MOnSTER

Multi-Ontology Semantic Trial Enrichment Resource

Dissertation

to attain the degree of a

Doctor of Theoretical Medicine

of the Faculty of Medicine
of the Saarland University
Homburg / Saar

2022

submitted by

Holger Stenzhorn

born on April 27th, 1976
in Saarbrücken – Dudweiler

Dedication

I want to dedicate this dissertation to my parents Gudrun and Gerd and to my wife Tatsiana: Without their everlasting love, support and encouragement I would have never ever been able to realize this!

Also, I want to thank my two little “office dogs” Joschi and Toschi who always kept calm and relaxed providing some positive energy even when times got busy and hectic...

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Dr. med. Norbert Graf for giving me the opportunity to be part of his research team and to work on such interesting and challenging topics.

Of course, I want to thank all my teammates and project collaborators, as the development of such a complex piece of software like ObTiMA can always only succeed when working together as a team.

Abstract / Zusammenfassung

Abstract

As the name suggests, one of the main goals of the Ontology-Based Trial Management Application, or ObTiMA for short, has always been the application of ontologies to achieve semantic interoperability within clinical trials. The Multi-Ontology Semantic Trial Enrichment Resource (MOnSTER) now represents a complete reimplementation of this task area as a new module component of the application: With the help of this component, it is now possible to semantically tag all parts of a clinical trial with corresponding concepts from freely addable ontologies and export the obtained study data in ODM, RDF and FHIR formats.

In this thesis, first the thematic background of syntactic and semantic interoperability is discussed. The following describes the technical implementation and integration of the component into the overall application. In particular, it addresses loading and managing ontologies, adding concepts to study components, and exporting study data in various formats. After that, it is shown how a user can apply these functionalities of the component with the help of Graphical User Interface and a web service. In the discussion, an evaluation of the component by the users is described, possible future extensions are pointed out, and a comparison with similar systems is given.

Zusammenfassung

Wie der Name bereits sagt, war eines der Hauptziele von Ontology-Based Trial Management Application, kurz ObTiMA, schon immer die Anwendung von Ontologien, um semantische Interoperabilität innerhalb klinischer Studien zu erreichen. Die Multi-Ontology Semantic Trial Enrichment Resource (MOnSTER) stellt nun eine vollständige Neuimplementierung dieses Aufgabenbereichs als neue Modulkomponente der Anwendung dar: Mithilfe dieser Komponente ist es nun möglich, alle Teile einer

klinischen Studie mit entsprechenden Konzepten aus frei hinzufügbaren Ontologien semantisch zu taggen und die gewonnenen Studiendaten in den Formaten ODM, RDF und FHIR zu exportieren.

In dieser Arbeit wird zunächst der thematischen Hintergrund der syntaktischen und semantischen Interoperabilität erörtert. Im Folgenden wird die technische Umsetzung und Integration der Komponente in die Gesamtanwendung beschrieben. Insbesondere wird hier auf das Laden und Verwalten von Ontologien, das Hinzufügen von Konzepten zu den Studienkomponenten und das Exportieren von Studiendaten in verschiedenen Formaten eingegangen. Danach wird gezeigt, wie ein Benutzer diese Funktionalitäten der Komponente mithilfe der grafischen Oberfläche und eines Webservices anwenden kann. In der Diskussion wird eine Evaluation der Komponente durch die Anwender beschrieben, es werden mögliche, zukünftige Erweiterungen aufgezeigt, sowie ein Vergleich mit ähnlichen Systemen gegeben.

Contents

Part I

Introduction

Chapter 1

General Background

Clinical trials are the de-facto gold standard in medical research to assess and evaluate diagnostic interventions, treatment procedures, drugs and devices, and so on, before any of them can be applied in daily routine medical care (Friedman et al., 2015). For most current trials, both volume and structure of the collected data are now so extensive and complex that any paper-based management, as practiced in the past, is simply not possible anymore (Walther et al., 2011).

Therefore, a large variety of software systems has been developed over the years to address and meet this challenge in different ways and at different levels: While smaller trials might rely on using homegrown, specific tools and databases that cover only certain aspects of a trial, larger, multicentric trials usually employ complete, integrated software solutions to manage a trial's entire lifecycle and tasks, from its planning, preparation and execution to its reporting and analysis (Nourani et al., 2019).

The Ontology-Based Trial Management Application, or ObTiMA for short, represents precisely such a clinical trial management system , or CTMS for short (Stenzhorn et al., 2010): Initially developed within the scope of several European projects, like ACGT (Advancing Clinico Genomic Trials on Cancer) (Martin et al., 2011) and p-medicine (From Data Sharing and Integration via VPH Models to Personalized Medicine) (Marés et al., 2014), as a prototypical demonstrator to showcase the latest insights in ontology-based interoperability, the system has been continuously extended and elaborated further into a full-fledged application fulfilling all requirements towards such a software in the setting of real-world clinical trials (Kuchinke et al., 2016; Stenzhorn et al., 2012).

From the beginning, the main distinguishing feature of ObTiMA compared to other systems has therefore been its clear focus on ontologies: For this, the application has allowed the creation of so-called "ontologized" questionnaires, or CRFs (Case Report Forms), to perform the collection of subject data within a trial. This means that all items within such a CRF, that is, questions and answer options, could be defined on the

basis of ontological concepts predefined in a custom-developed ontology built directly into the application. Ultimately, the goal of this endeavor has been to automatically "ontologically enrich" the collected subject data and thereby promoting their semantic interoperability at a conceptual level.

Unfortunately, this ontological foundation has recently fallen out of focus somewhat, as the continued development has increasingly targeted on some other aspects that are (more) essential for a productive use of the application, such as security, robustness and scalability. It is therefore precisely the aim of this thesis to revisit that given original objective: Taking the original realization of the topic as starting point, this previous work is evaluated in light of the most current advances regarding interoperability in both research and technology. Based on this, a novel software component is proposed, which, on the one hand, takes up the fundamental ideas as introduced in the initial realization, but, on the other hand, is a completely new development that incorporates and implements the named scientific and technological advances.

In more concrete terms, a MOnSTER (Multi-Ontology Semantic Trial Enrichment Resource) is created that supports its host ObTiMA by providing novel capabilities to communicate with the "outside world" both in respect to common, shared semantics as well as standardized syntaxes, that is, formats. Regarding semantics, this means that with the help of the new component it becomes possible to not only create ontology-based questions within a CRF, but that all parts of a trial, like its SEs (Study Events) or the trial itself, can be semantically enriched with ontological concepts. The new component also allows ontologies to be freely chosen and used for this purpose, rather than having to rely on the single, built-in ontology. Regarding syntaxes, it is, of course, essential that the component keeps and improves the existing support for ODM as one of the de-facto standards for exchanging clinical trial data (Hume et al., 2016). But by providing RDF (Schreiber et al., 2014) and FHIR (HL7, 2019) as additional export options, interoperability is broadened to a wider range of systems covering additional topical areas and environments.

Finally, it should be emphasized that the development of this component is not just about technology but also about its actual usage: Thus, it is not the sole aim to develop a software which follows and includes the latest research and technological trends. Rather,

the component tries to be as easy to use as possible and to seamlessly integrate and blend itself into the existing application and its workflows.

In this following thesis, it is thus described how these given aspects are implemented within and by MOnSTER and how an actual user can apply the resulting, new functionalities in the daily work when using ObTiMA.

Chapter 2
Interoperability

For a better understanding of the specific background of MOnSTER, it is necessary to first provide an overview of the more general aspects that are being addressed by this thesis' work. The most important and overarching among them is interoperability.

The need for interoperability is both urgent and ubiquitous in medical routine care as well as in research in the healthcare and life sciences: Albeit fundamentally expressing the same underlying content, the actual data that represents that very content often exhibits large variations in regard to several different aspects from location to location and system to system. Therefore, the multitude and magnitude of differences in their type, format, quality, as well as mismatches in the used terminology or jargon render it difficult to link and integrate data from different sources and, in turn, to collaborate both in daily routine and in research.

Now, looking at the current publications, a variety of definitions exist for the term interoperability, depending on the basic scope and environment where each is being employed. A prominent example for such a definition in the scope of healthcare is provided by the Healthcare Information and Management Systems Society (HIMSS, 2021) who declare that

> (Interoperability) is the ability of different information systems, devices and applications (systems) to access, exchange, integrate and cooperatively use data in a coordinated manner, within and across organizational, regional and national boundaries, to provide timely and seamless portability of information and optimize the health of individuals and populations globally.
>
> Health data exchange architectures, application interfaces and standards enable data to be accessed and shared appropriately and securely across the complete spectrum of care, within all applicable settings and with relevant stakeholders, including the individual.

In the given (bio)medical context, the European Union gives yet another often cited definition of the term in its regulations on medical devices and in-vitro diagnostics, with similar meaning to the above (EP-EC, 2017a, 2017b) by stating that

> (Interoperability) is the ability of two or more devices, including software, from the same manufacturer or from different manufacturers, to:
>
> (a) exchange information and use the information that has been exchanged for the correct execution of a specified function without changing the content of the data, and/or
> (b) communicate with each other, and/or
> (c) work together as intended.

Returning to the first definition above, HIMSS also proposes to divide (general) interoperability further into four general but distinct levels, as listed below (HIMSS, 2021; IBM, 2021):

Foundational This first level defines the foundational interoperability requirements for systems, applications and devices so that they can securely interconnect to send data to and receive data from each other. It is only concerned with the actual transport and transmission of the data, but neither with their interpretation nor with their conversion into specific forms or formats.

Structural The second level deals with conditions toward structural interoperability, that is, the standardization of data by defining common and uniform structures, formats and syntaxes for data. In this way, the various systems, applications, and devices can all structurally interpret data conforming to these stipulations.

Semantic At this third level, requirements for semantic interoperability are specified, including common information models and data elements using publicly available, standardized coding systems and vocabularies. This is to ensure that all components involved in the process share a common understanding and meaning of the given data.

Organizational The fourth and highest level, declares the preconditions for the communication, exchange and use of data between various organizations, entities

and individuals. These include a variety of aspects arising from governance, political, social, legal and organizational considerations.

Regarding ObTiMA, the fourth and highest level can be seen as out-of-scope for this work, as it obviously goes beyond what is technically feasible from a software system. Here, social and also societal factors play the main role, which might be influenced and supported to a certain degree by some technical solutions, yet cannot be fundamentally solved by them. Also, even though the first, lowest level refers to technology directly, this one does not fit the focus of the given work either as it addresses the underlying infrastructural requirements for data transport and transmission only, such as the basic network protocols, like HTTP. Hence, the work in this thesis is concerned with the two levels in between, and thus focuses on how data can be defined and provided in such a way that both its structure, that is, its format, and its meaning, that is, its encoding, can be understood by different systems and thus exchanged in between them.

To make the above more tangible, one particular case where interoperability, or rather the lack thereof, is playing an acute and highly important role is the COVID-19 pandemic. For example, the testing regimen for infections in the USA highlights the considerable problems arising in the initial phase of the pandemic at all above-stated interoperability levels, causing a massive impairment in the efficient processing and exchange of the result data, as well as their joint analysis (Greene et al., 2021). Much of this is due to semantic heterogeneity between different parties and the missing of common, standardized coding information based on common terminologies, eventually leading to a wide variation in the interpretation of single data values and, in turn, whole data sets.

Of course, this problem also immediately affects the many clinical trials started worldwide in the context of the pandemic. Here, already the sheer, massive amount of concurrent trials with the aim to rapidly produce results in this situation, is by itself already extremely challenging (Dron et al., 2021). The need to establish interoperability based on common data standards is emphasized as a necessary prerequisite as well, so that data from those trials can be readily shared, aggregated and integrated with each other as quickly as possible, such as for overarching meta-analyses. In this context, the CIA (COVID-19 Interoperability Alliance) is established for exactly this task, namely to collaboratively develop and share interoperability resources, such as terminology standards, common

code and value sets, or implementation guidelines, that can (and should) be commonly used by research activities in the scope of COVID-19 (CIA, 2021).

2.1 Structural / Syntactic Interoperability

In order to clarify the distinction as introduced above, interoperability on the structural and syntactic level is explained here first. This kind of interoperability is essentially concerned with the development and utilization of standardized so-called information models to describe data and the respective technical implementations used to express them. These implementations include, on the one hand, specifications, descriptions and (public) provisioning of the required formats, and on the other hand, the development of interfaces based on those formats and their adaptation in the relevant software systems and components. Thus, the main goal here is that these specifications and implementations ensure that data can be moved from one location or system to another by defining and realizing rules to properly transmit data. For this, agreements need to be established between all involved partners about the specific orders and hierarchies of the data and their corresponding properties and types. As a simple example, when encoding dates, its format must be clearly predefined in advance, so that both data producers and consumers both apply the same. For example, if one system uses `1981-01-04` following the ISO (International Organization for Standardization) standard (ISO, 2019), whereas the other uses `04.01.1981` following the DIN standard (DIN-NIA, 2020), then neither system can correctly interpret dates from the other.

Now, to achieve successful syntactic interoperability, several factors come into play: One of them, for example, is the feasibility of implementing specifications. The more complex a given specification is, the more complicated its actual implementation becomes, which in turn renders its application and dissemination more problematic. On the one hand, there is a risk that the complexity of such a specification means that it will not be implemented in full, and on the other hand that such complexity leads to (hidden) errors in their implementation. Nonetheless, any specification must still be defined precisely enough to prevent them from becoming ambiguous and thus to be applied in different, possibly also contradictory ways.

Another factor is the openness and free availability of specifications and standards: Creating such in open, public and clearly defined processes allows their intended user community to participate in their development, which in turn ensures that specific needs from that community are better addressed. Furthermore, the adaptation of such standards is promoted if they are made openly and freely available under a permissive (open source) license, without having to pay user fees here - at least for academic and non-commercial use. One example for a basal, syntactic standard specification is XML (Extensible Markup Language), as a highly general and generic markup format (Bray et al., 2008). The development of this standard, as well as further descendant standards, occurred in a, open and transparent process coordinated by the W3C (World Wide Web Consortium), which involved not only the (user) community but also academia as well as commercial companies (Connolly, 2003).

Actually, XML can also be seen as both a positive and negative example in regard to the first factor stated above. Due to its relative simplicity and clear definition, the implementation adn application of this standard by its users can be seen as unproblematic. Thus, building on this, a plethora of both open source and commercial systems and components have been developed for processing XML, which in turn, of course, simplifies the development of custom interfaces and software based on that standard.

Yet, the strong genericity of XML might also be seen as causing the problem mentioned above: The standard declares only on a highly general level how a (syntactically) correct structure must be formed. Yet, it does not declare how a structure should look like for any concrete use case and it also does not declare how each element within that structure should look like or how they relate. So, for example, the standard does not prescribe whether a separate XML element needs to be used for storing some particular value in a particular case, or whether that value should be simply encoded as an attribute as part of another element. Hence, by relying only on pure XML alone it is fundamentally not possible to declare what content (types) can or has to be stored in some specific document. For this purpose, it is necessary to first create a concrete domain- or task-specific information model and then to take XML as a foundation for its implementation. That means that XML itself provides only the basic structural skeleton for documents but the specific elements that may appear in such documents and their respective, allowed position in that structure

need to be specified additionally on top of that. For this, another specification from the W3C, namely XSD (Gao et al., 2012), that itself uses XML as its underlying base, can be employed to express such a model in technical, machine-processable terms.

A concrete example for this is the ODM (Operational Data Model) standard for exchanging clinical trial data (CDISC, 2013). Here, its information model defines exactly the data elements and their structural relations required for this given task. This model is both described in a human-readable textual representation with the necessary descriptions and explanations as well as by a XSD file which describes the same model but in a machine-processable form that software tools can employ to validate the correctness of ODM files. So, by employing this standard, the electronic and structured representation of data at all stages of a clinical trial is made possible (Huser et al., 2015) and thus CTMSs supporting this format can readily export their data and any ODM-enabled analytics tools can in turn read, process and analyze this (Brix et al., 2018). Also, the use of ODM also allows for a uniform way to archive clinical trial data (Kuchinke et al., 2009), and therefore ODM also represents one of the fundaments for managing trial data compliant with the principles of GCP (Good Clinical Practice) (Ohmann et al., 2011).

Another example is the FHIR (Fast Healthcare Interoperability Resources) standard (HL7, 2019), whose main target is the exchange of data between systems used in healthcare, and whose information model also has one implementation using XML as its base. Here, from the XML perspective, only by properly defining the technical realization of all information model entities and their dependencies through corresponding XML elements and their structuring, data can be successfully represented in FHIR format. FHIR is also generally a very good example of how such a standard can promote (syntactic) interoperability: This standard is rapidly gaining acceptance and application not only in purely clinical environments alone, where with its help their different, internal subsystems of the HIS (Hospital Information System) can be interconnected. In fact, more and more (portable) applications and devices are being developed by major industry players for both patients and healthcare professionals, to provide and exchange data, such as Apple with its HealthKit, ResearchKit and CareKit APIs (Apple, 2021, 2022).

Nonetheless, a major problem in regard to syntactic interoperability within both routine medical care and research, like in clinical trials, is that in many cases the data there

are either largely or even completely unstructured. Examples from routine care include clinical reports, consent forms, and referral documents. These are often pure text documents based solely on natural language, which, however, do not have a machine-readable and interpretable structure or formatting of the information contained. So to again come back to the example of XML here: The mere fact that this basic format is employed does not provide any help by itself. To exaggerate, it is possible to take a single XML element and, simply put, in this a (complete) text containing all information. Syntactic interoperability can, however, only succeed if all individual information elements are expressed by suitable data elements and presented in a truly structured and well-defined manner. Without such basis, the next level, that is, semantic interoperability, is hard or even impossible to achieve.

2.2 Semantic Interoperability

This kind of interoperability builds on the syntactic one, but extends it in the sense that not only common information models and structures are specified and shared, but also a deeper and common understanding of the data is intended.

To illustrate what is meant by this, the so-called semiotic triangle in Figure 2.2.1 can be used, which is a model of how the symbols of natural language are connected to actual objects in reality (Odgen & Richards, 1923). For example, when the words “Jane Doe” are used then this is a symbol that stands for the concrete referent, that is, the concrete flesh-and-blood person with that name. Now, when someone reads or hears the words “Jane Doe,” this symbolizes the reader’s or listener’s mind the concept, or reference, that relates to the actual person. Then, if Jane is married to John Doe, the words “John’s wife” stand for exactly the same person, that is, the same referent, and the reader or hearer thus has the same reference in mind referring to the same actual person. Exactly the same then applies when the German words “Johns Frau” are used, as both referent and reference are then just the same again, only expressed in a different language’s words.

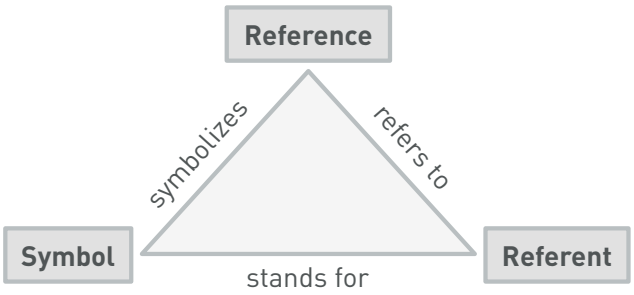


Figure 2.2.1 – *Ogden and Richards' semiotic triangle*

In the field of medicine, of course, this issue is ubiquitous: For the same diseases, the same therapies, or the same drugs, and so forth, there exist almost always several different natural language terms or expressions, which have exactly the same meaning. An example of this is the disease nephroblastoma, the most common malignant renal tumor in childhood, originating from embryonic tissue remnants of the kidney (Yiallouros et al., 2009). Now, since the German surgeon Max Wilms was the first to describe this disease, it is often also referred to as Wilm's tumor. So, albeit "nephroblastoma" and "Wilm's tumor" are two quite different looking and sounding terms, or symbols, both stand for exactly the same referent and are also linked to the very same reference and, hence, have the very same synonymous, underlying meaning.

Likewise, in medicine, the exact opposite case can be often found as well and cause confusion, that is, polysemy, where single natural language expressions have multiple (potentially strongly) different meanings. To express this again with the semiotic triangle from above, in this case there are two or more references and referents for one single symbol. For example, the word "inflammation" has basically two related but still distinct meanings in medicine (Pisanelli et al., 2004): On the one hand, it is a pathologic process that occurs in blood vessels and tissues in response to an injury or abnormal stimulation. On the other hand, it is the actual inflammation "object" within the body and which has, for example, some given specific shape or diameter.

If acronyms are also considered as independent symbols in the sense of the semiotic triangle, the very same issue arises: In medicine, there are a variety of acronyms whose different meanings can only be recognized if they are written out. Consequently,

individual acronym strings are symbols in their own right, which thus have again multiple referents and hence references. As an example, when using the acronym “AMI” then this can actually refer to three completely different meanings, namely amitriptyline, amifostine, as well as to acute myocardial infarction (Davis, 2020).

Therefore, it is the very goal of semantic interoperability to develop and provide technical solutions for exactly such issues, which allow the formulation and exchange of data based on precisely predefined, unambiguous, and shared meanings. Due to the ambiguity as outlined above, such an implementation cannot be based, or at least cannot be solely based, on natural language elements. For example, to identify a particular meaning in an explicit and unequivocal way requires a unique, distinct symbol used to symbolize that particular conceptual reference, and exactly that one only. To ensure this, in most instances some abstract, so-called ”non-speaking” identifiers or codes are employed. Revisiting the above nephroblastoma example, the NCIt (National Cancer Institute Thesaurus) as a large reference ontology providing a vocabulary for clinical care, translational and basic research (NCI, 2022; Sioutos et al., 2007), contains the unique code `C3267` to (conceptually) reference this exact type of tumor. Attached to the code are its preferred (natural language) name “Wilms Tumor”, its synonym “Nephroblastoma”, and also spelling variants, like “Wilms’ Tumor”.

Here, another advantage of such codes is their language independence: If trial metadata within a registry contains such codes, a lookup for the concept, “meaning” code `C3267` would return all corresponding trials, regardless of the language of the trial’s title and description. A simple string-based search would fail here already if a trial’s title only contains the term “Wilm’s tumor” but the term “nephroblastoma” is entered as query. For a human user having the background knowledge of the field, it is easy to understand that both terms should be used to retrieve all fitting results in a search. From a machine perspective though, without any additional, machine-understandable information, the terms are just two quite differently looking character strings without any relation.

For the case just described, it could still be said that it is easier for a user to manually search for the two terms than to first find the matching NCIt code and then perform the search based on this. Nonetheless, if the goal is to realize semantic interoperability on data sets of realistic real-world size, a manual approach is obviously no longer possible.

It is essential that all (technical) parties involved have the exact same understanding of the data to be processed, so that the meaning of each contained data element and each respective value is obvious and shared by all. For example, in a clinical trial, a subject's (biological) sex might be asked for and there are (simplified) the two answers male and female. This can be expressed by using another established artifact, namely SNOMED CT (Systematized Nomenclature of Medicine Clinical Terms), a reference terminology standard for clinical documentation and reporting (Millar, 2016; SNOMED, 2022): To express the question, that is, the data element, itself with SNOMED CT, the code `734000001 (Biological sex)` can be used, and for the two answer choices, that is, the actual data values, the two codes `248153007 (Male)` and `248152002 (Female)`. So to stay in the context of trials: If the data collection within a large clinical trial is partitioned between two different CTMS, but both share and apply the same common data elements and data values with the same (conceptual) codes, then also the trial's data can be easily shared and integrated between the two CTMS and also external systems for common analyses.

Still, there is yet another important aspect related to ontologies not addressed in the discussion so far: Ontologies are usually not flat artifacts, but generally exhibit a more or less extensive and complex hierarchical structure between the contained concepts. Through this so-called subsumption or taxonomic hierarchy, concepts are linked from more general superconcepts to increasingly specific subconcepts through parent-child relationships. As the small excerpt in Figure 2.2.2 shows, NCIt's concept `C3046 (Fracture)` has both one direct, more generic superconcept, namely `C35487 (Bone Injury)`, as well as several direct, more specific subconcepts that further concretize the concept fracture based on different characteristics. One distinguishing characteristic is here whether a given fracture is open or not, resulting in the two subconcepts `C34623 (Closed Fracture)` and `C34624 (Open Fracture)`.

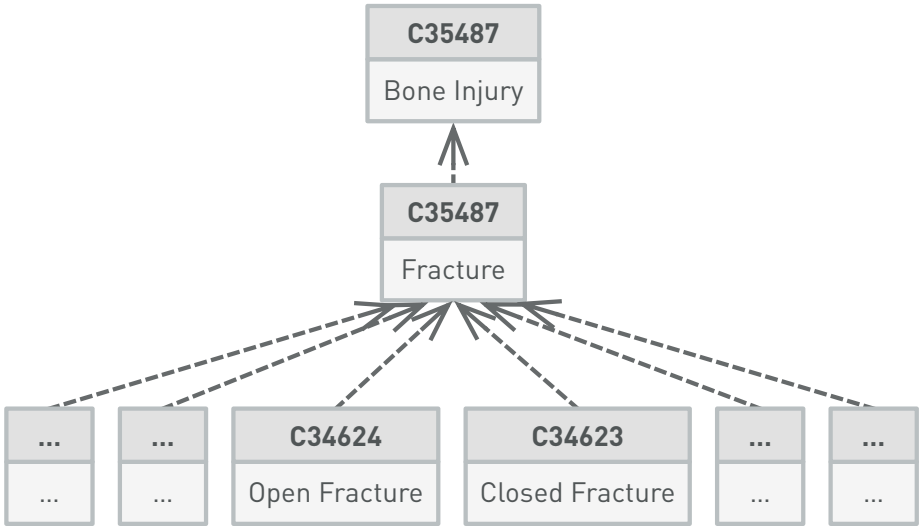


Figure 2.2.2 – *Small excerpt from the NCIt hierarchy*

An exemplary use case for this would be the (semantic) search in an ontology-enabled trial registry: If a user first searches for all clinical trials marked with the concept `C34624` as their research topic, then all trials related to on open fractures will be returned. Along with the search results, a small (visual) ontology excerpt similar to Figure 2.2.2 could be presented. Thus, in case no suitable results are returned, the user can subsequently navigate to the superconcept `C3046` to return all studies on all types of fractures.

Moreover, the vast majority of ontologies are further structured in such a way that their individual concepts are logically linked and connected through the use of semantic relations, such as to provide support for expressing the connections between concepts in the context of polysemy. When revisiting the word “nephroblastoma”, its use may refer to the actual Wilm’s tumor on the one hand, or to the associated disease on the other. For both, SNOMED CT contains the corresponding concepts, which are `25081006 (Nephroblastoma (morphologic abnormality))` for the tumor and `302849000 (Nephroblastoma (disorder))` for the disease. Although the two concepts are independent in themselves, there is a direct, logical and semantic relationship between both of them, namely the `116676008 (Associated Morphology)` that links the disease to the affected object, as shown in Figure 2.2.3. Such kind of distinction is found in many places within this ontology as well as, in fundamentally, in many biomedical

ontologies. Therefore, to guarantee a properly functioning semantic interoperability, such seemingly subtle distinctions that might be mistakenly considered as irrelevant must be always observed and considered when encoding individual data elements and their connections.

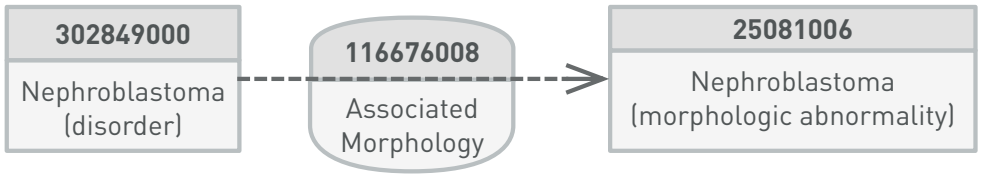


Figure 2.2.3 – *Example from SNOMED CT how a disease is related to its object*

Now, just as it is the case with syntactic interoperability, a key to successfully adapt semantic interoperability is the openness and availability of the respective standards: Since the goal is here to create and share some common understanding, the goal should also be to share and collaborate in developing this understanding. To further support such common understanding, all relevant resources should be openly and freely accessible and available to all and at all times, without major fees or licensing restrictions, again at least for non-commercial research and applications.

Yet, there are some important caveats to be made here, which arise from two opposing approaches causing two opposing problems: On the one hand, there are a lot of smaller ontologies especially in the field of biomedical research. Some examples of this can be found in the OBO (Open Biological and Biomedical Ontologies) Foundry, a community-driven initiative to develop and maintain interoperable ontologies in that thematic scope (OBO-TWG, 2021). The goal of using such smaller ontologies is to cover specific (sub)specialties, as in the case of OBO's ontology for oral health and disease to represent the content of dental office health records. While this is good as it clearly separates the contents of the respective ontologies and thus clearly outlines their respective application domains, a user has to keep track to find the appropriate ontology in each case, and also may have to use at the same time a lesser or greater number of different ontologies if data from multiple disciplines are involved.

On the other hand, the quasi-reverse problem occurs with some larger ontologies. Here,

it is the case that such ontologies are precisely not specific and focused towards a particular (biomedical) field. As a result, such ontologies often cover a multitude of subject specialities simultaneously, leading to the issue that they have a non-trivial overlap of their content with other ontologies. In the case that no further stipulations or guidelines towards the use of some specific ontology are available, it is difficult again for a user to decide which ontology should be applied for some given task.

Again, nephroblastoma may serve as a fitting example: Both NCIt and SNOMED CT contain a corresponding code for this particular tumor, that is, `C3267` and `25081006`, respectively. But also the MedDRA (Medical Dictionary for Regulatory Activities), as a required standard classification of adverse drug reactions for electronic transmission to the authorities in the EU and USA, contains a separate code for this tumor, namely `10029145`. Based on this last code example, it becomes evident that if semantic interoperability is to be achieved on a large scale, it is not only necessary to define common semantic standards, but also to clearly define their concrete application or, as in the case above, even to require them in regulatory terms.

A further example for a regulation that can serve as such guide are the MIO (Medizinische Informationsobjekte, Medical Information Object) (KBV, 2022), recently introduced in Germany, which form the basis of the DiGA (Digitale Gesundheitsanwendungen, Digital Health Application) (Weber & Heitmann, 2021) that can be prescribed to patients by physicians. Their application within the DiGA is mandated by the DVG (Digitale-Versorgung-Gesetz, Digital Healthcare Act) (Gerke et al., 2020) and illustrate both semantic and syntactic interoperability well. For the syntactic level, MIO defaults to FHIR as the interchange format. For semantic representation, SNOMED CT is used as the preferred and thus reference ontology within all MIO. In addition, others are to be used when certain things cannot be expressed via SNOMED CT and / or when a particular ontology is a (de-facto) standard in a particular domain, such as LOINC (Logical Observation Identifiers Names and Codes) for identifying laboratory observations (McDonald et al., 2003).

If no predefined regulation or suitable guideline is not provided, a “workaround” can be applied as a viable solution: It is not uncommon to use multiple codes from different ontologies rather than just one concept code from one ontology for one particular, single

data element or value. To stay with the example, it would be helpful here to add all three above codes to the trial's metadata in a CTMS: When a trial is exported along with all codes, for example, for further analyses, then the trial data can be readily integrated and processed together with other (external) data, no matter if they are encoded either with codes from NCIt, SNOMED CT or MedDRA.

To conclude, it should be noted as well that for the purpose of this work, the term "ontology" is employed in the above and in the following as a generic umbrella term covering all suitable (technical) artifacts for enabling semantic interoperability. Although a distinction is often propagated between ontologies, terminologies, classifications, and so forth, this distinction is in many cases hard to make, or even artificial and highly fuzzy as there exists a wide range of variability in the definitions, which, moreover, not infrequently contradict each other (van Rees, 2008; Zemmouchi-Ghomari & Ghomari, 2012). Evidence of this confusion is provided, for example, by NCIt itself: While it provides a controlled vocabulary, like an ordinary thesaurus, whose terms are connected by synonymy relations, it also has, like a proper classification, a hierarchical structure arranging and structuring individual terms and concepts on the basis of super- and subordinate concepts, and also, like an ontology, supplies additional (semantic) associations and relations between individual concepts.

Chapter 3

Scope and Contributions

The fundamental objective of the work is to improve and extend the initial semantic and syntactic interoperability capabilities of ObTiMA. To begin, the overall task is to identify both strengths and weaknesses of its current realization and to investigate and, if applicable, propose possible alternative solutions. To achieve this, and to meet the actual needs of the users, a review of the existing functionality has been performed jointly with the users. In this analysis, the following points have been discovered regarding the status-quo of ObTiMA:

Single Hardwired Ontology Within the application, only a single, predefined ontology can be used at a time. Also, since this ontology is hardwired into the application code, it can only be replaced by making the changes to that code.

At the beginning of the development of ObTiMA within the ACGT project, this was the ACGT MO (Master Ontology), which aimed to represent the domain of cancer research and management in a computationally tractable way (Brochhausen et al., 2011). Within the p-medicine project, this one was then replaced by the HDOT (Health Data Ontology Trunk), a core middle-layer ontology integrating several smaller modules related to personalized medicine, such as for pathology or biobanking. Both ontologies were therefore very much tailored to their given topic areas.

This means that both ontologies had to be continuously extended by hand with the necessary concepts in case they could not cover some additional (sub)domain, which, in turn, causes two problems: First, creating the suitable and fitting concepts within an ontology is not a trivial task, but requires manual, time-consuming work with the necessary technical and domain knowledge. Second, and most importantly, these own concepts are independent of those from standard ontologies with the same meaning, and therefore have, for example, their own, idiosyncratic identifiers.

Unfortunately, when data within ObTiMA now relies on these concepts, this data is not

externally interoperable, as external data employs the standard ontologies' concepts and not the "homegrown" ones. For this reason, it is crucial to give the user the possibility not only to use a single and hardwired ontology in the application, but also to add and use multiple and self-selected (standard) ontologies simultaneously, depending on the user's concrete needs and requirements.

Ontology vs. Terminology vs. Classification Building on the above, there is also some need, in this sense, for some further conceptual expansion of the existing handling of semantic sources.

The mentioned ACGT MO and HDOT are "complete" ontologies as they provide formal and comprehensive definitions and representations of the contained concepts in the given domain, together with all their relevant properties and relations. Although such ontologies are useful, since they describe a domain in a (technically) precise manner and allow, for example, automatic semantic reasoning, this kind of fully defined ontologies is not always available.

Yet, in everyday (bio)medical practice, some standard terminology systems, such as SNOMED CT (SNOMED, 2022), or classification systems, like ICD-10-GM (WHO, 2022), are generally used to perform (clinical) coding, where their respective contents are defined and provided in varying levels of complexity and structures. Due to their widespread and accepted use in both medical applications and research, and their support by various standards organizations, it is actually inevitable to make them available in ObTiMA as well, even though they may not meet the criteria for ontology as originally predefined.

It should be noted at this point too that, for the sake of brevity, only the term ontology is to be used in the following thesis description, even if a particular artifact should be a terminology or a classification in the strict sense.

Ontology Concepts for CRF Questions Only In the original realization of the ontology functionality within ObTiMA, the user can employ ontological concepts only for the definition of questions and the respective answers when creating or editing a CRF. The use of the ontology for any other task or in any other areas of the application was not intended and therefore also not implemented then.

Yet, a closer look at ObTiMA reveals that there are several other places and levels within a trial where ontologies and their concepts can be beneficial: The fundamental goal of using ontological concepts is to provide some more accurate and semantically based descriptions of things. Thus, it becomes evident that it is useful to describe not only the questions of a CRF in a semantically precise way, but all elements of some given trial. This means concretely that it must be possible to add such semantic metadata to the trial itself, to all of its SEs and CRFs, and therein to all question groups, as well as to the questions and the respective answer options.

Support of Export in ODM Only In order to be a "full-fledged" CTMS, it is an absolute basic requirement for ObTiMA to be able to export all of a trial's data to the standard ODM format, as this is still one of the most commonly used formats for the exchange, storage and archiving of clinical trial data (Kuchinke et al., 2006; Ohmann et al., 2011). Thus, all established standard software systems for the analysis and evaluation of clinical trials, such as SAS (Holland & Shostak, 2016), as well as more recent developments (Brix et al., 2018), can be used directly.

Nonetheless, considering the fact that ontologies are used for semantic enrichment of trial data and ontologies represent one of the core components of the Semantic Web (Taye, 2010), it becomes natural to also make possible the export of trial data in its standard data format, namely RDF. In this way, the exported trial data can be integrated or enriched with, for example, other publicly and freely available biomedical research data on the Semantic Web (Egaña Aranguren et al., 2014; Kamdar et al., 2019) to enable analyses that are not possible based on the trial data alone.

Finally, to further promote ObTiMA's interoperability towards new areas, the support of FHIR as export option can be helpful too. As it is one of the original objectives for the use of FHIR to provide a standard that includes all necessary information model entities and format for exchanging EHR (Electronic Health Record) data within and between different HISs (Hospital Information Systems) (Mandel et al., 2016), and lately also for encoding and exchanging PHR (Personal Health Record) data (Saripalle et al., 2019), it is now sensible as well, to express collected trial data in this format too. By doing so, this enables an easier integration of patient data generated and collected in routine clinical care with data collected within a clinical trial. Exporting metadata

of a trial and its components to FHIR is further of interest, for example, to make them available through a corresponding registry system (Gulden et al., 2021).

Need for Better Usability Another objective is to foster the usability in the context of applying ontologies for interoperability. As a core focus of ObTiMA is to provide a CTMS that stands out from similar systems by its ease of use, it is also a focus to provide ontology support at the same level of user-friendliness.

That means that this functionality needs to be offered through an interface which makes the inherent complexity of both ontologies and their underlying processing transparent to the user. The goal must be to make everything related to the utilization of ontologies accessible not only to technologically and ontologically savvy users, but rather also by medical and trial domain experts without such prior knowledge.

Hence, this also means that all new functionality needs to be available in a way that it ties in with existing operating and interface approaches, and only cautiously expands this with elements familiar to the user from daily use of regular software applications. For example, the user is provided with an autocomplete GUI element to search for matching terms and concepts, as is similarly known, for example, in the medical field from coding tools that enable searching for matching codes from ICD-10-GM or OPS relevant for reimbursement.

It is precisely these just described aspects that form the basic thematic background and provide the general objectives for the work within this thesis. Therefore, the main result and contribution of this work is the design and development of the MOnSTER component addressing the above issues by providing the following functionality:

- Ontologies can now be freely loaded, managed and used in the application without any limitation: There are no restrictions on how many ontologies are in the system at the same time and the user can add any ontology that meets their specific needs and concrete requirements.
- Since not all ontologies are provided in the standard format for ontologies, that is, OWL (Hitzler et al., 2012), they can now also be provided as files with custom line-based formats. This is helpful as there are quite some ontological artifacts that

are not implemented in OWL but in custom, often line-based, formats.

- Ontologies can now be applied to all components of a trial, which means that their ontological concepts can be (visually) added as metadata to all components via so-called semantic tags. In order to render this functionality as simple as possible for users, it is realized such that adding tags is performed in the very same way using the same GUI element for all trial components.
- The data of a trial can now be exported not only via ODM format, but also in RDF format as well as FHIR format. For this purpose, mappings are created that map this trial data represented in the ObTiMA internal information model to the corresponding information models of RDF and FHIR. In the case of RDF, a corresponding vocabulary is also being developed in parallel.

At this point, it is worth mentioning yet another aspect that is crucial for all MOnSTER design and development efforts: It is not the intention to develop an artifact that can only be utilized as a demonstrative research prototype. The objective for MOnSTER is rather to be a robust, stable and performant tool which can be easily and safely used also in productive environments, that is, for real clinical trials with actual trial patient data. For this reason, the approach to component development is very much engineering-driven, which is detailed also in the following chapters.

Based on the just said, it is also important to clarify what is not the goal of MOnSTER: Of course, since ObTiMA is an established CTMS, it can be neither desirable nor possible to perform massive, and thus risky, changes to the core architecture and implementation of the application. Therefore, the declared goal in developing the component is to integrate it carefully into the existing implementation context by reusing as much of the given infrastructure base as possible and changing only what is truly necessary. At no time may MOnSTER or changes made for it cause the integrity or security of the application and the data stored therein to be compromised.

Finally, it should be noted as well that it is unfortunately not possible within the scope of this thesis to reproduce here the entire code base of the component or to list in their entirety the sample exports that are generated to demonstrate the complete functionality and operability of the component. This, along with further information and materials about MOnSTER, is available online (Stenzhorn, 2022).

Part II

Materials and Methods (Implementation)

Chapter 4

General Overview

The general approach applied for the development of MOnSTER follows a strongly engineering-oriented scheme and therefore adheres to several principles, which may not have a pronounced scientific relevance by themselves, yet are of great importance when realizing a software artifact intended for any productive use. The intention that lies behind this is simple: The goal is not just to develop a prototype for demonstration purposes, but rather to create a robust, secure and powerful software artifact that can be readily deployed in real-world usage scenarios. The component itself should not only be a research object, but rather a vehicle that enables research based on it.

Therefore, a major guiding principle in the development is to follow established best practices and proven approaches as much as possible, and to avoid the "reinvention of the wheel", by employing both established architectural, design and implementation patterns (Bloch, 2018; Fowler et al., 2011), as well as widely-used and strongly supported technical tools, environments and infrastructure components.

Another guiding principle is to develop MOnSTER in such a way that it integrates itself seamlessly into its ObTiMA host, both in terms of the visible interface and the realization of the underlying back-end. It is also precisely for this reason that the development of this component can be neither considered nor described independently of its embedding application. Thus, the implementation aspects of ObTiMA that directly pertain to the functioning of MOnSTER are discussed in the below as well.

To render the above more tangible, the following two more specific points arise from what is just stated:

Separation of Concerns ObTiMA is realized as a classical web application with a clear separation of concerns between front-end and back-end, implementing a common multi-tier, client-server architecture. Thus, all interactions with the application occur on the client, that is, on the user's machine only through a browser-based interface, and

all of the underlying processing and storage happens (remotely) on dedicated servers.

This physical separation also implies a further logical, task-based separation into distinct layers for presentation and interaction, application logic, and the management of data. Here, the goal is to allow each layer to be developed independently, without internal changes within one layer affecting the others. To ensure this, however, each layer's concrete functionality and interfaces must be clearly and unambiguously defined to avoid unintended dependencies.

Modularity Orthogonal to the that separation, the application's ObTiMA's architecture also relies strongly on modularization, that is, the logical and technical grouping and encapsulation of functionality into distinct modules and more specific submodules. For example, all functionality related to trial management is bundled into one general module, which in turn contains submodules for each specific task area, such as the management of users within a trial. Almost more than with the separation above, great care must be taken to clearly define and delineate the individual modules and their respective functionality, and, of course, define their interfaces properly as well.

In this context, however, MOnSTER might be regarded as a somewhat special case: For example, creating a CRF (template) is quite different from actually entering data in a CRF (instance), and thus both are implemented in the different modules for trial and patient management. This component, on the other hand, can be considered cross-module, as its functionality is (also visually) embedded into several other modules, as shown in Figure 4.0.1. For the trial management and export in particular, MOnSTER's GUI elements merge with the existing ones, so they are hardly perceivable as independent. Therefore, in this case, it becomes even more important to create well-developed interfaces to guarantee proper inter-module interactions.

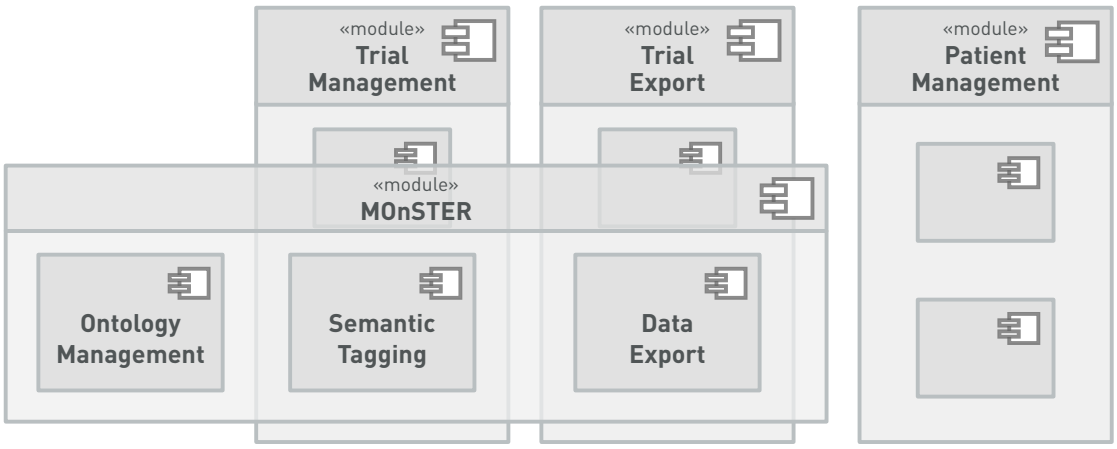


Figure 4.0.1 – *MOnSTER modules embedded in ObTiMA's core modules*

The sections below now present how these two aspects are realized within the implementation and set-up of both the application in general and the component in particular. To support the description, visualizations and graphical models are presented based on the UML (Unified Modeling Language) (OMG, 2017), as it represents the de-facto standard for specifying, designing, and documenting software systems.

4.1 Infrastructure

As stated in the above, ObTiMA represents a traditional web application, which entails that the infrastructure requirements are very moderate on the side of the actual user: To access and use the font-end of the application, all that is required here is a recent browser which supports the common web standards, and an Internet connection with access to the server on which the application's back-end is running.

As for the server side, the necessary environment is naturally more complex and involves some general, so called, web (application) stack components, that comprise

- the operating platform(s) for hosting the server infrastructure,
- the programming language and libraries for implementing the web application,
- the database server for storing the application's data, and
- the web application server for actually running the application.

In each case, for the selection of suitable solutions, their availability as open source software (OSI, 2007) is an essential factor for several reasons, among which are

- a better and quicker access to the latest innovations, technologies and tools,
- readily available documentation and direct support by the community,
- more robust and secure solutions via an active, open development approach,
- a lower total cost of ownership due to free availability without license fees.

Taking these considerations into account, the following solutions were chosen for use in the development and deployment of both the application and the component:

Programming Language / Libraries The Java programming language (Oracle, 2022a) is chosen as the underlying core of the application's implementation. The two main reasons for this choice are, first, that Java is used in industry in many non-trivial, real-world situations, and second, many (open source) software development libraries exist with predefined APIs and components. The descriptions given in Section 4.2 and Chapters 6 to 7 show the concrete use of such libraries in the development.

Web / Application Server A suitable server must support and implement both Java itself, as well as all relevant web standards and Java-based web application specifications (Eclipse, 2022b). Another stringent requirement for such a server is, of course, that it must be robust, provide adequate security mechanisms, and be scalable for a large number of simultaneous users and requests. These requirements are fully met by the two alternatives Apache Tomcat (ASF, 2022b) and Eclipse Jetty (Eclipse, 2022f). Even though both can be readily applied in the given context here, the first is chosen as the default as for its wider usage in the community.

Database Server The application's persistence and storage mechanism requires a database server as its base that follows the traditional approach based of a RDBMS (Relational Database Management System) and the SQL (Structured Query Language). Here, too, a solution needs to be industry-ready, that is, it must be robust, secure, and scalable both in terms of large volumes of data and concurrent queries. Of the two best suitable candidates, that is, PostgreSQL (PostgreSQL, 2022) and MySQL (Oracle, 2022b), the first one is selected as it adheres more closely to the standard

SQL, provides better concurrency handling and provides better licensing conditions.

Operating Platform Since both Java, the web application and database servers, as well as all development environment tools, are available for the most common operating systems, which are macOS, Windows and the different variants of Linux, the basic infrastructure can run on all of these systems. Therefore, also all three are applied as working platform for developing and testing the application. For the productive scenario, however, the server variant of Ubuntu Linux (Canonical, 2022) is applied, as this operating system is also well established in many mission-critical commercial environments within the industry.

Coming back to the aforementioned separation of concerns, it is worth noting too that in real-world, production usage scenarios of the application, a physical separation of both the actual (web) application and the database is enforced in the backend as well: To improve and increase both performance and scalability, as well as security, the application server and the database server run on two distinct machines, each tightly secured with, for example, its own encrypted filesystem and user management, and linked with each other via a highly secured connection.

4.2 Application Layers

In order to be able to implement the modularity and separation of concerns mentioned in the previous section, it makes sense, or rather is necessary, to divide the architecture and implementation of the application into different layers, whereby each layer comprises a concrete, larger task area. In this context the layering orients itself at the so-called MVC (Model-View-Controller) pattern (Fowler et al., 2011), which basically breaks a software artifact down roughly into the three components data model, presentation or view, and program controller, each of which is described below.

4.2.1 Persistence / Storage

The task of this layer is to provide the application logic layer with the necessary methods and mechanisms to persist, retrieve and supply data. That is, it includes the actual database (system) for storing data, the internal programming code to access and query

the database, and the external interface for the application logic layer.

The actual database is running on the PostgreSQL system (PostgreSQL, 2022), selected as it represents one of the most established open source RDBMS (Relational Database Management System) at this time and as it is also widely used in commercial environments with non-trivial requirements towards robustness, security and scalability.

For the interaction with the database, it was decided against the traditional approach of formulating queries directly via SQL (Structured Query Language) statements and embedding them in Java code. Instead, the ORM (Object Relational Mapping) approach (Ambler, 2013) was chosen, where object-oriented information models and structures are automatically mapped and transferred to tables and records in a RDBMS. This has the advantage that no low-level, and often complex SQL statements need to be created manually, but instead the familiar programming paradigm using regular objects can be applied. For the concrete realization of this approach within the application, the JPA (Jakarta (previously, Java) Persistence API) (Eclipse, 2022c) was chosen, which provides a foundational ORM API specification for Java, along with Hibernate ORM (Red Hat, 2022), which acts as a so-called JPA provider, implementing and making available the API as an open source software library. As with PostgreSQL, Hibernate ORM is selected on for its wide use, also in industry, and its large open source community.

Using this framework, all data is then represented through entity classes. These classes form the basis for so-called "plain objects", as they do not provide any functionalities themselves, that is, callable methods to execute processes, but only contain fields in which individual data elements can be stored. As an example, in Figure 4.2.1 the classes to represent a trial, its SEs and its CRFs are shown: Unlike in some other CTMS, this approach means that it is not necessary to create new databases and database tables specifically for each new trial, SE or CRF. Rather, in this approach only single, generic tables are created in the database with one for all trials, one for all SEs and one for all CRFs. Within these tables, the individual studies, and so forth, are then each distinguished by identifiers (`id`), automatically generated and assigned by the Hibernate ORM framework. Now, if there are references between entities, such as here that a `Trial` entity contains a list of `StudyEventTemplate` entities, and this in turn contains a list of `CRFEventTemplate` entities, then this is automatically realized in the

background by the framework by mapping the underlying identifiers. The entities, or classes / objects, just mentioned here represent only a very small part of ObTiMA's more complex information model, which has the model of the ODM standard (CDISC, 2013) as its original foundation and starting point, as described in Section 7.2.

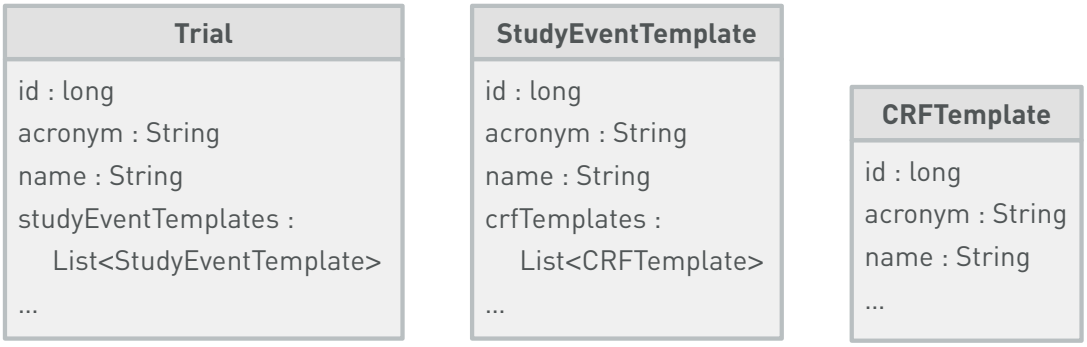


Figure 4.2.1 – *Trial*, *SE template and CRF template classes*

For the implementation of the mentioned interface to the application logic layer, the DAO (Data Access Object) pattern is used: In general terms, DAOs provide specific methods for searching, retrieving, or storing data, but encapsulate and make transparent the actual necessary steps and interactions with the database. A simple example of such a DAO is the `GenericDAO`, shown in Figure 4.2.2, providing generic methods to find, store and delete data (objects) in the underlying database. For example, the method call `find(Trial.class, "status", "RUNNING")` returns the list of all trials marked as running that are currently registered in the application.

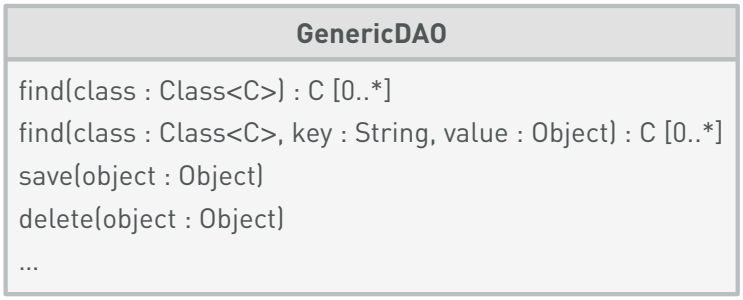


Figure 4.2.2 – *Generic DAO class*

For more complex queries, which usually include multiple attributes from multiple entities, special DAOs have been developed. As an example, to return all patients of a certain trial site, first the entity, and hence database table, holding all trial sites has to be queried for that particular site and then the entity / table with all patients for the ones linked to that trial site. In this still quite straightforward case, one could indeed use the `GenericDAO` and first query for the trial site and then use the result to query for the patients in turn. Although this would at least encapsulate the actual database access in the layer provided for this purpose, two queries would still have to be executed one after the other from the application logic layer and then linked to each other in this layer as well: This mixes both the actually intended tasks of the two layers and can potentially reduce performance. Therefore, it is sensible to create specialized DAOs where single, dedicated methods combine interrelated queries and subqueries and run those bundled on the database, as for the example within a `PatientDAO` into the specific method `findAllPatientsForSite(name : String)`.

4.2.2 Application Logic

This layer represents the business logic of the system, that is, fundamentally its actual functionality. Essentially, it receives requests from the user interface layer, performs processing on those requests, for that interacting with the persistence and storage layer to retrieve and store data, and finally sends a response back to the user.

Due to the fact that ObTiMA is a complex application covers a plethora of different tasks, the underlying application logic is accordingly also very extensive and complex.

Therefore, within the scope of this work, which focuses on the MOnSTER component, it is not possible to cover all of their specific aspects here.

However, one common point that affects all the components within the application equally is the aforementioned modularization. Since the correct implementation of such modularization is anything but a trivial endeavor, and as it forms the very foundation of all proper interaction between the various modules, this needs to be supported by a standard and robust environment and framework. It is for this reason, that the Spring Framework (VMware, 2022a) is selected. This concrete framework is chosen, as with the other software libraries applied in this context, based on the fact that it is open source, has long proven itself also in large industrial settings, and, of course, offers the specific functionality required. The most important, relevant capabilities provided in here are that this framework provides the ability to loosely couple individual modules, automatically resolve their dependencies and manage their lifecycles, as well as provide some unified APIs for their development and an overarching configuration management. Another advantage of this framework is its support of all libraries used in the other two application layers, which simplifies the development of interfaces to and between them.

For example, the `OntologyProvider` class, discussed in the next section, is provided as a so-called Spring "bean", which means that an object of this class is automatically created and configured by the framework on application startup without any additional manual effort. To access the database, the class / object internally requires the aforementioned `GenericDAO` object as interface to the persistence and storage layer. Here, the framework automatically checks whether such kind of object already exists and makes it directly available to the `OntologyProvider` or, if not, creates one and associates it with the provider class, again without any manual effort either.

4.2.3 User Interface

The user interface layer represents the actual interface for the users to interact with the application. Here, users trigger the underlying application logic to execute processes, that is, to perform the actual work, and present back the received outcomes.

Commonly, the term user interface is understood to mean the graphical user interface, or

GUI for short, only, as discussed in the below. However, in the context of MOnSTER, users can also access some of its functionality, namely to export trial data, through another interface, that is, via a web service, described in Subsection 7.2.2 and Section 11.2.

For realizing the GUI, the JSF (JavaServer Faces) is utilized, which is a Java-based specification for developing component-oriented user interfaces for web applications (Eclipse, 2022d) and additionally provides a reference implementation library of this specification encompassing all the GUI components described therein (Eclipse, 2022g). As with the libraries and tools before, JSF is selected because of its standardization, open source availability, and proven applicability also in large commercial applications.

Further, on top of this, the PrimeFaces library (PrimeTek, 2022) is employed, providing some additional JSF components for GUI development. This library is chosen since its focus lies on providing components and elements that allow the creation of dynamic, responsive interfaces for applications with a high degree of user interactions, as it is exactly the case for both ObTiMA and MOnSTER.

The libraries are used concretely in the implementation as follows: For the actual view of the application, that is, what is presented to the user in the browser, a set of multiple (web) base pages is created, such as one for providing and presenting the application's trial management functionality or for the functionality to add, list and edit ontologies. These pages are based fundamentally on HTML (WHATWG, 2019), yet they are not static as in the case of regular HTML pages. Rather, they use the provided JSF built-in mechanisms and contain this frameworks (component) elements to import and integrate different subpages or subviews, like in the trial management the (sub)view to manage the trial's general parameters or its CRFs and SEs. They are also dynamic in the respect that the actual HTML of all interaction elements provided by both JSF and PrimeFaces is not hardcoded in advance but generated dynamically by the libraries and thus linked to the defined methods in the back-end layers of the application. Thus, doing do enables that when an interaction element on the trial management page is activated, such as by clicking the Save button, in this case, all relevant data on that page is automatically transmitted using the frameworks mechanisms to the application server and processed in the application logic layer by the appropriate class and method, such as in this case the class `TrialHandlerAction` and its method `saveTrial`.

4.3 Usage Scenarios

In software engineering, before any architectural or implementation work can be carried out, it must be clearly and precisely (pre)defined which actual usages and usage scenarios are envisaged for some planned piece of software. So the goal of setting-up such scenarios here is to describe at a more abstract, higher level what kind of overall functionality the component should provide at all and how this can be decomposed into more specific, individual "work packages", and to define what tasks and features these packages should encompass in turn. As their name implies, these scenarios are usage-centric and thus describe from the user's point of view which concrete tasks the user should be able to perform when using that piece of software. It should be noted that the fundamental requirements, such as the possibility of concurrently using multiple ontologies in the application, as mentioned in Chapter 3 before, are to be implicitly included as well when defining such scenarios.

Basically, the following three scenarios, as visually summarized in the diagram in Figure 4.3.1, are distinguished when using the component. On their basis, and according to the earlier described modularization approach, the component is in turn logically and technically divided into three subcomponents, one for each usage scenario.

- To fundamentally enable semantic interoperability in the application, the ontological base needs to be enabled first. Hence, in order to be able to use ontologies at all, they must, of course, be available in the application. It is for this reason, that the first scenario deals precisely with how ontologies can be added by some user to the application in general and how existing ontologies can be managed, edited and removed if necessary.
- The second usage scenario addresses the goal of semantic interoperability in the sense that semantic information can be added to the various components of a trial. For this, it needs to be ensured first that ontologies are also available for the specific trial at hand and then, in turn, that all individual trial components can be readily semantically enriched and tagged with the desired semantic information based on concepts from the available ontologies.
- Finally, to also ensure syntactic interoperability, all (semantically enriched) trial data

added to and stored within the application must be made available to further, external systems in the appropriate formats, as described before. This scenario therefore includes exactly the functionalities required by the user for this purpose, that is, who trial data can be exported in different formats.

Now, to ensure the clarity of this thesis documentation, the division is also taken up in all of the following, so that the respective descriptions of the implementation, the results and the evaluation are all structured according to the given three scenarios.

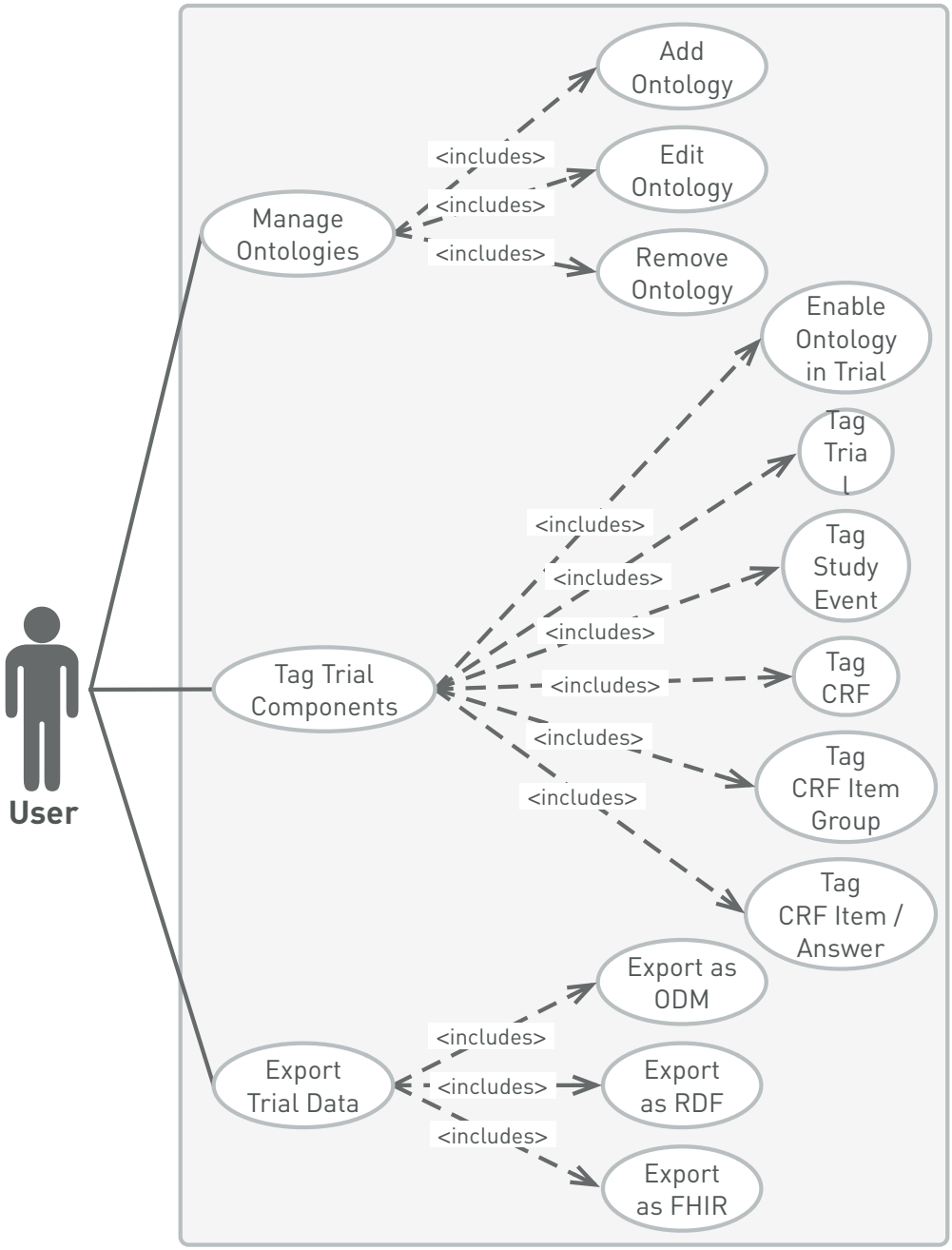


Figure 4.3.1 – *Usage scenarios for the MOnSTER component*

Chapter 5

Ontology Management

To be able to perform any ontology-based semantic tagging with MOnSTER at all, the desired ontologies need to be made available in the application first. Therefore, the task of this subcomponent is to provide the functionality to add ontologies to the application, to manage and edit them within it, and to remove them again from it if necessary.

The implementation of its presentation layer, that is, the necessary GUI, follows ObTiMA's standard approach and relies on several predefined JSF and PrimeFaces elements, which are customized and configured for their use by the component. Here, the main elements are the modal form popup for adding and editing an ontology, the dynamic data table for listing the available ontologies, and the confirmation dialog when removing of an ontology. The actual implementation encompasses the creation of a dynamic HTML page in which the just named elements are embedded and configured to interact with the respective back-end layer on the server. That is, for example, when clicking the `Save` button during the editing of an ontology, the respective `save` method in the back-end is called, which receives the data from the client (browser), validates the provided data and either, on success, the ontology is saved, or on failure, an error message is sent back to the client and there presented to the user. A detailed description of these resulting elements is given in Chapter 9.

For this presentation to function, the underlying processing is performed in the application layer by the `OntologyProvider` class, as shown in Figure 5.0.1, which provides the necessary (interface) methods for both saving and indexing ontologies, as well as for removing and deindexing them, together with the internal (non-interface) methods that are needed to process ontologies, validate them, and so forth.

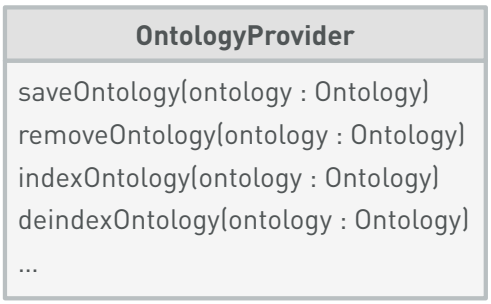


Figure 5.0.1 – *Ontology provider class*

The basic attributes of an ontology are represented within the persistence and storage layer by the `Ontology` entity class, depicted in Figure 5.0.2(a), along with the supporting `Namespace` class in (b) and the `Format` enumerator in (c). The specific meaning and use of each attribute is described within the results in Chapter 9. The implementation utilizes JPA again as primary mechanism for mapping this class onto the corresponding database tables. Since there are no complex queries required for storing and retrieving an ontology and its attributes, there is also no need to develop a specialized DAO, but the ObTiMA core `GenericDAO` can be readily used here as well.

At this point, it is important to emphasize that the approach on how an ontology is processed and stored can be seen as “hybrid”: The just described is concerned with the basic (metadata) attributes of an ontology only. However, for performance reasons, the processing, storage and retrieval of the actual content of an ontology, that is, its concepts, does not rely on JPA and the database, but rather employs a specific indexing approach. The background for this is that, especially for fast sequential queries, this functions much more rapidly using a (locally) available index together with a suitable high-performance search engine, instead of repetitively querying a possibly remote database.

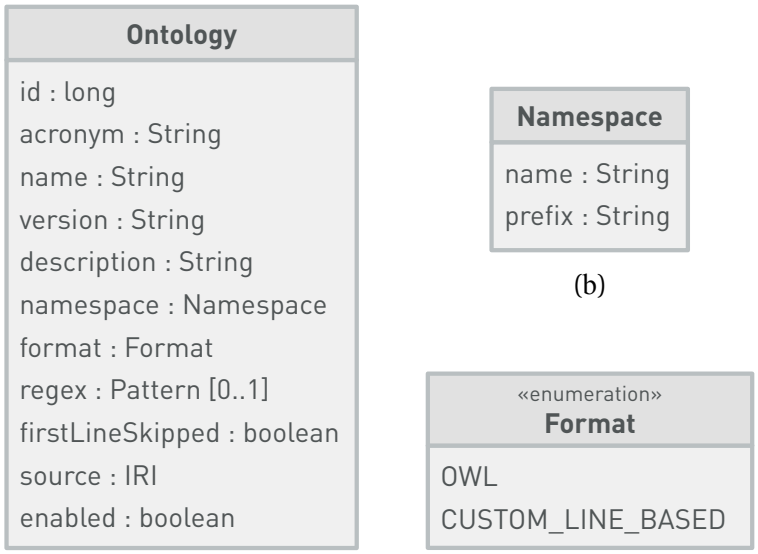


Figure 5.0.2 – *Ontology-related base classes and enumeration*

The implementation of this index is realized via Lucene, a widely used, open source software library providing full-text search capabilities, familiar from the major Internet search engines (ASF, 2022a). Now, since the concepts are searched for via their respective labels, within this index, these form the actual lead search element, or field in the library's naming scheme. That means that for each label of a concept, a so-called document with the fields in Table 5.0.1 is created and indexed, as explained below. (The search on this index is described in the next chapter.)

Field	Description
`code`	Unique identifier of the concept within an ontology, as specified in it
`ontology`	Unique identifier, that is, ObTiMA's internal system `id` of the ontology containing the concept, as shown in Figure 5.0.1
`label`	Single label of the concept
`language`	Language identifier of the label, following the ISO 639-1 standard (ISO, 2002)
`preferred`	Marker whether the label is a preferred one

Table 5.0.1 – *Concept index fields*

To now combine all the above into a workflow, the following steps are executed when adding an ontology to the system:

For providing an ontology file to the system, MOnSTER gives the user three options: It can be either manually (pre)loaded onto the server in advance, uploaded there by using the GUI, or a remote location can be specified, and the file is automatically downloaded from there to the server. The implementation of the upload uses a PrimeFaces component, the download of remote ontology files the built-in standard Java HTTP client.

After loading an ontology onto the server, it is in turn further processed according to the given format of the file, as described below.

OWL-Based Ontology Files

Here, the file can be read directly into an in-memory OWL model using the OWL API library (Horridge & Bechhofer, 2011; Horridge et al., 2022). In this model, all concepts are encoded as OWL classes and corresponding labels via annotation properties. To ensure that only relevant concepts are processed, only those fitting the predefined namespace are considered, see Chapter 9.

For the associated labels, the standard RDFS label annotation property (Brickley et al.,

2014) is included, as well as the ones for preferred and alternative (synonym) labels defined by SKOS (Isaac & Summers, 2009). Here, each label can optionally contain an additional identifier for its language.

For indexing, the matching OWL classes are iterated over, and for each class in turn the matching labels / annotations are iterated over, so that a corresponding Lucene `Document` object is generated with the appropriate field attributes, as shown in Table 5.0.1, and added to the index. Also, either the first label of an OWL class is marked as preferred or, if existing, the one marked as SKOS preferred label.

An example excerpt from an OWL ontology file is provided in Section 9.1.

Custom Line-Based Ontology Files

In this case, the processing of the ontology file is performed on a line by line basis. Custom line-based files often take the form of a CSV (Comma Separated Value), or more generally, a DSV (Comma Separated Value) file, that is, a comma or another specific character is used to separate values from each other on a line. But it is also possible to use files in this context, which use a more complex line structure, which cannot be readily mapped using CSV or DSV. For this reason, MOnSTER's implementation is based on applying regular expressions to extract the required parameters from each line. The concrete structure of such an expression is described in Section 9.1.

The implementation relies on Java's core APIs for file input and output, as well as for regular expressions and pattern matching: With these, the ontology file is read in line by line in a so-called streaming mode for efficiency reasons. As soon as new line is read into memory, the pattern matching on this line with the given regular expression is performed. If the required elements are found, that is, the code and label of a concept, and optionally a language tag and a flag whether the given label is the preferred one, then these are extracted. A Lucene document is created from the extracted elements, as in the case above for OWL ontologies, and, in turn, this document is added to the index. For this as well, an example for a custom line-based ontology file together with the necessary regular expression is given later in Section 9.1.

Chapter 6

Semantic Tagging

After one or more ontologies have been added to the system, the concepts contained therein can be used to semantically tag trial-related elements.

Again, the presentation layer's implementation uses the default approach of ObTiMA using specifically adapted JSF and PrimeFaces elements. The main component in here is an autocomplete element where search terms can be entered to look up ontology concepts based on their labels, and which displays a dynamically updated list of matching concepts from the predefined ontologies, as presented in Chapter 10. Here, the dynamic update always happens immediately when the query string in the auto-complete's text field changes, that is, some characters are entered or deleted again. This functionality is implemented through a server-side callback method which is executed whenever such change within the query string occurs. The `completeTag` method, found in the `TagProvider` class and shown in Figure 6.0.1, performs the three steps:

1. Create a Lucene search query,
2. Search the index with this query, and
3. Wrap the results in `Tag` objects, as depicted in Figure 6.0.2, and return them.

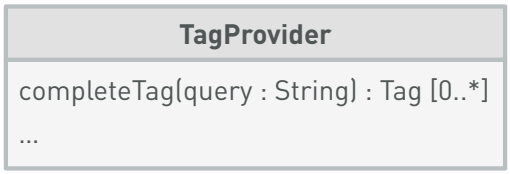
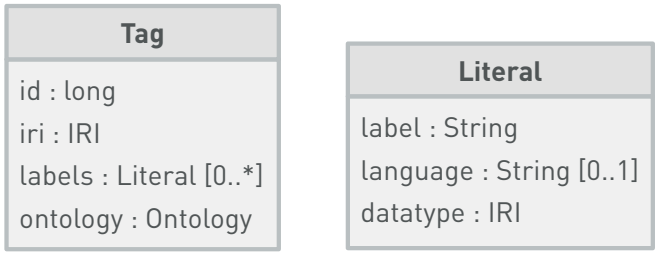


Figure 6.0.1 – *Tag provider class*



(a) (b)

Figure 6.0.2 – *Tag-related base classes*

The generation of the search query is also more complex and involves four steps:

1. A subquery is created for selecting all `Documents` in the index where the `ontology` field is equal to the identifier of one of the preselected ontologies.
2. The query string in the auto-complete text field is tokenized based on the spaces it contains, and for each resulting token, that is substring, a subquery is generated selecting all `Documents` in which the `label` field contains this token.
3. The last step's subqueries are combined into an boolean "or" query which returns all index `Documents` that match at least one of the subqueries, that is, contain at least one of the tokens in the `label` field.
4. The very first subquery is again combined with the last step's boolean "or" query to form a boolean "and" query, which means that now only the `Documents` are matched whose `ontology` belongs to the preselected ones and at least one of the tokens is included in the `label`.

Based on this combined query, the search is performed in the index and the list of all matching `Documents` is returned. In order to simplify the subsequent processing of the list, the resulting `Documents` are already sorted by the Lucene engine during the search process: To do this, the actual sort fields are passed directly to the `search` method. Here the sorting is done according to the combination of the specified fields in the specified order. This means, that the results are sorted according to the `ontology` field, and hence grouped for to the respective source ontology. Then the results are sorted by preferred and non-preferred labels based on the `preferred` field. Finally, the `label` field is used to sort the results alphabetically by the labels.

The next step is to wrap the results in `Tag` objects, or rather, create those objects from the individual returned Lucene `Document` objects. In order to send these objects to the client, that is, to the browser GUI, they are automatically serialized into JSON format by the PrimeFaces library. From that, the autocomplete component generates the drop-down menu with each row based on a single (JSON-serialized) `Tag` object.

Now, when the users clicks on a result row, the `Tag` is added to the trial component. So the storage of the respective `Tag` object is triggered, persisting it in the database using ObTiMA's `GenericDAO` and added to the `tags` list attribute of that component's object. This link is also reflected directly in the GUI, as the (visual) tag element is now shown in the tags element of the trial component.

If the user subsequently removes the tag from the trial component on the GUI, the reverse happens, namely, the `tag` is removed from the corresponding list of that component using the given DAO and then deleted from the database itself.

Chapter 7

Data Export

The realization of the data export is a key aspect in achieving MOnSTER's goal to promote both semantic and syntactic interoperability.

Here, semantic interoperability aims not only at achieving a common conceptual understanding via ontologies, but it also addresses the distinct motivations behind ODM, RDF and FHIR: While the first targets the exchange and archiving of trial data, the second provides an open, generic model to exchange (basically) any kind of data, and the third aims at becoming the standard for data exchange in healthcare.

By embracing all these formats, MOnSTER also seeks to bridge the conceptual gap between them and facilitate the integration of trial data with (large) linked datasets from the Semantic Web (Egaña Aranguren et al., 2014; Kamdar et al., 2019) or with data from routine clinical systems (Lehne et al., 2019) to foster research.

7.1 Basic Implementation

The implementation of MOnSTER's export features builds upon the existing export functionality, which represents one of the core functionalities of the application from its very outset. Its central access point in the application logic layer is the `ExportAction` class with its `export` method, as depicted in Figure 7.1.1(a). Previously, this method could only prepare and produced results in the ODM format, but for this work it is extended to return RDF and FHIR as well. It performs its task in three steps:

1. All relevant data elements for the given trial are looked up and retrieved using the corresponding DAOs from the persistence and storage layer.
2. The data is mapped and transformed into the information model of the stated format, that is, one of (b), by the respective mapper, as described in Section 7.2.
3. The mapped data is returned in the specified, format-specific serialization, namely

one of (c), as presented in Section 7.3.

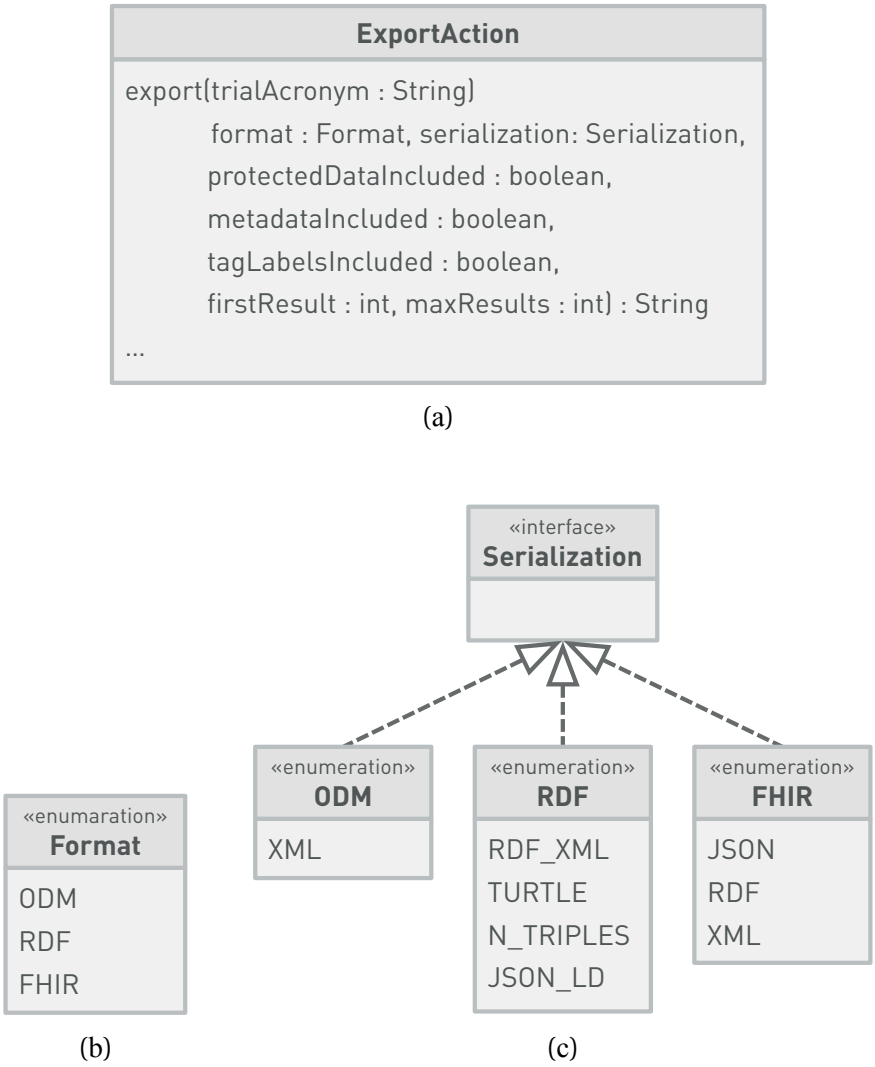


Figure 7.1.1 – *Export action with format and serialization enumerations*

7.2 Format Mappings

Referring to the previous section, the second step which the `ExportAction` class' `export` method needs to perform is the format mapping, that is, to link and transform all relevant entities in ObTiMA's internal information model to the corresponding entities

in the information models of the ODM, RDF, and FHIR export formats.

In order to accomplish this task, the following general steps are performed:

1. Each information model is visualized by an ERM (Entity Relationship Model), a type of flowchart that graphically represents the entities of a domain together with their relationships and dependencies (Elmasri & Navathe, 2015).
2. Based on these visualizations, all relevant entities and contained attributes are identified within each of the information models.
3. Also using the visualizations, correspondences between the identified entities and attributes in the internal and the target information models are determined.
4. The required transformations for found correspondences are encoded first as pseudocode (Cormen et al., 2009) and then realized as programming code.

The ERM visualization of ObTiMA's internal information model is shown in Figure 7.2.1, for whose creation reverse engineering is applied: As the existing information model is implemented based on JPA and Hibernate entity classes, the functionality provided by the IDE IntelliJ IDEA (JetBrains, 2022) to automatically produce a corresponding ERM in the UML standard notation is used.

In this visualization, each named box represents an entity, and a line between two indicates that the entity marked with a diamond contains (or may contain) elements of the other entity. For example, one `CRFTemplate` holds zero or more (`0..*`) `Tags`, that is, a CRF may be tagged with one or more semantic tags, or not at all.

If the individual connections between the entities of the test components are now followed from top to bottom, a strongly nested structure emerges here. This means, for example, that a `Trial` contains one or more `StudyEventTemplates`, and each `StudyEventTemplate` in turn contains one or more `CRFTemplates`, and so on.

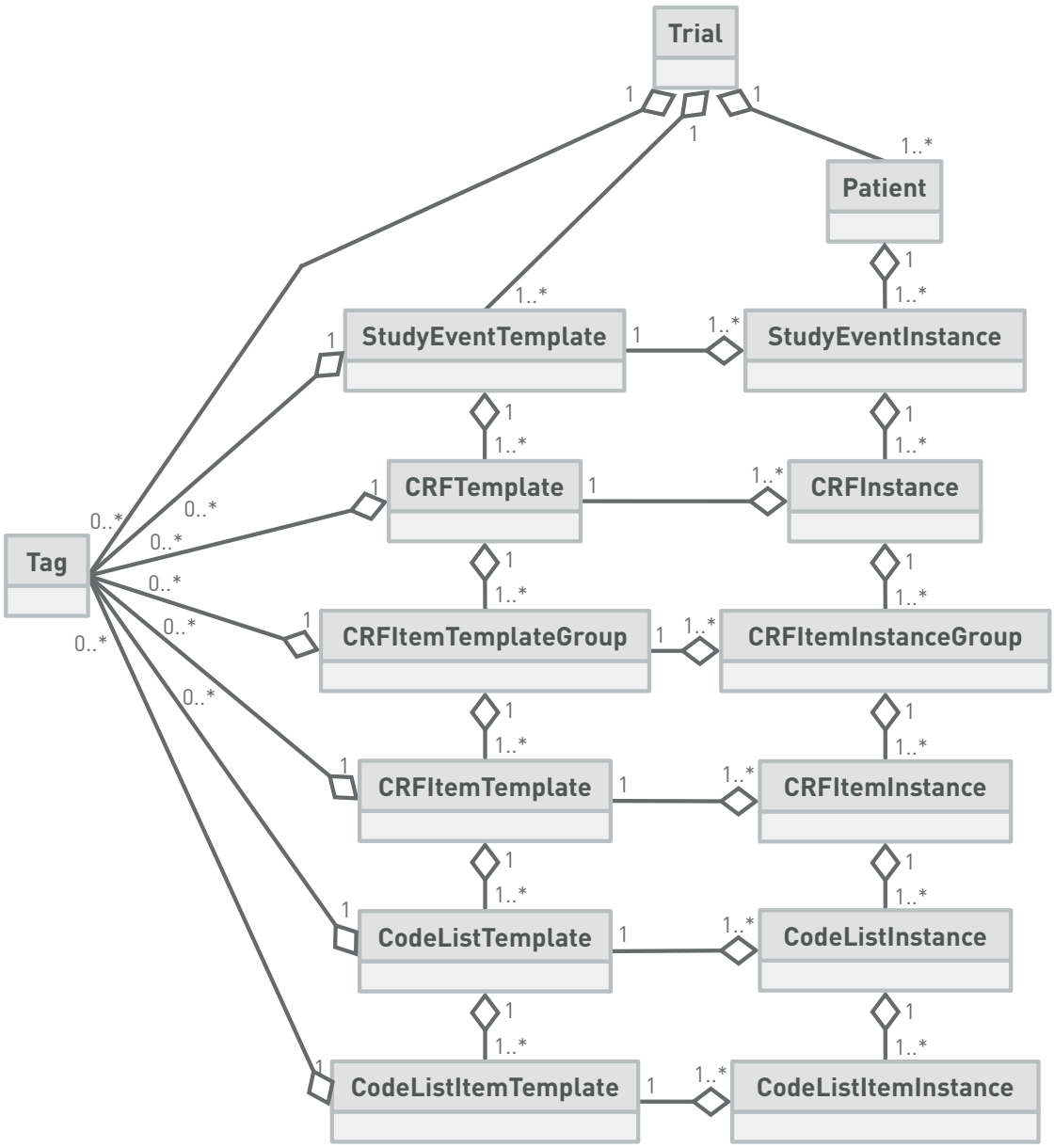


Figure 7.2.1 – *Entities of ObTiMA's internal information model*

Therefore, to map this structure and hence the information model, this processing must occur top-down too. So, staying with the above example, a `TrialMapper` first processes a `Trial` and for each `StudyEventTemplate` contained therein, calls a `StudyEventTemplateMapper` to process it, and so forth. Importantly, since each

target format's data modes are different, the mappers are implemented specifically for each of them. Their realizations are described in Subsections 7.2.1 to 7.2.3.

As an additional note, it should be mentioned here that within the internal information model instances are always based on and thus linked to an appropriate template. Therefore, as an example, `CRFInstances` are always associated with a specific `CRFTemplate`, that is, the latter contains all question definitions and the former the related response values and data. This must also be taken into account when creating the mapping, since the dependencies between the instances and the respective templates must also be reflected accordingly in the target model.

In the Figure 7.2.2 below, an overview is given of the general outline and structure of the classes necessary to map each part of a trial from the internal information model to the target one. For each target model, the respective, suitable mapper classes need to be developed. They are described in the sections to follow.

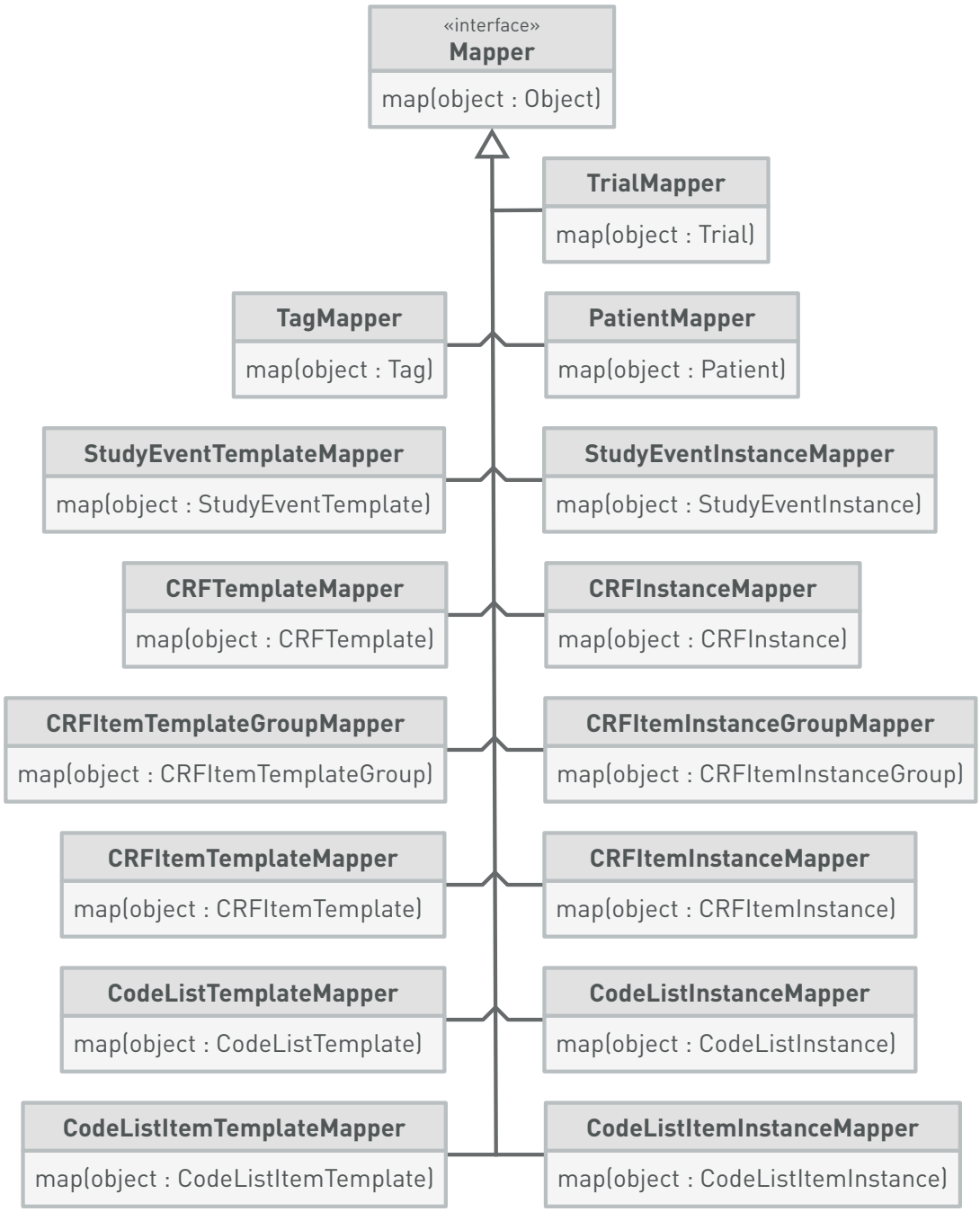


Figure 7.2.2 – *General outline and structure of the necessary mapper classes*

7.2.1 ODM (Operational Data Model)

In the early development phase of ObTiMA, the ODM format as specified by the CDISC (Clinical Data Interchange Standards Consortium) (CDISC, 2013) served as a fundational blueprint when defining the application's trial-related entities. It is against this background that the two underlying information models are still very similar and agree in many aspects, often differing in many places only by a different naming convention. For this reason, the realization of both logical mapping and programmatic implementation prove to be straightforward and attainable with moderate effort and resources.

The technical specification of the ODM standard provides, in addition to a textual description, a set of XSD (XML Schema Definition) files (Gao et al., 2012; Peterson et al., 2012) to formally define all elements of the ODM XML serialization (CDISC, 2013). An initial visualization of the XSD is automatically generated using the Oxygen XML Editor (Soft, S., 2022), which in turn is manually transformed into UML notation. Based on the latter together with ODM's textual description, the correspondences between this information model and the one of ObTiMA are determined and the respective pseudocode developed to transform between them. An example of a direct correspondence found is shown in Figure 7.2.3, where ObTiMA `CRFTemplate` corresponds directly to ODM `FormDef`, and ObTiMA `CRFInstance` directly to ODM `FormData`, where basically only the respective entity name needs to be mapped and also the attributes of each are highly similar and therefore straightforward to map.

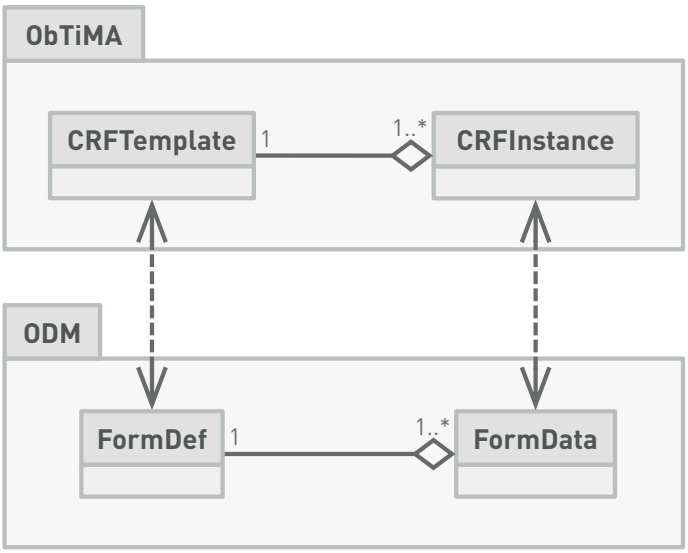


Figure 7.2.3 – *Example for a direct correspondence between ObTiMA and ODM entities*

For the programmatic implementation of the mapping, a workaround is introduced, since no official ODM Java library exists providing the standard's elements as dedicated Java classes: Applying the JAXB (Jakarta XML Binding, formerly Java Architecture for XML Binding), a standard API and library for mapping classes and objects to and from XML (Eclipse, 2022a, 2022e), is used to automatically generate Java classes for each ODM XML element. This representation as standard classes enables Java's regular object-oriented approach and mechanisms for instantiating objects and setting, reading, and modifying their attributes.

The format-specific realization of the mapping now uses these entity classes and implements a suitable mapper class for each original ObTiMA entity, proceeding again from top to bottom adhering to the information model's nested structure.

This mapping now also includes MOnSTER's semantic tags, which are not part of the original ObTiMA information model. To include them in the ODM export, this standard's extension mechanism is employed: With the help of such extensions, further elements can be added to ODM to represent data (types) not covered by the actual standard. The creation of this MOnSTER extension involves two steps:

1. An additional XSD is developed, where the necessary elements together with their attributes and dependencies are defined.
2. ODM's default extension XSD is adapted to include a reference to the MOnSTER XSD and so to allow all adapted elements to contain tags.

Based on JAXB, the appropriate classes are automatically generated again, and the respective `TagMapper` developed. The complete MOnSTER together with the ODM extension XSDs are listed in Appendix A.

For example, in the (simplified) code snippet in Figure 7.2.4, the `map` method of the `CRFTemplateMapper` class initially generates a ODM `FormDef` object and sets its `name` attribute to the respective value from the original ObTiMA `CRFTemplate` object. Now, for all `ChildCRFItemTemplateGroup` objects that the `CRFTemplate` object holds, the `map` method of the `CRFItemTemplateGroupMapper` class is called, and the `FormDef` object's `itemGroupRef` attribute set to the collected result list. Finally, the same is applied using the `TagMapper` class' `map` method to process the contained `Tag` objects. The resulting `TagElement` objects list is then wrapped inside a `TagsElement` object and added to the `FormDefElementExtension` list attribute.

```
FormDef map(CRFTemplate crfTemplate) {
  var formDef = new FormDef();
  ...
  formDef.setName(crfTemplate.getName());
  ...
  formDef.setItemGroupRef(
      crfTemplate.getChildCRFItemTemplateGroups().stream()
                 .map(crfItemTemplateGroupMapper::map)
                 .collect(toList()));
  ...
  formDef.getFormDefElementExtension().add(
      createTagsElement(crfTemplate.getTags().stream()
                                   .map(tagMapper::map)
                                   .collect(toList())));
  ...
  return formDef;
}
```

Figure 7.2.4 – *Example mapping of an ObTiMA entity onto an ODM entity*

7.2.2 RDF (Resource Description Framework)

In contrast to the mapping of ObTiMA's internal information model to ODM, an additional step is necessary here: RDF, which was originally intended to describe metadata, now represents a cornerstone of the Semantic Web (Berners-Lee et al., 2001) and provides a generic method for creating and exchanging basically any graph-based information models (Cyganiak et al., 2014).

It is because of this generality, that an appropriate vocabulary must be utilized in order to apply RDF meaningfully to some particular task. The task of vocabularies in this context is therefore the (technical) definition of terms and their interconnecting relationships to describe and represent a specific area of interest.

Although there were efforts by CDISC (Clinical Data Interchange Standards Consortium) to develop such RDF vocabularies for some of its standards, this was not the case for ODM (PhUSE-CS-STWG, 2015; Williams & Oliva, 2017). For this reason, the development of MOnSTER also includes the creation of a vocabulary for modeling ODM in RDF. Here, the implementation is based on RDFS (RDF Schema) (Brickley et al., 2014) and closely follows the general structural and technical approach of PhUSE-CS-STWG; Williams and Oliva. Furthermore, the prototypical proposals of the W3C HCLS (Healthcare and Life Sciences) IG (Interest Group) (W3C-SW-HCLS-IG, 2012) are integrated in the vocabulary implementation as well and its development follow the good practice principles for Managing RDF vocabularies (Kendall et al., 2008). Fundamentally, the goal is to express the original ODM information model as accurately as possible in terms of a native, graph-based RDF model (vocabulary) without compromising the logic and completeness of the original ODM, but also to avoid workarounds contradicting RDF's fundamental approach and principles (Schreiber et al., 2014).

Since, as just mentioned, the developed RDF vocabulary represents the ODM entities very closely, the development of the logical mapping from ObTiMA's information model is straightforward, as the mapping already created for the original ODM can be used as a basis. From this follows that the abstract pseudocode from that mapping can be reused to a large extent as well.

The actual realization of the mapping must, of course, take into account the specifics

of RDF's triple statement approach to create (data) models (Cyganiak et al., 2014). For example, the natural language sentence "The researcher John Doe is PI in the trial UMBRELLA." could be split into the following three triple statements:

- "John Doe" is a resource of type "Researcher".
- "UMBRELLA" is a resource of type "Trial".
- "UMBRELLA" has the property "has PI " pointing to the resource "John Doe".

Now, for its realization, the RDF4J library (Eclipse, 2022h) is applied, based on which specific mappers for each ObTiMA information model entity are developed.

To take up the example from before, the `CRFTemplateMapper` iterates over each attribute of the ObTiMA `CRFTemplate` object, generating a triple statement for each and adding it to the RDF model: In the (simplified) code snippet in Figure 7.2.5, first, an IRI is created as identifier for the given `CRFTemplate`. Then, two statements are created and added to the model, the first one stating that the RDF resource identified by `crfTemplateIRI` is of `TYPE` (ODM) `FORM`, and the second one stating that this resource has a `NAME` whose values are taken from the corresponding attribute of the `CRFTemplate` object. Now, the `CRFItemTemplateGroupMapper` class' `map` method gets called for each `ChildCRFItemTemplateGroup` object contained in the `CRFTemplate` object, and the collected list added as `CHILDREN` to the RDF resource `crfTemplateIRI`. Finally, the same applies for all contained `Tag` objects, that is, the `TagMapper` class' `map` method is called for each, and the resulting list added as `TAGS`.

```
IRI map(CRFTemplate crfTemplate) {
  var crfTemplateIRI = createIRI(crfTemplate);
  ...
  addToModel(crfTemplateIRI, TYPE, FORM);
  addToModel(crfTemplateIRI, NAME, crfTemplate.getName());
  ...
  addToModel(crfTemplateIRI, CHILDREN,
      crfTemplate.getChildCRFItemTemplateGroups().stream()
                 .map(crfItemTemplateGroupMapper::map)
                 .collect(toList()));
  ...
  addToModel(crfTemplateIRI, TAGS,
      crfTemplate.getTags().stream()
                 .map(tagMapper::map).collect(toList()));
  ...
  return crfTemplateIRI;
}
```

Figure 7.2.5 – *Example mapping of an ObTiMA entity onto RDF statements*

It is worth noting here, that by using the RDF4J API to create the RDF it is always guaranteed that a syntactically valid model is generated in that process. For this reason, during the development of the (programmatic) mapping, the validity of the generated RDF is iteratively checked using the developed RDFS of the vocabulary.

7.2.3 FHIR (Fast Healthcare Interoperability Resources)

The underlying logic and structure of ObTiMA's information model differs to quite some extent to the one of FHIR, both logically and structurally. Therefore, the mapping is not entirely straightforward in all places and is not one-to-one for a number of entities from the original information model. Nevertheless, it is possible to find corresponding suitable entities in the target model for all source data entities and thus create a complete mapping in the case of FHIR as well.

Regarding the logical mapping, the current work is strongly based upon existing work where both the ODM metadata containing definitions of the trial data collection instruments, that is, SEs, CRFs, and so forth, as well as the actual collected trial data are

mapped onto the fitting FHIR entities, or resources in the FHIR naming scheme (Doods et al., 2016; Leroux et al., 2017). Although this work is focused on ODM's information model as input, it can support the development here, because of the aforementioned close match between the ODM's and ObTiMA's information models. In addition, further, recent work on the representation of clinical research using FHIR is also included in this work as well (Leroux et al., 2019).

Since the FHIR standard specification already represents its entities as UML diagrams, these can be directly adopted as a base for creating the mapping visualization. For this purpose, the above-referenced publications are evaluated and all FHIR entities and their connections mentioned in each of them are included in the visualization. The next step is to assess the mappings of ODM to FHIR described in the named publications and to relate and combine them with the mapping of the information model of ObTiMA to the one of ODM, as described earlier.

Now, albeit there are some differences between the ObTiMA and FHIR information models, the mapping from the first onto the second is possible without major difficulties. For example, a direct and complete match can be found with respect to the core entities required here: Looking at Figure 7.2.6(a), an obvious direct semantic correspondence between ObTiMA `CRFTemplate` and FHIR `Questionnaire`, as well as between ObTiMA `CRFInstance` and FHIR `QuestionnaireResponse` exists, as their respective purpose and content are equivalent. Another direct correspondence is shown in Figure 7.2.6(b), as the ObTiMA `Trial` entity can be readily mapped to the FHIR `ResearchStudy` one, since both provide the (metadata) information about some given trial as well as its overall structure.

However, even for entities in the original information model for which no readily obvious match exists, suitable entities and connecting links can be identified in the target model that allow all elements of the original information model entities to be mapped. For the example in (b), there are no semantically complete equivalents for ObTiMA's `StudyEventTemplate` and `StudyEventInstance` in FHIR. Therefore, as a "workaround", a FHIR `PlanDefinition` entity is generated to model the arrangement of SEs (definitions) in a trial, containing each SE template mapped onto an `ActivityDefinition` entity, and each activity, in turn, references the

Questionnaire entities mapped to from the SE's CRFTemplate entities of the original SE. Then for each ObTiMA StudyEventInstance entity a FHIR Encounter entity is created and linked to the corresponding ActivityDefinition. The encounter is subsequently added to a CarePlan, which is based on the respective, fitting PlanDefinition, and finally all QuestionnaireResponses mapped from the CRFInstances of the SE are linked to the encounter.

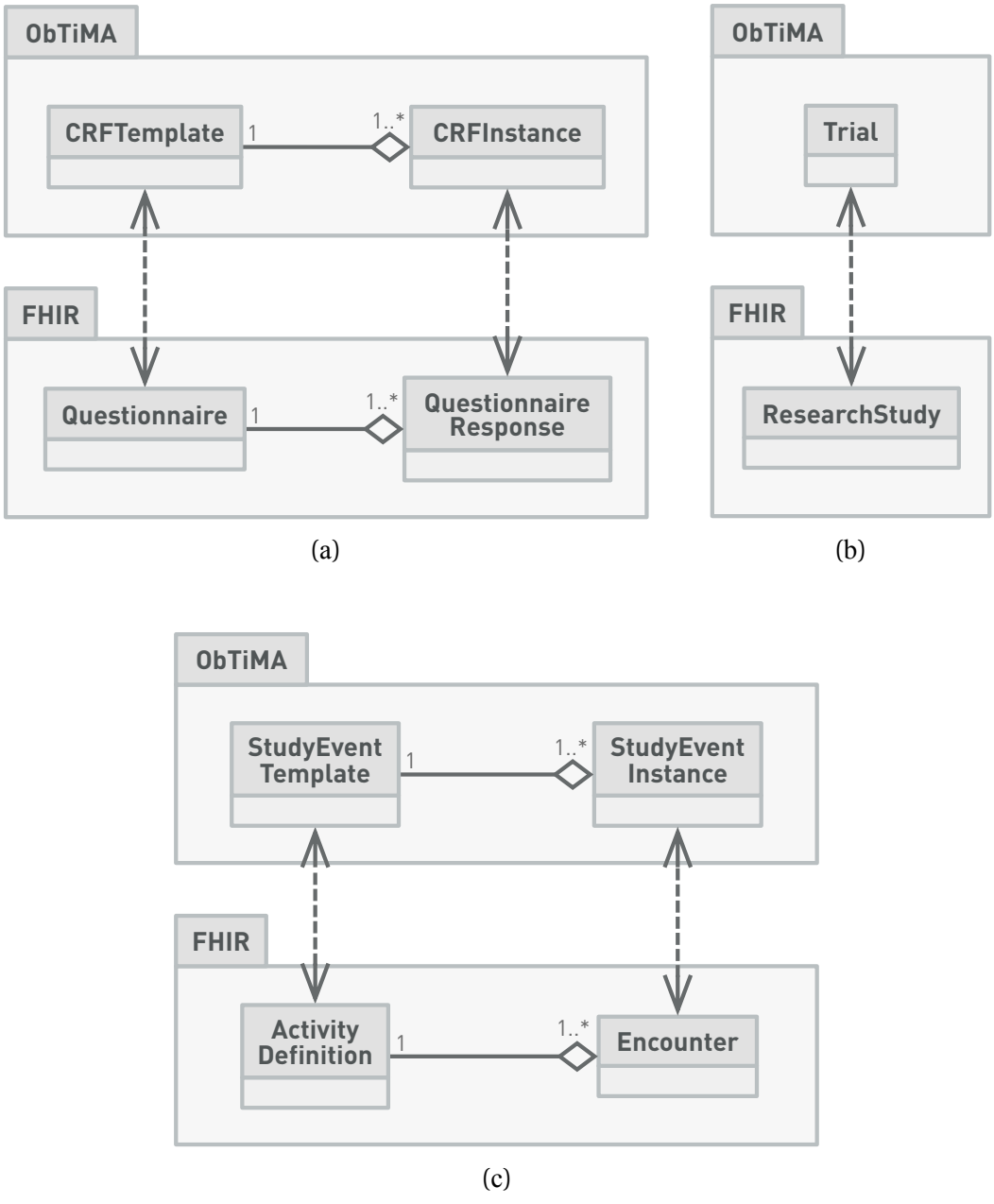


Figure 7.2.6 – *Example for correspondences between ObTiMA and FHIR entities*

For the programmatic implementation, the HAPI FHIR library (HAPI FHIR, 2022) is adopted, since it serves as the reference implementation for processing FHIR in Java and provides all entities / resources defined by the specification (HL7, 2019) as regular

(entity) classes. This allows for the development of the necessary mappers that the FHIR information model is expressed in For this reason, the object-oriented approach of Java and the mechanisms for creating and manipulating objects, that is, setting and changing their attributes' values, are also readily applicable here.

Reusing the previous example again, the `map` method of the `CRFTemplateMapper` class, as shown in the (simplified) code snippet in Figure 7.2.7, first creates a FHIR `Questionnaire` object and then sets its `name` attribute to the one from the ObTiMA `CRFTemplate` object. In the next step the `map` method of the `CRFItemTemplateGroupMapper` class is called for each of the `CRFTemplate` object's `ChildCRFItemTemplateGroup` object, and the collected result set as `item` (list) attribute of the `Questionnaire` object. Again, all contained `Tag` objects are mapped using the `TagMapper` class' `map` method and the result list added as `code` list attribute to the `Questionnaire` object.

```
Questionnaire map(CRFTemplate crfTemplate) {
  var questionnaire = new Questionnaire();
  ...
  questionnaire.setName(crfTemplate.getName());
  ...
  questionnaire.setItem(
      crfTemplate.getChildCRFItemTemplateGroups().stream()
                 .map(crfItemTemplateGroupMapper::map)
                 .collect(toList()));
  ...
  questionnaire.setCode(
      crfTemplate.getTags().stream()
                 .map(tagMapper::map).collect(toList()))
  ...
  return questionnaire;
}
```

Figure 7.2.7 – *Example mapping of an ObTiMA entity onto a FHIR entity*

Here, it must be noted that the above use of the HAPI FHIR API always ensures that the FHIR generated with it structurally conforms to the underlying specification. Yet, this does not by itself guarantee that all FHIR entities created in the mapping process are

also filled with the necessary values. In order to ensure this, during the development of MOnSTER, the validity of the generated FHIR is iteratively checked using the official FHIR validator (HL7, 2022). For example, for a `QuestionnaireResponse` entity, it is always necessary to specify the status of the entity, such as whether it is currently in progress or completed, which is then easy to provide as the source `CRFInstance` entity contains just such a flag attribute.

Finally, it is worth mentioning that in the ongoing development of FHIR for its next official release version R5, the area for handling studies / trials is also under investigation and is to be expanded. Therefore, it is also planned here to review and reinvestigate the current mapping and its implementation and to adapt both for the changes and extensions of this mentioned version.

7.3 Format Serializations

The previous section presents how the internal information model is mapped onto those of ODM, RDF and FHIR. This section describes for each format its serialization procedure, that is, the translation into the respective external form which can be exchanged and externally stored.

It is important to emphasize that all serializations of a given single format are completely equivalent in regard to their actual content. They just only express this content in different syntactic formalisms. The rationale for this is that in some cases external tools wanting to read and process data may understand a certain format, but may not be able to read all available serializations.

ODM

This information model implementation is based on classes automatically generated by JAXB from ODM's standard XSD files. These files' fundamental task is to formally define how correct ODM XML files need to look. Using JAXB, it is now possible to "marshal" objects created on the basis on these classes, that is, to generate and output respective XML without the need for any additional transformation code. Concretely, the `marshal` method of JAXB's `Marshaller` class is called with ODM's top level (trial)

entity, and the library traverses all dependent, nested entities top-down generating and returning the appropriate XML for each.

RDF

A standard serialization mechanism provided by the selected library, namely RDF4J, is used for this information model also. This library includes a toolkit, called Rio ("RDF I/O"), with several built-in parsers and writers for input and output of RDF in all of its standard serializations. Here, the `write` method of the `Rio` class simply gets the RDF-based information model along with the parameter of the serialization to be used, see Figure 7.1.1(c), and the library converts the information model accordingly and returns the result. As a note, in this context the entire model without the need for a specific start entity, since the internal representation of the RDF model is not nested but rather a simple, flat list of triple statements. MOnSTER currently supports the RDF export in TURTLE (Beckett et al., 2014), RDF/XML (Gandon et al., 2014), N-Triples (Beckett, 2014), and JSON-LD (Sporny et al., 2020).

FHIR

Again, the HAPI FHIR library too provides by default the necessary functionality to parse (external) FHIR input and write such out for all serialization types that the FHIR standard defines. Based on the specified serialization type, see Figure 7.1.1(c), first a corresponding object of the class `Parser` whose `encodeResourceToString` method then translates the information model, encoded as a bundle with resources for all entities, into the serialization. Again, only the bundle must be specified here without a top / start entity, as the bundle contains all generated resources as flat list that can be processed sequentially. The FHIR export of MOnSTER can produce JSON, XML, and TURTLE/RDF (HL7, 2019).

7.4 Web Service Interface

Previously, the export of trial data could only be performed in ODM format via the web interface of the application. Building on this, MOnSTER has slightly extended the given

interface to enable the export of not only ODM, but now RDF and FHIR as well, as presented in Section 11.1.

As an additional possibility to execute the export, MOnSTER introduces a dedicated export web service. Here, a web service is a software system to support interoperable machine-to-machine interaction over a network with an interface described in some machine-processable format (Haas & Brown, 2004). For ObTiMA, this means that with the help of this web service, the export can now be performed independently of the user interface without the need for any direct user interactions. Hence, it is now possible to perform the export from external scripts or applications and integrate it seamlessly into, for example, automated data analysis workflows and processes.

The realization of the web service follows the general architectural style and concepts of REST (Representational State Transfer) (Fielding, 2000), which means that it offers its functionality via a web-based API, accessible through a predefined URI scheme, using the standard HTTP methods and media types relevant for the given formats and serializations. All parameters possible here are discussed in Section 11.1.

To ensure that the REST interface can be readily applied for its intended task, its definition adheres closely to the best practices and guidelines produced within the open source community and by major industry vendors (Gossman et al., 2022). Also, existing APIs are evaluated in terms of their design and their implementation and used as blueprints for the work on MOnSTER (Google, 2022; Microsoft, 2021).

Its implementation also strictly follows the previously emphasized separation of concerns and is therefore very lightweight: It only implements and provides the actual web service API in the `ExportWebService` class, as shown in Figure 7.4.1, and delegates the execution of the export to the designated class in the application logic layer, namely the `ExportAction` class, as described in Section 7.1. So the web service's `export` method itself only mirrors the corresponding application method: It receives a HTTP `GET` request from a remote (REST) client which contains all necessary parameters, "unwraps" those and passes them on to the actual `export` method. From there it receives back a string with the export artifact, that is, the ODM, RDF, or FHIR code, "wraps" it into a HTTP response and sends it back to the client.

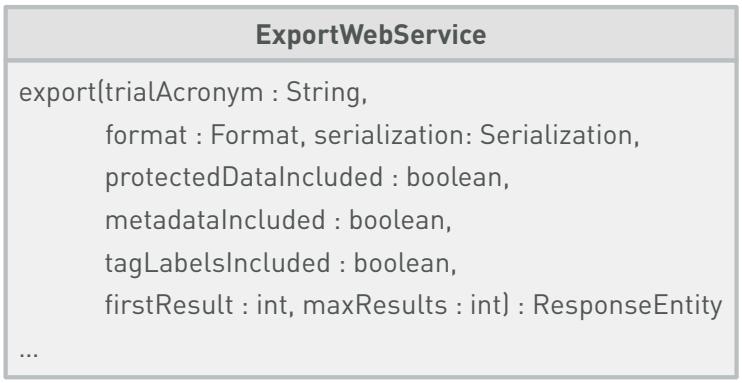


Figure 7.4.1 – *Export service class*

The development of the web service uses the MVC (Model-View-Controller) and web component of the Spring Framework as its fundation. This allows the entire service to be developed completely in Java and be implemented within the given, single class only. From this code, the framework automatically generates the REST API and exposes the method to trigger the export.

It is important to note that the web service is configured to be protected through the HTTP basic authentication mechanism (IETF, 2015) as provided by the core ObTiMA which, in turn, uses Spring Security (VMware, 2022b) as its base. This allows any new authentication mechanism, once added to the core application, to be readily used by the web service as well.

The aforesaid machine-readable description of the web service's interface is realized using the OpenAPI specification (OAI, 2021), selected for its public, open source availability and as de-facto standard for creating vendor-neutral descriptions of REST-based web service APIs. The full OpenAPI description is listed in Appendix B and the definition of all parameters is presented in Section 11.1.

Part III

Results

Chapter 8

General Background

In order to make the achieved results more tangible and render their real-world relevance clearer, they are presented below in the style of a user software manual applying a recent, concrete use case scenario, introduced in Section 8.1. In this context, it should be noted that the focus of the following lies on MOnSTER and, therefore, refers to ObTiMA 's core parts only if relevant and necessary. A complete documentation covering the entire application is available on request too, see (ObTiMA, 2022).

8.1 Project / Data

Quickly after the outbreak of COVID-19, the scientific community launched a large amount of research activities to investigate the various aspects of the pandemic and to develop effective vaccines, drugs and treatments as quickly as possible. To this end, tremendous amounts of highly diverse data has been collected and analyzed worldwide within a multitude of diverse, concurrent research studies and projects. In this context, however, it is not only problematic that the data itself is highly heterogeneous but also its representation, that is, different formats with different identifier schemes and different measurement units render its integration and analysis extremely difficult. For tackling this challenge, the GECCO (German Corona Consensus) dataset (Sass et al., 2020) is set-up within the NUM (Network Univerity Medicine) initiative (NUM, 2020), which aims at bringing together experts from all German university hospitals to develop and evaluate diagnostic and treatment strategies for the optimal care of COVID-19 patients. Based on recent both national and international work and collaborating with (bio)medical experts from hospitals, professional associations, and research initiatives, a collection of 83 data (description) elements relevant to COVID-19 research with 281 answer options is created and organized into 13 categories (von Kalle et al., 2021), as shown in Table 8.1.1.

Category	Description
Demographics	Data like gender, date of birth, weight, height, ethnic group
Anamnesis / Risk Factors	Data regarding pre-existing conditions, such as cardiovascular diseases, cancer diseases, HIV, or diabetes
Imaging	Data about imaging procedures during COVID-19 treatment, like computer tomography, radiography, ultrasonography
Epidemiological Factors	Indicator if the patient had contact with a person likely or proven to be infected with COVID-19 within the last 14 days
Complications	Data on COVID-19-related complications, like thromboembolic complications, pulmonary embolism, myocardial infarction
Onset of Illness / Admission	Indicator for the stage of illness when COVID-19 was diagnosed and the patient admitted to the hospital
Laboratory Values	Data about laboratory values collected in COVID-19 therapy
Medication	Data on drugs administered for COVID-19 or other diseases
Outcome at Discharge	Indicator for the respiratory outcome and the type of discharge from the hospital
Study Enrollment / Inclusion Criteria	Indicator whether a confirmed COVID-19 diagnosis was the main reason for admission to a the hospital
Symptoms	Data covering loss of taste, abdominal pain, diarrhea, vomiting, cough, nausea, fever, or dyspnea
Therapy	Data regarding the therapy that the patient received, like regular vs. intensive care, ventilation
Vital Signs	Data about vital signs, such as body temperature, blood pressure, heart rate, or oxygen saturation

Table 8.1.1 – *GECCO categories for categorizing the single data elements*

The reason for the particular relevance of GECCO in the context of the given work here is its reliance on and application of several standard ontologies. This means concretely that all stated categories, data elements, as well as the associated response options are encoded using concepts, and thus identifier codes, from the ontologies listed below, see Appendix E for all references:

- SNOMED CT (Systematized Nomenclature of Medicine Clinical Terms)

- LOINC (Logical Observation Identifiers Names and Codes)
- ICD-10-GM (ICD, 10th Revision, German Modification)
- OPS (Operation and Procedure Classification System)
- ATC (Anatomical Therapeutic Chemical Classification System)

As part of the NUM initiative, the CODEX (COVID-19 Data Exchange Platform) project (Prokosch et al., 2022) now employs GECCO as its basis to establish a central research data platform for sharing data among its partners. Here, the general idea is to pull and extract all of the relevant data directly from the clinical information systems used in patient care and map this according to the above categories and data elements.

However, as many necessary data elements are, at least currently, not routinely captured in these systems yet, project partners can enter the missing data by means of an additional EDC (Electronic Data Capture) tool, namely REDCap (Harris et al., 2009, 2019). For this purpose, CRFs are created, where each of them represents on one of the above categories, and the questions and answers contained therein are in turn based on the data elements and answer options within each category.

As part of the recent testing of MOnSTER, these CRFs are re-implemented in ObTiMA with a particular focus on the component's specific features for ontology management, semantic tagging and data export. In this, to test and evaluate some specific aspects of MOnSTER, such as the support for multilingualism or the simultaneous use of multiple ontologies, as highlighted in Appendix E, the following additional ontologies originally not used in the GECCO dataset, have been applied as well:

- MedDRA (Medical Dictionary for Regulatory Activities)
- NCIt (National Cancer Institute Thesaurus)
- ORDO (Orphanet Rare Disease Ontology)
- CTCAE (Common Terminology Criteria for Adverse Events)

8.2 Procedure / Realization

Considering the MOnSTER component, it can be said that there are both direct as well as indirect users of its functionality: On the one hand, direct means that users interact concretely and actively with the component's functionalities, that is, they manage ontologies, semantically tag trial elements, or export data. On the other hand, indirect then means that users do not actively interact with the component directly, like by performing semantic tagging, but rather they (only) apply the "outcome" of the component, that is, they fill in CRFs with previously semantically tagged questions and answers.

Therefore, the focus for the result presentation below lies on the direct and thus visible and tangible application of the component from the viewpoint of an actual user. This presentation is thus divided into three distinct parts, with each of them reflecting one of the general usage scenarios and the respective part of the component. To obtain the given results, the several steps below are performed:

- Creation of a trial with all necessary elements, namely SEs, CRFs, CRF sections, questions and answer options
- Identification, loading and adding of all relevant ontologies to the trial according to the GECCO specification (with the named additions)
- Semantic tagging of all created above trial elements with ontological concepts following the GECCO specification (again, with the mentioned additions)
- Creation of ten fictional patients, and for them, filling in all SEs and CRFs with realistic but artificial, manually created data
- Export of the trial and its data in all combinations of the different formats and serializations available
- Identification within the generated files of the different semantically tagged data elements and their evaluation and validation

Chapter 9

Ontology Management

The following chapter describes from the perspective of a user how to add and manage ontologies with the help of the component. Its underlying technical realization is presented previously in Chapter 5.

9.1 Adding and Editing Form

Before MOnSTER can be used for semantic tagging, the intended ontologies first need to be available in the application. For this, the user selects Administration from the main menu (Figure 9.1.1, ①), then Ontologies in the submenu (②), and in the subsubmenu Add (③) opening a form to specify the parameters of the ontology to add (see below). Here, the user can also select Manage (④) to open a table listing all already loaded ontologies and use the Add button there, as described in Section 9.2.

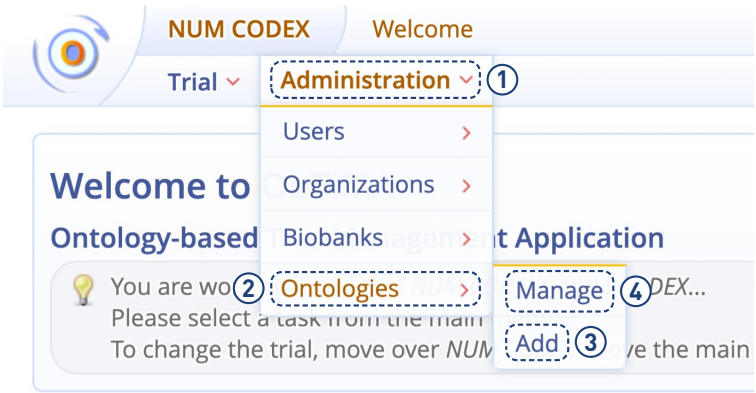


Figure 9.1.1 – *Main menu entries for adding and managing ontologies*

After clicking the link or button, the ontology can be added and the required parameters, shown in Table 9.1.1, can be specified. Each parameter is explained in the following

based on two examples, one for ontologies in OWL format, as shown in Figure 9.1.2, and one for ontologies having a custom line-based format, as depicted in Figure 9.1.4.

Parameter	Required	Examples / Note
Acronym	●	"GO "
Name	●	"Gene Ontology "
Version		"21.10d", "2021"
Enabled	○	
Description		
Source	●	cf. Subsection 9.1.1
Namespace	●	cf. Subsection 9.1.2
Format	○	cf. Subsection 9.1.3

Table 9.1.1 – *Parameters of an ontology*

Here, the combination of acronym, name and version must be unique, which means that it is (intentionally) not allowed to add two different ontologies where these three parameters taken together hold the same the values for each each ontology.

9.1.1 Source

This parameter, to be found at Figure 9.1.2, ②, holds the location of the actual file which contains the ontology and can be provided by

- uploading a local ontology file by clicking the file selection button (③), opening a file chooser, and selecting the appropriate file, so the parameter is automatically set,
- manually preloading the ontology file onto the server, so the path to that file on the server needs to be manually specified in the File / URL field, or
- by pointing to a remote location, in which case the full URL where the ontology file can be found, must be specified in this field.

In the first and last case, the ontology is uploaded from the local file or from the remote location to the server when the Save button is clicked. As ontology files can potentially

be very large, it is also possible to provide them in ZIP- or GZIP-compressed form.

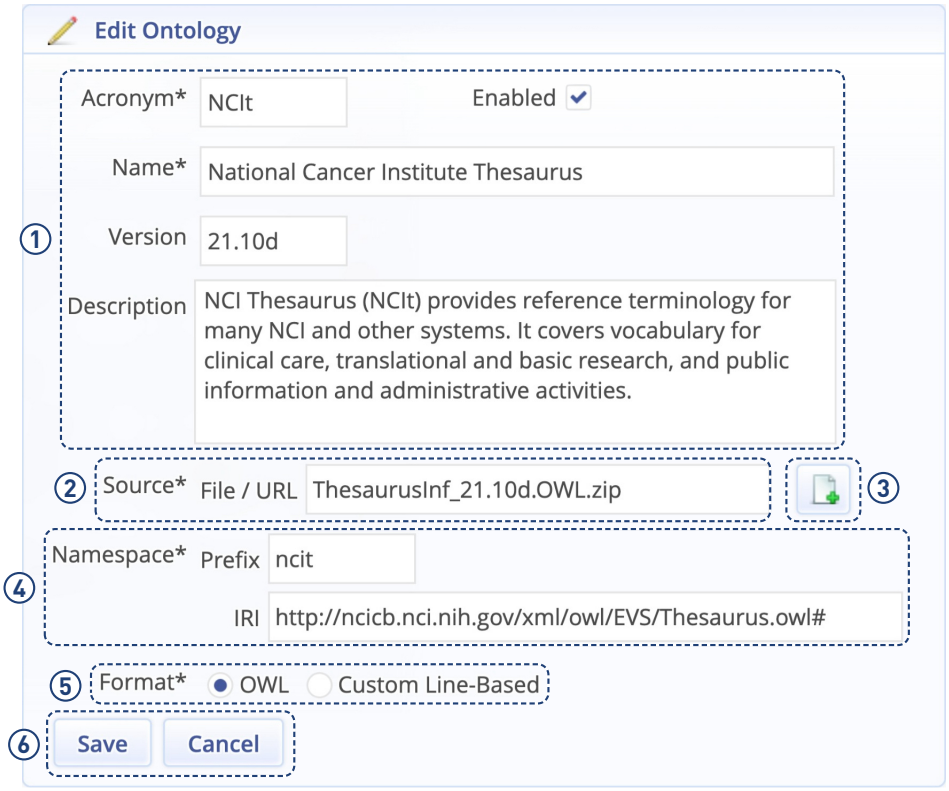


Figure 9.1.2 – *Popup for editing the metadata of an OWL ontology*

9.1.2 Namespace

A namespace is used to provide an unambiguous identifier, usually also unique for an ontology and its contained concepts. It follows the basic definition from XML (Bray et al., 2009) and consists of a short prefix and so-called name. The name, represented here as an IRI (④), acts as the actual identifier and is mostly hidden within the GUI, such as in the semantic tagging. Rather, the Prefix being a short, abbreviatory string, is used both in the GUI and in the export to improve clarity.

For OWL-based ontologies, the appropriate namespace is provided in the header of the ontology file. For ontologies with custom line-based format, an (official) namespace can usually be found in their documentation or in specific listings (HL7, 2020).

9.1.3 Format

As mentioned, MOnSTER can process ontologies provided in either the Semantic Web's standard OWL format (Hitzler et al., 2012) or in some Custom Line-Based format.

OWL If the supplied ontology file is in this format, the component is able to process the file directly without the user having to specify any additional information.

For example, Figure 9.1.3 presents a small OWL file excerpt for a single concept (class) from SNOMED CT with the identifier code `840539006` in TURTLE (Beckett et al., 2014) serialization. As can be seen, this class holds five different labels, where the first one uses the standard RDFS label annotation (Brickley et al., 2014), the second one the SKOS preferred label and the last three ones the SKOS alternative label annotation (Isaac & Summers, 2009). For each label the language can be specified following the ISO 639-1 standard (ISO, 2002), such as `en-gb` for the English variant spoken in Great Britain. As the SKOS preferred label always has priority, this means that in a search for creating semantic tags, as described in the next chapter, the second label is displayed first before all other concept labels.

```
<http://snomed.info/id/840539006> rdf:type owl:Class ;
  rdfs:label "Disease caused by severe acute respiratory syndrome
  ↪ coronavirus 2 (disorder)"@en ;
  ...
  skos:prefLabel "COVID-19"@en ;
  skos:altLabel "Disease caused by 2019 novel coronavirus"@en ,
    "Disease caused by severe acute respiratory syndrome
    ↪ coronavirus 2"@en ,
    "Disease caused by 2019-nCoV"@en-gb .
```

Figure 9.1.3 – *Example of an OWL class with labels of different type*

Custom Line-Based In the case that a custom line-based ontology file is provided, the user needs to specify the two additional parameters below, shown in Figure 9.1.4, ①.

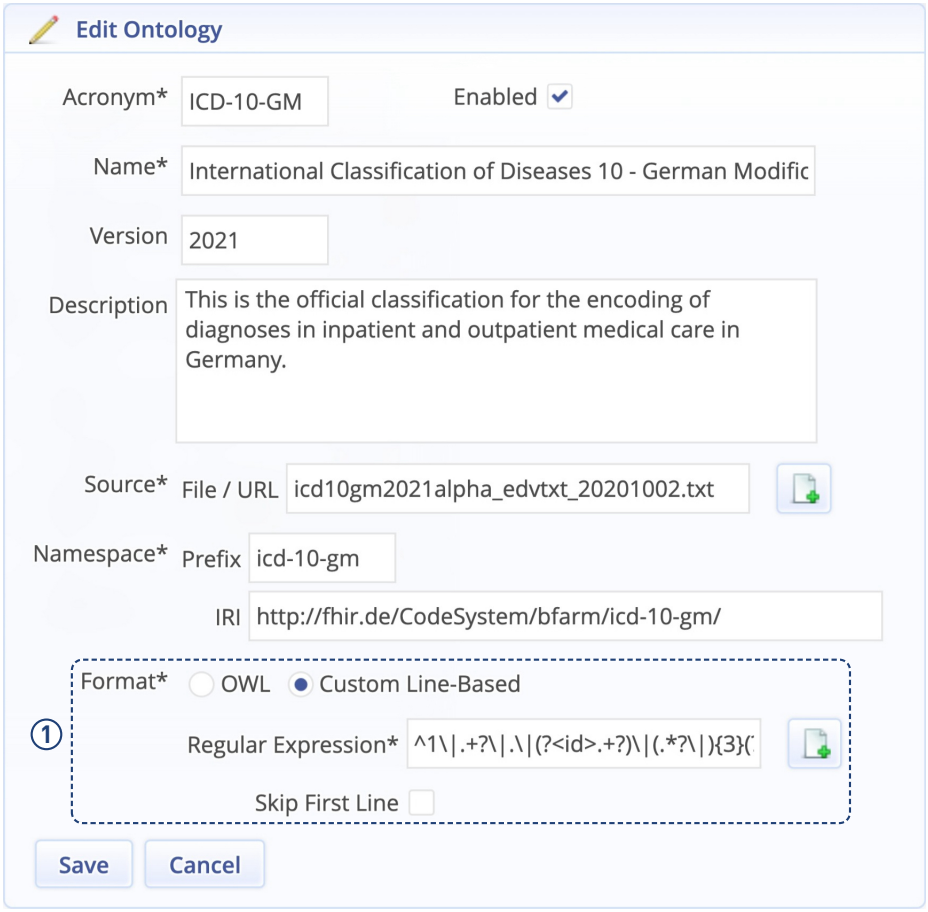


Figure 9.1.4 – *Popup for editing the metadata of a custom line-based ontology*

Skip First Line Regarding CSV or DSV files, their first lines often contain a header declaring the individual columns' names or definitions. In such a case this parameter needs to be enabled so that the first line is disregarded during in the further processing.

Regular Expression As for the appearance of custom line-based files, MOnSTER is highly flexible. The only strict requirement is that each line in such an ontology file must at least always contain the code of a concept and an associated label. In addition, the language for that label can be specified, and further, whether the label in this line is the preferred one for the given concept. If a single concept has multiple, possibly multilingual labels, for each label a separate line needs to be provided within the file. For each item to be extracted from a line, the regular

expression needs to contain a so-called named capturing group, that is, `<code>` for the concept's identifier code, `<label>` for a (single) label of the concept, `<preferred>` to state the preference of that label, and finally `<language>` to specify the label's language.

To illustrate this, the snippet in Figure 9.1.5 contains the same SNOMED CT concept `840539006` with the same labels as in the above OWL example, only in CSV-based form now. In this particular example, at the beginning of each line, the code of the given concept is to be found. This is followed by the respective label and its language according to the ISO 639-1 standard (ISO, 2002). If a label is to be the preferred one then this is marked with `p`, as shown in the second line. (If a label is not preferred then any other or no value at all can be given here.)

It is important to note that the arrangement within a line and the used comma delimiter, as shown here, is only exemplary. As long as the two necessary parameters, that is, `code` and `label`, are contained in a line and can be extracted by the given regular expression, then a given file can be used as ontology source.

```
840539006,,Disease caused by severe acute respiratory syndrome
↪ coronavirus 2 (disorder),en
840539006,p,COVID-19,en
840539006,,Disease caused by 2019 novel coronavirus,en
840539006,,Disease caused by severe acute respiratory syndrome
↪ coronavirus 2,en
840539006,,Disease caused by 2019-nCoV,en-gb
```

Figure 9.1.5 – *Lines from an custom line-based ontology file*

Now to extract all of the required parameters from a line, a regular expression needs to be created. In Figure 9.1.6 the necessary expression is shown which can be used to do this for the given example. As shown there, a matching expression must provide a so-called named group for each parameter to be extracted. Since only the concept's code and label must be necessarily be contained in each line, a valid regular expression may also contain only the two corresponding groups `<code>` and `<label>`. As specifying language and preference of a label are

optional, their groups <language> and <preferred> are optional too.

```
^(?<code>.+?),(?<preferred>p?),(?<label>.+?),
↪ (?<language>\w{2}(-\w{2})?)$
```

Figure 9.1.6 – *Regular expression to extract the parameters from the line*

Since it is unfortunately not possible to give a comprehensive introduction to the creation of regular expressions within the scope of this thesis, reference is made here to one of the numerous introductions available online, such as (Nield, 2017).

9.2 Overview and Selection Table

The table in Figure 9.2.1 now provides a listing of all ontologies currently loaded and available in the application together with the identifying information for each, that is, Acronym, Name, Version, and if currently Enabled. It can be sorted by clicking the little red arrows above the columns, such as for example at ① and the number of ontologies concurrently displayed can be adjusted (②), as well as if more ontologies are available than can be shown at once, pagination is possible also(③).

It should be noted here that there exist two distinct entries for ICD-10-GM in the table: It is possible to have several versions of a single ontology concurrently in the system. Each one is processed and handled independently and, therefore, can be used both individually and simultaneously within a given trial.

To add another ontology, the user can click on the Add button (④) to open the respective form, as detailed in Section 9.1. It is important to emphasize that when ontologies are added, they are always automatically available system-wide in ObTiMA for all existing trials as well as when creating new ones.

To edit an ontology, the user can click either on Acronym, Name or Version of an ontology or on the pencil icon to the right side of the table.

Figure 9.2.1 – *List of all ontologies currently loaded*

To remove of an ontology, the user clicks on the icon showing a red circle with a white minus, which opens a dialog box depicted in Figure 9.2.2 to either confirm or cancel the removal of the ontology. This icon appears only if an ontology can actually be removed, that is, if it has not been selected for use in some trial, see Section 10.2.

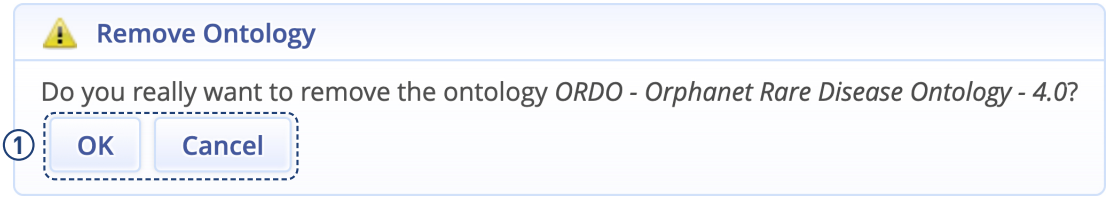


Figure 9.2.2 – *List of all ontologies currently loaded*

Chapter 10

Semantic Tagging

After adding an ontology to the system, its concepts can be used to create semantic tags. This chapter now illustrates how this can be achieved and how such tags are added to the different components of a trial within ObTiMA.

10.1 Ontology Preselection

Once added to the system, an ontology is immediately available system-wide and can be used in any new or existing trial. Yet, as not all ontologies are relevant for all trials, it is possible to manually select or deselect a particular ontology for a given trial. As a result, only selected, relevant ontologies are further displayed and used when creating and adding semantic tags to trial component.

To select an ontology for the currently active trial, the user performs the following steps: First, to open the trial's overview, the user chooses Trial from the main menu, shown at Figure 10.1.1, ①, and then Manage from the submenu at ②.

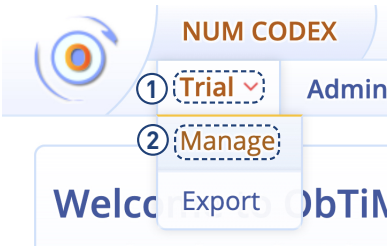


Figure 10.1.1 – *Main menu entry for trial management*

The trial overview given in Figure 10.1.2 can be divided into three logical parts:

- Input elements for specifying the trial's general attributes (①)
- Table for (pre)selecting the ontologies for the trial (②, see below)

- Element for semantically tagging the trial (①, see Section 10.2 for details)

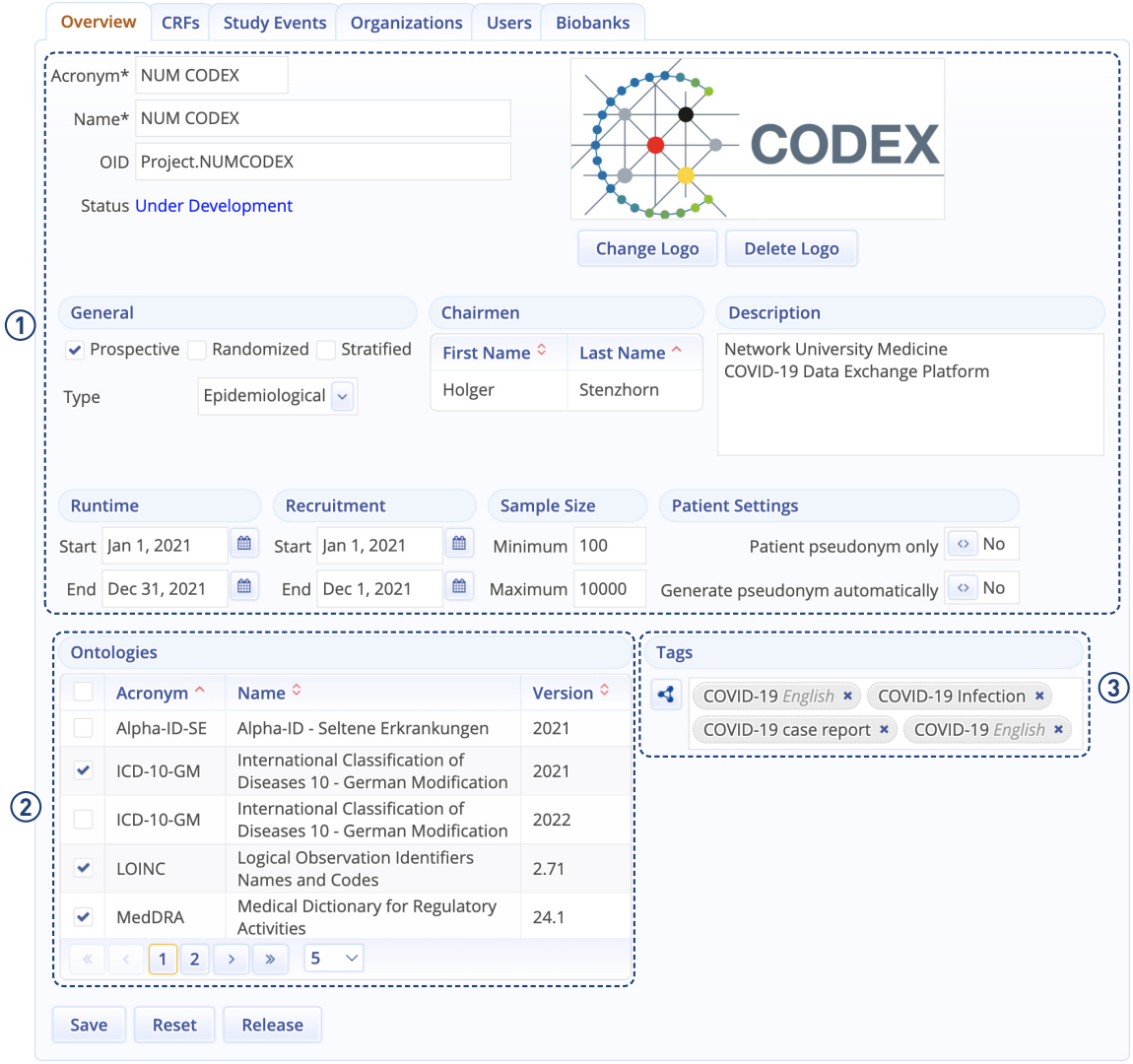


Figure 10.1.2 – *Overview of the currently active trial*

To select or deselect an ontology for the given trial, the user enables or disables its checkbox at Figure 10.1.3, ①. To select or deselect all ontologies at once, the top-most checkbox (②) can be used. If more ontologies are available than can be concurrently displayed, the user can paginate (③) or vary the number of shown ontologies (④).

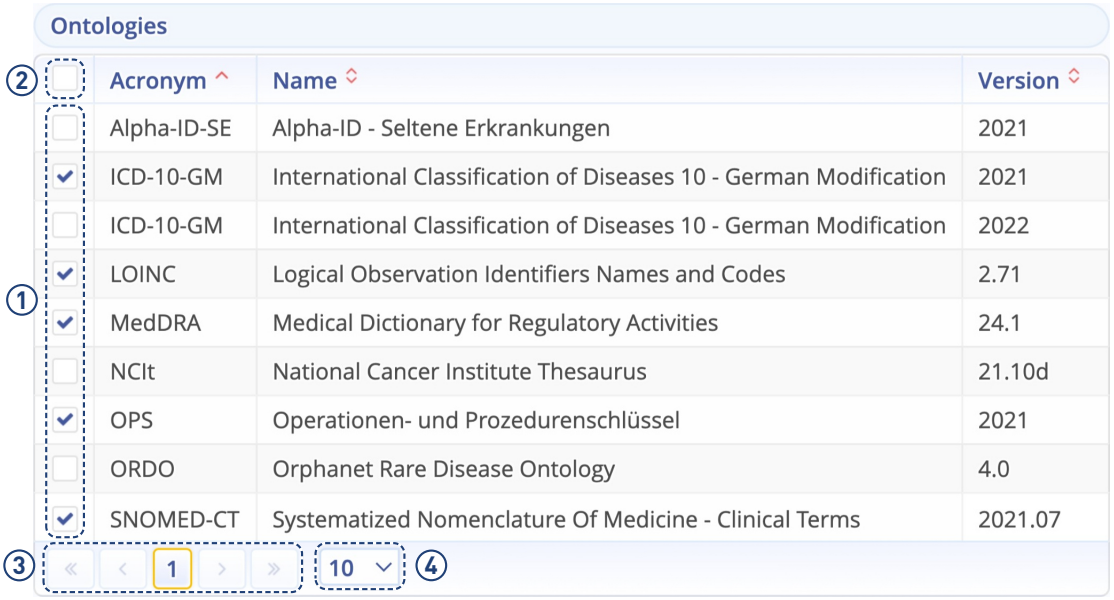


Figure 10.1.3 – *List of ontologies enabled for the trial*

10.2 Concept Selection and Tag Creation

As stated, one advantage realized by MOnSTER is that ontological concepts can now be applied to all components of a clinical trial, and not just for creating questions within a CRF. Semantic tags can therefore be added to give additional semantic metadata and to hence provide semantic "enrichment", to all the following components:

- trial
- study event (SE)
- case report form (CRF)
- question group
- question
- answer option

To do so, the user can employ the input element depicted in Figure 10.2.1, which is available in this form when editing any of the above components. This element allows searching for one or more suitable concepts within one or more ontologies, and adding

the fitting ones as tags to the given trial component.

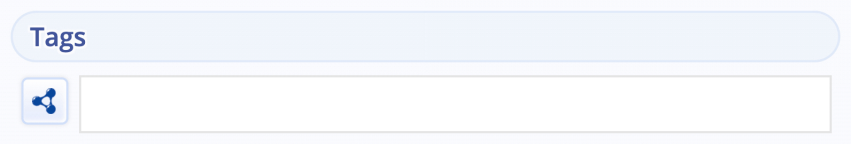


Figure 10.2.1 – *Entry field to search concepts and select a tag*

The previous section showed already how to generally enable or disable an ontology for a trial. When searching for concepts, the user can now also dynamically further restrict the ontologies included in this search.

For this, the user clicks the button with the ontology icon at Figure 10.2.2, ①. To include or exclude a single ontology from search, the user enables or disables the corresponding checkboxes (②). Using the topmost checkbox (③), the user can select or deselect all ontologies at once. By default, all ontologies are preselected.

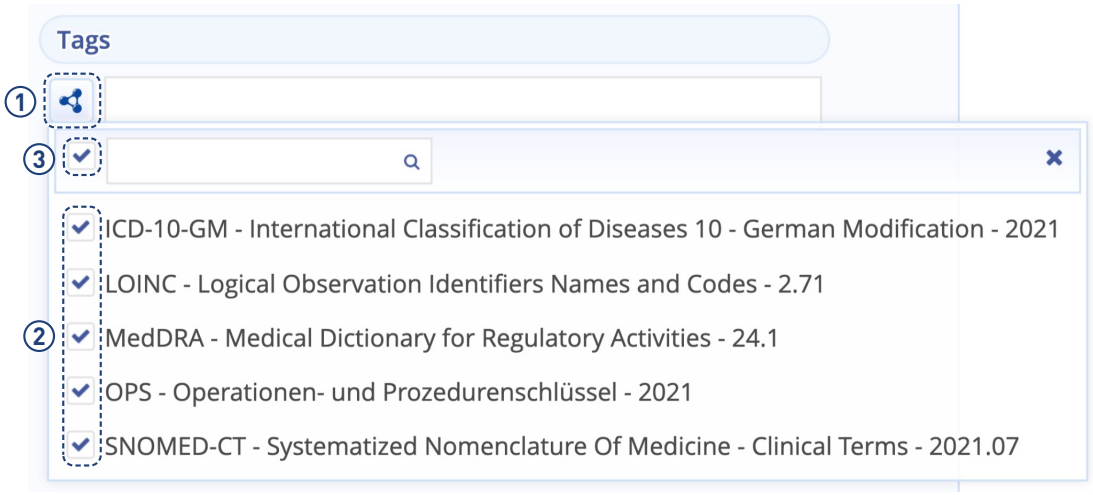


Figure 10.2.2 – *List of ontologies all selected for tagging in a trial*

To further simplify the selection here, the displayed ontologies can be dynamically filtered: As shown in the example, only ontologies with the (case-insensitive) string medic, as shown at Figure 10.2.3, ①, in their acronym or name are displayed.

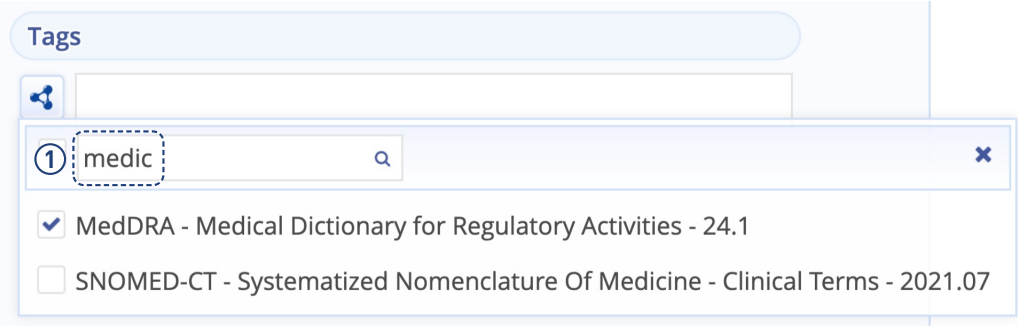


Figure 10.2.3 – *List of all ontologies available for tagging filtered by string matching*

To now actually perform the search for concepts, the user enters a query in the text field. This query, such as corona infek in Figure 10.2.4, ①, is first tokenized, that is, split into its substrings, like corona and infek, and based upon these the concept index is searched (see Chapter 5). Now, a concept matches, if at least one of the substrings is part of one of that concept's labels, and the concept belongs to an ontology included for search.

Note, that the search includes all labels, that is, if one of the substrings is found in a synonym label of a concept, then this concept is matched as well. It is also important to mention, that the search works incrementally, that is, the list of found concepts is automatically updated as soon as the query in the input field changes.

The presentation of the matching concepts is organized as follows: All concepts are grouped according to the ontologies they originate from, and above each group the corresponding ontology's acronym, name and version are displayed (②). Then, for each concept, its prioritized label is shown (③), and if the user hovers over that label, the concept's (ontology) code is presented to the right (④), and, if existing, further synonym or multilingual labels (⑤). Within each group, the list of concepts is sorted alphabetically according to their prioritized labels. Also, the found substrings from the query are marked in red within the label.

Note, that for simplifying the initial presentation, the search is limited to the ICD-10-GM ontology, and a query is chosen that matches a single concept only.

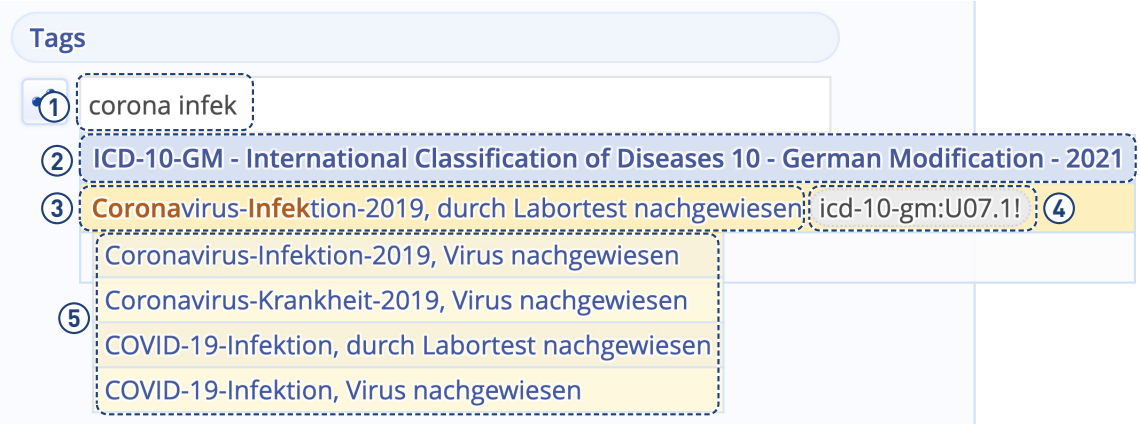


Figure 10.2.4 – *Found concept with synonym labels*

By clicking on the (prioritized) label of a found concept, the user creates a semantic tag and adds it to the given trial component, as shown in Figure 10.2.5, ①. The tag can be removed again from that component, by clicking the small x at the tag's right side.

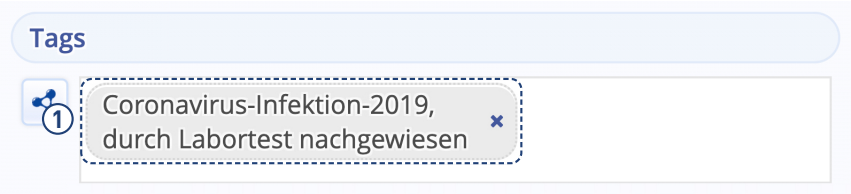


Figure 10.2.5 – *Semantic tag added to a component*

To view the full information of a semantic tag again, the user hovers over the desired tag at Figure 10.2.6, ①, and then, the original concept including its source ontology, code, and all synonymous and multilingual labels (②) are redisplayed.

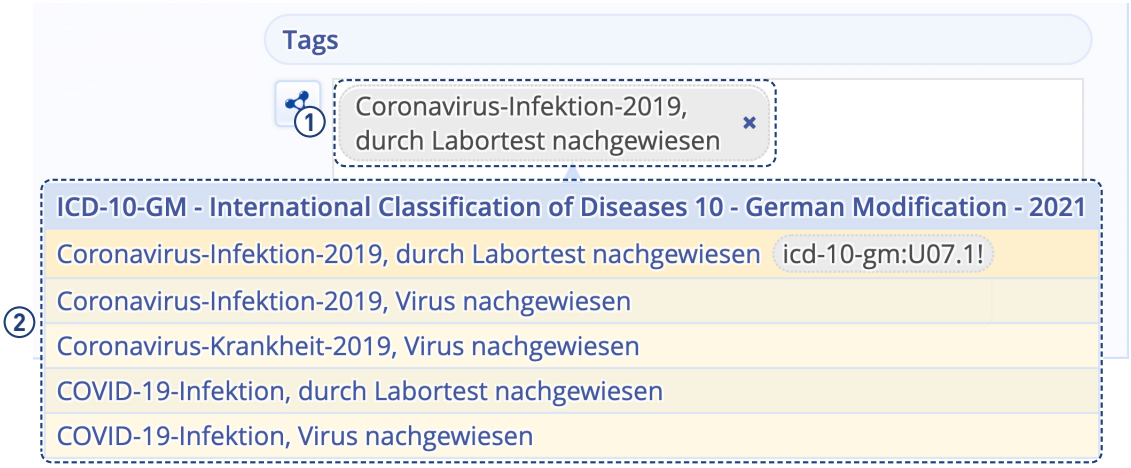


Figure 10.2.6 – *Same semantic tag as in Figure 10.2.5 with synonym labels shown when hovering over it*

To discover the multilingual capabilities of MOnSTER, multiligual ontologies can also be used when search concepts. In the example, the search is restricted to the MedDRA ontology, which has labels in different languages for each concept. Here the user enters the query corona infec into the text field at Figure 10.2.6, ① for which several concepts in this ontology are matching and shown below at ②. The user first hovers the second match (③) of the results, so that all multilingual concept labels are also displayed (④), and then selects it. For each label, the language of that label is also displayed.

In the case of multilingual ontologies, the user can also express queries in the languages and respective alphabets used in them. This means that, for the given example, the same shown result is obtained when entering the Russian query корона инфек.

At this point, it is worth mentioning, that the display of the different language labels takes into account the user's system settings regarding language, which means that labels in the user's specified language, such as English here, will be shown first if available.

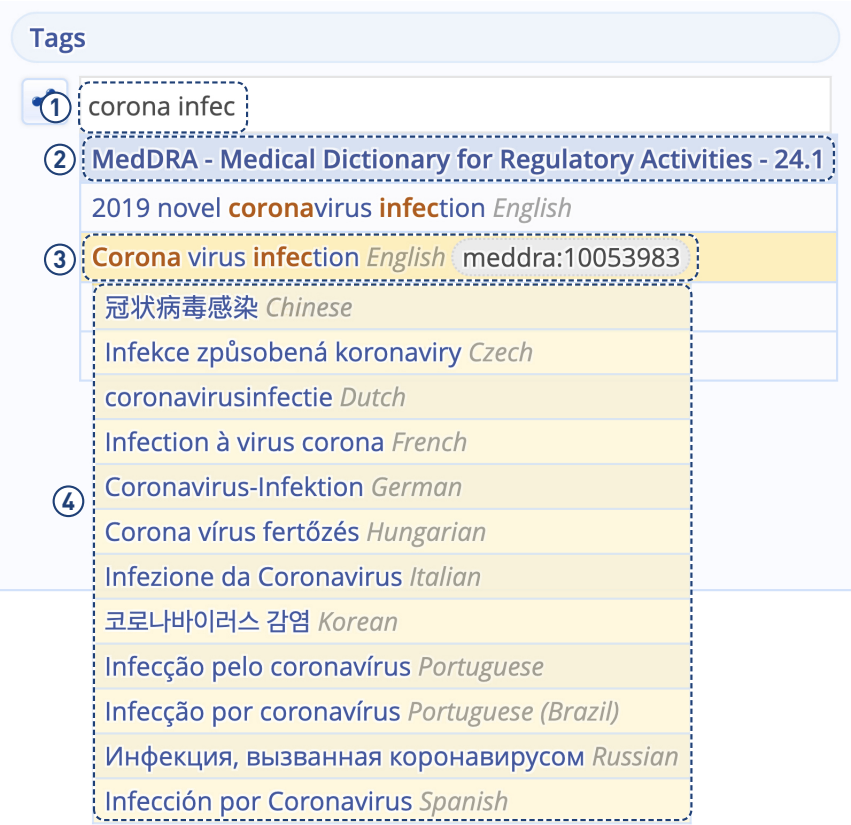


Figure 10.2.7 – *Found concept with multilingual labels*

As with semantic tags having labels in only a single language, when hovering over the tag at Figure 10.2.8, ①, the complete concept information can be redisplayed again for multilingual tags too (②). Note again, that the label in the language of user as specified in the system settings is shown at the top here as well. Also note, that here you can see the language directly in the tag as well. The language is displayed here only if the ontology specifies a language for its concepts.

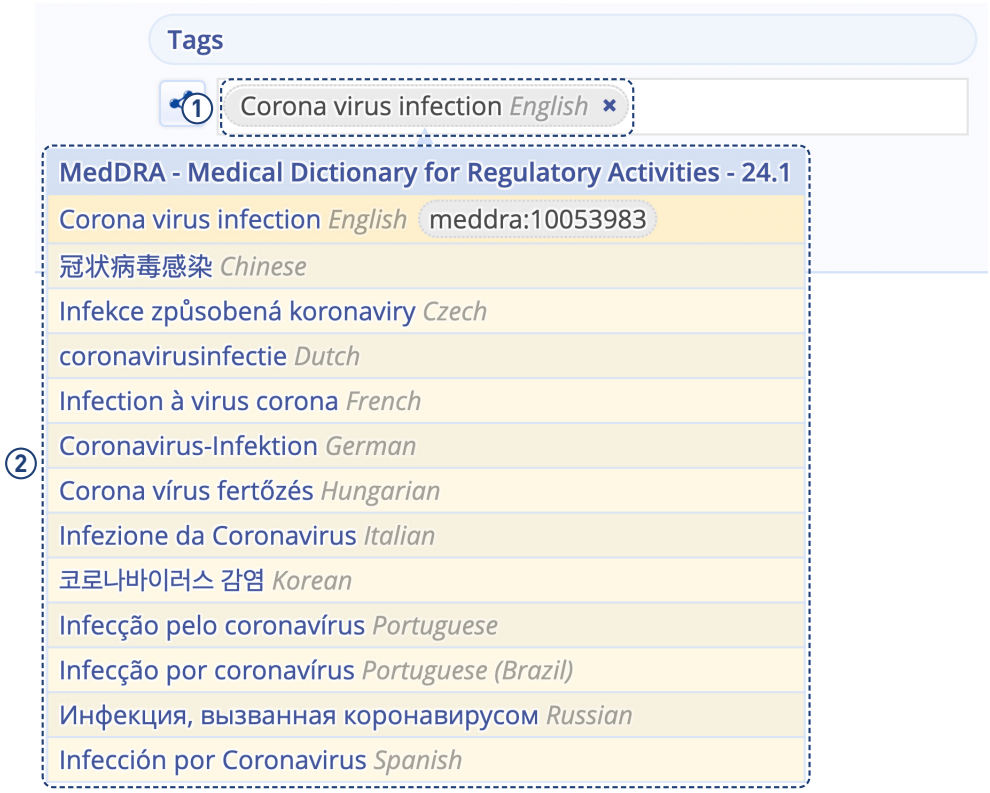


Figure 10.2.8 – *Semantic tag added to the component with multilingual labels when hovering over it*

To render the presentation clearer and simpler, the above examples are limited to the use of the ontology to be searched. In a normal setting, however, the user most often performs a search over multiple ontologies simultaneously.

When entering the query covid infe, as shown in Figure 10.2.9, ①, and under the condition that the search is not restricted to a single ontology, the result is a list of concepts (②) that originate from all ontologies selected for the given trial. Here, the respective groups are again sorted alphabetically for the ontology's acronym, name and version.

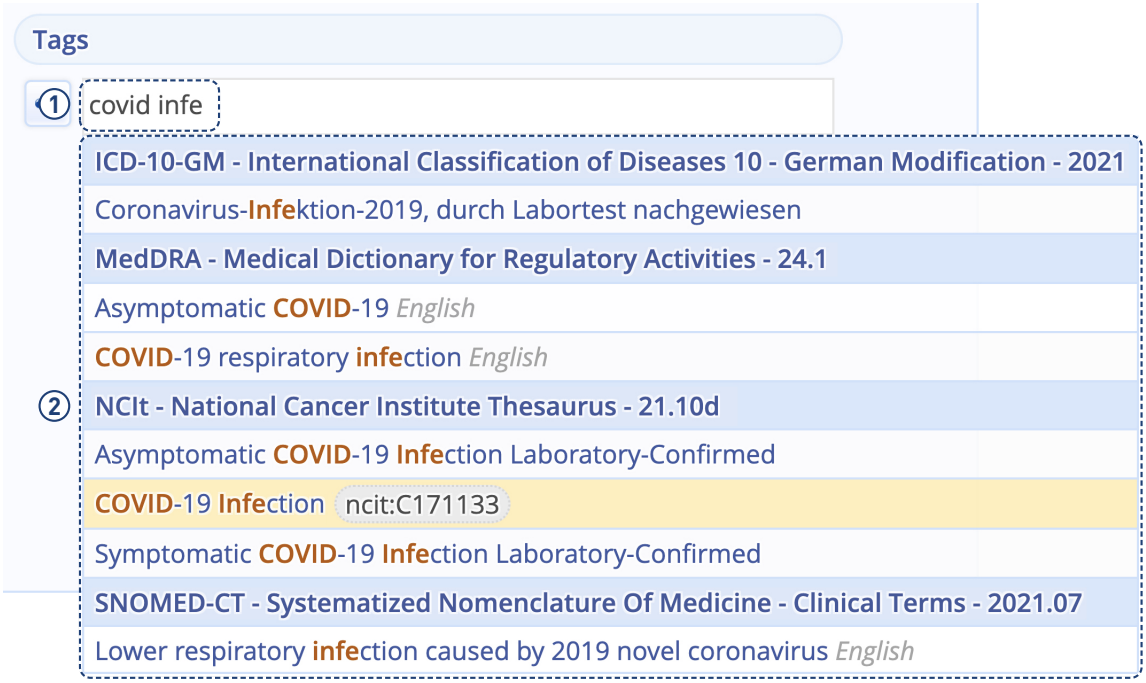


Figure 10.2.9 – *Tags from multiple ontologies matching a query string*

Also, to simplify the presentation of the above examples, only one semantic tag is added there at a time. In many cases, this approach of adding only a single tag to some trial component is sufficient, also in real-world settings. Nevertheless, there are situations in which several concepts are necessary to semantically describe such component sufficiently. This is possible without any problems, and the concepts can stem from different ontologies, from a single one, or a mix of both.

In the given example, presented in Figure 10.2.10, ①, four semantic tags are added to the trial, all of which are generally about COVID-19, yet are based on concepts from different ontologies. The hovering over the last tag shows another particular feature: Although SNOMED CT concepts are labeled with standard English terms, this ontology also contains labels based on local language term variants. For example, the last label (②) is based on a term specifically used in the United Kingdom.

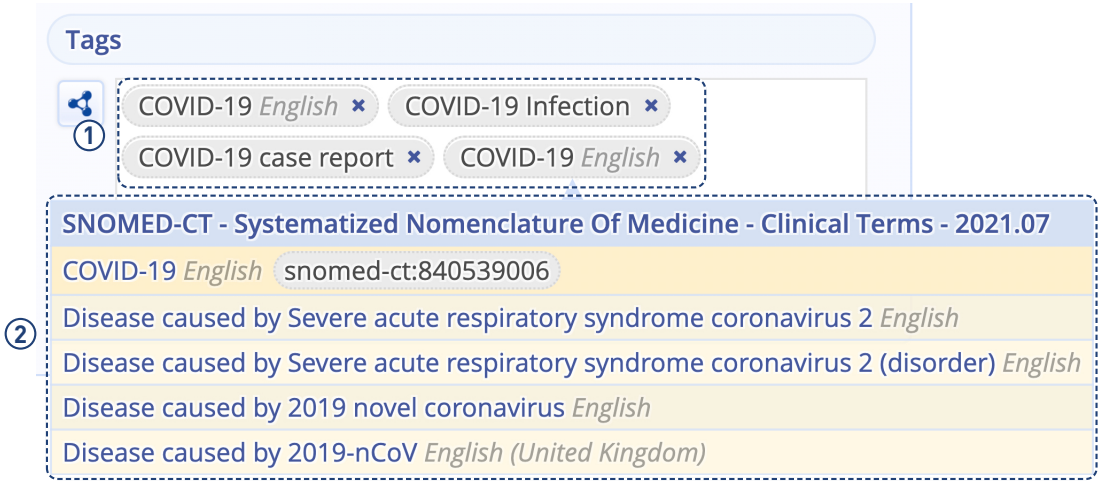


Figure 10.2.10 – *Multiple tags added with hovering over the last one showing its labels*

10.3 CRF Question and Answer Option Tagging

The previous section described how trial components within ObTiMA can be semantically tagged. In that sense, the tagging of CRF questions and answer options does basically not differ from that ot other trial components. It is specifically discussed here, as creating and using CRFs reflect the most important parts of the system's trial data management.

As noted, an introduction to managing CRFs can be found in ObTiMA's core documentation and this topic is considered here only insofar as it affects MOnSTER's functionality directly. In this context, nonetheless, an important and fundamental distinction in ObTiMA lies between two types of questions:

- Questions without answer options are those where values can be specified directly, such as a patient's weight or date of birth.
- Questions with answer options are those where multiple values are predefined for selection, such as a patient's gender or pregnancy status.

This distinction is quite important for MOnSTER: While for the first type, the question itself is semantically taggable alone, that is, to provide semantic meta-data about the question, for the second type, such additional meta-data can be added via semantic tags

to each answer option too.

Figure 10.3.1 shows the CRF Demography, which is based on the corresponding category and data items found in the GECCO dataset (see Section 8.1). From the corresponding icon symbols, it can be seen that, for example, question Biological Sex is tagged with a single tag (①) while question Biological Weight holds two or more tags (②). Accordingly, it can also be seen that, for example, the first answer option for question Pregnancy Status is tagged by two or more tags (③), but the second option with only one (④).

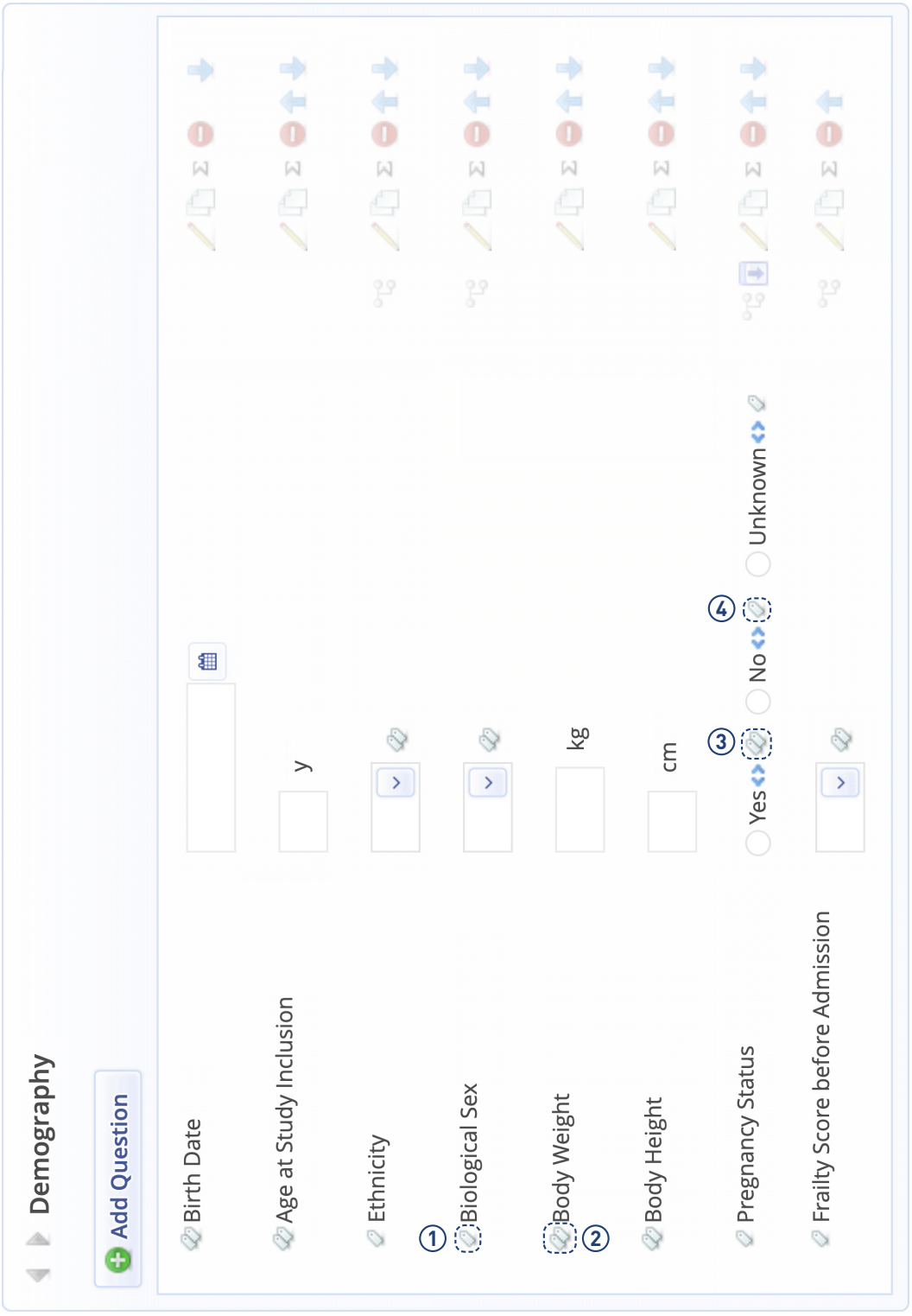


Figure 10.3.1 – *Definition of the CRF* Demography

When adding or editing a question without answer options, like the Body Height of a patient (Figure 10.3.2), the user specifies the question's basic attributes (①), selects the particular Answer Type (②), that is, Input Number in here for entering numeric values, and, if needed, additional parameters relevant for that answer type (③). Semantic tags can be added here too, exactly analogous as described in the previous section.

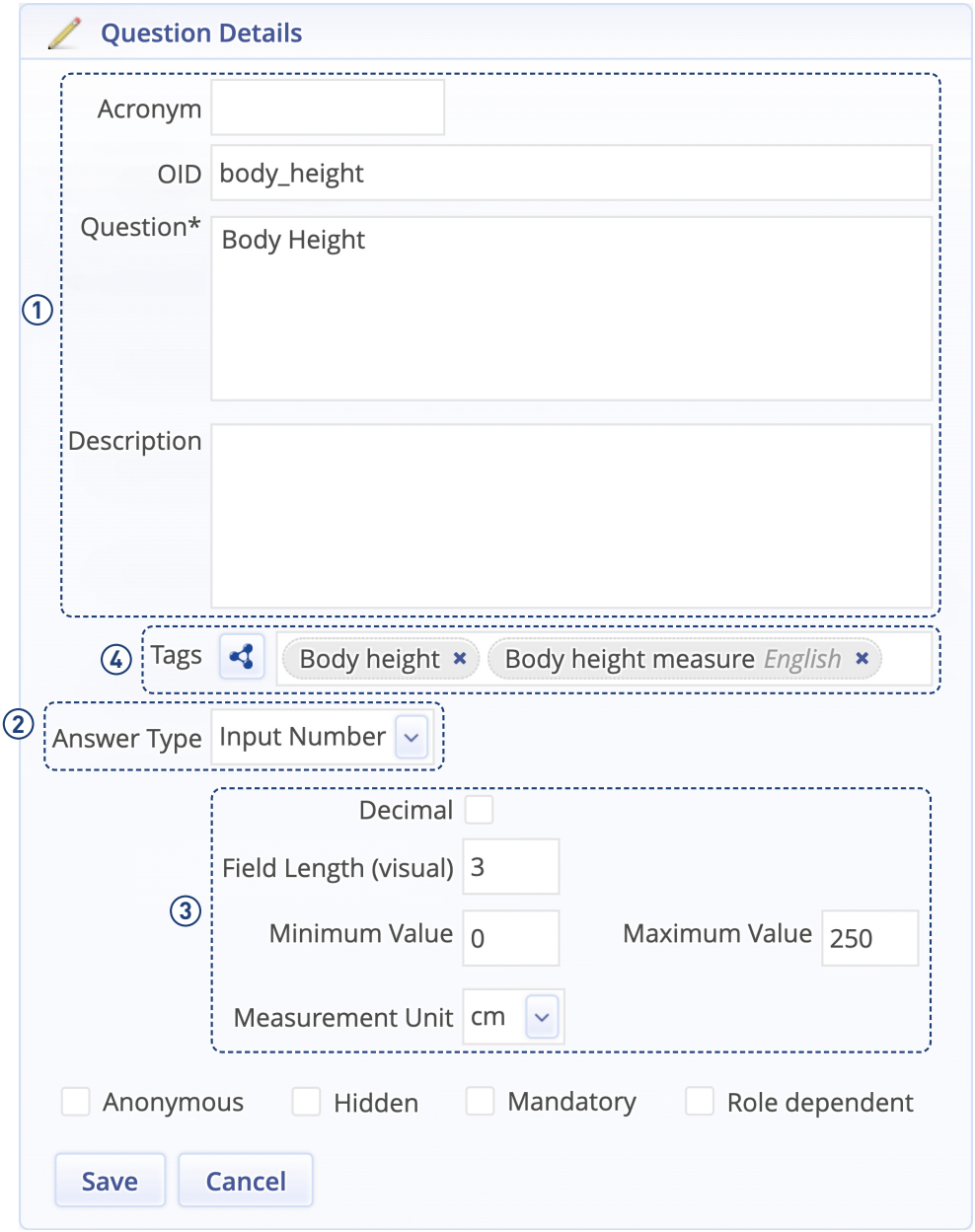


Figure 10.3.2 – *Definition of a question without answer options*

For a question that should have answer options, like for establishing the Pregnancy Status (Figure 10.3.3), the user selects a fitting Answer Type, such as Select One Radio (①) and adds a new answer option or edits an existing one (③). Note here, that for the question itself, semantic tags can be added as before (②).

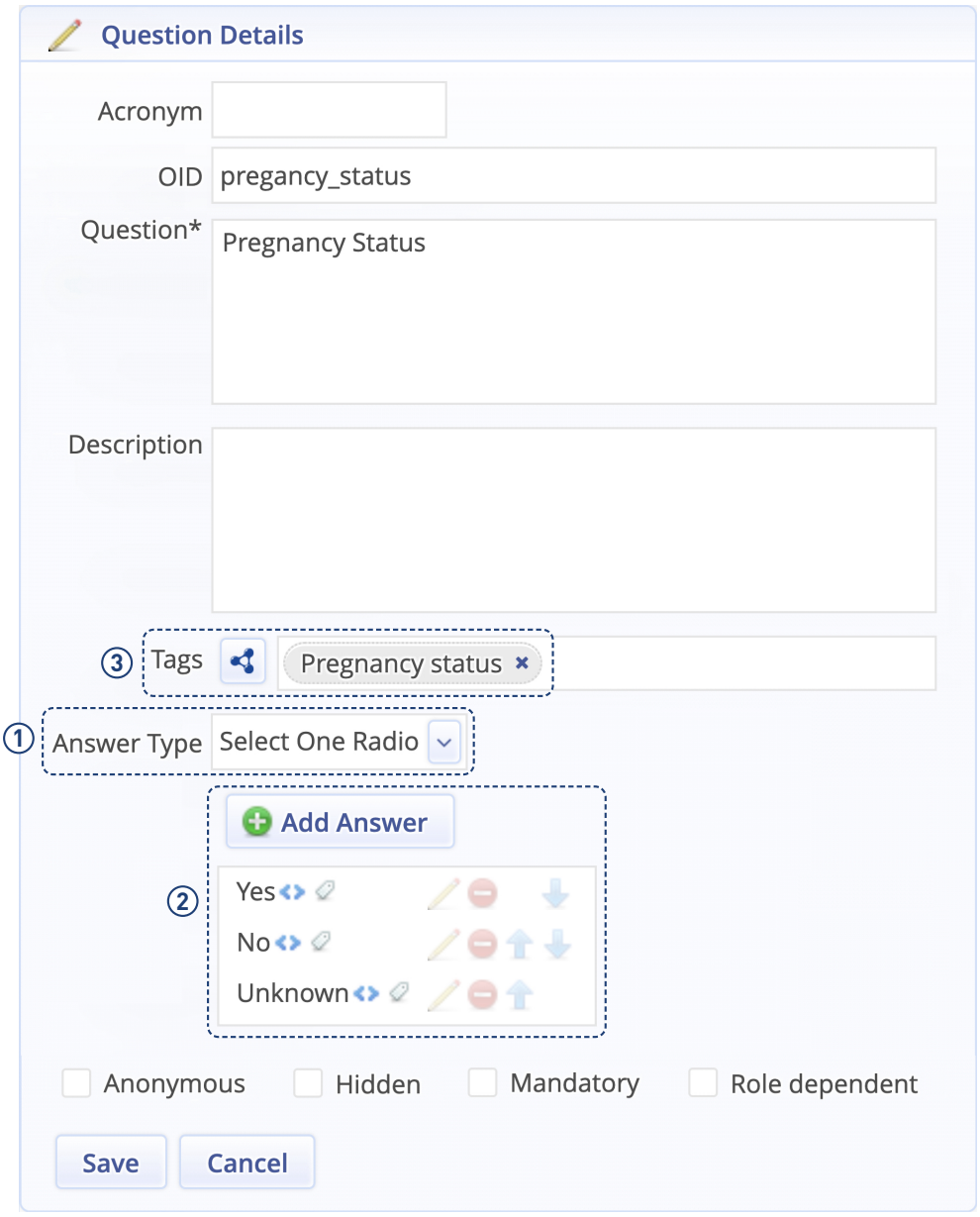


Figure 10.3.3 – *Definition of a question with answer options*

Also, when editing an answer option, along with its basic attributes, semantic tags can be provided in this case as well.

Figure 10.3.4 – *Definition of an answer option*

One important aspect of semantic tagging must be mentioned, which has been only implicit in the above: Tagging occurs alone when editing one of the different trial components, and so tags are only visible in the GUI for the users that have the system right to edit those components.

For users who only enter data in the system, such as study nurses, the tags are intentionally hidden in the GUI (Figure 10.3.5). The reasoning behind this approach is that these users are usually interested in entering a patient's data as quickly as possible, so, for example, whether a patient is pregnant (yes, no, unknown), but not in the technical details of the actual data encoding in the background. Thus, the addition to display semantic tags would unnecessarily clutter the the data entering GUI without benefit to the users.

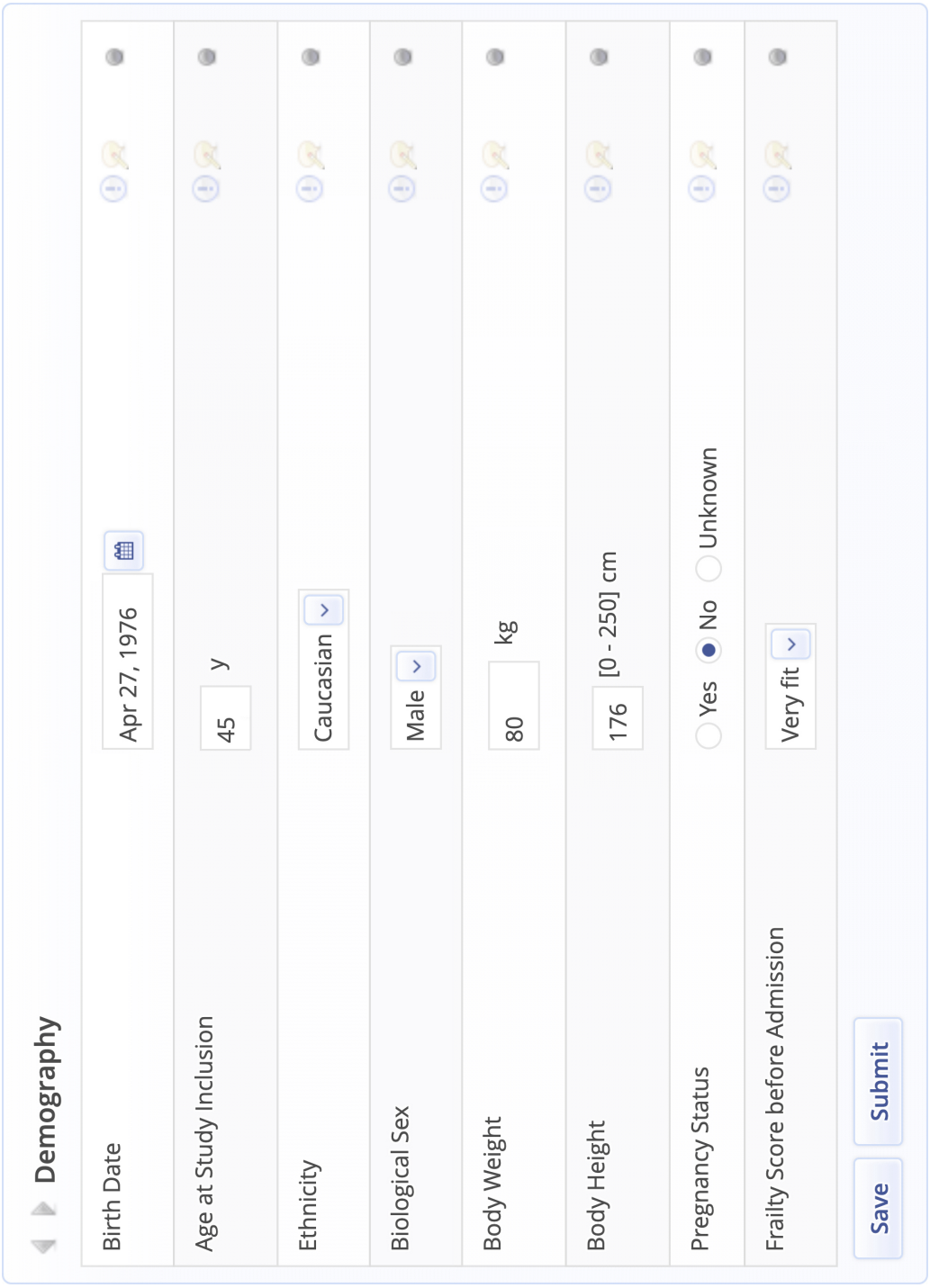


Figure 10.3.5 – *Definition of the* (*first*) *answer option* Pregnant

Chapter 11

Data Export

Finally, this chapter describes how a user exports trial data with the help of MOnSTER via the GUI as well as via a dedicated web service. The implementation details for these functionalities are shown in Chapter 7.

Since the ability to export trial data forms one of the core functionalities of any CTMS, this feature has also been an integral part of ObTiMA from its very beginning. For this, ODM was chosen as initial format, as it was and still is one of the most prominent and established standards for storing and exchanging trial data. In the initial implementation, the user could trigger the export manually via the GUI and the generated ODM could then be stored locally to be subsequently processed in special analysis tools.

As mentioned above, MOnSTER extends this functionality in two ways: First, the export format ODM itself is extended to embed semantic tags, and the formats RDF and FHIR with their different serializations are added. Secondly, the export can now be initiated not only manually in the GUI, but also triggered via a new, special web service.

Now, the following concentrates on the actual exporting via the GUI in Section 11.1 and the web service in Section 11.2, as the (logical) content and generation of the formats have already explained in detail in Section 7.2.

The foundation for this is the realization of the trial, SEs, CRFs, questions and answer options based on the GECCO dataset, as shown in the previous chapters, together with entering the respective data of ten fictional patients. After this, the trial data was exported in all available combinations of formats and serializations, by applying both the GUI and the web service. The simultaneous use of the GUI and the web service is here only due to the test purpose. In practice, of course, either the GUI or the web service are usually used, but not simultaneously.

Unfortunately, since the files resulting from the export are very large, it is not possible

to reproduce them in their entirety here. However, the export files in all formats and serializations are available for download online (Stenzhorn, 2022).

Nevertheless, for highlighting the differences and commonalities between the various formats and serializations, exemplary excerpts taken from those export files are reproduced at Appendix E. For this, the first excerpt for each format and serialization always contains the tags mentioned in Table 11.0.1 and the second excerpt the tag described in Table 11.0.2: All tags were added to the trial (description) itself, with all of them having the general meaning "COVID-19". It needs to be stressed that the actual (information) content in each excerpt is the very same, just each expressed in a different format and serialization. Note also, that to save some space, the output of labels is disabled for each first excerpts.

Ontology / Terminology	**Concept / Code**
ICD-10-GM	U07.1
LOINC	95412-3
MedDRA	10084382
NCIt	C171133
SNOMED CT	840539006

Table 11.0.1 – *Multiple tags related to COVID-19*

Among those tags, one is based on a code from MedDRA as this terminology is multilingual and hence contains labels in several languages for the selected code in Table 11.0.2 and in particular labels with distinct alphabets and characters, that is, Chinese, Japanese, Korean and Russian / Cyrillic.

Language	Label
Chinese	2019 冠状病毒疾病
Czech	Onemocnění způsobené koronavirem 2019
Dutch	Coronavirusziekte 2019
English	Coronavirus disease 2019
French	Maladie à coronavirus 2019
German	Coronavirus-Krankheit 2019
Hungarian	Koronavírus okozta megbetegedés 2019
Italian	Malattia da Coronavirus 2019
Japanese	2019 コロナウイルス病気
Korean	2019 코로나바이러스 감염증
Portuguese	Doença por coronavírus 2019
Portuguese (Brazil)	Doença pelo coronavírus de 2019
Russian	Коронавирусная инфекция 2019 года
Spanish	Enfermedad por coronavirus 2019

Table 11.0.2 – *Multilingual labels of the MedDRA code* `10084382`

11.1 GUI

The functionality to export clinical trial data in ODM format using the GUI (Graphical User Interface) has been available as key functionality of ObTiMA from its very beginning. For MOnSTER, as with semantic tagging, the goal is to keep that familiar environment as much as possible and hence reuse the existing interface and adapt only with the elements needed for providing the component's functionality.

To access the export for the current trial, the user selects first Trial from the main menu, as depicted in Figure 9.1.1, ①, and then Export from the submenu at ②.

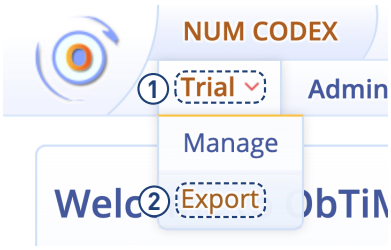


Figure 11.1.1 – *Main menu entry for exporting a trial*

After doing so, the interface of the export functionality opens, shown in Figure 11.1.2. On this page, the user can first select which SEs and CRFs to export: By activating or deactivating the checkbox at the SE level at ①, the user can select or deselect all CRFs of this SE. Also, the user can select or deselect single CRFs the same way (②).

To select from which patients the data will be actually included in the export, the user can also select or deselect individual patients (④) or all at once (⑤). If no patient at all is selected, then only the metadata of the trial is exported. (Following the notion of ODM, metadata does not only include the actual trial metadata, such as its name, start date, PI, and so on, but also the (complete) definitions of all selected SEs and CRFs.)

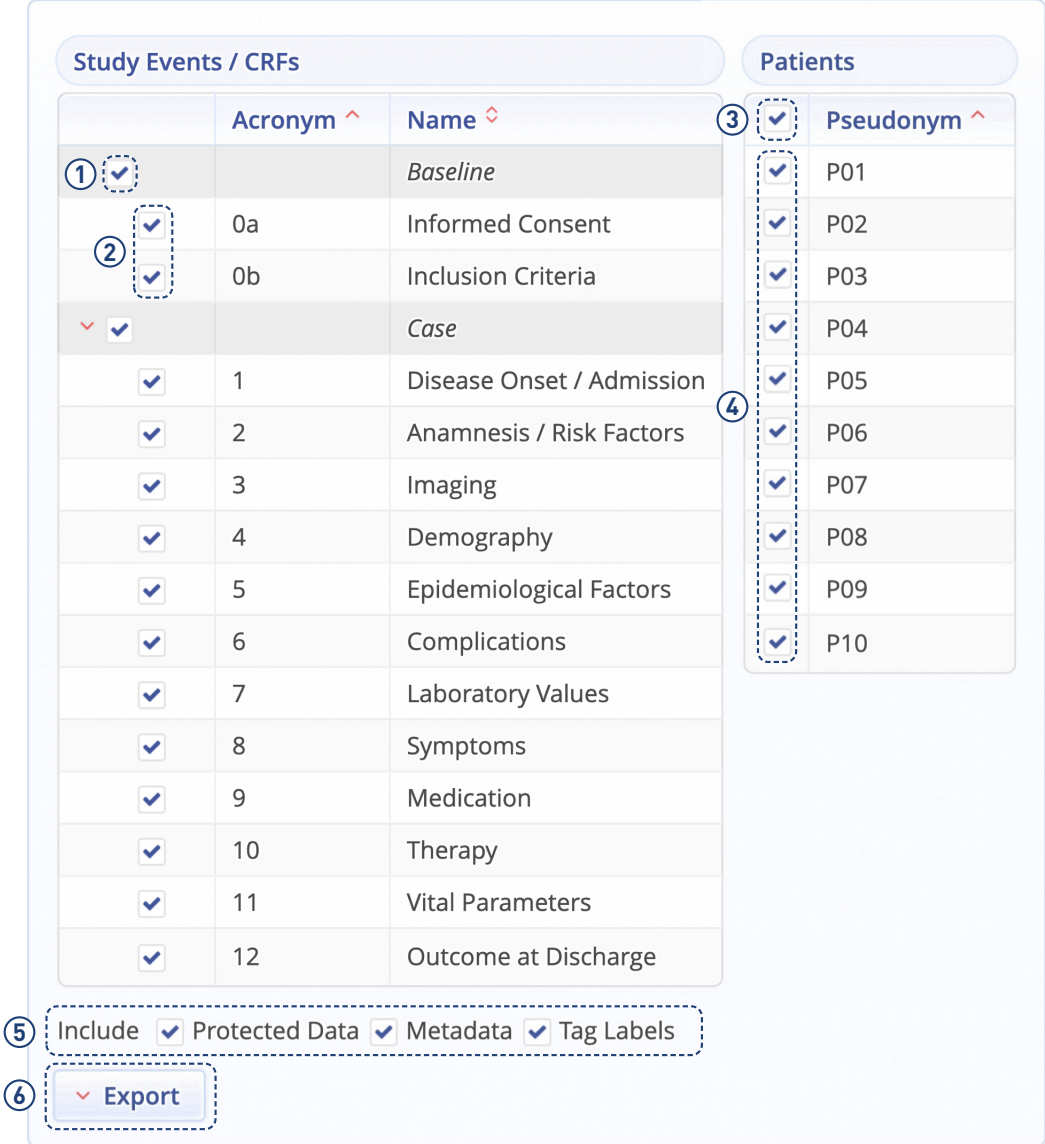


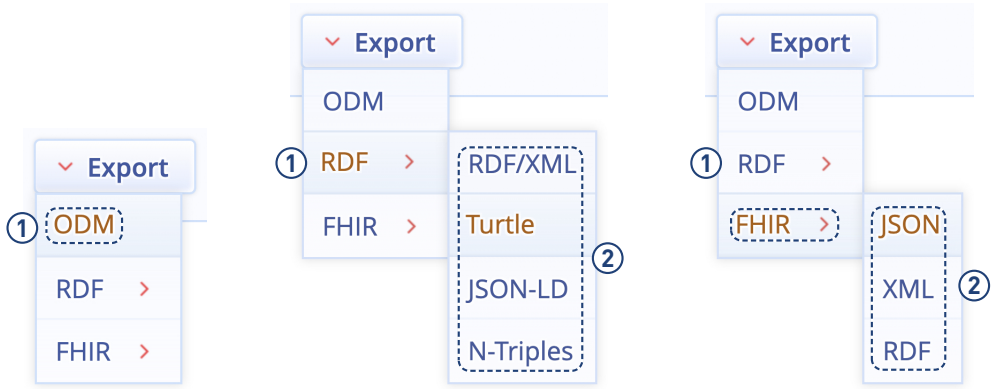
Figure 11.1.2 – *Dialog to export SEs, CRFs and patients of a trial*

Additionally some further parameters can be specified at ⑤ and which are explained in Table 11.1.1. With the help of these parameters the user can specify for specific data elements whether they should be included in the export or not.

Inclusion Option	Description
Protected Data	Include answers to questions marked as protected during the CRF creation. A question marked as such could include highly sensitive items containing unique identifiers, such as the health insurance number.
Metadata	Include not only patient data but all trial metadata, that is, the definitions of SEs, CRFs, and so forth. If no patients are selected at all for the export, then this option cannot be deselected.
Tag Labels	Include (multilingual) labels of semantic tags along with their codes / URLs. Normally, systems processing the tags are not interested in the natural-language labels but only in their actual codes. Therefore, if this option is disabled, only codes are exported, potentially rendering the export artifact smaller.

Table 11.1.1 – *Export inclusion options*

After selecting all relevant SEs, CRFs and patients and setting the desired options, the actual export is initiated by clicking on the export button at ⑥. By clicking the button, the cascaded drop-down menu, shown in Figure 11.1.3, opens: Here, if the user clicks the first ODM option ((a), ①), then the export is directly initiated. If the second option RDF is selected ((b), ①), then a sub-menu opens to chose the serialization (②), after which the export is processed. Finally, when chosing the FHIR option ((c), ①) then again the desired serialization can be selected (②) to execute the export.



(a) *ODM* (b) *RDF* (c) *FHIR*

Figure 11.1.3 – Export *button expanded for specific formats and serializations*

After initiating the export, the necessary background processing starts immediately and so all necessary trial and patient data are read from the database and mapped and transformed into the selected format and serialization as described in Chapter 7. When this process is completed after a short instance, the file just generated is ready and its download is automatically initiated. For this, the browser's file dialog opens to specify the name of the export file and the location where it should be saved.

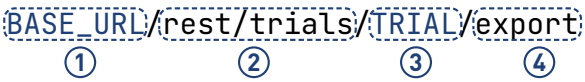
11.2 Web Service Interface

To perform the export of trial data via the GUI, a manual action by the user within the application required each time. This can, of course, be seen a major drawback, as it makes it impossible to use the export within any automated scenarios. Therefore, a web service is provided is now offering the same functionality range as the GUI but functions independently of it. To use this REST-based service meaningfully, the user should have at least a basic understanding of this approach and its usage.

To use this particular web service, the URL template in Figure 11.2.1 needs to be instantiated with the appropriate URL and query parameters according to the settings of the local environment and the desired outcome, as described in Table 11.2.1. Here the only really necessary parameter is to specify `TRIAL` at ③, specifying the acronym

of the experiment to be exported. All other (query) parameters are optional and if not specified, then the respective default value as noted in the mentioned table is applied. (The meaning of all parameters is described in the previous chapter.)

That means, that if only the trial acronym is specified without any other query parameters then the trial is exported in `format` `odm` using its standard `serialization` `xml`, with `protectedData` and `metadata` but not with `tagLabels` included, and with all patients included as `firstResult` is `0` and `maxResults` is not defined.



(a) *URL template*

(b) *Query parameters*

Figure 11.2.1 – *URL template with* (*query*) *parameters for the export web service*

URL Component	Description
`BASE_URL`	Variable for the base URL of the ObTiMA instance where the trial to export is located.
`rest/trials`	Fixed string indicating that the URL is a REST call refering to trials in the application.
`TRIAL`	Variable for the acronym of the trial to export.
`export`	Fixed string indicating the actual REST method to perform, that is, to export.

Table 11.2.1 – *Components of the export web service URL*

Query Parameter	Possible Values	Default Value
`format`	`odm`, `rdf`, `fhir`	`odm`
`serialization`	• `odm` – `xml` • `rdf` – `turtle`, `rdf_xml`, `n_triples`, `json_ld` • `fhir` – `json`, `ndjson`, `xml`, `rdf`	• `odm` – `xml` • `rdf` – `turtle` • `fhir` – `json`
`protectedData`	`true`, `false`	`true`
`metadata`	`true`, `false`	`true`
`tagLabels`	`true`, `false`	`false`
`firstResult`	≥ `0`	`0`
`maxResults`	> `0`	*undefined*

Table 11.2.2 – *Query parameters of the web service*

It needs to be stressed, of course, that the access to this web service is just as restricted and protected as the access to the export functionality within the GUI. On the one hand, this means that the service is protected in the same way as the ObTiMA GUI via basic authentication, that is, by specifying a username plus password. On the other, that specified user (account) must have the appropriate rights to export data from the specified trial too. (Since the authentication of the web service shares the one of the core system, additional authentication methods are directly available to the service once they are implemented in the core system.)

Further to this textual description, Appendix B also contains a machine-readable description of the web service based on the OpenAPI specification, as the de-facto standard for describing REST-based web service interfaces (OAI, 2021).

A small example call to the export web service is given in Figure 11.2.2. Here, the used ObTiMA instance resides at the `BASE_URL` `https://obtima.org/monster/demo` and the trial to be exported has the acronym `GECCO`. The exported file will have as its `format` `rdf` and as its `serialization` `rdf_xml`. Also, for all exported semantic tags the respective `tagLabels` are exported as well. Finally, the returned result data set starts

with the `firstResult` at (patient) `10` and `maxResults` tells that the data of at most `20` patients are returned.

```
https://obtima.org/monster/demo/rest/trials/GECCO/export?format=rdf
↪ &serialization=rdf_xml&tagLabels=true&firstResult=10&maxResults=20
```

Figure 11.2.2 – *Example of a concrete call to the web service*

Part IV

Discussion

Chapter 12
Evaluation

As mentioned in the beginning, the use of ontologies in the management of clinical trials is possibly not widespread, yet in fact nothing completely novel. Therefore, the distinguishing factor of the work here lies on its particular focus on usability and utility: This means that all functionality should be both easy to use, also for users without specific ontology or technology background, and also that all functionality should offer some concrete benefit and added value.

Now, as with any scientific work, it must be evaluated whether these goals were ultimately achieved and, if not, what the exact causes were. For this, as a single user does not necessarily come into contact with all parts of MOnSTER, and to ensure that valid and helpful results are obtained in the evaluation, it is divided in thematically. For example, a user creating CRFs is usually a medical expert, who probably rarely exports data, and is hence likely less familiar with the relevant formats. Therefore, to increase validity, evaluators should be familiar to some degree with the basic thematic background some functionality under review.

Unfortunately, such an approach is difficult to take in the context of the present evaluation: As the number of users who agree to participate in the evaluation is already low, it is imperative that all of them evaluate all areas, regardless of their prior knowledge. To mitigate this problem to some extent, all participants receive a comprehensive introduction to MOnSTER in general and to its individual functionalities, including illustrated instructions, as in Chapters 9 to 11. Also, some basic background to the topics interoperability and ontology, as in Chapter 1, is provided as well. However, it is helpful that all participants have worked with ObTiMA in the past and are thus familiar with the general functioning of the system. In order to strengthen the validity of the evaluation and to be able to draw better conclusions from it for further development, it is necessary to increase the number of participants with appropriate knowledge.

The actual realization of the evaluation follows the guidelines proposed by DAkkS

(DAkkS, 2010) and also includes results from previous evaluations of the ObTiMA core (Christ-Neumann et al., 2014) in its approach. For this purpose, the descriptions of the tasks to be performed are provided in written form and deliberately formulated as general, simple and concise as possible, so that only absolutely necessary information are given, but no additional notes or help. This procedure should make it possible to better deduce how easy and fast it is for the user to complete a certain task and at which points problems or questions arise. Also, it is important to note that the focus lies on evaluating MOnSTER itself, and so ObTiMA's core functionality is considered only if strictly necessary.

Anticipating the next sections, some general conclusions can be already drawn:

- To perform a usability and, especially, utility evaluation is a major challenge in the context of MOnSTER, as a large number of different parameters interact here, and also because of its tight, almost transparent, integration in ObTiMA.
- Without at least some prior knowledge on ontologies and interoperability, but also on managing clinical trial data in general and within ObTiMA, any sensible application of MOnSTER is extremely limited: Hence, it is absolutely necessary to train users adequately before using MOnSTER.

12.1 Ontology Management

The goal of the first part of the evaluation is to assess the functionality provided to make ontologies available to the system and to manage them.

Tasks

Here, each user needs to perform the following four tasks in the given order:

Task 1 Add GO (Gene Ontology) to the system

Task 2 Add ICD-10-GM to the system

Task 3 Add a description for GO

Task 4 Remove GO from the system

For the first two tasks, some background information about each ontology, the respective necessary parameters, and the actual ontology file is provided. In the first task, the file has OWL format, serialized in OWL/XML (Motik et al., 2012), and in the second, it has a idiosyncratic, line-based format.

Results

To start with the last two tasks, users express that they do not face any questions or problems here, and are able to perform them without any issue.

Regarding the first two tasks, both the general procedure and the GUI are essentially considered to be straightforward too by the users. Given the provided information and ontology files, they encountered no problems in specifying their parameters and adding the files. Yet, several users state major issues in two specific places:

Namespace Although values for both the prefix and the IRI are provided, their concrete origin, meaning and use is not clear without any (technical) explanation for the users.

Regular Expression Even though instructions are given and despite of additional, personal hints and support, none of the users is able to create the expression required for the second ontology.

Therefore, it must unfortunately be stated that it is difficult for common users without appropriate prior technical knowledge to add ontologies, even if the actual GUI for this is considered as not problematic. Especially creating regular expressions is considered as too complex without some more in-depth prior knowledge.

Also, independent of the actual ontology management functionality, some users state another issue: In the above tasks, the ontologies to be used are explicitly named and all necessary information as well as the ontology (file) itself are provided. Of course, this would usually not be the case in a real-word environment. The resulting problem for users is, which ontology or ontologies to use for some trial if no guidance or instructions are provided in this regard. (See also the limitations in Chapter 14.)

12.2 Semantic Tagging

The second part of the evaluation covers the functionality offered for semantically tagging the different elements of a trial.

Tasks

The user is asked to execute the following five tasks in that order:

Task 1 Enable SNOMED CT and NCIt for the current trial

Task 2 Add two tags related to COVID-19 using SNOMED CT and NCIt to the current trial overview

Task 3 Add tags to a question and its answer possibility related to pregnancy status using any enabled ontology

Task 4 Show the information and labels of the tags in the trial overview

Task 5 Remove a tag from the trial overview

As a precondition, the two mentioned ontologies are preloaded into the system.

Results

Starting again with the last two tasks, users mentioned here too that they do not have any questions or problems, and can execute them without issue. Regarding the first task, all users provided the same positive feedback as well.

For the second and third task, the users indicate once again that the actual method and the interface to semantically tag trial elements is unproblematic by itself. The realization of the concept search via text field for search terms and list area for found concepts is perceived by the users as helpful and "natural", since such an approach is known from other applications. The possibility of narrowing down the search to specific ontologies is also found helpful by users here and is therefore well used.

However, some users also indicate that searching for concepts based on string matching alone, that is, a purely lexical search, is a limiting factor for them and would also like to search semantically. For example, a query with the string wilms tumor should also return concepts where only the string nephroblastoma appears in its label. However, the aforementioned problem is mitigated in most larger ontologies by the fact that they often also provide the corresponding synonyms and spelling variants for each concept. Since these are considered by MOnSTER in the search as well, corresponding concepts are also found.

The visual representation of the concepts found is generally deemed as easy to understand, but it is criticized by some users for quickly becoming confusing when a multitude of concepts from several ontologies are returned for a search. Therefore, for future work on MOnSTER, new possibilities are being investigated to make the display of results visually clearer and, in turn, more effective for the user.

Another suggestion from users is to introduce color coding when displaying tags, that is, tags from different ontologies are to be shown in different colors. It is planned to implement this proposal as part of the next MOnSTER development iteration.

Yet, in contrast to the rather favorable feedback above, the semantic tagging is still generally perceived as highly difficult: In the vast majority of searches, several similar matching concepts are returned for some given search query, often stemming from different ontologies. In this case, the evaluating users express that is difficult or even not possible at all to judge which of those concept actually represents the right one. (See also the limitations in Chapter 14.)

12.3 Data Export

Finally, in the third part of the evaluation, the functionality offered to export trial data is reviewed. This part slightly differs from the previous two: While these mainly examined the user's interaction with MOnSTER, the actual result of the component, that is, the exported artifacts, are to be assessed as well.

Tasks

The user is asked to execute the following five tasks in that order: Here, each user needs to perform the following four tasks in the given order:

The user is prompted to perform the following three tasks, in any order:

Task 1 Export the trial in ODM format

Task 2 Export the trial in RDF format and TURTLE serialization

Task 3 Export the trial in FHIR format and JSON serialization without tag labels

As basis, users received a complete trial, that is, the trial introduced in Part III based on NUM CODEX, including all SEs, CRFs and its (artificial) patients.

Results

The opinion towards the only MOnSTER element visible here, that is, the extended export button allowing also the selection of export format and serialization, is divided among users: Although all users are able to solve the given tasks quickly and without any issues, some dislike the realization of that selection. In their view, it is not sensible that they need to click the button first and only then perform the selection. As an alternative, they suggest a separation of button and selection menus, with the latter positioned above the button and hence always visible.

Chapter 13

Limitations and Mitigations

Although both usability and usefulness are among the key aspects of MOnSTER development, some problematic aspects emerge in both the evaluation and the actual application of the component.

13.1 Usability

In the previous evaluation, the received results are indeed quite ambivalent. As mentioned above, the basic usability and usefulness of the component is generally rated as basically good by the test users. Thus, the integration of MOnSTER into the application's overall GUI is considered straightforward and the functionality to be easily accessible.

On the other hand, users still feel very overwhelmed when using the component, whereas, as already alluded to before, it is not the component itself that causes problems here: Rather, it is related to the fact that users are not at all clear about exactly which ontologies they can or should use for their particular tasks. For example, because both SNOMED CT, NCIt, or ICD-10-GM each contain specific concepts to express a COVID-19 infection, it is not obvious to the user which of these three ontologies now contains the correct concept and which concept should thus be selected. But even if a single ontology is already preselected or predefined, this often contains, and especially in the case of larger ontologies like SNOMED CT, several concepts which might all have a highly similar or possibly even the same meaning, so that is again not clear at all to the user which concept to select in the end.

However, exactly these two problems are unfortunately a well-known general and fundamental problem when using of ontologies, regardless of their concrete task or field of application, be it to add concepts to clinical research questions on CRFs as in this thesis' case (Andrews et al., 2007; Patrick et al., 2008) or when annotating laboratory data (Lin et al., 2011): As soon as the topical scope of a given ontology is not clearly

and unambiguously delimited to a specific and well-defined area, its thematic content may overlap with other ones' to some lesser or greater extent. Here, an example of an ontology whose content focuses on a very specific subject area is the HPO (Human Phenotype Ontology), which contains concepts about medically relevant phenotypes, disease phenotype annotations, and the needed algorithms for this (Köhler et al., 2021; Robinson & Mundlos, 2010). On the other hand, if SNOMED CT is considered, the exact opposite is true, as this one aims to comprehensively capture clinical documentation and thus includes such diverse domains as clinical findings, procedures, body structures, organisms, drugs, devices, and specimens, for each of which there exist also other more targeted ontologies.

In this context, the actual advantage of MOnSTER, that any ontology can be used by this component and all at the same time, can even be seen as a disadvantage here. Unfortunately, it seems highly unlikely at this point that any technical or automatic solution can be developed for this problem at all.

In order to contain and mitigate this issue, at least to some degree, when using MOnSTER, the following is suggested:

Ontology Directories There exist several services providing ontology directories and repositories, the two most prominent ones being the OLS (Ontology Lookup Service) (EMBL-EBI, 2021; Jupp et al., 2015) and the BioPortal (NCBO, 2021; Whetzel et al., 2011), which both encompass a large number of ontologies in the field of biomedicine. For each ontology included in these, the respective directory contains corresponding basic information, such as its creator, license, last update date, or the number of contained concept, and descriptions to facilitate finding a suitable ontology.

Guidelines Another option that could promise to help in both the selection of the appropriate ontolologies and the selection of the suitable concepts is the use of predefined guidelines and rules (Miñarro-Giménez et al., 2018). Here, on the one hand, exact specifications are given as to which concrete ontologies and/or concepts should be used for a specific task, or even for specific data element, and on the other hand, assistance is given for decisions in unclear cases. But even if such guidelines or rules do not yet exist or do not fit for a particular use case, it is still useful in this

case to design them yourself, especially in view of the fact that, for the specific case of MOnSTER, different people are involved in developing a trial in ObTiMA.

Another "remedy" to this issue is directly built into MOnSTER itself and basically shown at the end of Section 10.2: The component allows adding multiple semantic tags to one single trial component. So, if it is either not clear or not defined from which ontology concepts for tagging a particular component must come, or if multiple concepts in one ontology are applicable for tagging, then in these cases all tags from one or more ontologies deemed to be suitable can be added to a component.

13.2 Utility

The usefulness of MOnSTER can best be assessed in the context of its extensions to the existing export functionality. An important factor here is that the export no longer has to be executed actively and manually by the user via the GUI, but that a web service now enables its scripted execution in automated environments. Another key point that should improve the given export functionality is the extension of ODM and the inclusion of the new formats RDF and FHIR. It is hence necessary to examine the extent to which these additional formats are now actually proving useful.

ODM MOnSTER extends standard ODM by adding only very few elements, namely `tags` and `tag`, through a mechanism built into the standard itself, and leaving all other standard elements untouched. Therefore, tools capable of processing ODM, such as SAS CST (Clinical Standards Toolkit) (SAS, 2021), can ingest the extended ODM with minimal or no change at all. Admittedly, in the case of such additional extension elements, they are usually simply ignored by those tools by default. However, due to the fact that MOnSTER extends ODM, and the processing tools, such as SAS CST (Holland & Shostak, 2016), are extendable too, MOnSTER's semantic tags can be processed and productively used within analyses of trial data as well.

RDF Even though the ODM RDF vocabulary developed as part of MOnSTER orients itself towards (PhUSE-CS-STWG, 2015) in its overall structure and scope, it is not an official and supported standard. In the context of the Semantic Web, however, it is a very common and widespread approach to create specific vocabularies which are

tailored for some particular task.

For example, the LOV catalog provides many openly available task-specific vocabularies – at the time of this writing, 774 – with their corresponding metadata and descriptions in order for people wanting to create and publish data can find the vocabulary appropriate for their tasks at hand (Vandenbussche et al., 2017, 2021).

In any case, it should be ensured that such vocabularies are of high quality and comply with the aforementioned best practices for vocabulary development (Kendall et al., 2008) and publication (Berrueta et al., 2008). As noted, the development of the MOnSTER ODM vocabulary strictly adheres to these given principles.

FHIR Also regarding the FHIR mapping of ObTiMA's information model, it is not official either and based in principle "only" on the three referenced scientific publications (Doods et al., 2016; Leroux et al., 2017, 2019).

In reality, however, this point can be safely disregarded for several reasons: Indeed, not for all entities of the original information model semantically completely matching entities can be found in the target one, so that the actual information content of the original entities can still be fully expressed. Nevertheless, the chosen target entities still provide a sufficiently large coverage for all original entities and their respective attributes. Moreover, the `CRFTemplate` and `CRFInstance` core entities, as actual "data carriers" of the source model, can be completely and directly mapped onto the `Questionnaire` and `QuestionnaireResponse` entities in the target FHIR information model.

It is also important to mention that the created FHIR entities are fully standard compliant and, therefore, can be processed by FHIR tools or stored on appropriate FHIR servers without issues. For this, the FHIR generated and exported using MOnSTER is successfully validated using the official FHIR validator (HL7, 2022)

Chapter 14

Related Work

In Chapter 3 of the introduction, an overview of the overall scope and the contributions of the developed component is given. To summarize this briefly: Since its beginnings ObTiMA contains an ontology component which can be used in CRFs to create questions based on a predefined ontology, hardwired into the application, and it can export the trial data in the ODM format. MOnSTER now replaces this functionality by allowing the simultaneous use of multiple ontologies within the application for semantic tagging of all trial components and exporting the data to an (extended) ODM and now also to RDF and FHIR. This needs to be kept in mind when comparing MOnSTER concretely with other related work, as a distinction should be made here between the core functionality of ObTiMA on the one hand, and on the other, the functionality that is actually provided by the newly developed component. Nonetheless, the descriptions in the previous parts show that, to some extent, such a distinction is difficult to realize, because MOnSTER's functionality is, after all, very tightly integrated with the core application, both in terms of the GUI and underlying code.

Now, before referring to the individual related systems in the below, some general aspects and differences with regard to these systems need to be illustrated first: Because of MOnSTER's stated tight integration, it is a strict and explicit goal in its development to adopt and reuse ObTiMA's existing architecture and implementation as comprehensively as possible. Hence, an important part of this effort is the adoption of the underlying internal information model, which is logically based on the ODM standard, as mentioned in Section 7.2, and technically implemented through an ORM (Object Relational Mapping) methodology, as described in Subsection 4.2.1. To put it somewhat exaggeratedly, because of its clear ODM foundation, it is possible to say that the very core of ObTiMA's realization is not actually, as its name implies, ontology-based on the general conceptual level, and therefore does not employ any ontology-related technologies, such as OWL, to model and define its (trial-related) data elements or RDF as a means to store any of the application's (trial) data.

This is a fundamental difference from the systems below, as each of them defines all necessary trial data elements using ontologies. This means, that both the general trial components that make up every trial are defined, as well as the specific instantiations of these components to realize a particular trial in the system. As for the general components, by creating ontological concepts, it is (conceptually) defined what, for example, a CRF is and contains, or that contained trial items can either query values or provide answer options. Within ObTiMA, this task is assumed by the corresponding predefined entities of the ODM based information model, realized as ORM entity classes. As for the trial-specific realization, in the mentioned systems, either new specific concepts are derived from the general one for each trial or instances of these general concepts are created directly. When creating a new trial or new trial-specific component in ObTiMA, then for each an instance object of the corresponding ORM entity classes is created.

Now, by applying an (open) ontological methodology, these systems may be considered as generally more flexible in two ways: First, they can define and develop their own information model independently in terms of content according to their specific needs and priorities, without being restricted by external stipulations or having to take them into account. Second, when employing appropriate software libraries to process ontologies, such information models can be potentially generated and used more or less dynamically without the need to hardwire their components into the program code.

However, this flexibility also entails two corresponding disadvantages: First, if such information models are created and used in complete independence, this leads to a problem that ontologies are supposed to solve in the first place, namely that so-called data silos evolve which are incompatible and semantically not interoperable due to their disparate underlying data definitions. Second, even though dynamic information models without hard coding are ideal in theory, their comprehensive realization and practical implementation in reality is challenging though (Knublauch et al., 2006), especially in case of complex ontological structures, which in turn can also lead to performance issues when handling larger amounts of data.

Therefore, it must be weighed up whether such flexibility outweighs the advantages of a possibly more rigid standard, like ODM. In ObTiMA's case, the ODM foundation of its internal information model is very helpful, as it allows the quasi direct mapping onto

ODM for export, without the need for extensive transformations that always include the danger of possible information loss or data misinterpretation. Admittedly, in ObTiMA the ODM standard is not translated one-to-one into programming code, but in some places the resulting Java classes are slightly adapted and extended to allow for an easier and more performant processing with the relevant software libraries. However, the overall semantics of ODM and its elements is fully adopted and retained, and the mentioned changes are of technical nature only. In this respect, the "invariable" nature of these elements is not causing any major issues to date when setting-up and defining trials and their components in ObTiMA. This also allows applying the given ORM approach with fixed and independent entity classes for all information model elements, which is highly relevant for a real-world CTMS: On the one hand, this simplifies the implementation of the data management and processing, as all common Java methodologies and relevant APIs/ libraries can also be applied here. On the other hand, established, robust and scalable standard database systems can be used, which are highly beneficial for handling non-trivial amounts of data.

At this point, after what is said above, it is certainly necessary to clarify a bit more precisely what role ontologies in ObTiMA and MOnSTER play concretely in this context. As described, it is not the intention to design and develop own ontologies and concepts for defining a own information model, as this relies on an ODM basis. Rather, with the help of MOnSTER, it is intended to employ established, standard ontologies in order to apply their concepts for defining metadata of all trial-related components. In this sense, the goal of this component, from the ontological point of view, is different and maybe less ambitious, since ontologies are not used as extensively as in the other systems. Nevertheless, the component achieves exactly its intended interoperability objective through the given approach: The (re)use of ODM enables syntactic interoperability in the context of clinical trials by allowing the exchange of trial data with other CTMS and systems, as well as their integration, based on a widely used, accepted standard. Furthermore, the added support for FHIR, as another established standard, now enables syntactic interoperability alss beyond clinical trials. Semantic interoperability is, in turn, achieved by ensuring that the actual data (values) being collected and stored can also reference and be based upon standardized ontological artifacts.

Coming back to the related work, the topic of semantic interoperability in clinical trials is, of course, very broad and highly diverse, which makes it impossible to even try to address all the possibly relevant existing work within this area. Therefore, the following focuses on systems directly related to the application of ontologies in clinical trial (data) management, and thus comparable to the overall task of ObTiMA and MOnSTER.

In this, Löbe et al. (Löbe et al., 2009) proposes a system to enable the creation of CRFs and the management of clinical trial items, that is, questions in the ObTiMA naming, within a repository. For this purpose, a general vocabulary ontology was initially manually developed as its basis holding concepts to define the generic components of a trial, such as the hierarchical structure of containers for CRFs, modules and data values for input or check fields, or code lists contained therein. Based on these generic concepts, a browser-based GUI allows creating specific item instances, like for asking a creatinine level, and to store them on a RDF basis in a triplestore referencing the generic, ontological concept. From the user's perspective, the (visual) definition of such items is logically very similar to the approach used in ObTiMA, as in both cases, the GUI makes the inherent complexities the underlying ontology or information model transparent to user. However, a fundamental difference here is that in its published state, the system does not integrate any external ontologies, such as SNOMED CT, LOINC or ICD in its implementation, albeit it is deemed to be advantageous.

In contrast, the system of Esteban-Gil and Fernández-Breis is presented as a full CTMS. Thus, the system not only allows to define trials and their data collection items, but also supports the actual collection of corresponding patient data, exactly as it is the case with ObTiMA. For defining trial items, an approach similar to the previous system is taken: That is, a foundational ontology with generic trial concepts was manually created first by the authors and which needs to be extended by appropriate concepts for each new trial and its trial items. Now, initially, this ontology extension was to be performed manually, using the ontology editor Protégé (Gonçalves et al., 2021; Musen & Team, 2015). This was rejected by the users / researchers, as such work requires experts with the appropriate ontological and technical background. Therefore, a dedicated editor was developed providing a simpler GUI for defining trial items, shielding users from the ontology's complexity. Based on these definitions, browser-based CRFs are automatically generated

to allow the entry of patient data. Compared to MOnSTER, it is directly not clear from the publication whether and how the inclusion of external standard ontologies and their concepts is possible.

Another earlier system is proposed by Tran et al. (Tran et al., 2011), which represents again a complete CTMS employing ontologies as its foundation. However, unlike the previous system, this system does not provide its own GUI functionality for creating or editing a trial and its items. Although the CRFs to enter patient data are generated automatically from ontological concept definitions too, these ontologies have to be manually edited and extended for each trial and trial item. As can be seen with the previous system, this is of course a significant disadvantage, as it means that only experienced experts can develop new trials and their items.

The very same issue arises in the approach described by Shankar et al. (Shankar et al., 2006, 2007): In this case, not a single CTMS is proposed but rather a complex ontology-based architecture together with a set of ontologies is proposed to provide support for the interoperation of different software applications involved in clinical trials. For this, not each application is "ontologized" regarding their internal data processing but rather a rule-based model-database mapper is employed along with an ontological knowledge base that maps data onto a common RDBMS. Here, all ontologies as well as the mapping rules are developed and curated by ontology experts and so, as with the previous system, if additional use cases, and data (elements) are to be covered, these experts must manually extend the ontologies and the rule set.

As a side note, it may also be useful to make a comparison at a more abstract level in addition to the direct system comparison: When comparing the above with the approach of ObTiMA and MOnSTER, the interpretation of semantic versus syntactic interoperability may be blurred to some extent. As mentioned, ObTiMA's internal information model is fundamentally based on ODM, which is mentioned in Chapter 2 in the context of syntactic interoperability. That is because this standard defines the precise (XML) syntax, that syntactically well-formed ODM documents must exhibit. Nonetheless, it is equally possible to regard the ODM definitions as semantic too as the standard also provides natural language descriptions for each element to define their concrete meaning. In the same chapter ontologies are declared

as vehicle of semantic interoperability by expressing concepts unambiguously and independently of natural language terms and labels. Looking at the related systems above, the concepts of the foundational ontologies are actually not completely used in that specific sense but rather also to describe content container elements and their linking (containment) relations to mimick syntactical structures. The RDF vocabulary developed to map from the ObTiMA-internal information model onto RDF for export, as described in Subsection 7.2.2, can be seen as a small ontology following this idea too, as it (re)models ODM's XML elements and structure through concepts and relations. For example, using this vocabulary, the statement `_:some_trial odm:studyEventDefs _:some_study_event_defs` tells that `some_trial` holds the list `some_study_event_defs` of SEs definitions and so provides a structural, containment statement. Therefore, ontologies might not be used to define semantics / meaning only, but can, to some extent, also prescribe structure, and so in a certain sense a syntax. Taking this into account, there is no fundamental difference in the underlying logic between the above systems' foundational ontologies and MOnSTER's RDF vocabulary. The biggest difference is though that the used ontologies are independent, "homegrown" developments, whereas this vocabulary, albeit also a custom development, is closely adhering to the official ODM standard, and therefore the generated RDF can be considered as ODM-compliant.

To conclude this overview, it is important to also reference the system by Dugas et al. (Dugas, 2016; Dugas et al., 2016). This system proposes a web-based CRFs editor with the intention to foster the development and reuse of more standardized CRFs by resuing existing data elements as much as possible. For this, the data elements are provided together with corresponding codes and are evaluated and maintained by experts in a MDR (Metadata Registry). Here the codes originate from the UMLS (Unified Medical Language System), a repository of a multitude controlled vocabularies in the biomedical sciences (Bodenreider, 2004; NLM, 2022). To find a suitable data element in the MDR, the user specifies a string with which all data elements stored there are searched, and then can select the appropriate element from the list of returned matches. If no suitable data element is contained within the MDR, then it is possible to perform a lookup in the UMLS and create a new data element.

From the above description it is clear that the basic idea here is very similar to semantic tagging at MOnSTER. However, there are two major differences between the two in this context: First, the given system refers to the data elements within a CRF alone, like the initial ontology implementation in ObTiMA, but the very goal of MOnSTER is to enable the semantic enrichment of all components of a clinical trial. Second, in this system UMLS is fixed as the source of semantic resources. While UMLS contains a very large number of different standard resources, such as SNOMED CT or LOINC, more specialized ontologies may not be found here. MOnSTER is not restricted in such a way and allows to include any ontology, and thus can cover any subject area as long as a corresponding ontology is available.

Chapter 15

Outlook and Perspective

As with every “living” piece of software there is the wish to not only foster its current deployment but also to continue its development and add some additional features. To this end, it is planned to address the issues raised by some of the users during the evaluation as quickly as possible, and to evaluate their expressed ideas and, if possible, realize them within the component. Some of the issues and ideas that are currently under investigation are presented in the following sections.

15.1 Improved Visualizations

As stated earlier, some users express the idea that the tags added to a trial component should have different colors depending on which ontology they come from, that is, to introduce a proper color coding scheme. When doing so, an option could be added when adding or editing an ontology to select some specific color for that particular ontology. This selected color could in turn be displayed as, for example, a small marker in

- the menu to enable ontologies for a trial,
- the selection of ontologies for a specific search,
- the ontology and concept labels in the drop-down list of found concepts, and
- the tag added to a trial component.

The idea is that such a relatively small change could possibly improve the usability to quite some extent, as users could then always easily detect which ontology is currently in focus or where it is used.

In addition, it should be further investigated whether improved display options can be developed for the concept search. As some users complain during the evaluation that they are overwhelmed by the sheer number of returned concepts, it needs to be examined whether some more manageable representations are possible here.

Because of the multitude of different options in this context, any development must be preceded by a comprehensive study including the opinions of the users and evaluate the respective effectiveness of each option (Joho & Jose, 2006). This also includes the development of visual mockups in order to be able to evaluate and, if necessary, adapt the conceived solution options together with the users before their actual programmatic implementation. In this context, it will also be investigated whether some more advanced visual techniques may be feasible, for example, to visually group and cluster large numbers of results (Anderson & Wischgoll, 2020).

15.2 Ontological Relations in Search

At this point, the search for concepts for semantic tagging is based solely on their natural language labels. Within (proper) ontologies, however, there always exist additional relations between the individual concepts. One such fundamental relation is the parent-child relation between a (more general) superconcept and a (more specific) subconcept, such as in SNOMED CT where concept `840539006` (COVID-19) is a child of its parent concept `186747009` (Coronavirus infection).

Based on this, the search could then be extended so that two buttons appear in the results' drop-down list for each concept found. Now, if you click on the "parents" button, all parents of the given concept will be displayed, and one of them can be selected as a semantic tag. The same applies for the "children" button, returning the concept's children in turn. However, it is not clear to what extent such an additional option represents an actually valuable and useful feature for the user in general, or whether some advanced visual presentation would be beneficial here as well.

It is assumed that the necessary implementation could be realized with manageable effort: For this, ontologies in OWL format are unproblematic, since expressing parent-child relationships between concepts is one of their essential, native features. To enable the search based on that relationship, the indexing as described in Chapter 5 needs to be expanded: When iterating over all matching OWL classes, the unique identifying codes of the parent class or classes are extracted and added to the `parent` field of the Lucene document for that class / concept and indexed accordingly.

With respect to custom line-based ontologies, all that would be required here is the ability to express that a concept has another concept as a parent. An example snippet is given in Figure 15.2.1 to demonstrate how this could be realized in a CSV file: Here, the two concepts `398447004` and `840539006`, each with two labels, have the common parent concept `186747009`, and the concept `398447004` has an additional parent `312133006`. Then, only an additional group `parents` needs to be added to the regular expression to capture the respective value. If a concept has multiple parents, then their codes could be concatenated with a space separating each code, like `186747009 312133006`. Again, each parent concept code is added to the concept's Lucene document and indexed.

```
186747009,27619001,p,Coronavirus infection,en
312133006,275498002 34014006,p,Viral respiratory
↪ infection,en
398447004,186747009 312133006,,SARS,en
398447004,186747009 312133006,p,Severe acute
↪ respiratory syndrome,en
840539006,186747009,,Disease caused by severe
↪ acute respiratory syndrome coronavirus 2,en
840539006,186747009,p,COVID-19,en
```

(a)

```
^(?<code>.+?),(?<parents>.+?),(?<preferred>p?),
↪ (?<label>.+?),(?<language>\w{2}(-\w{2})?)$
```

(b)

Figure 15.2.1 – *Lines from an custom line-based ontology file with parent provided and suitable regular expression with the added* `parents` *group*

15.3 Tag-based CRF Repository Search

In addition to the functionality for creating and editing CRFs, ObTiMA also possesses an independent CRF repository where templates of CRFs can be stored independently of the trial being edited to make them available to other trial. If another user now creates or edits another trial, then the CRF repository can be looked up to see if a suitable CRF template can be found there, which can reused in the created / edited trial.

Currently, the repository can be searched for the CRFs based on various criteria, such as its name or its creator. Since MOnSTER now also makes it possible to tag CRFs with semantic tags as metadata, it also makes sense to extend the aforementioned search within the CRF repository accordingly. Here it is then possible to specify semantic tags and appropriately tagged CRFs will be returned.

In this context, it should be evaluated whether it is also sensible and useful to extend the search in such a way that not only the metadata of the CRF itself is searched for by the specified tags, but the semantic tags / metadata of all of its contained components too. For example, if a specific tag is provided for the search, then a CRF also matches and is already returned if only one question in the CRF contains that tag.

15.4 Rules and Guidelines

As mentioned in the previous chapter, the use of rules and guidelines can provide support for the correct selection of ontologies and concepts. Therefore, it is expected to be very helpful if they could be included as additional information in MOnSTER as well.

However, a both meaningful and user-friendly implementation of this idea is not trivial either, as there exist, for example, a multitude of different approaches to the visual representation of guidelines and rules. Thus, to provide the most suitable and helpful solution, the users must again be involved in the planning and implementation from the beginning. For this, it is planned also to create visual mockups of the different possibilities to receive feedback early and plan the actual implementation accordingly.

15.5 Training

In addition to the above, the evaluation also clearly shows that there is a definite need for further training. Whereby not only the actual use of MOnSTER is meant, that is, the correct use of the GUI for ontology management, semantic tagging and export, but also the background behind the component.

This means concretely that a suitable training and corresponding material must be

developed, which does not only cover the handling of the actual GUI elements, but rather also encompasses the provision of both conceptual and technical descriptions on interoperability and ontologies. Nevertheless, great care must be taken, of course, to ensure that these topics are handled in a concise and comprehensible manner and not to overload users with unnecessary and unnecessarily extensive information.

Part V

Appendix

Appendix A

ODM Extension Definition

The following lists the MOnSTER and ODM extension XSDs as introduced in Subsection 7.2.1. These can be used to test whether some given ODM file containing tag elements as proposed by the component is fully compliant with the prescribed element definitions.

MOnSTER XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://obtima.org/schema/odm-ext/monster/v1.0"
↪ xmlns:xs="http://www.w3.org/2001/XMLSchema"
↪ xmlns:odm="http://www.cdisc.org/ns/odm/v1.3"
↪ targetNamespace="http://obtima.org/schema/odm-ext/monster/v1.0"
↪ elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:import namespace="http://www.cdisc.org/ns/odm/v1.3"
  ↪ schemaLocation="ODM1-3-2-foundation.xsd"/>

  <xs:element name="Tags">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Tag" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Tag">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Labels"/>
      </xs:sequence>
      <xs:attribute name="URI" type="xs:anyURI"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="Labels">
```

```
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="odm:TranslatedText" minOccurs="1"
        ↪ maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

ODM Extension XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.cdisc.org/ns/odm/v1.3"
↪ xmlns:xs="http://www.w3.org/2001/XMLSchema"
↪ xmlns:monster="http://obtima.org/schema/odm-ext/monster/v1.0"
↪ targetNamespace="http://www.cdisc.org/ns/odm/v1.3">

  <xs:import namespace="http://obtima.org/schema/odm-ext/monster/v1.0"
  ↪ schemaLocation="monster-1.0.xsd"/>

  <xs:redefine schemaLocation="ODM1-3-2.xsd">
    <xs:group name="GlobalVariablesElementExtension">
      <xs:sequence>
        <xs:element ref="monster:Tags" minOccurs="0" maxOccurs="1"/>
        <xs:group ref="GlobalVariablesElementExtension"/>
      </xs:sequence>
    </xs:group>

    <xs:group name="StudyEventDefElementExtension">
      <xs:sequence>
        <xs:element ref="monster:Tags" minOccurs="0" maxOccurs="1"/>
        <xs:group ref="StudyEventDefElementExtension"/>
      </xs:sequence>
    </xs:group>

    <xs:group name="FormDefElementExtension">
      <xs:sequence>
        <xs:element ref="monster:Tags" minOccurs="0" maxOccurs="1"/>
        <xs:group ref="FormDefElementExtension"/>
      </xs:sequence>
    </xs:group>
```

```
    <xs:group name="ItemGroupDefElementExtension">
      <xs:sequence>
        <xs:element ref="monster:Tags" minOccurs="0" maxOccurs="1"/>
        <xs:group ref="ItemGroupDefElementExtension"/>
      </xs:sequence>
    </xs:group>

    <xs:group name="ItemDefElementExtension">
      <xs:sequence>
        <xs:element ref="monster:Tags" minOccurs="0" maxOccurs="1"/>
        <xs:group ref="ItemDefElementExtension"/>
      </xs:sequence>
    </xs:group>

    <xs:group name="CodeListItemElementExtension">
      <xs:sequence>
        <xs:element ref="monster:Tags" minOccurs="0" maxOccurs="1"/>
        <xs:group ref="CodeListItemElementExtension"/>
      </xs:sequence>
    </xs:group>
  </xs:redefine>

</xs:schema>
```

Appendix B

OpenAPI Description

The below lists the complete OpenAPI description as introduced in Subsection 7.2.2 and Section 11.2 which gives a machine-readable, standards compliant description of the web service interface that is provided by MOnSTER.

```
openapi: 3.1.0
info:
  version: 1.0.0
  title: ObTiMA Export REST API Documentation
servers:
  - url: '{scheme}://{host}:{port}/{basePath}'
    variables:
      scheme:
        enum: [http, https]
        default: https
      host:
        default: obtima.org
      basePath:
        default: test/rest
      port:
        default: '443'
paths:
  "/trials/{trialAcronym}/export":
    get:
      summary: Export a trial
      parameters:
        - name: trialAcronym
          in: path
          description: Acronym of the trial
          required: true
          schema:
            type: string
        - name: format
          in: query
          description: Target of the export
          required: false
          schema:
```

```
      type: string
      enum: [odm, fhir, rdf]
      default: odm
  - name: serialization
    in: query
    description: Format of the export
    required: false
    schema:
      type: string
      enum: [xml, turtle, rdf_xml, n_triples, json_ld, json,
      ↪ rdf]
      default: xml
  - name: protectedData
    in: query
    description: Include protected data
    required: false
    schema:
      type: boolean
      default: true
  - name: metadata
    in: query
    description: Include trial metadata including definitions of
    ↪ study events and case report forms
    required: false
    schema:
      type: boolean
      default: true
  - name: firstResult
    in: query
    description: First result to return
    required: false
    schema:
      type: integer
      minimum: 0
      default: 0
  - name: maxResults
    in: query
    description: Maximum number of results returned
    required: false
    schema:
      type: integer
      minimum: 1
      default: 0x7fffffff
```

```
responses:
  "200":
    description: OK
    content:
      application/xml: {}
      application/ld+json: {}
      application/json: {}
      application/rdf+xml: {}
      text/turtle: {}
      application/n-triples: {}
  "401":
    description: Unauthorized
  "403":
    description: Export of trial denied
  "404":
    description: Trial not found
  "500":
    description: Export failed
  "501":
    description: Format not supported
  "503":
    description: RDF export failed because MOnSTER is disabled
```

Appendix C

Applied Specifications

The table below provides references to all of the relevant specifications employed in the development and application of MOnSTER. These encompass the formats and serializations that the export is capable to produce (ODM, RDF and FHIR), the format of ontologies that can be loaded by the component (OWL), the format used for creating the technical webservice description (OpenAPI), and the format employed for the creating the code visualizations in this thesis' descriptions (UML).

Name	Version	Serialization	Reference
FHIR	R4 (4.0.1)		(HL7, 2019)
ODM	1.3.2		(CDISC, 2013)
OpenAPI	3.1.0		(OAI, 2021)
OWL	2		(W3C-OWG, 2012)
RDF	1.1		(Cyganiak et al., 2014)
		RDF/XML	(Gandon et al., 2014)
		Turtle	(Beckett et al., 2014)
		JSON-LD	(Sporny et al., 2020)
		N-Triples	(Beckett, 2014)
UML	2.5.1		(OMG, 2017)

Table C.1 – *Specifications used for developing and in applying MOnSTER*

Appendix D

Applied Libraries and Licenses

The following tables list all of the software libraries used for developing and in applying MOnSTER along with the respective license employed by each. Their actual use in the component's implementation is described in Part II.

In this context, it must be emphasized that when selecting these libraries, the primary consideration is, of course, their functionality and usefulness, but another important factor is their licensing: Great care is taken to select libraries that have open source licenses that are permissive in the sense that they do not restrict their application to specific usage scenarios, such as exclusive use in non-commercial environments, or that force developers to distribute an application's source code along with its binary artifacts.

Name	Version	License	Reference
Spring Framework	5.3.16	Apache-2.0	(VMware, 2022a)
JPA	2.2.3	EPL-2.0	(Eclipse, 2022c)
Hibernate ORM	5.6.5	LGPL-3.0	(Red Hat, 2022)
JSF	2.3.17	EPL-2.0	(Eclipse, 2022d, 2022g)
PrimeFaces	8.0.0	MIT	(PrimeTek, 2022)
JAXB	2.3.6	EDL-1.0	(Eclipse, 2022a, 2022e)
OWL API	5.1.20	Apache-2.0	(Horridge et al., 2022)
RDF4J	3.7.4	EDL-1.0	(Eclipse, 2022h)
Jackson	2.13.1	Apache-2.0	(FasterXML, 2022)
HAPI FHIR	5.7.0	Apache-2.0	(HAPI FHIR, 2022)
Lucene	9.0.0	Apache-2.0	(ASF, 2022a)

Table D.1 – *Libraries used for developing and in applying MOnSTER*

Name	Version	SPDX	Reference
Apache License	2.0	Apache-2.0	(ASF, 2014)
Eclipse Distribution License	1.0	EDL-1.0	(Eclipse, 2007)
Eclipse Public License	2.0	EPL-2.0	(Eclipse, 2017)
(GNU) Lesser General Public License	3.0	LGPL-3.0	(FSF, 2007)
MIT License	-	MIT	(OSI, 1987)

Table D.2 – *Licenses applied by the libraries listed in Table D.1*

Appendix E

Applied Ontologies

The table below provides a list of all ontologies successfully tested with the component until now. These include both the ones referenced and applied for the GECCO use case Chapter 12, but also additional ones that are employed in other projects and for testing purposes only. In this context, successful testing means that, without any kind of problems, an ontology can be loaded into the application and that any component of a trial can be semantically tagged with concepts from that given ontology.

For the sake of brevity, only the latest version of each tested ontology is listed in the below, even if earlier versions were successfully tested as well before.

Name	Version	Reference
ChEBI	2021-12-01	(EMBL-EBI, 2018)
CTCAE	5.0	(NCI, 2017)
FMA	5.0.0	(SIG, 2019)
GO	2021-11-16	(GOC, 2021)
ICD-10-GM	2021	(BfArM, 2022a)
LOINC	2.71	(Regenstrief, 2022)
MedDRA	24.1	(ICH-TRPHU, 2022)
NCIt	21.11	(NCI, 2022)
OBI	2021-08-18	(OBI, 2021)
OPS	2021	(BfArM, 2022b)
ORDO	4.0	(Orphanet, 2022)
SNOMED CT	2022-01	(SNOMED, 2022)

Table E.1 – *Ontologies succesfully tested and used in MOnSTER*

Appendix F

Data Export Excerpts

The following are exemplary excerpts from data exports in all formats and respective serializations which MOnSTER currently supports, that is, in ODM, RDF serialized in RDF/XML, Turtle, JSON-LD, and N-Triples, and FHIR serialized in JSON, XML, and RDF/Turtle. A thematic background on their overall content is given in Chapter 8 and details on their actual, concrete content in Section 11.1. All files in all formats and serializations are available in their entirety online, see (Stenzhorn, 2022).

F.1 ODM

Multiple Tags without Labels

```
<ODM FileType="Snapshot" Granularity="All" FileOID="1939501217"
↪ CreationDateTime="1111-11-11T11:11:11.111+01:00" ODMVersion="1.3.2"
↪ SourceSystem="ObTiMA" xmlns="http://www.cdisc.org/ns/odm/v1.3"
↪ xmlns:monster="http://obtima.org/schema/odm-ext/monster/v1.0">
  <Study OID="Project.NUMCODEX">
    <GlobalVariables>
      <StudyName>NUM CODEX (NUM CODEX)</StudyName>
      ...
      <monster:Tags>
        <monster:Tag
        ↪ URI="http://terminology.hl7.org/CodeSystem/mdr/10084382"/>
        <monster:Tag
        ↪ URI="http://fhir.de/CodeSystem/bfarm/icd-10-gm/U07.1"/>
        <monster:Tag URI="http://loinc.org/95412-3"/>
        <monster:Tag URI="http://snomed.info/id/840539006"/>
        <monster:Tag URI="http://ncicb.nci.nih.gov/xml/owl/EVS/
        ↪ Thesaurus.owl#C171133"/>
      </monster:Tags>
    </GlobalVariables>
    ...
  </Study>
</ODM>
```

Single Tag with Multilingual Labels

```
<monster:Tag URI="http://terminology.hl7.org/CodeSystem/mdr/10084382">
  <monster:Labels>
    <TranslatedText xml:lang="cn">2019 冠状病毒疾病</TranslatedText>
    <TranslatedText xml:lang="cz">Onemocnění způsobené koronavirem
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="de">Coronavirus-Krankheit
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="en">Coronavirus disease
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="es">Enfermedad por coronavirus
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="fr">Maladie à coronavirus
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="hu">Koronavírus okozta megbetegedés
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="it">Malattia da Coronavirus
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="jp">2019
    ↪ コロナウイルス病気</TranslatedText>
    <TranslatedText xml:lang="ko">2019 코로나바이러스
    ↪ 감염증</TranslatedText>
    <TranslatedText xml:lang="nl">Coronavirusziekte
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="pt-BR">Doença pelo coronavírus de
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="pt">Doença por coronavírus
    ↪ 2019</TranslatedText>
    <TranslatedText xml:lang="ru">Коронавирусная инфекция 2019
    ↪ года</TranslatedText>
  </monster:Labels>
</monster:Tag>
```

F.2 RDF

F.2.1 RDF/XML

Multiple Tags without Labels

```
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
↪ xmlns:export="https://obtima.org/export/"
↪ xmlns:odm="https://rdf.cdisc.org/odm#"
↪ xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
↪ xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 <rdf:Description rdf:about="https://obtima.org/export/1939501217">
  <rdf:type rdf:resource="https://obtima.org/export/Export"/>
  ...
  <export:contains
  ↪ rdf:resource="https://obtima.org/export/81075958-164"/>
 </rdf:Description>
 ...
 <rdf:Description rdf:about="https://obtima.org/export/81075958-164">
  <rdf:type rdf:resource="https://rdf.cdisc.org/odm#Study"/>
  <odm:acronym>NUM CODEX</odm:acronym>
  <odm:name>NUM CODEX</odm:name>
  ...
  <odm:tags rdf:nodeID="node1fna3gt02x130343"/>
 </rdf:Description>
 ...
 <rdf:Description rdf:nodeID="node1fna3gt02x130343">
  <rdf:type
  ↪ rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
  <rdf:first rdf:resource="https://obtima.org/export/83834-1070"/>
  <rdf:rest rdf:nodeID="node1fna3gt02x130344"/>
 </rdf:Description>
 <rdf:Description rdf:nodeID="node1fna3gt02x130344">
  <rdf:first rdf:resource="https://obtima.org/export/83834-1071"/>
  <rdf:rest rdf:nodeID="node1fna3gt02x130345"/>
 </rdf:Description>
 <rdf:Description rdf:nodeID="node1fna3gt02x130345">
  <rdf:first rdf:resource="https://obtima.org/export/83834-1072"/>
  <rdf:rest rdf:nodeID="node1fna3gt02x130346"/>
 </rdf:Description>
 <rdf:Description rdf:nodeID="node1fna3gt02x130346">
  <rdf:first rdf:resource="https://obtima.org/export/83834-1073"/>
```

```
  <rdf:rest rdf:nodeID="node1fna3gt02x130347"/>
 </rdf:Description>
 <rdf:Description rdf:nodeID="node1fna3gt02x130347">
  <rdf:first rdf:resource="https://obtima.org/export/83834-1074"/>
  <rdf:rest
 ↪ rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
 </rdf:Description>
 ...
 <rdf:Description rdf:about="https://obtima.org/export/83834-1070">
  <rdf:type rdf:resource="https://rdf.cdisc.org/odm#Tag"/>
  <rdfs:seeAlso
 ↪ rdf:resource="http://terminology.hl7.org/CodeSystem/mdr/10084382"/>
 </rdf:Description>
 <rdf:Description rdf:about="https://obtima.org/export/83834-1071">
  <rdf:type rdf:resource="https://rdf.cdisc.org/odm#Tag"/>
  <rdfs:seeAlso
 ↪ rdf:resource="http://fhir.de/CodeSystem/bfarm/icd-10-gm/U07.1"/>
 </rdf:Description>
 <rdf:Description rdf:about="https://obtima.org/export/83834-1072">
  <rdf:type rdf:resource="https://rdf.cdisc.org/odm#Tag"/>
  <rdfs:seeAlso rdf:resource="http://loinc.org/95412-3"/>
 </rdf:Description>
 <rdf:Description rdf:about="https://obtima.org/export/83834-1073">
  <rdf:type rdf:resource="https://rdf.cdisc.org/odm#Tag"/>
  <rdfs:seeAlso rdf:resource="http://snomed.info/id/840539006"/>
 </rdf:Description>
 <rdf:Description rdf:about="https://obtima.org/export/83834-1074">
  <rdf:type rdf:resource="https://rdf.cdisc.org/odm#Tag"/>
  <rdfs:seeAlso
 ↪ rdf:resource="http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl
 ↪ #C171133"/>
 </rdf:Description>
 ...
</rdf:RDF>
```

Single Tag with Multilingual Labels

```
<rdf:Description rdf:about="https://obtima.org/export/83834-1070">
 <rdf:type rdf:resource="https://rdf.cdisc.org/odm#Tag"/>
 <rdfs:label xml:lang="cn">2019 冠状病毒疾病</rdfs:label>
 <rdfs:label xml:lang="cz">Onemocnění způsobené koronavirem
↪ 2019</rdfs:label>
 <rdfs:label xml:lang="de">Coronavirus-Krankheit 2019</rdfs:label>
```

```
 <rdfs:label xml:lang="en">Coronavirus disease 2019</rdfs:label>
 <rdfs:label xml:lang="es">Enfermedad por coronavirus
↪ 2019</rdfs:label>
 <rdfs:label xml:lang="fr">Maladie à coronavirus 2019</rdfs:label>
 <rdfs:label xml:lang="hu">Koronavírus okozta megbetegedés
↪ 2019</rdfs:label>
 <rdfs:label xml:lang="it">Malattia da Coronavirus 2019</rdfs:label>
 <rdfs:label xml:lang="jp">2019 コロナウイルス病気</rdfs:label>
 <rdfs:label xml:lang="ko">2019 코로나바이러스 감염증</rdfs:label>
 <rdfs:label xml:lang="nl">Coronavirusziekte 2019</rdfs:label>
 <rdfs:label xml:lang="pt-BR">Doença pelo coronavírus de
↪ 2019</rdfs:label>
 <rdfs:label xml:lang="pt">Doença por coronavírus 2019</rdfs:label>
 <rdfs:label xml:lang="ru">Коронавирусная инфекция 2019
↪ года</rdfs:label>
 <rdfs:seeAlso
↪ rdf:resource="http://terminology.hl7.org/CodeSystem/mdr/10084382"/>
</rdf:Description>
```

F.2.2 Turtle

Multiple Tags without Labels

```
@prefix export: <https://obtima.org/export/> .
@prefix icd-10-gm: <http://fhir.de/CodeSystem/bfarm/icd-10-gm/> .
@prefix loinc: <http://loinc.org/> .
@prefix meddra: <http://terminology.hl7.org/CodeSystem/mdr/> .
@prefix ncit: <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#> .
@prefix odm: <https://rdf.cdisc.org/odm#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix snomed-ct: <http://snomed.info/id/> .
...
export:1939501217 a export:Export; ... export:contains
↪ export:81075958-164 .
...
export:81075958-164 a odm:Study; odm:acronym "NUM CODEX"; odm:name
↪ "NUM CODEX"; ... odm:tags _:node1 .
...
_:node1 a rdf:List; rdf:first export:83834-1070; rdf:rest _:node2 .
_:node2 rdf:first export:83834-1071; rdf:rest _:node3 .
_:node3 rdf:first export:83834-1072; rdf:rest _:node4 .
```

```
_:node4 rdf:first export:83834-1073; rdf:rest _:node5 .
_:node5 rdf:first export:83834-1074; rdf:rest rdf:nil .
...
export:83834-1070 a odm:Tag; rdfs:seeAlso meddra:10084382 .
export:83834-1071 a odm:Tag; rdfs:seeAlso icd-10-gm:U07.1 .
export:83834-1072 a odm:Tag; rdfs:seeAlso loinc:95412-3 .
export:83834-1073 a odm:Tag; rdfs:seeAlso snomed-ct:840539006 .
export:83834-1074 a odm:Tag; rdfs:seeAlso ncit:C171133 .
...
```

Single Tag with Multilingual Labels

```
export:83834-1070 a odm:Tag; rdfs:seeAlso meddra:10084382 . rdfs:label
↪ "2019 冠状病毒疾病"@cn, "Onemocnění způsobené koronavirem 2019"@cz,
↪ "Coronavirus-Krankheit 2019"@de,"Coronavirus disease 2019"@en,
↪ "Enfermedad por coronavirus 2019"@es, "Maladie à coronavirus
↪ 2019"@fr, "Koronavírus okozta megbetegedés 2019"@hu, "Malattia da
↪ Coronavirus 2019"@it, "2019 コロナウイルス病気"@jp, "2019
↪ 코로나바이러스 감염증"@ko, "Coronavirusziekte 2019"@nl, "Doença pelo
↪ coronavírus de 2019"@pt-BR, "Doença por coronavírus 2019"@pt,
↪ "Коронавирусная инфекция 2019 года"@ru;
```

F.2.3 JSON-LD

Multiple Tags without Labels

```
[
 {
  "@id" : "https://obtima.org/export/1939501217",
  "@type" : [ "https://obtima.org/export/Export" ],
  ...
  "https://obtima.org/export/contains" : [ {
   "@id" : "https://obtima.org/export/81075958-164"
  } ]
 },
 ...
 {
  "@id" : "https://obtima.org/export/81075958-164",
  "@type" : [ "https://rdf.cdisc.org/odm#Study" ],
  "https://rdf.cdisc.org/odm#acronym" : [ { "@value" : "NUM CODEX" }
  ↪ ],
  "https://rdf.cdisc.org/odm#name" : [ { "@value" : "NUM CODEX" } ],
```

```
  ...
  "https://rdf.cdisc.org/odm#tags" : [ {
   "@list" : [ { "@id" : "https://obtima.org/export/83834-1070" },
         { "@id" : "https://obtima.org/export/83834-1071" },
         { "@id" : "https://obtima.org/export/83834-1072" },
         { "@id" : "https://obtima.org/export/83834-1073" },
         { "@id" : "https://obtima.org/export/83834-1079" } ]
  } ]
 },
 ...
 {
  "@id" : "https://obtima.org/export/83834-1070",
  "@type" : [ "https://rdf.cdisc.org/odm#Tag" ],
  "http://www.w3.org/2000/01/rdf-schema#seeAlso" : [ {
   "@id" : "http://terminology.hl7.org/CodeSystem/mdr/10084382"
  } ]
 }, {
  "@id" : "https://obtima.org/export/83834-1071",
  "@type" : [ "https://rdf.cdisc.org/odm#Tag" ],
  "http://www.w3.org/2000/01/rdf-schema#seeAlso" : [ {
   "@id" : "http://fhir.de/CodeSystem/bfarm/icd-10-gm/U07.1"
  } ]
 }, {
  "@id" : "https://obtima.org/export/83834-1072",
  "@type" : [ "https://rdf.cdisc.org/odm#Tag" ],
  "http://www.w3.org/2000/01/rdf-schema#seeAlso" : [ {
   "@id" : "http://loinc.org/95412-3"
  } ]
 }, {
  "@id" : "https://obtima.org/export/83834-1073",
  "@type" : [ "https://rdf.cdisc.org/odm#Tag" ],
  "http://www.w3.org/2000/01/rdf-schema#seeAlso" : [ {
   "@id" : "http://snomed.info/id/840539006"
  } ]
 }, {
  "@id" : "https://obtima.org/export/83834-1074",
  "@type" : [ "https://rdf.cdisc.org/odm#Tag" ],
  "http://www.w3.org/2000/01/rdf-schema#seeAlso" : [ {
   "@id" :
  ↪ "http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C171133"
  } ]
 },
 ...
```

```
]
```

Single Tag with Multilingual Labels

```
{
 "@id" : "https://obtima.org/export/83834-1070",
 "@type" : [ "https://rdf.cdisc.org/odm#Tag" ],
 "http://www.w3.org/2000/01/rdf-schema#label" : [
  { "@language" : "cn", "@value" : "2019 冠状病毒疾病" },
  { "@language" : "cz", "@value" : "Onemocnění způsobené koronavirem
 ↪ 2019" },
  { "@language" : "de", "@value" : "Coronavirus-Krankheit 2019" },
  { "@language" : "en", "@value" : "Coronavirus disease 2019" },
  { "@language" : "es", "@value" : "Enfermedad por coronavirus 2019"
 ↪ },
  { "@language" : "fr", "@value" : "Maladie à coronavirus 2019" },
  { "@language" : "hu", "@value" : "Koronavírus okozta megbetegedés
 ↪ 2019" },
  { "@language" : "it", "@value" : "Malattia da Coronavirus 2019" },
  { "@language" : "jp", "@value" : "2019 コロナウイルス病気" },
  { "@language" : "ko", "@value" : "2019 코로나바이러스 감염증" },
  { "@language" : "nl", "@value" : "Coronavirusziekte 2019" },
  { "@language" : "pt-br", "@value" : "Doença pelo coronavírus de
 ↪ 2019" },
  { "@language" : "pt", "@value" : "Doença por coronavírus 2019" },
  { "@language" : "ru", "@value" : "Коронавирусная инфекция 2019 года"
 ↪ } ],
 "http://www.w3.org/2000/01/rdf-schema#seeAlso" : [ {
  "@id" : "http://terminology.hl7.org/CodeSystem/mdr/10084382"
 } ]
}
```

F.2.4 N-Triples

Multiple Tags without Labels

```
<https://obtima.org/export/1939501217>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪ <https://obtima.org/export/Export> .
<https://obtima.org/export/1939501217>
↪ <https://obtima.org/export/contains>
↪ <https://obtima.org/export/81075958-164> .
```

```
...
<https://obtima.org/export/81075958-164>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪ <https://rdf.cdisc.org/odm#Study> .
<https://obtima.org/export/81075958-164>
↪ <https://rdf.cdisc.org/odm#acronym> "NUM CODEX" .
<https://obtima.org/export/81075958-164>
↪ <https://rdf.cdisc.org/odm#name> "NUM CODEX" .
...
<https://obtima.org/export/81075958-164>
↪ <https://rdf.cdisc.org/odm#tags> _:node1 .
...
_:node1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#List> .
_:node1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first>
↪ <https://obtima.org/export/83834-1070> .
_:node1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:node2 .
_:node2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first>
↪ <https://obtima.org/export/83834-1071> .
_:node2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:node3 .
_:node3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first>
↪ <https://obtima.org/export/83834-1072> .
_:node3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:node4 .
_:node4 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first>
↪ <https://obtima.org/export/83834-1073> .
_:node4 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:node5 .
_:node5 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first>
↪ <https://obtima.org/export/83834-1074> .
_:node5 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#nil> .
...
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪ <https://rdf.cdisc.org/odm#Tag> .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#seeAlso>
↪ <http://terminology.hl7.org/CodeSystem/mdr/10084382> .
<https://obtima.org/export/83834-1071>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪ <https://rdf.cdisc.org/odm#Tag> .
<https://obtima.org/export/83834-1071>
↪ <http://www.w3.org/2000/01/rdf-schema#seeAlso>
↪ <http://fhir.de/CodeSystem/bfarm/icd-10-gm/U07.1> .
```

```
<https://obtima.org/export/83834-1072>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪ <https://rdf.cdisc.org/odm#Tag> .
<https://obtima.org/export/83834-1072>
↪ <http://www.w3.org/2000/01/rdf-schema#seeAlso>
↪ <http://loinc.org/95412-3> .
<https://obtima.org/export/83834-1073>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪ <https://rdf.cdisc.org/odm#Tag> .
<https://obtima.org/export/83834-1073>
↪ <http://www.w3.org/2000/01/rdf-schema#seeAlso>
↪ <http://snomed.info/id/840539006> .
<https://obtima.org/export/83834-1074>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪ <https://rdf.cdisc.org/odm#Tag> .
<https://obtima.org/export/83834-1074>
↪ <http://www.w3.org/2000/01/rdf-schema#seeAlso>
↪ <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C171133> .
...
```

Single Tag with Multilingual Labels

```
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪ <https://rdf.cdisc.org/odm#Tag> .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#seeAlso>
↪ <http://terminology.hl7.org/CodeSystem/mdr/10084382> .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "2019 冠状病毒疾病"@cn .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Onemocnění způsobené
↪ koronavirem 2019"@cz .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Coronavirus-Krankheit
↪ 2019"@de .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Coronavirus disease
↪ 2019"@en .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Enfermedad por
↪ coronavirus 2019"@es .
```

```
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Maladie à coronavirus
↪ 2019"@fr .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Koronavírus okozta
↪ megbetegedés 2019"@hu .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Malattia da
↪ Coronavirus 2019"@it .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "2019
↪ コロナウイルス病気"@jp .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "2019 코로나바이러스
↪ 감염증"@ko .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Coronavirusziekte
↪ 2019"@nl .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Doença pelo
↪ coronavírus de 2019"@pt-BR .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Doença por coronavírus
↪ 2019"@pt .
<https://obtima.org/export/83834-1070>
↪ <http://www.w3.org/2000/01/rdf-schema#label> "Коронавирусная
↪ инфекция 2019 года"@ru .
```

F.3 FHIR

F.3.1 JSON

Multiple Tags without Labels

```
{
  "resourceType": "Bundle",
  "id": "d62042b2c2b6cabc",
  "type": "transaction",
  "entry": [ {
    "fullUrl": "0d15be86abd7eea5",
    "resource": {
```

```
    "resourceType": "CarePlan",
    "id": "0d15be86abd7eea5",
    "identifier": [ {
      "system": "https://obtima.org/fhir/NamingSystem/acronym",
      "value": "NUM CODEX"
    }, {
      "system": "https://obtima.org/fhir/NamingSystem/name",
      "value": "NUM CODEX"
    } ],
    ...
    "addresses": [ { "reference": "Condition/f3ebecd70f219cd9" },
                   { "reference": "Condition/3b7d56d64176a370" },
                   { "reference": "Condition/af52d0d05763d100" },
                   { "reference": "Condition/00f3a1c86a3dc1a9" },
                   { "reference": "Condition/2077111e09bb927a" } ],
    ...
  }
}, {
...
}, {
  "fullUrl": "f3ebecd70f219cd9",
  "resource": {
    "resourceType": "Condition",
    "id": "f3ebecd70f219cd9",
    "code": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/mdr",
        "code": "10084382"
      } ]
    }
  }
}, {
  "fullUrl": "3b7d56d64176a370",
  "resource": {
    "resourceType": "Condition",
    "id": "3b7d56d64176a370",
    "code": {
      "coding": [ {
        "system": "http://fhir.de/CodeSystem/bfarm/icd-10-gm",
        "code": "U07.1"
      } ]
    }
  }
```

```
  }, {
    "fullUrl": "af52d0d05763d100",
    "resource": {
      "resourceType": "Condition",
      "id": "af52d0d05763d100",
      "code": {
        "coding": [ {
          "system": "http://loinc.org",
          "code": "95412-3"
        } ]
      }
    }
  }, {
    "fullUrl": "00f3a1c86a3dc1a9",
    "resource": {
      "resourceType": "Condition",
      "id": "00f3a1c86a3dc1a9",
      "code": {
        "coding": [ {
          "system": "http://snomed.info/id",
          "code": "840539006"
        } ]
      }
    }
  }, {
    "fullUrl": "2077111e09bb927a",
    "resource": {
      "resourceType": "Condition",
      "id": "2077111e09bb927a",
      "code": {
        "coding": [ {
          "system": "http://ncithesaurus-stage.nci.nih.gov",
          "code": "C171133"
        } ]
      }
    }
  }, {
  ...
  } ],
  ...
}
```

Single Tag with Multilingual Labels


```
{
  "fullUrl": "f3ebecd70f219cd9",
  "resource": {
    "resourceType": "Condition",
    "id": "f3ebecd70f219cd9",
    "code": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/mdr",
        "code": "10084382",
        "display": "Coronavirus-Krankheit 2019",
        "_display": {
          "extension": [ {
            "url":
            ↪ "http://hl7.org/fhir/StructureDefinition/translation",
            "extension": [ { "url": "lang",  "valueCode": "cn" }, {
              "url": "content",
              "valueString": "2019 冠状病毒疾病"
            } ]
          }, {
            "url":
            ↪ "http://hl7.org/fhir/StructureDefinition/translation",
            "extension": [ { "url": "lang",  "valueCode": "cz" }, {
              "url": "content",
              "valueString": "Onemocnění způsobené koronavirem 2019"
            } ]
          }, {
            "url":
            ↪ "http://hl7.org/fhir/StructureDefinition/translation",
            "extension": [ { "url": "lang", "valueCode": "de" }, {
              "url": "content",
              "valueString": "Coronavirus-Krankheit 2019"
            } ]
          }, {
            "url":
            ↪ "http://hl7.org/fhir/StructureDefinition/translation",
            "extension": [ { "url": "lang", "valueCode": "en" }, {
              "url": "content",
              "valueString": "Coronavirus disease 2019"
            } ]
          }, {
```

```
      "url":
      ↪ "http://hl7.org/fhir/StructureDefinition/translation",
      "extension": [ { "url": "lang", "valueCode": "es" }, {
        "url": "content",
        "valueString": "Enfermedad por coronavirus 2019"
      } ]
    }, {
      "url":
      ↪ "http://hl7.org/fhir/StructureDefinition/translation",
      "extension": [ { "url": "lang", "valueCode": "fr" }, {
        "url": "content",
        "valueString": "Maladie à coronavirus 2019"
      } ]
    }, {
      "url":
      ↪ "http://hl7.org/fhir/StructureDefinition/translation",
      "extension": [ { "url": "lang", "valueCode": "hu" }, {
        "url": "content",
        "valueString": "Koronavírus okozta megbetegedés 2019"
      } ]
    }, {
      "url":
      ↪ "http://hl7.org/fhir/StructureDefinition/translation",
      "extension": [ { "url": "lang", "valueCode": "it" }, {
        "url": "content",
        "valueString": "Malattia da Coronavirus 2019"
      } ]
    }, {
      "url":
      ↪ "http://hl7.org/fhir/StructureDefinition/translation",
      "extension": [ { "url": "lang", "valueCode": "jp" }, {
        "url": "content",
        "valueString": "2019 コロナウイルス病気"
      } ]
    }, {
      "url":
      ↪ "http://hl7.org/fhir/StructureDefinition/translation",
      "extension": [ { "url": "lang", "valueCode": "ko" }, {
        "url": "content",
        "valueString": "2019 코로나바이러스 감염증"
      } ]
    }, {
```

```
            "url":
            ↪ "http://hl7.org/fhir/StructureDefinition/translation",
            "extension": [ { "url": "lang", "valueCode": "nl"
            }, {
              "url": "content",
              "valueString": "Coronavirusziekte 2019"
            } ]
          }, {
            "url":
            ↪ "http://hl7.org/fhir/StructureDefinition/translation",
            "extension": [ { "url": "lang", "valueCode": "pt-BR" }, {
              "url": "content",
              "valueString": "Doença pelo coronavírus de 2019"
            } ]
          }, {
            "url":
            ↪ "http://hl7.org/fhir/StructureDefinition/translation",
            "extension": [ { "url": "lang", "valueCode": "pt" }, {
              "url": "content",
              "valueString": "Doença por coronavírus 2019"
            } ]
          }, {
            "url":
            ↪ "http://hl7.org/fhir/StructureDefinition/translation",
            "extension": [ { "url": "lang", "valueCode": "ru" }, {
              "url": "content",
              "valueString": "Коронавирусная инфекция 2019 года"
            } ]
          } ]
        }
      } ]
    }
  }
}
```

F.3.2 XML

Multiple Tags without Labels

```
<Bundle xmlns="http://hl7.org/fhir">
  <id value="d62042b2c2b6cabc"/>
  <type value="transaction"/>
```

```
<entry>
  <fullUrl value="0d15be86abd7eea5"/>
  <resource>
    <CarePlan xmlns="http://hl7.org/fhir">
      <id value="0d15be86abd7eea5"/>
      <identifier>
        <system
        ↪ value="https://obtima.org/fhir/NamingSystem/acronym"/>
        <value value="NUM CODEX"/>
      </identifier>
      <identifier>
        <system value="https://obtima.org/fhir/NamingSystem/name"/>
        <value value="NUM CODEX"/>
      </identifier>
      ...
      <addresses><reference
      ↪ value="Condition/f3ebecd70f219cd9"/></addresses>
      <addresses><reference
      ↪ value="Condition/3b7d56d64176a370"/></addresses>
      <addresses><reference
      ↪ value="Condition/af52d0d05763d100"/></addresses>
      <addresses><reference
      ↪ value="Condition/00f3a1c86a3dc1a9"/></addresses>
      <addresses><reference
      ↪ value="Condition/2077111e09bb927a"/></addresses>
      ...
    </CarePlan>
  </resource>
</entry>
...
<entry>
  <fullUrl value="f3ebecd70f219cd9"/>
  <resource>
    <Condition xmlns="http://hl7.org/fhir">
      <id value="f3ebecd70f219cd9"/>
      <code>
        <coding>
          <system
          ↪ value="http://terminology.hl7.org/CodeSystem/mdr"/>
          <code value="10084382"/>
        </coding>
      </code>
    </Condition>
```

```
    </resource>
  </entry>
  <entry>
    <fullUrl value="3b7d56d64176a370"/>
    <resource>
      <Condition xmlns="http://hl7.org/fhir">
        <id value="3b7d56d64176a370"/>
        <code>
          <coding>
            <system
            ↪ value="http://fhir.de/CodeSystem/bfarm/icd-10-gm"/>
            <code value="U07.1"/>
          </coding>
        </code>
      </Condition>
    </resource>
  </entry>
  <entry>
    <fullUrl value="af52d0d05763d100"/>
    <resource>
      <Condition xmlns="http://hl7.org/fhir">
        <id value="af52d0d05763d100"/>
        <code>
          <coding>
            <system value="http://loinc.org"/>
            <code value="95412-3"/>
          </coding>
        </code>
      </Condition>
    </resource>
  </entry>
  <entry>
    <fullUrl value="00f3a1c86a3dc1a9"/>
    <resource>
      <Condition xmlns="http://hl7.org/fhir">
        <id value="00f3a1c86a3dc1a9"/>
        <code>
          <coding>
            <system value="http://snomed.info/id"/>
            <code value="840539006"/>
          </coding>
        </code>
      </Condition>
```

```
    </resource>
  </entry>
  <entry>
    <fullUrl value="2077111e09bb927a"/>
    <resource>
      <Condition xmlns="http://hl7.org/fhir">
        <id value="2077111e09bb927a"/>
        <code>
          <coding>
            <system value="http://ncithesaurus-stage.nci.nih.gov"/>
            <code value="C171133"/>
          </coding>
        </code>
      </Condition>
    </resource>
  </entry>
  ...
</Bundle>
```

Single Tag with Multilingual Labels

```
<entry>
  <fullUrl value="f3ebecd70f219cd9"/>
  <resource>
    <Condition xmlns="http://hl7.org/fhir">
      <id value="f3ebecd70f219cd9"/>
      <code>
        <coding>
          <system value="http://terminology.hl7.org/CodeSystem/mdr"/>
          <code value="10084382"/>
          <display value="Coronavirus-Krankheit 2019">
            <extension
            ↪ url="http://hl7.org/fhir/StructureDefinition/translation">
              <extension url="lang"><valueCode value="cn"/>/extension>
              <extension url="content"><valueString value="2019
              ↪ 冠状病毒疾病"/></extension>
            </extension>
            <extension
            ↪ url="http://hl7.org/fhir/StructureDefinition/translation">
              <extension url="lang"><valueCode
              ↪ value="cz"/></extension>
              <extension url="content"><valueString value="Onemocnění
              ↪ způsobené koronavirem 2019"/></extension>
```

```
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="de"/></extension>
  <extension url="content"><valueString
  ↪ value="Coronavirus-Krankheit 2019"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="en"/></extension>
  <extension url="content"><valueString value="Coronavirus
  ↪ disease 2019"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="es"/></extension>
  <extension url="content"><valueString value="Enfermedad
  ↪ por coronavirus 2019"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="fr"/></extension>
  <extension url="content"><valueString value="Maladie à
  ↪ coronavirus 2019"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="hu"/></extension>
  <extension url="content"><valueString value="Koronavírus
  ↪ okozta megbetegedés 2019"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="it"/></extension>
  <extension url="content"><valueString value="Malattia da
  ↪ Coronavirus 2019"/></extension>
</extension>
```

```
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="jp"/></extension>
  <extension url="content"><valueString value="2019
  ↪ コロナウイルス病気"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="ko"/></extension>
  <extension url="content"><valueString value="2019
  ↪ 코로나바이러스 감염증"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="nl"/></extension>
  <extension url="content"><valueString
  ↪ value="Coronavirusziekte 2019"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="pt-BR"/></extension>
  <extension url="content"><valueString value="Doença pelo
  ↪ coronavírus de 2019"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="pt"/></extension>
  <extension url="content"><valueString value="Doença por
  ↪ coronavírus 2019"/></extension>
</extension>
<extension
↪ url="http://hl7.org/fhir/StructureDefinition/translation">
  <extension url="lang"><valueCode
  ↪ value="ru"/></extension>
  <extension url="content"><valueString
  ↪ value="Коронавирусная инфекция 2019
  ↪ года"/></extension>
</extension>
```

```
          </display>
        </coding>
      </code>
    </Condition>
  </resource>
</entry>
```

F.3.3 RDF

Multiple Tags without Labels

```
@prefix fhir: <http://hl7.org/fhir/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

fhir:d62042b2c2b6cabc a fhir:Bundle ;
  fhir:nodeRole fhir:treeRoot ;
  fhir:Resource.id _:node1 ;
  fhir:Bundle.type _:node2 ;
  ...
  fhir:Bundle.entry _:node3, _:node4, _:node5, _:node6, _:node7,
  ↪ _:node8, ... .
_:node1 fhir:value "d62042b2c2b6cabc" .
_:node2 fhir:value "transaction" .
...
_:node3 fhir:Bundle.entry.fullUrl _:node9 ;
  ...
  fhir:Bundle.entry.resource
  ↪ <http://hl7.org/fhir/CarePlan/0d15be86abd7eea5> .
_:node9 fhir:value "0d15be86abd7eea5"^^xsd:anyURI .
<http://hl7.org/fhir/CarePlan/0d15be86abd7eea5> a fhir:CarePlan ;
  fhir:Resource.id _:node10 ;
  ...
  fhir:CarePlan.identifier _:node11, _:node12 ;
  fhir:CarePlan.addresses _:node13, _:node14, _:node15, _:node16,
  ↪ _:node17.
_:node11 fhir:Identifier.system _:node18 ;
  fhir:Identifier.value _:node19.
_:node18 fhir:value
↪ "https://obtima.org/fhir/NamingSystem/name"^^xsd:anyURI .
_:node19 fhir:value "NUM CODEX" .
```

```
_:node12 fhir:Identifier.system _:node20 ;
  fhir:Identifier.value _:node21 .
_:node20 fhir:value
↪ "https://obtima.org/fhir/NamingSystem/acronym"^^xsd:anyURI .
_:node21 fhir:value "NUM CODEX" .
...
_:node13 fhir:Reference.reference _:node22 .
_:node22 fhir:value "Condition/f3ebecd70f219cd9" .
_:node4 fhir:Bundle.entry.fullUrl _:node23 ;
  ...
  fhir:Bundle.entry.resource
  ↪ <http://hl7.org/fhir/Condition/f3ebecd70f219cd9> .
_:node23 fhir:value "f3ebecd70f219cd9"^^xsd:anyURI .
<http://hl7.org/fhir/Condition/f3ebecd70f219cd9> a fhir:Condition ;
  fhir:Resource.id _:node24 ;
  fhir:Condition.code _:node25 .
_:node24 fhir:value "f3ebecd70f219cd9" .
_:node25 fhir:CodeableConcept.coding _:node26 .
_:node26 fhir:Coding.code _:node27 ;
  fhir:Coding.system _:node28 .
_:node28 fhir:value
↪ "http://terminology.hl7.org/CodeSystem/mdr"^^xsd:anyURI .
_:node27 fhir:value "10084382" .
...
_:node14 fhir:Reference.reference _:node29 .
_:node29 fhir:value "Condition/3b7d56d64176a370" .
_:node5 fhir:Bundle.entry.fullUrl _:node30 ;
  ...
  fhir:Bundle.entry.resource
  ↪ <http://hl7.org/fhir/Condition/3b7d56d64176a370> .
_:node30 fhir:value "3b7d56d64176a370"^^xsd:anyURI .
<http://hl7.org/fhir/Condition/3b7d56d64176a370> a fhir:Condition ;
  fhir:Resource.id _:node31 ;
  fhir:Condition.code _:node32 .
_:node31 fhir:value "3b7d56d64176a370" .
_:node32 fhir:CodeableConcept.coding _:node33 .
_:node33 fhir:Coding.code _:node34 ;
  fhir:Coding.system _:node35 .
_:node35 fhir:value
↪ "http://fhir.de/CodeSystem/bfarm/icd-10-gm"^^xsd:anyURI .
_:node34 fhir:value "U07.1" .
...
_:node15 fhir:Reference.reference _:node36 .
```

```
_:node36 fhir:value "Condition/af52d0d05763d100" .
_:node6 fhir:Bundle.entry.fullUrl _:node37 ;
  ...
  fhir:Bundle.entry.resource
  ↪ <http://hl7.org/fhir/Condition/af52d0d05763d100> .
_:node37 fhir:value "af52d0d05763d100"^^xsd:anyURI .
<http://hl7.org/fhir/Condition/af52d0d05763d100> a fhir:Condition ;
  fhir:Resource.id _:node38 ;
  fhir:Condition.code _:node39 .
_:node38 fhir:value "af52d0d05763d100" .
_:node39 fhir:CodeableConcept.coding _:node40 .
_:node40 fhir:Coding.code _:node41 ;
  fhir:Coding.system _:node42.
_:node42 fhir:value "http://loinc.org"^^xsd:anyURI .
_:node41 fhir:value "95412-3" .
...
_:node16 fhir:Reference.reference _:node43 .
_:node43 fhir:value "Condition/00f3a1c86a3dc1a9" .
_:node7 fhir:Bundle.entry.fullUrl _:node44 ;
  ...
  fhir:Bundle.entry.resource
  ↪ <http://hl7.org/fhir/Condition/00f3a1c86a3dc1a9> .
_:node44 fhir:value "00f3a1c86a3dc1a9"^^xsd:anyURI .
<http://hl7.org/fhir/Condition/00f3a1c86a3dc1a9> a fhir:Condition ;
  fhir:Resource.id _:node45 ;
  fhir:Condition.code _:node46 .
_:node45 fhir:value "00f3a1c86a3dc1a9" .
_:node46 fhir:CodeableConcept.coding _:node47 .
_:node47 fhir:Coding.code _:node48 ;
  fhir:Coding.system _:node49 .
_:node49 fhir:value "http://snomed.info/id"^^xsd:anyURI .
_:node48 fhir:value "840539006" .
...
_:node17 fhir:Reference.reference _:node50 .
_:node50 fhir:value "Condition/2077111e09bb927a" .
_:node8 fhir:Bundle.entry.fullUrl _:node51 ;
  ...
  fhir:Bundle.entry.resource
  ↪ <http://hl7.org/fhir/Condition/2077111e09bb927a> .
_:node51 fhir:value "2077111e09bb927a"^^xsd:anyURI .
<http://hl7.org/fhir/Condition/2077111e09bb927a> a fhir:Condition ;
  fhir:Resource.id _:node52 ;
  fhir:Condition.code _:node53 .
```

```
_:node52 fhir:value "2077111e09bb927a" .
_:node53 fhir:CodeableConcept.coding _:node54 .
_:node54 fhir:Coding.code _:node55 ;
  fhir:Coding.system _:node56 .
_:node56 fhir:value
↪ "http://ncithesaurus-stage.nci.nih.gov"^^xsd:anyURI .
_:node55 fhir:value "C171133" .
```

Single Tag with Multilingual Labels

```
_:node1 fhir:Reference.reference _:node2 .
_:node2 fhir:value "Condition/f3ebecd70f219cd9" .
_:node3 fhir:Bundle.entry.fullUrl _:node4 ;
  ...
  fhir:Bundle.entry.resource
  ↪ <http://hl7.org/fhir/Condition/f3ebecd70f219cd9> .
_:node4 fhir:value "f3ebecd70f219cd9"^^xsd:anyURI .
<http://hl7.org/fhir/Condition/f3ebecd70f219cd9> a fhir:Condition ;
  fhir:Resource.id _:node5 ;
  fhir:Condition.code _:node6 .
_:node5 fhir:value "f3ebecd70f219cd9" .
_:node6 fhir:CodeableConcept.coding _:node7 .
_:node7 fhir:Coding.code _:node8 ;
  fhir:Coding.system _:node9 ;
  fhir:Coding.display _:node10 ;
_:node9 fhir:value
↪ "http://terminology.hl7.org/CodeSystem/mdr"^^xsd:anyURI .
_:node8 fhir:value "10084382" .
_:node10 fhir:Element.extension _:node11, _:node12, _:node13,
↪ _:node14, _:node15, _:node16, _:node17, _:node17, _:node18,
↪ _:node19, _:node20, _:node21, _:node22, _:node23 ;
  fhir:value "Coronavirus-Krankheit 2019" .
_:node11 fhir:Element.extension _:node24, _:node25 ;
  fhir:Extension.url _:node26 .
_:node24 fhir:Extension.url _:node27 ;
  fhir:Extension.valueString _:node28 .
_:node27 fhir:value "content"^^xsd:anyURI .
_:node28 fhir:value "Doença por coronavírus 2019" .
_:node25 fhir:Extension.url _:node29 ;
  fhir:Extension.valueCode _:node30 .
_:node29 fhir:value "lang"^^xsd:anyURI .
_:node30 fhir:value "pt" .
```

```
_:node26 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node12 fhir:Element.extension _:node31, _:node32 ;
  fhir:Extension.url _:node33 .
_:node31 fhir:Extension.url _:node34 ;
  fhir:Extension.valueString _:node35 .
_:node34 fhir:value "content"^^xsd:anyURI .
_:node35 fhir:value "Коронавирусная инфекция 2019 года" .
_:node32 fhir:Extension.url _:node36 ;
  fhir:Extension.valueCode _:node37 .
_:node36 fhir:value "lang"^^xsd:anyURI .
_:node37 fhir:value "ru" .
_:node33 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node13 fhir:Element.extension _:node38, _:node39 ;
  fhir:Extension.url _:node40 .
_:node38 fhir:Extension.url _:node41 ;
  fhir:Extension.valueString _:node42 .
_:node41 fhir:value "content"^^xsd:anyURI .
_:node42 fhir:value "Coronavirus-Krankheit 2019" .
_:node39 fhir:Extension.url _:node43 ;
  fhir:Extension.valueCode _:node44 .
_:node43 fhir:value "lang"^^xsd:anyURI .
_:node44 fhir:value "de" .
_:node40 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node14 fhir:Element.extension _:node45, _:node46 ;
  fhir:Extension.url _:node47 .
_:node45 fhir:Extension.url _:node48 ;
  fhir:Extension.valueString _:node49 .
_:node48 fhir:value "content"^^xsd:anyURI .
_:node49 fhir:value "Malattia da Coronavirus 2019" .
_:node46 fhir:Extension.url _:node50 ;
  fhir:Extension.valueCode _:node51 .
_:node50 fhir:value "lang"^^xsd:anyURI .
_:node51 fhir:value "it" .
_:node47 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node15 fhir:Element.extension _:node52, _:node53 ;
  fhir:Extension.url _:node54 .
_:node52 fhir:Extension.url _:node55 ;
  fhir:Extension.valueString _:node56 .
_:node55 fhir:value "content"^^xsd:anyURI .
```

```
_:node56 fhir:value "Maladie à coronavirus 2019" .
_:node53 fhir:Extension.url _:node57 ;
  fhir:Extension.valueCode _:node58 .
_:node57 fhir:value "lang"^^xsd:anyURI .
_:node58 fhir:value "fr" .
_:node54 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node16 fhir:Element.extension _:node59, _:node60 ;
  fhir:Extension.url _:node61 .
_:node59 fhir:Extension.url _:node62 ;
  fhir:Extension.valueString _:node63 .
_:node62 fhir:value "content"^^xsd:anyURI .
_:node63 fhir:value "Koronavírus okozta megbetegedés 2019" .
_:node60 fhir:Extension.url _:node64 ;
  fhir:Extension.valueCode _:node65 .
_:node64 fhir:value "lang"^^xsd:anyURI .
_:node65 fhir:value "hu" .
_:node61 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node17 fhir:Element.extension _:node66, _:node67 ;
  fhir:Extension.url _:node68 .
_:node66 fhir:Extension.url _:node69 ;
  fhir:Extension.valueString _:node70 .
_:node69 fhir:value "content"^^xsd:anyURI .
_:node70 fhir:value "2019 冠状病毒疾病" .
_:node67 fhir:Extension.url _:node71 ;
  fhir:Extension.valueCode _:node72 .
_:node71 fhir:value "lang"^^xsd:anyURI .
_:node72 fhir:value "cn" .
_:node68 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node17 fhir:Element.extension _:node73, _:node74 ;
  fhir:Extension.url _:node75 .
_:node73 fhir:Extension.url _:node76 ;
  fhir:Extension.valueString _:node77 .
_:node76 fhir:value "content"^^xsd:anyURI .
_:node77 fhir:value "Coronavirusziekte 2019" .
_:node74 fhir:Extension.url _:node78 ;
  fhir:Extension.valueCode _:node79 .
_:node78 fhir:value "lang"^^xsd:anyURI .
_:node79 fhir:value "nl" .
_:node75 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
```

```
_:node18 fhir:Element.extension _:node80, _:node81 ;
  fhir:Extension.url _:node82 .
_:node80 fhir:Extension.url _:node83 ;
  fhir:Extension.valueString _:node84 .
_:node83 fhir:value "content"^^xsd:anyURI .
_:node84 fhir:value "2019 코로나바이러스 감염증" .
_:node81 fhir:Extension.url _:node85 ;
  fhir:Extension.valueCode _:node86 .
_:node85 fhir:value "lang"^^xsd:anyURI .
_:node86 fhir:value "ko" .
_:node82 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node19 fhir:Element.extension _:node87, _:node88 ;
  fhir:Extension.url _:node89 .
_:node87 fhir:Extension.url _:node90 ;
  fhir:Extension.valueString _:node91 .
_:node90 fhir:value "content"^^xsd:anyURI .
_:node91 fhir:value "Enfermedad por coronavirus 2019" .
_:node88 fhir:Extension.url _:node92 ;
  fhir:Extension.valueCode _:node93 .
_:node92 fhir:value "lang"^^xsd:anyURI .
_:node93 fhir:value "es" .
_:node89 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node20 fhir:Element.extension _:node94, _:node95 ;
  fhir:Extension.url _:node96 .
_:node94 fhir:Extension.url _:node97 ;
  fhir:Extension.valueString _:node98 .
_:node97 fhir:value "content"^^xsd:anyURI .
_:node98 fhir:value "Onemocnění způsobené koronavirem 2019" .
_:node95 fhir:Extension.url _:node99 ;
  fhir:Extension.valueCode _:node100 .
_:node99 fhir:value "lang"^^xsd:anyURI .
_:node100 fhir:value "cz" .
_:node96 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node21 fhir:Element.extension _:node101, _:node102 ;
  fhir:Extension.url _:node103 .
_:node101 fhir:Extension.url _:node104 ;
  fhir:Extension.valueString _:node105 .
_:node104 fhir:value "content"^^xsd:anyURI .
_:node105 fhir:value "2019 コロナウイルス病気" .
_:node102 fhir:Extension.url _:node106 ;
```

```
  fhir:Extension.valueCode _:node107 .
_:node106 fhir:value "lang"^^xsd:anyURI .
_:node107 fhir:value "jp" .
_:node103 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node22 fhir:Element.extension _:node108, _:node109 ;
  fhir:Extension.url _:node110 .
_:node108 fhir:Extension.url _:node111 ;
  fhir:Extension.valueString _:node112 .
_:node111 fhir:value "content"^^xsd:anyURI .
_:node112 fhir:value "Coronavirus disease 2019" .
_:node109 fhir:Extension.url _:node113 ;
  fhir:Extension.valueCode _:node114 .
_:node113 fhir:value "lang"^^xsd:anyURI .
_:node114 fhir:value "en" .
_:node110 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
_:node23 fhir:Element.extension _:node115, _:node116 ;
  fhir:Extension.url _:node117 .
_:node115 fhir:Extension.url _:node118 ;
  fhir:Extension.valueString _:node119 .
_:node118 fhir:value "content"^^xsd:anyURI .
_:node119 fhir:value "Doença pelo coronavírus de 2019" .
_:node116 fhir:Extension.url _:node120 ;
  fhir:Extension.valueCode _:node121 .
_:node120 fhir:value "lang"^^xsd:anyURI .
_:node121 fhir:value "pt-BR" .
_:node117 fhir:value
↪ "http://hl7.org/fhir/StructureDefinition/translation"^^xsd:anyURI .
```

Appendix G

Acronyms

ACGT Advancing Clinico Genomic Trials on Cancer
API Application Programming Interface
ATC Anatomical Therapeutic Chemical Classification System

CDISC Clinical Data Interchange Standards Consortium
CIA COVID-19 Interoperability Alliance
CODEX COVID-19 Data Exchange Platform
COVID-19 Coronavirus Disease 2019
CRF Case Report Form
CST Clinical Standards Toolkit
CSV Comma Separated Value
CTCAE Common Terminology Criteria for Adverse Events
CTMS Clinical Trial Management System

DAkkS Deutsche Akkreditierungsstelle (German Accreditation Body)
DAO Data Access Object
DiGA Digitale Gesundheitsanwendung (Digital Health Application)
DIN Deutsches Institut für Normung (German Institute for Standardization)
DSV Comma Separated Value
DVG Digitale-Versorgung-Gesetz (Digital Healthcare Act)

EDC Electronic Data Capture
EHR Electronic Health Record
ERM Entity Relationship Model
EU European Union

FHIR Fast Healthcare Interoperability Resources

GCP Good Clinical Practice
GECCO German Corona Consensus Dataset
GO Gene Ontology

GUI Graphical User Interface

HCLS Healthcare and Life Sciences
HDOT Health Data Ontology Trunk
HIMSS Healthcare Information and Management Systems Society
HIS Hospital Information System
HIV Human Immunodeficiency Virus
HPO Human Phenotype Ontology
HTML Hypertext Markup Language
HTTP Hypertext Transfer Protocol

ICD International Statistical Classification of Diseases and Related Health Problems
ICD-10-GM ICD, 10th Revision, German Modification
IDE Integrated Development Environment
IG Interest Group
IRI Internationalized Resource Identifier
ISO International Organization for Standardization

JAXB Jakarta XML Binding, formerly Java Architecture for XML Binding
JPA Jakarta (previously, Java) Persistence API
JSF JavaServer Faces
JSON JavaScript Object Notation
JSON-LD JSON for Linked Data

LOINC Logical Observation Identifiers Names and Codes
LOV Linked Open Vocabularies

MDR Metadata Registry
MedDRA Medical Dictionary for Regulatory Activities
MIO Medizinisches Informationsobjekt (Medical Information Object)
MO Master Ontology
MOnSTER Multi-Ontology Semantic Trial Enrichment Resource
MVC Model-View-Controller

NCIt National Cancer Institute Thesaurus

NUM Netzwerk Universitätsmedizin (Network University Medicine)

OBO Open Biological and Biomedical Ontologies
ObTiMA Ontology-Based Trial Management Application
ODM Operational Data Model
OLS Ontology Lookup Service
OPS Operationen- und Prozedurenschlüssel (Operation and Procedure Classification System)
ORDO Orphanet Rare Disease Ontology
ORM Object Relational Mapping
OWL Web Ontology Language

p-medicine From Data Sharing and Integration via VPH Models to Personalized Medicine
PHR Personal Health Record
PI Principal Investigator

RDBMS Relational Database Management System
RDF Resource Description Framework
RDFS RDF Schema
REDCap Research Electronic Data Capture
REST Representational State Transfer

SE Study Event
SKOS Simple Knowledge Organization System
SNOMED CT Systematized Nomenclature of Medicine Clinical Terms
SQL Structured Query Language

TURTLE Terse RDF Triple Language

UI User Interface
UML Unified Modeling Language
UMLS Unified Medical Language System
URI Uniform Resource Identifier

W3C World Wide Web Consortium

XML Extensible Markup Language
XSD XML Schema Definition

Bibliography

Ambler, S. (2013). *Mapping Objects to Relational Databases*: *O/R Mapping In Detail*. http://www.agiledata.org/essays/mappingObjects.html (Last Access: 03/31/2022)

Anderson, J., and Wischgoll, T. (2020). *Visualization of Search Results of Large Document Sets*. Electronic Imaging, 2020 (1), 388-1–388-7.

Andrews, J., Richesson, R., and Krischer, J. (2007). *Variation of SNOMED CT Coding of Clinical Research Concepts Among Coding Experts*. Journal of the American Medical Informatics Association, 14 (4), 497–506.

Apache Software Foundation (ASF). (2014). *Apache License, Version 2.0*. https://www.apache.org/licenses/LICENSE-2.0 (Last Access: 03/31/2022)

Apache Software Foundation (ASF). (2022a). *Lucene Core*. https://lucene.apache.org/core (Last Access: 03/31/2022)

Apache Software Foundation (ASF). (2022b). *Tomcat*. https://tomcat.apache.org (Last Access: 03/31/2022)

Apple Incorporated. (2021). *ResearchKit and CareKit*. https://www.researchandcare.org (Last Access: 03/31/2022)

Apple Incorporated. (2022). *HealthKit*. https://developer.apple.com/documentation/healthkit (Last Access: 03/31/2022)

Beckett, D. (2014). *RDF 1.1 N-Triples*: *A Line-Based Syntax for an RDF Graph* (*W3C Recommendation*). https://www.w3.org/TR/n-triples (Last Access: 03/31/2022)

Beckett, D., Berners-Lee, T., Prud'hommeaux, E., and Carothers, G. (2014). *RDF 1.1 TURTLE*: *Terse RDF Triple Language* (*W3C Recommendation*). https://www.w3.org/TR/turtle (Last Access: 03/31/2022)

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). *The Semantic Web*: *A New Form of Web Content That Is Meaningful to Computers Will Unleash a Revolution of New Possibilities*. Scientific American, 284 (5), 34–43.

Berrueta, D., Phipps, J., Miles, A., Baker, T., and Swick, R. (Eds.). (2008). *Best Practice Recipes for Publishing RDF Vocabularies* (*W3C Working Group Note*). https://www.w3.org/TR/swbp-vocabpub (Last Access: 03/31/2022)

Bloch, J. (2018). *Effective Java* (*Third Edition*).

Bodenreider, O. (2004). *The Unified Medical Language System* (*UMLS*): *Integrating Biomedical Terminology*. Nucleic Acids Research, 32 (Database Issue), D267–D270.

Bray, T., Hollander, D., Layman, A., Tobin, R., and Thompson, H. (Eds.). (2009). *Namespaces in XML 1.0* (*Third Edition*) (*W3C Recommendation*). https://www.w3.org/TR/xml-names (Last Access: 03/31/2022)

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (Eds.). (2008). *Extensible Markup Language* (*XML*) *1.0* (*Fifth Edition*) (*W3C Recommendation*). https://www.w3.org/TR/xml (Last Access: 03/31/2022)

Brickley, D., Guha, R., and McBride, B. (Eds.). (2014). *RDF Schema 1.1* (*W3C Recommendation*). https://www.w3.org/TR/rdf-schema (Last Access: 03/31/2022)

Brix, T., Bruland, P., Sarfraz, S., Ernsting, J., Neuhaus, P., Storck, M., Doods, J., Ständer, S., and Dugas, M. (2018). *ODM Data Analysis - A Tool for the Automatic Validation, Monitoring and Generation of Generic Descriptive Statistics of Patient Data*. PLOS ONE, 13 (6), e0199242.

Brochhausen, M., Spear, A., Cocos, C., Weiler, G., Martín, L., Anguita, A., Stenzhorn, H., Daskalaki, E., Schera, F., Schwarz, U., Sfakianakis, S., Kiefer, S., Dörr, M., Graf,

N., and Tsiknakis, M. (2011). *The ACGT Master Ontology and Its Applications - Towards an Ontology-Driven Cancer Research and Management System*. Journal of Biomedical Informatics, 44, 8-25.

Bundesinstitut für Arzneimittel und Medizinprodukte (BfArM). (2022a). *ICD-10-GM: Internationale statistische Klassifikation der Krankheiten und verwandter Gesundheitsprobleme, German Modification*. https://www.bfarm.de/DE/Kodiersysteme/Klassifikationen/ICD/ICD-10-GM/_node.html (Last Access: 03/31/2022)

Bundesinstitut für Arzneimittel und Medizinprodukte (BfArM). (2022b). *OPS: Operationen- und Prozedurenschlüssel*. https://www.bfarm.de/DE/Kodiersysteme/Klassifikationen/OPS-ICHI/OPS/_node.html (Last Access: 03/31/2022)

Canonical Limited. (2022). *Ubuntu*. https://www.ubuntu.com (Last Access: 03/31/2022)

CDISC Consortium. (2013). *ODM-XML 1.3.2*. https://www.cdisc.org/standards/foundational/odm-xml/odm-xml-v1-3-2 (Last Access: 03/31/2022)

Christ-Neumann, M.-L., Escrich, A., Anguita, A., Stenzhorn, H., Taylor, M., Ramay, H., Rüping, S., Krauth, C., Kuchinke, W., Graf, N., and Rossi, S. (2014). *Usability on the p-medicine Infrastructure: An Extended Usability Concept*. ecancermedicalscience, 8, 399.

Connolly, D. (Ed.). (2003). *XML Development History*. https://www.w3.org/XML/hist2002 (Last Access: 03/31/2022)

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2009). *Introduction to Algorithms (Third Edition)*

COVID-19 Interoperability Alliance Members. (2021). *COVID-19 Interoperability Alliance*. https://covid19ia.org (Last Access: 03/31/2022)

Cyganiak, R., Wood, D., Lanthaler, M., Klyne, G., Carroll, J., and McBride, B. (Eds.). (2014). *RDF 1.1 Concepts and Abstract Syntax* (*W3C Recommendation*). https://www.w3.org/TR/rdf11-concepts (Last Access: 03/31/2022)

Davis, N. (2020). *Medical Abbreviations That Have Contradictory or Ambiguous Meanings*. https://www.ismp.org/resources/medical-abbreviations-have-contradictory-or-ambiguous-meanings (Last Access: 03/31/2022)

Deutsche Akkreditierungsstelle (DAkkS). (2010). *Leitfaden Usability*.

DIN-Normenausschuss Informationstechnik und Anwendungen (NIA). (2020). *DIN 5008 Schreib- und Gestaltungsregeln für die Text- und Informationsverarbeitung*.

Doods, J., Neuhaus, P., and Dugas, M. (2016). *Converting ODM Metadata to FHIR Questionnaire Resources*. Studies in Health Technology and Informatics, 228, 456–460.

Dron, L., Dillman, A., Zoratti, M., Haggstrom, J., Mills, E., and Park, J. (2021). *Clinical Trial Data Sharing for COVID-19-Related Research*. Journal of Medical Internet Research, 23 (3), e26718.

Dugas, M. (2016). *Design of Case Report Forms Based on a Public Metadata Registry*: *Re-Use of Data Elements to Improve Compatibility of Data*. Trials, 17 (1), 566.

Dugas, M., Meidt, A., Neuhaus, P., Storck, M., and Varghese, J. (2016). *ODMedit*: *Uniform Semantic Annotation for Data Integration in Medicine Based on a Public Metadata Repository*. BMC Medical Research Methodology, 16, 65.

Eclipse Foundation. (2007). *Eclipse Distribution License* (*EDL*) *v 1.0*. https://www.eclipse.org/org/documents/edl-v10.php (Last Access: 03/31/2022)

Eclipse Foundation. (2017). *Eclipse Public License* (*EPL*) - *v 2.0*. https://www.eclipse.org/legal/epl-2.0 (Last Access: 03/31/2022)

Eclipse Foundation. (2022a). *Eclipse Implementation of JAXB*. https://projects.eclipse.org/projects/ee4j.jaxb-impl (Last Access: 03/31/2022)

Eclipse Foundation. (2022b). *Jakarta EE Web Profile*. https://jakarta.ee/specifications/webprofile (Last Access: 03/31/2022)

Eclipse Foundation. (2022c). *Jakarta Persistence*. https://jakarta.ee/specifications/persistence (Last Access: 03/31/2022)

Eclipse Foundation. (2022d). *Jakarta Server Faces*. https://jakarta.ee/specifications/faces (Last Access: 03/31/2022)

Eclipse Foundation. (2022e). *Jakarta XML Binding*. https://jakarta.ee/specifications/xml-binding (Last Access: 03/31/2022)

Eclipse Foundation. (2022f). *Jetty*. https://www.eclipse.org/jetty (Last Access: 03/31/2022)

Eclipse Foundation. (2022g). *Mojarra*: *Eclipse EE4J Implementation of Jakarta Faces*. https://eclipse-ee4j.github.io/mojarra (Last Access: 03/31/2022)

Eclipse Foundation. (2022h). *RDF4J*. https://rdf4j.org (Last Access: 03/31/2022)

Egaña Aranguren, M., Fernández-Breis, J., and Dumontier, M. (2014). *Special Issue on Linked Data for Health Care and the Life Sciences*. Semantic Web, 5 (2), 99–100.

Elmasri, R., and Navathe, S. (2015). *Fundamentals of Database Systems* (*Seventh Edition*).

Esteban-Gil, A., and Fernández-Breis, J. (2016). *Case Report Form Based on Semantic Web Technologies*. Proceedings of the International Conference on Semantic Web Applications and Tools for Life Sciences (SWAT4LS).

European Molecular Biology Laboratory - European Bioinformatics Institute (EMBL-EBI). (2018). *Chemical Entities of Biological Interest* (*ChEBI*). https://www.ebi.ac.uk/chebi (Last Access: 03/31/2022)

European Molecular Biology Laboratory - European Bioinformatics Institute (EMBL-EBI). (2021). *Ontology Lookup Service* (*OLS*). https://www.ebi.ac.uk/ols (Last Access: 03/31/2022)

European Parliament and European Council (EP-EC). (2017a). *Regulation* (*EU*) *2017/745 on Medical Devices*. https://eur-lex.europa.eu/eli/reg/2017/745 (Last Access: 03/31/2022)

European Parliament and European Council (EP-EC). (2017b). *Regulation* (*EU*) *2017/746 on In Vitro Diagnostic Medical Devices*. https://eur-lex.europa.eu/eli/reg/2017/746 (Last Access: 03/31/2022)

FasterXML. (2022). *Jackson*. https://github.com/FasterXML/jackson (Last Access: 03/31/2022)

Fielding, R. (2000). *Representational State Transfer* (*REST*). Architectural Styles and the Design of Network-Based Software Architectures.

Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., and Stafford, R. (2011). *Patterns of Enterprise Application Architecture* (*Seventeenth Edition*).

Free Software Foundation (FSF). (2007). *GNU Lesser General Public License* (*LGPL*) - *Version 3*. https://www.gnu.org/licenses/lgpl-3.0.html (Last Access: 03/31/2022)

Friedman, L., Furberg, C., DeMets, D., Reboussin, D., and Granger, C. (2015). *Fundamentals of Clinical Trials*.

Gandon, F., Schreiber, G., and Beckett, D. (Eds.). (2014). *RDF 1.1 XML Syntax* (*W3C Recommendation*). https://www.w3.org/TR/rdf-syntax-grammar (Last Access: 03/31/2022)

Gao, S., Sperberg-McQueen, C. M., Thompson, H., Mendelsohn, N., Beech, D., and Maloney, M. (Eds.). (2012). *XML Schema Definition Language* (*XSD*) *1.1*: *Part 1*: *Structures* (*W3C Recommendation*). https://www.w3.org/TR/xmlschema11-1 (Last Access: 03/31/2022)

Gene Ontology Consortium (GOC). (2021). *The Gene Ontology Resource*: *Enriching a Gold Mine*. Nucleic Acids Research, 49 (D1), D325–D334.

Gerke, S., Stern, A., and Minssen, T. (2020). *Germany's Digital Health Reforms in the COVID-19 Era*: *Lessons and Opportunities for Other Countries*. NPJ Digital Medicine, 3 (1), 94.

Gonçalves, R., Hardi, J., Horridge, M., Tu, S., and Musen, M. (2021). *Protégé*: *A Free, Open-Source Ontology Editor and Framework for Building Intelligent Systems*. https://protege.stanford.edu (Last Access: 03/31/2022)

Google LLC. (2022). *Cloud Healthcare API Documentation*. https://cloud.google.com/healthcare-api/docs (Last Access: 03/31/2022)

Gossman, J., Mullins, C., Jones, G., Dolin, R., and Stafford, M. (Eds.). (2022). *Microsoft REST API Guidelines*. https://github.com/microsoft/api-guidelines (Last Access: 03/31/2022)

Greene, D., McClintock, D., and Durant, T. (2021). *Interoperability*: *COVID-19 as an Impetus for Change*. Clinical Chemistry, 67 (4), 592–595.

Gulden, C., Blasini, R., Nassirian, A., Stein, A., Altun, F. B., Kirchner, M., Prokosch, H.-U., and Boeker, M. (2021). *Prototypical Clinical Trial Registry Based on Fast Healthcare Interoperability Resources* (*FHIR*): *Design and Implementation Study*. JMIR Medical Informatics, 9 (1), e20470.

Haas, H., and Brown, A. (Eds.). (2004). *Web Services Glossary* (*W3C Working Group Note*). https://www.w3.org/TR/wsgloss (Last Access: 03/31/2022)

HAPI FHIR Team. (2022). *HAPI FHIR*. https://hapifhir.io (Last Access: 03/31/2022)

Harris, P., Taylor, R., Minor, B., Elliott, V., Fernandez, M., O'Neal, L., McLeod, L., Delacqua, G., Delacqua, F., Kirby, J., and Duda, S. (2019). *The REDCap Consortium*: *Building an International Community of Software Platform Partners*. Journal of Biomedical Informatics, 95, 103208.

Harris, P., Taylor, R., Thielke, R., Payne, J., Gonzalez, N., and Conde, J. (2009). *Research Electronic Data Capture* (*REDCap*) - *A Metadata-Driven Methodology and Workflow Process for Providing Translational Research Informatics Support*. Journal of Biomedical Informatics, 42 (2), 377–381.

Health Level 7 (HL7) International. (2019). *FHIR Release 4 Specification* (*R4*). https://hl7.org/fhir/R4 (Last Access: 03/31/2022)

Health Level 7 (HL7) International. (2020). *Hl7 Terminology* (*3.1.0*). https://terminology.hl7.org (Last Access: 03/31/2022)

Health Level 7 (HL7) International. (2022). *Using the FHIR Validator*. https://confluence.hl7.org/display/FHIR/Using+the+FHIR+Validator (Last Access: 03/31/2022)

Healthcare Information and Management Systems Society Incorporated (HIMSS). (2021). *Interoperability in Healthcare*. https://www.himss.org/resources/interoperability-healthcare (Last Access: 03/31/2022)

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P., and Rudolph, S. (Eds.). (2012). *OWL 2 Web Ontology Language Primer* (*Second Edition*) (*W3C Recommendation*). https://www.w3.org/TR/owl2-primer (Last Access: 03/31/2022)

Holland, C., and Shostak, J. (2016). *Implementing CDISC Using SAS*: *An End-to-End Guide* (*Second Edition*).

Horridge, M., and Bechhofer, S. (2011). *The OWL API*: *A Java API for OWL Ontologies*. Semantic Web, 2 (1), 11–21.

Horridge, M., Palmisano, I., Spero, S., and Ansell, P. (2022). *OWL API*. https://github.com/owlcs/owlapi (Last Access: 03/31/2022)

Hume, S., Aerts, J., Sarnikar, S., and Huser, V. (2016). *Current Applications and Future Directions for the CDISC Operational Data Model Standard*: *A Methodological Review*. Journal of Biomedical Informatics, 60, 352–362.

Huser, V., Sastry, C., Breymaier, M., Idriss, A., and Cimino, J. (2015). *Standardizing Data Exchange for Clinical Research Protocols and Case Report Forms*: *An Assessment of the Suitability of the Clinical Data Interchange Standards Consortium* (*CDISC*) *Operational Data Model* (*ODM*). Journal of Biomedical Informatics, 57, 88–99.

International Business Machines Corporation (IBM). (2021). *Interoperability in Healthcare*. https://www.ibm.com/topics/interoperability-in-healthcare (Last Access: 03/31/2022)

International Council for Harmonisation of Technical Requirements for Pharmaceuticals for Human Use (ICH-TRPHU). (2022). *MedDRA*. https://www.meddra.org (Last Access: 03/31/2022)

International Organization for Standardization (ISO). (2002). *ISO 639 Language Codes*. https://www.iso.org/iso-639-language-codes.html (Last Access: 03/31/2022)

International Organization for Standardization (ISO). (2019). *ISO 8601 Date and Time Format*. https://www.iso.org/iso-8601-date-and-time-format.html (Last Access: 03/31/2022)

Internet Engineering Task Force (IETF). (2015). *The Basic HTTP Authentication Scheme*: *Request for Comments 7617*. https://datatracker.ietf.org/doc/html/rfc7617 (Last Access: 03/31/2022)

Isaac, A., and Summers, E. (Eds.). (2009). *SKOS Simple Knowledge Organization System Primer* (*W3C Recommendation*). https://www.w3.org/TR/skos-primer (Last Access: 03/31/2022)

JetBrains s.r.o. (2022). *IntelliJ IDEA*. https://www.jetbrains.com/idea (Last Access: 03/31/2022)

Joho, H., and Jose, J. (2006). *A Comparative Study of the Effectiveness of Search Result Presentation on the Web*. Advances in Information Retrieval, 302–313.

Jupp, S., Burdett, T., Malone, J., Leroy, C., Pearce, M., McMurry, J., and Parkinson, H. (2015). *A New Ontology Lookup Service at EMBL-EBI*. Proc. of the International Conference on Semantic Web Applications and Tools for Life Sciences (SWAT4LS).

Kamdar, M., Fernández, J., Polleres, A., Tudorache, T., and Musen, M. (2019). *Enabling Web-Scale Data Integration in Biomedicine Through Linked Open Data*. NPJ Digital Medicine, 2, 90.

Kassenärztliche Bundesvereinigung (KBV). (2022). *Medizinische informationsobjekte (MIO)*. https://mio.kbv.de (Last Access: 03/31/2022)

Kendall, E., Novacek, V., Baker, T., and Miles, A. (Eds.). (2008). *Principles of Good Practice for Managing RDF Vocabularies and OWL Ontologies (W3C Editor's Draft)*. https://www.w3.org/2006/07/SWD/Vocab/principles (Last Access: 03/31/2022)

Knublauch, H., Oberle, D., Tetlow, P., Wallace, E., Pan, J., and Uschold, M. (Eds.). (2006). *A Semantic Web Primer for Object-Oriented Software Developers (W3C Working Group Note)*. https://www.w3.org/TR/sw-oosd-primer (Last Access: 03/31/2022)

Köhler, S., Haendel, M., and Robinson, P. (2021). *The Human Phenotype Ontology*. https://hpo.jax.org (Last Access: 03/31/2022)

Kuchinke, W., Aerts, J., Semler, S., and Ohmann, C. (2009). *CDISC Standard-Based Electronic Archiving of Clinical Trials*. Methods of Information in Medicine, 48 (5), 408–413.

Kuchinke, W., Ohmann, C., Stenzhorn, H., Anguista, A., Sfakianakis, S., Graf, N., and Demotes, J. (2016). *Ensuring Sustainability of Software Tools and Services by Cooperation With a Research Infrastructure*. Personalized Medicine, 13 (1), 43–55.

Kuchinke, W., Wiegelmann, S., Verplancke, P., and Ohmann, C. (2006). *Extended Cooperation in Clinical Studies Through Exchange of CDISC Metadata Between Different Study Software Solutions*. Methods of Information in Medicine, 45 (4), 441–446.

Lehne, M., Luijten, S., Vom Felde Genannt Imbusch, P., and Thun, S. (2019). *The Use of FHIR in Digital Health - A Review of the Scientific Literature*. Studies in Health Technology and Informatics, 267, 52–58.

Leroux, H., Denney, C., Hastak, S., and Glover, H. (2019). *A Framework for Representing Clinical Research in FHIR*. Proceedings of the International Workshop on Semantic Web Applications and Tools for Life Sciences (SWAT4LS), 26–35.

Leroux, H., Metke-Jimenez, A., and Lawley, M. (2017). *Towards Achieving Semantic Interoperability of Clinical Study Data With FHIR*. Journal of Biomedical Semantics, 8 (1), 41.

Lin, M.-C., Vreeman, D., and Huff, S. (2011). *Investigating the Semantic Interoperability of Laboratory Data Exchanged Using LOINC Codes in Three Large Institutions*. Proceedings of the Annual Symposium of the American Medical Informatics Association (AMIA), 805–814.

Löbe, M., Knuth, M., and Mücke, R. (2009). *TIM*: *A Semantic Web Application for the Specification of Metadata Items in Clinical Research*. Proceedings of the International Workshop on Semantic Web Applications and Tools for Life Sciences (SWAT4LS)

Mandel, J., Kreda, D., Mandl, K., Kohane, I., and Ramoni, R. (2016). *Smart on FHIR*: *A Standards-Based, Interoperable Apps Platform for Electronic Health Records*. Journal of the American Medical Informatics Association (JAMIA), 23 (5), 899–908.

Marés, J., Shamardin, L., Weiler, G., Anguita, A., Sfakianakis, S., Neri, E., Zasada, S., Graf, N., and Coveney, P. (2014). *p-medicine*: *A Medical Informatics Platform for Integrated Large Scale Heterogeneous Patient Data*. Proceedings of the Annual Symposium of the American Medical Informatics Association (AMIA), 872–881.

Martin, L., Anguita, A., Graf, N., Tsiknakis, M., Brochhausen, M., Rüping, S., Bucur, A., Sfakianakis, S., Sengstag, T., Buffa, F., and Stenzhorn, H. (2011). *ACGT*: *Advancing Clinico-Genomic Trials on Cancer - Four Years of Experience*. Studies in Health Technology and Informatics, 169, 734–738.

McDonald, C., Huff, S., Suico, J., Hill, G., Leavelle, D., Aller, R., Forrey, A., Mercer, K., DeMoor, G., Hook, J., Williams, W., Case, J., and Maloney, P. (2003). *LOINC, A Universal Standard for Identifying Laboratory Observations: A 5-Year Update*. Clinical Chemistry, 49 (4), 624–633.

Microsoft Corporation. (2021). *Azure Health Data Services Documentation*. https://docs.microsoft.com/en-us/azure/healthcare-apis (Last Access: 03/31/2022)

Millar, J. (2016). *The Need for a Global Language - SNOMED CT Introduction*. Studies in Health Technology and Informatics, 225, 683–685.

Miñarro-Giménez, J., Martínez-Costa, C., Karlsson, D., Schulz, S., and Rosenbeck-Gøeg, K. (2018). *Qualitative Analysis of Manual Annotations of Clinical Text With SNOMED CT*. PLOS ONE, 13 (12).

Motik, B., Parsia, B., Patel-Schneider, P., Bechhofer, S., Cuenca Grau, B., Fokoue, A., and Hoekstra, R. (Eds.). (2012). *OWL 2 Web Ontology Language XML Serialization* (*Second Edition*) (*W3C Recommendation*). https://www.w3.org/TR/owlxml-serialization (Last Access: 03/31/2022)

Musen, M., and Protégé Team (2015). *The Protégé Project: A Look Back and a Look Forward*. AI Matters, 1 (4), 4–12.

National Cancer Institute (NCI). (2017). *Common Terminology Criteria for Adverse Avents* (*CTCAE*) *v5.0*. https://ctep.cancer.gov/protocoldevelopment/electronic_applications/ctc.htm#ctc_50 (Last Access: 03/31/2022)

National Cancer Institute (NCI). (2022). *NCI Thesaurus*. https://ncithesaurus.nci.nih.gov (Last Access: 03/31/2022)

National Center for Biomedical Ontology (NCBO). (2021). *BioPortal*. https://bioportal.bioontology.org (Last Access: 03/31/2022)

Nationales Forschungsnetzwerk der Universitätsmedizin zu COVID-19 (NUM). (2020). *Nationales Forschungsnetzwerk der Universitätsmedizin zu COVID-19*

(*NUM*). https://www.netzwerk-universitaetsmedizin.de (Last Access: 03/31/2022)

Nield, T. (2017). *An Introduction to Regular Expressions*: *Decoding Simple Regex Features* https://www.oreilly.com/content/an-introduction-to-regular-expressions (Last Access: 03/31/2022)

Nourani, A., Ayatollahi, H., and Dodaran, M. S. (2019). *A Review of Clinical Data Management Systems Used in Clinical Trials*. Reviews on Recent Clinical Trials, 14 (1), 10–23.

Object Management Group (OMG). (2017). *OMG Unified Modeling Language* (*UML*) - *2.5.1*. https://www.omg.org/spec/UML/2.5.1 (Last Access: 03/31/2022)

OBO Technical WG. (2021). *The Open Biological and Biomedical Ontology* (*OBO*) *Foundry* https://obofoundry.org (Last Access: 03/31/2022)

ObTiMA Team. (2022). *ObTiMA - Ontology-Based Trial Management Application*. https://obtima.org (Last Access: 03/31/2022)

Odgen, C., and Richards, I. (1923). *The Meaning of Meaning*: *A Study of the Influence of Language Upon Thought*.

National Library of Medicine (NLM) (2022). *Unified Medical Language System* (*UMLS*). https://www.nlm.nih.gov/research/umls (Last Access: 03/31/2022)

Ohmann, C., Kuchinke, W., Canham, S., Lauritsen, J., Salas, N., Schade-Brittinger, C., Wittenberg, M., McPherson, G., McCourt, J., Gueyffier, F., Lorimer, A., and Torres, F. (2011). *Standard Requirements for GCP-Compliant Data Management in Multinational Clinical Trials*. Trials, 12, 85.

Ontology for Biomedical Investigations (OBI). (2021). *Ontology for Biomedical Investigations*. http://obi-ontology.org (Last Access: 03/31/2022)

Open Source Initiative (OSI). (1987). *The MIT License*. https://opensource.org/licenses/MIT (Last Access: 03/31/2022)

Open Source Initiative (OSI). (2007). *The Open Source Definition*. https://opensource.org/docs/osd (Last Access: 03/31/2022)

OpenAPI Initiative (OAI). (2021). *OpenAPI Specification*. https://github.com/OAI/OpenAPI-Specification (Last Access: 03/31/2022)

Oracle Corporation. (2022a). *Java*. https://www.java.com (Last Access: 03/31/2022)

Oracle Corporation. (2022b). *MySQL*. https://www.mysql.com (Last Access: 03/31/2022)

Orphanet. (2022). *Orphanet Rare Disease Ontology* (*ORDO*). http://www.orphadata.org/cgi-bin/index.php#ordomodal (Last Access: 03/31/2022)

Patrick, T., Richesson, R., Andrews, J., and Folk, L. (2008). *SNOMED CT Coding Variation and Grouping for* "*Other Findings*" *in a Longitudinal Study on Urea Cycle Disorders*. Proceedings of the Annual Symposium of the American Medical Informatics Association (AMIA), 11–15.

Peterson, D., Gao, S., Malhotra, A., Michael, S.-M. C., Thompson, H., and Biron, P. (Eds.). (2012). *XML Schema Definition Language* (*XSD*) *1.1*: *Part 2*: *Datatypes* (*W3C Recommendation*). https://www.w3.org/TR/xmlschema11-2 (Last Access: 03/31/2022)

PhUSE CS Semantic Technology Working Group (PhUSE-CS-STWG). (2015). *CDISC Standards in RDF Reference Guide* (*Technical Report*).

Pisanelli, D., Gangemi, A., Battaglia, M., and Catenacci, C. (2004). *Coping With Medical Polysemy in the Semantic Web*: *The Role of Ontologies*. Studies in Health Technology and Informatics, 107 (1), 416–419.

PostgreSQL Global Development Group. (2022). *PostgreSQL*. https://www.postgresql.org (Last Access: 03/31/2022)

PrimeTek Informatics. (2022). *PrimeFaces*. https://primefaces.org (Last Access: 03/31/2022)

Prokosch, H.-U., Bahls, T., Bialke, M., Eils, J., Fegeler, C., Gründner, J., Haarbrandt, B., Hampf, C., Hoffmann, W., Hund, H., Kampf, M., Kapsner, L., Kasprzak, P., Kohlbacher, O., Krefting, D., Mang, J., Marschollek, M., Mate, S., Müller, A., Prasser, F., Sass, J., Semler, S., Stenzhorn, H., Thun, S., Zenker, S., and Eils, R. (2022). *The COVID-19 Data Exchange Platform of the German University Medicine* Proceedings of the Medical Informatics Europe Conference (MIE). (accepted).

RedHat. (2022). *Hibernate ORM*. https://hibernate.org/orm (Last Access: 03/31/2022)

Regenstrief Institute. (2022). *Logical Observation Identifiers Names and Codes* (*LOINC*). https://loinc.org (Last Access: 03/31/2022)

Robinson, P., and Mundlos, S. (2010). *The Human Phenotype Ontology*. Clinical Genetics, 77 (6), 525–534.

Saripalle, R., Runyan, C., and Russell, M. (2019). *Using HL7 FHIR to Achieve Interoperability in Patient Health Record*. Journal of Biomedical Informatics, 94, 103188.

SAS Institute Incorporated. (2021). *SAS Clinical Standards Toolkit*. https://support.sas.com/rnd/base/cdisc/cst (Last Access: 03/31/2022)

Sass, J., Bartschke, A., Lehne, M., Essenwanger, A., Rinaldi, E., Rudolph, S., Heitmann, K., Vehreschild, J., von Kalle, C., and Thun, S. (2020). *The German Corona Consensus Dataset* (*GECCO*): *A Standardized Dataset for COVID-19 Research in University Medicine and Beyond*. BMC Medical Informatics and Decision Making, 20 (1), 341.

Schreiber, G., Raimond, Y., Manola, F., Miller, E., and McBride, B. (Eds.). (2014). *RDF 1.1 Primer* (*W3C Working Group Note*). https://www.w3.org/TR/rdf11-primer (Last Access: 03/31/2022)

Shankar, R., Martins, S., O'Connor, M., Parrish, D., and Das, A. (2006). *Epoch*: *An Ontological Framework to Support Clinical Trials Management*. Proceedings of the International Workshop on Healthcare Information and Knowledge Management, 25–32.

Shankar, R., Martins, S., O'Connor, M., Parrish, D., and Das, A. (2007). *An Ontology-Based Architecture for Integration of Clinical Trials Management Applications*. Proceedings of the Annual Symposium of the American Medical Informatics Association (AMIA), 661–665.

Sioutos, N., de Coronado, S., Haber, M., Hartel, F., Shaiu, W.-L., and Wright, L. (2007). *NCI Thesaurus*: *A Semantic Model Integrating Cancer-Related Clinical and Molecular Information*. Journal of Biomedical Informatics, 40 (1), 30–43.

SNOMED International. (2022). *SNOMED CT*. https://www.snomed.org/snomed-ct (Last Access: 03/31/2022)

Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Champin, P.-A., and Lindström, N. (2020). *JSON-LD 1.1*: *A JSON-Based Serialization for Linked Data* (*W3C Recommendation*). https://www.w3.org/TR/json-ld11 (Last Access: 03/31/2022)

Stenzhorn, H. (2022). *MOnSTER Online*. https://purl.org/monster (Last Access: 03/31/2022)

Stenzhorn, H., David, R., and Kuchinke, W. (2012). *Report on the Validation and Certification of ObTiMA and DoctorEye*: *Deliverable No. 9.3* (*Technical Report*).

Stenzhorn, H., Weiler, G., Brochhausen, M., Schera, F., Kritsotakis, V., Tsiknakis, M., Kiefer, S., and Graf, N. (2010). *The ObTiMA System - Ontology-Based Managing of Clinical Trials*. Studies in Health Technology and Informatics, 160 (2), 1090–1094.

Structural Informatics Group (SIG). (2019). *Foundational Model of Anatomy* (*FMA*). http://si.washington.edu/projects/fma (Last Access: 03/31/2022)

SyncRO Soft SRL. (2022). *Oxygen XML Editor*. https://www.oxygenxml.com (Last Access: 03/31/2022)

Taye, M. (2010). *Understanding Semantic Web and Ontologies*: *Theory and Applications*. Journal of Computing, 2, 182–192.

Tran, V.-A., Johnson, N., Redline, S., and Zhang, G.-Q. (2011). *OnWARD*: *Ontology-Driven Web-Based Framework for Multi-Center Clinical Studies*. Journal of Biomedical Informatics, S48–S53.

Vandenbussche, P.-Y., Atemezing, G., and Poveda-Villalón. (2021). *Linked Open Vocabularies* (*LOV*). https://lov.linkeddata.es (Last Access: 03/31/2022)

Vandenbussche, P.-Y., Atemezing, G., Poveda-Villalón, M., and Vatant, B. (2017). *Linked Open Vocabularies* (*LOV*): *A Gateway to Reusable Semantic Vocabularies on the Web*. Semantic Web, 8 (3), 437–452.

van Rees, R. (2008). *Clarity in the Usage of the Terms Ontology*, *Taxonomy and Classification*. Civil Engineering, 20, 432.

VMware, Incorporated. (2022a). *Spring Framework*. https://spring.io/projects/spring-framework (Last Access: 03/31/2022)

VMware, Incorporated. (2022b). *Spring Security*. https://spring.io/projects/spring-security (Last Access: 03/31/2022)

von Kalle, C., Thun, S., and Vehreschild, J. (2021). *German Corona Consensus Data Set* (*GECCO*) (*Technical Report*).

W3C OWL Working Group (W3C-OWG). (2012). *OWL 2 Web Ontology Language Document Overview* (*Second Edition*) (*W3C Recommendation*). https://www.w3.org/TR/owl2-overview (Last Access: 03/31/2022)

W3C Semantic Web Healthcare and Life Sciences Interest Group (W3C-SW-HCLS-IG). (2012). *Clinical Observations Interoperability - CDISC*. https://www.w3.org/wiki/HCLS/ClinicalObservationsInteroperability/CDISC (Last Access: 03/31/2022)

Walther, B., Hossin, S., Townend, J., Abernethy, N., Parker, D., and Jeffries, D. (2011). *Comparison of Electronic Data Capture (EDC) With the Standard Data Capture Method for Clinical Trial Data*. *PLOS ONE*, 6 (9), e25348.

Web Hypertext Application Technology Working Group (WHATWG). (2019). *HTML (Living Standard)*. https://html.spec.whatwg.org (Last Access: 03/31/2022)

Weber, S., and Heitmann, K. (2021). *Interoperabilität im Gesundheitswesen: Auch für digitale Gesundheitsanwendungen (DiGA) verordnet*. Bundesgesundheitsblatt - Gesundheitsforschung - Gesundheitsschutz, 64 (10), 1262–1268.

Whetzel, P., Noy, N., Shah, N., Alexander, P., Nyulas, C., Tudorache, T., and Musen, M. (2011). *BioPortal: Enhanced Functionality via New Web Services From the National Center for Biomedical Ontology to Access and Use Ontologies in Software Applications*. Nucleic Acids Research, 39 (Web Server issue), W541-5.

World Health Organization (WHO). (2022). *International Statistical Classification of Diseases and Related Health Problems (ICD)*. https://www.who.int/standards/classifications/classification-of-diseases (Last Access: 03/31/2022)

Williams, T., and Oliva, A. (2017). *Breaking the Mold: Clinical Trials Data as RDF*. Proceedings of the PhUSE Annual Conference.

Yiallouros, M., Graf, N., and Tallen, G. (Eds.). (2009). *Wilms Tumour (Nephroblastoma) – Brief Information*. https://www.gpoh.de/kinderkrebsinfo/content/diseases/solid_tumours/wilms_tumour_nephroblastoma/pohwilms_patinfokurz120120611/index_eng.html (Last Access: 03/31/2022)

Zemmouchi-Ghomari, L., and Ghomari, A. (2012). *Ontology Versus Terminology, From the Perspective of Ontologists*. International Journal of Web Science, 1, 315–331.

Curriculum Vitae

For data protection reasons, the curriculum vitae is not published in the electronic version of the dissertation.

Erklärung gemäß § 7 Abs. 1 Nr. 2

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Bei der Auswahl und Auswertung folgenden Materials haben mir die nachstehend aufgeführten Personen in der jeweils beschriebenen Weise

☐ unentgeltlich

☐ entgeltlich

geholfen:

1. -
2. -

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberaterinnen/Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder in ähnlicher Form in einem anderen Verfahren zur Erlangung des Doktorgrades einer anderen Prüfungsbehörde vorgelegt.

Ich versichere an Eides statt, dass ich nach bestem Wissen die Wahrheit gesagt und nichts verschwiegen habe.

Die Bedeutung der eidesstattlichen Erklärung und die strafrechtlichen Folgen einer unrichtigen oder unvollständigen eidesstattlichen Erklärung sind mir bekannt.

Sankt Ingbert, den 27. April 2022

Holger Stenzhorn

Tag der Promotion:	03. August 2022
Dekan:	Univ.-Prof. Dr. med. Michael D. Menger
Berichterstatter:	Prof. Dr. med. Norbert Graf
	Univ.-Prof. Dr. rer. nat. Andreas Keller