

**Enhancing user-controlled
contexts:
An innovative approach for
handling complex scientific data**

Patricia Hartmann

Master of Science
Data Science
School of Informatics
University of Edinburgh
2021

Abstract

The remark “Big Data is the oil of the 21st century” [107] nicely summarises the relevance which complex data has already reached in our lives. Still, such massive (digital) data piles impose a huge challenge on storage and maintenance systems and particularly science is exposed to this problem. Their current, mostly unsustainable silo-solutions [3, 17] require novel approaches like the one the DARE Knowledge Base (DKB) implements: the notion of user-controlled contexts, introduced in 2019 by Atkinson et al. [5] as an innovative way of partitioning the knowledge space and thus storing its diverse, interlinked data. However, since the current DARE Knowledge Base (DKB) prototype is not (yet) production-ready, the aim and objectives of this project are to improve the existing DKB application by identifying weaknesses, determining and resolving multiple of these issues (with particular focus on conceptual limitations) and assessing their value since eventually the (scientific) user community should benefit from any adjustments. To accomplish this, the project is divided into five phases: (1) To begin with, an appropriate number (390 MB) of synthetic and real-world records is generated before (2) thorough functionality and performance tests are performed and their outcome summarized. Thereafter, (3) a justified selection of dedicated bugs and functionality deficiencies is made, followed by an extensive literature review to contrast and reason various (conceptualization and implementation) choices. Finally, the (4) agreed developments are performed and (5) all achievements evaluated, both with functionality and performance tests and a user questionnaire. Since this project is rather focused on research than on development, the two key contributions are the 1) conceptualized and realized data querying improvements as well as the 2) collections conceptualization (the bug fixes are seen as a prerequisite for pursuing the later work on DKB). Notably, the two major (conceptual) ‘functionality’ achievements also received the best overall user rating (average of 4.7 out of 5.0), which further stresses their significance.

Acknowledgements

First and foremost, I would like to thank my supervisor, Malcolm Atkinson, for his continuous support and guidance throughout the project. Thank you for keeping on challenging and encouraging me!

I would also like to wholeheartedly thank Federica Magnoni, who works at the *National Institute of Geophysics and Volcanology (INGV)*, not only for her help with the generation of (scientific) real-world test data but also for pointing me to lots of relevant DARE sources. Thank you for our pleasant and fruitful discussions, I hope to meet you sometime in Italy!

Additionally, I want to sincerely thank Rosa Filgueira, Assistant Professor at *Heriot-Watt University*, and Oscar Corcho, Professor at the *Technical University of Madrid* and member of the Ontology Engineering Group. I am extremely grateful for your input on the DARE (CWL and dispel4py) workflows and your distributed SPARQL work!

Finally, I would like to express a heartfelt thank-you to all members of the former DARE project team (EU H2020 research and innovation program under grant agreement No 777413) as well as the University of Edinburgh's DIRG group for filling in my user questionnaire.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Patricia Hartmann)

Table of Contents

1	Introduction	1
1.1	The 21st century - Digital Age and Big Data era	1
1.2	DARE & DKB - Coping with complex scientific data	2
1.3	DKB's current limit - Research proposal	3
2	Background and Related Work	5
2.1	The DARE Knowledge Base (DKB)	5
2.2	Owlready2 - Following the graph database concept	7
2.2.1	Brief introduction to NoSQL databases	7
2.2.2	Graph databases - One of four main NoSQL types	8
2.2.3	Owlready2 - An optimized quadstore	9
3	Approach, Methodology and Achievements	10
3.1	Test data generation	11
3.1.1	Synthetic test data	11
3.1.2	Real-world test data	11
3.2	System analysis	12
3.2.1	Test case preparation	13
3.2.2	Test case execution and results	14
3.3	Project focus and literature review	17
3.3.1	Test result assessment, Issue selection and justification	18
3.3.2	Formulated research and development plan	20
3.3.3	Solution approach and conceptualisation	21
3.4	Software development	33
3.4.1	Utilized methodology	34
3.4.2	Coding style	35
3.4.3	Accomplished developments	35

3.5	Achievements and their assessment	36
4	Closing discussion and Future Work	37
	Bibliography	41
A	Test cases and results	53
A.1	Site functions	53
A.1.1	User Manual updates	54
A.2	Context functions	55
A.2.1	Application bug	60
A.2.2	User Manual updates	60
A.3	Concept functions	62
A.3.1	Application bugs	65
A.3.2	Functionality deficiencies	66
A.3.3	User Manual update	68
A.4	Instance functions	68
A.4.1	Application bugs	71
A.4.2	Functionality deficiency	72
A.4.3	User Manual update	73
A.5	Performance	73
A.5.1	Site functions	73
A.5.2	Concept and instance functions	74
B	Application bug fixes	77
B.1	Instance creation issue	77
B.2	Data access after DKB server restart	79
B.2.1	Find and get instance	79
B.2.2	Get concept	80
B.3	Find issue	85
C	User Manual updates	87
C.1	Running the DKB	87
C.1.1	Owlready2 warning message [75]	87
C.1.2	For Windows users only: potential run issues [75]	88
C.2	Find() search criteria and expected values	89
C.2.1	Available property search criteria and expected values [75]	89

C.2.2	Range sort: selection according to ASCII sort order [75]	90
C.3	Added notes	91
D	Survey questionnaire	92
D.1	Questionnaire	93
D.2	Survey results	96
D.3	Information sheet	98
D.4	Plain consent form	101

List of Figures

2.1	Concept of user-controlled contexts (Atkinson et al. (2019) [5])	6
2.2	DKB architecture, taken from Levray and Zhao (2020) [76]	6
2.3	Sample graph, taken from Webber and Van Bruggen (2020) [116]	8
3.1	Project phases	10
3.2	Spiral methodology, according to Despa (2020) [30]	34
A.1	DKB: entering and resetting a ‘frozen’ context	61
A.2	DKB: accessing previously created concept after server restart	65
A.3	DKB: find concept issues	66
A.3	DKB: Find concept issues (cont.)	67
A.4	DKB: accessing previously created instance after server restart	71
A.5	DKB: find instance issues	72
A.6	DKB login performance	73
A.7	DKB site function performance	74
A.8	DKB: <code>get()</code> function performance	74
A.8	DKB: <code>get()</code> function performance (cont.)	75
A.9	DKB: <code>find()</code> function performance	75
A.9	DKB: <code>find()</code> function performance (cont.)	76
C.1	DKB set-up: Owlready2 warning message	87
C.2	DKB set-up: common txt file location	88
C.3	DKB set-up: linked directory	88
C.4	DKB find: string range results in accordance with American Standard Code for Information Interchange (ASCII) sort order [100]	90
C.5	DKB enter context: MultiUser exception	91
D.1	Survey questionnaire: first section	93
D.2	Survey questionnaire: second section	94

D.3	Survey questionnaire: third section	95
D.4	Survey questionnaire: fourth section	95
D.5	Survey results: first section	96
D.6	Survey results: second section	96
D.7	Survey results: third section	97
D.8	Survey results: fourth section	97

List of Tables

3.1	Python comparison operators (<i>Python Software Foundation</i> [42]) . . .	15
3.2	Extract of benchmark comparison carried out by Jean-Baptiste Lamy [70]	31
A.1	DKB site functions	54
A.2	DKB context functions	60
A.3	DKB concept functions	65
A.4	DKB instance functions	70
C.1	DKB find: Concept search parameters, expected values and examples (as added to [75])	89
C.2	DKB find: instance search parameters, expected values and examples (as added to [75])	90

List of abbreviations

AB	Application bug
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CWL	Common Workflow Language
DARE	Delivering Agile Research Excellence on European e-Infrastructures
DB	database
DKB	DARE Knowledge Base
DIRG	Data-Intensive Research Group
EPOS	European Plate Observing System
FD	Functionality deficiency
FIFO	First-In, First-Out
GB	Gigabyte
GO	Gene Ontology
ID	identifier
IDE	Integrated Development Environment
I/O	Input/Output
KB	Kilobyte
MB	Megabyte
MT3D	Moment Tensor in 3D
NoSQL	“Not only SQL”
ODMG	Object Database Management Group
OWL	Web Ontology Language

PC	Prefetched Cache
PDB	Python Debugger
PE	Processing Element
PID	Persistent Identifier
RA	Rapid Ground Motion Assessment
RDF	Resource Description Framework
SNS	Social Networking Sites
SQL	Structured Query Language
UC	Used Cache
UM	User Manual
UMU	User Manual update
URL	Uniform Resource Locator
W3C	World Wide Web Consortium

Chapter 1

Introduction

1.1 The 21st century - Digital Age and Big Data era

For quite a while now, the buzzword “Big Data” has been floating around. Early statements such as “data is the new oil” [58] by Clive Humby and its refined version “Big Data is the oil of the 21st century” [107] by Peter Sondergaard, Gartner’s Global Head of Research, already indicate the influence that (Big) Data already has and increasingly will gain in our everyday lives. The fact that around 90% of the world’s data has been created between 2014 and 2016 (declared by IBM, back in 2016) [99] and that Big Data gains more and more attention during happenings such as the World Economy Forum in Davos [39, 78], further back its importance.

But what does ‘Big Data’ actually stand for and what are its drivers? Due to the high attention ‘Big Data’ has attracted, a multitude of definitions has evolved. Yet Gartner’s “three Vs” (Volume¹, Velocity² and Variety³) [32], introduced to characterize data, are the foundation for most of them. An example are De Mauro et al. (2016) [29], who have built upon Gartner’s three Vs to combine the various existing Big Data definitions into one that is universal.

The growth of these (digital) data piles is accelerated by the rapid spreading of mobile devices (such as phones, smartwatches and fitness trackers [11]) and the continuous progression of the Internet. Particularly Social Networking Sites (SNS)⁴ like Twitter and Facebook (with their hundreds of millions monthly users [11]), search

¹*Volume* describes data’s fast-growing amount and size [114, 32].

²*Velocity* refers to the expediting rate at which data is being produced [114, 32].

³*Variety* represents the distinct data formats – from structured (table-like) to unstructured (e.g., images, social network posts or sound files [80]) – as well as data’s complexity [114].

⁴Such online platforms empower the users to post and discuss their opinion on any topic [11].

engines⁵ such as (the most well-known [15]) Google, Microsoft Bing or Yahoo, corporate websites⁶ and sensors⁷ contribute to the nowadays around 2.5 billion GB of daily generated data [99].

To make use of these huge volumes of un- and semi-structured data, more efficient ways of dealing with and capturing complex data are needed: relational databases (DBs) with their inflexible, tabular structure reach their limits when it comes to data that lacks structure and differs in format (e.g., pictures, sound and text files) [103, 96].

1.2 DARE & DKB - Coping with complex scientific data

Due to its multilayered, large-scale and regularly accessed data [102], science is one of the domains which is particularly exposed to the challenge of handling Big Data. With the ever-growing number of experiments, simulations and measurements, the overall data volume constantly builds up, which further exacerbates the problem and results in the evolution of more individual (division-specific) implementations [3]. These silo-structures however impede the reutilisation and sharing of acquired know-how [3], which ultimately yields in unsustainable throw-away solutions [17].

To tackle this issue, the EU Commission has launched the (recently completed) *Delivering Agile Research Excellence on European e-Infrastructures (DARE)* project [17]. DARE's underlying intention has been to introduce a "unifying hyper-platform and development context" [17], which scientific communities can utilize for their (rapid⁸) prototyping⁹ and data-intensive activities in an effective and reproducible manner [17]. Therefore, the DARE platform, takes advantage of Big Data technologies and analytics¹⁰ to ensure that domain-specific solutions can be developed [17].

One of DARE's major components should be the *DARE Knowledge Base (DKB)*, a central interface for accessing, creating and distributing data that aims to facilitate scientists' collaboration with each other [75]. To achieve this goal, an edge-cutting, new approach for subdividing and storing data has been suggested by Atkinson et

⁵Enabling the Internet's exploration for any piece of information linked to one's area of interest [11].

⁶A company's online presence usually maintains information about its business, provides the opportunity to purchase product(s) and view customers' (product) assessments [11].

⁷The most common sensors types are those installed in mobile devices and card readers, which can backtrack locations, observe health data and record transactions [11].

⁸In the prototyping context, *rapid* illustrates the pace at which each prototype is designed [31].

⁹As an operable version, the *prototype* exemplifies program properties such as layout, computed results and their calculation time [31]. With the help of a prototype, requirements can be validated and (technical) problems spotted (and thus repaired) early on in the process [31].

¹⁰So far, Big Data and Analytics have been used in commercial, but not yet in scientific contexts. [17]

al. (2019) [5]: the user-controlled contexts (figure 2.1 in *Chapter 2* illustrates their construct). The idea is to set up multiple user-specific (local) contextual workspaces and embed those in other, more general (global) context settings [5]. These global and persistent context areas then provide the generally applicable knowledge, whereas the local environments assist the use of personal assumptions and know-how during individual experiments¹¹ and (rapid) prototyping⁹ [5].

1.3 DKB's current limit - Research proposal

As elaborated above, the DARE Knowledge Base (DKB) – with its notion of user-controlled context – should be one of DARE's main parts. The emphasis however is on 'should': Since DKB's design has not progressed as fast as DARE's, it could not (yet) be adopted for DARE. Thus, additional (conceptualization and implementation) work is necessary to prove that the continuously rising data complexity can be handled more efficiently when DKB's context concept is utilized.

Correspondingly, the aim of this project is to improve the existing DKB application by determining and resolving one or more defined issues, while keeping in mind that ultimately any enhancement(s) should benefit the (scientific) user community. To identify possible key topics, a thorough system analysis will be conducted. Since this project is primarily a research one with some development percentage, the problem investigation will pay particularly attention to DKB's conceptual weaknesses. Accordingly, the objectives can be divided into three coherent stages:

1. Identification of current weaknesses
2. Topic selection and resolution
3. Achievement evaluation

To begin with, an exhaustive test and analysis phase is carried out to reveal DKB's existing deficiencies. Besides the creation of a sufficiently large amount of (test) data¹², both the execution of functionality and performance tests and the examination of their results is involved. Outcome of these first tasks is the generation of around 390 MB of synthetic and real-world test data as well as the exposure of three general problem clusters – namely, 'application bugs (ABs)', 'functionality deficiencies (FDs)' and 'User Manual updates (UMUs)' – to which all the individual findings can be allocated. After

¹¹For example, users can produce 'new sets of validated time series, new versions of an analysis process, new methods and new concepts' [5].

¹²An adequate test database is a prerequisite for all subsequent (functionality and performance) tests.

those weak spots have been pointed out, the most practicable¹³, acute and research-intensive issues are picked as the thesis' main topics:

- Enhancement of the `find()` function (more search criteria and operators)
- Establishment of a conceptual model of *collections*

To focus on these though, the detected application errors have to be fixed first since they severely hamper DKB's usability¹⁴. This however then enables the conceptualization and implementation (design) of the main tasks. Finally, the bug fixes and `find()` improvement developments are examined the same way as before: with functionality and performance tests. Additionally, a user survey has been prepared to measure all achievements' value via a user rating. The test and assessment results clearly prove the implementations' and conceptualizations' success by passing all the test and receiving an average rating of 4.9 (find) and 4.5 (collections) out of the maximum score of 5.0.

Paper outline To accomplish the above described objectives, the thesis is split into a total of four sections: First of all, *Chapter 2* provides the relevant background information on the DARE Knowledge Base (section 2.1) and NoSQL databases such as Owlready2 (section 2.2), since the DKB is based on Owlready2. This is followed by the main and most important part, *Chapter 3*, which details each of the five project phases: 'test data generation' [section 3.1], 'system analysis' [section 3.2], 'project focus' [section 3.3], 'software development' [section 3.4] and 'achievements' assessment' [section 3.5]. Sections 3.1 and 3.2 thereby target the first objective, the identification of existing weaknesses, while sections 3.3 and 3.4 aim at the second object, the (focus) topic selection and solution. The dissertation's major contribution is also located in section 3.3, which raises and answers various research questions to:

- Cover the reasoning for the choice of certain (focus) topics,
- Explore possible solution approaches (literature-based) and
- Conceptualise their solution design (section 3.3.3)

Section 3.5, which deals with the third and last objective - the solutions' test and (user) evaluation, concludes the third chapter. It should be noted that the three problem clusters – 'ABs', 'FDs' and 'UMUs' – and their order are kept throughout sections 3.2, 3.3 and 3.4. Finally, *Chapter 4* discusses the achievements as well as their evaluation, while critically reviewing those and any open points that have been left as future work.

¹³With respect to the imposed time constraint of two and a half months

¹⁴Making it close to impossible to properly work with the application prototype.

Chapter 2

Background and Related Work

2.1 The DARE Knowledge Base (DKB)

DKB's functional concept As stated in the beginning, the era of 'Big Data' with the rapidly growing amounts of interlinked, manifold and complex data is now. Still, particularly the scientific community lacks reproducible and reusable solutions for handling and analysing these (ever-growing) data volumes [17]. New approaches are required and this is where DKB comes into play: it assists scientists from any domain by implementing the completely novel concept of user-controlled contexts¹. The underlying idea of setting up one (or more) individual (local) context workspace(s) which is (/are) integrated into surrounding, more generic (global) contexts (the closer to the outer layer, the more general the context; pictured in figure 2.1) [5]:

- Ensures that users have direct control over the contexts and
- Enables a more informal, natural way of thinking (without having to consider formalities such as ontologies²).

Additionally, this approach satisfies two other key points:

- Data dynamics: since data nowadays is widely distributed and continuously updated, DKB's method is to store references to already existing data instead of physically saving (thus, duplicating) the (constantly changing) source data [7].
- Conceptual autonomy: it is quite common that multiple scientists are concurrently working with the same data. In such cases it is essential that each of them

¹As of now, contexts can incorporate *concepts* (abstract conceptual definition, e.g., of a workflow or experimental run) and *instances* (instantiation of previously defined concepts, e.g., containing execution parameters); partially realizing the specification of "data, methods, concepts and collections" [75].

²An ontology is a "knowledge base system" which consists of "terms" (domain-specific concepts) and "semantics" (relationships between the "terms") [69]. Its main purpose is to use this uniform and formally specified vocabulary to ease the reuse and distribution of know-how [51, 7].

can independently model and test ideas, without being limited by the others [7].

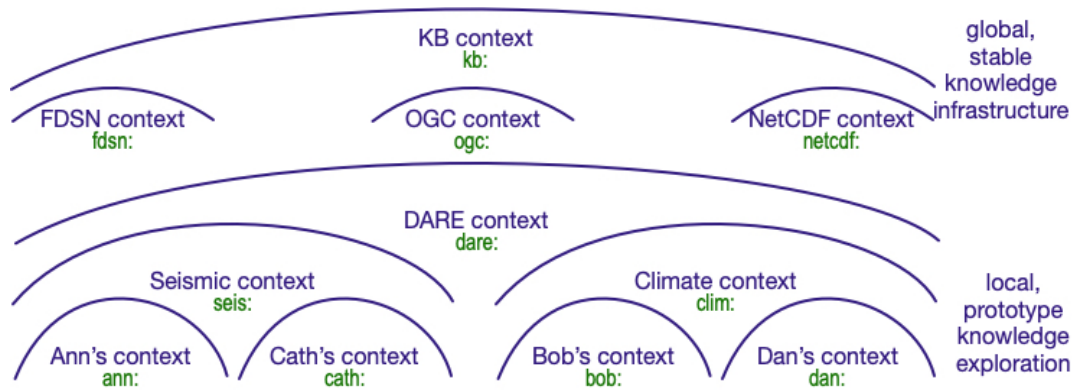


Figure 2.1: Concept of user-controlled contexts (Atkinson et al. (2019) [5])

Overall, DKB helps prevent throw-away solutions³, manage complex data⁴ and ease the running of data-driven experiments and their outcome distribution⁴ [17, 75].

DKB's technical set-up As displayed in figure 2.2, the client-server DKB application comprises a backend server and a frontend client [75]. Every user is expected to set up and interact⁵ with their own (personal) DKB client, which consists of a Python library – `DKBlib` – and a web service [7]. The web service uses `Owlbready2` (further details about `Owlbready2` in *section 2.2.3*) to convert the ontology²-like information to the Web Ontology Language (OWL)⁶ [7, 76].

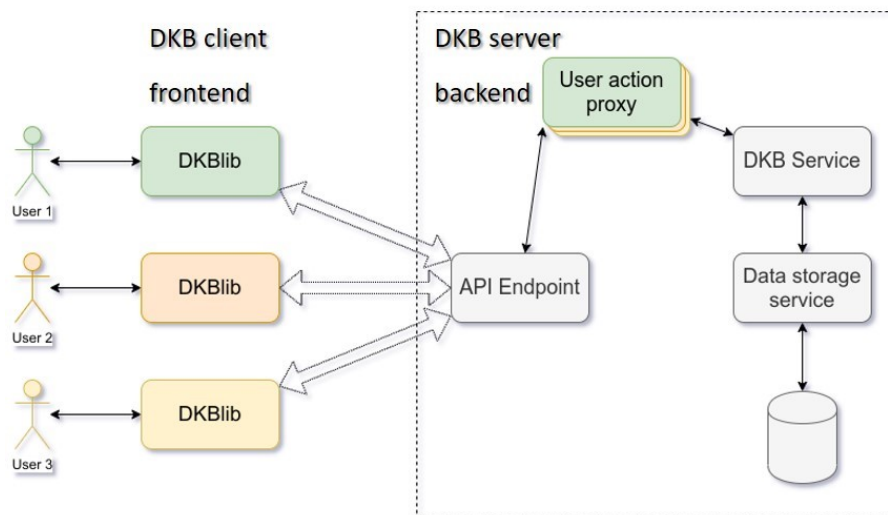


Figure 2.2: DKB architecture, taken from Levray and Zhao (2020) [76]

³By providing solution which are tailored to the individual area of expertise and reusable [17].

⁴Through the DKB interface, which is responsible for accessing, creating and spreading data [75].

⁵For instance by utilizing the web-based Jupyter notebook [7]

⁶OWL is World Wide Web Consortium (W3C)'s (Semantic) Web language for portraying ontologies (“rich and complex knowledge about things, groups of things, and relations between things”) [48].

Right now, the DKB server has to be installed locally before DKB can be launched for test or development reasons. In future however, the DKB server will either become an element of the DARE hyper-platform or run as a standalone solution [75]. Figure 2.2 demonstrates the DKB server's four main components [76]:

- The *Application Programming Interface (API) endpoint*, in charge of handling the communication between user client(s) and the server the client is logged into.
- An *user action proxy* to translate the user inputs/ commands into their corresponding server functions and afterwards call those.
- The *DKB service* with its server-side methods to verify and execute the user activities (and potentially connect to the *data storage service* to retrieve DB data).
- The *data storage service* to access, refresh and save the queried data from/ to the database without prior validation (the *DKB service* takes care of the verification).

2.2 Owlready2 - Following the graph database concept

2.2.1 Brief introduction to NoSQL databases

In particular the recent 'Big Data' challenge and the thereto relating need to handle and store massive amounts of manifold and connected data gave rise to (various types of) the so-called "Not only SQL" (NoSQL) databases. Usually NoSQL databases are divided into four main groups⁷:

- Key-Value stores: since the data is kept in a schema-less way, the data objects can only be queried via their key values [79].
- Document databases: unlike *key-value stores*, document DBs are structured [79]. However in contrast to *relational databases*, they do not have one overarching schema all documents have to comply with [79].
- Columnar stores: as opposed to the row-oriented *relational DBs*, *columnar stores* pack their data into column-wise blocks [79].
- Graph DBs (covered in depth in *section 2.2.2*): keep data in a graph structure and are therefore ideal for highly interlinked data.

Unlike relational (alternatively also called 'Structured Query Language (SQL)'⁸) databases, NoSQL DBs rely on an adaptable schema and do not adhere to the (rigid) rela-

⁷Nonetheless, also notable NoSQL databases are the "scientific DBs" such as the SkyServer database, which contains data about the universe (for example, galaxies and stars images) [47, 1].

⁸Derived from the fact that SQL is relational DBs' most frequently used query language [82].

tional data model⁹ [79]. This provides them with the necessary flexibility for dealing with large amounts of both structured and unstructured data [10].

One major disadvantage almost all NoSQL DBs types have in common is the fact that they – alike *relational DBs* – predominantly store chunks of disconnected data, which limits their use for highly interlinked and complex data [96]. *Graph DBs* overcome this downside by storing the data in graph structures, which can represent connectivity via graph relationships [79]. Therefore, especially *graph databases* are becoming more and more attractive for depicting diverse and complex connections [79].

2.2.2 Graph databases - One of four main NoSQL types

A graph essentially consists of three main components: nodes or vertices, relationships or edges and constraints [116]. *Nodes* “typically represent some entity, such as person [or] product” [116] while *relationships* help connect these nodes with each other [116]. *Constraints* are imposed to ensure certain properties are filled: for instance, a person node may require forename and surname [116]. Figure 2.3 shows an illustrative graph, where nodes are drawn as circles and relationships as pointing arrows (to indicate subject and object) with descriptive predicates (e.g., ‘HAS_INSURANCE’) [116].

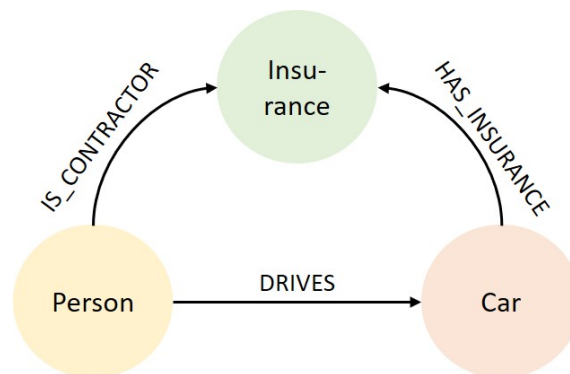


Figure 2.3: Sample graph, taken from Webber and Van Bruggen (2020) [116]

Graph databases for scientific purposes Not only commercial firms such as flight booking services, LinkedIn or Facebook increasingly benefit from graph databases, also science progressively exploits (and thus profits from) them [120]. For this reason, researchers like Yoon et al. (2017) [120], Salehpour and Davis (2020) [98] or Effendi et al. (2020) have investigated their eligibility for managing huge volumes of scientific (e.g., biological and spatio-temporal) data [98, 35], which includes the comparison

⁹The relational data model captures data in a tabular (row- and column-based) structure and is still the most regularly applied model [115]

with other (partially open-source) edge-cutting relational databases like MySQL or PostgreSQL [35, 120]. Their findings confirm graph DBs' superiority when it comes to response time [35] and performance (improvement of up to 40%) [120].

Further, more earth-related examples are graph DBs' usage for power grids [93], water drainage networks [23] or bioinformatics databases such as Reactome [37, 50].

DKB's key differentiator from other in use graph DBs are the user-controlled contexts¹⁰. As of now, such an approach has neither been examined nor put into practice.

2.2.3 Owlready2 - An optimized quadstore

The DARE Knowledge Base (DKB) implements Owlready2, a Python package for ontology-oriented programming which means that “OWL⁶ 2.0 ontologies” can be loaded, altered and maintained as Python objects [72]. Owlready2 comes with a (performance- and memory consumption-wise) optimized quadstore, which is based on a SQLite3 (file) database [72]. The ‘quad’¹¹ store thereby defines *how* all information is kept (as “RDF¹² quads”), while the SQLite3 DB deals with the *where*: it provides the flexibility to either retain all data quads in memory or store them on a harddrive (by default they are kept in memory) [72]. The persistent storage in the SQLite3 file (database) is particularly appealing for huge ontologies¹³ since loading all their data into memory takes a while whereas opening a (database) file and retrieving only a few (requested) quads can be significantly faster [72]. Moreover, Jean-Baptiste Lamy, Owlready2's inventor, claims “that Owlready2 [even] performs a little faster when storing the quadstore on [...] disk” [72]. In general, Owlready2 retrieves all information dynamically once it's queried, caches it in memory and discards it later, when it is no longer required [72].

Thus, although Owlready2 contains an integrated, relational database, it captures and treats data like a graph database (using RDF¹² quads¹¹ and respecting relations between different quads) and thereby follows a graph database's main concept.

¹⁰Introduced to easily gather, reuse and share knowledge [17, 75].

¹¹‘Quad’ due to its four elements: “(subject, predicate, object, ontology)” [72].

¹²The Resource Description Framework (RDF) is one of W3C's information exchange guidelines, which integrates data relations as directed, tagged graphs [49].

¹³Such as the Gene Ontology (GO) with its round about 170 MB of data [72].

Chapter 3

Approach, Methodology and Achievements

The (investigative) work performed on the existing DARE Knowledge Base (DKB) prototype is a combination of research and software development, with the overall goal of enhancing the current application by:

- Determining existing deficiencies and limitations,
- Selecting and solving a subset of those and
- Assessing the added value these application improvement(s) offer.

For this reason, the project has been divided into five consecutive phases, as illustrated in figure 3.1, which will be elaborated further in the subsequent sections.

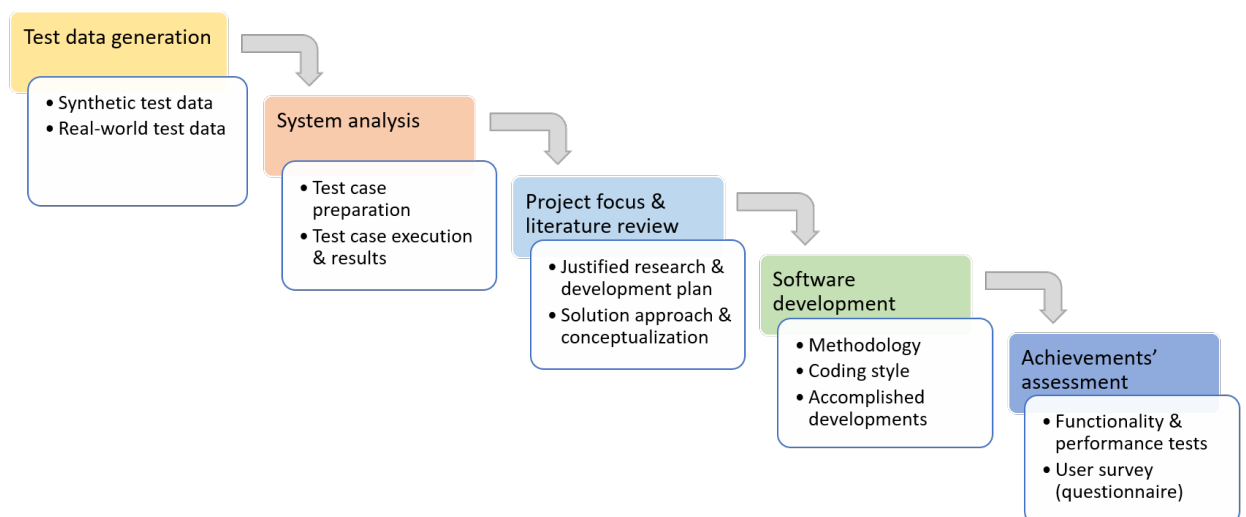


Figure 3.1: Project phases

3.1 Test data generation

A balanced mix of a quantitatively adequate number of (test) data records and a realistic (test) data representation is a prerequisite for performing a thorough system analysis. For this reason - and because the existing database consisted of only a handful of data samples so far -, two types of test data have been generated:

- Synthetic data, to satisfy the quantitative need for mass data, and
- Real-world data, to meet the demand of a sufficiently realistic data set.

Overall, the underlying database volume has been enlarged from around 20 KB of data to around 390 MB¹ or 0.38 GB. This amount has been considered appropriate since it roughly doubles the size of the GO (170 MB), which the inventor of Owlready2, Jean-Baptiste Lamy, calls a large ontology and which - according to him - takes noticeable time to load [72].

3.1.1 Synthetic test data

Following DKB's approach of utilizing user-controlled contexts and the embedded concepts and instances to partition and store data [5], a *synthetic data generation program* with adjustable parameters, through which the number of created contexts, concepts per context and instances per concept can be controlled, has been set up. Additionally, one can specify a general context, concept and/ or instance name prefix, which is then complemented by a (continuously increasing) numbered counter to create each (synthetic) context's/ concept's/ instance's name.

For re-usability reasons and because the scientific community, whom DKB aims to support during their day-to-day business, works with Jupyter notebooks almost daily, the *synthetic data generation program* is not only available as an executable Python file but also as a Jupyter notebook with detailed descriptions and (usage) instructions (located on [Gitlab](#) [54]).

3.1.2 Real-world test data

As opposed to the (automated) generation of large amounts of synthetic test data, the real-world samples have been manually designed in close consultation with former DARE (Delivering Agile Research Excellence on European e-Infrastructures) and

¹Most of this is synthetic data, which has been generated by multiple runs of the below described program on a personal laptop. In total there have been ten runs, of which each has run for several days to create 10 contexts with 50-100 concepts per context and 100-1,000 instances per concept.

DKB scientists. Based on the DARE European Plate Observing System (EPOS) use cases [28], samples for both *Rapid Ground Motion Assessment (RA)* and *Moment Tensor in 3D (MT3D)* scenarios and their associated workflows [25] have been considered.

As part of the *RA*, seismologists evaluate seismic wavefields which appear straight after (large) earthquakes to immediately create ground motion predictions (e.g., in terms of speed and acceleration) and apply them as part of their emergency response [27]. Later, these estimations are compared with the actual, registered ground motion data to further advance their seismological models and knowledge of the Earth's behaviour after an earthquake [27].

Similarly, the *MT3D* also concentrates on earthquakes: the focus here lies on their source parameters (mainly, location and moment tensor for characterizing the earthquake rupture's magnitude and mechanism) and the uncertainty assigned to the parameters' impact on wave transmission and risk assessment [26]. Using 3D instead of the previous 1D seismic wavespeed models and running multiple simulations with varying perturbed source parameters, helps reduce the gap between the observed and the artificial wavefields and identify the most realistic model (parameters) [26].

For both of the above described scenarios, workflows² (with according parameters) and runs have been set-up as part of DARE EPOS [25]. Therefore, this general, abstract workflow and run structure has been mapped to DKB's concepts, whereas the actual workflows and runs with their execution parameters have been added as DKB instances.

Alike the synthetic data generation, the real-world records are also stored in both an executable Python file and a Jupyter notebook with detailed descriptions and (usage) instructions³ (and are again located on [Gitlab](#) [53]).

3.2 System analysis

Based on the previously created synthetic and real-world test data, an extensive system analysis, consisting of two kinds of tests, has been performed:

²Namely, the Python-based `dispel4py` [109] and the YAML-based Common Workflow Language (CWL) [4] to cover both streaming- (several tasks can run concurrently, as soon as the required resources are available) and task-oriented (each task waits for the preceding task to complete) workflows [65, 38, 25, 24]. With `dispel4py`, each task or Processing Element (PE) is essentially a Python class and represents a workflow step, which `dispel4py` eventually joins together in a graph [38]. Thus, `dispel4py` contains the entire workflow logic [38]. CWL on the contrary, does not implement the logic: instead, YAML files are used to specify the order of the various scripts, which contain the workflow logic [38].

³Reasons: re-usability and scientists' habit of preferring Jupyter notebooks

1. On the one hand, *functionality tests* have been set up to analyse the existing prototype's functionality and answer two research questions:
 - Does the application and its functions work as promised?
 - Is the prototype with its current functionality what (scientific) users expect?
2. On the other hand, *performance tests* have been included to cover non-functional specifications, evaluate the system's behaviour and speed and thereby solve the question of whether the current performance is sufficient to keep users engaged.

3.2.1 Test case preparation

The first task hereby has been to build test cases according to DKB's main features and functionalities. Therefore, not only the User Manual⁴ [75], Design⁵ [8] and Specification⁶ documents [76], but also the supervisor and former DARE scientists have been consulted. Together with some general instructions and detailed explanations, all the thereby identified test scenarios have been then captured in a [Jupyter notebook](#)³ [56].

Thereafter, performance measurements – in form of taking the exact time before and after each function's execution, to calculate the difference between these two points in time and report the overall execution time – have been added to the Jupyter notebook. The decision to use Python's built-in *datetime* [43] library – instead of an external software – for this matter, is based on several factors:

- First of all, their – for this straightforward use case of measuring only response time – functionality overload: 'Solarwinds Database Performance Monitoring' for instance offers (24/7) database health, performance troubleshooting and code deploy features in addition to the – for this project solely required – system performance function [77]. And both 'Apache Jmeter' [40] and 'LoadNinja' [105] for example provide supplementary test (plan) recordings [40, 105], an offline test result access, a command-line mode and dynamic HTML report [40].
- Secondly, their costs: due to their extensive range of features most of the software is chargeable and/ or offers solely a 14-days trial period.
- Thirdly and lastly, the set-up procedure: all of the researched software products require an installation and set-up process as well as some time for familiarisation, whereas the Python built-in libraries are well-known and fairly easy to use.

⁴Besides describing DKB's available functions, the User Manual lists all possible input parameters and gives illustrative examples. [75]

⁵The DKB Design paper focuses on functionality and their implementation. [8]

⁶Alike the DKB Design document, DKB's Specification explains DKB's purpose and functionalities as well as the current implementation approach. [76]

3.2.2 Test case execution and results

After all the collected test cases had been written in Python ([Jupyter notebook](#)³ [56]) and captured in a test document (Appendix A), they have been executed and their results – together with illustrative screenshots – recorded, both in the test document (Appendix A) and a [GoogleDoc](#) [6].

In general, the findings can be clustered into three broad categories:

1. *Application bugs (ABs)*: all cases, where the research question, whether the application and its functionalities work as promised, has been denied.
2. *Functionality deficiencies (FDs)*: those occurrences, where the application with its existing functionality differs from what a typical (scientific) user would expect or where the performance is insufficient to keep the users engaged (thus negating the second and third research inquiries).
3. *User Manual updates (UMUs)*: documentation-related and dealing with the query of whether the User Manual is lacking helpful⁷ information.

Please refer to the detailed test documentation in Appendix A for more exhaustive specifics than the following summaries.

3.2.2.1 Application bugs (ABs)

The application’s functionality is clearly not what users demand when they execute the functions precisely as described in the User Manual [75] and receive an error message instead of the (expected) result. This however has happened for the following operations, which therefore have been logged as application bugs:

- Instance creation with `new_instance()`: returns an *Internal Server Error*
- Concept and instance lookup with `find()`: results in various *type errors*
 - `TypeError: (“get_instance() missing 1 required positional argument: ‘name’”)`
 - `TypeError: unhashable type: ‘dict’, []`
 - `TypeError: “unhashable type: ‘IndividualValueList’”, []`

as well as an *Internal Server Error*

- Concept and instance data access with `get()`, `find()` and `context_reset` after a DKB server restart: again, returns either an *Internal Server Error* or a *TypeError*: (“`get_instance()` missing 1 required positional argument: ‘name’”)

⁷Helpful in terms of explanations and clarifications to

- Avoid confusion with regards to what is possible and what (potential) prerequisites are,
- Ensure a correct application usage,
- And expand the users’ knowledge (with respect to input parameters for instance).

3.2.2.2 Functionality deficiencies (FDs)

Functionality expectations The scientific user community, whom DKB is targeted at, consists of well-trained mathematicians (e.g., seismologists and climatologists) who are familiar with Python and work with Jupyter notebooks almost daily. Therefore, DKB’s functions and their (input) parameters are somewhat supposed to cover Python’s basic principals such as the variety of data types (‘integers’, ‘floats’, ‘strings’, ‘booleans’, ‘lists’ or ‘dictionaries’, to name a few [41]) or the (comparison) operator scope [42], which is pictured in table 3.1.

Operation	Meaning
<	Strictly less than
<=	Less than or equal
>	Strictly greater than
>=	Greater than or equal
==	Equal
!=	Not equal

Table 3.1: Python comparison operators (*Python Software Foundation* [42])

Performance benchmark Moreover, several user studies [84, 13, 87] (a.o. by behavioral scientist Robert Miller (1968) [84]) have proven, that users are rather impatient and tend to get distracted quite easily if the system’s response time is too long (in general, more than 15 seconds [84]). The most widespread and quoted (e.g., by Nielsen (1994) [85], who himself is frequently cited) advice with regards to an application’s response time has broadly been the same for the last 50 years [84, 13]:

- Solely a splitsecond (0.1 second) ensures the users’ perception of an instantaneously reacting system [85].
- 1-4 seconds are required to keep the users committed to the interaction [13, 85]. App users for instance, demand (mobile) apps to react within two seconds or less [87], which already Robert Miller, back in 1968, has reported to be users’ response expectations to simple queries [84]. For more complex queries though, up to four seconds are allowed although two seconds are still preferable [84]. If however, this response time is exceeded, the users are likely to feel bored while waiting for a response [13].
- Finally, data loads should ideally be finished within 15 seconds, which – according to Miller (1968) – is also the maximum time the user stays in a “problem-

solving” mindset [84]. Similarly, Card, Robertson and Mackinlay (1991) [13] investigated that 5-30 seconds are acceptable when completing very complicated requests [13]. Jim Gray, Alexander Szalay and colleagues [108, 86] also stress the importance of parallel processing for reducing a query’s execution time⁸ [86] as well as a “as-soon-as-possible data push strategy”⁹ [108].

Especially for rather simple¹⁰ (data querying) functions, such as DKB’s, exceeding these timeframes quite likely leads to user distraction [84]. Accordingly, they have been considered as the execution time limits.

Results Accordingly, the observed functionality deficiencies are as follows:

- The concept creation (`new_concept()`) parameters can take ‘string’ and ‘integer’ types and none of the other basic Python types such as ‘boolean’ or ‘float’ [41].
- As of now, one cannot assign default values to the concept creation (`new_concept()`) parameters, which again is common Python practice when specifying function/method arguments [66]. For example, a DKB concept defines the general object structure (like a function) and this concept’s DKB instances then populates the frame with actual values (similarly to the function call). If a user – during the instance creation `new_instance()` – prefers to not specify each parameter’s value, but to instead rely on default settings, he/ she can currently not realize this.
- At present, users can solely perform equality checks with two search criteria (`PID` and `prefix`) with the `find()` function. Other typical¹¹ arithmetic comparison operators, as listed in table 3.1, or search parameters are not available.
- In their conference paper, Atkinson et al. (2019) [5] briefly touch on the idea of collections as a part of their newly introduced context concept. They explicitly state the requirement of having “to support all four aspects of a context: concepts, methods, data and collections” [5], however the collections proposition has neither been conceptualized nor realized so far.
- The execution performance measurements revealed that:

⁸Since users might “be willing to wait for 3 minutes but very few will ever wait for 11 minutes in front of their browsers” [86].

⁹It ensures that users can instantly view all the (up to the current point in time) selected data records, which match the query condition(s) [108]. This is particularly beneficial for long-running queries [108].

¹⁰As opposed to more computational-intensive tasks (e.g., machine learning with its training runs) for which users are prepared to wait for days.

¹¹For SQL queries (`WHERE` condition) [94] or various Python (comparison) operations which for instance are used in `IF`-statements [42]

1. The response time increased with the amount of available data.
2. On an adequately large underlying database (in this case of around 390 MB or 0.38 GB), various DKB functions can take multiple (up to 90) seconds and therefore potentially harm the user engagement:
 - Especially the initial login, the closing logoff and the ongoing session's first concept creation (`new_concept()`) last up until 90 seconds (exceeding Miller's (1968) reported maximum acceptable data load delay of up to one minute [84]).
 - Other function calls such as `get()` or a (complex¹²) `find()` also take up to five or 15 seconds respectively (again potentially exceeding the upper tolerance limit for simple and complex requests [84, 13]).

3.2.2.3 User Manual updates (UMUs)

Together with the following test execution and result documentation, the review of the User Manual (UM) (as part of the initial test case creation) has revealed a few information shortcomings. For instance, the following topics have not been covered:

- For Windows users, the DKB set-up has led to some unforeseen installation and run issues which are not described in the User Manual.
- Furthermore, neither the possible concept and/ or instance `find()` search criteria nor their expected values are outlined anywhere.
- Additionally, the naming convention for (context) creations is not specified.
- Finally, some functionality prerequisites and/ or conditions are missing, for example when entering, leaving, freezing, deprecating or resetting a context.

All of these instruction shortages have resulted in extra work¹³ and time spent to figure out how to correctly run these functions.

3.3 Project focus and literature review

As elaborated in the introduction, this section contains the dissertation's main contribution: the solution conceptualizations in *subsection 3.3.3*. To prepare for the resolution drafting, a variety of research questions are raised which:

- Justify the choice of the selected (focus) topics and
- Perform an evaluation of possible (literature-based) resolution approaches,

¹²The complexity hereby derives from query nesting (through 'AND' and/ or 'OR').

¹³In terms of multiple (trial & error) executions as well as some code analysis

before eventually conceptualising a solution for the chosen key topic. Since this project is primarily a research one with some development percentage, the topic selection will particularly consider DKB's conceptual weaknesses while answering the following (*subsection 3.3.1*) two investigative (user- and developer-related) questions.

3.3.1 Test result assessment, Issue selection and justification

The comprehensive system analysis has exposed more application bugs and functionality deficiencies than can be resolved within the given time frame of two and a half months. Therefore, a feasible and achievable subset of the identified issues has to be selected. The decision, which of the detected problem(s) to tackle, has been based on an (internal) prioritization, which takes the following questions during each issue's evaluation into account:

- From a user's perspective: Does an alternative or workaround exist? Does/ Do the issue(s) negatively impact DKB's usage?
- From a researcher/ developer's angle: How likely is it to conceptualize/ implement the solution(s) in the time given?

3.3.1.1 Application bugs

All of the discovered prototype defects severely harm both the users' experience and DKB's usability in general, since certain (expected) features are not working at all. Instead, their execution aborts with various error messages. Because all the detected application bugs compromise DKB's usability so severely, partially even making it nearly impossible to work with the prototype, their resolution becomes a prerequisite for any later (conceptual or implementation) improvements. Since the investigation, debugging and development effort also seems manageable from an analysis and implementation point of view, they will be resolved as part of this project.

3.3.1.2 Functionality deficiencies

Data types At first sight, the limited data types (solely 'string' and 'integer') might very well damage the users' acceptance and usage of the DKB prototype. But in principle, users can still record 'boolean', 'dictionary' or (rounded) 'float' entries: as an (intermediate) workaround the other data types' values can be captured as either a 'string' or an 'integer', which significantly reduces the negative impact caused by their unavailability. Moreover, it has been possible to create all the exemplary real-world

use cases with the current solution, which again proves that resolving this matter is less important than others.

Therefore the conclusion is drawn to postpone this resolution to a later point in time (leaving it either as future work or picking it up towards the project end).

Default values Similarly, the standard value option is more ‘nice to have’ (to help cut down manual user inputs and thus effort) than a serious functionality constraint: as of now, users ‘have to’ enter the default values and cannot define them solely once. Also, as mentioned before, so far all the illustrative real-world use cases could be generated with the existing solution, which shows that this matter is less important than others.

Hence, the decision is made to defer this problem as well (leaving it again either as future work or picking it up towards the project end, if time allows).

Find functionality As highlighted in paragraph *Functionality expectations*, a typical DKB user, who is familiar with the Python language, its structure and concepts, requires certain basic operators besides the equality (‘==’) operation. The lack of these (comparison) operators hampers DKB’s data querying capabilities quite substantially and therefore makes a notable conceptual shortcoming. Furthermore, with only two parameters the range of available search criteria is very restricted, which again negatively affects DKB’s data retrieval ability. An efficient implementation, which achieves an adequate response time while the complexity of the (data) selection expression grows, thereby presents a core research challenge.

Since both the research (of additional search parameters, comparison operators and ways of efficiently implementing those) and their successive development appear achievable within the two and a half months, the find functionality enhancement is added to the project scope as one of its major contributions.

Collections concept As stated before, Atkinson et al. (2019) [5] have suggested the introduction of collections to DKB. Since this proposition has not yet been investigated (which again forms a conceptual weakness), it will be explored as part of this project.

People have the natural tendency to collect things and the scientific user community is no different: they ‘collect’ anything from (input) variables, to simulation results and their true (observed) outcomes as well as new occurrences [5]. Therefore, enabling this common behaviour as part of DKB would greatly benefit the targeted scientific users and make the application even more appealing to them. It also poses research issues

regarding the nature of collections, their representation and performance. If users can gain control over a collection while its instantiation minimises data traffic (within DKB or even between DKB and a remote system), substantial savings will ensue.

Since the general conceptualisation also seems doable within the given time frame, the investigation of collections becomes a major research focus.

Performance improvement As pictured in paragraph *Performance benchmark*, some general (application-specific) response time boundaries do exist [84, 13, 87] and breaching those – as in DKB’s case – may quite quickly lead to a loss of user attention and engagement [84]. However, unlike other applications, the DARE Knowledge Base (DKB)’s (test) data stock consists mainly of artificially created (synthetic) data, which possibly is the reason why the performance results could be viewed with suspicion: the generation of large amounts of nested data¹⁴ might have been carried too far¹⁵, thus one should exercise caution with those findings. To put them into perspective, additional research questions have to be considered:

- Do other underlying DBs handle large amounts of nested data more efficiently?
- What other options for improving the current system’s performance remain?

Only thereafter a (realistic) estimate regarding the extent, to which some of the researched and outlined suggestions should and can be realized, can be given.

3.3.1.3 User Manual updates

All of highlighted instruction shortcomings have led to extra work and additional time spent on figuring out how to correctly run the functions. Therefore, adding the described pieces of information will help DKB’s usability and comprehensibility. Moreover, such elaborations are perfectly viable within the available time frame and will therefore be completed as part of this project.

3.3.2 Formulated research and development plan

In conclusion, the following research and development plan has been framed in close consultation with the supervisor:

¹⁴Context with hundreds of concepts, each of them again consisting of hundreds (sometimes even thousands) of instances.

¹⁵The real-world samples are nowhere near the synthetic data’s nesting. Additionally, Jean-Baptiste Lamy, the inventor of Owlready2, noted that the loading of large ontologies such as the GO, around 170 MB of data which is less than DKB, can take a while [72].

- Bug fixing, including the revision and correction of the existing ontology mappings, to recover DKB's functionality and ensure the application's usability.
- Enhancement of the `find()` function (adding further search criteria as well as operators such as inequality and range) in combination with the
- Establishment of a conceptual model of collections, to further broaden DKB's feature range and meet users' expectations regarding (basic) functionality.
- Research performance benchmark results as well as improvement possibilities to boost response time and keep users engaged.
- Perform necessary UM updates to avoid misunderstandings and waste of time.

3.3.3 Solution approach and conceptualisation

Based on the above presented research and development plan, this section states each focus topic's underlying research question(s), outlines the approach for resolving the matter and covers the detailed solution conceptualisation where applicable.

3.3.3.1 Bug fixing

The key to resolving any application errors is to narrow down and identify their root cause. Only thereafter, possible fixes can be investigated and implemented. Thus, the two questions that arise as part of DKB's bug fixing task are:

- Which of the existing debugging¹⁶ strategies can be adopted?
- Which debugging tools¹⁷ are available to facilitate the root cause determination?

Debugging strategies Although debugging is one of the programmer's most fundamental, constantly used skill, it – together with potential strategies – is hardly ever taught and therefore acquired 'the hard way' in practice, once programs crash or output something unexpected [83]. Still, a variety of debugging strategies, commonly divided into **forward reasoning** and **backward reasoning** [63], exists. *Forward reasoning* thereby refers to starting the bug search from the written program code itself, whereas *backward reasoning* comprises strategies that begin with the program's erroneous behaviour (e.g., an incorrect output) and work backwards, towards the problem's cause

¹⁶Debugging, one of the most crucial programming competence, is commonly known as the approach of revealing and resolving (application) errors [83].

¹⁷The debugger is the most widespread and well-known tool, however of course the overall toolset is not limited to the debugger [83].

[97, 63]. Katz and Anderson (1987) [63] list *comprehension*¹⁸ and *hand-simulation*¹⁹ as examples of the first, and *simple mapping*²⁰ and *causal reasoning*²¹ of the later.

Since programmers, when using the two mentioned *forward reasoning* procedures *comprehension* and *hand-simulation*, only focus on the written code itself and do not run it step-by-step (e.g., with the help of breakpoints), Romero et al. (2007) [97] introduce the so-called *following execution*. *Following execution* extends the *forward reasoning* methods by adding visualisations and the gathering of code information through a stepwise code execution [97]. And Whalley et al. (2021) [117] propose to make a distinction between ‘static’ (simple code scanning) and ‘dynamic’ (including the usage of print statements) *comprehension*.

More recent approaches than Katz and Anderson’s (1987) [63] (refined)²² *forward* and *backward reasoning* are the

- **Scientific Method** [122] and
- **Algorithmic** (or declarative) **debugging** [104].

The *Scientific Method* is named after (natural) sciences, since the scientists’ approach of formulating a theory is adhered to [122]. Therefore, this model implies roughly the following steps: First of all, excessive testing to identify program bugs (“observation(s)” [122]) [83]. Thereafter, phrasing of (root cause) hypotheses (“tentative description”, in line with previous “observation(s)” [122]), which are then either proven or refined through (multiple) experimental runs (repeated “experiments”, until the divergence between “hypotheses” and “experiments” is eliminated [122]) [83].

The *algorithmic debugging* is a semi-automatic method, where the debugger produces a set of questions²³, which an “oracle” (usually the programmer) answers [104]. Those replies then tell the debugger, whether the program’s (intermediate) computation is working as requested or not [104].

Because the DKB application and its coding had been completely unknown beforehand, a mixture of *dynamic comprehension*, *following execution* and *simple mapping* has been chosen. With the help of *dynamic code comprehension* and numerous print

¹⁸During the *comprehension* process, the programmer constructs a (mental) program representation, for instance by transcribing the code [97, 63].

¹⁹When ‘*hand-simulating*’, the programmer acts as a computer to analyze the code likewise [63].

²⁰*Simple mapping* implies that the program result itself immediately leads to faulty code line(s) [63].

²¹Alike the more general *backward reasoning*, *causal reasoning* commences from the wrong output and heads backwards (by utilizing one’s own insights into the underlying program) to find the bug [63].

²²For instance by Romero et al. (2007) [97] and Whalley et al. (2021) [117]

²³Whether or not the output of an (intermediate) calculation is correct or not

statements, an initial familiarisation with the application logic and coding has been achieved. Thereafter, mainly the program execution strategy has been followed in a *forward reasoning* manner by utilizing *following execution* and various breakpoints to verify temporary results and to find the issue's root cause. Partially, also the *simple mapping* procedure has been adopted for cases, where the error messages have been self-explanatory and pointed to dedicated code lines.

Main debugging tool Generally speaking, there are two prominent Python debugger options:

1. Python's built-in debugger PDB [44] and
2. The Integrated Development Environment (IDE)'s (internal) debugging alternative, in this case Atom's²⁴ python-debugger package [21].

The Python Debugger (PDB)'s major advantage over Atom's debugging solution is the fact, that one can both view (temporary) variable values and evaluate (partial) expression results anytime [61], whereas Atom's Python debugger does not offer "watched variables or expressions" [21]. Additionally, PDB provides a wide variety of commands to (slowly) step through the code and reproduce each step's results [61]. Although the circumstance of interacting with PDB via the command prompt might put people off, it is more convenient in the DKB context since the DKB server communication also happens completely via the command prompt. For all this reasons, the tool of choice is Python's built-in debugger PDB.

3.3.3.2 Find functionality enhancement

DARE Knowledge Base's find feature basically represents a different way of querying data from the database according to certain selection criteria, similarly to what users can achieve with a Structured Query Language (SQL) query on a relational database. To define find's (advanced) scope and conceptualize its implementation, three questions have to be taken into account:

- What do scientific users expect a basic data query feature to look like?
- Considering the underlying RDF storage, is there a best practice approach for developing these feature improvements?

²⁴Essentially, Atom is a open-source desktop text editor which comes with some basic core functionality [22]. Its main core however is easily extendable through numerous individual packages [22, 19], such as Hydrogen (for interactively running (Python, R or JavaScript) code [20]), the Github/ Gitlab integrations (for working with Github/ Gitlab repositories (e.g., by staging, committing, pulling and pushing changes) in the Atom environment [18]) or the Python debugger (for debugging Python projects [21]).

- Given that a RDF data querying recommendation does exist, how can this be applied to the existing Owlready2 data store?

Basic data querying functionality Essentially, the first of the three research questions has already been answered during the *system analysis* phase: to identify potential functionality gaps during the tests, fundamental user expectations had to be disclosed beforehand (paragraph *Functionality expectations*).

Thus, table 3.1 lists typical arithmetic comparison operators, which the users²⁵ expect and which therefore have to be added. Moreover, users assume that all the parameters, which they specify during a concept's or instance's creation, are also available as search criteria. Respectively, the range of lookup criteria has to be supplemented by name, description, state, timestamp, mutability, translation, `py_class` and methods.

RDF best practice Upon recalling that Owlready2 stores data as “RDF quads” [72], the question arises whether the World Wide Web Consortium (W3C) also recommends a standard for querying such RDF data²⁶: and indeed six years after Resource Description Framework (RDF)'s release in 1998, the ‘RDF Data Access Working Group’ presents SPARQL as the first RDF query language draft, which – again four years later, in 2008 – becomes the official recommendation [91, 90, 89].

At first glance, SPARQL might remind oneself of SQL due to its structural and syntactical similarities [67]. However, the most important distinction is that SPARQL extracts graph²⁷ data while SQL returns relational (tabular) data [67]. Since SPARQL can be adopted by any RDF data storage system, (code) reusability is ensured even if the underlying database storage system is changed at a later point in time.

Owlready2 SPARQL implementation options The Owlready2 documentation [72, 74] contains detailed information on SPARQL queries and how they can be implemented in and run with Owlready2. Basically, there are two options [74]:

- The native SPARQL engine and
- RDFlib.

²⁵Who are for instance familiar with SQL queries (and particularly its WHERE condition) [94] or various programming languages such as Python, their diverse (comparison) operations and use cases (e.g., as part of IF-statements) [42]

²⁶Since RDF is also one of W3C's standards for information exchange [49].

²⁷A RDF graph consists of RDF triples and therefore SPARQL queries such triples, which are made up of a subject, predicate and object [90].

The instructions state, that – since the *native SPARQL engine* has 1) hardly any dependencies, 2) a relatively smooth and thus fast start-up 3) and directly converts SPARQL queries into SQL ones before executing those SQL queries with SQLite3 – the *native SPARQL engine* is around 60 times quicker than its alternative, *RDFlib* [74]. However, the *native SPARQL engine* also has one major drawback: at the moment, it only covers a fraction²⁸ of SPARQL’s whole functionality range [74]. Depending on one’s own usage scope, this might or might not be an elimination criteria and definitely has to be kept in mind during the planning and conceptualisation phase.

RDFlib on the contrary, offers SPARQL’s whole functionality range and therefore loads Owlready’s quadstore as a *RDFlib* graph [74].

From the viewpoint of the anticipated find selection scope extension, inequality and range operators to be precise²⁹, the following SPARQL components are needed:

- PREFIX
- SELECT
- WHERE and
- FILTER.

Since all of those elements are available under the *native SPARQL engine* and because this project cares about a performant and efficient implementation, the data querying enhancements will be set-up with the *native SPARQL engine*.

3.3.3.3 Collection conceptualisation

The collection concept can be seen as a complement to DKB’s present functionality, to further mirror the (scientific) user community’s behaviour and needs and thereby valorise DKB’s value for them. As previously stated, people have the natural tendency to collect things: from the classic example of stamps to coins, model cars/ planes/ trains or liquids such as whisky. The scientific user community is thereby no different: as listed by Atkinson et al. (2019) [5], they ‘gather’ anything from (input) variables, to simulation results and their true (observed) outcomes and new occurrences. But then the questions arise,

- How can a collection be defined?
- What are its components and features?

²⁸For example, data inserts or deletions (with and without WHERE limitation) as well as CONSTRUCT, COPY or DROP queries are not available [74].

²⁹Besides of the extra selection criteria, which have no impact on the required SPARQL elements.

These will be answered by the successive collection conceptualisation, which takes a practical angle³⁰ whilst incorporating the theoretical (academic) perspective.

Collection definition Generally speaking³¹, a collection is an object (‘container’), which groups (multiple) individual elements into one unit [110, 16, 62, 88].

There are three ways of viewing collections, which have to be respected during the conceptualisation:

- *Local or user view*: Considers how users think of collections (usually as sets, bags, sequences and dictionaries, depending on the importance of a (sort) order and/ or the allowance of duplicates).
- *Knowledge base or organizer perspective*: Takes the backend (‘organizer’), which is in control of the data and their storage, into account and how collections should best be established from this point of view.
- *Global viewpoint*: Refers to remote, (possibly) distributed collections, which account for the majority of existing collections [112], and how to incorporate them into DKB.

Collection components In line with Van Lepthien’s and Anderson’s (2005) [110] suggested collection items³², the collection’s three major components are the:

1. *Container or collection wrapper*
2. *Content* in terms of the individual elements
3. *Methods / functions* for accessing and working with collections and their content

1) Collection wrapper: the overarching ‘wrapper’, framework for the comprised elements, should have a couple of tunable specification parameters:

- **Collection origin**: First of all, it is crucial to decide whether the collection will consist of local DKB data or whether it is based on a remote (distributed) collection. This is necessary to account for the global view with its remote and distributed collections: incorporating³³ such remote archives gives the users more flexibility and decision-making options, fosters reusability and potentially saves

³⁰By targeting the scientific user community and what is most useful to them.

³¹And in accordance with mathematics and computer science [16]

³²“Content”, “type(s) of elements” and “mechanism(s) for accessing the content” [110].

³³However, including extensive heterogeneous data (with varying semantics and underlying implementations) imposes a significant challenge, which a couple of researchers have focused on: Endris et al. (2019) [36] for instance propose “Ontario”, an innovative, SPARQL based processor for selecting information from Semantic Data Lakes. Verborgh et al. (2016) [112] frame the “Linked Data Fragments” to exploit “Linked Data” (instead of centralized SPARQL endpoints) and move as much of the processing as possible away from the servers and to the clients.

disk space if the remote collections remain referenced.

- Initial build: Based on the collection (data)'s provenance, the users can decide how to build their collection: either *manually*, by giving (a list of) DKB Persistent Identifier (PID)/ remote identification numbers, or *dynamically*, through DKB's find³⁴ or the reference to a distributed collection. This again offers users further flexibility and freedom of choice since they can both set up a new collection from scratch by hand or by utilizing existing (DKB) functionalities (the later is less time-intensive, whereas the first allows for (extremely) individualized collections), depending on their preferences and requirements.
- (Containment)³⁵ Type: Thereafter it is crucial to determine whether the collection is *static* and *materialized* (thus, the content of the specific collection is physically stored) or *virtual/ referenced* (hence the individual elements are not kept)³⁶. *Virtual/ referenced* further implies, that the collection can change each time the according expression or reference is called (since the underlying DKB database, on which the find is executed, or the remote collection can evolve). Especially under the aspects of ever-growing amounts of data, their multiple (physical) storage and the thereof resulting retention challenges, both the avoidance of (duplicate) physical storage and its replacement with virtual representations become more and more important. In case a local copy of a remote collection is saved, the questions, how updates to the referenced collection are handled within DKB (update the local duplicate likewise?) and how a notification mechanism (e.g., triggering cache refreshments via protocols) looks like, have to be asked.
- Element type³⁷: Facultative it is possible to regulate the kind of individual elements, which can be added to the collection (in DKB's case either contexts, concepts, instances or collections). This is particularly helpful in situations, where scientists for example want to compare the results of multiple simulation runs with the actual earthquake observation and thus only wish to collect particular

³⁴E. James Whitehead (2002) [119] even states that "queries are by far the most common functions used to dynamically create containers".

³⁵According to Whitehead (2002) [119], who introduces the "containment type" as a way of representing how elements are enclosed. Therefore Whitehead (2002) [119] differentiates between two types of containment: "inclusion" and "referential".

³⁶Besides Whitehead (2002) [119], also Johnston and Robinson (2002) [62] and Psaila (2011) [92] distinguish between "physical" and "digital items" [62] and talk about "archival fonds" and "artificial collection" [62] as well as "named (if materialized in the database)" and "unnamed (if temporary)" elements [92].

³⁷As described by Whitehead (2002) [119], "containers may limit, or explicitly state the type of objects that can be contained".

DKB instances³⁸. To avoid the unintentional inclusion of a concept or context, users can restrict the collection's element type.

- Size³⁹: Optionally, users can decide whether the total collection size is *limited* to a certain number of elements (upper boundary) or whether the size is *unrestricted* (default setting). This is especially useful for cases, where scientists want to publish a collection (containing simulation or observation results e.g.) as part of a conference or journal paper but have to adhere to their imposed size limitation.
- Specialisation (Membership and Ordering): The final optional (specialisation) setting follows Whitehead's (2001/ 2002) [118, 119] notion of "membership" and "ordering": users are able to define a(n)
 - Order (by key [64], value or insertion [64]; by default not ordered)
 - Duplicate removal (by entering a discriminating function as per which the duplicate decision is made: e.g., duplicates are identified based on the 'name' or on 'name' and 'year'; default setting allows duplicates).

Accordingly, the collection then becomes a⁴⁰

- Set (unordered, without duplicates [64]), realisable with Python's built-in set.
- Bag or multiset (unordered, with duplicates [16]), implementable as a Python collection.
- Map (ordered, without duplicates), presentable as a Python dictionary.
- Sequence (ordered, with duplicates [64, 16]), achievable as Python array⁴¹.

2) Collection content: following Atkinson et al.'s (2019) [5] statement that a "collections may be of anything: concepts, methods, data or collections", a collection in the DKB environment can consist of concepts, instances, contexts and collections⁴².

3) Collection functions: The first and most essential operation is to create a new collection, which distinguishes between

- Dynamic ways of generating (none-empty) collections (by evaluating/ storing the literal find expression or the remote collection reference) and
- Manual options to set up a new collection, either by
 - Explicitly naming the individual collection elements, or by

³⁸Instances represent actual workflows and (simulation) runs with according execution parameters.

³⁹What Whitehead (2002) [119] calls the "number of contained objects".

⁴⁰Similarly to what Object Database Management Group (ODMG) has specified as the four collection types: "sets, lists, arrays, and bag (multiset)" [88]

⁴¹According to Keedy and Rosenberg [64] the "default implementation for key order sequences".

⁴²Amongst others, Psaila (2011) [92] noted that one can deduces an object from itself (e.g., "containers from containers").

- Copying an existing collection.

All remaining methods for accessing and working with collections and their content can be divided into element- and collection-oriented operations:

- Element-oriented (in line with those established by Van Lepthien and Anderson (2005) [110], Psaila (2011) [92], Whitehead (2002) [119] as well as Keedy and Rosenberg (1989) [64])
 - Insertion of individual elements: It is quite common that users want to add further content at a later point in time, similarly to documents which users keep expanding. For ordered collections users might even want to fill in an element at a specific position [33].
 - Update specific elements: it is not unusual that content has to be altered later on, which again is comparable to documents that users keep changing.
 - Replacement of particular elements: from time to time certain components have to be exchanged, for instance with a newer version; just as with living documents, where various passages are continuously rewritten.
 - Deletion of outdated elements: every now and then content becomes obsolete (e.g., when a new finding or observation is made) or does no longer belong to the previously assigned group (e.g., a tsunami is allocated to an earthquake but further investigations reveal that another earthquake has caused it) and therefore has to be removed from it.
 - Listing all collection elements ('snapshot'): often it is necessary to get an overview of all the currently (thus 'snapshot') contained elements, for instance to report them or to review their correct assignment.
 - Extraction of a subset: Sometimes only a dedicated subgroup (e.g., all earthquakes in Greece for a particular year) is required, which can then be derived by DKB's find function. To perform look-ups within a specified collection though, the 'IN' operator is needed. This however is currently neither implemented nor in scope and thus has to be added to find().
 - Renaming of elements: as with any document, a later (refining⁴³) name change is not uncommon and should therefore be possible.
 - Viewing predecessor and successor record [33]: Especially for sorted collections it can be beneficial to inspect only the previous and subsequent entry. If, for example, users generate a (simulation run) collection ordered by execution timestamp and would like to compare successive runs, each

⁴³Especially once further progress is made or information is available

of which has been refined, the option to examine the predecessor and successor run is extremely helpful.

- Collection-oriented
 - Application of a (user-defined) function: Most of the scientific users, whom DKB is targeted at, are fluent in Python and thus comfortable with writing own (small) programs and scripts. Therefore they are accustomed to applying one function to a whole set of elements (e.g., if they would like to repeat all their previous (earthquake) simulation runs with a slightly increased moment tensor). Hence such a feature (of applying more or less complex user-defined functions like addition) is required for collections.
 - Deletion of a complete collection: similarly to documents, users sometimes wish to remove an entire collection, for instance (if it is no longer in use) to clean up their work space or to save storage space.
 - Ways of combining collections⁴⁴:
 - * Union: Combines all elements from any of the selected collections into one unified collection. For example useful for scenarios where users want to create an overarching collection with all Italian earthquakes, but only have region-specific (per Italian state) earthquake collections.
 - * Set difference: Returns all elements of the first listed collection, which do not appear in the other named collection(s). A possible application of such an operator could be the request to report all earthquakes, which can be unambiguously assigned to Greece (and did not occur close to its Bulgarian border for instance).
 - * Intersection/ Inner join: Shows those elements, which are contained in each of the mentioned collections. If we take the previous example again, the use case is now that all these ambiguous/ close to the border earthquakes between Greece and Bulgaria should be presented (which the scientists have included in both countries' earthquake collections).
 - * Joins (left, right and cross-product [14]): Specifically for comparisons between simulation runs and real-world observations (e.g., earthquakes) the various join options are of help. If the simulation results are either listed left or right and the according observations (right or left) should be mapped by date of occurrence and/ or name for instance, a left or right join can be utilized. If, on the contrary, simula-

⁴⁴In accordance with Whitehead (2002) [119] and Keedy and Rosenberg (1989) [64]

tions and their observations cannot be mapped by a unique column, a cross (/ cartesian) product, which matches all elements with each other and is therefore computationally expensive, might be required.

3.3.3.4 Performance improvement

As elaborated in paragraph *Performance improvement*, the previous performance findings have to be treated with caution and have to be put into perspective. To do so, questions such as

- How is Owlready2's performance compared to other (NoSQL and relational, to name the most popular) databases?
- How does Owlready2 aim to ensure an adequate performance?
- How can Owlready2's overall performance concept be improved, specifically for DKB?

have to be clarified.

Owlready2 benchmark results In his personal blog, “The flowers of evidence” [71], the Owlready2 architect Jean-Baptiste Lamy reports his databases comparison results for handling 10,000 nodes⁴⁵ per object via random access [71]. Table 3.2 displays an excerpt of the performed contrasting, which confirms that Owlready2 outperforms the other investigated options.

Base technology	Writing speed	Reading speed
Python + MongoDB (4.0.4)	2,289 obj/sec	4,723 obj/sec
Python + Neo4J (3.4.9)	245 obj/sec	223 obj/sec
Python + Owlready2 (0.16)	12,892 obj/sec	19,158 obj/sec

Table 3.2: Extract of benchmark comparison carried out by Jean-Baptiste Lamy [70]

Having said this, it is equally important to note that the Owlready2 performance results have not (yet) been confirmed by other researchers. Likewise, Jean-Baptiste Lamy has only presented findings for *random* access and not (as of now) for *sequential*⁴⁶ data access.

Thus overall, these observations are taken with a little scepticism and the remark, that the outcome should be verified (for example, with differing amounts of data and

⁴⁵As one of the three major graph components, nodes (or vertices) represent the graph's entities [116].

⁴⁶Sequential access is the preferred method for reading and writing data to disk since it shortens the time spent on disk Input/Output (I/O) operations (through decreasing seek time and rotational delay, two essential components of the total access time) [94].

access patterns). Since such an in-depth comparison with other database matches the workload of a separate, independent project, it will not be covered as part of this project. Instead, this project will go along with Jean-Baptiste Lamy results and leave their proof for future work.

Integrated Owlready2 optimizations For an enhanced performance and memory consumption [72], Owlready2 utilizes four (among them one optional) factors:

- **Cython module (optional):** Owlready2 comes with an optimized *Cython module*, which (if installed) enables an ontology loading speed-up of around 20% [72].
- **Dynamic data loads & caching:** Owlready2 offers *dynamic loads* from the underlying quadstore as well as a (temporary) in-memory *caching* possibility [72].
- **Integrated & persistent SQLite3 data store:** Owlready2's optimized quadstore is by default retained in memory, but can also be made persistent in a *SQLite3* file [72]. The later benefits the handling of and dealing with huge ontologies [72].
- **Indexing:** a (debugging) investigation has revealed, that Owlready2 exploits indexes on three of its in total eight internal database tables⁴⁷ and that Python file 'triplelite.py' is responsible for their creation and deletion.

Owlready2 improvement suggestions With the above knowledge, a search for potential enhancement opportunities has been conducted. First of all, Owlready2's indexing is rather generic and leaves room for (application-specific) customising: for the DARE Knowledge Base, further indexes on

- **PID** (a unique (context, concept and instance) identifier to accelerate their search),
- **Prefix** (the name of the context, the concepts and instances belong to, to speed up the selection of objects within a certain context) and
- **Instance name and/ or timestamp** (to facilitate future versioning (e.g., the look-up of all (successive) instance versions)

seem suitable and useful.

Moreover, Owlready2's existing caching approach could be adjusted: quite recently, dedicated and promising graph database caching approaches have emerged

⁴⁷Namely, on 'resource', 'objs' and 'datas'; 'resources' maps a (numbered) unique, internal storage identifier (ID) to the objects⁴⁸ ontology Uniform Resource Locator (URL), 'objs' defines the relations between the storage objects⁴⁸ and 'datas' contains the user specified values for each of the object's⁴⁸ parameters.

⁴⁸ An object may be anything from ontology class and instance annotations to DKB's user-defined contexts, concepts and instances.

(for example GraphCache⁴⁹ [113] or Bok et al.’s (2020) advanced “two-level caching scheme”⁵⁰ [12]), which can be further analyzed with regard to their adaptability.

Finally – and as suggested in *Owlready2 benchmark results* – Owlready2 as the underlying database could be questioned. Basically, there are two choices:

- Either replace Owlready2 and its integrated SQLite3 database with a different (graph) database (optimally one amongst the market leaders),
- Or alter Owlready2’s logic according to another (leading) database’s concept.

Nowadays, Neo4j is among the most popular graph databases [101, 68, 59, 95, 106], with Neo4j Inc. even stating that “Neo4j is the leader in graph database technology” [60]. And both Forrester [121] and Gartner [2] support this claim: their most recent research reports (2020) have ranked Neo4j [46] among the most relevant graph databases⁵¹. Thus, Neo4j might be one of the databases worth considering when performing a DB replacement investigation. Alternatively, the adaption of one of Neo4j superior concept’s (such as the two-tier index architecture⁵²) could be inspected [46].

Due to the limited time available for this project and the fact that all of the above easily accounts for another project proposal, the decision has been made to neither cover the proof of the Owlready2 performance results nor the examination of any enhancements but instead go along with Jean-Baptiste Lamy findings and leave the above highlighted points as future work opportunities.

3.4 Software development

Although software development’s goal is to generate high-quality (software) solutions within a given time and budget [45], it is often (justifiably) blamed for project delays [45, 34]. A possible explanation therefore is, that developers’ estimation regarding the implementation duration is rather often a delusive and hardly achievable “best-case scenario” [34]. To improve the software development’s planning, a wide variety of software development approaches exists nowadays: from waterfall (the first widely

⁴⁹A “full-fledged caching system for general subgraph/ supergraph query processing” [113].

⁵⁰The authors differentiate between Used Cache (UC) – recently used and probably queried again soon – and Prefetched Cache (PC) – unused, however most likely required by the next run – and work with various replacement policies such as First-In, First-Out (FIFO), if the cache is at full capacity [12].

⁵¹“Forrester’s research uncovered a market in which Neo4j, Amazon Web Services, TigerGraph, Microsoft, and Oracle are Leaders” [121], with Neo4j attaining the best scalability and performance scores [60].

⁵²Neo4j’s two-tier index architecture contains both an object (attributes with their according descriptions) and a triple (composed of subject, object and predicate, indicating their relationship) index [46].

accepted methodology [30]), to prototyping, iterative, rapid and agile methods [30], with the latter currently being one of the most frequently used techniques [34].

3.4.1 Utilized methodology

As elaborated by Ghule (2014) [45], one of software development's major problems is that a (upfront) proper risk analysis is often neglected, through which delays can quickly sneak in. Amid the very strict time limit for this project, a particular focus has been put on the method's consideration of an early risk assessment, while reviewing and deciding upon a software development approach. Despa (2014) [30] provides a detailed overview of 20 commonly used methodologies, their main characteristics and advantages and downsides. Based on Despa's (2014) [30] review, techniques which are primarily applicable for large-scale projects (e.g., prototyping) have been excluded as well as those which only receive (user) feedback at the very end (e.g., waterfall).

Eventually, the *spiral methodology*, which consists of four phases – namely, *planning*, *risk analysis*, *development* and *evaluation*, as illustrated in figure 3.2 [30] – has been chosen.

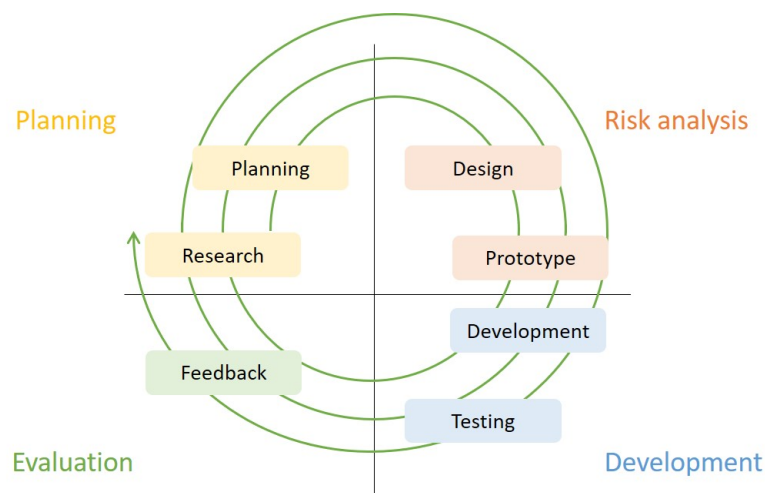


Figure 3.2: Spiral methodology, according to Despa (2020) [30]

To guarantee flexibility and a steady progress, all of the four phases are repeatedly passed [30]. As indicated above, the main arguments in favor of this approach are the continuous reiteration of all four phases (including an early and steady feedback collection) as well as its risk analysis focus, which result in [30]:

- Exploration and evaluation of diverse options before a decision, upon which to select, is made;
- Periodic (development) checks to ensure user satisfaction and high quality.

3.4.2 Coding style

During the software development special attention will be given to the creation of an efficient (both in terms of lines of code and runtime performance) and reusable program code. Therefore, the Python coding guidelines, which have been framed by Van Rossum et al. (2013) [111], will be followed and common compliance errors, such as those reported by Bafatakis (2019) [9], considered.

3.4.3 Accomplished developments

Now that the general development procedure and the coding rules have been touched on, the forthcoming section will focus on the achieved implementations.

3.4.3.1 Application bugs

Subsection *Application bugs (ABs)* covers all the observed application errors, while subsection *Application bugs* justifies why it has been important to resolve them. As aimed for, their⁵³ resolution has been accomplished as part of this two and a half month project (appendix *Application bug fixes* contains the coding snippets).

3.4.3.2 Functionality deficiencies

As discussed in subsection *Functionality deficiencies* and summarized in section *Formulated research and development plan*, the find functionality enhancement is in scope of the implementation.

Find functionality: The first step to add further (comparison) operators has been to expand the validation of the (user-supplied) query operator in the ‘DareKB.py’ file. To account for potential (syntactically) erroneous user entries, not only ‘<=’ and ‘>=’ but also ‘<’ and ‘>’ have been included and are later mapped to their appropriate counterpart. Now, it is possible to add the inequality and range operator to the internal operator checks, which are responsible for calling the according ‘storage.py’ functions (`find_all_tuples(self, o_type, *args)` and `find_tuples_inRange(self, user_query, o_type)`) for retrieving and returning the queried Persistent Identifiers (PIDs).

To ensure reusability, maintainability⁵⁴ and efficiency, for instance

⁵³To be precise, the ‘instance creation’, ‘concept and instance lookup (`find()`)’ as well as the ‘concept and instance data access (`find()` and `get()`) after a DKB server restart’ issue.

⁵⁴By writing as few code lines as possible and providing sufficient and sound code comments.

- Hardcoding has been avoided where possible (e.g., function `find_all_tuples` can be reused with any sort of look-up parameters `*args`),
- Dynamic compositions fostered (e.g., the SPARQL queries – function `find_tuples_inRange` – are dynamically created) and
- The most efficient ways of implementing considered (e.g., list comprehension instead of for-loops).

3.4.3.3 User Manual updates

Equally, the in subsection *User Manual updates (UMUs)* elaborated and in subsection *User Manual updates* reasoned adjustments to the [User Manual](#) [75] have been completed (appendix *User Manual updates* incorporates the performed modifications).

3.5 Achievements and their assessment

After the developments had been finished, an examination – like the one performed as part of the system analysis – has been carried out to investigate and contrast the improved system with its previous version. Therefore, once more *functionality* and *performance* test (to determine functionality as well as response time) have been executed (respective Jupyter notebook located on [Gitlab](#) [55]). Additionally, a questionnaire⁵⁵, for which ethical approval by the University of Edinburgh has been received – RT#5959 – and which, together with its results, the information sheet and consent form, is available in appendix D, has been set up and distributed among former DARE/DKB test users⁵⁶ as well as the Data-Intensive Research Group (DIRG)⁵⁷ members to gather their opinion and feedback.

An extensive discussion of the achievements, their (test and rating) results as well as open points will be conducted in the following and final chapter, *Closing discussion and Future Work*.

⁵⁵This survey, a quantitative method for collecting data [81], asks the participants to rate the different improvements on a scale from one to five (1 = Not helpful at all; 2 = Not helpful; 3 = Not sure; 4 = Helpful; 5 = Extremely helpful).

⁵⁶Those users have already collaborated on the current DARE Knowledge Base prototype and committed to support DKB further.

⁵⁷The DIRG is a study group at the University of Edinburgh, which conducts research on the optimization of data-intensive tasks.

Chapter 4

Closing discussion and Future Work

Achievement summary All in all, this project has achieved significant progress in multiple areas: First of all, a functional and bug-free DKB version has been produced (and already shared with the DIRG group) by resolving the detected application errors. Furthermore, DKB's existing (data querying) functionality has been enhanced by providing a variety of additional search parameters and comparison operators, to broaden DKB's feature range and meet users' expectations. Such an appropriate data retrieval (find) function is also necessary to properly interact with the newly established conceptual model of collections, a supplement to DKB's present functionality: collections mirror people's natural behaviour of gathering and clustering things into self-defined groups and therefore foster DKB's value appreciation. Finally, a detailed investigation of the current database's (performance) acceleration measures, followed by an extensive research of additional improvement options, has been composed.

Smaller accomplishments are the (synthetic) data generation program, the reusable (test) Jupyter notebooks as well as the User Manual updates and the detailed test findings, which offer future work opportunities.

Attainment evaluation To judge the benefits gained from the above described achievements, two types of assessment have been carried out: 1) (application) tests to validate the developments' functionality and investigate their performance and 2) a quantitative user survey to measure the implementations' and conceptualizations' value with the help of a user rating. Since any enhancements should eventually benefit the (scientific) user community, only former DARE/ DKB participants, who have collaborated on the current DKB prototype and its functionality scope, and DIRG members, who continue to work on DKB, have been asked to fill in the questionnaire (Appendix D.1).

Given the fact that this is a rather limited and elected circle, the ten received responses to the completely anonymous questionnaire represent a good amount of feedback. Ten replies are also an adequate indication of what the scientific community thinks about those adjustments and ideas. Overall the reaction to the three types of changes – ‘FDs’, ‘ABs’, and ‘UMUs’ – has been very favourable (refer to Appendix D.2 for the detailed results): Besides a median rating between 4 (‘helpful’) and 5 (‘extremely helpful’) –

- Functionality deficiencies: average of 4.7
- Application bugs: mean of 4.6
- User Manual updates: result of 4.4

–, the obtained comments are also extremely appreciative and positive (figure D.8), stating that the “adjustments [...] were really needed”, are “wonderful and helpful” and “increased the usability of the DKB”. They have further been declared a “significant progress” and a necessity for “a knowledge base like DKB to be operational”. As elaborated before, the goal of this project has been primarily to be a research one with some development percentage. Therefore, two main focus topics (the most acute and research-intensive), in fact the two listed functionality deficiencies, have been chosen:

- Improvement of the `find()` function (more search criteria and operators)
- Formulation of a conceptual model of *collections*

Particularly encouraging therefore is the circumstance that these two FDs have achieved the best overall result (average of 4.7, with *collections* reaching 4.5 and `find()` even a 4.9). The discrepancy between the *collection* and `find()` valuation however emphasises the challenge associated with conveying the theoretical concept of collection (even more so as part of a short questionnaire). Still, the *collections*’ conceptualization is seen as the thesis’ key outcome: as depicted before, the rising volumes of manifold, complex and interlinked data from all kinds of dispersed sources require fresh management and storage approaches, such as the completely novel concept of user-controlled context. In 2019 Atkinson et al. [5] introduced user-controlled contexts as an innovative way of structuring the information space and thereby facilitating knowledge self management. To reflect scientific users’ way of thinking, contexts incorporate concepts, for defining the general object structure (like a function), and instances, for populating the concept frame with actual values (similarly to the function call). This new approach, which has neither been examined nor put into practice so far, however currently lacks another important, natural habit: scientists’ tendency to collect things - from (input) variables, to simulation results and their true (observed) outcomes as well as new occurrences [5]. Thus, this is where the collection concept comes into play.

Another essential aspect, which the collection conceptualization covers and which is again related to the large amounts of distributed data, are the data storage options: virtual/ referenced and static/ materialized. With growing data volumes, storage and data maintenance become more challenging and it is preferable to not physically duplicate data from the golden source. Instead, this information can be referred to and loaded only upon request, which saves not only storage space but also maintenance (in terms of implementing update mechanism) effort. A possible solution for integrating such remote data (collections) could be Endris et al.'s (2019) [36] “Ontario”, an innovative and centralized SPARQL endpoint for gathering and including information.

Unlike the questionnaire, which has also included the conceptual achievements, the *functionality* and *performance* tests (captured in a Jupyter notebook on [Gitlab](#) [55]) solely focused on the implementations. The test results prove, that both the previously detected application bugs and the `find()` selection parameter and comparison operator enhancements work and return the expected results (e.g., the string range selection follows the ASCII sort order, as demonstrated and explained in section C.2.2).

Limitations and future work Nevertheless, the *performance* tests have also revealed some severe response issues for the ‘state’ and ‘mutability’ search parameters: after around one and a half hours, both query executions have been interrupted¹. The reason for their excessive runtime on the prepared test database is, that basically every concept and instance (and therefore almost every entry in the 390 MB database file) contains ‘state’ = ‘new’ and ‘mutability’ = ‘mutable’. Thus, accessing each of them to retrieve a subset of their available parameters has exceeded the acceptable wait time by far. Another observed (and related) limitation is specifically linked to Jupyter notebooks: only a certain number of records can be outputted, before an “*IOPub data rate exceeded; The notebook server will temporarily stop sending output to the client in order to avoid crashing it*” error is given. For both of the mentioned points a (user-defined or default, for the latter with an according informative message) *limit* on the number of returned records seems reasonable. Moreover, a *count* feature can help avoid too generic user queries by identifying the number of data sets, which match the data selection query, in advance. Other limitations – and thus future work possibilities – are missing implementations such as the collections, default values and the absent data types. Both² the unavailable default values and the limited data types

¹Repeating their execution on a smaller database then returned the correct results.

²Similarly to the systematical approach followed for the `find()` enhancement, these two conceptual shortcomings can also be dealt with by building upon the existing system and utilizing its options.

(solely ‘string’ and ‘integer’), which are described in paragraph *FD results*, have been deferred since a workaround exists and they are not decisive for DKB’s operability. The collections though are: As outlined in the preceding paragraph, collections can valorise DKB’s value since they correspond to scientists’ way of thinking and behaving. Moreover, remote (distributed) collections and their integration present a great opportunity and should therefore be realized as well. Finally, paragraphs *Integrated Owlready2 optimizations* and *Owlready2 improvement suggestions* depict Owlready2’s built-in optimizations and possible improvement suggestions, which can also be considered for future work on the DKB.

Conclusion To sum up, the thesis’ overarching goal has been to enhance the existing DKB application by determining and resolving multiple defined issues, which ultimately benefit the (scientific) user community. To do so, the main aim has been split into three objectives:

1. Identification of current weaknesses: with the help of a thorough test and analysis phase, which has involved the generation of a sufficiently large (390 MB) amount of synthetic and real-world (test) data as well as extensive functionality and performance tests and the evaluation of their results, three major problem categories have been identified: ‘application bugs’, ‘functionality deficiencies’ and ‘User Manual updates’.
2. Topic selection and resolution: Since the amount of work associated with all revealed ABs and FDs has exceeded the given time frame, a (justified) subselection has been made. Particular focus has thereby been put on the most pressing, research-intensive and achievable topics, namely: a) `find()` functionality enhancement and b) *collections* conceptualization. Before those could be approached though, all detected application errors, which made working with the DKB prototype almost impossible, had to and have been resolved. Thereafter, the two most important achievements have been accomplished: An efficient implementation of the additional `find()` search parameters and comparison operators, as well as the conceptualization of the collections (further accomplishments have been summarized above, *Achievement summary*).
3. Achievement evaluation: Finally, all attainments have been critically analysed through both application tests and a quantitative user survey (questionnaire). The details regarding the investigation and observed limits are listed above (*Attainment evaluation* and *Limitations and future work*).

Bibliography

- [1] Jennifer K Adelman-McCarthy, Marcel A Agüeros, Sahar S Allam, Kurt SJ Anderson, Scott F Anderson, James Annis, Neta A Bahcall, Coryn AL Bailer-Jones, Ivan K Baldry, JC Barentine, et al. The fifth data release of the Sloan Digital Sky Survey. *The Astrophysical Journal Supplement Series*, 172(2):634, 2007.
- [2] Merv Adrian, Afraz Jaffri, and Donald Feinberg. Market guide for graph database management solutions. *Gartner Research Inc*, 2021.
- [3] Anastasia Ailamaki, Verena Kantere, and Debabrata Dash. Managing scientific data. *Communications of the ACM*, 53(6):68–78, 2010.
- [4] Peter Amstutz, Michael R Crusoe, Nebojša Tijanić, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, et al. Common workflow language, v1. 0. 2016.
- [5] Malcolm Atkinson, Rosa Filgueira, Iraklis Klampanos, Antonis Koukourikos, Amrey Krause, Federica Magnoni, Christian Pagé, Andreas Rietbrock, and Alessandro Spinuso. Comprehensible control for researchers and developers facing data challenges. In *2019 15th International Conference on eScience (eScience)*, pages 311–320. IEEE, 2019.
- [6] Malcolm Atkinson and Patricia Hartmann. Status of DKB implementation [Unpublished]. <https://docs.google.com/document/d/10zE--xRvaSThLAsFbyvXCMnA3tHjriWC2eSnc7DN6wk/edit#heading=h.jju9how9gxxl>, 2021. Last accessed: 2021-07-25.
- [7] Malcolm Atkinson, Iraklis Klampanos, Valentina Andries, Aurora Constantin, Rosa Filgueira, André Genünd, Ellen Gottschämmer, Vangelis Karkaletsis, Antonis Koukourikos, Amélie Levray, Mike Linder, Federica Magnoni, Christian

- Pagé, Andreas Rietbrock, Alessandro Spinuso, Chrysoula Themeli, Xenofon Tsilimparis, and Wolf Fabian. Dare Architecture and Technology D2.2, December 2020. *This deliverable was approved internally and by external reviewers and recognised as reporting significant advances in distributed support for computationally and data intensive research and development of particular relevance for research developers and for those who develop and support their web-enabled work environments.*; Last accessed: 2021-08-10.
- [8] Malcolm Atkinson, Amélie Levray, and Rui Zhao. DKB Design. <https://docs.google.com/document/d/1hCy0qeB8R00v5ZcBZVxyCOAiOST7QEP90n37PngHxGs/edit#heading=h.4gp1bkwy09pn>, 2020. Last accessed: 2021-07-28.
- [9] Nikolaos Bafatakis, Niels Boecker, Wenjie Boon, Martin Cabello Salazar, Jens Krinke, Gazi Oznacar, and Robert White. Python coding style compliance on stack overflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 210–214. IEEE, 2019.
- [10] Jagdev Bhogal and Imran Choksi. Handling big data using NoSQL. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pages 393–398. IEEE, 2015.
- [11] Desamparados Blazquez and Josep Domenech. Big Data sources and methods for social and economic analyses. *Technological Forecasting and Social Change*, 130:99–113, 2018.
- [12] Kyoungsoo Bok, Seunghun Yoo, Dojin Choi, Jongtae Lim, and Jaesoo Yoo. In-memory caching for enhancing subgraph accessibility. *Applied Sciences*, 10(16):5507, 2020.
- [13] Stuart K Card, George G Robertson, and Jock D Mackinlay. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pages 181–186, 1991.
- [14] Antonio Celesti, Maria Fazio, and Massimo Villari. A study on join operations in mongodb preserving collections data models for future internet applications. *Future Internet*, 11(4):83, 2019.

- [15] Alex Chris. Top 10 Search Engines In The World (2021 Update). <https://www.reliablesoft.net/top-10-search-engines-in-the-world/>, 2021. Last accessed: 2021-04-14.
- [16] Paolo Ciccarese and Silvio Peroni. The collections ontology: creating and handling collections in owl 2 dl frameworks. *Semantic Web*, 5(6):515–529, 2014.
- [17] European Commission. Delivering Agile Research Excellence on European e-Infrastructures. <https://cordis.europa.eu/project/id/777413>, 2020. Last accessed: 2021-08-09.
- [18] Atom community. Atom gitlab-integration package. <https://atom.io/packages/gitlab-integration>, 2021. Last accessed: 2021-07-28.
- [19] Atom community. Atom Packages. <https://flight-manual.atom.io/using-atom/sections/atom-packages/>, 2021. Last accessed: 2021-07-28.
- [20] Atom community. Hydrogen. <https://atom.io/packages/hydrogen>, 2021. Last accessed: 2021-07-28.
- [21] Atom community. Python-Debugger package. <https://atom.io/packages/python-debugger>, 2021. Last accessed: 2021-07-28.
- [22] Atom community. Why Atom? <https://flight-manual.atom.io/getting-started/sections/why-atom/>, 2021. Last accessed: 2021-07-28.
- [23] Jaudete Daltio. Graph database with the entire water drainage network of the Brazilian hidrography. 2019.
- [24] DARE Consortium. DARE Platform tutorial. <https://gitlab.com/project-dare/dare-examples/-/blob/master/tutorial/DARE%20platform%20tutorial.ipynb>, 2020. Last accessed: 2021-07-25.
- [25] DARE Consortium. DARE Platform tutorial - WP6 MT3D test case. https://gitlab.com/project-dare/dare-examples/-/blob/master/tutorial/WP6_MT3D_tutorial.ipynb, 2020. Last accessed: 2021-07-22.
- [26] DARE Consortium. EPOS — MT3D – Moment Tensor In 3D. <http://project-dare.eu/epos/mt3d-moment-tensor-in-3d/>, 2020. Last accessed: 2021-07-22.

- [27] DARE Consortium. EPOS — RA – Rapid Ground Motion Assessment. <http://project-dare.eu/epos/ra-rapid-ground-motion-assessment/>, 2020. Last accessed: 2021-07-22.
- [28] DARE Consortium. The EPOS Use Case. <http://project-dare.eu/epos/>, 2020. Last accessed: 2021-07-22.
- [29] Andrea De Mauro, Marco Greco, and Michele Grimaldi. A formal definition of Big Data based on its essential features. *Library Review*, 2016.
- [30] Mihai Liviu Despa. Comparative study on software development methodologies. *Database systems journal*, 5(3):37–56, 2014.
- [31] Nitish M Devadiga. Tailoring architecture centric design method with rapid prototyping. In *2017 2nd International Conference on Communication and Electronics Systems (ICCES)*, pages 924–930. IEEE, 2017.
- [32] Laney Douglas. 3d data management: Controlling data volume, velocity and variety. *Gartner. Retrieved*, 6(2001):6, 2001.
- [33] Nick Drummond, Alan Rector, Robert Stevens, Georgina Moulton, Matthew Horridge, Hai H Wang, and Julian Seidenberg. Putting owl in order: Patterns for sequences in owl.
- [34] Charles Edeki. Agile software development methodology. *European Journal of Mathematics and Computer Science*, 2(1), 2015.
- [35] Sedick Baker Effendi, Brink van der Merwe, and Wolf-Tilo Balke. Suitability of graph database technology for the analysis of spatio-temporal data. *Future Internet*, 12(5):78, 2020.
- [36] Kemele M Endris, Philipp D Rohde, Maria-Esther Vidal, and Sören Auer. Ontario: Federated query processing against a semantic data lake. In *International Conference on Database and Expert Systems Applications*, pages 379–395. Springer, 2019.
- [37] Antonio Fabregat, Florian Korninger, Guilherme Viteri, Konstantinos Sidiropoulos, Pablo Marin-Garcia, Peipei Ping, Guanming Wu, Lincoln Stein, Peter D’Eustachio, and Henning Hermjakob. Reactome graph database: Efficient access to complex pathway data. *PLoS computational biology*, 14(1):e1005968, 2018.

- [38] Rosa Filguiera, Amrey Krause, Malcolm Atkinson, Iraklis Klampanos, and Alexander Moreno. dispel4py: A python framework for data-intensive scientific computing. *The International Journal of High Performance Computing Applications*, 31(4):316–334, 2017.
- [39] World Economic Forum. Big data, big impact: New possibilities for international development. *World Economic Forum*, 2012.
- [40] Apache Software Foundation. Apache JMeter™. <https://jmeter.apache.org/>, 2021. Last accessed: 2021-07-25.
- [41] Python Software Foundation. Built-in Types. <https://docs.python.org/3/library/stdtypes.html#>, 2021. Last accessed: 2021-07-26.
- [42] Python Software Foundation. Built-in Types - Comparisons. <https://docs.python.org/3/library/stdtypes.html#comparisons>, 2021. Last accessed: 2021-07-26.
- [43] Python Software Foundation. datetime — Basic date and time types. <https://docs.python.org/3/library/datetime.html>, 2021. Last accessed: 2021-07-25.
- [44] Python Software Foundation. pdb — The Python Debugger. <https://docs.python.org/3/library/pdb.html>, 2021. Last accessed: 2021-07-28.
- [45] Sheel Ghule. Risk analysis and mitigation plan in software development. 2014.
- [46] Faming Gong, Yuhui Ma, Wenjuan Gong, Xiaoran Li, Chantao Li, and Xiangbing Yuan. Neo4j graph database realizes efficient storage performance of oil-field ontology. *PloS one*, 13(11):e0207595, 2018.
- [47] Jim Gray, Alex S Szalay, Ani R Thakar, Peter Z Kunszt, Christopher Stoughton, Don Slutz, and Jan vandenBerg. Data mining the SDSS SkyServer database. *arXiv preprint cs/0202014*, 2002.
- [48] OWL Working Group. Web Ontology Language (OWL). <https://www.w3.org/OWL/>, 2012. Last accessed: 2021-07-10.
- [49] RDF Working Group. Resource Description Framework (RDF). <https://www.w3.org/2001/sw/wiki/RDF>, 2014. Last accessed: 2021-07-10.

- [50] Reactome group. Reactome. <https://reactome.org/>, 2021.
- [51] Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5-6):907–928, 1995.
- [52] Patricia Hartmann. DareKB.py. https://gitlab.com/S2057482/dare_kb/-/blob/master/src/server/dare_kb/server/DareKB.py, 2021. Last accessed: 2021-08-04.
- [53] Patricia Hartmann. DKB - Creation of real-world test data. https://gitlab.com/S2057482/dare_kb/-/blob/master/tests/Test%20data%20creation/Real-world%20test%20data.ipynb, 2021. Last accessed: 2021-07-22.
- [54] Patricia Hartmann. DKB - Creation of synthetic test data. https://gitlab.com/S2057482/dare_kb/-/blob/master/tests/Test%20data%20creation/Synthetic%20test%20data.ipynb, 2021. Last accessed: 2021-07-22.
- [55] Patricia Hartmann. DKB - Functionality retests. https://gitlab.com/S2057482/dare_kb/-/blob/master/tests/Test_files/Retests_Jupyter%20notebook.ipynb, 2021. Last accessed: 2021-08-09.
- [56] Patricia Hartmann. DKB - User tests. https://gitlab.com/S2057482/dare_kb/-/blob/master/tests/Test_files/User%20tests_Jupyter%20notebook.ipynb, 2021. Last accessed: 2021-07-25.
- [57] Patricia Hartmann. storage.py. https://gitlab.com/S2057482/dare_kb/-/blob/master/src/server/dare_kb/server/storage.py, 2021. Last accessed: 2021-08-04.
- [58] Clive Humby. Data is the new oil. *Proc. ANA Sr. Marketer's Summit. Evanston, IL, USA*, 2006.
- [59] G2 Inc. Best Graph Databases. <https://www.g2.com/categories/graph-databases>. Last accessed: 2021-07-30.
- [60] Neo4j Inc. Neo4j Named a Leader in Graph Data Platforms by Independent Research Firm). <https://neo4j.com/press-releases/neo4j-leads-graph-data-platform-wave/>, 2020. Last accessed: 2021-07-30.

- [61] Nathan Jennings. Python Debugging With Pdb. <https://realpython.com/python-debugging-pdb/>. Last accessed: 2021-07-28.
- [62] Pete Johnston and Bridget Robinson. *Collections and collection description*. UKOLN, 2002.
- [63] Irvin R Katz and John R Anderson. Debugging: An analysis of bug-location strategies. *Human-Computer Interaction*, 3(4):351–399, 1987.
- [64] J Leslie Keedy and John Rosenberg. Uniform support for collections of objects in a persistent environment. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume II: Software Track*, volume 2, pages 26–27. IEEE Computer Society, 1989.
- [65] Iraklis A Klampanos, Chrysoula Themeli, Alessandro Spinuso, Rosa Filgueira, Malcolm Atkinson, André Gemünd, and Vangelis Karkaletsis. DARE Platform: a Developer-Friendly and Self-Optimising Workflows-as-a-Service Framework for e-Science on the Cloud. *Journal of Open Source Software*, 5(54):2664, 2020.
- [66] Dave Kuhlman. *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman Lutz, 2009.
- [67] Archana P Kumar, Abhishekh Kumar, and Vipin N Kumar. A comprehensive comparative study of sparql and sql. *International Journal of Computer Science and Information Technologies*, 2(4):1706–1710, 2011.
- [68] Rohit Kumar Kaliyar. Graph databases: A survey. In *International Conference on Computing, Communication & Automation*, pages 785–790. IEEE, 2015.
- [69] Raoul Kwiimi and Jean Vincent Fonou-Dombeu. Storing and querying ontologies in relational databases: An empirical evaluation of performance of database-based ontology stores. In *Proceedings of the 9th International Conference on Advances in Semantic Processing (SEMAYRO 2015)*, pages 6–12, 2015.
- [70] Jean-Baptiste Lamy. Owlready2 - Benchmark. http://www.lesfleursdunormal.fr/static/informatique/owlready/benchmark_en.html. Last accessed: 2021-07-30.

- [71] Jean-Baptiste Lamy. The flowers of evidence. http://www.lesfleursdunormal.fr/static/index_en.html. Last accessed: 2021-07-30.
- [72] Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine*, 80:11–28, 2017.
- [73] Jean-Baptiste Lamy. Owlready 2 - 0.13. <http://owlready.8326.n8.nabble.com/Owlready-2-0-13-td756.html>, 2018. Last accessed: 2021-07-21.
- [74] Jean-Baptiste Lamy. SPARQL queries. <https://owlready2.readthedocs.io/en/latest/sparql.html>, 2019. Last accessed: 2021-07-29.
- [75] Amélie Levray, Patricia Hartmann, Rui Zhao, and Malcolm Atkinson. DARE Knowledge Base User Manual. <https://docs.google.com/document/d/1u62251KnRURztDyBbxo77yfGGBpjxjzlWB1SinArxX0/edit#heading=h.mhz5evrki0j1>, 2020. Last accessed: 2021-08-10.
- [76] Amélie Levray and Rui Zhao. DKB Specification. https://docs.google.com/document/d/1bh2CzZJOUOYL_1nPJI8f5hcokHYxB1m9NHr9owWq_F0, 2020. Last accessed: 2021-07-28.
- [77] SolarWinds Worldwide LLC. Database Performance Monitor. <https://www.solarwinds.com/database-performance-monitor>, 2021. Last accessed: 2021-07-25.
- [78] Steve Lohr. The Age of Big Data. <https://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html>, 2012. Last accessed: 2021-08-05.
- [79] José María Cavanillas, Edward Curry, and Wolfgang Wahlster. *New horizons for a data-driven economy: a roadmap for usage and exploitation of big data in Europe*. Springer Nature, 2016.
- [80] Bernard Marr. What Is Unstructured Data And Why Is It So Important To Businesses? An Easy Explanation For Anyone. <https://www.forbes.com/sites/bernardmarr/2019/10/16/what-is-unstructured-data-and-why-is-it-so-important-to-businesses-an-easy-explanation-for-anyone/>, 2019. Last accessed: 2021-04-14.

- [81] Kevin McCusker and Sau Gunaydin. Research using qualitative, quantitative or mixed methods and choice based on the research. *Perfusion*, 30(7):537–542, 2015.
- [82] Jim Melton and Alan R Simon. *SQL: 1999: Understanding relational language components*. Elsevier, 2001.
- [83] Tilman Michaeli and Ralf Romeike. Improving debugging skills in the classroom: The effects of teaching a systematic debugging process. In *Proceedings of the 14th workshop in primary and secondary computing education*, pages 1–7, 2019.
- [84] Robert B Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, 1968.
- [85] Jakob Nielsen. *Usability engineering*. Morgan Kaufmann, 1994.
- [86] Maria A Nieto-Santisteban, Aniruddha R Thakar, Alexander S Szalay, and Jim Gray. Large-scale query and xmatch, entering the parallel zone. *arXiv preprint cs/0701167*, 2007.
- [87] Hewlett Packard. Failing to meet mobile app user expectations: a mobile user survey. *Tech. rep.*, 2015.
- [88] Eric Pardede, J Wenny Rahayu, and David Taniar. Mapping methods and query for aggregation and association in object-relational database using collection. In *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, volume 1, pages 539–543. IEEE, 2004.
- [89] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *International semantic web conference*, pages 30–43. Springer, 2006.
- [90] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45, 2009.
- [91] Eric Prud’Hommeaux, Andy Seaborne, et al. Sparql query language for rdf. w3c. *Internet: <https://www.w3.org/TR/rdf-sparql-query/>*, 2008. Last accessed: 2021-07-29.

- [92] Giuseppe Psaila. A database model for heterogeneous spatial collections: Definition and algebra. In *2011 International Conference on Data and Knowledge Engineering (ICDKE)*, pages 30–35. IEEE, 2011.
- [93] Hongbin Qiu, Aihua Zhou, Bin Hu, Bo Chai, Yan Song, and Rui Chen. Design and Implementation of Power Grid Graph Data Management Platform based on Distributed Storage. In *IOP Conference Series: Earth and Environmental Science*, volume 234, page 012026. IOP Publishing, 2019.
- [94] Raghu Ramakrishnan, Johannes Gehrke, and Johannes Gehrke. *Database management systems*, volume 3. McGraw-Hill New York, 2003.
- [95] Pat Research. Top 27 Graph Databases. <https://www.predictiveanalyticstoday.com/top-graph-databases/>. Last accessed: 2021-07-30.
- [96] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases: new opportunities for connected data*. O’Reilly Media, Inc., 2015.
- [97] Pablo Romero, Benedict Du Boulay, Richard Cox, Rudi Lutz, and Sallyann Bryant. Debugging strategies and tactics in a multi-representation software environment. *International Journal of Human-Computer Studies*, 65(12):992–1009, 2007.
- [98] Masoud Salehpour and Joseph G Davis. Knowledge Graphs for Processing Scientific Data: Challenges and Prospects. *arXiv preprint arXiv:2004.06203*, 2020.
- [99] Christie Schneider. The biggest data challenges that you might not even know you have. *IBM Watson*, 2016.
- [100] Ascii Set. ASCII characters 0 to 127. <http://asciiset.com/>, 2019. Last accessed: 2021-07-26.
- [101] GauravVaswani ShefaliPatil and Anuradha Bhatia. Graph databases-an overview. *IStudent, ME Computers, Terna College of Engg, Navi Mumbai*, 2:657–660, 2014.
- [102] Arie Shoshani, Frank Olken, and Harry Kwing Tong Wong. Characteristics of scientific databases. 1984.

- [103] Aisha Siddiqa, Ahmad Karim, and Abdullah Gani. Big data storage technologies: a survey. *Frontiers of Information Technology & Electronic Engineering*, 18(8):1040–1070, 2017.
- [104] Josep Silva. A survey on algorithmic debugging strategies. *Advances in engineering software*, 42(11):976–991, 2011.
- [105] SmartBear Software. LoadNinja. <https://loadninja.com/>, 2021. Last accessed: 2021-07-25.
- [106] solidIT GmbH. DB-Engines Ranking of Graph DBMS. <https://db-engines.com/en/ranking/graph+dbms>, 2021. Last accessed: 2021-07-30.
- [107] Peter Sondergaard. Big data fades to the algorithm economy. Retrieved from *Forbes*: <http://www.forbes.com/sites/gartnergroup/2015/08/14/big-data-fades-to-the-algorithm-economy>, 2015.
- [108] Alexander S Szalay, Peter Z Kunszt, Ani Thakar, Jim Gray, Don Slutz, and Robert J Brunner. Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey. *ACM SIGMOD Record*, 29(2):451–462, 2000.
- [109] The University of Edinburgh. dispel4py 2015.06 documentation. <https://pythonhosted.org/dispel4py/>, 2014. Last accessed: 2021-07-25.
- [110] William Van Lenthien and Kenneth M Anderson. A metainformatical view of collections. In *Proceedings of the 2005 symposia on Metainformatics*, pages 16–es, 2005.
- [111] Guido van Rossum, Barry Warsaw, and Nick Coghlan. PEP 8 – Style Guide for Python Code. <https://www.python.org/dev/peps/pep-0008/#programming-recommendations>, 2013. Last accessed: 2021-07-28.
- [112] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple pattern fragments: a low-cost knowledge graph interface for the web. *Journal of Web Semantics*, 37:184–206, 2016.
- [113] Jing Wang, Nikos Ntarmos, and Peter Triantafillou. Graphcache: A caching system for graph queries. 2017.

- [114] Jonathan Stuart Ward and Adam Barker. Undefined by data: a survey of big data definitions. *arXiv preprint arXiv:1309.5821*, 2013.
- [115] Adrienne Watt, Nelson Eng, et al. *Database design - 2nd Edition*. BCcampus, 2014. Retrieved from <https://opentextbc.ca/dbdesign01/>.
- [116] Jim Webber and Rik Van Bruggen. *Graph Databases for Dummies*. John Wiley & Sons, Inc., 2020.
- [117] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. Novice reflections on debugging. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 73–79, 2021.
- [118] E James Whitehead Jr. Design spaces for link and structure versioning. In *Proceedings of the 12th ACM conference on Hypertext and Hypermedia*, pages 195–204, 2001.
- [119] E James Whitehead Jr. Uniform comparison of data models using containment modeling. In *Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, pages 182–191, 2002.
- [120] Byoung-Ha Yoon, Seon-Kyu Kim, and Seon-Young Kim. Use of graph database for the integration of heterogeneous biological data. *Genomics & informatics*, 15(1):19, 2017.
- [121] Noel Yuhanna, Gene Leganza, and Daniel Weber. The forrester wave™: Graph data platforms, q4 2020. *Forrester Research Inc*, 2020.
- [122] Andreas Zeller. *Why programs fail: a guide to systematic debugging*. Elsevier, 2009.

Appendix A

Test cases and results

A.1 Site functions

“Each DKB site, shared by users, is set up using management tools [...] [which] are described in [DKB User Manual’s] section DKB site management. Thus, [this] configures the site_name used by login and other site properties. It also pre-populates the site with a standard library of concepts to help you get started.” (Taken from the DKB User Manual [75])

Test scenario	Expected outcome	Test result
DKB login ¹ : First of all, import the client library DKBlib via import DKBlib as DKB before executing the function login(<i>site_name</i> , <i>username</i>) on the DKB object with site- and username and storing the returned DKBService in object dkb .	Success message such as “INFO:werkzeug:localhost-[DD/MM/YYYY HH:MM:SS]” ←[37mPOST /user/login HTTP/1.1←[0m”200-” appearing in the command prompt to reflect the successful DKB login.	Success
DKB status ² : On DKBService object dkb execute the status() function to retrieve the status information.	Printing the status information should return a list with the following parameters and their according values: <pre>{‘contexts_available’: [‘’], ‘current_context’: ‘’, ‘session_id’: ‘’, ‘site_name’: ‘’, ‘username’: ‘’}.</pre>	Success

¹“The function login allows any user to connect to a site previously created by specifying the site_name and a username. An optional argument is session_id. This function returns a DKBService.” (Taken from DKB User Manual [75])

²“The status function returns a dictionary of information about the current use of the DKB site. It contains the session_id, the list of contexts available, the current context, and the name of the site as

<p>DKB logout³: On DKBSERVICE object <code>dkb</code> execute the <code>logout()</code> function to disconnect from the DKB server.</p>	<p>Success message such as “INFO:werkzeug:localhost-[DD/MM/YYYY HH:MM:SS]” ←[37mPOST /user/logout HTTP/1.1←[0m”200-” appearing in the command prompt to reflect the successful DKB logout.</p>	<p>Success</p>
---	---	----------------

Table A.1: DKB site functions

A.1.1 User Manual updates

Owlready2 warning message Depending on one’s own Owlready2 version and available modules, the following warning message may appear after the DKB server has been started: **“Owlready2 Warning: optimized Cython parser module ‘owlready2_optimized’ is not available, defaulting to slower Python implementation”**.

From a functionality point of view it is not required to use the optimized module, however, using it can result in a 20% performance acceleration for parsing ontologies [73, 72] (please refer to Appendix C.1.1).

Windows symbolic link issue Windows users may encounter problems with using DKB if – for instance – the `../common` path in the `common.txt` files is not resolving properly (symbolic link issue). In such a case users have to manually re-create those links as now described in the User Manual [75] (please refer to Appendix C.1.2).

well as the currently logged in user.” (Taken from DKB User Manual [75])

³“The `close` function records in the storage all of the information created. It terminates interaction with the DKB started at login.” (Taken from DKB User Manual [75])

A.2 Context functions

Test scenario	Expected outcome	Test result
<p>Creation of new context⁴: on DKBService object <code>dkb</code>, execute the <code>new_context('context_name', 'context_description')</code> function with a specified <i>context name</i> and <i>description</i> to create a new context.</p>	<p>The newly created context should be listed under the available contexts when executing <code>status(['contexts_available'])</code> on the DKBService object <code>dkb</code> and printing the results.</p>	Success
<p>Context status⁵: on DKBService object <code>dkb</code> execute the <code>context_status('optional_context_name')</code> function both</p> <ul style="list-style-type: none"> • Without a context name (optional parameter <i>optional_context_name</i> empty) • With a specified context name <i>optional_context_name</i> (not current context) <p>to retrieve the (current/ given) context's status information.</p>	<p>In both cases, printing the (current/ given) context's status information should return a list with the following parameters and their according values: {'concepts': {}, 'identifier': '', 'instances': {}, 'mode': '', 'owner': '', 'prefix': '', 'search_path': [], 'state': '', 'title': '', 'users': []}.</p>	Success

⁴“`new_context()` creates a new context. The owner defaults to the current user. The new context has a `search_path` that is a list of contexts to search after looking for a name in the new context. Each context has a `prefix` that must be unique for this site. It may have a title that indicates its purpose.” (Taken from DKB User Manual [75])

⁵“The function `context_status()` returns a dictionary of information about the current context or any other context if an argument is provided. The dictionary contains information like: the identifier, the owner, the prefix, the search_path, the state, the title, the list of concepts and the list of instances, the mode: ‘R’ or ‘W’, users.” (Taken from DKB User Manual [75])

<p>Switching into a different context⁶: Again on DKBSer-vice object <code>dkb</code>, execute the <code>enter('context_name', 'mode')</code> function with both</p> <ul style="list-style-type: none"> • Read 'R' and • Write 'W' mode <p>and print the re-sult of <code>dkb.status()</code> [<code>'current_context'</code>] and <code>dkb.context_status()['mode']</code> to verify the current context and its access mode.</p>	<p>Both the current context and its access mode should match the parameters specified in the <code>enter('context_name', 'mode')</code> function</p>	<p>Success</p>
<p>Leaving the current context⁷: Execute the <code>leave()</code> function on DKBSer-vice object <code>dkb</code> and check with either <code>dkb.status()['current_context']</code> or <code>dkb.context_status()</code> what the current context is.</p>	<p>According to the DKB User Manual, executing the <code>leave()</code> function on DKBSer-vice object <code>dkb</code> should leave the user with no assigned context.</p>	<p>Differs (UMU)</p>
<p>Context search path: <i>Get context search path</i>⁸: on DKBSer-vice object <code>dkb</code>, execute the <code>get_search_path()</code> function.</p>	<p>The expected result would be a list of all the context prefixes which are contained in the current context's search path.</p>	<p>Success</p>

⁶“The `enter()` function allows you to switch between contexts, leaving your current one and entering the one specified in the argument `a_prefix`. The second argument specifies whether you are entering in reading mode or writing mode. It accepts two values 'R' and 'W'.” (Taken from DKB User Manual [75])

⁷“The `leave` function quits the current context. It leaves you with no context set and therefore you are prevented from creating new concepts or instances until you enter another context.” (Taken from DKB User Manual [75])

⁸“The `get_search_path()` function returns the list of prefixes that are in the `search_path` of the current context.” (Taken from DKB User Manual [75])

<p>Context search path: <i>Set context search path</i>⁹: Enter an existing context in both</p> <ul style="list-style-type: none"> • Read ‘R’ and • Write ‘W’ mode <p>and afterwards execute the <code>set_search_path('new_search_path')</code> function on DKBService object <code>dkb</code> with both a(n)</p> <ul style="list-style-type: none"> • Existing context names as ‘<code>new_search_path</code>’ and • Invented context names as ‘<code>new_search_path</code>’. <p>Check the result by running <code>dkb.get_search_path()</code>.</p>	<ul style="list-style-type: none"> • Write mode: search path matches (list) of given context prefixes • Read mode: user not able to adjust context search path • Existing context name(s): search path matches (list) of given context prefixes • Invented context names: error message <i>Context ‘xyz’ cannot be found, nonexistent or access not granted. Function set_search_path failed.</i> 	Success
<p>Context freeze¹⁰: Enter an existing context in both</p> <ul style="list-style-type: none"> • Read ‘R’ and • Write ‘W’ mode <p>and afterwards execute the <code>context_freeze('optional_context_name')</code> function on DKBService object <code>dkb</code>, both</p> <ul style="list-style-type: none"> • Without a context name (optional parameter <i>optional_context_name</i> empty) • With a specified context name <i>optional_context_name</i> (not current context). 	<p>For all context states and no matter whether a <i>optional_context_name</i> has been entered or not, the result should be the same:</p> <ul style="list-style-type: none"> • Write mode: the context state changed to ‘frozen’ • Read mode: the context state is unchanged. 	Success

⁹“The `set_search_path()` function replaces the `search_path` of the current context with the one provided (`new_search_path`).” (Taken from DKB User Manual [75])

¹⁰“The function `context_freeze()` freezes the current context or the specified context into a state that doesn’t permit any further modifications to instances in this context nor to the search path. A method may use this to ensure that the state of a context is preserved for diagnostic and audit purposes. Since

<p>Additionally, check function <code>context_freeze('optional_context_name')</code> on contexts in different states:</p> <ul style="list-style-type: none"> • 'active' • 'new' <p>Finally, check the result by running <code>dkb.context_status('optional_context_name')['state']</code>.</p> <p>Context deprecate¹¹: Enter an existing context in both</p> <ul style="list-style-type: none"> • Read 'R' and • Write 'W' mode <p>and afterwards execute the <code>context_deprecate('optional_context_name')</code> function on DKBService object <code>dkb</code>, both</p> <ul style="list-style-type: none"> • Without a context name (optional parameter <code>optional_context_name</code> left empty) • With a specified context name <code>optional_context_name</code> (not current context). 	<p>For all context states <u>except</u> 'frozen': no matter whether a <code>optional_context_name</code> has been entered or not, the result should be the same:</p> <ul style="list-style-type: none"> • Write mode: the context state changed to 'deprecated' • Read mode: The context state is unchanged. 	<p>Success</p>
---	--	----------------

that method itself may be being debugged, `context_reset()` is still permitted.” (Taken from DKB User Manual [75])

¹¹“The function `context_deprecate()` changes the state of the context to 'deprecated'. This still allows the user to enter and modify the context, but on entry they will be warned that the context is soon to be discarded.” (Taken from DKB User Manual [75])

<p>Additionally, check function <code>context_deprecate('optional_context_name')</code> on contexts in different states:</p> <ul style="list-style-type: none"> • 'active' • 'new' • 'frozen' <p>Finally, check the result by running <code>dkb.context_status('optional_context_name')['state']</code>.</p> <p>Context reset¹²: enter an existing context in both</p> <ul style="list-style-type: none"> • Read 'R' and • Write 'W' mode <p>and afterwards execute the <code>context_reset('optional_context_name')</code> function on DKBService object <code>dkb</code>, both</p> <ul style="list-style-type: none"> • Without a context name (optional parameter <code>optional_context_name</code> left empty) • With a specified context name <code>optional_context_name</code> (not current context). 	<p>For all context states and no matter whether a <code>optional_context_name</code> has been entered or not, the result should be the same:</p> <ul style="list-style-type: none"> • Write mode: the context state changed to 'discarded' • Read mode: the context state is unchanged. 	<p>Failure (AB)</p>
--	---	---------------------

¹²“The function `context_reset()` discards the context, making its prefix available to be reused. All concepts and instances in this context are marked 'discarded' and therefore their names and identities may be recreated. This is designed to support development and debugging, as a new version of that the code that created the context and its contents can now be rerun.” (Taken from DKB User Manual [75])

<p>Additionally, check function <code>context_reset('optional_context_name')</code> on contexts in different states:</p> <ul style="list-style-type: none"> • 'active' • 'new' • 'frozen' • 'deprecated' <p>Finally, check the result by running <code>dkb.context_status('optional_context_name')['state']</code>.</p>		
---	--	--

Table A.2: DKB context functions

A.2.1 Application bug

Discarding (resetting) a frozen context As of now, it is not possible to `context_reset` a 'frozen' context (figure A.1(b)) since such a context – with state 'frozen' – cannot be entered (`enter()`) in write mode 'W' (as figure A.1(a) illustrates). This however is (currently) required to discard the context via `context_reset`.

A.2.2 User Manual updates

Context naming convention Some information regarding the context naming convention have been added (e.g., no whitespace allowed, else a **DKBException: IdentifierWrong** is thrown (please refer to Appendix C.3).

Entering a context One is not able to `enter()` a context – neither in write 'W' nor in read 'R' mode – if another user is currently logged into the same context in write 'W' mode (**'MultiUser' DKBException**). The fact, that this only works if both (or all) users access the context in read mode 'R' has been added to the User Manual (please refer to Appendix C.3).

Leaving the current context According to the DKB User Manual, executing the `leave()` function on DKBService object `dkb` should leave the user with no assigned context.

After the context was successfully frozen, we can test whether changes to search path and instances are possible (by checking whether a context access in write mode is possible). The expectation is to receive an `WritingPermissionDeniedError` error.

```

M try:
    dkb.enter(currContext_freeze, 'W')
    print("!! Error !! Entering a frozen context in write mode should have been denied.")
except Exceptions.WritingPermissionDeniedError:
    print("Entering a frozen context was successfully denied.")

```

Entering a frozen context was successfully denied.

Read access on the other hand should still be possible.

```

M try:
    dkb.enter(currContext_freeze, 'R')
    print("Entering a frozen context in read mode was successful.")
except:
    print("!! An error occurred !!")

```

Entering a frozen context in read mode was successful.

(a) Enter frozen context

```

M # Enter context
dkb.enter(currContext_freeze, 'R')
print("Check the current context", dkb.status()['current_context'] +
      "\nstate:", dkb.context_status()['state'])
dkb.context_reset()

print("Verify context", currContext_freeze + "\nstate:",
      "#\nCurrent context:", dkb.context_status()['current_context'],
      "#\nContext state:",
      dkb.context_status(currContext_freeze)['state'])

```

```

286 def context_deprecate(self, a_prefix = None) -> None:
287
    \github\dare_kb\src\client\DKBlib\csbridge.py in wrapper(*args, **kwargs)
    37     return func(*args, **kwargs)
    38     except DKBException as e:
--> 39         e.coerce()
    40     return wrapper
    41
    74     \github\dare_kb\src\client\DKBlib\common\cs_exception.py in coerce(self)
    75     def coerce(self):
--> 76         raise find_exception(self.error_code)(self.message, *self.extra)
    77
    78
WritingPermissionDeniedError: You are not allowed to do this action. reset_context
([],)

```

(b) Reset frozen context

Figure A.1: DKB: entering and resetting a 'frozen' context

However, instead of leaving the user with no context, the user is being assigned to the first context ever created (“default” context) and does not leave the user with no context at all; Thus, it is still possible to create concepts and instances and therefore the User Manual has been updated accordingly (please refer to Appendix C.3).

Freeze (), Deprecate () and Reset () a context None of the three functions `context_freeze()`, `context_deprecate()` and `context_reset()` works if the given context has not been entered in Write mode ‘W’. Trying to do so results in a `WritingPermissionDeniedError`. Therefore, this information has been added to the User Manual (please refer to Appendix C.3).

A.3 Concept functions

Test scenario	Expected outcome	Test result
<p>Creation of new concept¹³: on DKBService object <code>dkb</code>, execute the <code>new_concept ('concept_name')</code> function with a specified <i>concept name</i> to create a new concept. Additionally, create concepts which contain at least one the following parameters each time:</p> <ul style="list-style-type: none"> • specialises • mutability • required • recommended • optional • translation • description • methods • py_class. <p>Thereafter verify the created concepts by printing the results of <code>dkb.context_status() ['concepts']</code>.</p>	<p>All created concepts should be listed under context status 'concepts'.</p>	<p>Success</p>

¹³“The function `new_concept ()` creates and stores a new concept developed by the user. A concept is specified by a name, it can be a subclass of another concept, specified by `specialises`. And have multiple optional arguments that are defined below. The concept created belongs to the context it has been created in but can be accessed from any context, normally one with it in its `search_path`.” (Taken from DKB User Manual [75])

<p>Get concept¹⁴: On DKBSer- vice object <code>dkb</code> execute the <code>get('concept_name' OR 'concept_PID')</code> function with both</p> <ul style="list-style-type: none"> • Concept name <code>concept_name</code> • Concept PID <code>concept_PID</code> <p>to retrieve the specified concept. Moreover, test the two optional pa- rameters (separately and together):</p> <ul style="list-style-type: none"> • 'only_these' (empty¹⁵ OR se- lection of <ul style="list-style-type: none"> - specialises - mutability - required - recommended - optional - translation - description - methods - py_class. • 'ignore_discarded' (Boolean) <p>Additionally, check the <code>get('concept')</code> function on concepts in different states (same state as context they belong to):</p> <ul style="list-style-type: none"> • 'active' • 'new' • 'frozen' • 'deprecated' • 'discarded' 	<p>In any of the described cases, the according concept together with its specified parameter(s) should be re- turned.</p>	<p>Partial failure</p>
---	--	------------------------

¹⁴“The `get()` function allows the user to look for a specific concept or instance given an identity passed as a parameter. It returns a dictionary of the attributes of the concept or instance. The identity parameter can be a name only, a name with the context specified, or a PID. If it is only the name, the search paths determine where to look if it is not found in the current context. The function has two optional parameters: `only_these` which allows selection of attributes the user wants returned. And `ignore_discarded` which allows it to include concepts and instances that have been discarded.” (Taken from DKB User Manual [75])

¹⁵'only_these' = [] solely verifies that the concept exists

<p>Find concept¹⁶: Again, on DKBService object <code>dkb</code> execute the <code>find('query')</code> function. The <i>query</i> should look as follows: <code>'(operator, search_criteria, search_value)'</code>, where <i>operator</i> can take <code>'=='</code>, <code>'isa'</code> and <code>'isa.exactly'</code> values and the <i>search_criteria</i> can be anything from <code>'PID'</code>, <code>'prefix'</code>, <code>'description'</code>, <code>'state'</code>, <code>'mutability'</code>, <code>'timestamp'</code>, <code>'translation'</code>, <code>'method'</code> and <code>'py_class'</code>. Moreover, test the three optional parameters (separately and together):</p> <ul style="list-style-type: none"> • <code>'pid_only'</code> (Boolean) • <code>'only_these'</code> (selection of <ul style="list-style-type: none"> - specialises - mutability - required - recommended - optional - translation - description - methods - py_class. • <code>'ignore_discarded'</code> (Boolean) <p>Additionally, check <code>find('query')</code> on concepts in different states (state alike context they belong to):</p> <ul style="list-style-type: none"> • <code>'active'</code> • <code>'new'</code> • <code>'frozen'</code> • <code>'deprecated'</code> 	<p>In any of the described cases (and with any parameter selection and combination), the matching concept(s) together with its/ their specified parameter(s) should be returned.</p>	<p>Partial failure</p>
---	--	------------------------

¹⁶“The `find()` function allows the user to query the knowledge base about any instances (concepts included) that exist. It takes a query and three optional arguments: `pid_only`, `only_these` and

<ul style="list-style-type: none"> • ‘discarded’ <p>Finally, test also more complex queries, concatenated either by ‘AND’ or ‘OR’ (e.g., query = (‘AND’, (cond1, cond2)) with cond1 = cond2 = (operator, search_criteria, search_value) as described above.</p>		
--	--	--

Table A.3: DKB concept functions

A.3.1 Application bugs

General find() and get() concept issue after DKB server restart Once the DKB server had been restarted via the command prompt¹⁷, it was no longer possible to access the previously (during the former DKB server session) created and meanwhile saved concepts and instances with the get() and find() functions: For concepts, get() and find() returned an “**Internal Server Error**”, as illustrated in figure A.2.

This error has meanwhile been corrected: previously created concepts are now also selectable after a DKB server restart.

```

Re-logout after disconnecting from the server
The issue of no longer being able to get() and find() previously created concepts and instances happens once the server was shutdown and started again at a
later point in time.
Closing the connection to the server and logging in again with the same user (or a different user) at a later point in time while the same server session was
before is still running, does not result in the 500 Internal Server Error.

M import DKBlib as DKB
import datetime

startTime_relogin = datetime.datetime.now()
dkb = DKB.login('ex3KB', 'testuser')
endTime_relogin = datetime.datetime.now()
print('Time it takes to login to DKB:', str(endTime_relogin - startTime_relogin))

Time it takes to login to DKB: 0:00:16.766665

M #dkb.enter(context_TestConcepts, 'W')
dkb.enter('context_ConceptCreation', 'W')
print(dkb.status()['current_context'])

TestCon = 'ex3KB:context_ConceptCreation@:TestCon'
tci = 'ex3KB:context_ConceptCreation:1B:tci'
Context_ConceptCreation

M start_time_ConceptGetName_relogin = datetime.datetime.now()
print('Get() concept by name:\n', dkb.get('TestCon')) # By name
end_time_ConceptGetName_relogin = datetime.datetime.now()
print('Time it takes to get a concept:', TestCon + ", by name:",
      str(end_time_ConceptGetName_relogin - start_time_ConceptGetName_relogin))

start_time_ConceptGetPID_relogin = datetime.datetime.now()
print('Get() concept by PID:\n', dkb.get(TestCon)) # By PID
end_time_ConceptGetPID_relogin = datetime.datetime.now()
print('Time it takes to get a concept:', TestCon + ", by name:",
      str(end_time_ConceptGetPID_relogin - start_time_ConceptGetPID_relogin))

61     github/dare_kb/src/client/DKBlib/cbridge.py in _do_request(site, method, path, parse_json, **data)
62         if 'Content-Type' in r.headers and r.headers['Content-Type'] == 'application/json':
--> 63             msg = json.loads(r.text)
64             handle_server_exception(r.status_code, msg)
65             return msg
66
67     github/dare_kb/src/client/DKBlib/common/cs_exception.py in handle_server_exception(status_code, message)
68         raise DKBException(error_code, message, extra)
--> 66         raise Exception(message)
67
68

Exception: <DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>500 Internal Server Error</title>
<n>Internal Server Error</n>
<p>The server encountered an internal error and was unable to complete your request. Either the server is overloaded or
there is an error in the application.</p>

```

Figure A.2: DKB: accessing previously created concept after server restart

ignore_discarded. These last two arguments work the same way as for get. The query supports different operations.” (Taken from DKB User Manual [75])

¹⁷ Server shut down: CTRL + C for quitting, Server start: dkb_server command

Find concept issues Previously, `find()` concepts did not work for the search criteria name, description, state, mutability, timestamp, translation, method and `py_class` - solely **PID** and **prefix** returned the expected outcome. The other parameter either issued an empty result or one of the following errors (as depicted in figure A.3):

- **TypeError**
 - Unhashable type: 'dict', []
 - Unhashable type: 'IndividualValueList', []
 - `get_instance()` missing 1 required positional argument: 'name', []
- **Internal Server Error**

After debugging and correcting the error, `find()` now returns the specified concepts and instances without any such error message.

```

3.3 Find() concept
The find() function allows the user to query the knowledge base about any instances (concepts included) that exist. It takes a query and three optional arguments: pid_only, only_these and ignore_discarded. These last two arguments work the same way as for get. The query supports different operations:
• AND and OR query
• The logic condition of the form (operation, property, value) with the supported operation: '='
• The logic condition of the form (operation, concept) with the supported operations
• 'is' which is true if an instance is of the concept or a concept is concept specialises - a transitive relationship
• 'is_exactly' which is satisfied if an instance is of the specified concept

Type of arguments and return:
find
↳ Query pid_only: boolean = True
only_these: List[str] = None
ignore_discarded: boolean = False
↳ Union(List[PID], List[Instance])

(Taken from the DKB User Manual)

M db.enter(context_testconcepts, "M")
db.enter(context_testconcepts, "W")
db.starts()

Find by name:
M query_equalname = ("=", "name", "Testcon")
print("Query with name:", query_equalname)
Query with name: ("=", "name", "Testcon")

M start_time_ConceptIndName = datetime.datetime.now()
res_EqualName = db.find(query_equalname, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state', 'mutability'],
ignore_discarded = False)
print(res_EqualName)
end_time_ConceptIndName = datetime.datetime.now()
print("Time it takes to find a concept:", Testcon, "by name and print only a few parameters:",
str(end_time_ConceptIndName - start_time_ConceptIndName))
-----
61 if 'content-type' in headers and headers['content-type'] == 'application/json':
62     msg = json.loads(r.text)
63     handle_server_exception(r.status_code, msg)
64     return msg
65
Exception: 500 Internal Server Error The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Exception: 500 Internal Server Error The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Find by description:
M query_equalitydescription = ("=", "description", "")
print("Query with description:", query_equalitydescription)
# CR: 'onlydescription'
query_equalitydescription = ("=", "description", "this is a concept with a description")
print("Query with description:", query_equalitydescription)
Query with description: ("=", "description", "")
Query with description: ("=", "description", "this is a concept with a description")

M start_time_ConceptIndDescription = datetime.datetime.now()
res_EqualityDescription = db.find(query_equalitydescription, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state',
mutability'], ignore_discarded = False)
print(res_EqualityDescription)
end_time_ConceptIndDescription = datetime.datetime.now()
print("Time it takes to find a concept:", Testcon, "by description and print only a few parameters:",
str(end_time_ConceptIndDescription - start_time_ConceptIndDescription))
-----
242 res = find(self.site, self.session_id, query, pid_only, only_these, ignore_discarded)
243     if not self.raw_data:
244         for i in res:
TypeError: ("unhashable type: 'IndividualValueList'", [])

Find by state:
M query_EqualState = ("=", "state", "new")
print("Query with state:", query_EqualState)
Query with state: ("=", "state", "new")

M start_time_ConceptIndState = datetime.datetime.now()
res_EqualState = db.find(query_EqualState, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state',
mutability'], ignore_discarded = False)
print(res_EqualState)
end_time_ConceptIndState = datetime.datetime.now()
print("Time it takes to find a concept:", Testcon, "by state and print only a few parameters:",
str(end_time_ConceptIndState - start_time_ConceptIndState))
-----
242 res = find(self.site, self.session_id, query, pid_only, only_these, ignore_discarded)
243     if not self.raw_data:
244         for i in res:
TypeError: ("get_instance") missing 1 required positional argument: "name", []

TypeError: ("get_instance") missing 1 required positional argument: "name", []

Find by mutability:
M query_EqualMutu = ("=", "mutability", "mutable")
print("Query with mutability:", query_EqualMutu)
Query with mutability: ("=", "mutability", "mutable")

M start_time_ConceptIndMutu = datetime.datetime.now()
res_EqualMutu = db.find(query_EqualMutu, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state', 'mutability'],
ignore_discarded = False)
print(res_EqualMutu)
end_time_ConceptIndMutu = datetime.datetime.now()
print("Time it takes to find a concept:", Testcon, "by mutability and print only a few parameters:",
str(end_time_ConceptIndMutu - start_time_ConceptIndMutu))
-----
242 res = find(self.site, self.session_id, query, pid_only, only_these, ignore_discarded)
243     if not self.raw_data:
244         for i in res:
TypeError: ("get_instance") missing 1 required positional argument: "name", []

TypeError: ("get_instance") missing 1 required positional argument: "name", []

Find by translation:
M query_EqualTrans = ("=", "translation", "")
print("Query with translation:", query_EqualTrans)
# CR: 'ConceptIndTranslation'
query_EqualTransCR = ("=", "translation", "name: 'string'")
query_EqualTransCR = ("=", "translation", "name: String")
print("Query with translation:", query_EqualTransCR)
print(db.find("=", "pid", CR, pid_only = False))
Query with translation: ("=", "translation", "")
Query with translation: ("=", "translation", "name: 'string'")
Query with translation: ("=", "translation", "name: String")
[{"name": "ConceptIndTranslation", "specialises": "concept_creation", "pid": "ex:context_conceptcreation:concept",
"translation": "specialises: 'ex:sk:concept', instance_of: 'ex:sk:concept', description: '', state: 'new',
timestamp: datetime.datetime(2020, 6, 16, 11, 49, 866666), mutability: 'mutable', translation: 'name: 'string'",
py_class: None, methods: [], optional: [], required: [], recommended: []}]

M start_time_ConceptIndTrans = datetime.datetime.now()
res_EqualTrans = db.find(query_EqualTrans, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state',
mutability'], ignore_discarded = False)
print(res_EqualTrans)
end_time_ConceptIndTrans = datetime.datetime.now()
print("Time it takes to find a concept:", Testcon, "by translation and print only a few parameters:",
str(end_time_ConceptIndTrans - start_time_ConceptIndTrans))
-----
242 res = find(self.site, self.session_id, query, pid_only, only_these, ignore_discarded)
243     if not self.raw_data:
244         for i in res:
TypeError: ("unhashable type: 'dict'", [])

```

(a) Name and description

(b) State, mutability and translation

Figure A.3: DKB: find concept issues

A.3.2 Functionality deficiencies

Concepts creation: limited attribute types Currently only 'string' and 'integer' types can be set during the concept attribute specification (e.g., specifying a Boolean variable returns `InstanceNotFoundError`: The instance for : 'variable_type' does not exist or cannot be found. Function resolve failed.)

```

Find by py_class
M query_EqualClass = ('==', 'py_class', None)
print('Query with py_class', query_EqualClass)
# C9P: 'QuitMyClass'
query_EqualClassC9P = ('==', 'py_class', 'String')
print('Query with description:', query_EqualClassC9P)
#print(dkb.find( ('==', 'pid', 'c9p'), pid_only = False ))

Query with py_class: ('==', 'py_class', None)
[{'name': 'QuitMyClass', 'prefix': 'Context_ConceptCreation', 'pid': 'ex3k8:Context_ConceptCreation:11:QuitMyClass', 'specialises': 'ex3k8:lib1:Concept', 'instance_of': 'ex3k8:lib1:Concept', 'description': '', 'state': 'New', 'timestamp': datetime.datetime(2021, 6, 16, 11, 1, 10, 426695), 'mutability': 'mutable', 'translation': {}, 'py_class': 'String', 'method_s': {}, 'optional': {}, 'required': {}, 'recommended': {}}]

M start_time_ConceptFindClass = datetime.datetime.now()
res_EqualClass = dkb.find(query_EqualClass, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state', 'mutability'], ignore_discarded = False)
print(res_EqualClass)
end_time_ConceptFindClass = datetime.datetime.now()
print("Time it takes to find a concept", TestCon, "by class and print only a few parameters:",
      str(end_time_ConceptFindClass - start_time_ConceptFindClass))
-----
61         if 'content_type' in r.headers and r.headers['content-type'] == 'application/json':
62             msg = json.loads(r.text)
--> 63             handle_server_exception(r.status_code, msg)
64         return msg
65

Exception: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final/EN">
<title>500 Internal Server Error</title>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.</p>

Find by methods
M query_EqualMethod = ('==', 'methods', {})
print('Query with methods', query_EqualMethod)
# C9P: 'QuitMethods'
query_EqualMethodC9P = ('==', 'methods', {'OneMethod': 'String'})
print('Query with description:', query_EqualMethodC9P)
#print(dkb.find( ('==', 'pid', 'c9p'), pid_only = False ))

Query with methods: ('==', 'methods', {})
[{'name': 'QuitMethods', 'prefix': 'Context_ConceptCreation', 'pid': 'ex3k8:Context_ConceptCreation:18:QuitMethods', 'specialises': 'ex3k8:lib1:Concept', 'instance_of': 'ex3k8:lib1:Concept', 'description': '', 'state': 'New', 'timestamp': datetime.datetime(2021, 6, 16, 11, 1, 6, 934688), 'mutability': 'mutable', 'translation': {}, 'py_class': None, 'methods': {'OneMethod': 'String'}, 'optional': {}, 'required': {}, 'recommended': {}}]

M start_time_ConceptFindMethod = datetime.datetime.now()
res_EqualMethod = dkb.find(query_EqualMethod, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state', 'mutability'], ignore_discarded = False)
print(res_EqualMethod)
end_time_ConceptFindMethod = datetime.datetime.now()
print("Time it takes to find a concept", TestCon, "by method and print only a few parameters:",
      str(end_time_ConceptFindMethod - start_time_ConceptFindMethod))
-----
242         ret = find(self.site, self.session_id, query, pid_only, only_these, ignore_discarded)
243         if not self.FIND_ONCE:
244             for i in ret:

TypeError: ('unhashable type: 'dict'', [])

Find by timestamp
M # Convert 'TestCon' timestamp: 'timestamp': datetime.datetime(2021, 6, 16, 10, 38, 4, 751172)
timestamp_string = datetime.datetime(2021, 6, 16, 10, 38, 4, 751172)
print(str(timestamp_string))
2021-06-16 10:38:04.751172

M query_EqualTWP = ('==', 'timestamp', '2021-06-16T10:38:04.751172')
#query_EqualTWP = ('==', 'timestamp', 'timestamp_string')
query_EqualTWP = ('==', 'timestamp', datetime.datetime(2021, 6, 16, 10, 38, 4, 751172))
print('Query with timestamp:', query_EqualTWP)

Query with timestamp: ('==', 'timestamp', '2021-06-16T10:38:04.751172')

M start_time_ConceptFindTWP = datetime.datetime.now()
res_EqualTWP = dkb.find(query_EqualTWP, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state', 'mutability'], ignore_discarded = False)
print(res_EqualTWP)
end_time_ConceptFindTWP = datetime.datetime.now()
print("Time it takes to find a concept", TestCon, "by timestamp and print only a few parameters:",
      str(end_time_ConceptFindTWP - start_time_ConceptFindTWP))
[]

Time it takes to find a concept, ex3k8:Context_ConceptCreation:0:TestCon by timestamp and print only a few parameters: 0:0
0:00.018866

```

(c) Py_class, methods and timestamp

Figure A.3: DKB: Find concept issues (cont.)

Concepts creation: lacking default values Moreover, no default values (such as Boolean = True) can be specified during the concept attribute specification.

Find concepts So far, the find() functionality (to query data from the database) has been rather limited in regards to lookup parameters and operators:

- For concepts, solely *PID* and *prefix* worked.
- Moreover, equality ('==') checks have been implemented but no inequality or range selection

Now, the following parameters and operators are in place:

- For concepts: in addition to the above two parameters, *description*, *state*, *mutability*, *timestamp*, *translation*, *method* and *py_class* are also working.
- Operators: an *inequality* ('!=') and various *range* operators ('<', '<=' or '<=', '>', '>=' or '>=') have been added.

A.3.3 User Manual update

Find concepts The general User Manual `find()` chapter has been extended through sections

- “Available property search criteria and expected values” and
- “Range sort: selection according to ASCII sort order”,

which explain the available input parameters and expected values as well as the way the range selection works for ‘string’ values (please refer to Appendix C.2).

A.4 Instance functions

Test scenario	Expected outcome	Test result
<p>Creation of a new instances¹⁸: on DKBService object <code>dkb</code>, execute the <code>new_instance('concept_name', 'inst_name')</code> function with the <i>name of the concept</i>, of which an instance called <i>inst_name</i> should be created. Additionally, create instances which contain at least one the following parameters each time:</p> <ul style="list-style-type: none"> • specialises • mutability • required • recommended • optional • translation • description • methods • py_class. <p>Thereafter verify the created instances by printing the results of <code>dkb.context_status()['instances']</code>.</p>	<p>All created instances should be listed under context status ‘instances’.</p>	<p>Failure</p>

¹⁸“The function `new_instance()` creates an instance of a concept, by specifying the concept name, and filling attributes. In particular, required attributes are mandatory, but they can be filled with `None`, and later be updated.” (Taken from DKB User Manual [75])

<p>Get instance¹⁴ On DKB-Service object <code>dkb</code> execute the <code>get('inst_name' OR 'inst_PID')</code> function with both</p> <ul style="list-style-type: none"> • Instance name <i>inst_name</i> • Instance PID <i>inst_PID</i> <p>to retrieve the specified instance. Moreover, test the two optional parameters (separately and together):</p> <ul style="list-style-type: none"> • 'only_these' (empty¹⁹ OR selection of <ul style="list-style-type: none"> - specialises - mutability - required - recommended - optional - translation - description - methods - py_class. • 'ignore_discarded' (Boolean) <p>Additionally, check the <code>get('instance')</code> function on instances in different states (same state as context they belong to):</p> <ul style="list-style-type: none"> • 'active' • 'new' • 'frozen' • 'deprecated' • 'discarded' 	<p>In any of the described cases, the according instance together with its specified parameter(s) should be returned.</p>	<p>Partial failure</p>
<p>Find instance¹⁶ Again, on DKB-Service object <code>dkb</code> execute the <code>find('query')</code> function. The <i>query</i> should look as follows:</p>	<p>In any of the described cases (and with any parameter selection and combination), the matching instance(s) together with its/ their</p>	<p>Partial failure</p>

¹⁹'only_these' = [] solely verifies that the instance exists

<p><code>(operator, search_criteria, search_value)</code>, where <i>operator</i> can take <code>'=='</code>, <code>'isa'</code> and <code>'isa_exactly'</code> values and the <i>search_criteria</i> can be anything from <code>'PID'</code>, <code>'prefix'</code>, <code>'state'</code>, <code>'mutability'</code> and <code>'timestamp'</code>. Moreover, test the three optional parameters (separately and together):</p> <ul style="list-style-type: none"> • <code>'pid_only'</code> (Boolean) • <code>'only_these'</code> (selection of <ul style="list-style-type: none"> - specialises - mutability - required - recommended - optional - translation - description - methods - <code>py_class</code>. • <code>'ignore_discarded'</code> (Boolean) <p>Additionally, check <code>find('query')</code> on instances in different states (state alike context they belong to):</p> <ul style="list-style-type: none"> • <code>'active'</code> • <code>'new'</code> • <code>'frozen'</code> • <code>'deprecated'</code> • <code>'discarded'</code> <p>Finally, test also more complex queries, concatenated either by <code>'AND'</code> or <code>'OR'</code> (e.g., <code>query = ('AND', (cond1, cond2))</code> with <code>cond1 = cond2 = (operator, search_criteria, search_value)</code>).</p>	<p>specified parameter(s) should be returned.</p>
---	---

Table A.4: DKB instance functions

A.4.1 Application bugs

General find() and get () instance issue after DKB server restart Once the DKB server had been restarted via the command prompt¹⁷, it was no longer possible to access the previously (during the former DKB server session) created and meanwhile saved concepts and instances with the get () and find() functions: for instances, get () and find() resulted in various `TypeError`s, as illustrated in figure A.4. This error has meanwhile been corrected: previously created instances are now also selectable after a DKB server restart.

Re-logout after disconnecting from the server

The issue of no longer being able to get() and find() previously created concepts and instances happens once the server was shutdown and started again at a later point in time.
Closing the connection to the server and logging in a again with the same user (or a different user) at a later point in time while the same server session as before is still running, does not result in the 500 Internal Server Error.

```

import DKBlib as DKB
import datetime

startTime_relogin = datetime.datetime.now()
dkb = DKB.Login('ex3KB', 'Testuser')
endTime_relogin = datetime.datetime.now()
print("Time it takes to login to DKB:", str(endTime_relogin - startTime_relogin))

Time it takes to login to DKB: 0:00:16.786605

# dkb.enter(context_TestConcepts, 'W')
dkb.enter('context_ConceptCreation', 'W')
print(dkb.status()['current_context'])

TestCon = 'ex3KB:Context_ConceptCreation:0:TestCon'
tci = 'ex3KB:Context_ConceptCreation:18:tci'

Context_ConceptCreation

# start_time_InstanceGetNameTci_relogin = datetime.datetime.now()
print("Get() instance by name:\n", dkb.get('tci')) # By name
end_time_InstanceGetNameTci_relogin = datetime.datetime.now()
print("\nTime it takes to get an instance, ", tci + ", by name:",
      str(end_time_InstanceGetNameTci_relogin - start_time_InstanceGetNameTci_relogin))

start_time_InstanceGetPIDTci_relogin = datetime.datetime.now()
print("\nGet() instance by PID:\n", dkb.get(tci)) # By PID
end_time_InstanceGetPIDTci_relogin = datetime.datetime.now()
print("\nTime it takes to get an instance, ", tci + ", by PID:",
      str(end_time_InstanceGetPIDTci_relogin - start_time_InstanceGetPIDTci_relogin))

--> 229         ret = get(self.site, self.session_id, identity, only_these, ignore_discarded)
230         if not self.raw_data:
231             if ('timestamp' in ret.keys()):

c:\users\patty\github\dare_kb\src\client\DKBlib\csbridge.py in wrapper(*args, **kwargs)
37         return func(*args, **kwargs)
38     except DKBException as e:
--> 39         e.coerce()
40     return wrapper

c:\users\patty\github\dare_kb\src\client\DKBlib\common\cs_exception.py in coerce(self)
74
75     def coerce(self):
--> 76         raise find_exception(self.error_code)(self.message, *self.extra)
77
78

TypeError: ("get_instance() missing 1 required positional argument: 'name'", [])

```

Figure A.4: DKB: accessing previously created instance after server restart

Find instance issues Previously, find() instance did not work for the search criteria name, state and mutability - solely PID, prefix and timestamp returned the expected outcome. The other three parameter returned a `TypeError` stating “`get_instance() missing 1 required positional argument: 'name', []`” (also exemplified in figure-A.5. After debugging and correcting the error, find() now returns the specified concepts and instances without any such error message.

```

4.3 Find() instance
The find() function allows the user to query the knowledge base about any instances (concepts included) that exist. It takes a query and three optional arguments: pid_only, only_these and ignore_discarded. These last two arguments work the same way as for get. The query supports different operators:
• AND and OR query
• The tuple condition of the form: (operation, property, value) with the supported operation: '==', '<=', '>=', '<', '>'
• The tuple condition of the form: (operation, concept) with the supported operators:
  • 'is' which is true if an instance is of the concept or of a concept its concept specialises - a transitive relationship
  • 'is_a_exactly' which is satisfied if an instance is of the specified concept

Type of arguments and return:
find()
q: Query pid_only boolean = True
  only_these List[db] = None
  ignore_discarded boolean = False
  → Union[Link(PID), List[Attributes]]

(Taken from the DKB User Manual)

M dbk.enter(context='testConcepts', 'w')
  wdbk.enter('context_conceptcreation', 'w')
  dbk.status()['current_context']
  'context_conceptcreation'

Find by name
M queryInst_EqualName = ('==', 'name', 'tci')
  print("Query with name:", queryInst_EqualName)
  Query with name: ('==', 'name', 'tci')

M start_time_instanceIndName = datetime.datetime.now()
  resInst_EqualName = dbk.find(queryInst_EqualName, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state', 'mutability'], ignore_discarded = False)
  print(resInst_EqualName)
  end_time_instanceIndName = datetime.datetime.now()
  print("Time it takes to find an instance:", tci, "by name and print only a few parameters:",
        str(end_time_instanceIndName - start_time_instanceIndName))
--> 242 ret = find(self.site, self.session_id, query, pid_only, only_these, ignore_discarded)
  243     if not self.raw_data:
  244         for i in ret:
  TypeError: ("get_instance() missing 1 required positional argument: 'name'", [])

```

(a) Name

```

Find by state
M queryInst_EqualState = ('==', 'state', 'new')
  print("Query with state:", queryInst_EqualState)
  Query with state: ('==', 'state', 'new')

M start_time_instanceIndState = datetime.datetime.now()
  resInst_EqualState = dbk.find(queryInst_EqualState, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state', 'mutability'], ignore_discarded = False)
  print(resInst_EqualState)
  end_time_instanceIndState = datetime.datetime.now()
  print("Time it takes to find an instance:", tci, "by state and print only a few parameters:",
        str(end_time_instanceIndState - start_time_instanceIndState))
--> 242 ret = find(self.site, self.session_id, query, pid_only, only_these, ignore_discarded)
  243     if not self.raw_data:
  244         for i in ret:
  TypeError: ("get_instance() missing 1 required positional argument: 'name'", [])

TypeError: ("get_instance() missing 1 required positional argument: 'name'", [])

Find by mutability
M queryInst_EqualMuta = ('==', 'mutability', 'mutable')
  print("Query with mutability:", queryInst_EqualMuta)
  Query with mutability: ('==', 'mutability', 'mutable')

M start_time_instanceIndMuta = datetime.datetime.now()
  resInst_EqualMuta = dbk.find(queryInst_EqualMuta, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state', 'mutability'], ignore_discarded = False)
  print(resInst_EqualMuta)
  end_time_instanceIndMuta = datetime.datetime.now()
  print("Time it takes to find an instance:", tci, "by mutability and print only a few parameters:",
        str(end_time_instanceIndMuta - start_time_instanceIndMuta))
--> 242 ret = find(self.site, self.session_id, query, pid_only, only_these, ignore_discarded)
  243     if not self.raw_data:
  244         for i in ret:
  TypeError: ("get_instance() missing 1 required positional argument: 'name'", [])

TypeError: ("get_instance() missing 1 required positional argument: 'name'", [])

```

(b) State and mutability

Figure A.5: DKB: find instance issues

A.4.2 Functionality deficiency

Find instances So far, the `find()` functionality (to query data from the database) has been rather limited in regards to lookup parameters and operators:

- For instances only PID, prefix and timestamp returned results.
- Moreover, equality ('==') checks have been implemented but no inequality or range selection

for instances Now, the following parameters and operators are in place:

- For instances: besides the above three parameters, one can now find instances via their state and mutability.
- Operators: an *inequality* ('!=') and various *range* operators ('<', '<=' or '<=', '>', '>=' or '>=') have been added.

A.4.3 User Manual update

Find instances The general User Manual `find()` chapter has been extended through sections

- “Available property search criteria and expected values” and
- “Range sort: selection according to ASCII sort order”,

which explain the available input parameters and expected values as well as the way the range selection works for ‘string’ values (please refer to Appendix C.2).

A.5 Performance

To measure each of the above described *site*, *context*, *concept* and *instance* functions’ **execution time**, the Python’s built-in *datetime* library has been utilized. For this reason, the *datetime* functionality has been called before and after executing each function to track the execution performance, just as figure A.6 exemplifies.

1. Site functions

1.3 DKB closure

The close function records in the storage all of the information created. It terminates interaction with the DKB started at login. (Taken from the [DKB User Manual](#))

```

▶ startTime_close = datetime.datetime.now()
  dkb.close()
  endTime_close = datetime.datetime.now()
  print("Time it takes to close the DKB and save all changes:", str(endTime_close - startTime_close) )
Time it takes to close the DKB and save all changes: 0:01:11.079543

```

Figure A.6: DKB login performance

A.5.1 Site functions

The performance tests revealed, that the login and logoff time increased with the amount of data stored in the database. As figures A.7 illustrate, both the login and logoff time took around 75 seconds for around 390 MB of data.

1. Site functions

1.3 DKB closure

The close function records in the storage all of the information created. It terminates interaction with the DKB started at login. (Taken from the [DKB User Manual](#))

```

M | startTime_close = datetime.datetime.now()
   | dkb.close()
   | endTime_close = datetime.datetime.now()
   | print("Time it takes to close the DKB and save all changes:", str(endTime_close - startTime_close) )
   |
   | Time it takes to close the DKB and save all changes: 0:01:11.079543

```

(a) DKB login performance

```

M | startTime_close_relogin = datetime.datetime.now()
   | dkb.close()
   | endTime_close_relogin = datetime.datetime.now()
   | print("Time it takes to close the DKB and save all changes:", str(endTime_close_relogin - startTime_close_relogin) )
   |
   | Time it takes to close the DKB and save all changes: 0:01:19.975815

```

(b) DKB logoff performance

Figure A.7: DKB site function performance

A.5.2 Concept and instance functions

As figure A.8(a) presents, `get()` with PID takes around 13 times longer than with name, which – in any case – still is less than one second.

Test that `get()` works both with name and PID.

```

M | start_time_InstanceGetNameTC1 = datetime.datetime.now()
   | print("Get() instance by name:\n", dkb.get('tc1') ) # By name
   | end_time_InstanceGetNameTC1 = datetime.datetime.now()
   | print("\nTime it takes to get an instance," tc1 + ", by name:",
   |       str(end_time_InstanceGetNameTC1 - start_time_InstanceGetNameTC1) )
   |
   | start_time_InstanceGetPIDTC1 = datetime.datetime.now()
   | print("\nGet() instance by PID:\n", dkb.get(tc1) ) # By PID
   | end_time_InstanceGetPIDTC1 = datetime.datetime.now()
   | print("\nTime it takes to get an instance," tc1 + ", by PID:",
   |       str(end_time_InstanceGetPIDTC1 - start_time_InstanceGetPIDTC1) )
   |
   | Get() instance by name:
   | {'instance_of': 'ex3KB:Context_ConceptCreation:0:TestCon', 'mutability': 'mutable', 'name': 'tc1', 'pid': 'ex3KB:Context_C
   | onceptCreation:18:tc1', 'prefix': 'Context_ConceptCreation', 'state': 'new', 'timestamp': datetime.datetime(2021, 6, 16, 2
   | 2, 46, 21, 681378), 'x': 'a_string', 'y': 5}
   |
   | Time it takes to get an instance, ex3KB:Context_ConceptCreation:18:tc1, by name: 0:00:00.037954
   |
   | Get() instance by PID:
   | {'instance_of': 'ex3KB:Context_ConceptCreation:0:TestCon', 'mutability': 'mutable', 'name': 'tc1', 'pid': 'ex3KB:Context_C
   | onceptCreation:18:tc1', 'prefix': 'Context_ConceptCreation', 'state': 'new', 'timestamp': datetime.datetime(2021, 6, 16, 2
   | 2, 46, 21, 681378), 'x': 'a_string', 'y': 5}
   |
   | Time it takes to get an instance, ex3KB:Context_ConceptCreation:18:tc1, by PID: 0:00:00.472142
   |
M | start_time_InstanceGetNameTC2 = datetime.datetime.now()
   | print("Get() instance by name:\n", dkb.get('tc2') ) # By name
   | end_time_InstanceGetNameTC2 = datetime.datetime.now()
   | print("\nTime it takes to get an instance," tc2 + ", by name:",
   |       str(end_time_InstanceGetNameTC2 - start_time_InstanceGetNameTC2) )
   |
   | start_time_InstanceGetPIDTC2 = datetime.datetime.now()
   | print("\nGet() instance by PID:\n", dkb.get(tc2) ) # By PID
   | end_time_InstanceGetPIDTC2 = datetime.datetime.now()
   | print("\nTime it takes to get an instance," tc2 + ", by PID:",
   |       str(end_time_InstanceGetPIDTC2 - start_time_InstanceGetPIDTC2) )
   |
   | Get() instance by name:
   | {'instance_of': 'ex3KB:Context_ConceptCreation:0:TestCon', 'mutability': 'mutable', 'name': 'tc2', 'pid': 'ex3KB:Context_C
   | onceptCreation:19:tc2', 'prefix': 'Context_ConceptCreation', 'state': 'new', 'timestamp': datetime.datetime(2021, 6, 16, 2
   | 2, 48, 0, 365109), 'x': 'b_string', 'y': 4}
   |
   | Time it takes to get an instance, ex3KB:Context_ConceptCreation:19:tc2, by name: 0:00:00.037417
   |
   | Get() instance by PID:
   | {'instance_of': 'ex3KB:Context_ConceptCreation:0:TestCon', 'mutability': 'mutable', 'name': 'tc2', 'pid': 'ex3KB:Context_C
   | onceptCreation:19:tc2', 'prefix': 'Context_ConceptCreation', 'state': 'new', 'timestamp': datetime.datetime(2021, 6, 16, 2
   | 2, 48, 0, 365109), 'x': 'b_string', 'y': 4}
   |
   | Time it takes to get an instance, ex3KB:Context_ConceptCreation:19:tc2, by PID: 0:00:00.535134

```

(a) Get () with PID vs. name

Figure A.8: DKB: `get()` function performance

Furthermore the tests showed, that the execution time for `get ()` has risen with the size of the concept (refer to figure A.9(b)).

Chosen example concept and instance:

- 'Concept_prefix95': 'ex3KB:Context_prefix11:95:Concept_prefix95'
- 'aConcept_prefix18_11_18_65': 'ex3KB:Context_prefix11:1965:aConcept_prefix18_11_18_65'

```

M getLargeCon_currSession_pid = 'ex3KB:Context_prefix11:95:Concept_prefix95'
getLargeCon_currSession_name = 'Concept_prefix95'

start_time_ConceptGetPref11 = datetime.datetime.now()
print("\nGet() concept:\n", dkb.get(getLargeCon_currSession_pid,
    only_these = ['pid', 'name', 'mutability', 'state', 'timestamp', 'required'],
    ignore_discarded = True) )

end_time_ConceptGetPref11 = datetime.datetime.now()
print("\nTime it takes to get a large concept with only a few selected parameters,", getLargeCon_currSession_pid +
    ", by PID:", str(end_time_ConceptGetPref11 - start_time_ConceptGetPref11) )

Get() concept:
{'mutability': 'mutable', 'name': 'Concept_prefix95', 'pid': 'ex3KB:Context_prefix11:95:Concept_prefix95', 'required': {'attrib0': 'Integer', 'attrib1': 'String', 'attrib10': 'Integer', 'attrib11': 'String', 'attrib12': 'Integer', 'attrib13': 'String', 'attrib14': 'Integer', 'attrib15': 'String', 'attrib16': 'Integer', 'attrib17': 'String', 'attrib18': 'Integer', 'attrib19': 'String', 'attrib2': 'Integer', 'attrib20': 'Integer', 'attrib21': 'String', 'attrib22': 'Integer', 'attrib23': 'String', 'attrib24': 'Integer', 'attrib25': 'String', 'attrib26': 'Integer', 'attrib27': 'String', 'attrib28': 'Integer', 'attrib29': 'String', 'attrib3': 'String', 'attrib30': 'Integer', 'attrib31': 'String', 'attrib32': 'Integer', 'attrib33': 'String', 'attrib34': 'Integer', 'attrib35': 'String', 'attrib36': 'Integer', 'attrib37': 'String', 'attrib38': 'Integer', 'attrib39': 'String', 'attrib4': 'Integer', 'attrib40': 'Integer', 'attrib41': 'String', 'attrib42': 'Integer', 'attrib43': 'String', 'attrib44': 'Integer', 'attrib45': 'String', 'attrib46': 'Integer', 'attrib47': 'String', 'attrib48': 'Integer', 'attrib49': 'String', 'attrib5': 'String', 'attrib50': 'Integer', 'attrib51': 'String', 'attrib52': 'Integer', 'attrib53': 'String', 'attrib54': 'Integer', 'attrib55': 'String', 'attrib56': 'Integer', 'attrib57': 'String', 'attrib58': 'Integer', 'attrib59': 'String', 'attrib6': 'Integer', 'attrib60': 'Integer', 'attrib61': 'String', 'attrib62': 'Integer', 'attrib63': 'String', 'attrib64': 'Integer', 'attrib65': 'String', 'attrib66': 'Integer', 'attrib67': 'String', 'attrib68': 'Integer', 'attrib69': 'String', 'attrib7': 'String', 'attrib70': 'Integer', 'attrib71': 'String', 'attrib72': 'Integer', 'attrib73': 'String', 'attrib74': 'Integer', 'attrib75': 'String', 'attrib76': 'Integer', 'attrib77': 'String', 'attrib78': 'Integer', 'attrib79': 'String', 'attrib8': 'Integer', 'attrib80': 'Integer', 'attrib81': 'String', 'attrib82': 'Integer', 'attrib83': 'String', 'attrib84': 'Integer', 'attrib85': 'String', 'attrib86': 'Integer', 'attrib87': 'String', 'attrib88': 'Integer', 'attrib89': 'String', 'attrib9': 'String', 'attrib90': 'Integer', 'attrib91': 'String', 'attrib92': 'Integer', 'attrib93': 'String', 'attrib94': 'Integer'}, 'state': 'new', 'timestamp': datetime.datetime(2021, 6, 18, 13, 1, 9, 453728)}

Time it takes to get a large concept with only a few selected parameters, ex3KB:Context_prefix11:95:Concept_prefix95, by PID: 0:00:04.508151

```

(b) Get () with large concepts

Figure A.8: DKB: `get ()` function performance (cont.)

Lastly, the system analysis uncovered that `find ()` with prefix took around 30 times longer than with PID, as figure A.9 demonstrates.

```

M start_time_InstanceFindPID_OT_ID3 = datetime.datetime.now()
resInst_EqualPID_OT_ID3 = dkb.find(queryInst_EqualPID, only_these = ['description', 'translation', 'py_class', 'methods',
    'prefix', 'optional', 'required', 'x', 'y',
    'recommended', 'timestamp', 'name'],
    ignore_discarded = False)
print(resInst_EqualPID_OT_ID3)
end_time_InstanceFindPID_OT_ID3 = datetime.datetime.now()
print("\nTime it takes to find an instance,", tc1 + ", by PID and print a few dedicated parameters:",
    str(end_time_InstanceFindPID_OT_ID3 - start_time_InstanceFindPID_OT_ID3) )

[{'prefix': 'Context_ConceptCreation', 'x': 'a_string', 'y': 5, 'timestamp': datetime.datetime(2021, 6, 16, 22, 46, 21, 681378), 'name': 'tc1'}]

Time it takes to find an instance, ex3KB:Context_ConceptCreation:18:tc1, by PID and print a few dedicated parameters: 0:00:00.563913

```

(c) Find () with PID

Figure A.9: DKB: `find ()` function performance

Find by prefix:

```

M queryInst_EqualPrefix = ('==', 'prefix', 'Context_ConceptCreation')
print("Query with prefix:", queryInst_EqualPrefix)

```

```

Query with prefix: ('==', 'prefix', 'Context_ConceptCreation')

```

```

M start_time_InstanceFindPrefix = datetime.datetime.now()
resInst_EqualPrefix = dkb.find(queryInst_EqualPrefix, only_these = ['pid', 'name', 'specialises', 'instance_of', 'state',
                                                                    'mutability'], ignore_discarded = False)
print(resInst_EqualPrefix)
end_time_InstanceFindPrefix = datetime.datetime.now()
print("nTime it takes to find an instance, tc1, by prefix and print only a few parameters:",
      str(end_time_InstanceFindPrefix - start_time_InstanceFindPrefix))

```

```

[{'pid': 'ex3KB:Context_ConceptCreation:20:tc3', 'name': 'tc3', 'instance_of': 'ex3KB:Context_ConceptCreation:0:TestCon',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:17:Instance_for_opt_2', 'name': 'Instance_for_opt_2',
  'instance_of': 'ex3KB:Context_ConceptCreation:7:ConceptWithOptionalAtt', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:13:Instance_for_att', 'name': 'Instance_for_att', 'instance_of': 'ex3KB:Context_ConceptCreation:5:FirstWithAttribute',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:10:CWithMethods', 'name': 'CWithMethods',
  'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:16:Instance_for_opt', 'name': 'Instance_for_opt', 'instance_of': 'ex3KB:Context_ConceptCreation:7:ConceptWithOptionalAtt',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:10:CWithMethods', 'name': 'CWithMethods',
  'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:11:CWithPy_class', 'name': 'CWithPy_class', 'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:19:tc2', 'name': 'tc2',
  'instance_of': 'ex3KB:Context_ConceptCreation:0:TestCon', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:23:Instance_for_att_3', 'name': 'Instance_for_att_3', 'instance_of': 'ex3KB:Context_ConceptCreation:6:SecWithAtt',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:3:ASecondsSpe', 'name': 'ASecondsSpe',
  'specialises': 'ex3KB:Context_ConceptCreation:1:FirstOne', 'instance_of': 'ex3KB:kb:1:Concept', 'state': 'new',
  'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:5:FirstWithAttribute', 'name': 'FirstWithAttribute',
  'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:6:SecWithAtt', 'name': 'SecWithAtt', 'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:4:FirstMutability', 'name': 'FirstMutability',
  'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:18:tc1', 'name': 'tc1', 'instance_of': 'ex3KB:Context_ConceptCreation:0:TestCon',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:1:FirstOne', 'name': 'FirstOne',
  'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:22:Instance_for_mutability', 'name': 'Instance_for_mutability', 'instance_of': 'ex3KB:Context_ConceptCreation:4:FirstMutability',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:15:Instance_for_att_missing',
  'name': 'Instance_for_att_missing', 'instance_of': 'ex3KB:Context_ConceptCreation:6:SecWithAtt',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:9:CWithDescription', 'name': 'CWithDescription',
  'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:0:TestCon', 'name': 'TestCon', 'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:26:Instance_for_method',
  'name': 'Instance_for_method', 'instance_of': 'ex3KB:Context_ConceptCreation:10:CWithMethods', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:14:Instance_for_att_2', 'name': 'Instance_for_att_2', 'instance_of': 'ex3KB:Context_ConceptCreation:5:FirstWithAttribute',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:25:Instance_for_translation',
  'name': 'Instance_for_translation', 'instance_of': 'ex3KB:Context_ConceptCreation:8:ConceptWithTranslation',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:12:First_Instance',
  'name': 'First_Instance', 'instance_of': 'ex3KB:Context_ConceptCreation:1:FirstOne', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:8:ConceptWithTranslation', 'name': 'ConceptWithTranslation', 'specialises': 'ex3KB:kb:1:Concept',
  'instance_of': 'ex3KB:kb:1:Concept', 'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:7:ConceptWithOptionalAtt',
  'name': 'ConceptWithOptionalAtt', 'specialises': 'ex3KB:kb:1:Concept', 'instance_of': 'ex3KB:kb:1:Concept',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:2:ASpeOfFirstOne',
  'name': 'ASpeOfFirstOne', 'specialises': 'ex3KB:Context_ConceptCreation:1:FirstOne', 'instance_of': 'ex3KB:kb:1:Concept',
  'state': 'new', 'mutability': 'mutable'}, {'pid': 'ex3KB:Context_ConceptCreation:21:tc4', 'name': 'tc4',
  'instance_of': 'ex3KB:Context_ConceptCreation:0:TestCon', 'state': 'new', 'mutability': 'mutable'},
 {'pid': 'ex3KB:Context_ConceptCreation:24:Instance_for_PyClass', 'name': 'Instance_for_PyClass', 'instance_of': 'ex3KB:Context_ConceptCreation:11:CWithPy_class',
  'state': 'new', 'mutability': 'mutable'}]

```

```

Time it takes to find an instance, ex3KB:Context_ConceptCreation:18:tc1 by prefix and print only a few parameters: 0:00:15.234900

```

(b) Find() with prefix

Figure A.9: DKB: find() function performance (cont.)

Appendix B

Application bug fixes

The following chapter includes all the code corrections for the observed application bugs (to be precise, the ‘instance creation’, ‘concept and instance lookup (find)’ as well as the ‘concept and instance data access (find and get) after a DKB server restart’ issue). The below code extracts have been taken from the according Gitlab files [storage.py](#) [57] and [DareKB.py](#) [52].

B.1 Instance creation issue

‘Storage.py’ file, line 518 ff.:

```
def on_new_instance(self, instance: Instance, session):
    def create_owl_instance(ins, session):
        with self._namespace as ns:
            owl_instance = self._get_owl_entry_by_pid(ins.cls)
                (self._normalise_name(ins.pid))
            # owl_instance =
                self._namespace.Instance(self._normalise_name(ins.pid)) ##
            Alternative
            owl_instance.name_dkb = ins.name
            owl_instance.prefix = ins.prefix
            owl_instance.pid = ins.pid
            owl_instance.state = ins.state
            owl_instance.mutability = ins.mutability
            owl_instance.timestamp = ins.timestamp
            owl_instance.predecessor = ins.predecessor
            owl_instance.successor = ins.successor
```

```

owl_instance.instance_last_updated = 1
# owl_instance.session = self._ontology.search(pid = session)
## ERROR, causing instance creation issue
owl_instance.session = self._ontology.search(pid = session)[0]
## Correction, take list element
for k, v in ins.extras.items():
    owl_k = ns[k] # FIXME: what if name conflict
    if not owl_k: # If the property doesn't exist (not owl_k),
        ignore it. TODO: Improve this behaviour
        continue
    if v is None: # If the value is null ('v == None', thus
        'not v'), do not store it because owlready doesn't allow
        to store a null value. This behaviour is needed for
        future update (e.g., circular dependencies), so it
        should be no harm
        continue
    if issubclass_owlready(owl_k, owl.ObjectProperty):
        if (v == ''):
            ## to be set later
            owl_instance.__setattr__(k, ns._nil)
            comment[owl_instance, owl_k, ns._nil] = ['pid']
            continue
        ## check if v is a pid
        if (not is_pid(v)):
            owl_obj = self._find_pid_given_name(v)
            comment[owl_instance, owl_k, owl_obj] = ['name']
        else:
            owl_obj = self._get_owl_entry_by_pid(v) # v is the
                PID of the Instance
            comment[owl_instance, owl_k, owl_obj] = ['pid']
            owl_instance.__setattr__(k, owl_obj)
        else: # It is a data property
            owl_instance.__setattr__(k, v)
    return owl_instance
owl_instance = create_owl_instance(instance, session)
instance.my_instance = owl_instance

```

‘Storage.py’ file, line 398 ff.:

```
def on_new_concept(self, c: Concept, session) -> None:
    def create_owl_concept_basic(c, session):
        with self._namespace:
            paren_concept = self._get_owl_entry_by_pid(c.sub_class)
            my_concept =
                types.new_class(self._normalise_name(c.identifier),
                    (paren_concept,))

            my_concept.class_description=c.description
            my_concept.class_pid=c.identifier
            my_concept.class_name=c.label
            my_concept.class_prefix=c.prefix
            my_concept.class_state=c.state
            my_concept.class_mutability=c.mutability
            my_concept.class_timestamp = c.timestamp
            my_concept.class_methods = json.dumps(c.methods)
            my_concept.class_translation = json.dumps(c.translation)
            my_concept.class_py_class=c.py_class
            if (session is not None):
                # my_concept.session = self._ontology.search(pid =
                    session) ## ERROR, causing instance creation issue
                my_concept.session = self._ontology.search(pid =
                    session)[0] ## Correction, take list element
            return my_concept
```

B.2 Data access after DKB server restart

B.2.1 Find and get instance

‘Storage.py’ file, line 505 ff.: new `get_instance_by_pid` method

```
### NEW code beginning
## New method to get instance by PID, resolving 'TypeError:
    ("get_instance() missing 1 required positional argument: 'name'", [],)'
## one gets one using the above 'get_instance' method with PID
```

```

def get_instance_by_pid(self, pid: PID):
    # Get instance with PID
    res = self._get_owl_entry_by_pid(pid)
    # If no instance for given PID was found, an 'InstanceNotFoundError'
    # is raised
    if not res:
        raise InstanceNotFoundError(f'{pid}', 'get_instance')
    # Return the instantiated instance
    return self._instantiate_python_instance(res)
### NEW code end

```

'DareKB.py' file, line 531 ff.:

```

### Instance functions

def _load_python_instace_from_storage(self, pid: PID):
    # inst_kb = self._storage.get_instance(pid) ## Error: 'TypeError:
    # ("get_instance() missing 1 required positional argument: 'name"',
    # [],)' when using the 'get_instance' method with PID
    inst_kb = self._storage.get_instance_by_pid(pid) ## Correction: New
    # method 'get_instance_by_pid'
    self._cache_instance(inst_kb)

```

B.2.2 Get concept

Due to timestamp-related problems: 'DareKB.py' file, line 566 ff.:

```

### get and find and update functions
@__require_open_dkb
def get_concept(self, pid: PID, ignore_discarded) -> Concept:
    if (not self._concept_loaded_pid(pid)):
        self._load_python_concept_from_storage(pid)
    # conc = self.pid_concepts[pid] ## OLD code

    ## NEW code beginning: Resolving the get() concept issue (Part 1, due
    ## to AttributeError: 'strftime' is not a language code)
    conc = self.pid_concepts[pid]
    # Retrieved timestamp is a list after the concept has been
    # flushed to the DB

```

```

# Thus, additional check whether timestamp is of type
'IndividualValueList'
if isinstance(conc.timestamp, owlready2.prop.IndividualValueList):
    # In this case: Take list element
    conc.timestamp = conc.timestamp[0]
# Issue does not occur if concept is not loaded from the DB, thus
code as before
else: conc = self.pid_concepts[pid]
## NEW code end

if (conc.state == instance_state_list[4] and not ignore_discarded):
    raise InstanceNotFoundError(pid, 'get')
return conc

```

Due to empty (self) properties: 'storage.py' file, line 259 ff.:

```

def get_concept_by_pid(self, pid: PID) -> Concept:
    owl_concept = self._get_owl_entry_by_pid(pid)
    subclass = owl_concept.is_a[0]
    # assert self.owl_root_concept() in subclass.ancestors(), f"{pid} is
not a regular Concept as its super class is not its first type.
{owl_concept.is_a}" # TODO: Improve the retrieval of super Concept
if subclass == self.owl_root_concept():
    subclass = self.id_for_root_concept()
else:
    subclass = subclass.class_pid

required = {}
recommended = {}
optional = {}
# New lists for storing the 'required', 'recommended' and 'optional'
attributes as DataProperties
requ = {}
recomm = {}
opt = {}

# Only the direct constraints are retrieved here. Indirect ones are
dealt with in MyContext

```

```

with self._ontology as onto:
    # If 'with ... as' variable assignment is not working
    if onto is None:
        onto = self._ontology
    for sup in onto.get_parents_of(owl_concept):
        if isinstance(sup, Restriction):
            ### OLD code beginning
            # if issubclass_owlready(sup.property,
            #   onto.dkb_data_property):
            #     p_name = sup.property.name
            #     p_range = sup.value
            # elif issubclass_owlready(sup.property,
            #   onto.dkb_object_property):
            #     p_name = sup.property.name
            #     p_range = sup.value.pid
            # else:
            #     continue # Unknown. Probably other subclass-of
            #     restrictions, so we ignore them.
            # if sup.type == 26: # exactly
            #     required[p_name] = p_range
            # elif sup.type == 28: # max
            #     if ('optional' in sup.property.comment):
            #         optional[p_name] = p_range
            #     if ('recommended' in sup.property.comment):
            #         recommended[p_name] = p_range
            ### OLD code end

            ### NEW code beginning: Resolving the get() concept issue
            #   (Part 2, due to empty 'self.properties')
            ## Approach similar to instances:
            # First of all check whether attribute is object property
            if issubclass_owlready(sup.property, owl.ObjectProperty):
                # Check for object property
                p_name = sup.property.name
                p_range = sup.value.pid
            # Else it is a data property
            else:

```

```
p_name = sup.property.name
p_range = sup.value

## Check for attribute type
# Required == 26
if sup.type == 26: # exactly
    # Dictionary containing the 'required' values, used for
    # Concept creation
    required[p_name] = p_range
    # A new 'MyDataProperty' object for storing the
    # 'required' information
    requ[p_name] = MyDataProperty(p_name,
        owl_concept.class_pid[0], p_range, True,
        sup.property.comment[0])
# Optional and recommended == 28
elif sup.type == 28: # max
    # Distinguish between 'optional' and 'recommended'
    # attributes ('property.comment' stores information)
    if ('optional' in sup.property.comment):
        # Dictionary containing the 'optional' values, used
        # for Concept creation
        optional[p_name] = p_range
        # A new 'MyDataProperty' object for storing the
        # 'optional' information
        opt[p_name] = MyDataProperty(p_name,
            owl_concept.class_pid[0], p_range, True,
            sup.property.comment[0])
    if ('recommended' in sup.property.comment):
        # Dictionary containing the 'recommended' values,
        # used for Concept creation
        recommended[p_name] = p_range
        # A new 'MyDataProperty' object for storing the
        # 'recommended' information
        recomm[p_name] = MyDataProperty(p_name,
            owl_concept.class_pid[0], p_range, True,
            sup.property.comment[0])
## Move 'else' (belonging to restriction check: 'if
```

```

        isinstance(sup, Restriction)') to appropriate part
    else:
        continue # Unknown. Probably other subclass-of
                restrictions, so we ignore them.
    ### NEW code end

translation = owl_concept.class_translation
if translation:
    translation = json.loads(translation[0]) ## Type <class
        'owlready2.prop.IndividualValueList'>
else:
    translation = {}

methods = owl_concept.class_methods
if methods:
    methods = json.loads(methods[0])          ## Type <class
        'owlready2.prop.IndividualValueList'>
else:
    methods = {}

concept = Concept(owl_concept.class_prefix[0],
                 owl_concept.class_name[0], subclass,
                 owl_concept.class_mutability[0], owl_concept.class_state[0],
                 required = required,
                 recommended = recommended,
                 optional = optional,
                 translation = translation,
                 identifier = owl_concept.class_pid[0],
                 timestamp = owl_concept.class_timestamp,
                 methods = methods,
                 py_class = owl_concept.class_py_class)
concept.my_concept = owl_concept

### NEW code beginning
# Dictionary of dictionaries containing 'direct' and 'indirect'
# DataProperties
concept_properties = {

```



```

        'direct': {
            'required': requ,
            'recommended': recomm,
            'optional': opt,
        },
        'indirect': {
            'required': {},
            'recommended': {},
            'optional': {},
        }
    }

# Adding the above created DataProperty dictionary to the previously
# created concept
concept.add_properties(concept_properties)
### NEW code end

return concept

```

B.3 Find issue

‘Storage.py’ file, line 71 ff.:

```

self._proj_instance = {
    'pid': 'pid',
    'prefix': 'prefix',
    'name': 'name_dkb',
    'state': 'state',
    'mutability': 'mutability',
    # 'timestandamp': 'timestamp', ## Typo, resulting in empty find()
    # results for concept timestamp search
    'timestamp': 'timestamp', ## Correction
}

```

‘Storage.py’ file, line 635 ff.:

```

def find_tuple(self, t, type):
    op = t[0]
    prop = t[1]

```

```
value = t[2]
les_pids = []

if type == MixType.CONCEPT:
    prop = self._proj_concept[prop]
    pids = self._ontology.search(** {prop : value})
    # pids = self._ontology.search(** {prop : str(value)})
    for pid in pids:
        ### OLD code beginning
        # if (pid.class_pid is not None):
        #     les_pids.append(pid.class_pid) ## Causing TypeError:
        #         unhashable type: 'IndividualValueList
        ### OLD code end

        ### NEW code beginning
        # Check if '.class_pid' returns a list (contains elements)
        if (pid.class_pid[0] is not None):
            # If so, add the element the final list 'les_pids'
            les_pids.append(pid.class_pid[0])
        # Else check whether '.class_pid' contains an entry
        elif (pid.class_pid is not None):
            # If so, add it to the final list 'les_pids'
            les_pids.append(pid.class_pid)
        ### NEW code end
```

Appendix C

User Manual updates

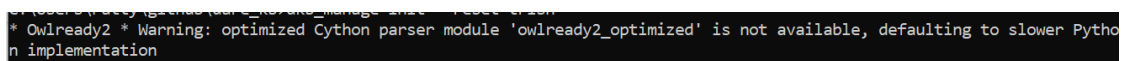
All the DKB tests and discussions have also revealed some necessary [User Manual](#) updates. The following lists the sections and (paragraph) adjustments which have been added and performed to the [User Manual](#) [75].

C.1 Running the DKB

For example, the “[How to run the DKB \(only in self-managed use\)](#)” chapter was extended by two sections to address potential installation and DKB run issues (sections and their content following below).

C.1.1 Owlready2 warning message [75]

“Depending on one’s own Owlready2 version and available modules, the following warning message may appear after the DKB server was started: “**Owlready2 Warning: optimized Cython parser module ‘owlready2_optimized’ is not available, defaulting to slower Python implementation**”.



```
Owlready2 * Warning: optimized Cython parser module 'owlready2_optimized' is not available, defaulting to slower Python implementation
```

Figure C.1: DKB set-up: Owlready2 warning message

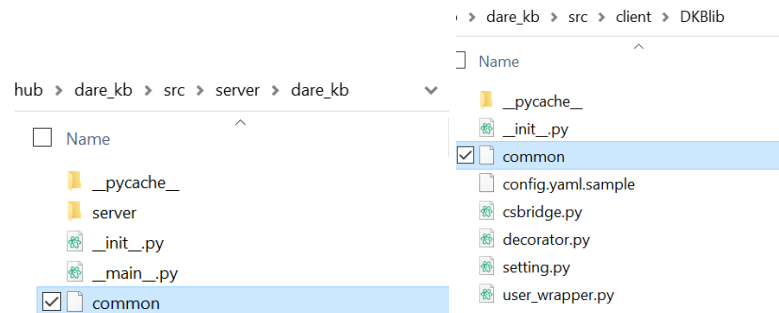
As elaborated by Jean-Baptiste Lamy, the inventor of Owlready2, it is not required to use the optimized module (the functionality will be the same) [73]. However, Cython is used to optimize some code in C, thus if Cython is available during the Owlready installation it can result in a 20% performance boost when parsing ontologies [72, 73]. To benefit from the performance boost, one has to:

- Install Cython ([pip install Cython](#))
- As well as a C Compiler (such as [GCC](#) for instance)
- Before forcing a [re-installation](#) of owlready2” [75]

C.1.2 For Windows users only: potential run issues [75]

“In case you are a Windows user and are encountering problems when using DKB (e.g., the `../common` path in the `common` txt file is not resolving properly (symbolic link issue)) you might want to consider the following:

1. First of all, it is important to run the `python setup.py develop` command from the Windows command prompt (`cmd`). In case you have installed an Ubuntu app (e.g.) this shouldn’t be used to perform the client and server installation.
2. Now, the two common files in the `../dare_kb/src/client/DKBlib` and `../dare_kb/src/server/dare_kb` directories have to be deleted.



(a) Server folder

(b) Client folder

Figure C.2: DKB set-up: common txt file location

3. Thereafter, command `mklink /d this-link-points-to c:/that-directory` can be utilized to manually re-create the symbolic links (“symlink”). You should again use the Windows `cmd`, `cd` into the `../dare_kb/src` directory and then run the following two commands:

(a) `mklink /d client/DKBlib/common ../../common`

(b) `mklink /d server/dare_kb/common ../../common`

This will create two linked directories, pointing to `../dare_kb/src`, e.g.:" [75]

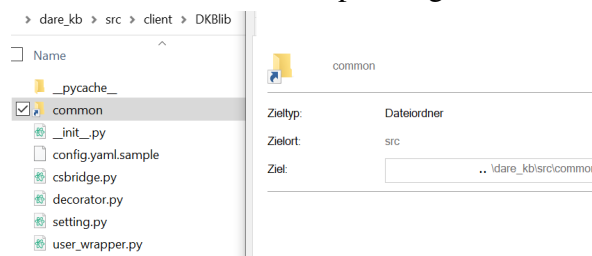


Figure C.3: DKB set-up: linked directory

C.2 Find () search criteria and expected values

Moreover, the “[find\(\)](#)” chapter has been expanded by two sections which explain the available input parameters and expected values as well as the way the range selection works for ‘string’ values (sections and their content following below).

C.2.1 Available property search criteria and expected values [75]

C.2.1.1 Concept

Property	Expected value	Example
‘name’	String surrounded by ‘ ’	‘TestCon’
‘prefix’	String surrounded by ‘ ’	‘ex3KB:kb:1:Concept’
‘pid’	String surrounded by ‘ ’	‘ex3KB:amelie3:0:TestCon’
‘description’	String surrounded by ‘ ’	‘This is a concept with description’
‘state’	String surrounded by ‘ ’	‘active’ (Other possible states: ‘new’, ‘frozen’, ‘deprecated’, ‘discarded’)
‘timestamp’	Timestamp string surrounded by ‘ ’, in format ‘YYYY-MM-DDTHH:MM:SS.MMMMMM’	‘2021-06-29T17:05:27.792212’
‘mutability’	String surrounded by ‘ ’	‘mutable’
‘translation’	Dictionary string surrounded by ‘ ’; It is important to additionally surround string dictionary keys/ values with “ “	{“name”: “String”}
‘py_class’	String surrounded by ‘ ’	‘String’
‘methods’	Dictionary string surrounded by ‘ ’; It is important to additionally surround string dictionary keys/ values with “ “	{“OneMethod”: “String”}

Table C.1: DKB find: Concept search parameters, expected values and examples (as added to [75])

C.2.1.2 Instance

Property	Expected value	Example
'pid'	String surrounded by ' '	'ex3KB:amelie3:0:tc1'
'prefix'	String surrounded by ' '	'ex3KB:amelie3:0:TestCon'
'name'	String surrounded by ' '	'tc1'
'state'	String surrounded by ' '	'active' (Other possible states: 'new', 'frozen', 'deprecated', 'discarded')
'timestamp'	Timestamp string surrounded by ' ', in format 'YYYY-MM-DDTHH:MM:SS.MMMMMM'	'2021-06-29T17:05:27.792212'
'mutability'	String surrounded by ' '	'mutable'

Table C.2: DKB find: instance search parameters, expected values and examples (as added to [75])

C.2.2 Range sort: selection according to ASCII sort order [75]

“**Please be aware** that the **String** range selection is based on the below illustrated **ASCII** sort order (<http://asciiset.com/>). This means for instance, that all uppercase letters come before the lowercase letters (e.g., C < b, thus Cat < bear) or that curly brackets { } came after all letters (uppercase and lowercase) and round brackets () as well as inverted commas ” before all letters (uppercase and lowercase).” [75]

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
32	20	Space	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	\$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(72	48	H	104	68	h
41	29)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	;	91	5B	[123	7B	{
60	3C	<	92	5C	\	124	7C	
61	3D	=	93	5D]	125	7D	}
62	3E	>	94	5E	^	126	7E	~
63	3F	?	95	5F	_	127	7F	□

Figure C.4: DKB find: string range results in accordance with ASCII sort order [100]

C.3 Added notes

Additionally, a few notes have been added to avoid misunderstandings and increase clarity:

- Context naming convention: *“Please respect the naming convention: whitespaces are not allowed, otherwise a **DKBException: IdentifierWrong** is thrown”* [75]
- Entering a context: *“Please note: You are only allowed to `enter()` a context – either in read ‘R’ or write ‘W’ mode – if no other user is currently logged into this context in write mode ‘W’. Otherwise, a **‘MultiUser’ DKBException** is thrown:”* [75]

```

M | dkb.enter('Context_prefix4', 'R')
-----
DKBException                                Traceback (most recent call last)
\github\dare_kb\src\client\DKBlib\csbridge.py in wrapper(*args, **kwargs)
    36     try:
--> 37         return func(*args, **kwargs)
    38     except DKBException as e:

\github\dare_kb\src\client\DKBlib\csbridge.py in enter(site, session_id, a_prefix, rw)
    81 def enter(site: str, session_id, a_prefix, rw) -> str:
--> 82     ret = _do_request(site, "POST", '/user/current_context', session=session_id, prefix=a_prefix, rw=rw)
    83     return ret

\github\dare_kb\src\client\DKBlib\csbridge.py in _do_request(site, method, path, parse_json, **data)
    62     msg = json.loads(r.text)
--> 63     handle_server_exception(r.status_code, msg)
    64     return msg

\github\dare_kb\src\client\DKBlib\common\cs_exception.py in handle_server_exception(status_code, message)
    63     error_code, message, extra = parts[0], parts[1], parts[2:]
--> 64     raise DKBException(error_code, message, extra)
    65     else:

DKBException: ('MultiUser', "Multi user in R is not supported for context Context_prefix4. Users: ['root'] are using Context_prefix4 in W mode.", [])

During handling of the above exception, another exception occurred:

MultiUserError                                Traceback (most recent call last)

```

Figure C.5: DKB enter context: MultiUser exception

- Leaving the current context: *“The `leave` function quits the current context. It leaves you with the first context ever created (“default” context).”* [75]
- Freeze(), Deprecate() and Reset() a context: *“Please note: You are only able to `freeze()` a context, if this context was entered in write mode ‘W’. Otherwise, a **WritingPermissionDeniedError** will be thrown.”* [75]

Appendix D

Survey questionnaire

This chapter contains the seven question long questionnaire, which is split into four sections – namely,

- A. Functionality deficiencies,
- B. Application bug fixes,
- C. User Manual updates and
- D. Any other comments

– as well as the survey outcome, the information sheet and (plain) user consent form.

D.1 Questionnaire

MSc project: Demonstrating the power of user-controlled contexts

The overall goal of this MSc project is to enhance the current DKB (DARE Knowledge Base) application prototype, which had been developed as part of the European funded DARE (Delivering Agile Research Excellence on European e-Infrastructures) project. To evaluate the value of the performed adjustments, this questionnaire has been set up: On a scale from 1 to 5 (1= Not helpful at all; 2 = Not helpful; 3 = Not sure; 4 = Helpful; 5 = Extremely helpful) participants should rate DKB's current state and the improvement(s).

The questionnaire does not collect any personal information and shouldn't take more than 5-10 minutes of your time. Please also ensure to not add any information, which make you personally identifiable, to the optional, final free text field.

Thanks in advance for your participation! :)

A. Functionality deficiencies

The DKB tests revealed several functionality deficiencies, out of which the below two enhancements were chosen as focus topics. Therefore, please assess each addition with regards to its usefulness for you as potential future DKB users.

Scale: 1= Not helpful at all; 2 = Not helpful; 3 = Not sure; 4 = Helpful; 5 = Extremely helpful

1

A.1 Collection conceptualization and basic implementation

People have a natural tendency to collect things - from the classic stamps to scientific ensembles of simulation results, relevant observations or examples of new phenomena. To further support you as scientists in your daily work we therefore decided to design and implement the concept of **Collections**, which we defined as *objects ("containers"), which group (multiple) individual elements into one unit*. They consist of:

1. The container/ collection wrapper, which can be:
 - a. Dynamically (query) or manually (object selection) created
 - b. Materialized or Referenced (storing actual data results or solely a reference)
 - c. Ordered or Unordered
 - d. With or without duplicates (based on a user-defined discriminating function)
2. Its content (the individual elements such as *concept, instances, contexts and collections*)
3. Methods/ functions for accessing and working with Collections and their content (such as:
 - a. *Element-oriented* operations like Insertion, Update, Removal, Listing or Subsetting of elements
 - b. *Collection-oriented* operations like Applying a function to all elements or creating Unions, Intersections and Set differences between collections

Not helpful  Extremely helpful

2

A.2. Find() functionality extension

Up until now, the **find()** functionality (for querying data from the database) was rather limited in regards to look-up parameters (solely **PID** and **prefix**) and operators (only **equality** checks).

Now, additional parameters (**name, description, state, mutability, timestamp, translation, method** and **py_class**) and operators (**inequality** ("!="), various **range operators** ("<", "<=", "<=", ">", ">=", ">=")) are also in place.

Not helpful  Extremely helpful

Figure D.1: Survey questionnaire: first section

MSc project: Demonstrating the power of user-controlled contexts ...

C. User Manual updates

All the tests and discussions also revealed some necessary User Manual updates (<https://docs.google.com/document/d/1u62251KnRURztDyBbxo77yfGGBpjzLWBLSinArxX0/edit#heading=h.eroja0zbn2tt>), for example two sections were added to address potential installation and DKB run issues. Moreover, the `find()` chapter was extended to explain the available input parameters and expected values as well as the way the range selection works for String values.

Additionally, a few **notes** were added to avoid misunderstandings and increase clarity, like the naming convention, some multi user information as well as other prerequisite when using certain functionalities.

6

C.1 User Manual updates

Please again rate how helpful you consider this additional information.

Scale: 1 = Not helpful at all; 2 = Not helpful; 3 = Not sure; 4 = Helpful; 5 = Extremely helpful

Not helpful ☆ ☆ ☆ ☆ ☆ Extremely helpful

Back Next Page 3 of 4

Figure D.3: Survey questionnaire: third section

MSc project: Demonstrating the power of user-controlled contexts ...

D. Any other comments

If you have any other comments regarding the performed DKB adjustments (and don't mind me potentially quoting those anonymous statements in my dissertation :)) or further ideas for future improvements, I would highly appreciate your input!

7

D.1 Comments/ Feedback (optional)

Enter your answer

You can print a copy of your answer after you submit

Back Submit Page 4 of 4

Figure D.4: Survey questionnaire: fourth section

D.2 Survey results

1. A.1 Collection conceptualization and basic implementation

[More Details](#)[Insights](#)

10

Responses



4.50 Average Rating

2. A.2. Find() functionality extension

[More Details](#)[Insights](#)

10

Responses



4.90 Average Rating

Figure D.5: Survey results: first section

3. B.1 Instance creation issue

[More Details](#)[Insights](#)

10

Responses



4.60 Average Rating

4. B.2 Data access after DKB server restart

[More Details](#)[Insights](#)

10

Responses



4.60 Average Rating

5. B.3 Find() issue

[More Details](#)[Insights](#)

10

Responses



4.70 Average Rating

Figure D.6: Survey results: second section

6. C.1 User Manual updates

[More Details](#)

 Insights

10

Responses



4.40 Average Rating

Figure D.7: Survey results: third section

7. D.1 Comments/ Feedback (optional)

[More Details](#)

 Insights

7

Responses

Latest Responses

"For a knowledge base like DKB to be operational, it is important that...

""

"The additional improvements help to improve some existing inefficie...



7. D.1 Comments/ Feedback (optional)

7 Responses

1	anonymous	Thanks to all those adjustments. They were really needed and are very useful!!!
2	anonymous	Very significant progress; thank you; when will it be made available so that we can use it?
3	anonymous	I appreciate your work to improve the DKB. It increased the usability of the DKB.
4	anonymous	NO. The DKB adjustment is wonderful and helpful.
5	anonymous	I'm a heavily biased and atypical responder!
6	anonymous	The additional improvements help to improve some existing inefficiencies!
7	anonymous	For a knowledge base like DKB to be operational, it is important that basic functions such as the ones that have been added, as well as good documentation, is provided.

Figure D.8: Survey results: fourth section

D.3 Information sheet

Page 1 of 3

Participant Information Sheet

Project title:	Demonstrating the power of user-controlled contexts
Principal investigator:	Malcolm Atkinson (Malcolm.Atkinson@ed.ac.uk)
Researcher collecting data:	Patricia Hartmann (P.R.Hartmann@sms.ed.ac.uk)
Funder (if applicable):	

This study was certified according to the Informatics Research Ethics Process, RT number #5959. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

MSc student: Patricia Hartmann

Supervisor: Malcolm Atkinson

What is the purpose of the study?

The overall goal of the underlying MSc project is to enhance an existing application prototype, namely the DARE Knowledge Base (DKB), which had been developed as part of the European funded DARE (Delivering Agile Research Excellence on European e-Infrastructures) project (<https://cordis.europa.eu/project/id/777413>), by identifying and resolving a dedicated issue.

Since eventually the (scientific) user community should benefit from any of DKB's enhancement(s), this survey is undertaken to evaluate the value of DKB's adjustments. With the help of a (completely anonymous) questionnaire the participants will be asked to review and rate DKB's current state and the improvement(s).

Why have I been asked to take part?

You have been invited to take part in the study since you were previously involved in the DARE project and/ or DKB's development (e.g., by providing requirements and feedback).

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time without giving a reason, up until the point in time you successfully completed and submitted the questionnaire. After this point, personal data will be deleted and anonymised data will be combined such that it is impossible to remove individual information from the analysis. Your rights will not be affected. If you wish to withdraw, contact the PI. We will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?

If you agree to participate in this survey, you will receive one questionnaire with several questions regarding the DKB application. Overall, this shouldn't take longer than 5-10 minutes to fill in. The questionnaire will neither ask nor capture any personal information since the aim is solely to review and rate DKB's current state and the implemented improvement(s).

Are there any risks associated with taking part?

There are no significant risks associated with your participation.

Are there any benefits associated with taking part?

An indirect benefit associated with this MSc project and your feedback is the opportunity to move DKB, which initially was planned to be an integral part of the DARE platform, towards a productive and thus more stable version, which can finally be utilized by the DARE platform.

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a maximum of four years. All potentially identifiable data will be deleted within this timeframe if it has not already been deleted as part of anonymization.

Data protection and confidentiality.

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher – Patricia Hartmann.

All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint) and all paper records will be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separately from your responses in order to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Patricia Hartmann (P.R.Hartmann@sms.ed.ac.uk).

If you wish to make a complaint about the study, please contact inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on <http://web.inf.ed.ac.uk/infweb/research/study-updates>.

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Patricia Hartmann (P.R.Hartmann@sms.ed.ac.uk).

General information.

For general information about how we use your data, go to: edin.ac/privacy-research

D.4 Plain consent form

Participant number: _____

Participant Consent Form

Project title:	Demonstrating the power of user-controlled contexts
Principal investigator (PI):	Malcolm Atkinson
Researcher:	Patricia Hartmann (P.R.Hartmann@sms.ed.ac.uk)
PI contact details:	Malcolm.Atkinson@ed.ac.uk

By participating in the study you agree that:

- I have read and understood the Participant Information Sheet for the above study, that I have had the opportunity to ask questions, and that any questions I had were answered to my satisfaction.
- My participation is voluntary, and that I can withdraw at any time without giving a reason. Withdrawing will not affect any of my rights.
- I consent to my anonymised data being used in academic publications and presentations.
- I understand that my anonymised data will be stored for the duration outlined in the Participant Information Sheet.

Please tick **yes** or **no** for each of these statements.

1. I allow my data to be used in future ethically approved research.

<input type="checkbox"/>	<input type="checkbox"/>
Yes	No

2. I agree to take part in this study.

<input type="checkbox"/>	<input type="checkbox"/>
Yes	No

Name of person giving consent	Date dd/mm/yy	Signature
_____	_____	_____
Name of person taking consent	Date dd/mm/yy	Signature
_____	_____	_____