologies	Ontologies might be created in different languages. For performing a matching with different ontologies a transformation of the used ontologies into one joint language is required by considering the risk of losing or producing wrong data.
R19	Expert involvement	The level of involvement of an expert to the ontology matching process needs to be defined by considering a user friendly approach supporting an expert with an efficient and reliable ontology matching approach.

Table 2.4: Requirements for Linked Web Data

2.5 Conclusions and Categorization of the Requirements

The described use case scenarios lead to specific and general requirements. This requirements have to be fulfilled through useful strategies such as distribution of matching processes on several computing resources or involving HPC resources as well as a strategy for matching ontologies in a scalable and effective fashion for ensuring a convenient reasoning. The ontology matching results being the knowledge base are the foundation for a reasoning performed by an expert. In addition, the aim of the ontology matching is identifying and providing semantic data being related to a specific scenario. In this sense, an individually customized ontology provides experts with necessary data being the knowledge base instead of a wide range of data. The set of requirements is listed in the tables 2.2, 2.3 and 2.4. A major issue of this work is dealing with complex calculation processes arising through performing an ontology matching. As it will be presented in this work, an ontology matching process can take a complete working day or more and thus block substantial working progress.
This work targets a fulfillment of the elaborated key issues by considering existing approaches by a state of the art analysis 3. Further, the listed requirements will be analyzed and evaluated in chapter 6.

Chapter 3

State of the Art

3.1 Ontology Matching Approaches

When thinking about the ontology matching research area, various approaches with different advantages and disadvantages can be used. For instance, Tao et al. describing ontologies as simple computational models based on *super-* and *sub-class* relationships [50]. Under that condition Tao et al. elaborating an approach for ontology mining taking into account semantic relations in ontologies such as *part-of*, *kind-of* and *related-to* for extracting information gathered from a keyword-based and a subject/concept-based proceeding. A fundamental essence of the analysis done by Tao et al. is the division of a subject and a relation in an ontology.
An ontology connects a distinct set of subjects by means of concepts. The connections between the concepts define a relationship among the concepts and in addition, the type of relation between concepts by defining rules such as *is-a* or *has-a* labels. Following this assumption, the next sections present existing ontology matching methods.

3.1.1 Information Retrieval

Karlgren et al. [51] present an overview of Information Retrieval (IR) standard methods for matching procedures in a text representation. For this, first a specified text is analyzed, and afterwards a query is build by analyzing a specific information request. In this standard view of information retrieval, text is defined as a repository and transmitter for human knowledge. In order to enable an information system to retrieve the required information the system needs to access the selected text, analyzes and represent it. The system needs to receive an information request, analyze it and construct a query. Thus, an information system is enabled to perform a matching procedure between the text representation and the query. Baeza-Yates et al. [52] present the information retrieval approach in little different but similar way by identifying eight steps:

1. **Information need:** An information is required.

2. **System selection and collection:** A system is selected as well as a collection of documents needed for the retrieval of information.

3. **Query construction:** A query is constructed in order to retrieve information relevant to the query.

4. **Query send:** The query is send to the system.

5. **Results receive:** The results are received as information items.

6. **Results interpretation:** The results are scanned, evaluated and interpreted based on the previously defined query.

7. **Stop and check:** The procedure stops or the next step is performed.

8. **Query Update:** The query is modified and step four is started again.

Both presented approaches are similar, a collection of documents containing the content needing to be identified and further, build a query. Following this, the query is used for receiving relevant information.
In general, an information system needs to reduce the amount of available information in order to make the text representation manageable and further, it needs to deal with vagueness, ambiguities and indeterminacy inherent in human language. Hence, two main challenges for information systems using information retrieval methods are identified as follows:

1. Focusing on relevant information: Reducing available amount of information by not disregarding relevant information.

2. Exposing meaning in text: Dealing with obscurities of human language by determining meaning in a text.

An information system as described above enables a matching between selected texts and a query. However, two challenges have to be solved in order to produce useful results out of the matching procedure. For this, probabilistic methods and vector based approaches, as proposed by and Landauer et al. [53], Salton et al. [54] and Schütze et al. [55], give a hint on how to deal with these two issues using an automated system. We will take a closer look to vector based approaches in the next section 3.1.2. The probabilistic method is summarized by Singhal [56] with the aim to clarify the probabilistic strategy in the information retrieval area. At this, the base idea is to rank documents of a collection by comparing them to a specific query of relevance, the so called Probabilistic Ranking Principle (PRP) [57]. After performing this step, the information retrieval model estimates the relevance of a document based on the produced

probability through applying a ranking strategy. Hence, a probabilistic based estimation of the document relevance is used to make an assumption about the pertinence of a selected document.
The scope of information retrieval is about receiving knowledge structures out of a set of information, e.g. text, documents or graphics. However, the integration of information is an important issue formalized by Lembo et al. [58].

$$< G, S, M >$$

Whereas, the meaning of the Variables is as follows:

- G is a global schema that is expressed in the global language L_G by usage of the alphabet A_G. L_G determines the expressiveness allowed for specifying G
- S is a set of local schema that is modeled in the source language L_S by usage of the alphabet A_S. When thinking of a global schema A_S determines the set of constraints that can be defined. Further, A_S is disjoint with A_G.
- M is the mapping of G and S

The described data integration definition is used for information retrieval approaches for receiving data from a target data set in order to integrate and therefore enhance a source data set. In this case it becomes possible to integrate the local schema (*target*) into the global schema (*source*). Moreover, the definition of the triple $< G, S, M >$ allows a generalized data integration from a target schema into source schema.
Su et al. [6] exposing an ontology mapping definition enabling a matching between a source and a target ontology.

Definition 1 (*Ontology mapping model*). An ontology mapping model is a 5-tuple [S,T,*F*,R(s_1, t_1), *A*] where

- S is a set composed of logical views (representation) for the elements in source ontology.
- T is a set composed of logical views (representation) for the elements in target ontology.
- **F** is a framework for representing ontology elements and calculating relationships between elements in the two ontologies.
- R(s_1, t_1) is a ranking function which associate a real number with an element $s_1 \in S$ and an element $t_1 \in T$. Such ranking defines an order among the elements in source ontology with regard to one element t_1 in the target ontology.
- A is a set composed of mapping assertions. A mapping assertions is a formal description of the mapping result, which supports further

description of the exact nature of the derived mappings. It has the following components:

- a pair of ontology elements,
- a type of correspondence,
- a degree of correspondence, and
- a set of sources of assertion.

Based on this definition the matching process between two ontologies is described as $S, T, \overset{F,R}{\rightarrow} A$. This leads to a method to receive content from a target ontology in order to update a single source ontology.

As presented, the transformation of data from a target source, by use of a data processing method, to a source data set is a challenge in the field of information retrieval as well as for ontology matching approaches. Before stating out the current progress in the area of ontology matching, vector based matching approaches used in the research field of information retrieval are elaborated in the next section.

3.1.2 Vector based Matching Approaches

The random indexing approach belongs to the field of word space approaches. Karlgren et al. [51], Chatterjee et al. [59] and Sahlgren [60] exposing various techniques for demonstrating the viability of representing word meanings as semantic vectors being computed by usage of the co-occurrence statistics of words in texts.

> The use of vector-based models of information for the purpose of representing word meanings in the area of research that has gained considerable attention over the last decade.

Vector based models use words as ambiguous and non-exclusive items considering only mathematical comprehensible facts. Further, key issue of semantic vector based analysis is the assumption that words with similar meanings occur with similar neighbors, assumed that enough text is compared ([51]). The differences of the approaches in the field of vector based analysis is the different idea on how to perform the analysis. One fundamental approach in the research field of word space approaches is the Latent Semantic Analysis (LSA) approach using vector spaces as Landauer et al. [61] presents:

> LSA is a fully automatic mathematical and statistical technique for extracting and inferring relations of expected contextual usage of words in passages of discourse. It is not a traditional natural processing of artificial intelligence program;

The LSA approach considers raw text parsed into words defined as unique character strings and separations of meaningful passages or only samples such as sentences or paragraphs. Related to this raw text snippets, a matrix is created measuring the occurrence frequency of a word in a specific passage or sample. The values stored in the matrix are indicators in a sense that they are mathematical useful values being processed in an algorithm. Further, the mathematical values are connectors in a way that they connect a word with a passage or sample by exposing the occurrence between word and such a passage or sample. The described matrix consists of various cells and each cell indicates the frequency of a context-sensitive word. This matrix is normalized by logarithms for word frequencies and entropies of words across all selected text sources. Through the transformation of the normalized matrix by using singular value decomposition it becomes a much smaller matrix.
The LSA approach was referred as a base for further approaches related to the field of word space approaches such as information retrieval, word sense disambiguation, various semantic knowledge tests and text categorization. Further, the random indexing is an efficient and scalable alternative to the traditional word space methods. The random indexing approach is divided into two steps:

1. Creation of the *index vector*: Each context, e.g. document or word, defined as an unique index vector. The dimensionality d of the index vector has an order of thousands.

2. Creation of the *context vector*: A text is scanned and every time the search term, a word, occurs in the context, the context's d-dimensional index vector is added to the context vector.

In comparison with the LSA approach the random indexing approach generates a matrix similar to the LSA word-by-document matrix whereas the words are the search terms and the documents are texts selected for the word search. However, the random indexing word-by-document is much more smaller then the LSA matrix because of involving the index and the context vector. The random indexing matrix accumulates a document's vector to the row for a given word each time the word appears in the selected text. Karlgren et al. [51] compare the LSA and the random indexing approach:

> By comparison, assuming a vocabulary of 60,000 words in 30,000 documents, LSA would represent the data in a $60,000 \times 30,000$ words-by-documents matrix, whereas the matrix in random indexing would be $60,000 \times 1,800$ when 1,800-dimensional index vectors are used.

The random indexing approach is a based on word space approaches as well such as the LSA. In general the idea of using word spaces is to create a high dimensional vector space for words and further to construct a statistic that is used for the previously mentioned vector space. When thinking of the vector space that is constructed by providence of the previously statistic, this strategy follows the conception that if a set

of words continuously appears in a text in the same context the meaning of the words is the same. Through this, it becomes possible to validate the terms of the matching result with a text document in order to verify if the terms appear in similar context. In case there is a high accordance between specific terms, the probability that the meaning of the terms is the same is very high. Nevertheless, the word space approaches face the challenge of scalability and efficiency in case the dimension of the elaborated vectors increases as stated out by Sahlgren [62]. The use of HPC resources covers this challenge by use of high efficient computing infrastructure but a more fine grained approach for dealing with this issue in order to reduce the amount of required computing resources is recommended.
For this, the simple vector based word space approach is enhanced by using the vector based random indexing approach creating models such as done by latent semantic analysis (LSA) methods. Following this approach, first of all an extensive co-occurrence matrix is created and then second a reduction phase of the co-occurrence matrix is performed that limits the size of the used matrix. Within the reduction phase, the occurrence of upcoming vectors of terms in a specific context is aggregated to accumulated context vectors. Hence, the random indexing approach reduces the amount of required computing resources to perform the validity check in the given period of time.
Nevertheless, the vector based word space approaches are facing the problem of producing useless results under the condition that the text documents that were used for the analysis are leading to misleading results. In order to avoid the generation of not usable results an expert has to perform a very fine grained selection of text documents in order to receive adequate matching results. However, such a selection requires a very good knowledge of the expert that is aware of the content of the text documents and the term that is used as value for the random indexing. Through the involvement of a human expert which such specific knowledge about the context of the terms and text documents, it is questionable if the vector based word space approaches are useful in an automated approach for producing reliable result. This issue counters the problem of decreasing the overall performance by demanding a high effort from an expert.

3.1.3 Data Mining

Data mining methods make use of automated data scanning techniques for specific patterns, models or divergences. Especially a repetitive occurring of a certain pattern can be recognized by data mining methods as stated out by Fayyad et al. [63], Witten et al. [64] or Han et al. [65]. These methods are often statistical mathematical founded. As we have seen in the previous section 3.1.1 information retrieval approaches make use of data collections as well. Nevertheless, data mining and information retrieval are two related but different disciplines. While the focus of information retrieval is the receiving of information based on a certain query, the focus of data mining is the

recognition of particular patterns. However, information retrieval techniques might use data mining methods as a base for collecting required data used for the afterwards performed information retrieval approach.
Traditionally, data mining techniques are used to collect a specific kind of data by, for instance, searching for a certain pattern within a data collection. Therefore, the data collection methods are used for knowledge discovery, proposed by Fayyad et al. [66], and for a statistical learning of machines, proposed by Hastie et al. [67]. The close link between data mining and knowledge discovery becomes obvious when stating out the need for humans to capture useful information (knowledge) held in large digital data sets. Following this approach one can say, methods for data mining can be used to gain required knowledge from digital data. However, one vital problem of combining the both fields of data mining and knowledge discovery is the mapping of low-level data from the data mining procedure into other more complex forms. This is an crucial issue when thinking of the need to provide human usable knowledge from such a procedure. Nevertheless, the bridge between data and knowledge needs to be gaped. Especially, when identifying knowledge as the end product constructed out of data. This idea is presented by Piatetsky-Shapiro [68] by exposing knowledge as the end product of a discovery based on data in a database. However, common knowledge discovery strategies are presented in the next section 3.1.4. Beside the use of knowledge discovery methods combined with data mining approaches, one possibility is the use of probability logic and the elaboration of relations between partial implications between evidence and conclusion. Glymour et al. [69] are pointing out various approaches for a statistical centered data mining by use of different probabilistic based methods. Common data mining methods are presented in the next sections such as a probability distribution (3.1.3.1), a gibbs sampling (3.1.3.2), a hypothesis testing (3.1.3.3), a model scoring (3.1.3.4) and a rational decision making and planing (3.1.3.5).

3.1.3.1 Probability Distribution

A data mining technique for collecting required data out of a collection is to use methods to measure the probability distribution of a certain pattern. Eskin [70] presents an approach for detection of so called anomalies within a data set. However, Eskin refers to a detection of anomalies within a data set of noisy data but the proposed method is probabilistic based and therefore relevant for a probabilistic centered data mining strategy. This strategy covers several steps for the anomaly detection such as modeling a probability distribution, the detection of the anomaly, applying such an anomaly detection to a intrusion detection and detecting of anomalies in sequences of system calls. The idea of probability distribution is presented by Jain et al. [71] as well by presenting techniques for data clustering. Though, data clustering is used for grouping a set of entities into a collection, the cluster, so that the entities in each cluster are more similar to each other than entities in other clusters. This grouping of entities can be achieved through use techniques for identifying a distribution of probabilities

by use of probabilistic clustering methods. In summary, probability distribution techniques are used to identify the occurrence of certain probabilities over a set of data. These proceeding enables an estimation of relevance for specific data patterns based on the probability. A specific method for constructing and identifying a probability distribution is the Gibbs Sampling method (3.1.3.2).

3.1.3.2 Gibbs Sampling

Gibbs Sampling is an algorithm for constructing a sequence of samples of joint probability distributions of two or more random variables. As presented by Gelfand et al. [72] the Gibbs Sampling method is used for building marginal distributions by use of a non normalized joint density. This understanding of Gibbs Sampling is as well proposed in a former work by Gelfand et al. [73]. However, Gelfand et al. determine that this understanding fails in case the Gibbs Sampling method is used as a technique for fitting statistical models. However, the Gibbs Sampling can be used for data mining by constructing the mentioned joint probability distributions.

3.1.3.3 Hypothesis Testing

Hypothesis testing approaches are based on the assumption that a hypothesis is true or false. Such an approach is used in order to perform a statistical hypothesis test in order to make an decision over data. Newey at al. [74] present a method for a large sample estimation by use of hypothesis testing methods. For the strategy proposed by Newey et al. they make use of Generalized Method of Moments (GMM) estimators for testing hypothesis while the GMM estimators itself provide the required functionality for estimations of parameters in statistical models. A former work related to this topic is a hypothesis testing by use of moment estimations presented by Newey et al. [75]. Through the use of moment estimations it becomes possible to verify a hypothesis. When thinking of hypothesis testing in the are of data mining, the aim to examine hypothesis over a collection of data in order to identify relevant data and patterns is one possibility to solve data mining issues.

3.1.3.4 Model Scoring

The model scoring approach presents an strategy for rating certain models or patterns in a text in order to construct a probability value. This value is used for assumption making regarding the relevance of a model / pattern to a specific topic of interest. Glymour et al. [76] are describing the model scoring approach as a method for mapping models by use of a numerical ordering over a set of models or given data. Through

the constructed order it becomes possible to make statements regarding the topic of interest.

3.1.3.5 Rational Decision Making

Rational decision making strategies are often related to planing strategies as well. However, the area of rational decision making can be used for data mining issues for deciding which data to collect by elaborating rational decisions. Doyle [77] presents an approach for decision making and refers to techniques beyond the field of data mining such as reasoning whereas decision making is as well usable for data mining issues. One fundamental goal of data mining is often the extraction of causal information. This can be achieved through stating out cause and effect relations by use of a decision maker. Such a decision maker deals with decision rules for identifying relevant data out of a large data set.

The above described methods are common procedures for data mining. These methods are used to collect relevant data based on a data collection by identifying certain data structures and specific patterns.

3.1.4 Knowledge Discovery

The knowledge discover process can be divided into the following five steps:

1. Selection: Data are selected from a data source and identified as target data.

2. Preprocessing: An initial processing is performed on the produced target data.

3. Transformation: The preprocessed data are transformed.

4. Data Mining: Data mining methods are used to identify certain patterns out of the transformed data.

5. Interpretation / Evaluation: The last step produces the knowledge out of the patterns.

These steps for knowledge discover can as well be adapted to a knowledge discovery over databases as proposed by Fayyad et al. [78]. Within an database related knowledge discovery approach the used data are stored in the database and the knowledge discovery processes are adapted to this collection of data. However, when using knowledge discovery strategies on top of a database a distinction between the knowledge discovery and the data mining methods is required due to the fact that a data

mining algorithm is used for identifying relevant data out of the database. Additionally, knowledge discovery in databases is presented by Frawley et al. [79].

3.1.5 Overview of Ontology Matching Approaches

When thinking of reasoning the research field of ontology matching is relevant as well. In this context Zhang [80] describes the advantage of ontologies and the semantic web for representing information. Gal et al. [81] are elaborating the need of ontology matching by matching of concepts with the aim to describe the meaning of data by considering heterogeneous distributed data sources and considering uncertainties in ontologies. Furthermore, when thinking of an ontology matching approach, this also refers to semantic matching. Fundamentals of semantic matching as it is done by comparison of ontologies is given by Giunchiglia et al. [82]. Additionally, Huang et al. [83] are presenting an overview about the use of ontologies regarding bio-informatics through use of ontologies as formal knowledge representation models to offer knowledge to an expert. At this, ontology matching strategies are required as well for supporting an expert with knowledge that is represented in the ontologies of a various set of ontologies. However, Shvaiko et al. [84] are presenting a brought set of ontology matching approaches within their work and pointing out the fact that at current state lots of different ontology matching approaches are developed by the research community as, for instance, the ontology alignment evaluation initiative [85]. Hence, there is not a single approach available but different directions on how to match ontologies. The use of one of these approaches depends highly on the requirements of a specific use case scenario and the expected outcome of the individual work. Further, as Kalfoglou et al. [86] are describing the landscape of ontologies and mapping tools has increased continuously. Commonly, trends for selection of a specific ontology matching approach are noticed.
Due to the fact that various strategies for ontology matching have become more and more elaborated in the last years ontology matching approaches will be considered regarding this work as well. Some common approaches are listed below:

- A general analysis of ontology matching strategies, the large-scale evaluation of alignments.
- Alignment management with the aim to increase usable infrastructure and support for matching methods.
- Analysis of performance of ontology matching techniques in order to find the best fitting strategy.
- Discovering missing background knowledge to increase quality of ontology matching by using techniques such as data or text mining.

- Managing ontology matching with uncertainties by the use of probability values for matching results and dealing with fuzzy or noisy data.
- An ontology matcher selection out of a set in combination with a beneficial (self-)configuration and improving by adapting automatically matching solutions of such selected matchers.
- User involvement in case that additional information are required by the matching application or improving results by an expert considering matching explanations. The user involvement expects a Graphical User Interface (GUI) usable by human beings.
- Simple and clear explanations of ontology matching results for an expert.
- Social and collaborative ontology matching by the use of networks of users which are communicating to each other.
- Reasoning with alignments by adapting semantics to the ontology matching results and alignments.
- Schema matching approaches considering the structure of ontologies and the use of certain ontology languages.
- Lexical matching comparing terms character by character.
- Comparison of relations to other entities or the relations itself with the aim to reduce ambiguities by consideration of related entities.

The list of possible matching strategies illustrates the complexity of selecting a most suitable strategy. Thus, the required strategy depends on the scenario and the requirements and challenges of a specific research field.

3.1.6 Ontology Matching with Probability Values

A current problem of ontology matching is resolving semantic heterogeneity among information sources such as ontologies for information integration or interoperation. As already stated out in the information retrieval 3.1.1, the data mining 3.1.3 and knowledge discovery 3.1.4 sections a probabilistic measurement for constructing an estimation of relevance for selected contents is a known method in such related scientific fields. Sabou et al. [87] are facing the semantic heterogeneity problem by use of a matching algorithm that includes probabilistic methods as well. In order to ensure a heterogeneous result from an ontology matching approach, ontologies from different

sources can be used. Furthermore, the use of probability values for the matching processes enables an automated matching algorithm to make an assumption about the grade of reliability for the received results. The use of probabilistic methods is useful for handle uncertainties when matching ontologies.
In the following the ontology matching approach with probability values is described. The measurement of similarities is described by Pirró et al. [88] as well. First of all, a description of the necessity of a similarity measurement is presented.

- A measurement of the probability value indicates the strength of similarity, dissimilarity, distance, ultra-metric or normalized (dis)similarity of two entities (e.g. 0.31 = weak similarity , 0.98 = strong similarity)

- a probability value can be determined by comparing several ontologies and comparison of similarity of two entities in all selected ontologies

- a measurement can also be determined by considering the path length between two entities (minLength and maxLength)

Regarding the research field of ontology matching through the usage of similarity and the mentioned approaches such as measurement of similarity strength and the distance between similar entities as a measurement indicator, common relevant approaches are already available. When thinking of the identification of similarity values of compared entities first order logic should be considered. Through the usage of first order logic assumptions regarding the strength of similarity are defined and comparisons between the entities are possible. Furthermore, the usage of Markov Logic Network (MLN), which is closely related to first order logic, should be considered as well. Niepert et al.. [89] are presenting a probabilistic-logical framework for ontology matching that is based on Markov logic. For his they have figured out advantages for using Markov logic:

> Its main strength is rooted in the ability to combine *soft* and *hard* first-order formula. This allows the inclusion of both *known* logical statements and *uncertain* formula modeling potential correspondences and structural properties of the ontologies.

Based on Markov logic it becomes possible to describe matching approaches and to present an enhanced matching result between various ontologies in order to build an enhanced ontology that includes the new Markov logic based alignment. Additionally to that strategies for probabilistic reasoning in ontologies are identified by the use of Markov logic for presenting the data of the ontologies and the matching results.
When thinking of ontology matching as a process for representing connections between entities in heterogeneous ontologies it becomes quite beneficial to use Markov logic approaches for combining first order logic and undirected probabilistic graphical models [90]. Niepert et al. [89] are describing a MLN as a set of first order formula with weights. The level of evidence regarding truth value of a formula influences the

intensity of a formula as well. Hence, an appropriate matching formalism can be used together with a formula syntax and a semantic to achieve a probabilistic based matching.
Approaches regarding distance between semantic entities are relevant as well. Hence, the Normalize Google Distance (NGD) approach and related work will be considered for this work. Through usage of the NGD algorithm for semantic entities it becomes possible to compare the similarity of unknown semantic entities in order to make an assumption regarding the similarity grade of the compared entities. Cilibrasi et al. presenting an approach comparing words and phrases of texts found in the WWW by use of the Google search engine [91]. The WWW is the used database for this method facing the problem that the WWW is the largest available database. However, the presented method is usable with other search engines and databases as well. The produced results are presented as graphs by use of trees. The main result of the work done by Cilibrasi et al. is the use of an LSA based idea 3.1.2 by using Google to process a nearly unlimited size of documents found in the WWW whereby the primary LSA approach is limited in size for constructing the matrix of similarities for compared terms in texts. In addition, similar approaches using the NGD were performed searching in the WWW for texts usable for constructing similarities [92] or searching in Wikipedia in order to use common world knowledge [93].
In general, the mathematical definition of similarity between two entities is described by Euzenat et al. [9]:

> A similarity $\sigma : o \times o \rightarrow \Re$ is a function from a pair of entities to a real number expressing the similarity between two objects such that:
>
> (1) $\forall x, y \in o, \sigma(x, y) \geq 0$ *(positiveness)*
> (2) $\forall x \in o, \forall y, z \in o, \sigma(x, x) \geq \sigma(y, z)$ *(maximality)*
> (3) $\forall x, y \in o, \sigma(x, y) = \sigma(y, x)$ *(symmetry)*

The above presented formula expresses similarities between entities while formula 1 to 3 are necessary for constructing the similarity between two selected entities.
Beyond the mentioned approaches for measuring probabilities, Pirró et al. [94] state out the difference between methods for identifying similarities and relations. They distinguish between a similarity in case that for instance two concepts are alike each other and as related if the connection between both concepts is a relation. Most approaches are focused on either the similarity or the relation between entities or, as it is done in the work by Pirró et al., concepts. However, both research approaches are relevant for constructing an estimation regarding similarity between two entities. Nevertheless, the the distinction between entities itself is handled by adapting instance matching approaches as presented in the next section 3.1.7 to identify the meaning of an instance in an ontology.

3.1.7 Instance Matching

The instance matching domain is a certain research field of the ontology matching domain. However, instance matching focuses on the instances of ontologies with the aim to deal with ambiguities. In this scope research has be done as, for instance, by the instance matching initiative [95] belonging to the ontology alignment evaluation initiative (see 3.1.5). Research in this field has been done, among others, by Engmann et al. [96] making use of COMA++ 3.2 for matching schemas in order to identify semantic correspondences between such schema. Engmann et al. used constraint and content based techniques by adapting similarity values for presenting similarities between entities and parent entities. The aim of instance matching is to identify entities pointing to the same real world objects. In addition Ferrara et al. [97] perform a benchmark for instance matching by considering typographical errors of entities and the use of different standard formats of ontologies. Hence, they analyze, for instance, the structural heterogeneity by means of different levels of depth and aggregation criteria for entity representations and additionally taking into account missing value specifications. The benchmark is performed by observing the precision of results, the number of recalls, the errors and the execution time. Thus an identification of the quality of a matching algorithm becomes possible. In general, instance matching is crucial for ontology matching due to the need of identify the meaning of entities.

3.1.8 Graph Based Matching Approaches

An ontology can be described as a graph. In this context Mao et al. are referring to ontologies as a type of taxonomic trees including concepts, features and relations that are associated with instances [10]. Following this approach, the ontology developer assumes that the presented composition of an ontology is valid:

1. An ontology has one starting entity that is linked with other entities of the ontology.

2. Each entity that is linked with one or more other entities is a concept.

3. Each concept can be described with features. These features are containing values.

4. Each link between concepts makes an assumption about the type of relation between the linked concepts.

Considering the listed points above, when representing an ontology as a graph, the first concept of an ontology is the starting element of the ontology, the links are the edges and the concepts are the nodes. However, each node is described with features

containing specific values. The graph model is presented as a tree. Furthermore, the edges are annotated with remarks regarding the type of connection.
For the comparison of different ontologies an approach is to consider graph analysis to make an assumption about the structure and content of the used ontologies. Nevertheless, the content of the concepts (features with values) and the annotations of the edges have to be considered if the graph wise comparison should produce a result considering the information stored in the ontologies.
The ontology matching by use of graph analysis methods seems to be convenient under the four listed conditions that are mentioned above. In general, a graph G consists of nodes N and edges E that are connected to each other:

$$G = \{N, E\}$$

An ontology O that is represented as a graph consists as well of nodes N that are concepts and of edges E that are the relations. Additionally, each edge E can be marked with a label L that is a rule such as *is-a* or *has-a*, each node N has features F and values V of the features F. Therefore, the following formulas describe ontologies as graphs.

$$O = \{N, E\}, \text{ whereby } N = \{F, V\} \text{ and } E = \{L\}$$

- O is an ontology,
- N is a node that is a concept,
- E is an edge that is a relation between nodes,
- F is a feature of a node,
- V is a value of a feature,
- L is a label of an edge.

When thinking of the presented definition of an ontology, the use of graph based matching methods should be considered. Blum et al. [98] are presenting an approach to construct a graph plan as an improvement for a graph analysis. The graph plan described by Blum et al. is a complex structure that is analyzed by a graph plan procedure that always returns the shortest possible partial order plan or it produces a notification in case that no such partial order plan is available. The approach of Blum et al. is one way of processing a complex graph structure in order to state out the shortest graph plan. However, the matching of graphs requires fine grained elaboration of the graphs or the partial graphs that need to be explored. Often computing problems that are based on graph analyzing methods rely heavily on the structure of a graph such as a comparison of two graphs focuses mainly on the structure. Nevertheless, in case an ontology is presented as a graph not only the structure but even the

content is important. A reliable and trustworthy ontology matching approach needs to deal with a comparison of the structure in terms of the relations between concepts and the content that is stored within the examined ontologies. Hence, when applying graph based matching methods to an ontology matching procedure the meaning of the content that is stored in the ontologies has to be taken into account as well.

3.1.9 Distributed Ontology Matching

In the following section a brief overview about strategies on how to handle an ontology matching for reasoning issues in a distributed environment is given. Kalfoglou et al. [86] have already described the situation of ontologies at current state. The number of ontologies available in the web has continuously increased during the last years and the use of a single ontology for reasoning issues is not longer an appropriate solution because of the fact that lots of more information are needed as a reliable and trustful foundation. These information are often stored in a set of ontologies; not a single ontology. Hence, a set of relevant ontologies need to be taken into account when searching for a specific research topic. In addition Bhatt et al. [99] describing ontologies as a fundamental issue of the semantic web because of the possibility to split up information from the web and store them regardless of implementation language and structure of the data. Further, they state out the current situation of significantly growing amount of data in the web. This causes large scale web ontologies growing induced by more and more data in the web. In terms of scalability and efficiency the growing amount of data in ontologies needs to be addressed by current ontology matching approaches. Serafini et al. [100] counter this problem with an approach using multiple local ontologies connected via semantic mappings. This approach makes use of Distributed Description Logic (DDL) but facing the problem of inconsistencies in DDL needed to be solved. However, in order to make an assumption regarding relevant ontologies some open issues need to be answered such as:

- The overall topic and content of the selected ontologies.
- The structure and used ontology language of the selected ontologies; a matching of several ontologies will fail in case the structure and used language is not considered.
- The availability and location of the selected ontologies; especially in a distributed environment it is essential to clarify the location and access to an ontology on a specified computing resource.

Generally, ontology matching methods for providing required data and information need to deal with above mentioned issues as presented in section 3.1. Nevertheless, a distributed ontology matching approach has to cover aspects such as allocation of

required computing resources, distribution of raw data such as source ontologies, location of produced output data and distribution of the matching algorithm.

3.1.10 Ontology Matching and Integration

Ontology matching methods often consider information integration aspects. The goal is to provide an integrated and coherent view of information stored in multiple, possibly heterogeneous information sources, such as several distributed ontologies. It is one of the core problems in modern distributed, cooperative information systems in which the ability of different components and applications to share knowledge and inter-operate seamlessly is of utmost importance. Data from different, possibly incompatible and heterogeneous ontologies are combined, aggregated and reasoned upon in order to reach meaningful conclusions. The integration of ontology information is a major issue when thinking of ontology matching.
A general definition of the information integration problem is given in Calvanese et al. [101], Lembo et al [58]. In this model, the conceptual layer of an information integration problem distinguishes between a target model and multiple source models. The target model is a conceptual representation of the global concepts and relationships that are of interest to the application. The source model of an information source is a conceptual representation of the data residing in underlying information sources. This definition of an information integration system is general enough to capture virtually most approaches in the literature. Obviously, the nature of a specific approach depends on the characteristics of the mapping, and on the expressive power of the various schema and query languages. The main challenge is to discover the semantic relationships of terms in different ontologies. By accepting an ontology as a point of common reference, naming conflicts are eliminated and semantic conflicts are reduced.
In the majority of ontology integration approaches, matching is typically performed manually. Obviously, manually specifying schema matches is a time-consuming, error-prone, and therefore expensive process. Moreover, there is a linear relation between the level of effort and the number of matches to be performed, making manual approaches inadequate for huge data sets, where a faster and less labor-intensive integration approach is needed. This requires automated support for schema and data matching.
Another problem with current approaches is that, due to their complexity and lack of an intuitive interface, they only appeal to experts, thus making matching an expensive process. Given that the use of multiple matchers may be required, two sub-problems emerge. The first is the design of individual matchers, each of which computes a mapping based on a single matching criterion. The second is the combination of individual matchers, either by using multiple matching criteria within a hybrid matcher (e.g., name and type equality) or by combining multiple match results produced by different match algorithms in parallel or sequentially.

3.2 Relevant Tools and Frameworks

The following section presents common tools and frameworks in the domain of ontology matching followed by a summary of advantages and disadvantages based on the evaluation of the previously mentioned tools, the presented approaches and methods of section 3.1 and 3.1.9. However, the mentioned algorithms, tools and systems are considered as most relevant to the given work.

AgreementMaker The AgreementMaker system [102] is evaluated by Cruz et al. [103], it provides the possibility to match ontologies, schemas or schemas and instances and by consideration of up thousands of concepts. However, the end user of the AgreementMaker system needs to be a domain expert. The AgreementMaker system supports the matching of concepts and structure of ontologies in manual or automated manner. Furthermore, it provides a GUI presenting concepts and features of two different ontologies by linking similar entities of both.

Anchor-prompt Anchor-prompt is an algorithm finding semantically similar terms automatically. Noy et al. [104] implemented the Anchor-prompt algorithm for comparing two ontologies treated as graphs with a result of 75 percent correct matches. Musen et al. [105] describe Anchor-prompt as an approach dealing with a graph based method to treat ontologies. The assumption in Anchor-prompt is that two entities are similar and therefore there are similar structured graph connections and paths that can be analyzed.
It is possible to adapt well proofed graph based matching strategies to the Anchor-prompt approach. Compared ontologies need to have a similar structure so that the graph based matching approaches can be used. Furthermore, it is a time consuming way of performing the matching and the amount of produced correct results needs to be improved in order to deal with critical use case scenarios.

AROMA AROMA is an ontology matching approach using data mining methods by consideration of associations, it was developed in the scope of the AROMA project [106]. The focus of the AROMA project is the matching of OWL ontologies with the aim to explore relations between entities by means of equivalences and subsumptions. The relation based matching is useful to analyze similarities between selected entities but it does not deal with ambiguities such entities.

ASMOV The automated semantic mapping of ontologies with verification (ASMOV) includes an iterative similarity measurement, a semantic verification of the produced

results and a user involvement by allowing the user to interact with the ASMOV system. In addition, applications for information integration and semantic cataloging are supported. The ASMOV approach is divided into several steps, the first is a lexical matching in order execute a pre-processing for the similarity calculation. The results from the lexical matching are used for the similarity calculation in order to extract alignments verified by the semantic verification. Afterwards the produced alignments are set to final or rejected by the verification. The ASMOV approach is already producing accurate results by consideration of acceptable execution times but further work needs to be done for dealing with very large ontologies, for instance, by adapting parallel execution strategies as stated out by Jean-Mary et al. [107].

BLOOMS Jain et al. [108] developed a system called blooms based on the idea of bootstrapping information provided by linked open data (LOD). Blooms constructs a set of trees for each matched entity. Afterwards, the tree sets of the given entities are matched. The input for the BLOOMS system are two ontologies containing schema information. BLOOMS considers two main issues, first the bootstrapping by use of noisy data and second, the use of Wikipedia provided by DBPedia as data source for the bootstrapping.
The BLOOMS approach reaches a high precision of results and recall values. However, future work for this approach is the consideration of partonomical relationships or disjointness in the used LOD.

CAIMAN The CAIMAN system performs an exchange of relevant documents between users dispersed over different locations. In this scope each user organizes documents according with the individual domain knowledge and an own categorization scheme, the ontology. The role of CAIMAN is to explore such ontologies under the aspect of information retrieval and provides relevant information of the documents to the users. CAIMAN makes use concept matching methods in order to compare a concepts of an existing ontology with a concepts of a community ontology for extracting required information. Lacher et al. analyze the CAIMAN system [109].

Chimaera Chimaera [110] is a software system dealing as well with heterogeneity issues as presented by McGuiness et al. [111]. However, its main focus is supporting a user for the creation and maintenance of distributed ontologies in the WWW. The two main functions of Chimaera are first, the support for merging multiple ontologies together and second, analyzing single or multiple ontologies. Chimaera is usable as a user friendly tool providing a graphical representation.

CIDER CIDER [112] is a system for producing ontology alignments. For this, it compares two ontologies for extracting a semantic context by use of a semantic rea-

soner. Afterwards, the ontological context is used to compute a similarity between the matched ontological contexts by performing a matching in three steps, (i) a lexical,(ii) a taxonomical and (iii) a relational matching. The similarity value produced by the similarity computation is used by an artificial neural network to produce the matching results. The artificial neural network performs, for instance, a lexical distance analysis and a vector space modeling. CIDER reaches a high precision of the constructed results and an acceptable recall value. However, there is space for improvements, especially when dealing with very large ontologies.

COMA++ COMA is presented by Do and Rahm [113]. Based on COMA the newer version COMA++ is presented by Aumueller et al. [114], it considers schema level information for its matching strategy. The schema matching is performed by use of a taxonomy matcher comparing ontology schemas by linking them to a taxonomy used as an intermediate ontology. A similarity value for the matched entities is produced out of the matched schemas. Engmann et al. used COMA++ for schema matching as presented in 3.1.7.
COMA++ offers the possibility to utilize different matching strategies as, for instance, a constraint or content based matching by supporting the user with a GUI. Furthermore, COMA++ supports three matching strategies [115]:

1. Context based matching: The context based method is required for matching schemas with similar entities.

2. Fragment based matching: The fragment based matching is based on the divide and conquer idea, decomposing a large matching task into a subset of smaller matching tasks.

3. Reuse oriented matching: The reuse based matching makes use of already available matching results produced by a previous performed matching.

ContentMap ContentMap (An logiC-based ONtology inTEgratioN Tool using MAPpings) [116] is a system for mapping ontologies for integration and heterogeneity issues. ContentMap is a Protégé plug-in extending the Protégé GUI with necessary features to provide a mapping evaluation and in addition provide a mapping repair resolution method.

CROSI CMS The CROSI project [117] developed a semantic integration and interoperability survey, a framework for characterizing semantic integration systems as well as an architecture and a system. The overall system is an ontology mapping system. However, it makes use of the Jena libraries as described later on.

Cupid Cupid is presented by Madhavan et al. [118], it makes use of a matching strategy based on schema level information. For this, the schemas of selected ontologies are matched in order to make assumptions regarding the similarities of entities.

DSSim The DSSim algorithm can be used to manage uncertainties on the web by use of ontology mapping methods as presented by Nagy et al [119]. The presented approach deals with issues such as incomplete and inconsistent information produced by ontology mapping processes.

Falcon-AO Falcon-AO is an automatic ontology alignment tool for which a set of elementary matchers such as V-Doc, I-Sub and Graph Matcher for Ontologies (GMO) are implemented. Furthermore, Hu et al. [120] figure out Flacon-AO as a prominent component of Falcon for aligning ontologies automatically by combining the elementary matchers with the ontology partitioner, the Partition Based Block Matcher (PBM). The integration of PBM into large-scale ontologies and the usage of a central controller of Falcon-AO enables the matching with large-scale ontologies. The PBM divides the large-scale ontologies into matching blocks that are connected with anchors so that it becomes possible to match in the matching blocks and not in the whole large-scale ontology. The central controller integrates the various alignments by set of rules that are based on linguistic and structural comparability. Further, Falcon-AO makes use of the elementary implemented matchers.
Falcon-AO provides the PBM for matching large-scale ontologies. Through this, Falcon-AO achieves a good performance for effectiveness and efficiency [120]. The partitioning is not always optimal for ontologies with complex relations, Falcon-AO might fail to achieve a sufficient quality for matching results when applications are related to specific domains and Falcon-AO is not able to provide alignments with a semantic relationship [120].

FCA-Merge The FCA-Merge algorithm provides a method for merging ontologies by a bottom-up approach. Thus, it becomes possible to describe the structure of the merging process. Stumme et al. [121] present the FCA-Merge method in order to apply techniques from natural language processing and formal concept analysis.

FOAM FOAM is an ontology alignment tool [122] for performing a semi-automatically alignment between two OWL ontologies. For this, FOAM makes use of similarity heuristics in order to match the given entities such as concepts, relations and instances. The result set produced by FOAM are pairs of aligned entities.

GLUE GLUE is presented by Doan et al. [123], it considers instance data in order to find semantic mappings between two matched ontologies. For this, machine learning methods are considered in order to perform a usable matching producing similarity values for matched entities such as the concepts of the ontologies. Additionally, information regarding the structure of the compared ontologies are used with the aim to verify and update the previously generated similarity values.

HCONE As presented by Kotis et al. [124] HCONE is used for providing a mapping and merging of ontologies. For this, HCONE is used for ontology merging by interpretation of the concepts of selected ontologies. The interpretation is done via a mapping to WordNet senses and performing a semantic indexing and afterwards analyzing the definitions of concepts by use of reasoning methods using description logics.

Jena Semantic Web Framework The Jena Semantic Web Framework is developed for building java based semantic web applications. It provides an environment for RDF, RDFS, OWL and SPARQL and including a rule-based inference engine. Furthermore, the Jena Semantic Web Framework provides several operations on models.
The Jena Semantic Web Framework is an open source project and it [125] including:

- an RDF Application Programming Interface (API)
- methods for reading and writing RDF in three different notations (RDF/XML, N3, N-Triples)
- an OWL API
- an in-memory and a persistence storage
- a SPARQL query engine

Besides the included features, the Jena Semantic Web Framework is easy to use for Java developers and offers a wide range of functionalities regarding RDF, RDFS, OWL and SPARQL such as:

- navigation in a model in order to find resources and properties in a model
- querying a model by querying the complete model or a specific part of the selected model
- performing operations on models such as union for union of the statements in the model, intersection for constructing a new model with the listed statements

and difference for constructing a new model containing the statements that are not listed in another model

One typical disadvantage is the decreasing computational power in case of performing complex reasoning tasks. This issue might be addressed by providing HPC resources or a distribution of the tasks on several resources. In addition, the Jena Semantic Web Framework ontology support is build on top of RDF. Jena is mainly used for writing java applications for matching semantic content.

LarKC In the scope of the LarKC project (The Large Knowledge Collider (LarKC)) [16] a platform for massive distributed incomplete reasoning removing the scalability barriers of currently existing reasoning systems for the Semantic Web was developed. The LarKC project has identified Java based efficient data centric semantic web applications and defined a base for parallelisation of such applications. Furthermore, LarKC is not restricted to ontologies but it offers the possibility to process semantic data such as it is provided in ontologies in a distributed fashion. Thus LarKC distinguished between the LarKC platform and LarKC plugins.

- *LarKC platform*: The LarKC platform an platform for effective distributed reasoning, it is available as an open source sourceforge project [1].

- *LarKC plugin*: A LarKC plugin is a piece of code that follows the LarKC guidelines for developing LarKC plugins. LarKC plugins are used with the LarKC platform, they are plugged into it. Within LarKC a distinction between five types of plugins is done: identifier, transformer, selecter, reasoner and decider.

    1. Identifier: The identification of relevant data.

    2. Transformer: The transformation of the identified data so that it can be used for further tasks.

    3. Selecter: The selection of data that will be used for a reasoning task.

    4. Reasoner: The reasoning task using the selected data.

    5. Decider: The decision which results produced by the reasoning to chose.

The distinction between the platform and different types of plugins enables a developer to make use of the platform for distribution of the workload through development of individual plugins or reusage of existing plugins.

[1] The latest LarKC release: http://sourceforge.net/projects/larkc/

Lily Lily is a system for ontology mapping. It supports four matching functions, a generic ontology matching, a large scale ontology matching, a semantic based ontology matching and debugging of the mapping. However, the ontologies used for the Lily system are heterogeneous. The matching itself is performed by combining the four matching strategies and it is therefore a hybrid matching. Wang et al. [126] performed a benchmark for the Lily system in the scope of the ontology alignment evaluation initiative in 2009 and stated out the strengths and weaknesses of the Liliy system. According to Wang et al. Lily produces satisfactory alignment results for normal size ontologies but it needs to deal with the extraction of semantic subgraphs for all concepts and features which courses a decrease of performance.

MAFRA MAFRA (A MApping FRAmework for Distributed Ontologies) [127] is a framework for the mapping of distributed ontologies. MAFRA considers two ontologies a target and a source ontology and creates semantic relations between both ontologies. Beside the ontology mapping approach it provides a GUI. Maedche et al. [128] analyze MAFRA by focusing on a meta ontology for the semantic bridging the mapping between the two domain ontologies. In this scope the additionally used meta ontology is crucial for generating useful mapping results.

OntoBuilder The Onto Builder project [129] makes use of ontological constructs for an automated schema matching. For this purpose, ontologies from the WWW are discovered by web search engines. The selected ontologies might be from different domains. After the selection of the ontologies a mapping between them is performed in order to generate an improved single ontology.

OntoMediate The OntoMediate project (Ontological Mediation and Semantic Gateways for Domain/Enterprise Translation) [130] focuses on the exploitation of social and collaborative processes for improving ontology matching tasks by consideration of integrated data. For this, data of shared ontologies are used in order to support a flexible alignment generation. However, OntoMediate needs to deal with different formats and meanings of the ontologies and the entities. The OntoMediate approach takes into account a community support for evaluation of ontology vocabulary, an ontology mapping for dealing with different information representations of meaning and an information network containing different ontologies used for the OntoMediate approach. The OntoMEdiate approach relies on the data integration and the involvement of users performed by a community driven management.

plugIT The plugIT project (Plug Your Business Into IT (plugIT)) [131] provides the Next Generation Modeling Framework (NGMF) for plugin business knowledge into an Information Technology (IT) domain. The business knowledge as well as the IT

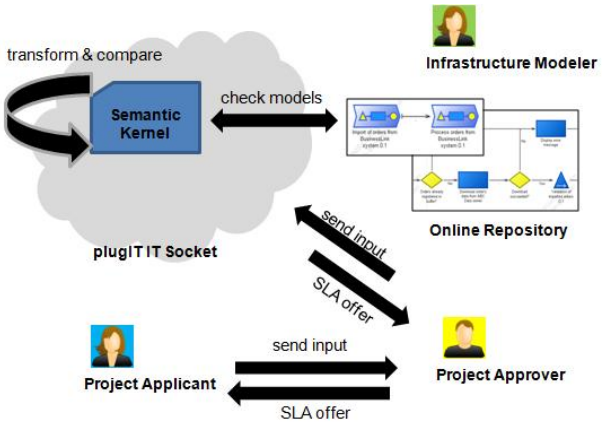

Figure 3.1: Resource Allocation Overview

processes are modeled in a modeling language. Furthermore, the models are translated into model ontologies (MOs) in the plugIT semantic workflow that are used to match business and IT ontologies. The ontology matching in the plugIT project is processed in the plugIT IT-Socket. In general, plugIT supports a semantic workflow for the needs of high performance computing issues in terms of offering SLAs about HPC resources to handle customer requests for such resources (Gilet et al. [132]). The semantic workflow includes the plugIT semantic kernel that matches the ontologies with the aim to recommend the best fitting SLA for IT resources fullfilling the requirements of the customer and the IT provider. Through this, the business needs and the IT solutions are aligned by the plugIT IT-Socket.
Figure 3.1 presents an overview about the interaction between involved persons and the plugIT IT-Socket. The plugIT approach is efficient for allocation of required HPC resources by use of SLAs developed in an automated fashion by the so called plugIT semantic kernel. However, the semantic kernel compares graphical models representing the HPC resources but considering the availability and current workload of the selected computing resources only in a limited degree. In the scope of the plugIT project and certain business scenarios the plugIT approach for allocation of resources fits perfect to the needs. The use of several SLAs with various priority for scheduling jobs on the selected HPC resources improves current efforts for resource allocation on HPC infrastructure. Therefore, the SLA approach developed in the plugIT project might be used as a well fitting foundation for allocation of computing resources for ontology matching tasks.
Through the use of the SLA based plugIT [131] approach the process of validation and allocation of HPC resources that is done by the IT expert is enhanced in a way that the IT expert has only to approve the recommendations from the IT socket. The general idea about the IT socket is to support an end user, that has the role of an project applicant (table 3.1), by offering the Online Proposal Submission (OPS) [133] application to the end user that is performing an online proposal submission. The OPS application supplies a form to the project applicant that requests all necessary information in order make an automatically assumption of about the HPC configuration required for

the scenario.
After receiving the recommendations from the IT socket the IT expert, that has the role of the project approver (table 3.1), validates the recommendations and sends the a notification with an recommendation from the IT socket to the project applicant. The recommendation from the IT socket is a service level agreement (SLA). For this work the used schema for the SLA is the Web Service Agreement (WSAG) specification [134]. In order to deal with the specific requirements of HPC additional elements are necessary that are covered by the WS-Agreement schema for HPC (High Performance Computing - Web Service Agreement (HPC-WSAG) [135]). This means that the recommendation proposed to the project approver is an SLA offer based on the HPC-WSAG schema.
However, the IT socket requires information from the HPC site for making an assumption about the most fitting resource configuration to offer based on the input done by the project applicant.

Role	**Person in charge**	**Task**
Project applicant	End user	The project applicant is the end user that is in the need of HPC resources that fit to the use case scenario.
Project approver	IT expert	The project approver is the IT expert that validates the recommendations of the IT socket.
Infrastructure modeler	IT expert	The infrastructure modeler is also an IT expert that knows about the HPC infrastructure in order to create graphical models for the HPC resources each time the HPC infrastructure

Table 3.1: Role Model for the Resource Allocation

The IT infrastructure, the SLAs and the criteria for the SLAs are described as graphical models in an online repository that is accessible by the IT socket. These models are created by an IT expert, the infrastructure modeler (table 3.1), that has very detailed knowledge about the existing HPC environment and useful SLAs describing the available computing resources. The benefit of this approach is that the models are created easily by the infrastructure modeler once in case new HPC resources are available or if the current hardware changes. Through this, the knowledge about the HPC infrastructure is not only accessible to one specific IT expert, it is mapped in the graphical models that are stored in the online repository. Based on this online repository the IT socket receives the required information for producing SLA offers automatically.
As it is presented in table 3.1, three roles of persons involved in this process are necessary: the project applicant, the project approver and the infrastructure modeler. With regard to the IT socket, the functionality for creating SLA offers out of the models

from the online repository and the input of the project applicant comes from the semantic kernel component of the IT socket. The semantic kernel transforms the input from the project applicant into ontologies in order to compare them with the models of the SLAs, SLA criteria and the IT infrastructure that have been transformed to a Model Ontology (MO). Further, a Domain Ontology (DO) for HPC is used to support the transformation to the MOs. This means, the functionality of the semantic kernel is twofold:

1. Transformation of project applicant input and models into ontologies and MOs.

2. Comparison of ontologies.

The recommendation of SLA offers to the project applicant automates the process of resource allocation and makes it more efficient compared to the manual workflow for resource allocation of HPC resources. The ontology matching is performed in order to make use of the semantic information provided by the ontologies.

PROMPT PROMPT is a matching system dealing with heterogeneity described by Noy and Musen [136], it is based on an algorithm for semi-automated ontology merging and alignment. The PROMPT system supports an expert with results based on the ontology merging, taking into account inconsistencies by making suggestions on how to deal with such inconsistencies.

Protégé Protégé [137] is a platform supporting the modeling of ontologies by providing ontology editors and different formats such as RDF, RDFS, OWL and XML schema. Protégé is based on Java and can be used as a knowledge base framework. Besides the functionalities for dealing with ontologies in various formats it provides a GUI.

RiMOM RiMON (Risk Minimization based Ontology Mapping) [138] is a tool for generation ontology alignments by combining different strategies such as alignment identification by semantic matching of entities in different ontologies. The semantic matching combines a textual and a structural matching of the given ontologies.

Similarity Flooding Similarity flooding, analyzed by Melnik et al. [139], is a system that deals with a graph based method to treat ontologies. Similarity flooding assumes that the comparison of entities depends on the connection to the neighboring entities by focusing on directed labeled graphs with the aim to analyze such neighboring entities. However, in case that ontologies are not expressed as directed labeled graphs

the similarity flooding approach does not produce beneficial results anymore. Furthermore, it requires a similar structure of the graphs in order to perform a useful matching.

SPIDER SPIDER (Schema mapPIng DEbuggeR) is a prototypical tool for debugging schema mappings as stated out by [140]. For debugging schemes it produces routes describing relationships between source and target data by considering the given mapping schema. The route engine of SPIDER produces one or all routes of the mapping.

BabelNet The Databases BabelNet 2.0, EuroWorldNet, WordNet 2.0 and GermaNet are a result of the combination of a dictionary and an encyclopedia called Thesaurus. It contains a huge number of relation and synonyms between words. BabelNet is a multilingual encyclopedic dictionary and semantic network providing a stand-alone resource with its Java API, a SPARQL endpoint and a Linked Data interface as part of the Linguistic Linked Open Data (LLOD) cloud [141]. Especially BabelNet 2.0 [142] and EuroWorldNet are capable to interconnect words within various languages and different meanings. The whole BabelNet 2.0 thesaurus can be downloaded on the homepage. The file works with Java, needs about 40 GB of disk space and is used with the LemonAPI for Eclipse.

GATE GATE (general architecture for text engineering) [143] is an open source software capable of solving text processing problems. It is used by a broad community of developers, users, educators, students and scientists. It focuses on text processing workflows by applying a repeatable process.
GATE includes ANNIE (A Nearly-New Information Extraction System) an information extraction system. ANNIE is a set of modules comprising a tokenizer, a gazetteer, a sentence splitter, a part of speech tagger, a named entities transducer and a coreference tagger.

Treetagger Treetagger [144] is used to divide sentences and to match its parts to single word classes. Mostly, Treetagger is used as a part of Gate or Text2Onto, it annotates text with part-of-speech and lemma information.

Text2Onto Text2Onto [145] is an extension of the Gate Project. It seems to be one of most promising programs, which are able to build up an entire RDFS or OWL model automatically from a natural text. While extracting natural texts, Text2Onto uses the following three programs:

- WordNet - a lexical database for englisch, [146]
- Treetagger - a language independent part-of-speech tagger, 3.2
- GATE - general architecture for text engineering, 3.2

3.3 Advantages and Disadvantages

The comparison of tools and frameworks (see 3.2) has shown that lot's of tools for extracting knowledge from texts and manipulating ontologies are usable as well as frameworks providing APIs for developers. In general, presented matching and extraction tools are following strategies for combining several matching approaches, by consideration of lexical similarities as well as comparing relations to neighboring entities and exposing the type relation type. In addition, another common approach is the verification of the matching results by consulting a source of evidence, e.g. a specific domain ontology, the WWW or manually involving human knowledge. Most matching approaches distinguish between a source and a target ontology with the aim to extend a target ontology. However, some cases only specify alignments, created without updating a complete ontology with produced matching results. The main goal of ontology matching methods is the maintenance and extraction of required knowledge from a set of ontologies by matching and generating facts. For this, similarities are analyzed and in most cases a value presenting the grade of similarity is created. In addition, the analyzes of the previously presented tools has shown, most matching systems produce a high number of correct results but also a certain amount of incorrect results. The incorrect results often occur by matching incomplete or ambiguous data. Additionally, the different structure and different ontology language can produce incorrectnesses. Simply another language might be also a cause for wrong results. Thus, a verification is highly recommended to reduce the amount of incorrect results.
Common ontology matching tools compare the selected ontologies by analyzing the given entities graph wise. Thus, a graph wise matching is beneficial but in case of high amounts of data, as for instance the matching with data from the WWW, most matching approaches produce insufficient delays due to a lack of performance and efficiency of the matching procedure. Dividing the matching task into several subtasks is a strategy to handle matching procedures with high data volumes. However, the separation of related data implies the problem of taking care of the relations of these entities in case of storing them in separated data blocks. A common proceeding is the relation storing as a list or table. Nevertheless, this proceeding causes additional effort in terms of time and complexity for each time a relation needs to be checked.

3.4 Relevant Approaches

When thinking of beneficial ontology matching approaches which are related to the considered use case, the following approaches are of high interest.

3.4.1 Information Retrieval Approach

As described ontology matching approaches are relevant to the research field of ontology matching. As presented in 3.1.1, Su et al. [6] are providing a definition for ontology mapping in terms of matching of two different ontologies, a target and a source. This approach is used by Calvanese et al. [101] and Lembo et al. [58] when presenting a method for schema integration. In this described approach a distinction between the target and multiple source models on a conceptual layer is proposed. An information retrieval based proceeding deals with extracting information from complex data sources, it is related to the field of data mining 3.1.3. Such a strategy is useful when thinking of gaining required information from a given data set or for result verification issues. To sum up, information retrieval approaches are focused on complex data sources and the extraction of relevant data from those.

3.4.2 Vector Based Word Space Approaches and Online Ontologies

The vector based matching approaches are taking into account methods such as random indexing 3.1.2. These methods are usable for generating a matrix regarding an entity occurrence in a text for making assumptions regarding the meaning of an entity. Sahlgren [147] describes this proceeding in his doctoral thesis and underlines the usability of word space focused approaches to extract semantic knowledge from usage data. In addition, such a proceeding is beneficial for verification issues of produced matching results or for applying additional information for a given entity in an ontology. However, the meaning of an entity and derivations of such a meaning might be different in case texts from different domains are used as Mihalcea et al. [148] emphasize when performing tests for verification of semantic meanings based on a text corpora. This affects produced results negatively in case the related domain does not fit to the given context. Hence, the domain selection of the used texts is a crucial issue. Besides the mentioned texts, online ontologies found in the WWW are usable for vector based word space approaches as well for verification of produced results and supporting required domain knowledge. Generally, vector based word space approaches including online ontologies make use of data structures. Nevertheless, data structures are usable for verification of single entities of ontologies.

3.4.3 Probability Measurement

The matching of entities in a given set of ontologies is often based on similarity measurements as stated out in 3.2. For this, a similarity can be expressed by a certain value calculated out of different matching methods as presented by Pilehvar et al. [149] or Snow et al. [150] who is focusing on measuring sense similarity. For performing those measurements several methods such as a matching of lexical similarities, relations or types of relations or the structure of an ontology are promising approaches. In addition, a vector based word space measurement as presented in 3.4.2 those methods are usable supplementary for verification purposes or contributing to the similarity value calculation. However, the more matching strategies are applied the more computational effort is required in terms of calculating the matching results and performing a useful verification of the results. In addition, a threshold is required defining a limit for the similarity value in order to verify if a produced result is considered as trustworthy. The trustworthiness of the produced results is crucial for providing reliable results.
When thinking of the probability measurement approach, it makes use of several data structures. A lexical matching of entities can be applied to single terms but calculating similarities of relations requires linked data sets such as ontologies.

3.4.4 Graph Analysis

When thinking of graph analysis approaches the comparison of graph paths becomes an issue for ontology matching strategies. Hu et al. [151] present an ontology matching system supporting graph based approaches by adapting functionalists for handing RDF ontologies used as RDF graphs. The use of the RDF standard enables the presentation of an ontology as graph due to the fact that entities in the ontologies are linked and in most cases such linking is labeled. Thus, RDF based ontologies can partially be presented as graphs with labeled edges. In addition, Husain et al. [152] introduce a framework for answering SPARQL queries based on RDF graphs. Nevertheless, the comparison of similarities should consider the similarity grade of the compared graph paths. However, this proceeding requires the definition of a search space for limitation of the paths that are snippets of the complete ontologies or the complete graphs will be parsed. Additionally, the matching of the graphs identifies the similarity of the structure of the relationships between the concepts of the ontologies that are used as nodes but the features and values are not considered when only thinking of the similarity grade of the path structure. Nevertheless, the use of such a graph analysis approach seams to be beneficial for the proposed strategy.
However, generally graph analyzing strategies provide methods for comparison of graphs and included nodes and edges. The matching of ontologies as graphs leads to the field of graph matching theory. The structure of a sub-graph of one of the selected graphs is mapped within the structure of the sub-graph of another graph and vice

versa or, for instance, a full graph matching is possible but computational expensive and time consuming depending on the size of a graph. Steps for using graph analysis methods regarding an ontology matching are included in current research as well as presented by Shvaiko et al. [153]. However, in general graph analysis methods are computationally intense which indicates that needed computing resources have to be available and that used algorithms for performing a graph matching have to be developed in an effective fashion. Furthermore, among methods for graph analysis a selection of useful graph matching strategies is required, for instance dealing with graph isomorphism. Currently, graph isomorphism is an unsolved problem for graph analysis in terms of effectiveness cause of the fact that until now no efficient graph analysis algorithm for graph isomorphism has been developed. Graph isomorphism is still an unsolved problem in the research field of computational complexity theory as exposed by Kobler et al. [154]. An issue, such as graph isomorphism, can be solved by involving a human expert in the ontology selection phase and in addition, perform a sub-graph based matching of selected ontology parts.
Generally, graph based matching methods refer to ontologies such as RDF graphs. For those, methods from the field of graph matching can be applied.

3.4.5 Distributed Ontologies

As presented in section 3.1.9, the amount of available ontologies is significantly growing as well as the amount of available data in the web. Hence, in order to keep the execution time of a reasoning task low, a set of distributed ontologies will be considered, in order to provide a matching of such ontologies for providing necessary data. Fan et al. [155] and Flahive et al. [156] describe the necessity of handling large data sets being ontologies through using distribution methods. Such distributing as well as previously presented matching methods will be used for ensuring an effective matching enabling reasoning based on a well prepared knowledge base.
Following the idea of this work, a set of ontologies is matched for producing alignment lists based on a priority ontology used for a specific reasoning task. The distribution focuses on the selection of ontologies distributed at different locations and ontologies locally available. Such ontologies are used for the matching procedure in order to generate similarity values by use of adequate matching strategies presented in the next chapter 4. The similarity value is used to make assumptions regarding the matched entities of the distributed ontologies. Out of the produced similarity value and the assumptions, a set of alignments will be generated based on those selected ontologies. The alignments are used for updating the priority ontology for providing a customized knowledge base. However, the execution times of such matching procedures need to be evaluated. Currently considered ontology matching approaches need to deal with large data sets (see chapter 2). Performing a matching task on such data in order to provide semantic data for a reasoning task can lead to performance and scalability problems. Thus, a distribution of the performed matching tasks for

increasing the overall performance will be elaborated in chapter 5.

Chapter 4

Ontology Matching

4.1 Improvement for Ontology Matching

In the previous section (see 3) various ontology matching methods and related approaches were described. When thinking of creating new facts as known from the ontology lifecycle (see 1.1) and indicated in figure 4.1. The matching of similarity values is a gainful approach for semi-automatization of a complete matching process. Hence, the matching approach used in this work is based on the elaboration of similarity values between ontology entities. In addition, ontologies can be handled as graphs when thinking of ontology structures. The required prerequisites have to be considered as defined in the following in the two main phases of a similarity matching.

Preparation for the similarity matching

1. Identification of relevant ontologies: *The required ontologies are defined from a known set related to the research field of interest. The preselection is a manual step done by the expert.*

2. Selection of relevant features of entities: *The entities are identified by considering the selected ontologies. The amount of matching iterations depends on the data set size. For each selected entity of the priority ontology a matching iteration is performed.*

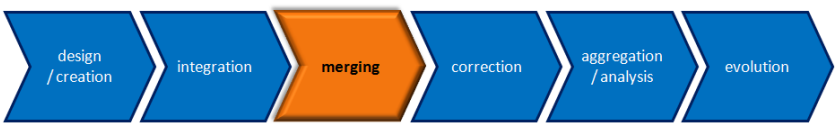

Figure 4.1: Ontology Lifecycle: Merging

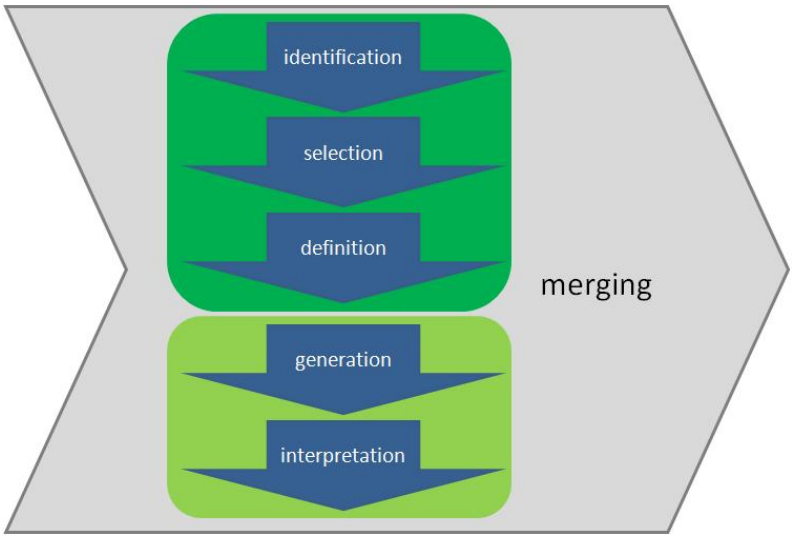

Figure 4.2: Similarity Matching Phases

3. Defining the search space: *The search space defines the number of neighboring entities which are matched. For instance, the relation of a single entity to another might be different in another ontology.*

Execution of the similarity matching

1. Generating the similarity value: *The similarity value is created out of a number of different values. The different values are generated from different similarity matching approaches such as features of the concepts and relations to the neighboring concepts and the structure of the ontologies by exploring the type of relations between the entities.*

2. Interpretation of generated similarity: *The similarity value is used for creating alignments out of the matched entities. A high similarity value leads to a high probability of similarity. This probability of similarity expresses the grade of similarity between matched entities.*

The two main phases are divided into sub-phases as presented in figure 4.2.

4.1.1 Identification

For finding relevant ontologies fitting to a domain of interest an ontology classification can support decision making about best preferred ontologies. Hence, the first step is highlighting every relevant ontology. Further, schema and structure of selected ontologies need to be similar as presented in 3.1.7 for the instance matching based approach. This indicates the ontology schema as relevant for successful matching

procedures. In the second step a decision is done based on the relevance level of an ontology.

1. Defining the Relevance: Considering *a)* the content, *b)* the schema and **c** the structure.

2. Ontology Rejection: Decision regarding the content, the schema and the structure.

The definition of relevance is categorized in three level of relevance assuming that an ontology with a not usable content will not be used or it needs to be translated into a fitting schema.

- LR Ontology - Low Relevance Ontology (LR)
- MR Ontology - Medium Relevance Ontology (MR)
- HR Ontology - High Relevance Ontology (HR)

For every level of relevance a detailed differentiation of the level of relevance is possible. The differentiation indicates a more detailed few regarding the significance of an ontology expressed with three different categories, LR, MR and HR.

1. *LR Ontology* {*1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9*}

2. *MR Ontology* {*2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9*}

3. *HR Ontology* {*3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9*}

The selected ontologies and the level of relevance is stored in a Database (DB) so that a repository is created. The ontology repository is accessible by the end user who is enabled to perform a selection of significant ontologies based on the ontology repository. Hereby, the reusability of the ontologies is guaranteed by storing URLs of the ontologies and the relevance level including the category of relevance. In case the end user is in the need of performing a similar matching of ontologies again he is supported with the ontologies and the related level of relevance. Figure 4.3 refers to the data model of the ontology repository DB that stores the ontologies and the level of relevance of an ontology. At this, each relevance is related to the specific matching topic. This means, that the same ontology information might be stored up to three times but in a different tables and with a different link to another matching topic:

1. *Ontology X* is stored in the *low_relevance_ontology* table and linked to *matching_topic* A

2. *Ontology X* is stored in the *medium_relevance_ontology* table and linked to *match-*

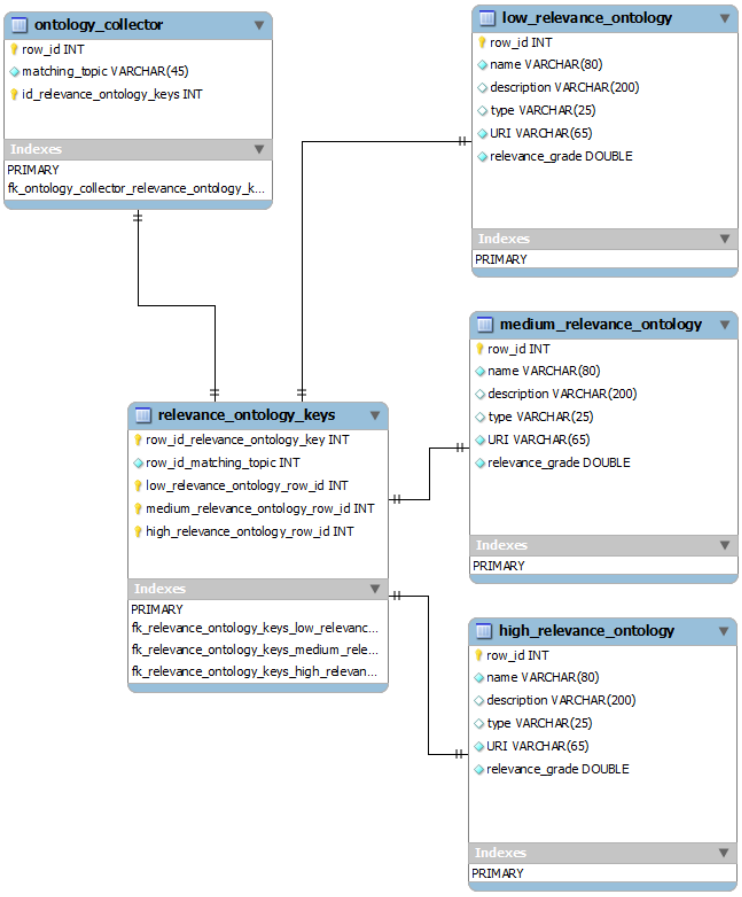

Figure 4.3: Ontology Repository DB

ing_topic B

3. *Ontology X* is stored in the *high_relevance_ontology* table and linked to *matching_topic* C

The *ontology_collector* is the entry point for storing an ontology. It stores the domain of interest and it is related to the unique Identifier (ID) of the ontologies and the domain by use of the *relevance_ontology_keys* table. The *relevance_ontology_keys* table stores the IDs and the ID of a related ontology stored in another table according to its relevancy regarding the selected matching topic. The ontologies are stored either in *low_relevance_ontology*, *medium_relevance_ontology* or *high_relevance_ontology*.
The Ontology Repository Database (OntoRepDB) is created based on MySQL Server (e.g. version 5.5.12). The implemented database as well as the implemented methods for manipulating the database support functions for *creating*, *reading*, *updating* and *deleting* (*CRUD*) of data in the database being implemented as Java methods. The selected methods are used for data manipulation of the *ontology_collector* table and each of the methods is either used for *creating*, *reading*, *updating* or *deleting* data.
The description of the *OntoRepDB* is ordered by consideration of the tables and differ-

ent types of data.

- **ontology_collector** The ontology collector table contains different types of matching domains, e.g. automotive data, linked web data or legal rules. The classification between different domains is useful because of the different types of used ontologies. More specifically, this means that each ontology stored in the OntoRepDB has a different grade of relevance for each matching domain.
  - *row_id* This is the id of the current record.
  - *matching_topic* This is the matching domain such as a HPC ontology.
  - *id_relevance_ontology_keys* This id is used in the next table to allocate the matching topic to relevant ontologies.
- **relevance_ontology_keys** The relevance ontology keys table stores the relations between the matching domains and the allocated ontologies.
  - *row_id_relevance_ontology_key* This is the id of the current record.
  - *row_id_matching_topic* This is the id of a selected record from the matching domains.
  - *low_relevance_ontology_row_id* In case an assigned matching domain is allocated to a low relevance ontology, the id of the low relevance ontology is stored.
  - *medium_relevance_ontology_row_id* In case an assigned matching domain is allocated to a medium relevance ontology, the id of the medium relevance ontology is stored.
  - *high_relevance_ontology_row_id* In case an assigned matching topic is allocated to a high relevance ontology, the id of the high relevance ontology is stored.
- **low_relevance_ontology** The low relevance ontology stores the ontologies that are rated with a low relevance to a specific matching topic.
  - *row_id* This is the id of the current record.
  - *name* This is the name of the selected ontology.
  - *description* This is a description about the content of the ontology. It is optional but recommended in order to support the end user.

  - *type* This is a description about the used ontology language of the ontology such as, for instance, OWL. It is required because of the *transformation* method, explained in the following lines of this section.

  - *URI* The URI points to the location where the ontology is stored. This might be a pointer to a WWW location or a local storage device.

  - *relevance_grade* The relevance grade is a detailed statement regarding the overall relevance of the selected ontology to a matching topic. The values are identical to the values for the LR, MR and HR Ontologies.

- **medium_relevance_ontology** This table is similar to the *low_relevance_ontology* but the overall relevance for a specific matching topic of the table *ontology_collector* is rated with medium.

- **high_relevance_ontology** This table is similar to the *low_relevance_ontology* but the overall relevance for a specific matching topic of the table *ontology_collector* is rated with high.

Besides the already mentioned selection of useful ontologies, it is also necessary to take a closer look at the type and the structure of the ontology. Different ontologies might be constructed in different ontology languages. Very common ontology languages are RDF and, for instance, OWL built on top of RDF. However, even if the used ontology language is similar, the created ontology might be presented in a different way then another. The presentation of the ontology depends on the used method for reading and / or writing the OWL or RDF file. For instance, the presented Jena Framework (section 3.2) offers various notation for reading and writing RDF and OWL files. This issue leads to the conclusion that it is necessary to use ontologies with same structure using a single binding ontology language. Through this, it becomes possible to secure a stable ontology matching without mismatches. However, a commitment allowing only a single type of ontology with a defined structure limits the amount of eligible ontologies. Using a *Transformer* creating ontologies with same notation out of a set of heterogeneous ontologies deals with this issue.
As already presented the *OntoRepDB* contains relevant information regarding the used ontologies. This includes the *type* row of the *low_relevance_ontology* table, the *medium_relevance_ontology* table and the *high_relevance_ontology* table. The *type* row in the tables of the database identifies the used ontology language. This information is required by the *Transformer* that aims at constructing ontologies with same structure and same notation. The *type* row indicates indicates the used ontology language for the *Transformer* that makes use of its rules for translating the selected source ontology into the transformed source ontology. The main functionality of the *Transformer* is twofold:

1. (a) Identify the used ontology language by means of the *type* row in the *On-*

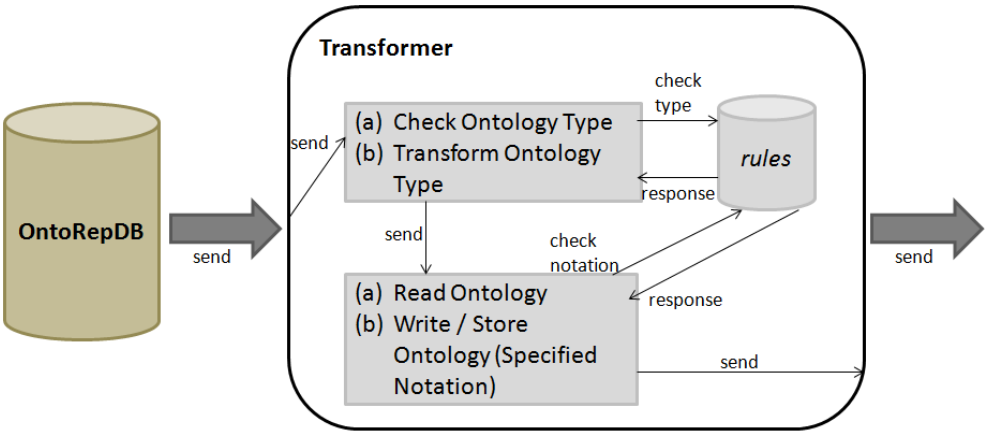

Figure 4.4: The Transformer

toRepDB and (b) perform a transformation into another type in the the case that the *Transformer* rules claim this.

2. (a) Read the source ontology and (b) write and store it in a notation specified by the *Transformer* rules.

The Transformer is presented in figure 4.4; it provides transforming functions for the ontology types and structures based on a set of rules. The rules are based on the assumption that in case no transformation guideline is available for a selected ontology, the selected ontology will be rejected. This default assumption is required for error prediction to ensure a matching with only allowed and valid ontology types.
The described DB includes significant ontologies supporting an end user by offering him a set of ontologies and pointing out the level of relevance for his requirements. This is described in the selection phase. Furthermore, the ontology types and structures are aligned by the *Transformer*.

4.1.2 Selection

The next phase is the selection being similar to the identification phase but concentrating on the ontology entities for validating the content. The selection of entities is done by the end user knowing best about the relevancy of available entities. Therefore, the end user has to specify requirements regarding the entities. At this step, human involvement in the ontology matching process is required. However, human involvement interrupts the complete ontology matching process. However, the selection of adequate ontologies is supported by the ontology repository being a database maintaining links to ontologies and domain descriptions of the ontologies.
Using all available ontologies instead of making a selection causes two critical issues:

1. Data overload through establishing a confusingly broad set of data

2. Incorrect assumptions caused by simply using all available data despite from their relevance

A data overload happens in case a huge set of ontologies or even a few very large ontologies are used during the matching process. However, in case the ontology repository contains only a small set of ontologies, the data overload might not be a major issue but there is still the issue of producing incorrect assumptions by using every available ontology without validating the need of ontologies and without a review of the content by checking the entities of the included data in the selected ontology. The use of an ontology for the matching process, content wise not adequate for the current matching domain, might lead to wrong assumptions used for updating the priority ontology. For instance, if the ontology topic is different, entity ambiguities might cause a comparison of entities without being aware that the meaning of the entities is different an used in another context. Of course, the matching procedure has to deal with entity ambiguities by using methods for validating meaning of entities and reviewing the context of the compared entities but a bad selection of ontologies in this step, such as just selecting all available ontologies, makes the whole matching process much more error prone. Hence, a good selection of ontologies reduces the source of errors and is therefore a necessary step.
For reducing needed time for the selection phase, it supports a history repository storing the domain of a matching process and the selected ontologies. In case, another matching iteration related to the same domain is performed, the previously used ontologies are recommended by use of the history repository.
The described selection phase is a key phase for the reliability of the produced matching results. Therefore, the end user needs to have knowledge about selected ontologies and the selection history needs to be integrated into the selection phase. The selection history is included in the ontology repository database. However, the end user has the possibility to access the database and update it whenever it is necessary. Through this, the end user is supported by using the ontology database including the relevance level of ontologies for a specific domain and by the history presenting previously performed selection iterations.

4.1.3 Definition

When the selection is done the last step to perform in the preparation phase is the definition of the search space. The search space is the part of the ontology selected for the matching. For instance, if the definition allows a maximum number of X neighboring elements this means that for each matching iteration the neighboring entities are considered up to a neighboring level of X.
When thinking of the search of an ontology as said the depth level for comparisons

needs to be defined whereby the maximum depth of the search space is the maximum number of concepts of the selected ontology and the minimum depth is one. In case the search depth is one, the current concept focused by the matching procedure is compared with only a single concept from the comparing ontology. However, in this case the number of concepts defines the number of concepts for different distances of concepts, e.g.

1. $ConceptA -> ConceptB -> ConceptC$ has a search space of 2.

2. $ConceptA -> ConceptB_1, ConceptB_2 -> ConceptC$ has a search space of 2.

Item one and item two have both a search depth of two, although first example contains three and second example contains four concepts.

$Search_{SpaceX} = MAX_{ConceptLevel}$ - ($MAX_{ConceptLevel}$ - X)

- $Search_{SpaceX}$ defines the search space that is considered during the matching procedure. The search depth is expressed by a number of concepts whereby the number of concepts describes the distances between the concepts, e.g. an ontology might contain Y concepts but the value of $MAX_{ConceptLevel}$ is less than Y.

- $MAX_{ConceptLevel}$ defines the maximum number of concept distances of an ontology. Each additional distance increases the $MAX_{ConceptLevel}$ by one whereby the minimum distance is always one. A distance of one means that only one selected concept is considered without validating relations to other concepts.

- X defines the amount of concept distances that are used for the matching procedure.

In case of using a full ontology, the value of $Search_{SpaceX}$ is the maximum number of concept distances. But, in case of large data sets and a time restriction the matching of all concepts might lead to a data overload and further the matching might lead to a matching of concepts obviously not relevant. Therefore, in order to minimize the required time and effort for computing and to use only relevant concepts for the matching procedure it is quite important to define an adequate search space.
More detailed, the maximum number of concepts of an ontology is different than the maximum number of concept distances. The maximum number of concepts of an ontology expresses all concepts contained in an ontology whereas the maximum number of concept distances expresses the longest possible distance within an ontology. If the search space is defined by the maximum of the concept distance this causes a comparison of every entity and every relation of the selected ontology.

4.1.4 Generation

The generation of the similarity value is a key benefit of the presented ontology matching approach. Through the similarity value a matching indicator is created making use of similar entities of another ontology. Hereby, it is important to define the matching indicator as accurate as possible. As Euzenat et al. [157] described, using a non absolute reliable similarity value as matching indicator requires an expert for monitoring the proposed matching results and mappings between matched entities:

> Nevertheless, the computed similarities suggest possible mappings of entities, hence they can support an alignment. One way of doing this consists in displaying the entity pairs with their similarity scores and/or ranks and leaving the choice of the appropriate pairs up to the user of the alignment tool. One could go a step further and attempt at defining algorithms that automate alignment extraction from similarity scores.

However, the aim of this work is to create an alignment list produced by the matching results. As presented in previous chapters (see 3), the ontology matching approach considers several aspects such as name of concepts, features, values of the features and the relations between the concepts. However, Farooq et al. [158] identify three different levels of measurement aspects for creating a meaningful value for similarity between semantic entities in an ontology. Here Farooq et al. are defining the *primary similarity* (1^{st} level), the *taxonomic similarity* (2^{nd} level) and the *non-taxonomic similarity* (3^{rd} level). During the 1^{st} level a lexical matching between concept names is performed, for the 2^{nd} level the parent concepts are compared to identify similarities and for the 3^{rd} level the role of the matched concepts is compared.
The ontology matching approach of this work performs a matching based on lexical matching and relations between concepts. The structure of the ontologies is considered in the previous steps in the preparation phase. The matching of the relations is a taxonomy matching when thinking of a matching between neighbored concepts. Furthermore, it is a non-taxonomy matching in case the role of a concept is evaluated by considering the label of a relation such as a *is-a* or *has-a* label.
Additionally the used matching strategy is based on graph analysis approaches. When thinking of an ontology as a graph the following distinction is valid:

- Each ontology is a graph and the first concept is the starting node.
- Each concept is a node that is described more detailed with features and values of the features.
- Each relation is a labeled edge.

The definition above will be used for adapting graph matching approaches to the matching of the ontologies in order to create the similarity value. However, the use of

graph based matching approaches faces the challenges of matching ontologies with a different structure as well as the fact that graph based matching methods are often time and resource consuming as it is figured out by Mao et al. [10]. Although, for this work there are four steps of matching that have to been taken into consideration.

1. The nodes are compared by matching the names of the nodes (the comparison is not case sensitive).

2. The names of the features of the nodes are compared (the comparison is not case sensitive).

3. The values of the features are compared (the comparison is not case sensitive).

4. The labeled edges between the nodes are compared by verifying the label of the edge and the nodes that are connected due to the labeled edge.

However, the four stages of matching are only performed if they are not in contrast to the settings of the selection or the definition phase. Thus the foundation of the similarity value is the matching by use of lexical, taxonomy and a non-taxonomy matching strategies and the utilization of graph analysis strategies. The similarity value is first created for the lexical matching of the concept, feature and value of a feature. Afterwards the similarity value is created for the taxonomy matching of the neighboring entities such as parent, sibling or child concept. The last step is to generate the similarity value for the non-taxonomic matching for a concept as an indicator that specifies the grade of similarity by considering the labels of the edges in order to identify the role of a concept. Thus three similarity values are created.

1. Lexical Similarity Value (LSV)

2. Taxonomy Similarity Value (TSV)

3. Non-Taxonomy Similarity Value (NTSV)

The LSV method is the lexical comparison of entities such as the name of a concept, the name of a feature and the value of a feature. Out of this matching a similarity value is created for each concept that is compared with another concept from the comparison ontology. This means, that for each concept of the source ontology one LSV similarity value is stored as well as the name of the concept of the comparison ontology. The notation for the LSV similarity is a tuple whereas the first element is the created similarity value, the second element is the name of the concept of the comparison ontology and the third element is the name of the comparison ontology itself.

$$< LSV_{SIM}, Concept, Ontology >$$

The TSV method is the taxonomy comparison that covers the comparison of a specific path from the source ontology with a specific path from the comparison ontology in order to make an assumption about the grade of similarity between neighbored concepts. The assumption regarding the similarity of neighbored concepts is based on the lexical similarity from the LSV method. Further, the maximum path length is restricted by the search space. The notation for the TSV similarity is a tuple whereas the first element is the created similarity value, the second element is an array that contains all names of all concepts in the path whereas the number of concepts in the path is max, the third element is the name of the comparison ontology and the fourth element is the unique name of the path.

$$< TSV_{SIM}, Concept[searchspace], Ontology, Path >$$

The $NTSV$ approach is based on the TSV method. However, in this step the similarity result from the TSV method is modified by including the labels of the edges between the concepts. The notation for the $NTSV$ similarity is a tuple whereas, compared with the TSV tuple, only the first element is updated.

$$< NTSV_{SIM}, Relations[searchspace], Ontology, Path >$$

All three similarity values are used to make an assumption about the general grade of similarity between matched entities. However, regarding the taxonomic matching graph based matching approaches such as the matching of different paths will be considered for this work. Furthermore, the end user has the possibility to make a selection out of the presented matching methods. Because of the dependencies between the different matching methods three different configuration settings are allowed, namely the $\{LSV\}$ or the $\{LSV, TSV\}$ or the $\{LSV, TSV, NTSV\}$ configuration setting. Figure 4.5 gives an overview of the three used matching procedures. As presented, after each step the results are stored. Such results are the produced alignments including the similarity values of the current matching step. Whereas the lexical and the taxonomy matching procedures make use of the the source and the target ontologies, the non-taxonomy matching procedure uses the alignment lists produced by the taxonomy matching procedure as input for matching them with the source ontology. After each performed matching procedure, an expert has the choice to end or proceed the next matching procedures.

4.1.4.1 Lexical Matching: LSV

When thinking of a lexical matching the selection of entities for comparison is crucial. The question is if only concepts are matched or rather features and values of the features are compared. A simple comparison of concepts will not offer the possibility to deal with different notations for same entities due to the issue that a different spelling will mean that the entity is different. This issue will be handled by the taxonomy

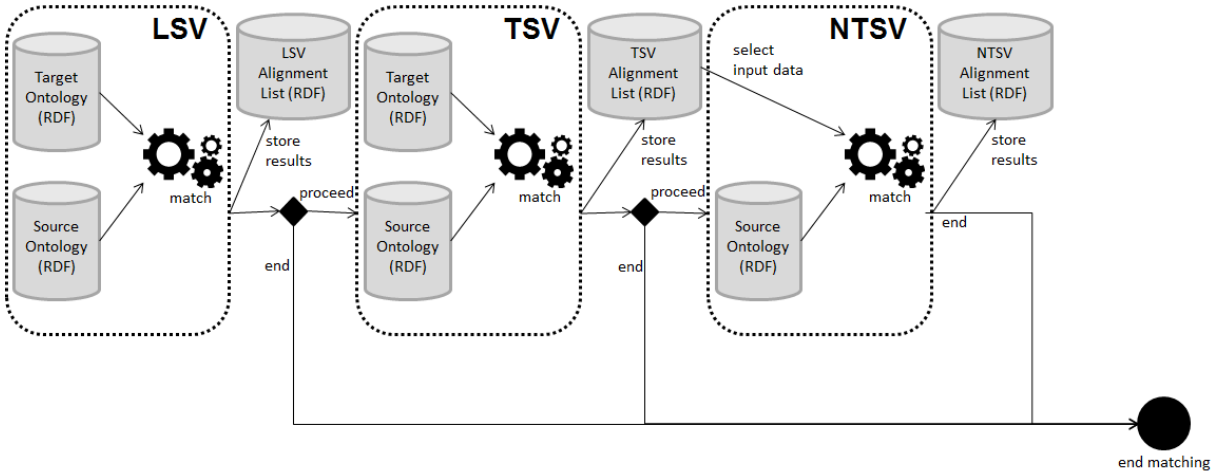

Figure 4.5: The Lexical, Taxonomy and Non-Taxonomy Matching Procedures

(4.1.4.2) and the non-taxonomy (4.1.4.3) matching. However, it becomes possible to tackle this issue even in a lexical matching procedure by involving a comparison between features and values of the features. For this, first a lexical matching approach by use of RDFS and second of OWL is proposed.
For RDF, a set of classes and of properties is available enhanced by RDF Schema syntax. The following table *RDF classes* taken from the World Wide Web Consortium (W3C) Recommendation [159] presents the classes in RDF that need to be considered for a comparison.

Class name	**comment**
rdfs:Resource	The class resource, everything.
rdfs:Literal	The class of literal values, e.g. textual strings and integers.
rdf:XMLLiteral	The class of XML literals values.
rdfs:Class	The class of classes.
rdf:Property	The class of RDF properties.
rdfs:Datatype	The class of RDF data types.
rdf:Statement	The class of RDF statements.
rdf:Bag	The class of unordered containers.
rdf:Seq	The class of ordered containers.
rdf:Alt	The class of containers of alternatives.
rdfs:Container	The class of RDF containers.
rdfs:ContainerMembershipProperty	The class of container membership properties, rdf:_1, rdf:_2, ..., all of which are subproperties of 'member'.
rdf:List	The class of RDF Lists.

Table 4.1: RDF classes

A simple matching of notations is performed by searching RDF files for a specific notation and validate if same notation is available in another RDF file. In order to make a more precise lexical matching the simple comparison of the classes is not sufficient. Hence, a comparison of RDF properties is performed. Thus the following table presents the features, RDF properties, by assigning them to the domain and the range. The table *RDF properties* is taken from the W3C recommendation website [159].

Property name	**comment**	**domain**	**range**
rdf:type	The subject is an instance of a class.	rdfs:Resource	rdfs:Class
rdfs:subClassOf	The subject is a subclass of a class.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	The subject is a subproperty of a property.	rdf:Property	rdf:Property
rdfs:domain	A domain of the subject property.	rdf:Property	rdfs:Class
rdfs:range	A range of the subject property.	rdf:Property	rdfs:Class
rdfs:label	A human-readable name for the subject.	rdfs:Resource	rdfs:Literal
rdfs:comment	A description of the subject resource.	rdfs:Resource	rdfs:Literal
rdfs:member	A member of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:first	The first item in the subject RDF list.	rdf:List	rdfs:Resource
rdf:rest	The rest of the subject RDF list after the first item.	rdf:List	rdf:List
rdfs:seeAlso	Further information about the subject resource.	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	The definition of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:value	Idiomatic property used for structured values.	rdfs:Resource	rdfs:Resource
rdf:subject	The subject of the subject RDF statement.	rdf:Resource	rdfs:Resource
rdf:predicate	The predicate of the subject RDF statement.	rdf:Resource	rdfs:Resource
rdf:object	The object of the subject RDF statement.	rdf:Resource	rdfs:Resource

Table 4.2: RDF properties

The RDF and RDFS elements described in table *RDF classes* and *RDF properties* are

used for the lexical matching. For this, first a concept for the matching procedure is identified and second, in case a matching concept was identified, the features of the concepts are compared. Lower and upper case notations are equal for the matching procedure. In general for every entity that is lexically compared the matching similarity is 1 or 0. A more fine grained distinction regarding the similarity value is achieved by taking into account the features of compared concepts. The similarity value for the lexical matching is generated by appropriating the single similarities of the features of a concept and dividing the aggregated feature similarities by the number of features multiplied by factor 2. Factor 2 is used because of the fact that not only the value of the features of a concept are used for the matching approach but the features of the selected concept as well. The similarity of the concept itself is not used for generation of the LSV similarity value because it is only 1 or 0. In case it is 0 no lexical matching takes place and in case it is 1, the lexical matching is performed.

$$LSV = AS \div (NF * 2) \tag{4.1}$$

- AS = Accumulation of Similarities (AS) of the selected concept.
- NF = Number of Features (NF) assigned to the selected concept.

As an example a RDF snippet is given, containing entities used for a lexical matching.
RDFS snippet:

```
<rdf:Property rdf:about="http://www.hlrs.de/2012/01/01-rdf-syntax-ns#type">
  <rdfs:isDefinedBy rdf:resource="http://www.hlrs.de/2012/01/01-rdf-syntax-
      ns#"/>
  <rdfs:label>type</rdfs:label>
  <rdfs:comment>description</rdfs:comment>
  <rdfs:range rdf:resource="http://www.hlrs.de/2012/01/01/rdf-schema#Class"
      />
  <rdfs:domain rdf:resource="http://www.hlrs.de/2012/01/01rdf-schema#
      Resource"/>
</rdf:Property>
```

Before performing the lexical matching, a similar concept of another ontology needs to be identified. For the given example the wanted concept is the RDF class property. In case this class is found in another ontology, the lexical matching takes place. The concept of the ontology is compared with another concept in order to find accordances of used RDF and RDFS terms. Afterwards, the values of the features, the RDFS properties, are compared to identify matches. In the given example the concept is the RDFS snippet describing the RDF class property. In case the lexical matching reaches a defined threshold, the result will be stored. Nevertheless, the usual RDF structure is presented in a triple $predicate, subject, object$. Such a triple provides information regarding the relation of the three entities *predicate*, *subject* and *object*, but in order to deal with more complex ontology structures containing more entities, and e.g. more properties related to a concept, OWL files are considered as well. Nevertheless, a lexical matching can be performed by use of RDF data. The following algorithm presents

a lexical matching by comparison of the lexical entities of RDF triples in pseudo code. Furthermore, the presented algorithm is extendable through adding more or other entities.

```
read RDFmodel1;
read RDFmodel2;
result_model := NULL;
WHILE RDFmodel1.hasNext(RDFmodel1.RDFstatement) DO
  int AS := 0;
  float LSV := 0;
  WHILE (RDFmodel2.hasNext(RDFmodel2.RDFstatement)) && (RDFmodel2.length <=
       textit{SearchSpace}) DO
    IF (RDFmodel1.RDFstatement.subject == RDFmodel2.RDFstatement.subject) {
       write(RDFmodel2.RDFstatement.subject,resultModel);
       AS++;
       }
    IF (RDFmodel1.RDFstatement.predicate == RDFmodel2.RDFstatement.
        predicate) {
       write(RDFmodel2.RDFstatement.predicate,resultModel);
       AS++;
       }
    IF (RDFmodel1.RDFstatement.object == RDFmodel2.RDFstatement.object) {
       write(RDFmodel2.RDFstatement.object,resultModel);
       AS++;
       }
    IF (count>0) {
       LSV := (float) AS / NF;
       } ELSE {
         LSV := NULL;
         }
  END
END
```

Such a matching is implemented prototypical by use of the Apache Jena™framework in order to compare RDF statements of two different RDF files. In general, this matching compares the lexical structure of two RDF models by dividing it into the RDF statements and compare each entity of the statement. However, this proceeding is extendable by adding more or other relevant entities. Each RDF statement of RDF model1 is compared with the RDF statements of RDF model2 until the search space is reached or there is no more RDF statement in RDF model2. The default value of the search space is the complete length of RDF model2. The similarity value is calculated by dividing the number of matches of each RDF statement by the number of checked entities of each RDF statement. In case the names of the matching features are matched as well, the *NF* need to be multiplied by factor two. The similarity values for each matching of RDF statements are usable for further calculations, e.g. identifying the overall similarity of the complete lexical matching procedure. The matching results are stored in a result model being an alignment list.
The presented lexical matching for RDF triples creates the *LSV* but the given formula could be much simpler due to the fact that the used RDF triple make use of an identical structure. Hence, a simple division of the *AS* divided by the factor three (the

number of elements included in a RDF triple) is sufficient in order to create the LSV. When performing the lexical matching by use of ontologies structured as OWL files, the above presented formula for identifying the LSV is required to deal even with concepts with different structure. In addition, generally OWL files contain more RDF elements than used in the RDF triple example but it is based on RDF. The focus of this approach will be RDF based ontologies.

4.1.4.2 Taxonomy Matching: TSV

The previously proposed lexical matching (4.1.4.1) is sufficient in case if similar concepts have the same notation. However, in case the notation of the concept names is different, another strategy is required in order to identify similar concepts with different names. In general, the lexical matching lacks of a sufficient solution to deal with similar entities with different notations. At this point the taxonomy matching offers a solution to handle similarities that are not identified by a lexical matching because of different notations. When considering the taxonomy in an ontology the neighboring entities, namely neighboring concepts, are relevant. Assuming an ontology is presented as a graph it is of high relevance to consider the parent nodes as well as the siblings. The number of parents and sibling nodes to be considered depends on the previously defined search space (see 4.1.3). The taxonomy matching procedure is triggered by an involved expert after the lexical matching was performed.
When thinking of RDF triples as a tree the structure is as follows: *¡subject, predicate, object¿*. The subject is the entity described by the predicate and the object whereby the predicate is the link to the object. The object is the last node of a branch or it is linked to one or more other objects which will make the object to the subject of another RDF triple. This is presented by the following example of an RDF taxonomy:

```
<subject_A, predicate_A, object_A>
<subject_B (previously object_A), predicate_B1, object_B>
<subject_B (previously object_A), predicate_B2, object_C>
<subject_C (previously object_B), predicate_C1, object_D>
<subject_C (previously object_B), predicate_C2, object_E>
<subject_D (previously object_C), predicate_D, object_F>
...
<subject_D (previously object_C), ...
...
<subject_E (previously object_D), ...
...
<subject_F (previously object_E), ...
...
```

The presented listing demonstrates the tree structure of an RDF based ontology by linking the subjects with objects through the predicates. For identifying similarities between such ontologies the given structure needs to be compared by consideration of the previously defined search space. The default value of the search space is the

complete ontology. Only in case, the expert wants to consider only partial structures of the selected ontologies it is recommended to change the default search space to a lower value. However, the aim is to update an alignment list in case a similarity is given. The similarity is identified by generating the TSV. The focus of the TSV generation is the linking of the subjects with the objects in order to identify similarities with neighboring entities. The TSV is generated by considering the matching similarities compared with the complete search space.

$$TSV = PC \div SD \tag{4.2}$$

- PC = The Path Count (PC) contains the value of the matching patch length.
- SD = The Search Depth (SD) is the search space of the target ontology, in most cases the search space has its default setting.

The outcome of the taxonomy matching is a set of alignment lists marked with the TSV. The involved expert is enabled to identify the usability of the listed alignments by use of the provided TSV. The TSV algorithm is presented in the following in pseudo code.

```
read SD;
read RDFmodel1;
read RDFmodel2;
result_model := NULL;
WHILE RDFmodel1.hasNext(RDFmodel1.RDFstatement) DO
  int PC := 1;
  WHILE (RDFmodel2.hasNext(RDFmodel2.RDFstatement)) && (RDFmodel2.length <=
       SD) DO
    IF (RDFmodel1.RDFstatement.subject == RDFmodel2.RDFstatement.subject)
        && (RDFmodel1.RDFstatement.object == RDFmodel2.RDFstatement.object
        ) {
      write.toPath(RDFmodel2.RDFstatement,resultModel.addPath);
      PC++;
      }
    float TMV = (float) PC / SD;
  END
END
```

As presented the path in the RDF graph from RDF model1 is matched with the path of another RDF graph from RDF model2. The path structure is identified by comparing the subject and the object of an RDF statement. In case of a match, the RDF statement is added to the graph of the result model and the PC increased. The similarity value is generated for each matching by the given formula. The presented algorithm is extendable through adding more or other entities similar to the LSV algorithm. The linking items, the predicates, are analyzed by the generation of the $NTSV$ in the next section (4.1.4.3).

4.1.4.3 Non-Taxonomy Matching: NTSV

The non-taxonomy matching is performed after the lexical and the taxonomy matching. In addition to the taxonomy matching it compares the relation between entities itself. This relations are the linking entities in a RDF structure; in a RDF triple this entities are the predicates. Each relation between entities is labeled with additional definition of the relation type. This type of relation is compared in order to identify the similarity grade. As presented in the listing above, the predicates identify the links between subjects and objects for an RDF based ontology. The amount of considered links depends on the previously defined search space of the ontology graph. The proceeding is similar to the taxonomy matching procedure but the linking entities itself are matched. Furthermore, the non-taxonomy matching is the last of the three performed matching steps. Its input data are the alignment lists produced by the previously performed taxonomy matching. The selection of the used input alignment lists is done by an expert or otherwise it is possible to automatically filter the taxonomy matching results by only making use of alignments reaching a previously defined threshold for the TSV. Anyhow, in both cases the input data for the non-taxonomy matching are the alignment lists generated by the taxonomy matching. The $NTSV$ is generated by considering the matching similarities of the linking entities compared with the complete search space.

$$NTSV = EC \div SD \tag{4.3}$$

- EC = The Entity Count (EC) contains the value of the matching patch length containing similar linking elements.

- SD = The *S*earch **D**epth is the search space of the target ontology, in most cases the search space has its default setting. However, the search space is different from the taxonomy procedures' search space due to the fact that the now used target ontology is an alignment list produced by the previously performed taxonomy matching.

The $NTSV$ algorithm is presented in the following in pseudo code.

```
read SD(FromTaxonomyMatching);
read RDFmodel1;
read resultModel(FromTaxonomyMatching);
RDFmodel2 := resultModel(FromTaxonomyMatching);
result_model := NULL;
WHILE RDFmodel1.hasNext(RDFmodel1.RDFstatement) DO
  int EC := 1;
  WHILE (RDFmodel2.hasNext(RDFmodel2.RDFstatement)) && (RDFmodel2.length <=
       SD) DO
    IF (RDFmodel1.RDFstatement.predicate == RDFmodel2.RDFstatement.
        predicate) && (RDFmodel1.RDFstatement.predicate == RDFmodel2.
        RDFstatement.predicate) {
      write.toPath(RDFmodel2.RDFstatement,resultModel.addLink);
```

```
11          EC++;
            }
       float NTMV = (float) EC / SD;
    END
END
```

As presented the linking entities in the RDF graph from RDF model1 are matched with the linking elements of another RDF graph from RDF model2 being a result from the previously performed taxonomy matching. The linking entities are identified by comparing the predicates of the RDF statements. In case of a match, the RDF statement is added to the result model and the EC increased. The similarity value is generated for each matching by the given formula. The presented algorithm is extendable through adding more or other entities similar to the LSV and TSV algorithm.

4.1.5 Interpretation

The interpretation is the last step of the overall ontology matching strategy. When the similarity values of the matching results are produced, the interpretation takes place. During the interpretation the ontology matching algorithm decides if a matching result is used based on the similarity value; the matching result is interpreted. However, the interpretation phase requires knowledge regarding the produced results. For this, the previously generated similarity values, $\{LSV, TSV, NTSV\}$, have been stored in a result data stores. In case the aggregation of the three similarity values reaches a defined threshold, a new generated alignment is used. The new alignments are generated automatically by the three matching procedures as follows:

1. **Concept Construction** A new concept by means of a subject is added to the new alignment list.

2. **Concept Description** The new subject is linked to an object by constructing a predicate.

3. **Concept Integration** The new concept is integrated into the complete alignment list structure by constructing a new object not present in the current alignment list or in case a relation is already given, the object is another already existing subject of the ontology.

However, the concept construction might also be the re-usage of an already existing concept in the alignment list. The aim of the concept construction is twofold: *(a)* the introduction of a new concept to the alignment list or *(b)* the update of an already existing concept. The produced alignment lists can be used as ontologies as well.
Additionally, the produced results need to be validate to prove their usability for the

expert. Hence, three approaches are used to prove the validity of the generated results.

1. The generated similarity values are considered in order to make an assumption if the result is useful for the experts needs.

2. The occurrence of the matched terms in reliable and domain related validation text documents is identified.

3. The generated alignment lists are matched with an reliable and domain relevant domain ontology by performing the three matching iterations with the produced results.

The three steps are performed consecutively. The validation of produced results will be explained more precise in 4.2.3.

4.2 Realization of Proposed Strategy

This section presents how the described ontology matching method is applied and utilized.

4.2.1 Applying Graph Based Matching Approaches

The use of the graph matching methods for performing a graph based matching approach opens a broad set of well proven matching approaches for ontologies. However, this section presents the use of the previously described ontology matching method by considering tree matching methods for performing the matching of the ontology structure (see 4.1.4.2). The matching of various ontologies is performed in three steps, the comparison of concepts, the comparison of relations and the comparison of the roles of the concepts. The matching of the concepts considers similarities between the concepts that are based on the syntax and on the semantic meaning of the selected concept. Identifying the similarity for the syntax is an easy task that can be solved by a simple string matching between concepts but it is time consuming. However, the semantic meaning is identified by matching the features and the values of the features of the selected concepts as they are presented by use of the RDF data structure.
The concepts are used as nodes of a path for a specific graph. By use of the relations between the concepts that have reached a certain similarity grade that is expressed by an similarity indicator the path is created. Furthermore, it is possible to make use of the roles of the concepts by including the labels of the edges. However, an ontology

language is required that is usable for graph matching approaches. For this, the RDF standard is used, it offers a set of statements that can be treated as items in a graph. Furthermore, it is the foundation for further standards. RDFS is an extension of RDF and can be used as well. A mapping between the RDFS statements and items in a graph are presented in the following table 4.3.

RDFS Element	Graph Item	Description
class	root element	The instance of the main class is the root element of the graph.
classes	node	The class concept creates an object and initiates an instance of an object by use of the *rdf:type* command. Each instance of a class is used as a node.
subClassOf	labled edge	A transitive property that defines a class hierarchy by use of the *rdfs:subClassOf* command.
subPropertyOf	labled edge	A transitive property that defines a property hierarchy by use of the *rdfs:subPropertyOf* command.

Table 4.3: Mapping between RDF-(S) elements and graph items

As presented four main categories are required for representing an RDFS structure as a graph:

1. *root element*: The root element is the main instance of the RDFS class in the graph.

2. *node*: Each instance of a class that is not the main instance is a node.

3. *labled edge*: The hierarchies of classes and properties are the relations in the graph.

For the application of the proposed strategy RDF ontologies will be considered. RDFS is an extension that can be used on top of the RDF structure with the presented approach. However, in order to apply the graph based approach the three steps of constructing the similarity value as already proposed in section 4.1.4 need to be considered. The first step, the lexical matching, is a matching of lexical entities in the selected ontology. Therefore, no graph matching is applied to the first step but the ontology structure is browsed as it is a graph. Only the matching of the lexical entities with the aim to construct the required similarity value for lexical matches (LSV) is performed for the first step.
The second phase of constructing a taxonomy similarity value (TSV) takes care of graph matching approaches by comparison of the relations between the entities. The use of the RDF standard allows the mapping of an RDF based ontology as graph. Such graph structure is used for the matching in order to find similarities.

Furthermore, the third step of constructing a similarity by considering the non taxonomy elements ($NTSV$) of the structure referring to the relations between the entities in the RDF structure itself and is therefore as well related to the graph based matching approach.

4.2.2 Use of the Proposed Strategy

The ontology matching process starts with an priority ontology PO that consists of concepts C, features of the concepts F, values of the features V and relations between the concepts R. The priority ontology is the source ontology. $C, F, R \in PO$ Every concept of the PO is matched with the concepts of comparison ontology O_x from the defined set that is in the previously defined search space. The matching is done by:

1. Comparison of the features and values of the concepts
    - High accordance of features of concepts: concepts might be the same
    - Indicator (probability value) of accordance of the matching is set
2. Comparison of the relations and type of relation between the concepts
    - High accordance of relations of concepts: concepts might be the same
    - Indicator (probability value) of accordance of the matching is set

The first comparison of the features of the concepts is performed in order to find similar concepts in the source and the target ontology. This step is required because two concepts of different ontologies might be the same but notated another syntax. Therefore, if only the syntax of concepts is compared not all similar concepts are identified. Through this, the comparison of the features and values of the concepts as it is described in the previous sections is a beneficial method for identifying similarities in different ontologies. This proceeding includes the semantic meaning and therefore enriches the identification process for concepts in different ontologies. The identification of the concepts is necessary for the second comparison step that identifies similar paths within two different ontologies presented as trees. The third step, the non taxonomy matching, required identified concepts as well. Nevertheless, it is possible to merge the matching results or to produce an alignment list. Both proceedings are described in the following.

Merging Matching Results The matching results are stored as taxonomies T in alignment lists. Every taxonomy is a path of a tree based on the used ontology, only

matching results that have an indicator (similarity value) reaching a certain threshold are used. Such threshold needs to be defined in the previously performed preparation phase.

- T_1 : results of matching for $C_1 \in PO$
- T_2 : results of matching for $C_2 \in PO$

Several taxonomies can be merged to a single ontology. For this a similarity value is the indicator for the matching results in the taxonomies. The merge of the taxonomies is a comparison of the results of the matched ontologies.

- Step 1: T_1 becomes the new ontology, T_1 = ontology
- Step 2: T_1 is compared with T_2 ; for the comparison the indicator (probability value) of the matching results of the ontology and T_2 is used; matching result with highest indicator is used; T_2 becomes the new ontology, $T_2 =$ ontology

Furthermore, the non-taxonomy matching is considered as well. The non-taxonomy matching considers the labels of the relations between the matched entities in the ontologies presented as tree in order to identify the type of relation between the entities. When thinking of an RDF structure, the labels of the relations are presented by the predicates. The non-taxonomy comparison matches the type of relations by comparing predicates for similar structures. For an RDF structure this means, in different taxonomies subjects and objects might be linked with different predicates. Hence, a similar proceeding needs to be performed as for the taxonomy comparison method. Thus predicates are identified and the type of relations between two concepts is explored.
Every taxonomy requires a new merging step to check if parts of the current ontology have to be replaced. Hence, the number of merging iterations depends on the number of created taxonomies. Furthermore, the threshold needs to be considered in order to ensure a useful level of trustworthiness regarding the matching results. The merging of produced matching results is a computing intensive procedure that needs to deal with trustworthiness problems when deciding which results to merge together. However, the proposed approach produces matching results that can be used as alignment lists or as merged alignment lists based on the generated matching results.

Producing Alignment Lists The described use case scenarios (see 2) need automated support for HPC terms. For this, an alignment list is helpful explaining the terms of such HPC texts. The generation of an merged alignment list does not face the problem of merging in an incorrect manner as it might happen when merging complete ontologies or several matching results. However, alignments can be related to each other when the RDF data standard is used but the set of alignment lists includes all

taxonomies generated by the ontology matching procedure. In case, different taxonomies describe different relations between the alignments, the different alignments will be stored as alternative results in the alignment list data store. The alternative alignments are those with a lower similarity value. They will be provided to an expert as well so the expert is enabled to select which alternative to select. Nevertheless, the proposed alignments are the same as produced when using the merging of matching results approach:

- T_1 : results of matching for $C_1 \in PO$
- T_2 : results of matching for $C_2 \in PO$

However, the new merged alignment list includes all taxonomies:

- Step 1: T_1 becomes the first alignment list, described with the similarity value
- Step 2: T_2 becomes the second alignment list, described with the similarity value
- Step ...: $T_{\cdot}$.. becomes the ... alignment list, described with the similarity value
- Step n-1: T_X becomes the last alignment list, described with the similarity value
- Step n: all taxonomies are ordered by the similarity value

The last step n is required to provide the best fitting taxonomy as the first ordered by the similarity value.

4.2.3 Validation of the Matching Results

Vector based techniques are a considered instrument to compare the matching results with a text document related to the domain of a given use case scenario (see 3.1.2). The text document is provided by an expert for the automated validity check. This Validity Check Document (VCD) contains content regarding the domain of the use case scenario and the domain of selected ontologies. Through random indexing techniques, the occurrence of terms are evaluated.
The random indexing approach is a based on word space approaches and therefore applicable for the required validity check with the VCD. In general the idea of using word spaces is to create a high dimensional vector space for words and further to construct a statistic that is used for the previously mentioned vector space. In the described urgent reasoning scenario the considered words for the validity check are the generated terms of the matching approach. When thinking of the vector space that is constructed by providence of the previously statistic, this strategy follows the

conception that if a set of words continuously appears in a text in the same context the meaning of the words is the same. Through this, it becomes possible to validate the terms of the matching result with the VCD in order to verify if the terms appear in similar context. However, in order to make an assumption about the context of the terms the validity check examines the related terms in the VCD and in the priority ontology with the aim to compare the related terms. In case there is a high accordance between the related terms in the VCD and the related terms in the priority ontology, the validity check accepts the matching result. Nevertheless, the word space approaches face the challenge of scalability and efficiency. The use of HPC resources covers this challenge but a more fine grained approach for dealing with this issue in order to reduce the amount of required computing resources is recommended.
For this, the simple vector based word space approach is enhanced by using the vector based random indexing approach that creates models such as done in latent semantic analysis (LSA) approaches. Following such approaches first of all an extensive co-occurrence matrix is created and then second a reduction phase of the co-occurrence matrix is performed that limits the size of the used matrix. Within the reduction phase, the occurrence of upcoming vectors of terms in a specific context is aggregated to accumulated context vectors. Hence, the random indexing approach reduces the amount of required computing resources to perform the validity check in the given period of time.
Nevertheless, the vector based techniques are facing the challenge of producing not usable results in case that the text documents for comparison are not well fitting to the specific scenario domain. This introduces the question, if the use of a vector based approach for a validity check as described previously is usable without demanding a high effort from an expert being in the need to do a very precise selection of text documents fitting to the use case scenarios domain. The evaluation of vector based approaches as well as a strategy for performing a validity check by use of comparison ontology are considered. In case a comparison ontology is used, the ontology is matched with the priority ontology with the aim to make a prove of confidence regarding the updated priority ontology. For this, online ontologies can be used as performing a validity check. Sabou et al. [87] are presenting a strategy for exploring the continuously increasing number of online available ontologies in order to use them as a source of evidence. This approach implies a continuous evaluation of online available ontologies. However, the use of an ontology as source of evidence from the ontology set, defined by an expert, permanently available by use of the ontology repository DB allows an easy to use selection of a validation ontology. However, the ontology used as a source of evidence needs to be reliable and trustworthy. The use of an inappropriate ontology is a risk for the validation and the creation of the alignments because of the possibility that the matching results are marked as not usable through the validation also if the validation itself is not trustworthy and reliable. Hence, the expert has to do a well-considered selection of the ontology used as the source of evidence. Even when the validation of the matching result is producing deviate results it is still an issue of trust and reliability if the expert decides to make use of the afterwards validation of the matching results.
To sum up the approach for validating the produced matching results, three steps

ensure reliable and trustworthy results. As a first step the expert considers the generated similarity values of the matching procedures. Afterwards, a VCD is used to identify the occurrence of the result terms and as a last step the generated alignment lists are matched again but with the domain ontology. Nevertheless, the validation of the results is a matter of trust and reliability as well.

4.2.4 The Priority Ontology

The priority ontology is a selected ontology used as a base for the ontology matching strategy in order to enable a better reasoning capability for the expert. To be more precise, the priority ontology is the source ontology used for matching as described above (see 4.1) with target ontologies. The result of the matching procedure consisting of the lexical, taxonomy and non-taxonomy matching are alignment lists containing the generated facts. However, before the alignment lists are finalized several iterations take place. Additionally, the quality of the results depends on the previously performed preparation steps. The quality of the matching results is ensured through the defined threshold for similarities and the afterwards performed prove of validity. In case the threshold is not reached, the matching result will not be used for the migration of the results into the alignment list. The matching results reaching the threshold are used for the migration by several iterations. However, the alignment lists will contain alternative matching results that have reached the threshold but were overruled by another matching result with a higher similarity grade. The alignment lists contain all matching results ordered by the similarity value reached the threshold generated through the described matching approach. For this the priority ontology is used as foundation to be matched with selected target ontologies.

4.3 Conclusions of Ontology Matching

The described ontology matching approach consists of a preparation phase, the matching by generation of a similarity value and an afterwards interpretation of the results by creation of an alignment list. The generated results are validated. Further, the ontology matching is performed by three sequentially performed matching procedures performed in several iterations. The matching methods are:

1. a **lexical matching**: the values of compared concepts are matched by validating the lexical structure;

2. a **taxonomy matching**: the neighboring concepts are identified and validated in order to examine similarities of the structure of an ontology by consideration of the search depth of the tree;

3. a **non-taxonomy matching**: the relations between the concepts are analyzed with the aim to identify the type of relation to the neighboring concepts.

For performing the matching iterations the source ontology as well as the target ontology used for this iteration are mapped as trees. Such trees are compared and the results are stored with the constructed similarity values in case the previously defined threshold is reached. In case the threshold is not reached, the result will not be stored. The complete matching of all ontologies is done by matching the source ontology with all assigned target ontologies. The validation of the matching results is ensured through the use of the three validation methods including the three different matching procedures (see 4.1.4) and the use of a similarity value as well as alternative matching results.
However, semantic data sets are growing even in the future. This effects directly the processing time and overall performance for matching procedures. Even a data set consisting of GB of data is not big data looking from the HPC perspective but it causes high execution times for the matching calculations. Thus, focusing on the matching approach GB of data are big data in terms of the performed calculation iterations growing massively with increasing data sets. Hence, matching strategies need to deal with growing data sets in order to deal with large data volumes to ensure an appropriate reasoning based on the matching results. The matching of large data sets described in 2 might lead to execution times of several hours or even days. The distribution of matching tasks can be distributed on several computing resources to face the challenge of performing a semantic matching by use of large data sources. For this, the next chapter proposes a strategy for handling very large semantic data sets in order to increase the performance of execution times for the proposed matching methods.

Chapter 5

Distributed Ontology Matching

As presented in chapter 4 a priority ontology is the foundation for updating and enhancing semantic structures with additional semantic data by adapting a matching strategy based on similarity values. This strategy is composed of a lexical, a taxonomy and a non-taxonomy matching procedure. For using the matching results a reasoning strategy is required dealing with an approach for performing inferences on base of the produced facts by consideration of scalability and performance. Thus, the first step is to enhance the matching strategy in a way enabling an expert to perform the matching procedures in a less time consuming manner. When thinking of reasoning, common reasoning strategies are forward and backward chaining producing new inferences at run-time or in advance. This chapter presents an improved ontology matching strategy and how to adapt a reasoning strategy taking into account current scalability and performance problems when executing inference rules.

5.1 Techniques for Improving Performance by Distribution

5.1.1 General Advantages of Distribution

The core concept of distribution techniques for increasing the performance is the idea to separate processes, workload or domains with the aim to run each item on several computing resources, as several threads on different nodes and so forth. Through this it becomes possible to run the different items in parallel on the allocated computing resources. However, the distribution of processes and processing of several processes at same time leads to challenges in the area of parallel computing such as dependencies between processes and latencies. The parallelization approach depends on several circumstances such as selected hardware architecture, decomposition of data, work or the domain and further, the selected programming model. In order to deal with an

ontology matching supporting a reasoning strategy in a distributed computing environment a brief overview about existing parallelization strategies is presented. This overview is a short summary for demonstrating the complexity of the parallelization challenge. In addition, the distribution and parallelization approach for the presented work is identified.
In general parallelism can be adapted implicit on job, task, block, instruction or BIT level. In case a job runs in parallel this means an execution of jobs at same time. Usual problems are different length of jobs, different speed of processors and the optimal minimalization of the total execution time. A parallel execution of tasks is given in case parts of a program are executed at same time. The task parallelism implies the problem of switching between sequential and parallel parts of selected tasks. Further, the block parallelism describes the parallelization of instruction blocks by consideration of data dependencies between instruction blocks. Such data dependencies occurs in case execution order or block statements change which will change the computation results. Additionally, procedural (given in the logic of a program) and operational (blocks of instructions need the same physical device) dependencies need to be taken into account as well. Beside the already mentioned issues, instructions can be performed in parallel by consideration of dependencies and the given logic of the instructions.
When thinking of parallelism strategies, a decomposition of work, data or the domain needs to be evaluated. The work decomposition is given in case several working steps are executed at same time. Data decomposition allows a parallel execution of data sets. Further, the domain decomposition describes the process of splitting up a problem into several domains and execute such domains in parallel.

5.1.2 Benefits of the LarKC platform

The European founded LarKC project provides a platform for massive distributed incomplete reasoning that aims to deal with the challenge of solving scalability problems for existing reasoning systems as it was presented in 3.2. The overall goal of LarKC is to deal with three major issues:

1. Enriching the current logic-based semantic web reasoning methods by enhancement and adaption of knowledge discovery for reasoning in order to use more efficient reasoning methods.

2. Employing cognitively inspired approaches and techniques for improving the reasoning process as well and also increase the effectiveness of the followed reasoning procedure.

3. Building a distributed reasoning platform with the aim to make use of a broad set of computing resources. Regarding the allocation of the used computing

resources LarKC supports a computing at home approach as well as the use of a high performance computing cluster that provides required computing resources in order to guarantee the amount of needed computing resources to perform a reasoning even for very large data sets and computationally demanding reasoning algorithms.

For the ontology matching approach described in 4 the distributed reasoning platform of the LarKC project is a considered existing system for dealing with the analyzed requirements that are improved by adapting computing resources being distributed as a cluster. Additionally, the described ontology matching approach is aligned with the first two major goals of LarKC, improvement of a reasoning strategy and using cognitively inspired approaches. Both issues are handled with the ontology matching strategy of this work by considering an effective reasoning based on ontologies in order to provide an expert with results that are produced by the ontology matching and afterwards available by the matching results. At this point there is a clear distinction between the results from the LarKC project and ontology matching approach presented in this work. The LarKC project focuses on reasoning methods to provide reasoning results in an efficient way. This work targets an efficient approach as well but the focus is the ontology matching in order to provide an individual knowledge base customized to a specific domain. This knowledge base is the result of the ontology matching. Further, the matching procedure will be improved by applying it to a HPC environment. Following, it becomes possible for an expert to execute a scalable an efficient matching for providing the results being the knowledge base for reasoning tasks. An overview of the LarKC project was already given in previous chapters 3.2.
It follows that the third major issue of LarKC is as well a major issue of this thesis. The ontology matching approach needs to be performed by use of needed computing resources such as a distributed shared memory system (e.g. provided by supercomputing centers such as HLRS). However, the use of an adequate matching strategy for reasoning is an open issue. Such matching strategy needs to deal with large data sets consisting of trillions of RDF triples. Looking at the Large Triple Stores presented in the abstract section, it was possible to process more than a trillion RDF triples in approximately 338 hours (approx. 829.556 triples per second) in August 2011. This example demonstrates the need of making RDF processing more efficient in order to deal with very large sets of data. While the LarKC project had its focus on en efficient reasoning strategy this work targets an improved ontology matching for supporting reasoning methods with an individual knowledge base.

5.2 Ontology Matching in a Distributed Environment

The presented work focuses on the ontology matching and gives an overview on how to enhance a matching in a distributed fashion usable for reasoning tasks. The ontol-

ogy matching is executed in order to provide an expert with a useful set of alignments containing data relevant to a specific domain. Reasoning is performed in order to make use of the information available in the alignment lists. The most data intensive task in the scenarios (see chapter 2) are the matching procedures for large data sets for enabling reasoning taking place after execution of the ontology matching approach. The ontology data data are small but the execution of the matching by using these data set becomes a challenge due to a large amount of comparisons between entities of the selected ontologies.
When thinking of an ontology matching for enabling a highly scalable matching and reasoning, sufficient computing resources need to be allocated. For this, the matching distribution for offering the possibility to perform reasoning tasks is a promising approach. In the following the resource allocation and the strategy for an efficient reasoning are presented. The resource allocation is a crucial task in this scenario due to the fact that for large computational procedures the provision of the needed computing resources takes time to be aligned with the given requirements.

5.2.1 From Queues to SLAs

At current state queues are often used to allocate computing resources as, for instance, the use of a prioritization model such as the use of a bronze, silver and gold queue. The computing jobs stored in the queue with the highest priority are selected first. Usually the price for a high priority queues is higher than the price for a low priority queue. However, beside the use of queues SLAs can be used to define requirements of a job and allocate adequate computing resources for the given job.
As already presented by means of the plugIT project SLAs are usable for identification and allocation of required HPC resource 3.2. The use of long-term SLAs enables an expert to improve the job scheduling on a HPC cluster (see Tenschert et al. [160]). The long-term SLA provides a method for storing configurations of HPC resources, e.g. type of used machine or installed software packages. In addition it is possible to prioritize SLAs in order to schedule jobs effective and avoid latencies for matching tasks depending on each other.
Today, HPC cluster consider a job scheduling done by using batch queues such as OpenPBS [161] or TORQUE [162]. Most often a HPC provider maintains several HPC resources with a diverse set of queues with different prioritizations. For the usage of batch queues for several submitted jobs various scenarios are possible such as the use of a "'first come first served"' (FCFS) algorithm in order to execute the job in the queue with the highest priority level first. Most HPC provider perform their job scheduling by use of best-effort methods.
Additionally, the job scheduling varies between local workstations and HPC systems. Though it is possible to submit a job directly on a node of a workstation the job submission at a HPC system is performed by use of *front end* or *login* nodes accessible from the outside. The *front end* or *login* nodes interact with the job scheduler for set-

ting up the job on the cluster.
When thinking of the previously described ontology matching approach (4) the performed ontology matching require a scheduling of given computing jobs. The use of a scheduling system as described above supports an expert for resource allocation. Additionally, the process for the provision of needed HPC resources is improved. The presented scheduling approach as it was presented by Tenschert et al. enables a scheduling based on QoS parameters allowing an expert to process computing jobs by a defined scheduling plan. Such scheduling plan will be used for the ontology matching approach described in this work. Thus, the distributed execution of the ontology matching allows a processing of the given ontology matching tasks by performing several jobs in parallel in an HPC environment. The next sections describe the job parallelization of the ontology matching approach. Following an overview of an improved usage of the ontology matching results is proposed to support an expert by increasing efficiency of reasoning tasks making use of the ontology matching results. However, the reasoning is up to the expert depending on individual needs.

5.2.2 Distributed Ontology Matching

The distributed ontology matching makes use of the allocated HPC resources provided by a job scheduling system as presented in the previous section. The proposed ontology matching strategy (see chapter 4) consists of three matching steps, the lexical, the taxonomy and the non-taxonomy matching. The three steps are performed sequential. However, within each of the three matching steps the matching can be performed in parallel by use of a distributed computing architecture. In this case, the most important issue is the allocation of computing resources and the execution of the identified jobs. Given the fact that the presented approach makes use of matching the tasks in parallel, the matching results reaching the previously defined threshold are merged in the alignment list. Hence, the ontology matching is performed by running the matching iterations in parallel. This is done for each matching step, the lexical, the taxonomy and non-taxonomy matching. However, the reasoning depends on producing results by applying inferences. Such inferences produce a high amount of assertional data. Thus a more efficient ontology matching is presented by use of HPC resources. Following an overview about possibilities on how to improve reasoning tasks is given.

5.2.3 Distribution on HPC Resources: Ontology Matching

As described SLAs can be used to define the amount, type and configuration of needed computing resources (see section5.2.1). The distribution of the presented ontology matching requires the provided HPC resources. When thinking of the distributed on-

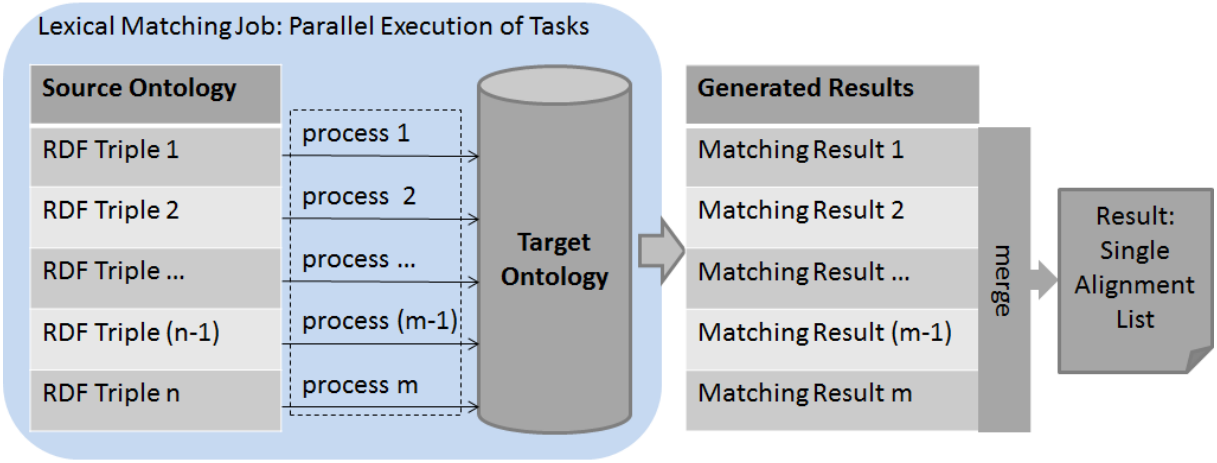

Figure 5.1: Parallel Job Execution of the Lexical Matching

tology matching approach, the three matching steps need to be considered. Thus, for each of the three matching steps the parallelization method is described.

Lexical Matching The complete matching task is divided into several independent tasks performed as single processes. When thinking of the proposed RDF ontologies, each concept in the source ontology is matched with the concepts in the target ontology. Thus, the parallelization on task level enables the matching of each concept in the source ontology with the concepts in the target ontology depending on the search space. As described, the default search space is the maximum number of possible matching operations leading to a matching of a set of concepts from the source ontology with all concepts of the selected target ontology. Figure 5.1 presents the parallel task execution for the lexical matching. The task parallelization is performed by running the tasks in several parallel processes distributed on the allocated nodes.

The amount of lexical matching tasks performed in parallel depends on the amount of data chunks of the source ontology. Each chunk of data becomes a single matching task. Further, after the lexical matching is performed, the matching results of each matching processes are merged to a single alignment list. Further, the amount of performed comparison operations is as follows:

- Entity A consists of A_n characters and is compared with entity B consisting of B_m characters. Thus, for each lexical comparison between two entities the amount of comparison operations is A_n multiplied with B_m.
- Entity A is compared with each entity from the selected search space.

This concludes in the following amount of lexical comparison operations per entity of the source ontology:

$$entityX(n) \times (entity \sum_{entity=1}^{S} \times m)$$

The selected entity being marked as entity X consists of n characters being compared to each entity of the target ontology being in the scope of the search space. Regarding the target ontology the first entity in the defined search space is identified with the index 1 and the last one with the index S being the maximum index. n is the amount of characters for the entity from the source and m is the amount of characters for each entity of the target. In case a comparison operation identifies a match this is registered. This implies that each identified match causes an additional write operation whereas each comparison operation causes a load operation for receiving the characters. When thinking of the scalability, sequential parts being performed before initializing the lexical matching approach are the three steps presented in section 4.1 including the identification, the selection and the definition phase and in addition, dividing the ontology into data chunks. The lexical matching being performed on several nodes makes use of the data chunks. When separating the data into chunks the search space is taken into account so that relevant data are available for comparison operations. Thus, when increasing the amount of nodes used for performing the lexical matching approach the execution time will decrease. The overall data size is fixed but it is divided to several nodes so that the matching approaches will be performed in parallel. It is expected that providing chunks of data so several nodes enhance the performance but in case of a match an additional write operation takes place. Thus, the execution time will vary between comparison operations identifying matches and those that are not matching. The data size used for the lexical matching approach is presented in the next chapter in section 6.3.1 and 6.3.2.

Taxonomy Matching After the lexical matching, the taxonomy is performed by consideration of the parallel task execution strategy. The parallel task execution is similar to the parallel lexical matching due to the fact that the linking of the taxonomy of the RDF graph paths is matched. Hence, chunks of data are generated. Again, the produced output is a set of alignment lists. Each matched path is stored as a single alignment list used by the afterwards performed non-taxonomy matching. Each matching matching task processes a graph matching of the taxonomy and is performed in parallel by running several processes on the allocated nodes. These tasks are executed in parallel using a distributed shared memory system. The results are not merged together because the non-taxonomy matching requires alignment lists for each matched path. Figure 5.2 presents the parallel taxonomy matching.

The amount of taxonomy matching tasks performed in parallel depends on the amount of identified RDF Triple paths when comparing the source with the target ontology. The search space limits the search depth. The amount of comparison operations is as follows.

- A path from the source P_s being limited by the search space is compared with the

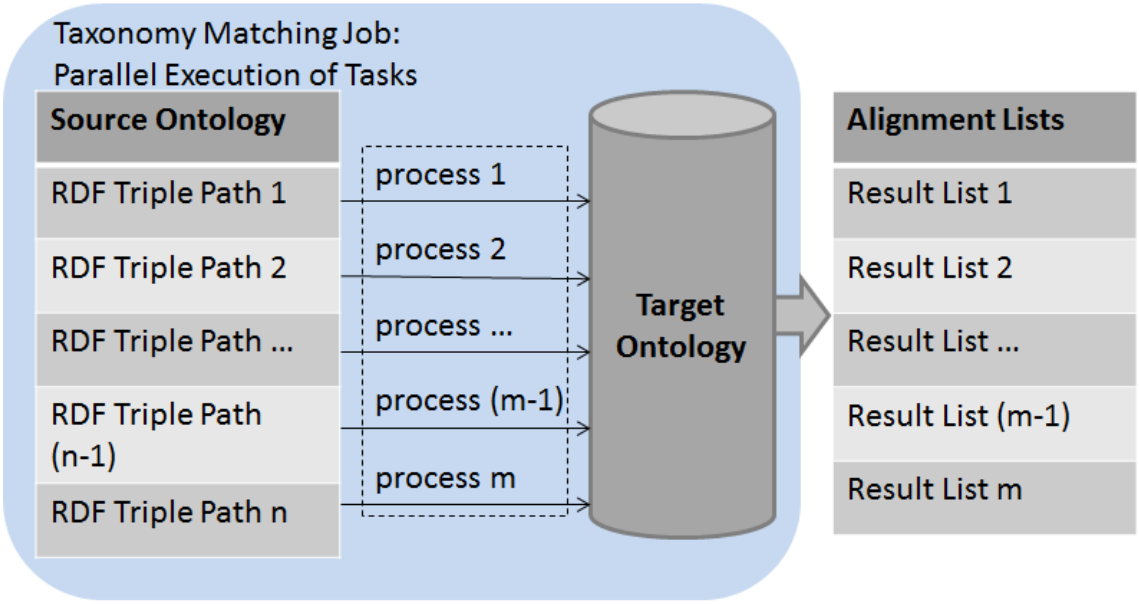

Figure 5.2: Parallel Job Execution of the Taxonomy Matching

target ontology. The comparison takes place between the entities of the source and the target ontology.

- For the comparison the entities from P_s are compared with the entities from the target ontology.

Thus, the amount of taxonomy comparison operations is as follows per path of the source ontology:

$$entity \sum_{entity=1}^{P} s \times entity \sum_{entity=1}^{S}$$

The selected path from the source being marked as path P_s is limited by the search space S. For both, the source and the target ontology the first entity in the defined search space is identified with the index 1 and the last one for the source ontology with P_s and for the target ontology with the index S. In case a comparison operation identifies a match this is registered, similar to the lexical matching approach. Also, this causes an additional write operation whereas each comparison operation causes a load operation for receiving the characters. As already done by the lexical matching approach, the data are divided into chunks for the taxonomy matching as well. Regarding the scalability, due to the fixed data size, when increasing the amount of nodes used for performing the taxonomy matching approach the execution time will decrease. The expectation is the same as for the lexical matching approach. Providing data chunks on several nodes will enhance the performance but a matching will cause an additional write operation and thus the execution time will vary depending on the amount of data chunks and matches.

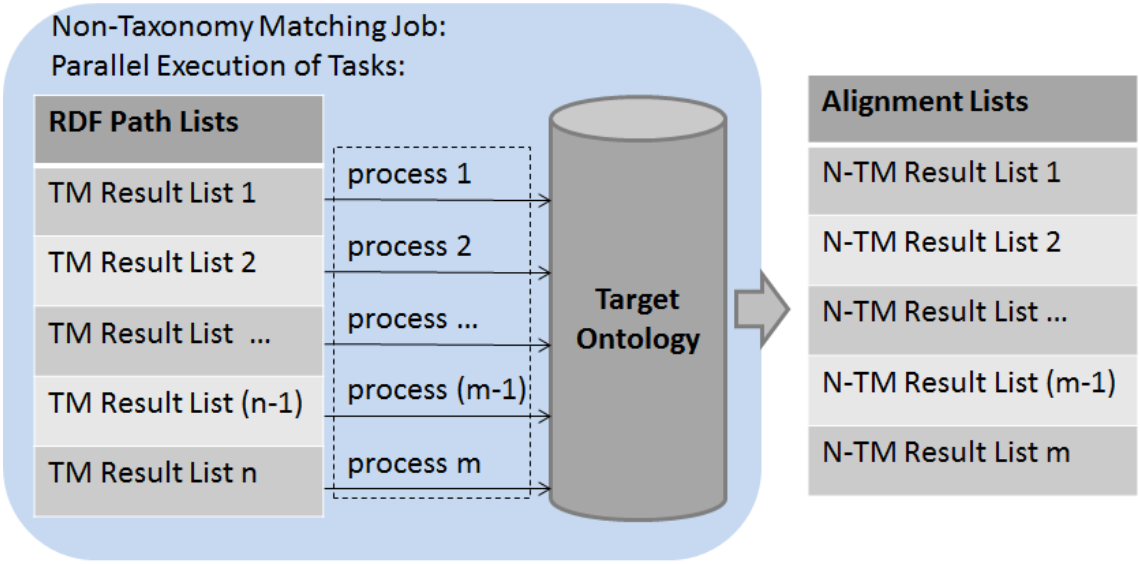

Figure 5.3: Parallel Job Execution of the Non-Taxonomy Matching

Non-Taxonomy Matching The last matching step is the non-taxonomy matching, it analyzes the relations between the concepts of the ontologies. The previously presented lexical and taxonomy matching make use of chunks of data. However, the parallel task execution of the non-taxonomy matching is performed by using the alignment lists generated through the taxonomy matching and perform them in parallel in the HPC environment. Thus, the data chunks used by the non-taxonomy matching are the results produced by the taxonomy matching. Figure 5.3 presents the non-taxonomy task execution.

The amount of non-taxonomy matching tasks performed in parallel depends on the amount of alignment lists produced by the taxonomy matching procedure presented as follows.

- A path from the source P_s being limited by the search space is compared with the target ontology. In contrast to the taxonomy matching approach, the comparison takes place between the source and the target ontology by using the labels of the edges.

- For the comparison the labels from P_s are compared with the labels from the target ontology.

Thus, the amount of non-taxonomy comparison operations is as follows per path of the source ontology:

$$label \sum_{label=1}^{S} \times label \sum_{label=1}^{S}$$

The data chunks used for the non-taxonomy are the results generated by the previously performed taxonomy matching iteration. Thus, it is a reduced data set. Due to the comparative small amount of data the search space S is per default set to the full data set. This implies that S is the maximum index. Similar to the lexical and the taxonomy matching approaches each comparison operation causes a load and each match a write operation. Hence, execution times can increase for data sets containing lots of matches. When thinking of the scalability, due to the fixed problem size of the data a reduction of execution times is expected but the data size needs to be considered. When using the reduced data sets the improvement of execution times might be negligible depending on the data size. In chapter 6 a measurement of performance takes place.

5.2.4 Distribution on HPC Resources: Closure and Materialization

The closure approach, performing the inference rules by using an ontology makes use of allocated computing resources. The closure approach is responsible for constructing new facts at load time to provide the expert with results for submitted queries. The use of the closure method indicates a workload partitioning through breaking down the problem to execute it in parallel. For use of the closure approach the RDF data are partitioned on several nodes. Each partitioned part of the data is used for applying inference rules to generate new facts. The adaption of the inference rules is executed in several processes executed in parallel. The complete forward-chaining based closure method is presented by Weaver and Hendler. In this work the closure approach is presented to partition the workload of a reasoning task by data partitioning. The steps to be performed are:

1. decomposition of reasoning tasks
2. scheduling of reasoning jobs
3. processing the inference rules

After the inference rules are processed, the results need to be merged.

5.3 Applying Distributed Matching Procedures

The distribution approach of the proposed ontology matching strategy needs to be adapted to the HPC environment. Thus, firstly the architecture is described in order

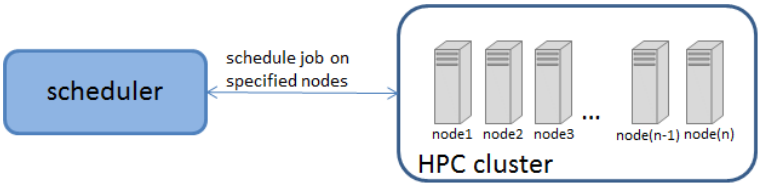

Figure 5.4: Job Scheduling Using HPC Cluster Nodes

to identify the given needs and dependencies. Afterwards, the ontology matching approach is presented based on the given infrastructure and finally an overview on how to adapt a reasoning strategy to the given approach is presented.

5.3.1 Ontology Matching on Several Nodes

The general idea is to make use of an HPC environment by use of a computing cluster. The cluster consists of several physical computing resources such as nodes. Each computing job executed in the cluster runs on these nodes. A scheduling system as it was presented in section 5.2.1 enables a job scheduling specified to the given needs as demonstrated in figure 5.4. A beneficial proceeding for parallel execution of matching tasks is the allocation of the complete matching process on several nodes. Thus, the matching jobs are processed as described in the previous sections but by defining the amount of used computing resources by selecting a cluster and defining the amount of used nodes and the required availability time of the system. For instance, at HLRS, the NEC NEHALEM cluster is used for running matching jobs through defining the amount of needed nodes. The NEC NEHALEM cluster is composed out of more than 700 computing nodes. The NEC NEHALEM cluster was ranked in the Top500 list [163] in 2009 to be the 77^{th} fastest computing system and in 2011 the 305^{th} fastest.
The proposed matching strategy consists of the lexical, the taxonomy and the non-taxonomy matching approach. Hence, such a proceeding indicates the processing of the three matching procedures as proposed before but defining the amount of used nodes. Through this proceeding, a lack of available computing resources is solvable by allocating a sufficient amount of computing nodes. The computing resources of a cluster provide more computing power than conventional PCs. Therefore, simply allocating a huge amount of nodes enables more complex calculations by decreasing the needed time. Such a proceeding is easy to use but not efficient when thinking of parallelization strategies. In case of using several nodes of a cluster it becomes possible to distribute the processing as it is required for the ontology matching on several nodes in parallel. Thus, the allocated computing resources are used more efficient. In the next section, the parallel execution of the proposed matching tasks is presented.

5.3.2 Distributed Ontology Matching Procedures

As proposed in section 5.3.1 a cluster is usable for using several nodes for performing matching tasks. However, besides the allocation of the required computing resources, the matching processes need to be divided to be runnable in parallel on several nodes. Thus, the matching procedures for each of the three matching steps need to be aligned to the parallelization strategy. The algorithms of the matching strategies need to be enhanced by enabling a parallel usage.
The following updates need to be included:

- LSV, TSV and NTSV are merged to one matching cycle
- the lexical matching will change. each matched term is matched sign by sign for each lexical matching iteration
- the distribution will be performed by splitting up the RDF data in several chunks. The RDF chunks need to include as well the data they point to, otherwise too much communication will be required when linking from one RDF chunk on a node to another one.

Parallel LSV Algorithm The matching is divided into several matching tasks by using the *LSV* algorithm and running parts of the outer *WHILE* loop in parallel depending on the defined chunks of data. Each chunk consists of a set of RDF triples. Hence, the matching iterations are divided into several matching tasks being performed in several processes depending on the amount of used nodes. Each presented matching job algorithm consists of an outer and an inner *WHILE* loop as demonstrated in the following in pseudo code.

```
WHILE (RDFmodel1.hasNext(RDFstatement)) DO

  <parallel>

  read RDFmodel1.RDFstatement(X);
  read RDFmodel2;
  result_model := NULL;
  int AS := 0;
  float LSV := 0;
  WHILE (RDFmodel2.hasNext(RDFmodel2.RDFstatement)) && (RDFmodel2.length <=
       textit{SearchSpace}) DO
  IF (RDFmodel1.RDFstatement.subject == RDFmodel2.RDFstatement.subject) {
      write(RDFmodel2.RDFstatement.subject,resultModel);
      AS++;
      }
  IF (RDFmodel1.RDFstatement.predicate == RDFmodel2.RDFstatement.predicate)
       {
      write(RDFmodel2.RDFstatement.predicate,resultModel);
      AS++;
```

```
      }
  IF (RDFmodel1.RDFstatement.object == RDFmodel2.RDFstatement.object) {
      write(RDFmodel2.RDFstatement.object,resultModel);
      AS++;
      }
  IF (count>0) {
       LSV := (float) AS / 3;
       } ELSE {
         LSV := NULL;
        }
  END

  </parallel>

END
```

In case of X data chunks composed out of RDF model1, X matching tasks are needed. Each result is stored in the result model being an alignment list.
The presented *LSV* pseude code compares subjects, predicates and objects as complete terms. In case of a match a counter (*AS*) is increased and divided by the number of matched terms (*3*). The result is a floating-point number between *0* and *1* expressing the grade of similarity, whereas *0* means no similarity and *1* means identical terms. For creating a more precise similarity the matching procedure for subjects, predicates and objects is enhanced through matching term by term, e.g. a subject consists of five tokens the matching will check the five tokens with the subject from the target ontology and afterwards divide it by five. The following pseudo code presents the required update needed for including this enhanced similarity calculation for the lexical matching.

FOR (i := 0; i ¡ SO_{tl}; i++;) BEGIN IF ($SO_{subject.token[i]}$ == $TO_{subject.token[i]}$) : AS++; END AS := (float) AS / SO_{tl};

- *i* is an index used to run the for loop depending on the length of the term from the source ontology.
- the term meaning is as follows: Source Ontology (SO); Target Ontology (TO); Term Length (TL).
- *subject* is replaced with the predicate or the object according to the matching step.
- *token* is the currently used token of the term used for matching it with another token from the target ontology.
- *AS* is the enhanced similarity value; in case a token matches with another token it is increased by one; after finishing the for loop the *AS* value for a subject,

predicate or object is calculated through dividing by the amount of tokens from the term of the source ontology.

The presented calculation enables a token-wise instead of the previously performed term-wised proceeding. This proceeding ensures a much more precise similarity value creation whereas for the benefit of performance it does not proof if the matching term from the target ontology includes more tokens than the source ontologys term. However, the following taxonomy matching detects mis-matches caused through this issue. Further, in case the target ontology term has less tokens it is handled by not increasing the *AS* value.
The parallel lexical matching is the first of three matching steps, it is followed by the taxonomy and the non-taxonomy matching.

Parallel TSV Algorithm Similar to the proceeding done for the *LSV*, the *TSV* algorithm is divided into several matching tasks runnable in parallel on several computing nodes. Parts of the outer *WHILE* loop are performed in parallel while the inner *WHILE* loop presents the taxonomy matching being part of each matching task as presented in the following in pseudo code.

```
WHILE (RDFmodel1.hasNext(RDFstatement)) DO

  <parallel>

  read SD;
  read RDFmodel1.RDFstatement(X);
  read RDFmodel2;
  result_model := NULL;
  int PC := 1;
  WHILE (RDFmodel2.hasNext(RDFmodel2.RDFstatement)) && (RDFmodel2.length <=
       SD) DO
    IF (RDFmodel1.RDFstatement.subject == RDFmodel2.RDFstatement.subject)
        && (RDFmodel1.RDFstatement.object == RDFmodel2.RDFstatement.object
        ) {
      write.toPath(RDFmodel2.RDFstatement,resultModel.addPath);
      PC++;
      }
    float TMV = (float) PC / SD;
  END

  </parallel>

END
```

Similar to the parallel lexical matching but focusing on the matching paths of the source and the target ontologies, the taxonomy similarity is created. The path count (*PC*) is increased by each iteration and finally divided by the search depth (*SD*).
In a similar manner each data chunk composed out of RDF model1 becomes a single

matching task. X is the amount of RDF statements and matching tasks. The results are stored in a result model.

Parallel NTSV Algorithm As before, the *NTSV* algorithm is divided into several matching tasks runnable in parallel on several computing nodes but by using the generated output of the taxonomy matching as input. Again, parts of the outer *WHILE* loop are processed in parallel while the inner *WHILE* presents the non-taxonomy matching as presented in the following in pseudo code.

```
WHILE (RDFmodel1.hasNext(RDFstatement)) DO

  <parallel>

  read SD(FromTaxonomyMatching);
  read RDFmodel1.RDFstatement(X);
  read resultModel(FromTaxonomyMatching);
  RDFmodel2 := resultModel(FromTaxonomyMatching);
  result_model := NULL;
  int EC := 1;
  WHILE (RDFmodel2.hasNext(RDFmodel2.RDFstatement)) && (RDFmodel2.length <=
       SD) DO
    IF (RDFmodel1.RDFstatement.predicate == RDFmodel2.RDFstatement.
        predicate) && (RDFmodel1.RDFstatement.predicate ==
        RDFmodel2.RDFstatement.predicate) {
      write.toPath(RDFmodel2.RDFstatement,resultModel.addLink);
      EC++;
      }
  float NTMV = (float) EC / SD;
  END

  </parallel>

END
```

Each matching iteration increases the entity count (*EC*) finally divided by the search depth (*SD*).
The alignment lists of the taxonomy matching become single matching tasks used for the non-taxonomy matching. The results are stored in a result model.

In order to avoid waiting times when submitting the matching jobs to the cluster, a queue containing all needed matching jobs is produced in advance. Afterwards, the jobs are performed on the cluster by use of the previously allocated computing resources including the defined amount of nodes. Each of the three matching jobs (lexical, taxonomy and non-taxonomy) contain several matching tasks depending on the amount of prepared data chunks.

5.3.3 Applying Reasoning: Closure and Materialization

This section deals with a general overview on how to apply reasoning to ontologies. The produced matching results are stored as alignment lists usable as ontologies. However, for the user another issue is to perform the reasoning itself in a more efficient manner. Nevertheless, reasoning is required for using the matching results but it is not the main focus of this work. Hence, the reasoning is presented in a descriptive manner. The idea is the use of HPC resources to face the problem of dealing with high amounts of data produced through inference making by producing assertional data. Two methods are beneficial for the reasoning, the closure and the materialization method. The closure method splits up the inference tasks on several nodes for performing them in parallel. The applying of the inference rules to the data set of an ontology is enhanced by use of the closure method. It separates the execution of the inference rules on several nodes. Additionally, it becomes possible to split each inference process as well. The materialization stores the results of the closure method.
The closure approach has already been developed as a module implemented in Python[TM1]. It is a brute force implementation using the RDFLib module. The mentioned modules are usable on personal computers or on HPC systems. Weaver et al. (mentioned in 1.2) have proposed a closure experiment running 128 processes for performing inferences in parallel. The initial data size was 345 million RDF triples while the result produced by the closure method consists of 650 million RDF triples. When thinking of the Large Triple Store [1] the number of used RDF triples can become more than one trillion. This numerous amount of RDF triples implies the need of an effective closure method dealing with such data sets.
Weaver and Hendler provide the parallel RDFS inferencing algorithm described in the following:

Listing 5.1: Parallel RDFS Inferencing Algorithm by Weaver and Hendler

```
for i = 0 to p - 1 do
  T_Ai = \{ t | t ∈ T_A ∧ t ∉ T_Aj , ∀ j ≠ i \}
    T_i = T_Ai ∪ T_O ∪ T_rdfs
    foreach rule ∈ finite RDFS rules do
      repeat
      apply rule to T_i to get inferences
      add inferences to T_i
    until no new inferences
  end
end
return ⋃_{i=0}^{p-1} T_i
```

The outer loop denotes the parallelism where i is the rank of the processes. The provided input is a set of assertional triples T_A, a set of ontological triples T_O and a number of processes p. T_{rdfs} is the set of finite axiomatic triples. The provided output are all triples together with all inferences from the computation of the finite RDF closure

[1]Python Programming Language: http://www.python.org/

approach. Waever and Hendler are describing an algorithm regarding the input and output. The presented approach is usable for executing the inference rules in parallel. Thus the provided algorithm is enhanced to run the algorithm in parallel on several nodes. The following enhancement of the algorithm by Weaver and Hendler suggests how to improve it when distributing it on several nodes in parallel.

Listing 5.2: Enhanced Parallel RDF Inferencing Algorithm

```
for n = 1 to n = m do
  for i = 0 to p - 1 do
    T_Ai = { t | t ∈ T_A ∧ t ∉ T_Aj , ∀ j ≠ i }
      T_i = T_Ai ∪ T_O ∪ T_rdfs
      foreach rule ∈ finite RDFS rules do
        repeat
        apply rule to T_i to get inferences
        add inferences to T_i
      until no new inferences
    end
  end
  return ⋃_{i=0}^{p-1} T_i
end
```

The enhanced approach is extended by another outer loop where n is the selected node and m is the maximum of available nodes. However, each node is connected to several cores, able to perform several jobs, depending on the hardware infrastructure. Thus, the extension of the Weaver and Hendler approach is the possibility to distribute the processes on a large set of nodes. This proceeding is similar to the parallel execution of the ontology matching approaches dividing the proceeding in several tasks.

The materialization method stores the previously produced results. Thus, in the materialization phase it is decided which results to be stored. A common proceeding is to sore all produced generated facts but a distinction between facts relevant to a certain domain is recommended. Through this the results can be made more adequate according to the domain of interest, for instance, by selecting the results by a list of terms relevant to the domain.

5.4 Conclusions of Distributed Ontology Matching

The last sections have shown the applicability of the lexical, the taxonomy and the non-taxonomy ontology matching approaches by aligning them to a HPC infrastructure. For this, firstly the needed HPC resources are allocated by using a SLA based scheduling system. Following, the ontology matching is performed by running the parallel matching jobs using allocated nodes.

When thinking of applying a distributed reasoning method, an improvement by involving HPC resources can be reached for the reasoning through allocation of several computing nodes. Through this, it becomes possible to handle large data sets such as the assertional data occurring when the generation of inferences takes place. The reasoning is performed after the ontology matching was done in order to produce new facts out of the matching results. The matching tasks are performed as jobs scheduled by SLAs on HPC resources. Through this, bottlenecks and latencies occurring due to a bad distribution of reasoning tasks can be avoided. Nevertheless, the distribution of the reasoning tasks requires a fitting distribution taking care of relations between the divided tasks. The afterwards merge of the distributed produced results needs to be taken into account as well. In addition, a distributed reasoning makes only sense in case the amount of data and produced assertional data is more complex than the process of distribution itself. When thinking of the previously ontology matching, only relevant results are considered and thus, often a usual reasoning method is sufficient. The goal of this work is the ontology matching with the aim to increase the efficiency, especially for large data sets. Thus, the parallel execution of the matching tasks on a cluster is a usable approach enabling an expert to run reasoning queries based on the provided matching results being the individual knowledge base. The matching preconditions, the similarity value generation and the following result evaluation ensures a high quality of produced results. Through the ontology matching, relevant data are extracted out of a huge bunch of information. The extracted data are the foundation for reasoning tasks.

Chapter 6

Improvements for Use Case Scenarios

This chapter compares and evaluates the benefits of the ontology matching approach presented in section 4 and 5. Afterwards, the applicability of the results regarding the use case scenarios are investigated by consideration of the previously defined requirements in 2.

6.1 Advantages of the Distributed Ontology Matching

In the following the allocation of required computing resources, the ontology matching and the reasoning approach is summarized.

Allocation of Computing Resources Beside the described approach for an efficient allocation and reservation of HPC resources, a distributed reasoning is presented and the realization of the ontology matching approach by use of the available computing resources. The ontology matching approach is applicable for HPC environments, e.g. the HLRS ([164]) HPC infrastructure. The SLA based approach is the foundation for resource allocation as it is required in this work. The SLA based approach was successfully used for constructing SLAs. Several SLA level allow a priorization and scheduling of computing jobs by considering the required computing resources as well as requirements concerning the HPC infrastructure. Besides the SLA a scheduler is used to handle the computing jobs depending on the SLAs. In section 5.2.1 the use of SLAs combined with a scheduler was presented and successfully used for running computing jobs in a HPC environment.

Enhanced Distributed Ontology Matching The generation of the alignment list through usage of a priority ontology enables the reasoning on customized data sources

containing all relevant information from a previously performed selection of ontologies. However, a strategy for matching the target ontologies of the set and further performing a validity check of the matching results is elaborated in this work, with the aim to examine the ontology matching approach. For performing an adequate matching a similarity value is created to express the level of accordance between matched entities. The matching process is performed in two major steps, the preparation and the execution, that are subdivided.

- Preparation of similarity matching:
  - Identification of relevant ontologies;
  - Selection of relevant features of entities;
  - Definition of the search space.
- Execution of similarity matching:
  - Generation of the similarity value;
  - Interpretation of the generated similarity value.

During the preparation phase adequate ontologies are identified by an expert in order to create the set of target ontologies to be compared with the priority ontology. Afterwards the selection of the entities takes place for ensuring a matching of the entities of the priority ontology with the target ontologies from the known set. For each entity of the priority ontology one matching iteration is performed. This means, the amount of matching iterations grows significantly with every entity of the priority ontology. The next step is the definition of the search space for defining the number of neighboring entities taken into account for the matching. Due to the fact that the relation of a single entity with its neighboring entities might be not similar in different ontologies, it is quite important to define the depth level, the search space, for comparison of the relations between entities in different ontologies. The more deeply the search space is defined, the more matching process are performed. This leads to a higher expense for the matching process but it is a necessary step in order to match the relations between entities for ensuring a high quality of matching results.
The execution phase covers first the generation of the similarity value. The similarity value is created out of a number of different values that are generated from different similarity matching approaches such as features of the concepts and relations to the neighboring entities. The numbers of considered neighboring concepts are defined previously in the search space definition step. The second step during the execution phase is the interpretation of the similarity value. The similarity value is used for merging entities and into the matched entity in an alignment list. A high similarity value leads to a high probability of similarity. This probability of similarity measures the grade of similarity between matched concepts. After each matching step a simi-

larity value is produced used as a base for deciding to conclude with the matching or stop the matching process.
The presented ontology matching approach is distributed on HPC resources for increasing the overall matching effectiveness. Computing resources are allocated through the SLAs and a job scheduling. Thus computing resources are used for running matching jobs in parallel. The presented distribution approach considers the lexical, taxonomy and the non-taxonomy matching. For each matching step, the ontology matching is divided into several tasks running on the allocated computing nodes of a cluster (see section 5.3). This HPC resource allocation and the distribution enables an improved ontology matching when thinking of effectiveness and scalability depending on the used HPC infrastructure.

Quality of Matching Results The quality of matching results is ensured through the validity check. For this, vector based techniques are elaborated as an instrument to compare the matching results with a text document related to the topic of the use case scenario. This text is an HPC related text as it was presented in 2. The text document is provided by the expert for the automated validity check. This validity check document (VCD) contains text about the domain of the use case scenario and the content of selected ontologies. Through random indexing, the occurrence of a term being the matching result is evaluated with the VCD. The random indexing approach for performing the validity check is divided into several phases. However, in case the vector based approaches are used for making an automated assumption about the meaning of terms, these approaches are facing the problem of producing unreliable results or demanding very high additional effort from an expert. The validity check of the produced matching results is performed by use of an additional ontology, a domain ontology, for comparison of the matching results with the alignment list. The use of a domain ontology for proving the quality of the matching results is easy adaptable by use of the presented ontology matching approach. The matching result becomes the priority ontology and the domain ontology becomes the target ontology. Now, the ontology matching is performed as described in the previous chapters. The quality of the matching results is expressed in the generated similarity values. The higher the similarity values, the higher is the quality of the matching results. Thus, the comparison of the matching results with a domain ontology is an easy to use approach when following the presented ontology matching strategy. It is up to the expert to decide what kind of validation mechanism to use but both, the random indexing and the comparison with a domain ontology approach, are relevant for ensuring the quality of the matching results.

Ontology Based Reasoning The focus of this work is the distributed ontology matching but the goal of the matching approach is enabling an efficient ontology based reasoning. Thus, a reasoning method was introduced making use of distributed computing resources as well. The distribution of the reasoning method is beneficial

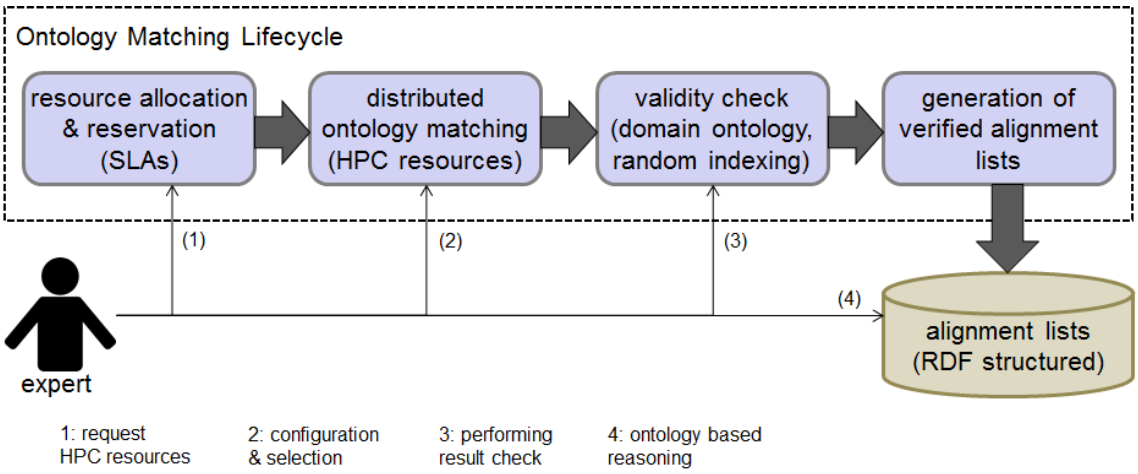

Figure 6.1: Overview of the Complete Ontology Matching Strategy

through the presented HPC resource allocation by use of SLAs. However, the usage of HPC resources requires a computing resource allocation by selecting nodes from a cluster. The proposed reasoning method is a forward-chaining reasoning strategy comprising the closure and the materialization method. The forward-chaining based reasoning approach enables an expert to receive needed facts from the previously matched result list immediately by use of the matching results. The foundation for this reasoning is the use of the matching results generated through the distributed ontology matching method.

6.2 Application of the New Proposed Strategy to the Use Case

The novel solution for ontology matching is demonstrated through both use case scenarios by adapting the presented improved ontology matching strategy. Additionally, the afterwards performed reasoning proves the usability of the generated results stored as alignment lists usable as ontologies. The produced alignment lists are RDF structured ontologies an therefore reasoning queries are performed by use of SPARQL.
The complete scenario can be divided into the matching based on allocated best fitting HPC resources and techniques for distributing the performed processes by use of the alignment lists. Though this, the whole process from receiving the required computing resource till receiving urgently needed information through reasoning by use of the matching results is covered by considering a restricted time frame for performing the matching in order to enable an afterwards performed reasoning. The general picture of the proposed strategy is presented in 6.1.

- The request and allocation for HPC resources;
- The configuration of the matching process and selection of a set of ontologies;
- The distributed ontology matching.
- The reasoning by use of the alignment lists generated through the distributed ontology matching.

The complete strategy is performed with the intervention of an expert. The experts requirements are considered in a convenient way by involving the expert in the matching process at the necessary points of intervention. The automated processing reasoning queries based on the alignment lists enables the generation of inferences from the ontologies. The time restriction of the urgent computing scenario is considered through the use of HPC resources and distribution techniques. Further, a validation of matching results with the aim to create a result list usable for reasoning methods is performed. The produced alignments are used to enhance the knowledge base of the expert by offering the alignments and providing semantically annotated data by using the RDF structure.

6.3 Result Evaluation

This section proves the described concepts of this work by using specific evaluation metrics. The concepts are proved by using the evaluation metrics for the prototypical implementations of the developments. The computing resource allocation by use of SLAs and job schedulers was already performed through research activities such as the SLA based job scheduling approach described in section 5.2.1. Thus, the evaluation will concentrate on the focused ontology matching approach by using HPC resources and performing the distribution approach on a single up to eight nodes. The evaluation metrics need to answer the following questions:

- Utility: who needs the information from this evaluation, and what information is needed?
- Feasibility: how much time and effort is required to make use of the presented concepts?
- Accuracy: are the results produced by the prototypical implementations usable and accurate?

When thinking of the accuracy of the matching results a domain ontology and a text based comparison is used to prove the quality of the produced results. The following

two sections demonstrate the described matching strategy for the ontology matching on a single machine (section 6.3.1) and in a distributed computing environment (section 6.3.2).

6.3.1 Result Evaluation: Ontology Matching - Single Machine

The tables 6.1, 6.2 and 6.3 demonstrate the proposed matching strategy by consideration of different amount of RDF structured data. First, the lexical, then the taxonomy and following the non-taxonomy matching approaches are evaluated. Regarding the evaluation it need to be considered that the lexical as well as the taxonomy matching procedure makes use of the same input RDF data. In addition, the last matching step, the non-taxonomy matching, makes use of the generated results from the previously taxonomy matching. This is caused through the fact that the non-taxonomy entities are the linking elements of a selected taxonomy, they are the labels between two entities. It follows that the data for the non-taxonomy matching is reduced set of data. Hence, the RDF data are less than the input data of the previous performed matching steps. Only in case same ontologies are compared, the input data for the non-taxonomy ontology will be the same.
For performing evaluations with RDF structured data the used HPC resources are listed in the appendix A. It is the configuration of the HLRS NEC NEHALEM cluster. Besides the tables, the results of the performed benchmark for evaluation are presented in the following. As presented the taxonomy matching is the most time consuming task closely followed by the lexical matching. The non-taxonomy matching depends strongly on the input data received by the previously performed taxonomy matching, the search space and the overall similarity of the ontologies. However, search space and similarity grade of used ontologies is crucial for the lexical and taxonomy matching as well but due to the fact that the non-taxonomy matching needs only to take care regarding the similar paths in an ontology it requires less time to execute the matching and additionally the data size is lower compared to the previous matching steps. For the presented results, the search space was set to the default value, the complete ontology. Furthermore, the taxonomy and the lexical matching approaches produce a set of result files as described in 4.1.4. Thus, an average value for data size for the non-taxonomy input data is used for each matching iteration and in addition, the selected file for the taxonomy and the non-taxonomy matching procedure is an average sample file from the previously generated data sets. For each data set one iteration with a single input file is performed.
A RDF graph is used for comparing and testing the execution times of the three matching procedures by use of different sample RDF data sets. The RDF data used for testing the implemented matching approach consist of RDF Triples being linked data used as ontologies. They are similar structured as already presented in the use case scenarios in section 2.2, 2.3 and 2.4. The RDF Triple consist of a subject, a predicate and an object such as presented in the following listing from a generated example

RDF Triple set.

```
<rdf:Description rdf:about="http://example5/ROOT">
...
    <j.1:value2>
      <rdf:Description rdf:about="http://example7y">
        <j.86:value0 rdf:resource="http://example2z"/>
        <j.107:value7 rdf:resource="http://example7x"/>
        <j.67:value8>
          <rdf:Description rdf:about="http://example2y">
            <j.69:value5 rdf:resource="http://example2z"/>
            <j.54:value6 rdf:resource="http://example6z"/>
...
          </rdf:Description>
        </j.67:value8>
        <j.74:value6 rdf:resource="http://example8x"/>
...
        <j.51:value4 rdf:resource="http://example7x"/>
      </rdf:Description>
    </j.1:value2>
    <j.175:value1 rdf:resource="http://example2x"/>
...
  </rdf:Description>
</rdf:RDF>
```

A subject is a description marked with the *rdf:about* tag, a predicate is the value of the linked object being, for instance, *j.86:value0* and the object is the linked resource *rdf:resource* tag. As demonstrated, the RDF triple are linked to other RDF Triples. The above selected example is presented in the XML notation and shown in the Annex B; additionally the annex contains same RDF data set in RDF Triple notation. It is a small data set consisting of about 300 RDF triples. The used data set is very close to the use case scenario data by using the RDF syntax being the foundation for the ontologies used in the three use case scenarios. Thus, the data set is representative for testing the provided matching approaches. As presented in the later performed tests most of the RDF triple data sets are larger amounts of data composed of thousands of RDF triple. When thinking of the three matching procedures, in this matching evaluation the non-taxonomy matching procedure includes only the matching of a single input file generated by the previous performed taxonomy matching. Thus, the used RDF triple set used for the non-taxonomy matching is a reduced data set composed out of the matching results from former matching iterations. This proceeding is the reason for using a RDF data set with a size in the range of kB. However, it is up to the expert to define the amount of input files for the non-taxonomy matching, e.g. 1 input file = N-TM Execution Time $*$ 1; 10 input files = N-TM Execution Time $*$ 10; 50 input files = N-TM Execution Time $*$ 50; and so forth. Thus, the tests for the non-taxonomy

RDF Data Amount (approx.)	Nodes	Time (min.)
100.000 – 17,5 MB	1	97,51
–	2	54,63
–	4	14,58
–	8	9,54
200.000 – 41,7 MB	1	212,23
–	2	118,73
–	4	37,67
–	8	25,00
300.000 – 76,6 MB	1	310,54
–	2	165,87
–	4	49,32
–	8	32,38
400.000 – 134,8 MB	1	487,81
–	2	277,38
–	4	85,77
–	8	60,83
500.000 – 193,5 MB	1	612,77
–	2	346,27
–	4	85,89
–	8	52,95

Table 6.1: Lexical Matching Evaluation

matching were performed for one matching iteration on several nodes (1 to 8). It is repeatable depending on the amount of individually needed matching iterations. The SD is set to the maximum of available entities of the source for the lexical matching and it is set to one for the taxonomy and the non-taxonomy matching for enabling a comparison with neighbored entities.

When looking at the measurements presenting the time needed for executing the ontology matching, it becomes obvious that there is a variability in the range of execution times, e.g. when executing a data set the matching is performed more than eight times faster than using a single node but performing the matching with another data set might reach a lower increase of performance. As exposed in the distributed matching approaches in 5.2.2, the data are distributed into data chunks while each identified matching causes an additional write operation. Further, the non-taxonomy matching is performed based on a reduced data set generated by the taxonomy matching procedure.
The execution time of the lexical and the taxonomy matching is decreased through the distribution of the data on several nodes and performing the matching in parallel. Same for the non-taxonomy matching but due to the very small amount of data it is only a minor improvement. The scalability is close to a super linear scaling caused

RDF Data Amount (approx.)	Nodes	Time (min.)
100.000 – 17,5 MB	1	105,42
–	2	67,54
–	4	18,16
–	8	12,59
200.000 – 41,7 MB	1	256,45
–	2	125,84
–	4	43,11
–	8	26,01
300.000 – 76,6 MB	1	331,98
–	2	178,76
–	4	52,86
–	8	36,45
400.000 – 134,8 MB	1	502,93
–	2	294,58
–	4	96,34
–	8	65,71
500.000 – 193,5 MB	1	648,33
–	2	362,64
–	4	91,73
–	8	61,56

Table 6.2: Taxonomy Matching Evaluation

Matching Iteration	Nodes	RDF Input (KB)	Time (min.)
1	1	324	34,65
–	2	324	11,64
–	4	324	8,95
–	8	324	7,32

Table 6.3: Non-Taxonomy Matching Evaluation (* = unquantifiable, time unit too low); reduced data set

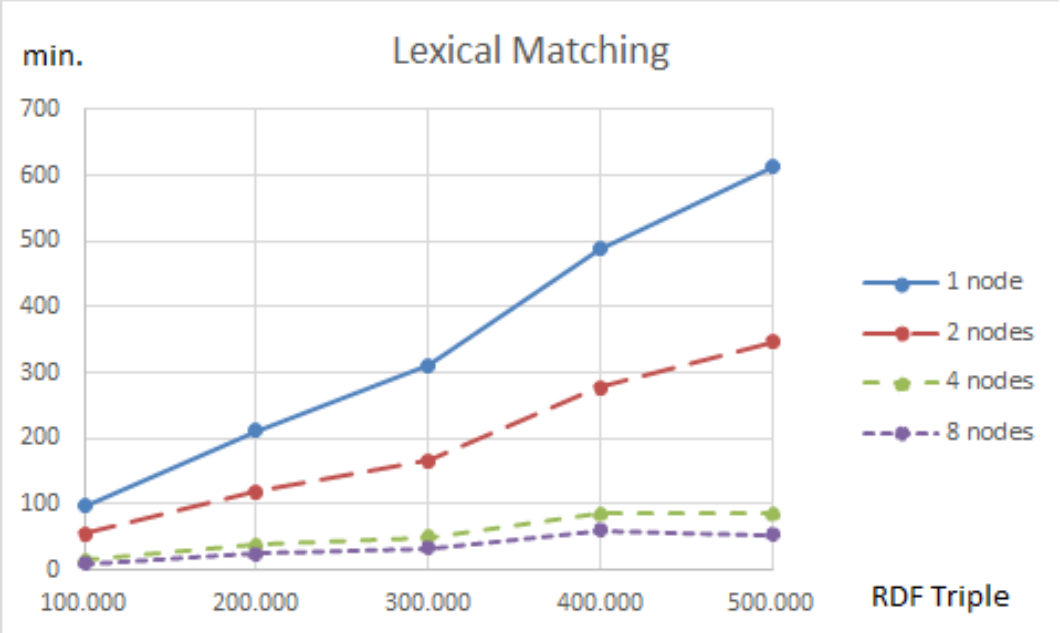

Figure 6.2: Matching Procedures Execution Times for Lexical Matching (minutes)

through the distribution of data chunks to the nodes.

When considering the ontology snippet from the beginning of this section 6.3.1, it has changed based on the previously performed matching. As presented in the following snippet the ontology provides only entities being convenient to an individual use case scenario. Thus, after each of the three matching iterations the similarity value is generated and marked in the RDF file. The following RDF example snippet is an intermediate file generated through the lexical matching procedure containing the accumulated similarities for the RDF triples. It is an example using a threshold for the similarity value of 0.60.

```
<rdf:Description rdf:about="http://example5/ROOT">1.00
...
    <j.1:value2>1.00
      <rdf:Description rdf:about="http://example7y">0.66
        <j.86:value0 rdf:resource="http://example2z"/>0.66
        <j.107:value7 rdf:resource="http://example7x"/>0.33
        <j.67:value8>0.33
          <rdf:Description rdf:about="http://example2y">0.66
            <j.69:value5 rdf:resource="http://example2z"/>0.00
            <j.54:value6 rdf:resource="http://example6z"/>1.00
...
          </rdf:Description>
        </j.67:value8>
```

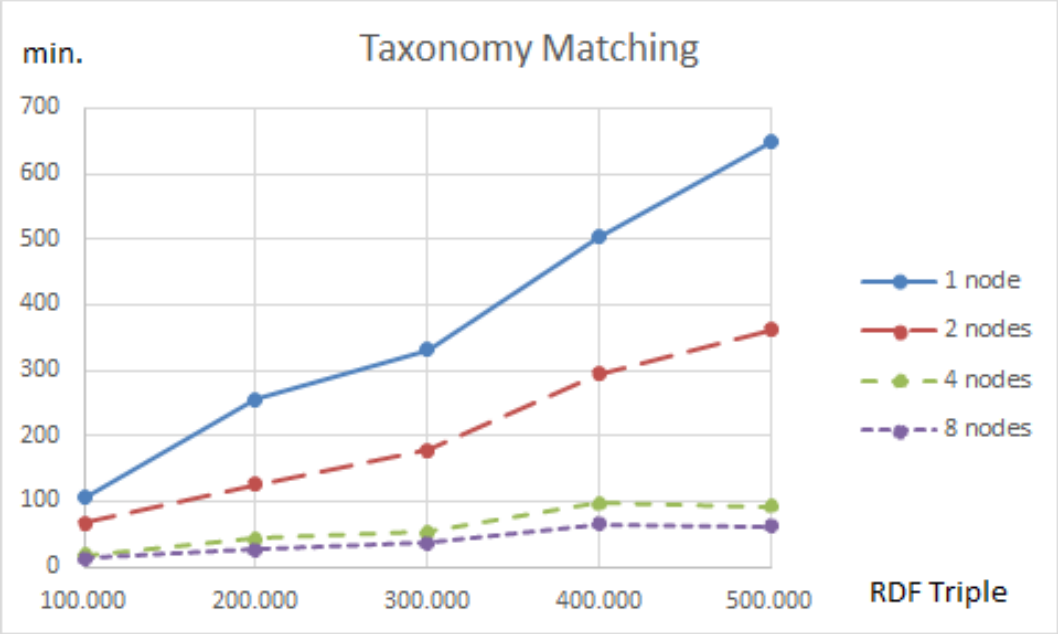

Figure 6.3: Matching Procedures Execution Times for Taxonomy Matching (minutes)

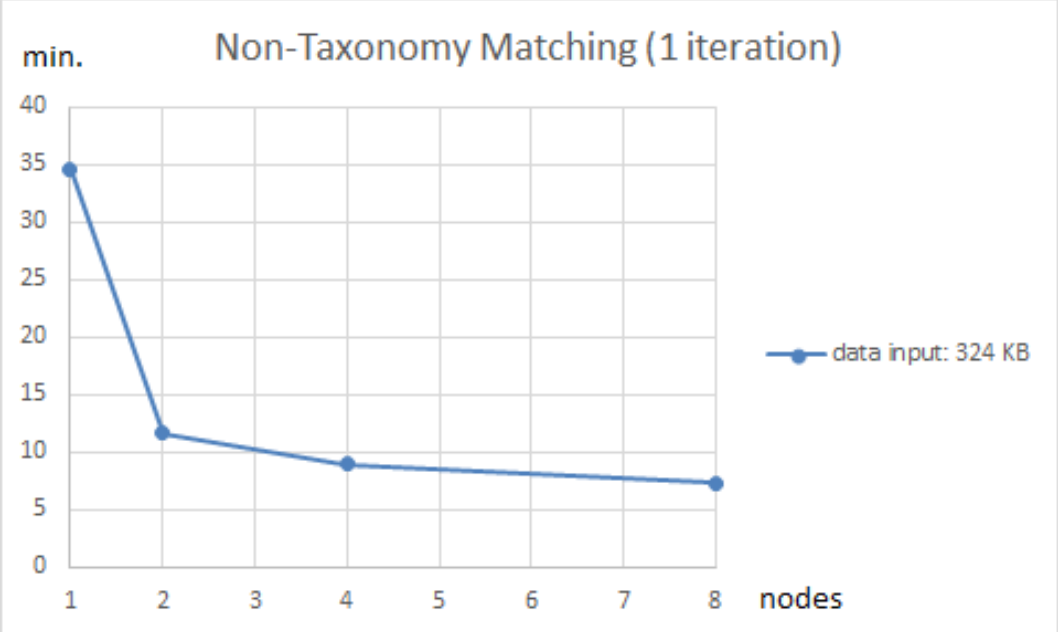

Figure 6.4: Matching Procedures Execution Times for Non-Taxonomy Matching (minutes), reduced data set

```
        <j.74:value6 rdf:resource="http://example8x"/>0.66
...
        <j.51:value4 rdf:resource="http://example7x"/>1.00
      </rdf:Description>
    </j.1:value2>
    <j.175:value1 rdf:resource="http://example2x"/>1.00
...
  </rdf:Description>
</rdf:RDF>
```

Out of the above presented intermediate RDF file with similarity values, an output file for an individual use case scenario is generated. Due to a threshold for the similarity value of 0.60, RDF triples marked with a similarity value lower than 0.60 are removed as presented in the following.

```
<rdf:Description rdf:about="http://example5/ROOT">
...
    <j.1:value2>
      <rdf:Description rdf:about="http://example7y">
        <j.86:value0 rdf:resource="http://example2z"/>
          <rdf:Description rdf:about="http://example2y">
            <j.54:value6 rdf:resource="http://example6z"/>
...
          </rdf:Description>
        <j.74:value6 rdf:resource="http://example8x"/>
...
        <j.51:value4 rdf:resource="http://example7x"/>
      </rdf:Description>
    </j.1:value2>
    <j.175:value1 rdf:resource="http://example2x"/>
...
  </rdf:Description>
</rdf:RDF>
```

The source ontology was reduced by removing data not reaching the threshold of the similarity value. Thus, the threshold is a parameter allowing an expert to adjust the level of accuracy for the generated matching results. Compared to the ontology data before the matching, the following data are the removed ones.

```
<j.107:value7 rdf:resource="http://example7x"/>
<j.67:value8>
<j.69:value5 rdf:resource="http://example2z"/>
```

However, the generated matching need to be proved. Hence the already mentioned VCDs are used whereas the amount of used VCDs is up to the expert and influences the validity prove of the matching results. As presented in figure 6.4, the occurrence of a term in the text is the amount of occurrence while the total occurrence is the percentage occurrence of the term calculated by the amount of all terms in the text divided through the occurrence amount. However, the percentage appearing of a text is generally very low.

Result Term	VCD Type	Occurrence in Text	Total Occurrence	Relevance
Search Term	HPC related	25	0,09 %	high
Search Term	HPC related	17	0.03 %	high
Search Term	HPC related	5	0.01 %	medium

Table 6.4: Occurrence of Result Terms Measured by a Set of VCDs

Additionally, an expert is enabled to proceed with an additional validity check by consideration of a domain ontology known trusted by the expert. Now, the expert compares the produced matching results with the trusted domain ontology by performing the previously executed matching steps but by using the result RDF files together with the domain ontology RDF file. With regard to the previously monitored execution times of the lexical, taxonomy and non-taxonomy matching there is a need of increasing the performance of the matching to reduce the overall execution time of the complete ontology matching process by including the validity prove. The overall execution time of the ontology matching procedure is as follows:

1. LM execution time + TM execution time + N-TM execution time

2. VCD execution time

3. Execution time of the domain ontology validity prove by performing step 1 again by use of the produced matching results and the domain ontology

In case of large data sets the execution of the presented complete matching strategy is time consuming. Hence, the presented distribution of the matching process is used and evaluated. As demonstrated, the data size of the RDF triples is usually low, e.g. kB or MB. Thus, the distributed matching is focused in the next section for analyzing the performance of the matching approaches.

6.3.2 Result Evaluation: Ontology Matching - HPC Environment

The data sets known from the evaluation for a single machine are used for testing the execution times of the presented matching procedures. As stated out, the eval-

uation environment is the NEC NEHALEM cluster of HLRS. Further, the NEC NEHALEM cluster makes use of a batch system for submitting jobs using Torque and Maui scheduler[165]. For performing the parallel execution of the matching jobs, FastMPJ [166] is used to implement the parallelization of the matching algorithms. The number of processes is defined by the run configutation of FastMPJ, e.g. us the *-np 2 -cp ${workspace_loc: ...};* command in the run configuration to run the job using two processes for performing the program identified through the classpath. Additionally, the amount of used cores is defined on the NEC NEHALEM cluster by running the *qsub* command, e.g. using two nodes for 30 minutes:

```
qsub -lnodes=2:nehalem,walltime=00:30:00 -I
```

Table 6.5 demonstrates the proposed matching strategy by consideration of RDF structured data by outlining the results. FastMPJ enables the distribution of the Java processes but it is limited to the configuration of the Java Virtual Machine (JVM) on the cluster. For testing the distribution on several nodes, the FastMPJ Java processes are distributing single processes on several nodes. The considered RDF data are large data sets.

RDF Data Amount (for LM and TM)	Nodes	LM Execution Time (min.)	TM Execution Time (min.)
100.000 – 17,5 MB	1	97,51	105,42
–	2	54,63	67,54
–	4	29,16	36,31
–	8	19,07	25,17
200.000 – 41,7 MB	1	212,23	256,45
–	2	118,73	125,84
–	4	75,34	86,21
–	8	49,99	52,01
300.000 – 76,6 MB	1	310,54	331,98
–	2	165,87	178,76
–	4	98,64	105,71
–	8	64,75	72,89
400.000 – 134,8 MB	1	487,81	502,93
–	2	277,38	294,58
–	4	171,54	192,67
–	8	121,65	131,42
500.000 – 193,5 MB	1	612,77	648,33
–	2	346,27	362,64
–	4	171,78	183,45
–	8	105,89	123,12

Table 6.5: Matching Procedure Evaluation using HPC Resources (2 nodes; 2 processes)

The non-taxonomy matching depends on the amount of input data produced by the lexical and the taxonomy matching as well as the individual selection of an expert. Thus it is not directly comparable with the taxonomy and the non-taxonomy matching. The measurements vary depending on the amount of performed operations being comparisons between entities and write operations. Assuming a small data set consisting of 300 RDF Triples the amount of comparison operations is presented in table 6.6.

Configuration	Lexical Matching	Taxonomy Matching	Non-Taxonomy Matching
The source and the target ontology consist of 300 RDF Triple assuming an average amount of characters for each entity of 8. The search depth is the maximum amount of available entities being 300.	Each character from each entity from the source is compared with a single character from each entity of the target. This concludes in $(300 \times 8) \times 300$ comparison operations being 720.000.	–	..
The source and the target ontology consist of 300 RDF Triple assuming an average amount of characters for each entity of 8 and an average linking of each entities to three neighbors. The search depth is set to one in order to investigate the direct neighboring entities.	–	Each character from each entity from the source is compared with a single character from each entity of the target. Due to the search depth of one each comparison with an entity causes the same proceeding for the neighbors. This concludes in $((300 \times 8) \times 3) \times 300$ comparison operations being 2.160.000.	–

The source ontology consist of 300 RDF Triple while the target is the produced result from the previous performed lexical and taxonomy matching. Thus, the target ontology is a reduced data set. The average amount of entities for the target ontology is assumed with 20 and the search space is set to one.	–	–	Each entity from the source is compared with similar entities from the target ontology including the neighbors. This concludes in $(300 \times 3) \times 300$ comparison operations being 270.000.

Table 6.6: Comparison Operations (300 RDF Triple)

When thinking of the amount of comparison operations it has emerged that it is highly dependent on the matching configuration of the expert. Increasing the SD will cause an exponentially increase of comparison operations. In addition, an identified match causes an additional write operation. Thus, the execution time of the matching methods are depending on the similarity grade of the source and the target ontology and the matching configuration defining the SD. Regarding the lexical matching the SD should be high (e.g. the maximum of entities of the source) to ensure that all similar entities of the target are identified. Regarding the taxonomy and the non-taxonomy matching a low SD already ensures a matching of the neighbors being an indicator for similarities.
The performed tests using HPC resources have shown a significant improvement considering the execution time (see figure 6.2, figure 6.3, figure 6.5) and figure 6.4). The three matching procedures were performed on the NEC NEHALEM cluster splitting up the input data for each matching step on several nodes. The successfully performed tests and the reached improved execution time has demonstrated the usability of the presented approach in a HPC environment. Further, the presented process is usable for several configuration scenarios through modifying the used amount of nodes and processes. However, the input data needs to be divided in as many chunks of input data as allocated processes. Otherwise, duplicated matches or inefficient usage of HPC resources might be the consequence. Further, the used implementation makes use of the JVM and FastMPJ. Thus, when increasing the amount of processes, the used JVM heap space needs to be aligned.
When thinking of large data sets, the processing of matching jobs in parallel on several nodes of a cluster has shown the applicability of the presented approach. Such large

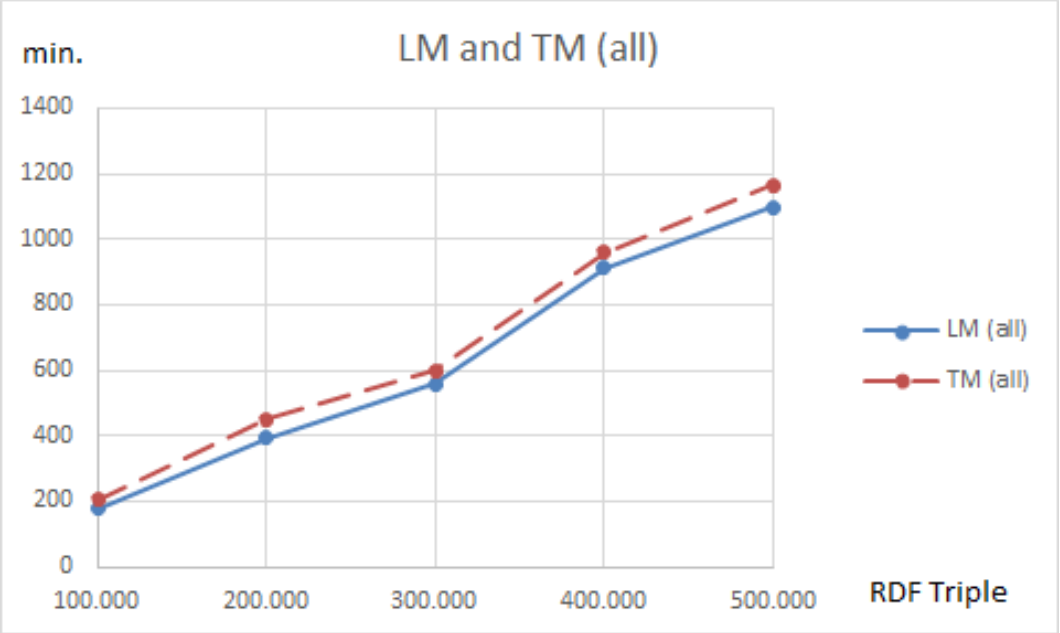

Figure 6.5: Summarized Matching Procedures Execution Times for Lexical and Taxonomy Matching (minutes)

data sets are processable in a time saving way, when distributing the matching jobs and process them parallel on the allocated computing nodes of the NEC NEHALEM cluster.

The comparison of the matching strategy performed by use of a single node in contrast to the HPC resources has demonstrated the improved performance when utilizing the distribution approach. Further, a resource allocation and a scheduling are beneficial methods for preparing the matching in a HPC environment. Both require human involvement and effort spent for such steps. Hence, the distributed ontology matching is beneficial when thinking of large data sets of RDF triples, but in case of small data sets of only hundreds of RDF triples the distribution is not necessary due to a small increasing of efficiency but in comparison a high increasing of additional effort for configuration issues. Actually, the execution time for only 100 RDF triples is higher when using the cluster. This is caused through extra effort required for the distribution of the processes on the nodes. Furthermore, aspects such as bandwidth, latencies and data transfer costs need to be considered using HPC systems.

6.4 Requirement Analysis

The previously defined use case requirements are used for measuring the evaluation results. In addition, evaluation criteria such as usability, feasibility and accuracy are

relevant for the use case evaluation.

- Utility: An expert requires information regarding specific materials for engineering techniques, skills, roles or competences related to individual tasks. When thinking of the law text scenario an expert requires knowledge regarding further laws and meanings. Generally, linked data need to be matched for generating a customized knowledge base for a specific scenario. These information are stored in provided ontologies composed by using the RDF standard. However, performing a query on large amount of data sets is inefficient and hence the ontology matching is performed for providing certain relevant data being the individually generated knowledge base usable for performing queries. Thus, the ontology matching approach is highly relevant when thinking or large amounts of semantic data.

- Feasibility: This work has focused on the ontology matching by consideration of large data sets. The evaluation has shown that a matching of large data sets becomes cost expensive but through using distribution techniques combined with HPC resource, the matching process is improved and feasible. Nevertheless, the evaluation has shown that the proposed distribution is beneficial for large data sets not small size data, e.g. consisting only of a few hundred RDF triples with a data size in the range of kB.

- Accuracy: The generated matching results are proved through a domain ontology and similarity values are generated during the matching for ensuring a high quality of the matching results. However, the setting of the threshold for the similarity values is up to the expert and therefore the quality of the result is configurable. Generally, an incorrect matching result is possible even by defining a high threshold. Thus, a validity check of the generated results through the expert is recommended.

The produced results are usable through performing SPARQL queries in order to receive certain information from the matching results. The structure of SPARQL queries is presented in the following by a SPARQL query searching for all lead alloys presented in an ontology for automobile material.

```
PREFIX am: <http://hlrs.de/ns/automobile-material#>
SELECT ?alloyName
WHERE
  { ?x  am:primaryStructure  "lead" .
        am:isPartOf  ?y .
    ?x  am:name  ?alloyName .
    ?y  am:type  am:alloy .
  }
```

Another example is given by an extended SPARQL query searching for all persons having a specific competence (e.g. ?competence = alloy) using an individually gener-

ated knowledge base for roles, skills and competences.

```
PREFIX ac: <http://hlrs.de/ns/automobile-competences#>
SELECT ?competence
WHERE
  { ?x  ac:competenceName  ?competence .
        ac:isCompetenceOf  ?y .
    ?y  ac:person .
  }
```

By using SPARQL queries certain information are usable based on the matching results. Through the use of the matching results the used ontology is already customized for a specified domain. The addressed requirements are listed in table 6.7.

Req. No.	Definition	Approach
R1	Domain knowledge	The ontology repository provides an expert with ontologies related to a domain of interest.
R2	Finding RDF structure	During the selection phase ontologies with adequate structures are identified. Further, using ontology converter support aligning ontology structures.
R3	Large data sets	The distributed matching approach using HPC resources enables a performant and time efficient ontology matching of large data sets.
R4	Validity check	The produced matching results are proved by use of a domain ontology and an additional domain related texts used as source of evidence. This increases the quality of produced results.
R5	Maintenance of result lists	A single ontology of the ontology set is used as base for all further matching steps, it is the priority ontology. The results are stored in an alignment list containing all results reaching the defined threshold of the similarity value. The alignment list is a single source of data usable for further reasoning tasks.
R6	Similarity levels	The proposed matching strategy supports reasoning through using the produced matching results. The evaluation has shown the usability of the produced ontology matching results by the similarity level definition indicating the operationality grade.
R7	Time efficiency	The distributed ontology matching approach using HPC resources improves the execution time drastically depending on the amount of allocated nodes.

R8	Information topicality	The produced information are valid until an ontology from the set is updated or not available anymore. This implies the re-calculation of the matching results.
R9	Computing resource allocation	SLAs are used to improve the queue based scheduling strategy for allocation of computing resources for computing jobs.
R10	Computing resource configuration	The needed computing resources are defined in the SLAs during the resource allocation phase.
R11	Homogenous data structure	Similar structures are handled by the presented ontology matching approach. Ontology transformation or a structure based selection of ontologies is not required for homogenous data structures.
R12	Highly linked data	The distributed matching approach considers linked data by matching selection regions of ontologies graph wise.
R13	Context sensitive relations	The taxonomy matching concentrates on the contexts of relations.
R14	Expert support	The proposed strategy supports an expert with needed information and proofing the matching quality through similarity levels (*R6*) and verifying the results with a source of evidence (*R4*). However, the approach requires an expert involvement for the final verification.
R15	Identifying RDF chunks	The RDF chunks can be identified through an expert or by automated selection through checking the required search depth.
R16	Cross-border matching	The search depth defines the RDF chunks and considers the relations and links to other RDF data chunks (see *R15*).
R17	Distributed Matching	The distributed RDF chunks consider *R11*, *R12* and *R13*.
R18	Transformation of ontologies	It is concentrated on adequate data structures (see *R2*) and further, ontology converters are used.
R19	Expert involvement	At each step of the ontology matching approach an expert is enabled to interact the process in case irregularities are observed.

Table 6.7: Requirement Result Analysis

The requirements of the use case scenarios are addressed through the distributed ontology matching approach being performed in a HPC system. The generated results are usable by performing reasoning tasks and supporting an expert with needed information. Due to required distribution effort, the distributed ontology matching focuses on large data sets. However, a residual risk of generating incorrect results remains even when defining a high similarity value threshold. Thus, a final result review of an expert is recommended for reducing this risk but finally, the produced results are usable through providing similarity levels and performing an automated validity check for ensuring a high result quality.

Chapter 7

Conclusions and Outlook

The underlying research question of this thesis was: *How to provide reliable semantic data in an efficient and time saving way for enabling reasoning?*. It was motivated by the possibility to adapt HPC resources for matching ontologies containing large scale semantic data sets not processable anymore in an efficient way due to the data size. During this work it turned out, the semantic data size is not a limiting factor for HPC resources but the calculation time for matching processes. While data size increase linear, the calculation time for matching processes increases exponential.

Summary The presented work describes and evaluates an automated ontology matching by using a domain dependent ontology set. Chapter 4 presents the ontology matching approach. A single ontology becoming the priority ontology is the source for further matching steps using target ontologies. The aim is to support an expert with required information by offering tailor-made matching results as alignment lists enabling reasoning tasks for individual use case scenarios. Additionally, the presented matching strategy offers highly reliable information by adapting a validity verification for the produced matching results. A probability value describing the level of similarity between the matched entities ensures the quality of the matching results. For this, a defined threshold needs to be achieved. Based on the matched results stored as alignment lists, the reasoning can be executed in a forward chaining way to ensure a quick response time when a query is submitted. The entire ontology matching process begins with a selection and identification of relevant ontologies. Afterwards, the matching process is configured by defining the search space and target ontologies. In addition, the required HPC resources are allocated for the matching processes using SLAs. Finally, the efficient distributed ontology matching presented in chapter 5 takes place by performing the lexical, the taxonomy and the non-taxonomy matching for generating matching results stored as alignment lists being used as ontologies. The selected ontologies and the produced alignment lists are RDF triple stores building the individual knowledge base used for domain specific queries. The matching is performed by consideration of similarity values expressing the grade of similarity between the matched entities. The produced matching results

are validated. More generally, the result of the ontology matching approach are ontologies usable as the foundation for performing reasoning tasks.
The presented strategy is applicable for time urgent computing scenarios by aligning the ontology matching and the reasoning approach to a distributed environment offering the required amount of computing resources. Especially, when thinking of the large data sizes of RDF triple stores the distribution of the matching processes on several nodes improves the effectiveness and enables an expert to receive required information in a short period of time. Using HPC resource for the presented ontology matching methods has emerged the scalability of the presented approach. Nevertheless, the matching configuration and the similarity grade of the used ontologies has an impact on the execution times. Further, matching procedures not utilizable on a personal computer due to exponentially growing sophisticated and expensive calculation times become feasible. Bringing HPC resources to the ontology matching domain improves the overall performance and enables new possibilities for ontology matching based on large scale data sets. Additionally, the reasoning produces high amounts of assertional data when new facts are generated at load time when following the forward chaining based reasoning approach. Such assertional data are covered by a distributed execution of inference rules at same time. However, before the reasoning takes place the alignment lists are generated through the focused automated matching algorithms. The matching approach enables an expert to configure the matching in accordance to the given scenario, e.g. identification of ontologies (see 4.1.1, selection of relevant entities (see 4.1.2) and the definition of the search space (see 4.1.3). For all three use case scenarios (automotive engineering [2.2], legal rule analysis [2.3] and linked web data [2.4]) relevant specific information for the selected domain of interest are provided. Thus, an individually generated knowledge base is provided for each use case scenario. Furthermore, the expert has the possibility to check the produced matching results after each matching step by validation of the produced similarity values. In general, the matching approach developed in this work by use of similarity values and HPC resource ensures a high quality of the produced results in a time saving way.

Limitations This work has proven the usability of an ontology matching approach developed for using it in a HPC environment. Thus, the matching processes can be distributed on several nodes in order to handle large scale data sets for increasing the overall effectiveness and execution time. However, a general issue when thinking of ontology matching, is the ontology structure and the used ontology language. The presented approach takes care of different ontology structures by presenting a transformer in order to align ontologies. Changing the structure can cause a loss of data or the generation of wrong data structures. It can become more critical when thinking of different ontology languages used for several ontologies selected for matching. However, translating the language of an ontology can also enhance the matching by translating the ontology into a more common language providing more target ontologies than available previously before the translation (e.g. translating from German

into English language). The presented transformer is an approach on how to handle ontologies with different structures but due to the fact that transforming ontologies is a distinct research field, this topic is beyond the scope of the ontology matching discussion in this work. Nevertheless, it is part of the preparation phase for the ontology matching process and therefore described briefly by presenting an ontology matching solution. The presented ontology matching approach considers RDF structured ontologies being RDF triple stores but by offering the presented transformation method the ontology matching approach is not restricted to a single ontology structure.
The goal of this work is the generation of matching results usable for reasoning queries. Thus, the focused issue is the ontology matching which is extensive discussed but the reasoning is based on the matching results. A reasoning can be performed by running SPARQL queries using the matching results. Besides the ontology matching topic, reasoning is another research field which is considered in the scope of this work due to the fact that reasoning is the cause for performing the ontology matching. However, the focus is the ontology matching approach. Nevertheless, when thinking of reasoning, several approaches are available. The approach presented and already performed by Weaver and Hendler is situated closely to the presented ontology matching approach through the applicability of parallel processing of the reasoning task on selected nodes and thus it is selected for demonstration issues.
Another important aspect is the validation of the matching results. The matching results can easily be validated by performing an ontology matching with a domain ontology by using the presented approach. However, the proposed random indexing can be added to the validation phase as well. Nevertheless, a residual risk of generating false matching results is unavoidable but it can be reduced to a minimum by adapting the similarity values and followed by this performing the validation. Finally, an expert needs to decide if the matching results are reliable but decision support is given through the mentioned validation methods.

Conclusions Beside the presented solution, a future research topic will be the analysis and execution of parallel reasoning approaches. This work gives an overview regarding a distributed reasoning strategy dealing with large data sets and high amounts of assertional data but the separation of the reasoning tasks on several nodes implies the challenge of storing the relations between the divided domains and in addition the risk of performing similar tasks in different nodes is given. Regarding the validation, trustworthiness of the produced matching results is proved through using a domain ontology as source of evidence to perform a validity check of the alignment list containing the matching results and applying random indexing methods. However, the use of machine learning techniques by use of specific features such as, for instance, linguistic, structural or even web features is a used method in order to increase the correctness of the matching results. Especially when thinking of the involvement of web features in the validation process, the WWW provides data usable as source of evidence. Additionally, using ontologies with different structure or different ontology languages is a wide research field requiring well defined strategies for transformation

of data and structures. In this scope, the transformation of ontologies is an interesting approach for the future.
Looking back at the results of this work, a distributed ontology matching approach using HPC resources was presented and tested successfully. The applicability of this approach is shown through developing the ontology matching approach based on related state of the art research. The ontology matching approach is distributed and processed in a HPC environment. Beyond the ontology matching, this work shows solutions for processing reasoning tasks based on the ontology matching results being the knowledge base. Generally, the aim to improve an ontology matching approach for increasing effective and scalable results, being reliable semantic data, is fulfilled. Thus, future research for processing of semantic data using HPC resources is a promising approach for handling large scale semantic data sets.

Bibliography

[1] The World Wide Web Consortium. The w3c large triple store page: http://www.w3.org/wiki/largetriplestores, 2011. (document), 1.2, 5.3.3

[2] The W3C. The w3c rdf page: http://www.w3.org/tr/rdf-concepts/, 2004. (document)

[3] The Semantic Web Activity Group of the W3C. The semantic web activity homepage: http://www.w3.org/2001/sw/, 2012. 1.1

[4] L. M. De Rijk. *Aristotle: General introduction, the works on logic*. Philosophia Antiqua. Brill, 2002. 1.1

[5] N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25–32, 1995. 1.1

[6] X. Su and J. A. Gulla. An information retrieval approach to ontology mapping. *Data & Knowledge Engineering*, 58:47–69, 2006. 1.1, 3.1.1, 3.4.1

[7] T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284(5), 2001. 1.1

[8] C. Ramesh and A. Gnanasekaran. Methodology based survey on ontology management. *International Journal of Computer Sciences & Engineering Survey (IJCSES)*, 1:1, 2010. 1.1, 1.2, 1.2

[9] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer, 2007. 1.1, 3.1.6

[10] M. Mao, Y. Peng, and M. Spring. Ontology mapping: as a binary classification problem. In *Proceedings of the 4th International Conference of Semantics, Knowledge and Grid (SKG)*, 2008. 1.1, 1.4.1, 3.1.8, 4.1.4

[11] A. Tenschert. Using similarity values for ontology matching in the grid. In *Proceedings of NDT 2010 Conference*, 2010. 1.1

[12] The W3C. The datasetrdfdumps wiki page: http://www.w3.org/wiki/datasetrdfdumps/, 2012. 1.1

[13] A. Maedche and S. Staab. Measuring similarity between ontologies. In *Proceedings of the 17th International Conference (EKAW)*, 2002. 1.1

[14] J. M. Taylor, V. Raskin, M. S. Petrenko, and C. F. Hempelmann. Multiple noun expression analysis: An implementation of ontological semantic technology. In *Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT)*, pages 517–524, 2010. 1.1

[15] M. Sahlgren and J. Karlgren. Automatic bilingual lexicon acquisition using random indexing of parallel corpora. *Natural Language Engineering, Special Issue on Parallel Texts*, 3:327–341, 2005. 1.2

[16] The LarKC Project Consortium. The larkc project - the large knowledge collider: http://www.larkc.eu/, 2011. 1.2, 3.2

[17] J. Weaver and J. Hendler. Parallel materialization of the finite rdfs closure for hundreds of millions of triples. In *8th International Semantic Web Conference (ISWC2009)*, 2009. 1.2

[18] A. Tenschert and A. Cheptsov. Effective ontology matching in high-performance computing environments. In *Proceedings of the SEMHE09 workshop. Part of the ECTEL Conference*, 2009. 1.2

[19] A. Tenschert, M. Assel, A. Cheptsov, and G. Gallizo. Parallelization and distribution techniques for ontology matching in urban computing environments. In *Proceedings of the Fourth International Workshop on Ontology Matching. Part of the ISWC Conference*, 2009. 1.2

[20] The Semantic Web Challenge Consortium. The semantic web challenge web page: http://challenge.semanticweb.org, 2015. 1.2

[21] The W3C. The large triple store homepage: http://www.w3.org/wiki/largetriplestores, 2011. 1.2

[22] The W3C. The large triple store homepage: http://www.w3.org/wiki/largetriplestores, 2011. 1.2

[23] The W3C. The large triple store homepage: http://www.w3.org/wiki/largetriplestores, 2011. 1.2

[24] A. Tenschert and P. Gilet. Reasoning on high performance computing resources. In *Proceedings of IMMM 2011 Conference*, 2011. 1.2

[25] The W3C. The extensible markup language homepage: http://www.w3.org/xml/, 2012. 1.4.1

[26] The W3C. The resource description framework homepage: http://www.w3.org/rdf/, 2012. 1.4.1

[27] The W3C. The rdf schema homepage: http://www.w3.org/tr/rdf-schema/, 2004. 1.4.1

[28] The W3C. The owl web ontology language homepage: http://www.w3.org/2004/owl, 2004. 1.4.1

[29] The W3C. The owl working group homepage: http://www.w3.org/2007/owl/wiki/owl_working_group, 2012. 1.4.1

[30] The W3C. The owl 2 web ontology language homepage: http://www.w3.org/tr/owl2-overview/, 2009. 1.4.1

[31] The W3C. The web service modeling ontology (wsmo) homepage: http://www.w3.org/submission/wsmo/, 2005. 1.4.1

[32] The SPARQL Working Group of the W3C. The sparql working group homepage: http://www.w3.org/2009/sparql/wiki/main_page, 2013. 1.4.1

[33] S. Powers. *Practical RDF*. O'Reilly & Associates, Inc., first edition edition, 2003. 1.4.1

[34] The Dublin Core Metadata Initiative Limited. The dublin core metadata initiative: http://dublincore.org/, 1994. 1.4.1

[35] T. Syldatke, W. Chen, J. Angele, A. Nierlich, and M. Ullrich. How ontologies and rules help to advance automobile development. In *Proceedings of the 2007 international conference on Advances in rule interchange and applications*, RuleML'07, pages 1–6. Springer-Verlag, 2007. 2.2.1

[36] M. Schraps, T. Ronneberger, and W. Golubski. Automotive engineering im enterprise-domain-network. In *INFORMATIK 2011 - Informatik schafft Communities*, volume 41. Jahrestagung der Gesellschaft fr Informatik, 2011. 2.2.1

[37] The Semantic Web Challenge Committee. Billion triple challenge 2012 dataset: http://km.aifb.kit.edu/projects/btc-2012/, 2012. 2.2.2

[38] ELSEVIER; Natural Language Processing Group; Karlsruher Institut for Technologie. The semantic web challenge: http://challenge.semanticweb.org/, 2012. 2.2.2

[39] R. Schönhof, A. Tenschert, and A. Cheptsov. Towards legal knowledge representation system leveraging rdf. In A.Cheptsov and C. Mavromoustakis, editors, *Proceedings of the Eighth International Conference on Advances in Semantic Processing*, SEMAPRO 2014. IARIA conference, 2014. 2.3.1

[40] L. Mehl. Automation in the legal world: From the machine processing of legal information to the law machine. In *Mechanisation of Thought Processes*, Legal Information to the Law Machine. Teddington Conference, 1958. 2.3.1

[41] G. Sartor, P. Casanovas, M. A. Biasiotti, and M. Fernndez-Berrera. *Approches to Legal Ontologies*. Springer Press, 2011. 2.3.1

[42] Federal Ministry of Justice and N. Mussett consumer protection, Germany. German civil code bgb: http://www.gesetze-im-internet.de/englisch_bgb/, July 2011. 2.3.1

[43] German Federal Parliament. Statistic of the bundestag: http://www.bundestag.de/, June 2014. 2.3.1

[44] R. Schönhof, A. Tenschert, and A. Cheptsov. Information extraction from unstructured texts by means of syntactic dependencies and constituent trees. In *Proceedings of the Ninth International Conference on Advances in Semantic Processing*, SEMAPRO 2015. IARIA conference, 2015. 2.3.2

[45] Ontotext AD. Linked life data (lld): http://linkedlifedata.com/, Nov. 2014. 2.4.1

[46] Linked Data Community. Linked data - connect distributed data across the web: http://linkeddata.org/, Nov. 2014. 2.4.1

[47] W3C Linked Open Data Community. Linked open data w3c sweo community project: http://www.w3.org/wiki/sweoig/taskforces/communityprojects/linkingopendata, Nov. 2014. 2.4.1

[48] Kno.e.sis (the Ohio Center of Excellence in Knowledge-enabled Computing). Linked sensor data: http://wiki.knoesis.org/index.php/ssw_datasets, Sept. 2010. 2.4.2

[49] H. Patni, C. Henson, and A. Sheth. Linked sensor data. In *Proceedings of Collaborative Technologies and Systems (CTS 2010)*, 2010. 2.4.2

[50] X. Tao, Y. Li, and R. Nayak. Ontology mining for semantic interpretation of information needs. In *Proceedings of the 2nd International Knowledge Science, Engineering and Management (KSEM) conference*, 2007. 3.1

[51] J. Karlgren and M. Sahlgren. From words to understanding. In *Foundations of*

Real-World Intelligence, pages 294–308. Uesaka, Y., Kanerva, P., Asoh, H., 2001. 3.1.1, 3.1.2, 3.1.2

[52] R. Baeza-Yates, B. A. N. Ribeiro, and B. Ribeiro-Neto. *Modern information retrieval*. ACM Press books. ACM Press, 1999. 3.1.1

[53] T. K. Landauer and S. T. Dumais. A solution to platos problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, 104(2):211240, 1997. 3.1.1

[54] G. Salton, A. Wong, and C. S. Yang. A vector space model for information retrieval. *Communications of the ACM*, 18(11):613–620, 1975. 3.1.1

[55] H. Schütze and J. O. Pedersen. A cooccurrence-based thesaurus and two applications to information retrieval. *Information Processing and Management*, 33(3):307318, 1997. 3.1.1

[56] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin, Special Issue on Text and Databases*, 24(4), 2001. 3.1.1

[57] S. E. Robertson. The probabalistic ranking principle in ir. *Journal of Documentation*, 33:294–304, 1977. 3.1.1

[58] The Infomix Project Consortium. *Review on models and systems for information integration*. Universita di Roma - La Sapienza, 2002. D1.1 from the Infomix project. 3.1.1, 3.1.10, 3.4.1

[59] N. Chatterjee and S. Mohan. Discovering word senses from text using random indexing. In *Proceedings of the 9th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, 2008. 3.1.2

[60] M. Sahlgren. An introduction to random indexing. In *Proceedings of the 7th International Conference on Terminology and Knowledge Engineering (TKE)*, 2005. 3.1.2

[61] T. K. Landauer, P. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998. 3.1.2

[62] M. Sahlgren. *The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. PhD thesis, Stockholm University, 2006. 3.1.2

[63] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *KDD-96 Proceedings*, 1996. 3.1.3

[64] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 3 edition, 2011. 3.1.3

[65] M. Kamber J. Han and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3 edition, 2011. 3.1.3

[66] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996. 3.1.3

[67] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 3 edition, 2003. 3.1.3

[68] G. Piatetsky-Shapiro. Knowledge discovery in real databases: A report on the ijcai-89 workshop. *AI Magazine*, 11(5):68–70, 1991. 3.1.3

[69] C. Glymour, D. Madigan, D. Pregibon, and P. Smyth. Statistical inference and data mining. *Communications of the ACM*, 39 (11):35 – 41, 1996. 3.1.3

[70] E. Eskin. Anomaly detection over noisy data using learned probability distributions. In *In Proceedings of the International Conference on Machine Learning*, pages 255 – 262. Kaufmann, M., 2000. 3.1.3.1

[71] A. K. Jain, M. N. Murty, and P. J. Flynn. *Data Clustering: A Review*. ACM Computing Surveys, 1999. 3.1.3.1

[72] A. E. Gelfand. Gibbs sampling. *Journal of the American Statistical Association*, 452, 1995. 3.1.3.2

[73] A. E. Gelfand and A. F. M. Smith. Sampling based approaches to calculating marginal densities. *Journal of American Statistical Association*, 85:398 – 409, 1990. 3.1.3.2

[74] W. K. Newey and D. McFadden. Chapter 36 large sample estimation and hypothesis testing. In R. F. Engle and D. L. McFadden, editors, *Handbook of econometrics*, volume 4 of *Handbook of Econometrics*, pages 2111 – 2245. Elsevier, 1994. 3.1.3.3

[75] W. K. Newey and K. D. West. Hypothesis testing with efficient method of moments estimation. *International Economic Review*, 28:777 787, 1988. 3.1.3.3

[76] C. Glymour, D. Madigan, D. Pregibon, and P. Smyth. Statistical themes and lessons for data mining. *Data Mining and Knowledge Discovery*, 1:11 – 28, 1997. 3.1.3.4

[77] J. Doyle. Rational decision making. *The MIT Encyclopedia of the Cognitive Sciences*, pages 701 – 703, 1998. 3.1.3.5

[78] U. M. Fayyad, D. Haussler, and Z. Stolorz. Kdd for science data analysis; issues and examples. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996. 3.1.4

[79] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, 13, No 3:57 – 70, 1992. 3.1.4

[80] J. Zhang. Ontology and the semantic web. In *Proceedings of the North American Symposium on Knowledge Organization (NASKO)*, 2007. 3.1.5

[81] A. Gal and P. Shvaiko. Advances in web semantics i. *Lecture Notes in Computer Science*, 4891/2009:176–198, 2009. 3.1.5

[82] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic matching. In *Encyclopedia of Database Systems*, pages 2561–2566. L. Ling and M. Tamer, 2009. 3.1.5

[83] J. Huang, D. Dou, L. He, J. Dang, and P. Hayes. Ontology-based knowledge discovery and sharing in bioinformatics and medical informatics: A brief survey. In *Proceedings of the 7th Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 2203 – 2208, 2010. 3.1.5

[84] P. Shvaiko and J. Euzenat. Ten challenges for ontology matching. In *Proceedings of the 7th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, 2008. 3.1.5

[85] Ontology Alignment Evaluation Initiative. Ontology alignment evaluation initiative - home page: http://oaei.ontologymatching.org/, 2012. 3.1.5

[86] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18:1:1–31, 2003. 3.1.5, 3.1.9

[87] M. Sabou, M. Fernandez, and E. Motta. Evaluating semantic relations by exploring ontologies on the semantic web. In *Proceedings of the 14th International Conference on Applications of Natural Language to Information Systems (NLDB)*, 2010. 3.1.6, 4.2.3

[88] G. Pirró and J. Euzenat. A semantic similarity framework exploiting multiple parts-of speech. In *Proceedings of the On the Move to Meaningful Internet Systems Conference(OTM)*, 2010. 3.1.6

[89] M. Niepert, C. Meilicke, and H. Stuckenschmidt. A probabilistic-logical frame-

work for ontology matching. In *Proceedings of the American Association for Artificial Intelligence (AAAI)*, 2010. 3.1.6

[90] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006. 3.1.6

[91] R. L. Cilibrasi and P. M. B. Vitanyi. The google similarity distance. *Knowledge and Data Engineering, IEEE Transactions on*, 19(3):370 –383, march 2007. 3.1.6

[92] R. Gligorov, W. Kate, Z. Aleksovski, and F. v. Harmelen. Using google distance to weight approximate ontology matches. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 767–776, 2007. 3.1.6

[93] D. Milne and I. H. Witten. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. *Proceeding of the AAAI Workshop*, 2008. 3.1.6

[94] G. Pirró and J. Euzenat. A feature and information theoretic framework for semantic similarity and relatedness. In *Proceedings of the International Semantic Web Conference*, 2011. 3.1.6

[95] IM@OAEI2011. Instance matching homepage: http://www.instancematching.org/, 2011. 3.1.7

[96] D. Engmann and S. Massmann. Instance matching with coma++. In *In Proceedings of the BTW 2007 Workshop*, 2007. 3.1.7

[97] A. Ferrara, D. Lorusso, S. Montanelli, and G. Varese. Towards a benchmark for instance matching. In *Proceedings Ontology matching 2008*, 2008. 3.1.7

[98] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997. 3.1.8

[99] M. Bhatt, A. Flahive, C. Wouters, W. Rahayu, D. Taniar, and T. Dillon. A distributed approach to sub-ontology extraction. In *In Proceedings of the 18th International Conference on Advances Information Networking and Applications*, 2004. 3.1.9

[100] L. Serafini, A. Borgida, and A. Tamilin. Aspects of distributed and modular ontology reasoning. In *In Proceedings of the 19th International Joint Conferences on Artificial Intelligence*, pages 570–575, 2005. 3.1.9

[101] D. Calvanese, G. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proceedings of the International*

Conference on Knowledge Representation and Reasoning (KR)., pages 2–13, 1998. 3.1.10, 3.4.1

[102] Advances in Information Systems Research Laboratory. The agreementmaker homepage: http://agreementmaker.org/, 2011. 3.2

[103] I. F. Cruz, F. P. Antonelli, and C. Stroe. Agreementmaker: Efficient matching for large real-world schemas and ontologies. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB)*, 2009. 3.2

[104] N. F. Noy and M. A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Conference Proceedings of the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, 2001. 3.2

[105] M. Musen and N. Noy. Anchor-prompt: Using a non-local context for semantic matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 2001. 3.2

[106] INRIA. The aroma project homepage: http://aroma.gforge.inria.fr/, 2009. 3.2

[107] Y. R. Jean-Mary, E. P. Shironoshita, and M. R. Kabuka. Ontology matching with semantic verification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):235 – 251, September 2009. 3.2

[108] P. Jain, P. Hitzler, A. P. Sheth, K. Verma, and P. Z. Yeh. Ontology alignment for linked open data. In *Proceedings of the 9th international semantic web conference on The semantic web*, volume Part I of *ISWC'10*, pages 402–417, Berlin, Heidelberg, 2010. Springer-Verlag. 3.2

[109] M. S. Lacher and G. Groh. Facilitating the exchange of explicit knowledge through ontology mappings. In *In Proceedings of the 14th Int. FLAIRS Conference*, pages 305–309. AAAI Press, 2001. 3.2

[110] AI Laboratory Knowledge Systems. The chimaera homepage: http://www.ksl.stanford.edu/software/chimaera/, 2005. 3.2

[111] D. McGuiness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning.*, 2000. 3.2

[112] SID Group. The cider homepage: http://sid.cps.unizar.es/semanticweb/alignment/, 2008. 3.2

[113] H. H. Do and E. Rahm. Coma: A system for flexible combination of schema

matching approaches. In *Proceedings of the 28th International Conference on Very Large Databases*, pages 610–621, 2002. 3.2

[114] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *Proceedings of ACM SIGMOD International Conference on Management of Data.*, pages 906–908, 2005. 3.2

[115] Database Working Group Leipzig University of Leipzig. Coma 3.0: http://dbs.uni-leipzig.de/research/coma.html, 2012. 3.2

[116] Temporal Knowledge Bases Group. The contentmap homepage: http://krono.act.uji.es/people/ernesto/contentmap, 2000-2012. 3.2

[117] Advanced Knowledge Technologies (AKT). The crosi project homepage: http://www.aktors.org/crosi/intro, 2004-2005. 3.2

[118] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Databases.*, pages 49–58, 2001. 3.2

[119] M. Nagy, M. Vargas-Vera, and E. Motta. Dssim - managing uncertainty on the semantic web. In *The 2nd International Workshop on Ontology Matching (OM-2007)*, Nov. 2007. 3.2

[120] W. Hu, G. Cheng, D. Zheng, X. Zhong, and Y. Qu. The results of falcon-ao in the oaei 2006 campaign. In *Proceedings of the International Workshop on Ontology Matching*, 2006. 3.2

[121] G. Stumme and A. Maedche. Fca-merge: Bottom-up merging of ontologies. In *In Proceedings of IJCAI*, pages 225–230, 2001. 3.2

[122] The Semantic Web Consortium. Foam: http://semanticweb.org/wiki/foam, Feb. 2012. 3.2

[123] A. H. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the 11th International WWW Conference.*, pages 662–673, 2002. 3.2

[124] K. Kotis and G. A. Vouros. The hcone approach to ontology merging. In *In Proceedings of the 21st International Symposium on Computer Architecture*, pages 137–151. IEEE Computer Society Press, 2004. 3.2

[125] The Apache Software Foundation. Jena semantic web framework: http://jena.sourceforge.net/index.html, 2012. 3.2

[126] P. Wang and B. Xu. Lily: Ontology alignment results for oaei 2009. In *Proceedings of the OAEI 2009*, 2009. 3.2

[127] The MAFRA Project Consortium. The mafra toolkit homepage: http://mafra-toolkit.sourceforge.net/, 2006. 3.2

[128] A. Maedche, B. Motik, S. Nuno, and R. Volz. Mafra - a mapping framework for distributed ontologies. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management, Ontologies and the Semantic Web*, EKAW '02, pages 235–250, London, UK, 2002. Springer-Verlag. 3.2

[129] The Onto Builder Project Consortium. The onto builder project homepage: http://ontobuilder.bitbucket.org/, 2006. 3.2

[130] The OntoMediate Project Consortium. Ontomediate: Ontological mediation and semantic gateways for domain/enterprise translations: http://www.ecs.soton.ac.uk/research/projects/466, 2009. 3.2

[131] The plugIT Project Consortium. The plugit project: http://plug-it.org, 2011. 3.2, 3.2

[132] P. Gilet, A. Tenschert, and R. Kübert. Use of graphical modelling, semantics and service level agreements for high performance computing. In *Proceedings of the eChallenges 2011 Conference.*, 2011. 3.2

[133] HLRS University of Stuttgart. Hlrs - online proposal submission: http://www.hlrs.de/userprojects/ops/, 2012. 3.2

[134] The (GRAAP) WG of the OGF. Web service agreement specification (ws-agreement): http://www.ogf.org/documents/gfd.107.pdf, 2007. 3.2

[135] B. Koller. *Enhanced SLA Management in the High Performance Computing Domain*. PhD thesis, HLRS - High Performance Computing Center Stuttgert, University of Stuttgart, 2010. 3.2

[136] N. Noy and M. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the 12th IAAI.*, pages 450–455, 2000. 3.2

[137] The Protégé PRoject Consortium. The protégé homepa: http://protege.stanford.edu/, 2012. 3.2

[138] The RiMOM Project Consortium. Risk minimization based ontology mapping (rimom): http://keg.cs.tsinghua.edu.cn/project/rimom/, 2006. 3.2

[139] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, 2002. 3.2

[140] B. Alexe, L. Chiticariu, and W.-C. Tan. Spider: a schema mapping debugger. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 1179–1182. VLDB Endowment, 2006. 3.2

[141] The Open Linguistics Consortium. Open linguistics: http://linguistics.okfn.org/resources/llod/, 2014. 3.2

[142] The BabelNet Consortium. Babelnet 2.0 web site: http://babelnet.org/, 2014. 3.2

[143] The GATE Consortium. Gate - general architecture for text engineering: https://gate.ac.uk/, 2014. 3.2

[144] TC project at the Institute for Computational Linguistics of the University of Stuttgart. Treetagger: http://www.cis.uni-muenchen.de/ schmid/tools/treetagger/, 2014. 3.2

[145] The Text2Onto consortium. Text2onto - a framework for ontology learning and data-driven change discovery: http://ontoware.org/projects/text2onto, 2014. 3.2

[146] The WordNet consortium. Wordnet - a lexical database for english: http://wordnet.princeton.edu/, 2013. 3.2

[147] M. Sahlgren. *The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. PhD thesis, Stockholm: Institutionen fr lingvistik, 2006. 3.4.2

[148] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of AAAI Conference*, volume 6, 2006. 3.4.2

[149] M. T. Pilehvar, D. Jurgens, and R. Navigli. Align, disambiguate and walk: A unified approach for measuring semantic similarity. In *ACL (1)*, pages 1341–1351, 2013. 3.4.3

[150] R. Snow, S. Prakash, D. Jurafsky, and A. Y. Ng. Learning to merge word senses. In *EMNLP-CoNLL*, volume 2007, pages 1005–1014, 2007. 3.4.3

[151] W. Hu and Y. Qu. Falcon-ao: A practical ontology matching system. *Web Se-*

mantics: Science, Services and Agents on the World Wide Web, 6(3):237–239, 2008. 3.4.4

[152] M. F. Husain, P. Doshi, L. Khan, and B. Thuraisingham. Storage and retrieval of large rdf graph using hadoop and mapreduce. In *Cloud computing*, pages 680–686. Springer, 2009. 3.4.4

[153] P. Shvaiko and J. Euzenat. Ontology matching: state of the art and future challenges. *Knowledge and Data Engineering, IEEE Transactions on*, 25(1):158–176, 2013. 3.4.4

[154] U. Schöning J. Kobler and J. Torán. *The graph isomorphism problem: its structural complexity*. Springer Science & Business Media, 2012. 3.4.4

[155] Q. Fang, Y. Zhao, G. Yang, and W. Zheng. Scalable distributed ontology reasoning using dht-based partitioning. In *The Semantic Web*, pages 91–105. Springer, 2008. 3.4.5

[156] A. Flahive, D. Taniar, W. Rahayu, and B. O. Apduhan. A methodology for ontology update in the semantic grid environment. *Concurrency and Computation: Practice and Experience*, 27(4):782–808, 2015. 3.4.5

[157] J. Euzenat and P. Valtchev. Similarity-based ontology alignment in owl-lite. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI)*, page 333337, 2004. 4.1.4

[158] A. Farooq, M. J. Arshad, and A. Shah. A layered approach for similarity measurement between ontologies. *Journal of American Science*, 6:69–77, 2010. 4.1.4

[159] The W3C. W3c recommendation: Rdf vocabulary description language 1.0: Rdf schema: http://www.w3.org/tr/rdf-schema/, February 2004. 4.1.4.1, 4.1.4.1

[160] A. Tenschert and R. Kübert. Sla based job submission and scheduling with the globus toolkit 4. *The Computer Science Journal*, 13(4):183–204, 2012. 5.2.1

[161] Argonne National Laboratory. Openpbs public home: http://www.mcs.anl.gov/research/projects/openpbs/, 2004. 5.2.1

[162] Adaptive Computing. Torque resource manager: http://www.adaptivecomputing.com/products/open-source/torque/, 2012. 5.2.1

[163] TOP500.Org Authors. Top 500 supercomputer sites: http://www.top500.org/lists/, 2013. 5.3.1

[164] HLRS University of Stuttgart. Hlrs - high performance computing center stuttgert, university of stuttgart: http://www.hlrs.de/. 6.1

[165] Inc. Adaptive Computing. Adaptive computing, inc. web site: http://www.adaptivecomputing.com/products/open-source/maui/, 2013. 6.3.2

[166] The FastMPJ Consortium. The fastmpj web site: http://fastmpj.com/, 2013. 6.3.2

Appendix

Appendix A

Cluster Configuration

Configuration of the NEC NEHALEM cluster at HLRS:

- 700 compute nodes are of type NEC HPC-144 Rb-1 Server
  - dual CPU compute nodes: 2x Intel Xeon X5560 Gainestown
    - 4 cores, 8 threads
    - 2.80 GHz (3.20 Ghz max. Turbo frequency)
    - 8MB L3 Cache
    - 1333 MHz Memory Interface, 6.4 GT/s QPI
    - TDP 95W, 45nm technology
    - Nehalem microarchitecture
  - standard: 12 GB RAM
  - 36 nodes upgraded to 24GB, 48GB, 128GB or 144GB RAM
  - 32 compute nodes have additional Nvidia Tesla S1070 GPU's installed
- Pre- & Postprocessing node
  - 8x Intel Xeon X7542 6-core CPUs with 2.67GHz (8*6=48 Cores)
  - 1TB RAM
  - shared access

- Node Upgrade:
  - 180 nodes Dual Intel 'Sandy Bridge' E5-2670
  - 2.6 Ghz, 8 Cores per processor, 16 Threads
  - 4 memory channels per processor, DDR3 1600Mhz memory
  - 12 nodes with 64GB RAM
  - 168 nodes with 32GB RAM
  - QDR Mellanox ConnectX-3 IB HCAs (40gbit)
- Additional large memory nodes:
  - 10 nodes Quad Socket AMD Opteron 6238
  - 2.6 Ghz, 12 cores per processor
  - 4 memory channels per processor, DDR3 1600Mhz memory
  - 256GB RAM
  - QDR Mellanox ConnectX-2 IB HCAs (40gbit)
- network: InfiniBand Double Data Rate:
  - switches for interconnect: Voltaire Grid Director 4036 with 36 QDR (40Gbps) ports (6 backbone switches)

Appendix B

Example Data Set - RDF Triple

For demonstration purposes an auto-generated data set consisting of about 300 RDF Triple is presented in the following.

XML notation:

```
<rdf:RDF
    xmlns:j.0="http://example6/5/localName8/"
    xmlns:j.2="http://example1/2/localName5/"
    xmlns:j.1="http://example5/5/localName5/"
    xmlns:j.6="http://example2/1/localName6/"
    xmlns:j.5="http://example6/1/localName4/"
    xmlns:j.4="http://example4/6/localName7/"
    xmlns:j.3="http://example4/1/localName7/"
    xmlns:j.7="http://example5/6/localName1/"
    xmlns:j.9="http://example7/2/localName5/"
    xmlns:j.8="http://example9/7/localName9/"
    xmlns:j.10="http://example0/5/localName8/"
    xmlns:j.12="http://example2/4/localName8/"
    xmlns:j.11="http://example3/8/localName9/"
    xmlns:j.13="http://example4/4/localName5/"
    xmlns:j.14="http://example3/4/localName2/"
    xmlns:j.15="http://example7/0/localName1/"
    xmlns:j.16="http://example8/2/localName3/"
    xmlns:j.17="http://example9/8/localName1/"
    xmlns:j.18="http://example2/6/localName4/"
    xmlns:j.19="http://example3/5/localName7/"
    xmlns:j.20="http://example9/0/localName6/"
    xmlns:j.21="http://example2/5/localName0/"
    xmlns:j.22="http://example5/9/localName5/"
    xmlns:j.23="http://example0/0/localName2/"
    xmlns:j.24="http://example2/3/localName3/"
```

```
xmlns:j.25="http://example6/5/localName9/"
xmlns:j.26="http://example7/3/localName2/"
xmlns:j.27="http://example8/1/localName6/"
xmlns:j.28="http://example8/6/localName4/"
xmlns:j.29="http://example1/6/localName2/"
xmlns:j.30="http://example3/9/localName2/"
xmlns:j.32="http://example6/8/localName8/"
xmlns:j.31="http://example5/1/localName7/"
xmlns:j.33="http://example0/8/localName5/"
xmlns:j.34="http://example6/2/localName8/"
xmlns:j.35="http://example4/3/localName2/"
xmlns:j.37="http://example4/9/localName6/"
xmlns:j.36="http://example8/2/localName2/"
xmlns:j.38="http://example3/4/localName3/"
xmlns:j.39="http://example4/5/localName5/"
xmlns:j.41="http://example0/0/localName1/"
xmlns:j.40="http://example3/7/localName4/"
xmlns:j.42="http://example2/5/localName1/"
xmlns:j.43="http://example7/1/localName6/"
xmlns:j.44="http://example7/3/localName6/"
xmlns:j.48="http://example4/8/localName6/"
xmlns:j.47="http://example7/7/localName5/"
xmlns:j.46="http://example7/4/localName1/"
xmlns:j.45="http://example2/1/localName9/"
xmlns:j.49="http://example6/1/localName6/"
xmlns:j.50="http://example0/4/localName7/"
xmlns:j.51="http://example5/0/localName5/"
xmlns:j.52="http://example0/9/localName0/"
xmlns:j.53="http://example5/1/localName0/"
xmlns:j.54="http://example2/4/localName6/"
xmlns:j.55="http://example3/8/localName7/"
xmlns:j.56="http://example5/5/localName1/"
xmlns:j.57="http://example3/6/localName5/"
xmlns:j.58="http://example1/8/localName8/"
xmlns:j.59="http://example7/5/localName2/"
xmlns:j.60="http://example7/1/localName1/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:j.61="http://example1/8/localName9/"
xmlns:j.62="http://example1/5/localName8/"
xmlns:j.63="http://example6/8/localName2/"
xmlns:j.64="http://example0/8/localName6/"
xmlns:j.65="http://example9/6/localName7/"
xmlns:j.66="http://example4/9/localName1/"
xmlns:j.67="http://example5/7/localName1/"
xmlns:j.68="http://example1/9/localName1/"
```

```
xmlns:j.69="http://example0/6/localName3/"
xmlns:j.70="http://example0/2/localName6/"
xmlns:j.71="http://example3/0/localName5/"
xmlns:j.72="http://example4/5/localName6/"
xmlns:j.73="http://example1/4/localName8/"
xmlns:j.74="http://example4/8/localName7/"
xmlns:j.75="http://example0/4/localName8/"
xmlns:j.76="http://example9/3/localName0/"
xmlns:j.77="http://example9/9/localName5/"
xmlns:j.78="http://example5/5/localName2/"
xmlns:j.79="http://example1/2/localName4/"
xmlns:j.80="http://example7/1/localName9/"
xmlns:j.81="http://example4/1/localName6/"
xmlns:j.82="http://example2/2/localName3/"
xmlns:j.83="http://example0/5/localName5/"
xmlns:j.84="http://example0/8/localName8/"
xmlns:j.85="http://example7/5/localName3/"
xmlns:j.86="http://example6/9/localName9/"
xmlns:j.87="http://example4/0/localName7/"
xmlns:j.88="http://example6/5/localName3/"
xmlns:j.89="http://example8/0/localName6/"
xmlns:j.90="http://example9/6/localName2/"
xmlns:j.91="http://example0/5/localName4/"
xmlns:j.92="http://example5/2/localName0/"
xmlns:j.93="http://example9/4/localName0/"
xmlns:j.94="http://example9/5/localName3/"
xmlns:j.95="http://example3/3/localName5/"
xmlns:j.96="http://example7/4/localName9/"
xmlns:j.97="http://example4/5/localName7/"
xmlns:j.98="http://example5/3/localName0/"
xmlns:j.100="http://example3/6/localName7/"
xmlns:j.99="http://example6/0/localName2/"
xmlns:j.101="http://example5/3/localName9/"
xmlns:j.102="http://example9/9/localName6/"
xmlns:j.103="http://example4/0/localName9/"
xmlns:j.104="http://example2/8/localName2/"
xmlns:j.105="http://example7/9/localName1/"
xmlns:j.107="http://example2/1/localName1/"
xmlns:j.106="http://example6/3/localName0/"
xmlns:j.108="http://example9/8/localName5/"
xmlns:j.109="http://example5/1/localName3/"
xmlns:j.110="http://example0/3/localName4/"
xmlns:j.111="http://example8/0/localName1/"
xmlns:j.112="http://example6/6/localName8/"
xmlns:j.114="http://example4/2/localName7/"
```

```
xmlns:j.113="http://example6/9/localName6/"
xmlns:j.115="http://example1/7/localName8/"
xmlns:j.116="http://example4/4/localName0/"
xmlns:j.117="http://example3/4/localName8/"
xmlns:j.118="http://example9/5/localName1/"
xmlns:j.119="http://example3/7/localName8/"
xmlns:j.120="http://example8/3/localName5/"
xmlns:j.121="http://example4/6/localName2/"
xmlns:j.122="http://example9/1/localName2/"
xmlns:j.123="http://example4/4/localName9/"
xmlns:j.124="http://example0/6/localName4/"
xmlns:j.125="http://example2/0/localName8/"
xmlns:j.128="http://example2/9/localName4/"
xmlns:j.127="http://example5/8/localName3/"
xmlns:j.126="http://example0/4/localName1/"
xmlns:j.130="http://example2/5/localName5/"
xmlns:j.129="http://example8/7/localName0/"
xmlns:j.132="http://example1/8/localName3/"
xmlns:j.131="http://example7/7/localName7/"
xmlns:j.134="http://example9/6/localName0/"
xmlns:j.133="http://example6/3/localName1/"
xmlns:j.135="http://example0/8/localName1/"
xmlns:j.138="http://example8/0/localName0/"
xmlns:j.137="http://example6/5/localName5/"
xmlns:j.136="http://example3/9/localName7/"
xmlns:j.139="http://example1/4/localName4/"
xmlns:j.140="http://example0/3/localName1/"
xmlns:j.141="http://example8/3/localName6/"
xmlns:j.142="http://example7/3/localName1/"
xmlns:j.143="http://example4/6/localName1/"
xmlns:j.144="http://example7/8/localName1/"
xmlns:j.145="http://example2/4/localName1/"
xmlns:j.146="http://example9/1/localName5/"
xmlns:j.147="http://example4/3/localName5/"
xmlns:j.148="http://example5/2/localName5/"
xmlns:j.150="http://example9/8/localName3/"
xmlns:j.149="http://example8/2/localName5/"
xmlns:j.151="http://example2/8/localName0/"
xmlns:j.153="http://example0/0/localName8/"
xmlns:j.152="http://example4/9/localName9/"
xmlns:j.156="http://example1/1/localName6/"
xmlns:j.155="http://example6/3/localName2/"
xmlns:j.154="http://example1/3/localName2/"
xmlns:j.157="http://example7/9/localName6/"
xmlns:j.158="http://example1/6/localName0/"
```

```
  xmlns:j.159="http://example8/6/localName2/"
  xmlns:j.161="http://example4/6/localName0/"
  xmlns:j.160="http://example7/0/localName5/"
  xmlns:j.162="http://example0/5/localName0/"
  xmlns:j.163="http://example9/0/localName5/"
  xmlns:j.164="http://example6/1/localName3/"
  xmlns:j.165="http://example8/5/localName9/"
  xmlns:j.166="http://example8/7/localName2/"
  xmlns:j.167="http://example7/4/localName5/"
  xmlns:j.168="http://example0/1/localName5/"
  xmlns:j.169="http://example1/8/localName5/"
  xmlns:j.170="http://example9/3/localName3/"
  xmlns:j.172="http://example6/9/localName5/"
  xmlns:j.171="http://example2/8/localName1/"
  xmlns:j.173="http://example0/8/localName3/"
  xmlns:j.175="http://example5/1/localName4/"
  xmlns:j.174="http://example3/9/localName9/"
  xmlns:j.176="http://example5/8/localName5/">
<rdf:Description rdf:about="http://example5/ROOT">
  <j.1:value2>
    <rdf:Description rdf:about="http://example7y">
      <j.86:value0 rdf:resource="http://example2z"/>
      <j.107:value7 rdf:resource="http://example7x"/>
      <j.67:value8>
        <rdf:Description rdf:about="http://example2y">
          <j.69:value5 rdf:resource="http://example2z"/>
          <j.163:value9 rdf:resource="http://example7z"/>
          <j.48:value0 rdf:resource="http://example6z"/>
          <j.13:value8 rdf:resource="http://example4z"/>
          <j.0:value5 rdf:resource="http://example8z"/>
          <j.45:value1>
            <rdf:Description rdf:about="http://example5z">
              <j.121:value7 rdf:resource="http://example6x"/>
              <j.84:value5 rdf:resource="http://example6x"/>
              <j.71:value6 rdf:resource="http://example4x"/>
              <j.165:value0>
                <rdf:Description rdf:about="http://example5y">
                  <j.136:value6 rdf:resource="http://example4z"/>
                  <j.144:value1 rdf:resource="http://example5z"/>
                  <j.131:value9 rdf:resource="http://example5z"/>
                  <j.89:value5 rdf:resource="http://example8z"/>
                  <j.62:value0 rdf:resource="http://example1z"/>
                  <j.101:value9 rdf:resource="http://example8z"/>
                  <j.8:value4 rdf:resource="http://example1z"/>
                  <j.124:value4 rdf:resource="http://example8z"/>
```

```
        <j.155:value9 rdf:resource="http://example7z"/>
        <j.135:value6 rdf:resource="http://example7z"/>
        <j.148:value3 rdf:resource="http://example7z"/>
        <j.119:value6 rdf:resource="http://example2z"/>
      </rdf:Description>
    </j.165:value0>
    <j.45:value1>
      <rdf:Description rdf:about="http://example2x">
        <j.27:value5 rdf:resource="http://example2y"/>
        <j.49:value8 rdf:resource="http://example9x"/>
        <j.38:value2>
          <rdf:Description rdf:about="http://example0y">
            <j.10:value7 rdf:resource="http://example6z"/>
            <j.92:value9 rdf:resource="http://example7z"/>
            <j.21:value0 rdf:resource="http://example8z"/>
            <j.56:value9 rdf:resource="http://example6z"/>
            <j.107:value7 rdf:resource="http://example8z"/>
            <j.130:value3 rdf:resource="http://example5z"/>
            <j.7:value0 rdf:resource="http://example9z"/>
            <j.115:value2 rdf:resource="http://example9z"/>
            <j.117:value2 rdf:resource="http://example9z"/>
            <j.33:value0 rdf:resource="http://example0z"/>
            <j.174:value7 rdf:resource="http://example4z"/>
            <j.132:value2 rdf:resource="http://example0z"/>
            <j.2:value7 rdf:resource="http://example7z"/>
          </rdf:Description>
        </j.38:value2>
        <j.128:value2>
          <rdf:Description rdf:about="http://example1y">
            <j.84:value5 rdf:resource="http://example6z"/>
            <j.16:value0 rdf:resource="http://example0z"/>
            <j.126:value2 rdf:resource="http://example7z"/>
            <j.129:value2 rdf:resource="http://example9z"/>
            <j.134:value0 rdf:resource="http://example5z"/>
            <j.31:value2 rdf:resource="http://example9z"/>
          </rdf:Description>
        </j.128:value2>
        <j.89:value5 rdf:resource="http://example2x"/>
        <j.164:value3>
          <rdf:Description rdf:about="http://example3y">
            <j.49:value8 rdf:resource="http://example7z"/>
            <j.100:value4 rdf:resource="http://example3z"/>
            <j.116:value0 rdf:resource="http://example9z"/>
            <j.15:value1 rdf:resource="http://example9z"/>
            <j.40:value5 rdf:resource="http://example2z"/>
```

```
      <j.63:value7 rdf:resource="http://example6z"/>
    </rdf:Description>
  </j.164:value3>
  <j.54:value6 rdf:resource="http://example4x"/>
  <j.31:value2 rdf:resource="http://example7x"/>
  <j.18:value5>
    <rdf:Description rdf:about="http://example6y">
      <j.34:value1 rdf:resource="http://example6z"/>
      <j.143:value4 rdf:resource="http://example9z"/>
      <j.51:value4 rdf:resource="http://example8z"/>
      <j.103:value4 rdf:resource="http://example8z"/>
      <j.37:value9 rdf:resource="http://example0z"/>
      <j.74:value6 rdf:resource="http://example1z"/>
      <j.167:value7 rdf:resource="http://example3z"/>
      <j.90:value0 rdf:resource="http://example6z"/>
      <j.82:value1 rdf:resource="http://example1z"/>
      <j.111:value5 rdf:resource="http://example3z"/>
    </rdf:Description>
  </j.18:value5>
  <j.106:value1 rdf:resource="http://example0y"/>
  <j.32:value5 rdf:resource="http://example1y"/>
  <j.109:value1 rdf:resource="http://example5x"/>
  <j.152:value7 rdf:resource="http://example8x"/>
  <j.15:value1 rdf:resource="http://example2x"/>
  <j.86:value0 rdf:resource="http://example8x"/>
  <j.103:value4 rdf:resource="http://example9x"/>
  <j.125:value3 rdf:resource="http://example3y"/>
  <j.39:value4 rdf:resource="http://example2y"/>
  <j.64:value3 rdf:resource="http://example3x"/>
  <j.174:value6>
    <rdf:Description rdf:about="http://example8y">
      <j.43:value4 rdf:resource="http://example2z"/>
      <j.81:value9 rdf:resource="http://example2z"/>
      <j.152:value7 rdf:resource="http://example6z"/>
      <j.176:value1 rdf:resource="http://example6z"/>
      <j.127:value4 rdf:resource="http://example8z"/>
      <j.83:value5 rdf:resource="http://example7z"/>
      <j.30:value7 rdf:resource="http://example2z"/>
      <j.149:value4 rdf:resource="http://example5z"/>
      <j.85:value9 rdf:resource="http://example1z"/>
      <j.169:value5 rdf:resource="http://example7z"/>
      <j.44:value0 rdf:resource="http://example0z"/>
    </rdf:Description>
  </j.174:value6>
  <j.18:value5 rdf:resource="http://example8y"/>
```

```
<j.135:value6 rdf:resource="http://example7x"/>
<j.53:value6 rdf:resource="http://example8y"/>
<j.132:value2 rdf:resource="http://example9x"/>
<j.6:value7 rdf:resource="http://example0y"/>
<j.14:value1 rdf:resource="http://example7y"/>
<j.16:value0 rdf:resource="http://example3x"/>
<j.122:value4 rdf:resource="http://example1x"/>
<j.120:value4 rdf:resource="http://example7y"/>
<j.174:value7 rdf:resource="http://example5x"/>
<j.46:value9 rdf:resource="http://example8y"/>
<j.63:value7 rdf:resource="http://example3x"/>
<j.9:value5 rdf:resource="http://example2y"/>
<j.30:value7 rdf:resource="http://example7x"/>
<j.43:value4 rdf:resource="http://example9x"/>
<j.113:value5 rdf:resource="http://example3x"/>
<j.154:value4 rdf:resource="http://example1y"/>
<j.144:value1 rdf:resource="http://example8x"/>
<j.156:value7 rdf:resource="http://example5y"/>
<j.158:value5 rdf:resource="http://example5y"/>
<j.85:value9 rdf:resource="http://example9x"/>
<j.35:value4 rdf:resource="http://example5y"/>
<j.138:value0>
  <rdf:Description rdf:about="http://example9y">
    <j.55:value3 rdf:resource="http://example4z"/>
    <j.11:value5 rdf:resource="http://example1z"/>
    <j.121:value7 rdf:resource="http://example0z"/>
    <j.93:value0 rdf:resource="http://example2z"/>
    <j.55:value4 rdf:resource="http://example1z"/>
  </rdf:Description>
</j.138:value0>
<j.98:value4 rdf:resource="http://example0y"/>
<j.48:value9 rdf:resource="http://example5y"/>
<j.79:value5>
  <rdf:Description rdf:about="http://example4y">
    <j.173:value2 rdf:resource="http://example1z"/>
    <j.17:value8 rdf:resource="http://example2z"/>
    <j.71:value6 rdf:resource="http://example1z"/>
    <j.122:value4 rdf:resource="http://example4z"/>
    <j.20:value3 rdf:resource="http://example9z"/>
    <j.147:value3 rdf:resource="http://example2z"/>
    <j.142:value1 rdf:resource="http://example6z"/>
    <j.109:value1 rdf:resource="http://example6z"/>
  </rdf:Description>
</j.79:value5>
<j.129:value2 rdf:resource="http://example0x"/>
```

```
      <j.23:value3 rdf:resource="http://example3y"/>
      <j.21:value0 rdf:resource="http://example5x"/>
      <j.33:value0 rdf:resource="http://example8x"/>
      <j.118:value1 rdf:resource="http://example2y"/>
      <j.55:value3 rdf:resource="http://example7x"/>
      <j.149:value4 rdf:resource="http://example0x"/>
      <j.8:value4 rdf:resource="http://example3x"/>
      <j.170:value6 rdf:resource="http://example4x"/>
      <j.61:value7 rdf:resource="http://example8y"/>
      <j.25:value5 rdf:resource="http://example4y"/>
      <j.69:value5 rdf:resource="http://example0x"/>
      <j.155:value3 rdf:resource="http://example7y"/>
      <j.104:value9 rdf:resource="http://example7x"/>
      <j.0:value5 rdf:resource="http://example8x"/>
      <j.4:value9 rdf:resource="http://example7y"/>
    </rdf:Description>
  </j.45:value1>
  <j.35:value9 rdf:resource="http://example2y"/>
  <j.73:value7 rdf:resource="http://example7y"/>
  <j.151:value2 rdf:resource="http://example9y"/>
  <j.100:value4 rdf:resource="http://example8x"/>
  <j.56:value9 rdf:resource="http://example6x"/>
  <j.159:value3 rdf:resource="http://example7y"/>
  <j.164:value8 rdf:resource="http://example2y"/>
  <j.145:value0 rdf:resource="http://example4y"/>
  <j.97:value0 rdf:resource="http://example8y"/>
  <j.20:value3 rdf:resource="http://example1x"/>
  <j.26:value7 rdf:resource="http://example7y"/>
  <j.37:value9 rdf:resource="http://example0x"/>
  <j.141:value6 rdf:resource="http://example4y"/>
  <j.5:value8 rdf:resource="http://example6y"/>
  <j.99:value3 rdf:resource="http://example1x"/>
  <j.119:value6 rdf:resource="http://example2x"/>
  <j.10:value7 rdf:resource="http://example7x"/>
  <j.108:value2 rdf:resource="http://example4y"/>
  <j.160:value1 rdf:resource="http://example5x"/>
  <j.60:value2 rdf:resource="http://example8y"/>
  <j.42:value4 rdf:resource="http://example1y"/>
  <j.66:value2 rdf:resource="http://example6y"/>
  <j.13:value8 rdf:resource="http://example3x"/>
  <j.52:value4 rdf:resource="http://example6y"/>
  <j.87:value8 rdf:resource="http://example0y"/>
  <j.70:value7 rdf:resource="http://example0y"/>
  <j.137:value5 rdf:resource="http://example7y"/>
  <j.88:value7 rdf:resource="http://example1x"/>
```

```
            <j.142:value1 rdf:resource="http://example4x"/>
            <j.165:value5 rdf:resource="http://example5x"/>
            <j.28:value8 rdf:resource="http://example7y"/>
            <j.95:value9 rdf:resource="http://example2y"/>
            <j.19:value8 rdf:resource="http://example5y"/>
            <j.41:value6 rdf:resource="http://example5y"/>
            <j.110:value8 rdf:resource="http://example0y"/>
            <j.115:value2 rdf:resource="http://example5x"/>
            <j.96:value3 rdf:resource="http://example3y"/>
            <j.166:value5 rdf:resource="http://example0y"/>
            <j.126:value2 rdf:resource="http://example8x"/>
            <j.168:value3 rdf:resource="http://example9x"/>
            <j.148:value3 rdf:resource="http://example2x"/>
            <j.131:value9 rdf:resource="http://example5x"/>
            <j.77:value3 rdf:resource="http://example7x"/>
            <j.90:value0 rdf:resource="http://example0x"/>
            <j.117:value2 rdf:resource="http://example4x"/>
            <j.150:value0 rdf:resource="http://example1y"/>
            <j.91:value6 rdf:resource="http://example3y"/>
            <j.173:value2 rdf:resource="http://example1x"/>
            <j.62:value0 rdf:resource="http://example0x"/>
            <j.143:value4 rdf:resource="http://example0x"/>
            <j.168:value7 rdf:resource="http://example4y"/>
            <j.172:value9 rdf:resource="http://example8x"/>
            <j.83:value5 rdf:resource="http://example9x"/>
            <j.116:value0 rdf:resource="http://example2x"/>
            <j.146:value0 rdf:resource="http://example5y"/>
            <j.153:value6 rdf:resource="http://example2y"/>
            <j.176:value1 rdf:resource="http://example8x"/>
          </rdf:Description>
        </j.45:value1>
        <j.36:value4 rdf:resource="http://example5z"/>
        <j.77:value3 rdf:resource="http://example4z"/>
        <j.104:value9 rdf:resource="http://example3z"/>
        <j.88:value7 rdf:resource="http://example3z"/>
        <j.102:value3 rdf:resource="http://example7z"/>
        <j.54:value6 rdf:resource="http://example6z"/>
      </rdf:Description>
    </j.67:value8>
    <j.74:value6 rdf:resource="http://example8x"/>
    <j.147:value3 rdf:resource="http://example4x"/>
    <j.81:value9 rdf:resource="http://example5x"/>
    <j.92:value9 rdf:resource="http://example4x"/>
    <j.157:value1 rdf:resource="http://example8y"/>
    <j.17:value8 rdf:resource="http://example6x"/>
```

```
<j.75:value3 rdf:resource="http://example2z"/>
<j.48:value0 rdf:resource="http://example1x"/>
<j.113:value5 rdf:resource="http://example5z"/>
<j.168:value3 rdf:resource="http://example9z"/>
<j.172:value9 rdf:resource="http://example2z"/>
<j.76:value4 rdf:resource="http://example6y"/>
<j.2:value7 rdf:resource="http://example2x"/>
<j.114:value5 rdf:resource="http://example1y"/>
<j.34:value1 rdf:resource="http://example9x"/>
<j.3:value9 rdf:resource="http://example2y"/>
<j.163:value9 rdf:resource="http://example6x"/>
<j.123:value3 rdf:resource="http://example6y"/>
<j.127:value4 rdf:resource="http://example6x"/>
<j.55:value4 rdf:resource="http://example9x"/>
<j.82:value1 rdf:resource="http://example0x"/>
<j.64:value3 rdf:resource="http://example7z"/>
<j.44:value0 rdf:resource="http://example4x"/>
<j.139:value9 rdf:resource="http://example7y"/>
<j.99:value3 rdf:resource="http://example6z"/>
<j.111:value5 rdf:resource="http://example6x"/>
<j.40:value5 rdf:resource="http://example0x"/>
<j.94:value5 rdf:resource="http://example0y"/>
<j.102:value3 rdf:resource="http://example8x"/>
<j.47:value3 rdf:resource="http://example9y"/>
<j.133:value9 rdf:resource="http://example6y"/>
<j.169:value5 rdf:resource="http://example3x"/>
<j.167:value7 rdf:resource="http://example1x"/>
<j.155:value9 rdf:resource="http://example5x"/>
<j.58:value8 rdf:resource="http://example5y"/>
<j.90:value7 rdf:resource="http://example9y"/>
<j.78:value2 rdf:resource="http://example6y"/>
<j.134:value0 rdf:resource="http://example5x"/>
<j.171:value1 rdf:resource="http://example0y"/>
<j.160:value1 rdf:resource="http://example2z"/>
<j.140:value2 rdf:resource="http://example0y"/>
<j.130:value3 rdf:resource="http://example2x"/>
<j.162:value4 rdf:resource="http://example5y"/>
<j.170:value6 rdf:resource="http://example9z"/>
<j.80:value9 rdf:resource="http://example9y"/>
<j.165:value5 rdf:resource="http://example7z"/>
<j.59:value9 rdf:resource="http://example6y"/>
<j.105:value5 rdf:resource="http://example2y"/>
<j.7:value0 rdf:resource="http://example1x"/>
<j.75:value3 rdf:resource="http://example7x"/>
<j.124:value4 rdf:resource="http://example0x"/>
```

```
        <j.26:value5 rdf:resource="http://example3y"/>
        <j.68:value5 rdf:resource="http://example5y"/>
        <j.161:value1 rdf:resource="http://example8y"/>
        <j.72:value1 rdf:resource="http://example4y"/>
        <j.22:value2 rdf:resource="http://example5y"/>
        <j.112:value4 rdf:resource="http://example8y"/>
        <j.36:value4 rdf:resource="http://example6x"/>
        <j.57:value0 rdf:resource="http://example0y"/>
        <j.29:value5 rdf:resource="http://example8y"/>
        <j.136:value6 rdf:resource="http://example2x"/>
        <j.50:value3 rdf:resource="http://example4y"/>
        <j.12:value0 rdf:resource="http://example2y"/>
        <j.65:value4 rdf:resource="http://example0y"/>
        <j.11:value5 rdf:resource="http://example6x"/>
        <j.175:value1 rdf:resource="http://example5z"/>
        <j.24:value0 rdf:resource="http://example6y"/>
        <j.101:value9 rdf:resource="http://example3x"/>
        <j.93:value0 rdf:resource="http://example5x"/>
        <j.51:value4 rdf:resource="http://example7x"/>
      </rdf:Description>
    </j.1:value2>
    <j.175:value1 rdf:resource="http://example2x"/>
  </rdf:Description>
</rdf:RDF>
```

RDF Triple notation:

```
http://example6y http://example6/2/localName8/value1 http://example6z .
http://example6y http://example4/6/localName1/value4 http://example9z .
http://example6y http://example5/0/localName5/value4 http://example8z .
http://example6y http://example4/0/localName9/value4 http://example8z .
http://example6y http://example4/9/localName6/value9 http://example0z .
http://example6y http://example4/8/localName7/value6 http://example1z .
http://example6y http://example7/4/localName5/value7 http://example3z .
http://example6y http://example9/6/localName2/value0 http://example6z .
http://example6y http://example2/2/localName3/value1 http://example1z .
http://example6y http://example8/0/localName1/value5 http://example3z .
http://example8y http://example7/1/localName6/value4 http://example2z .
http://example8y http://example4/1/localName6/value9 http://example2z .
http://example8y http://example4/9/localName9/value7 http://example6z .
http://example8y http://example5/8/localName5/value1 http://example6z .
http://example8y http://example5/8/localName3/value4 http://example8z .
http://example8y http://example0/5/localName5/value5 http://example7z .
http://example8y http://example3/9/localName2/value7 http://example2z .
```

```
http://example8y http://example8/2/localName5/value4 http://example5z .
http://example8y http://example7/5/localName3/value9 http://example1z .
http://example8y http://example1/8/localName5/value5 http://example7z .
http://example8y http://example7/3/localName6/value0 http://example0z .
http://example1y http://example0/8/localName8/value5 http://example6z .
http://example1y http://example8/2/localName3/value0 http://example0z .
http://example1y http://example0/4/localName1/value2 http://example7z .
http://example1y http://example8/7/localName0/value2 http://example9z .
http://example1y http://example9/6/localName0/value0 http://example5z .
http://example1y http://example5/1/localName7/value2 http://example9z .
http://example3y http://example6/1/localName6/value8 http://example7z .
http://example3y http://example3/6/localName7/value4 http://example3z .
http://example3y http://example4/4/localName0/value0 http://example9z .
http://example3y http://example7/0/localName1/value1 http://example9z .
http://example3y http://example3/7/localName4/value5 http://example2z .
http://example3y http://example6/8/localName2/value7 http://example6z .
http://example2x http://example8/1/localName6/value5 http://example2y .
http://example2x http://example6/1/localName6/value8 http://example9x .
http://example2x http://example3/4/localName3/value2 http://example0y .
http://example2x http://example2/9/localName4/value2 http://example1y .
http://example2x http://example8/0/localName6/value5 http://example2x .
http://example2x http://example6/1/localName3/value3 http://example3y .
http://example2x http://example2/4/localName6/value6 http://example4x .
http://example2x http://example5/1/localName7/value2 http://example7x .
http://example2x http://example2/6/localName4/value5 http://example6y .
http://example2x http://example6/3/localName0/value1 http://example0y .
http://example2x http://example6/8/localName8/value5 http://example1y .
http://example2x http://example5/1/localName3/value1 http://example5x .
http://example2x http://example4/9/localName9/value7 http://example8x .
http://example2x http://example7/0/localName1/value1 http://example2x .
http://example2x http://example6/9/localName9/value0 http://example8x .
http://example2x http://example4/0/localName9/value4 http://example9x .
http://example2x http://example2/0/localName8/value3 http://example3y .
http://example2x http://example4/5/localName5/value4 http://example2y .
http://example2x http://example0/8/localName6/value3 http://example3x .
http://example2x http://example3/9/localName9/value6 http://example8y .
http://example2x http://example2/6/localName4/value5 http://example8y .
http://example2x http://example0/8/localName1/value6 http://example7x .
http://example2x http://example5/1/localName0/value6 http://example8y .
http://example2x http://example1/8/localName3/value2 http://example9x .
http://example2x http://example2/1/localName6/value7 http://example0y .
http://example2x http://example3/4/localName2/value1 http://example7y .
http://example2x http://example8/2/localName3/value0 http://example3x .
http://example2x http://example9/1/localName2/value4 http://example1x .
http://example2x http://example8/3/localName5/value4 http://example7y .
```

```
http://example2x http://example3/9/localName9/value7 http://example5x .
http://example2x http://example7/4/localName1/value9 http://example8y .
http://example2x http://example6/8/localName2/value7 http://example3x .
http://example2x http://example7/2/localName5/value5 http://example2y .
http://example2x http://example3/9/localName2/value7 http://example7x .
http://example2x http://example7/1/localName6/value4 http://example9x .
http://example2x http://example6/9/localName6/value5 http://example3x .
http://example2x http://example1/3/localName2/value4 http://example1y .
http://example2x http://example7/8/localName1/value1 http://example8x .
http://example2x http://example1/1/localName6/value7 http://example5y .
http://example2x http://example1/6/localName0/value5 http://example5y .
http://example2x http://example7/5/localName3/value9 http://example9x .
http://example2x http://example4/3/localName2/value4 http://example5y .
http://example2x http://example8/0/localName0/value0 http://example9y .
http://example2x http://example5/3/localName0/value4 http://example0y .
http://example2x http://example4/8/localName6/value9 http://example5y .
http://example2x http://example1/2/localName4/value5 http://example4y .
http://example2x http://example8/7/localName0/value2 http://example0x .
http://example2x http://example0/0/localName2/value3 http://example3y .
http://example2x http://example2/5/localName0/value0 http://example5x .
http://example2x http://example0/8/localName5/value0 http://example8x .
http://example2x http://example9/5/localName1/value1 http://example2y .
http://example2x http://example3/8/localName7/value3 http://example7x .
http://example2x http://example8/2/localName5/value4 http://example0x .
http://example2x http://example9/7/localName9/value4 http://example3x .
http://example2x http://example9/3/localName3/value6 http://example4x .
http://example2x http://example1/8/localName9/value7 http://example8y .
http://example2x http://example6/5/localName9/value5 http://example4y .
http://example2x http://example0/6/localName3/value5 http://example0x .
http://example2x http://example6/3/localName2/value3 http://example7y .
http://example2x http://example2/8/localName2/value9 http://example7x .
http://example2x http://example6/5/localName8/value5 http://example8x .
http://example2x http://example4/6/localName7/value9 http://example7y .
http://example5y http://example3/9/localName7/value6 http://example4z .
http://example5y http://example7/8/localName1/value1 http://example5z .
http://example5y http://example7/7/localName7/value9 http://example5z .
http://example5y http://example8/0/localName6/value5 http://example8z .
http://example5y http://example1/5/localName8/value0 http://example1z .
http://example5y http://example5/3/localName9/value9 http://example8z .
http://example5y http://example9/7/localName9/value4 http://example1z .
http://example5y http://example0/6/localName4/value4 http://example8z .
http://example5y http://example6/3/localName2/value9 http://example7z .
http://example5y http://example0/8/localName1/value6 http://example7z .
http://example5y http://example5/2/localName5/value3 http://example7z .
http://example5y http://example3/7/localName8/value6 http://example2z .
```

```
http://example5/ROOT http://example5/5/localName5/value2
http://example7y .
http://example5/ROOT http://example5/1/localName4/value1
http://example2x .
http://example7y http://example6/9/localName9/value0 http://example2z .
http://example7y http://example2/1/localName1/value7 http://example7x .
http://example7y http://example5/7/localName1/value8 http://example2y .
http://example7y http://example4/8/localName7/value6 http://example8x .
http://example7y http://example4/3/localName5/value3 http://example4x .
http://example7y http://example4/1/localName6/value9 http://example5x .
http://example7y http://example5/2/localName0/value9 http://example4x .
http://example7y http://example7/9/localName6/value1 http://example8y .
http://example7y http://example9/8/localName1/value8 http://example6x .
http://example7y http://example0/4/localName8/value3 http://example2z .
http://example7y http://example4/8/localName6/value0 http://example1x .
http://example7y http://example6/9/localName6/value5 http://example5z .
http://example7y http://example0/1/localName5/value3 http://example9z .
http://example7y http://example6/9/localName5/value9 http://example2z .
http://example7y http://example9/3/localName0/value4 http://example6y .
http://example7y http://example1/2/localName5/value7 http://example2x .
http://example7y http://example4/2/localName7/value5 http://example1y .
http://example7y http://example6/2/localName8/value1 http://example9x .
http://example7y http://example4/1/localName7/value9 http://example2y .
http://example7y http://example9/0/localName5/value9 http://example6x .
http://example7y http://example4/4/localName9/value3 http://example6y .
http://example7y http://example5/8/localName3/value4 http://example6x .
http://example7y http://example3/8/localName7/value4 http://example9x .
http://example7y http://example2/2/localName3/value1 http://example0x .
http://example7y http://example0/8/localName6/value3 http://example7z .
http://example7y http://example7/3/localName6/value0 http://example4x .
http://example7y http://example1/4/localName4/value9 http://example7y .
http://example7y http://example6/0/localName2/value3 http://example6z .
http://example7y http://example8/0/localName1/value5 http://example6x .
http://example7y http://example3/7/localName4/value5 http://example0x .
http://example7y http://example9/5/localName3/value5 http://example0y .
http://example7y http://example9/9/localName6/value3 http://example8x .
http://example7y http://example7/7/localName5/value3 http://example9y .
http://example7y http://example6/3/localName1/value9 http://example6y .
http://example7y http://example1/8/localName5/value5 http://example3x .
http://example7y http://example7/4/localName5/value7 http://example1x .
http://example7y http://example6/3/localName2/value9 http://example5x .
http://example7y http://example1/8/localName8/value8 http://example5y .
http://example7y http://example9/6/localName2/value7 http://example9y .
http://example7y http://example5/5/localName2/value2 http://example6y .
http://example7y http://example9/6/localName0/value0 http://example5x .
```

```
http://example7y http://example2/8/localName1/value1 http://example0y .
http://example7y http://example7/0/localName5/value1 http://example2z .
http://example7y http://example0/3/localName1/value2 http://example0y .
http://example7y http://example2/5/localName5/value3 http://example2x .
http://example7y http://example0/5/localName0/value4 http://example5y .
http://example7y http://example9/3/localName3/value6 http://example9z .
http://example7y http://example7/1/localName9/value9 http://example9y .
http://example7y http://example8/5/localName9/value5 http://example7z .
http://example7y http://example7/5/localName2/value9 http://example6y .
http://example7y http://example7/9/localName1/value5 http://example2y .
http://example7y http://example5/6/localName1/value0 http://example1x .
http://example7y http://example0/4/localName8/value3 http://example7x .
http://example7y http://example0/6/localName4/value4 http://example0x .
http://example7y http://example7/3/localName2/value5 http://example3y .
http://example7y http://example1/9/localName1/value5 http://example5y .
http://example7y http://example4/6/localName0/value1 http://example8y .
http://example7y http://example4/5/localName6/value1 http://example4y .
http://example7y http://example5/9/localName5/value2 http://example5y .
http://example7y http://example6/6/localName8/value4 http://example8y .
http://example7y http://example8/2/localName2/value4 http://example6x .
http://example7y http://example3/6/localName5/value0 http://example0y .
http://example7y http://example1/6/localName2/value5 http://example8y .
http://example7y http://example3/9/localName7/value6 http://example2x .
http://example7y http://example0/4/localName7/value3 http://example4y .
http://example7y http://example2/4/localName8/value0 http://example2y .
http://example7y http://example9/6/localName7/value4 http://example0y .
http://example7y http://example3/8/localName9/value5 http://example6x .
http://example7y http://example5/1/localName4/value1 http://example5z .
http://example7y http://example2/3/localName3/value0 http://example6y .
http://example7y http://example5/3/localName9/value9 http://example3x .
http://example7y http://example9/4/localName0/value0 http://example5x .
http://example7y http://example5/0/localName5/value4 http://example7x .
http://example0y http://example0/5/localName8/value7 http://example6z .
http://example0y http://example5/2/localName0/value9 http://example7z .
http://example0y http://example2/5/localName0/value0 http://example8z .
http://example0y http://example5/5/localName1/value9 http://example6z .
http://example0y http://example2/1/localName1/value7 http://example8z .
http://example0y http://example2/5/localName5/value3 http://example5z .
http://example0y http://example5/6/localName1/value0 http://example9z .
http://example0y http://example1/7/localName8/value2 http://example9z .
http://example0y http://example3/4/localName8/value2 http://example9z .
http://example0y http://example0/8/localName5/value0 http://example0z .
http://example0y http://example3/9/localName9/value7 http://example4z .
http://example0y http://example1/8/localName3/value2 http://example0z .
http://example0y http://example1/2/localName5/value7 http://example7z .
```

```
http://example9y http://example3/8/localName7/value3 http://example4z .
http://example9y http://example3/8/localName9/value5 http://example1z .
http://example9y http://example4/6/localName2/value7 http://example0z .
http://example9y http://example9/4/localName0/value0 http://example2z .
http://example9y http://example3/8/localName7/value4 http://example1z .
http://example2y http://example0/6/localName3/value5 http://example2z .
http://example2y http://example9/0/localName5/value9 http://example7z .
http://example2y http://example4/8/localName6/value0 http://example6z .
http://example2y http://example4/4/localName5/value8 http://example4z .
http://example2y http://example6/5/localName8/value5 http://example8z .
http://example2y http://example2/1/localName9/value1 http://example5z .
http://example2y http://example8/2/localName2/value4 http://example5z .
http://example2y http://example9/9/localName5/value3 http://example4z .
http://example2y http://example2/8/localName2/value9 http://example3z .
http://example2y http://example6/5/localName3/value7 http://example3z .
http://example2y http://example9/9/localName6/value3 http://example7z .
http://example2y http://example2/4/localName6/value6 http://example6z .
http://example5z http://example4/6/localName2/value7 http://example6x .
http://example5z http://example0/8/localName8/value5 http://example6x .
http://example5z http://example3/0/localName5/value6 http://example4x .
http://example5z http://example8/5/localName9/value0 http://example5y .
http://example5z http://example2/1/localName9/value1 http://example2x .
http://example5z http://example4/3/localName2/value9 http://example2y .
http://example5z http://example1/4/localName8/value7 http://example7y .
http://example5z http://example2/8/localName0/value2 http://example9y .
http://example5z http://example3/6/localName7/value4 http://example8x .
http://example5z http://example5/5/localName1/value9 http://example6x .
http://example5z http://example8/6/localName2/value3 http://example7y .
http://example5z http://example6/1/localName3/value8 http://example2y .
http://example5z http://example2/4/localName1/value0 http://example4y .
http://example5z http://example4/5/localName7/value0 http://example8y .
http://example5z http://example9/0/localName6/value3 http://example1x .
http://example5z http://example7/3/localName2/value7 http://example7y .
http://example5z http://example4/9/localName6/value9 http://example0x .
http://example5z http://example8/3/localName6/value6 http://example4y .
http://example5z http://example6/1/localName4/value8 http://example6y .
http://example5z http://example6/0/localName2/value3 http://example1x .
http://example5z http://example3/7/localName8/value6 http://example2x .
http://example5z http://example0/5/localName8/value7 http://example7x .
http://example5z http://example9/8/localName5/value2 http://example4y .
http://example5z http://example7/0/localName5/value1 http://example5x .
http://example5z http://example7/1/localName1/value2 http://example8y .
http://example5z http://example2/5/localName1/value4 http://example1y .
http://example5z http://example4/9/localName1/value2 http://example6y .
http://example5z http://example4/4/localName5/value8 http://example3x .
```

```
http://example5z http://example0/9/localName0/value4 http://example6y .
http://example5z http://example4/0/localName7/value8 http://example0y .
http://example5z http://example0/2/localName6/value7 http://example0y .
http://example5z http://example6/5/localName5/value5 http://example7y .
http://example5z http://example6/5/localName3/value7 http://example1x .
http://example5z http://example7/3/localName1/value1 http://example4x .
http://example5z http://example8/5/localName9/value5 http://example5x .
http://example5z http://example8/6/localName4/value8 http://example7y .
http://example5z http://example3/3/localName5/value9 http://example2y .
http://example5z http://example3/5/localName7/value8 http://example5y .
http://example5z http://example0/0/localName1/value6 http://example5y .
http://example5z http://example0/3/localName4/value8 http://example0y .
http://example5z http://example1/7/localName8/value2 http://example5x .
http://example5z http://example7/4/localName9/value3 http://example3y .
http://example5z http://example8/7/localName2/value5 http://example0y .
http://example5z http://example0/4/localName1/value2 http://example8x .
http://example5z http://example0/1/localName5/value3 http://example9x .
http://example5z http://example5/2/localName5/value3 http://example2x .
http://example5z http://example7/7/localName7/value9 http://example5x .
http://example5z http://example9/9/localName5/value3 http://example7x .
http://example5z http://example9/6/localName2/value0 http://example0x .
http://example5z http://example3/4/localName8/value2 http://example4x .
http://example5z http://example9/8/localName3/value0 http://example1y .
http://example5z http://example0/5/localName4/value6 http://example3y .
http://example5z http://example0/8/localName3/value2 http://example1x .
http://example5z http://example1/5/localName8/value0 http://example0x .
http://example5z http://example4/6/localName1/value4 http://example0x .
http://example5z http://example0/1/localName5/value7 http://example4y .
http://example5z http://example6/9/localName5/value9 http://example8x .
http://example5z http://example0/5/localName5/value5 http://example9x .
http://example5z http://example4/4/localName0/value0 http://example2x .
http://example5z http://example9/1/localName5/value0 http://example5y .
http://example5z http://example0/0/localName8/value6 http://example2y .
http://example5z http://example5/8/localName5/value1 http://example8x .
http://example4y http://example0/8/localName3/value2 http://example1z .
http://example4y http://example9/8/localName1/value8 http://example2z .
http://example4y http://example3/0/localName5/value6 http://example1z .
http://example4y http://example9/1/localName2/value4 http://example4z .
http://example4y http://example9/0/localName6/value3 http://example9z .
http://example4y http://example4/3/localName5/value3 http://example2z .
http://example4y http://example7/3/localName1/value1 http://example6z .
http://example4y http://example5/1/localName3/value1 http://example6z .
```