FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Ontologický model pro RDF export z nástroje Data Stewardship Wizard
Student:	Bc. Jana Martínková
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Obor / specializace:	Manažerská informatika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Nástroj Data Stewardship Wizard (DSW) je flexibilním dotazníkovým nástrojem umožňujícím specifikaci dotazníků a šablon pro export dokumentů z odpovědí. V současné době existují různé šablony zaměřené na použití lidmi (čtení). Podporou exportu do RDF dle odpovídající ontologie by mohlo být umožněno další efektivní zpracování dat z DSW.

- Seznamte se s DSW, strukturou tzv. znalostních modelů a dotazníků i vývojem šablon pro export dokumentů.
- Seznamte se s technologiemi RDF a tvorbou RDFS/OWL ontologií.
- Navrhněte RDFS/OWL ontologii popisující struktury v DSW, zohledněte také možnost propojení s již existujícími ontologiemi a slovníky.
- Navrhněte RDF reprezentaci dat z dotazníků dle navržené ontologie.
- Implementujte šablonu pro export dotazníků v RDF dle návrhu.
- Popište a demonstrujte přínosy využití RDF pro reprezentaci dotazníků z pohledu uživatelů i organizací, které je vyhodnocují.
- Zhodnoťte možnosti tvorby šablon pro DSW a navrhněte možná zlepšení.

Elektronicky schválil/a Ing. David Buchtela, Ph.D. dne 25. ledna 2023 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Ontologický model pro RDF export z nástroje Data Stewardship Wizard

Bc. Jana Martínková

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

28. června 2023

Poděkování

Ráda bych poděkovala Ing. Markovi Suchánkovi za cenné rady, věcné připomínky a hlavě vstřícnost a ochotu během konzultací při vypracování této diplomové práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 28. června 2023 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Martínková, Jana. *Ontologický model pro RDF export z nástroje Data Stewardship Wizard*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Tato diplomová práce se věnuje návrhu ontologie a exportní šablony dat nástroje Data Stewardship Wizard do RDF reprezentace. V práci je popsán průběh návrhu ontologie struktur tohoto nástroje, zapsaná ve formátu Turtle, která využívá 24 konceptů z existujících ontologií. Dále práce obsahuje návrh RDF reprezentace dat ze zodpovězených dotazníků dle navržené ontologie, taktéž ve formátu Turtle, a popis implementace šablony pro export dat do navržené RDF reprezentace v jazyce Jinja2. V práci jsou představeny použité technologie jako RDF a jeho formáty, jazyky RDFS, OWL a nástroje užívané pro tvorbu a validaci ontologií i jazyk pro tvorbu šablon Jinja2. V závěru práce je zhodnocení přínosů výsledné RDF reprezentace a posouzení možností tvorby exportních šablon pro nástroj DSW.

Klíčová slova RDF, RDFS, OWL, DSW, Jinja2, Turtle, ontologie, exportní šablona

Abstract

This diploma thesis deals with the design of an ontology and an export template for the Data Stewardship Wizard tool into RDF representation. The thesis describes the proposed ontology of the tool's structures, written in the Turtle format, which utilizes 24 concepts from existing ontologies. Additionally, the thesis includes the design of RDF representation for data from answers to questionnaires based on the proposed ontology, also in the Turtle format, and a description of the implementation of the data export template into the designed RDF representation using the Jinja2 language. The thesis introduces technologies used, such as RDF and its formats, RDFS, OWL and tools used for ontology creation and validation, as well as the Jinja2 templating language. In conclusion, the thesis evaluates the benefits of the resulting RDF representation and assesses the possibilities of creating export templates for the DSW tool.

Keywords RDF, RDFS, OWL, DSW, Jinja2, Turtle, Ontology, Export Template

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Plánování správy dat v rámci projektu nabývá v dnešní době čím dál tím větší důležitosti, jelikož vhodná správa dat má za následek přesnější sběr, uložení i zacházení s daty. To zvyšuje jejich potenciální využitelnost a přínos nejen v rámci daného projektu, ale také v širších oblastech výzkumu. [1]

Existují různé nástroje pro usnadnění procesu vytváření plánu správy dat. Jedním z nich je Data Stewardship Wizard (DSW) [2], představující flexibilní dotazníkový nástroj, který umožňuje uživatelům specifikovat dotazníky a vytvářet šablony pro export dokumentů založených na odpovědích dotazníků. Tím se tento nástroj stává využitelný v mnoha dalších oblastech, kde je důležité zachytit informace získané na základě předdefinovaných dotazníků.

Nástroj DSW umožňuje z dat získaných z dotazníků generovat výsledné dokumenty. K tomuto generování se využívají exportní šablony. Momentálně dostupné exportní šablony nástroje jsou převážně zaměřeny na čitelnost a použití lidmi, nikoli pro strojové zpracování. Poskytují tak omezené množnosti pro další automatizované zpracování a výměnu dat. S ohledem na potřeby pokročilejšího zpracování a využití těchto dat je důležité implementovat možnost exportu do Resource Description Framework (RDF) [3] reprezentace spolu s využitím odpovídající ontologie. RDF s pomocí ontologie struktur nástroje DSW by zajistila přesný popis datových zdrojů, umožnila by jejich strojovou zpracovatelnost, garantovala by možnost komplexní analýzy dat v podobě dotazování a zajistila by možnou propojitelnost dat i s externími zdroji.

V této práci bude, na základě analýzy nástroje DSW, vytvořena ontologie popisující jeho struktury, se zvážením propojení s již existujícími ontologiemi za účelem dalšího zlepšení interoperability. Dále bude navržena RDF reprezentace dat plynoucích z odpovědí dotazníků s ohledem na vytvořenou ontologii. Dle tohoto návrhu bude implementována exportní šablona dat nástroje DSW do RDF reprezentace. Součástí práce bude posouzení přínosů plynoucích z tvorby a možného využití vytvořené exportní šablony do RDF reprezentace. Stejně tak zhodnocení možností tvorby exportních šablon pro nástroj DSW.

Kapitola 1

Cíl práce

Cílem této práce je formulovat ontologii, která popisuje struktury nástroje DSW. A navrhnout reprezentaci těchto struktur pomocí RDF, která je založená na vytvořené ontologii. Tato práce také zahrnuje implementaci exportní šablony pro převod dat z dotazníků nástroje DSW do navržené RDF reprezentace.

Nejprve bude provedena podrobná analýza nástroje se zaměřením na dotazníky, využívané znalostní modely a na možnosti tvorby exportních šablon včetně jazyka pro tvorbu šablon Jinja2. Budou představeny známé ontologie v oblasti Data Management Plan (DMP), používané technologie a standardy jako je RDF, jeho konkretní formáty, jazyky Resource Description Framework Schema (RDFS) a Web Ontology Language (OWL). Dále budou popsány vybrané nástroje pro tvorbu ontologií, jejich validaci i dokumentaci.

V této práci bude nejdříve navrhnuta ontologie struktur nástroje, včetně zvážení propojení s existujícími ontologiemi, která bude validována pomocí příslušných nástrojů. Navržená ontologie bude následně využita pro návrh RDF reprezentace dat z dotazníků a bude definován způsob zpracování uložených dat za účelem jejich výpisu. Dle návrhu RDF reprezentace bude implementována exportní šablona v jazyce Jinja2, jejíž export bude validován za účelem zajištění správnosti výsledných dat.

Součástí práce bude také posouzení přínosů využití RDF reprezentace odpovědí z dotazníku a také zhodnocení možností tvorby exportních šablon nástroje DSW.

Kapitola 2

Data Stewardship Wizard

Data Stewardship Wizard (DSW) [2] je nástroj navržen primárně pro plánování správy dat i tvorbu výsledného plánu neboli DMP. Jeho cílem je usnadnění průběhu vytvoření nejen plánu správy dat, ale obecně zachycení informací získaných na základě zodpovězených dotazníků s předdefinovanou strukturou. Celý průběh vytvoření výsledného dokumentu začíná u tvorby znalostního modelu, který zachycuje potřebné znalosti a požadovaná rozhodnutí. Tento model je následně převeden do dotazníku, který je jedinečný pro každý projekt. Uživatel, konkrétně data steward, následně vyplní dotazník na základě pravdivých informací o projektu. Po vyplnění dotazníku je možné získaná data a informace vygenerovat do výsledného dokumentu pomocí vybrané exportní šablony a výstupního formátu.

DSW se skládá z několika částí a to: znalostního modelu, open-source webového dotazovacího nástroje a nástroje pro sestavení výsledného dokumentu. Webový dotazovací nástroj byl vyvinut na České vysoké učení technické v Praze (ČVUT) a slouží k prezentaci dotazníků a ukládání odpovědí zadaných uživatelem.

Znalostní modely jsou určeny k uchování stromové struktury potřebných znalostí sloužících jako podklad pro dotazníky. V nástroji je možné vytvořit zcela nový znalostní model, anebo využít existující znalostní modely či jejich modifikace. Pro využití DSW v oblasti DMP existuje několik již předdefinovaných znalostních modelů, které vycházejí z myšlenkové mapy [4] zpracované Robem Hooftem, který ji formuje od roku 2013. Ta se soustředí primárně na oblast přírodních věd, ale poznatky z data managementu jsou aplikovatelné i na ostatní domény. Myšlenková mapa vychází z otázek důležitých při plánování projektu a možných odpovědí na základě zkušeností expertů. Obsahuje přes 600 uzlů a navíc tradiční stromovou strukturu doplňují křížové reference, které propojují odlišné části. Existuje několik znalostních modelů, z nichž základní, vycházející z myšlenkové mapy Roba Hoofta se nazývá *Common DSW Knowledge Model* [5]. Znalostní modely je možné modifikovat na základě požadavků projektu či instituce a takto modifikované modely je možné

sdílet v registru [6]. Struktura znalostních modelů je blíže popsána v 2.2.1.

Důležitou částí DSW je sestavení dat ze zodpovězeného dotazníku do výsledného dokumentu. Díky tomu, že DSW využívá strukturovaně uložená data, primárně z otázek s uzavřenými odpověďmi, na základě kterých generuje text, je výsledný vygenerovaný dokument lépe srozumitelný. Většina nástrojů totiž využívá části komentářů, které výzkumníci zapsali do textových odpovědí otevřených otázek. V těchto nástrojích jsou tak data uložena v nestrukturované podobě a to znemožňuje generovat výsledky dotazníku ve srozumitelné formě.

Existuje několik předdefinovaných šablon DMP dokumentů, do kterých je možné generovat data z dotazníku. Momentálně je k dispozici obecná šablona, která generuje všechny odpovědi z dotazníku *Questionnaire Report* [7], dále šablony založené na *Horizon Europe* [8], *Horizon 2020 DMP* [9] a *Science Europe* [10].

Šablony může data steward přizpůsobit na základě požadavků projektu nebo instituce. Díky strukturovanému uložení dat z dotazníků je možné využívat tyto upravené šablony. Vytvořené šablony mohou být sdíleny do registru [6], který obsahuje všechny šablony vytvořené DSW týmem a navíc šablony, které si uživatelé DSW zformovali a chtěli je sdílet dále pro užití dalšími projekty a institucemi.

Díky přizpůsobitelnému návrhu DSW je možné upravit jak znalostní modely, tak šablony pro export výsledných dokumentů dle potřeb použití. DSW se tak stává nejen nástrojem pro DMP, ale poskytuje téměř neomezené možnosti uplatnění.

2.1 Oblasti využití DSW

Původně je DSW nástrojem pro plánování i tvorbu plánu správy dat primárně v projektech v oblasti vědy a výzkumu. Tento plán je klíčovým prvkem pro správu dat a obsahuje mimo jiné informace o vzniku dat, jejich využití a dostupnosti. V posledních letech plánování správy dat v rámci projektu nabylo důležitosti hlavně ze stran grantových agentur či vědeckých institucí, a to za účelem využitelnosti výsledných dat ne jen v primárním projektu. Vhodná správa dat má tak za následek přesnější sběr, uložení a zacházení s daty, čímž je zvýšena jejich možná využitelnost a přínos i v projektech z jiných oblastí výzkumu. DSW podporuje projekt v celém jeho průběhu i po jeho skončení, a tak se minimalizují rizika spojená se správou dat v každé fázi výzkumu. [1]

Nástroj vyzdvihuje výhody plynoucí ze správy dat, jako například doporučení prostředků pro pomoc se sestavením a udržením metadat, doporučení služeb pro práci s příslušnými datovými formáty nebo ohodnocení přístupu k řízení dat na základě Findable, Accessible, Interoperable, Reusable (FAIR) principů [2, 11]. V *Common DSW Knowledge Model* [5] znalostním modelu a z něho vycházejících modelů, jsou uzavřené otázky obohaceny o tuto klasifi-

kaci. U jednotlivých odpovědí je zdůrazněno jakou metriku a jakým způsobem ji ovlivňuje. Díky tomu je data steward co nejvíce směrován k produkci otevřených dat, nicméně i tak může doporučení ignorovat.

DMP nástroje získávají informace od pracovníků o datech v projektu a projektu jako takovém na základě dotazování. Oproti ostatním dobře známým nástrojům pro správu dat využívá DSW v hojné míře uzavřené otázky s minimální množinou možných odpovědí. Předchází tím kopírování textů z předešlých projektů za účelem urychlení vyplnění dotazníku i s vědomostí vyplnění nepřesných informací. DSW dále umožňuje vybrat následující dotazy na základě již zodpovězených otázek tak, aby byly co nejrelevantnější a zároveň pokrývali všechny potřebné oblasti. Některé odpovědi jsou získány přímo z propojených externích služeb, což urychluje vyplnění dotazníku a zamezuje vzniku nepřesností.

DSW bylo využito již v několika oblastech mimo primární účel jeho použití. Dále jsou popsány tři případy, kdy bylo DSW využito v oblastech FAIR dat, sdílení zdravotních dat a zachycení software management plánů.

2.1.1 FAIR Implementation Profile Wizard

V roce 2004 iniciativa FAIR [11] vymezila pravidla pro udržitelné a efektivní publikování dat, které uskupila do tzv. *FAIR principů*. Tyto principy představují podněty a obecné postupy k dosažení objevitelných (findability), dostupných (accessibility), provázaných (interoperability) a znovupoužitelných (reusability) dat.

FAIR principy neurčují způsob implementace ani technické parametry či řešení, díky kterým data dosáhnou úspěšně FAIRifikace, ale jedná se pouze o doporučení v oblasti správy dat. V komunitě tak vznikají vlastní technická řešení, což může vést k eventuálnímu snížení kompatibility mezi nimi. Proto vznikla kolekce shromažďující různé implementace a lze si tak buď osvojit již existující řešení, nebo se rozhodnout pro svůj vlastní vývoj. Obsah této kolekce je tvořen FAIR Implementation Profile (FIP) [12]. FIP je koncept zachycující seznam zvolených technických řešení pro jednotlivé aspekty FAIR principů v rámci jedné organizace kolem domény znalostí.

FIP vytváří správce dat v organizaci a může ji vytvořit sám od základu, nebo k tomu využít vhodný nástroj. FAIR Implementation Profile Wizard (FIP Wizard) [13] je implementace FIP dotazníku v nástroji DSW. Dotazník zachycuje zvolená technická řešení díky spojení s registrem existujících technických implementací.

2.1.2 VODAN

V roce 2020 iniciativa Virus Outbreak Data Network (VODAN) [14] přišla s návrhem způsobu sdílení dat v návaznosti na pandemii koronaviru. Na rozšíření dostupnosti a přístupnosti k medicínským datům se instituce kon-

centrovali zejména po vypuknutí viru Ebola v letech 2013 až 2016. Pandemie koronaviru však přinesla nové výzvy především kvůli velkému objemu senzitivních dat, které se shromažďují v reálném čase, a tak nebylo možné využívat centrální datová úložiště.

VODAN navrhl uspořádání, ve kterém data nikdy neopustí instituci a díky tomu se instituce vypořádají s velkými objemy dat, jejich senzitivitou i regulacemi zamezujícími šíření těchto dat. Data jsou umístěna v úložišti instituce odkud pocházejí a algoritmy, vyhledávající a dotazující taková data, pouze datová úložiště navštěvují.

Tento prostředek pro sdílení dat je dostupný jako sada nástrojů VODAN in a Box (ViB) [15], který se skládá ze tří komponent. Dle [16] DSW zastává funkci nástroje pro vyplnění formulářů Electronic Case Report form (eCRF) pro hlášení pacientů World Health Organization (WHO). Díky tomu jsou data popsána strojově čitelným způsobem a jsou tak vystavena algoritmu pro vyhledávání a dotazování.

2.1.3 Software Management Plans

Díky univerzálnosti DSW je možné vyvinout doménově specifické znalostní modely i šablony dokumentů. Tato flexibilita umožnila vznik podpory pro Software Management Plan (SMP) [17, 18], který slouží pro popis postupů a metod ve všech fázích životního cyklu software. Je v něm zachycen například harmonogram, rozpočet, odpovědnosti členů týmu a jejich role. Podpora pro vývoj SMP vznikla v roce 2021 během události BioHackaton Europe.

2.2 Struktura znalostních modelů a dotazníků

Tato část se věnuje důkladně analýze prvků, které tvoří znalostní modely a dotazníky nástroje DSW. Struktura se skládá z několika komponent, které dohromady slouží k uchování znalostí a generování výsledných dokumentů. Tyto komponenty lze rozdělit do tří hlavních částí a to: znalostní model, doplňující informace a dotazníky s odpověďmi.

Pro účel analýzy jsou jednotlivé prvky podrobně popsány včetně jejich atributů a vlastností. Analýza byla tvořena na základě zkoumání samotného nástroje DSW [19] a jeho dokumentace [20].

2.2.1 Struktura znalostních modelů

Znalostní modely představují esenciální součást nástroje DSW, tvořící kompletní vědomostní základ pro vytvoření dotazníku. V této sekci jsou detailně popsány prvky struktury DSW spadající do této části.

Nástroj DSW k identifikaci prvků využívá jedinečný identifikátor typu Universal Unique Identifier (UUID) za účelem zabránění konfliktu při identifikaci.

2.2.1.1 Knowledge model

Knowledge model je základní entitou stromové struktury znalostního modelu. Skládá se z *Metrics*, *Phases*, *Question Tags*, *Integrations* a *Chapters*. Ke každému modelu znalostí je přiřazen identifikátor `uuid` a dále `annotations` neboli anotace představující dodatečnou informaci. Ty se skládají z odstavcového textu `value` a jednořádkového textu `key`, což umožňuje reprezentovat dvojici klíč-hodnota.

2.2.1.2 Metric

Metric čili metriky jsou klasifikace, o které jsou doplněny odpovědi na otázky typu *Options Question*. Na základě zodpovězeného dotazníku nebo jeho části, jsou metriky vyhodnoceny a mohou být součástí výsledného exportovaného dokumentu. V aktuální verzi základního modelu *Common DSW Knowledge Model* [5] jsou definovány klasifikace dle FAIR principů [11], otevřenosti dat a klasifikace na základě *Good DMP Practice*. *Metric* jsou identifikovány pomocí `uuid` a se skládají z:

- `title` slouží k označení metriky.
- `abberviation` určuje zkrácený název metriky, vycházející z definovaného `title`.
- `description` slouží k popisu metriky v odstavcovém textovém poli, ve kterém data steward může využít značkovací jazyk *Markdown*.
- `annotations` poskytují dodatečné informace a skládají se z `value` a `key`, které reprezentují dvojici klíč-hodnota.

2.2.1.3 Phases

Phases neboli fáze představují důležitý prvek pro určení stádia v projektu. V základním modelu *Common DSW Knowledge Model* [5] jsou definovány čtyři fáze rozdělující proces na dobu před odevzdáním návrhu, před odevzdáním DMP, před dokončením projektu a po jeho dokončení. Otázky mají určené fáze a spolu s určením fáze celého projektu je možné uživatele upozornit na nezodpovězené otázky, které patří do daného stádia projektu. *Phases* se identifikují pomocí `uuid` identifikátoru a dále se určuje:

- `title` pro pojmenování dané fáze.
- `description` slouží k popisu fáze, ve kterém výzkumník může využít značkovací jazyk *Markdown*.
- `annotations` slouží pro zachycení případných dodatečných informací. Jedná se dvojici klíč a hodnota.

2.2.1.4 Tag

Tag čili štítek slouží k označení otázek. DSW tímto způsobem odlišuje otázky spojené s určitým výzkumem či aspektem a je tak možné otázky filtrovat dle požadavků. Momentálně základní model *Common DSW Knowledge Model* [5] obsahuje čtyři štítky a to *Science Europe DMP*, *maDMP*, *Horizon 2020 DMP* a *Horizon Europe DMP*. Štítky jsou také identifikovány pomocí `uuid` a pak je u nich stanoven:

- `name` pro určení jména štítku.
- `description` pro popis štítku.
- `colour` slouží k definici barvy pro zvýraznění štítku. Výzkumník může zvolit z dvaceti předdefinovaných barev, které automaticky doplní *Hex colour code*, nebo může vepsat do pole *Hex colour code* požadovanou barvu sám.
- `annotations` slouží k doplnění případných informací.

2.2.1.5 Integrations

Integrations neboli integrace jsou určeny k propojení otázky s externími službami či zdroji za účelem poskytnutí odpovědi v podobě prvku z externího zdroje prostřednictvím Application Programming Interface (API). Nyní jsou v *Common DSW Knowledge Model* [5] modelu definované integrace s *FAIRsharing* [21], *Crossref*, *ROR* a *Wikidata*. *Integrations* se identifikují pomocí `uuid`, a dále se u nich určuje:

- `id` slouží k identifikaci integrace v rámci souboru *integration.yml*, který obsahuje konfigurační hodnoty, jako například API klíče nebo tokeny.
- `name` se užívá k označení integrace v rámci celého editoru znalostního modelu.
- `logo` slouží k uložení loga integrace, které se zobrazuje v editoru znalostního modelu. Je zadáváno do textového pole jako Uniform Resource Locator (URL) odkaz na logo nebo zakódované v *Base64*.
- `props` se využívá k definici proměnných doplňujících dotaz na službu. Uživatel zadá označení proměnné do jednořádkového textového pole. Hodnoty proměnných jsou vyplněny až uživatelem, který odpovídá na otázku obohacenou o integraci.
- `item URL` vyjadřuje URL adresu přesné položky služby. Jedná se o jednořádkové textové pole pro definici URL adresy a proměnných k upřesnění položky.

- `annotations` slouží k doplňujícím informacím. Jedná se dvojici klíč-hodnota.
- `type` určuje typ integrace. DSW momentálně rozděluje 2 typy integrací a to API a *Widget*, ze kterých výzkumník zvolí v *dropdown* výběru.
- `request` slouží ke konfiguraci parametrů odesílaného dotazu na službu pro typ API integrace. Skládá se z `rqUrl`, definující URL adresu služby, `rqMethod`, určující HTTP metodu dotazu, `rqHeaders`, definující *Request HTTP Headers*, `rqBody`, vymezující `HTTP body` a atribut `rqEmptySearch` označující zda je možné dané službě poslat prázdný dotaz. Tento atribut může být nastaven na jednu ze dvou hodnot - *pravda* nebo *nepravda*.
- `response` je určena k definici parametrů odpovědi služby typu API. Je složena z `rsListField` pro vyjádření cesty k části odpovědi, která obsahuje seznam položek. Dále obsahuje `rsItemId`, který slouží k určení konkrétní položky ze seznamu a `rsItemTemplate` určující způsob jakým se zobrazí nalezená položka uživateli vyplňujícímu DMP.
- `widgetUrl` je definována pouze v případě, že se jedná o integraci typu *Widget*. Slouží k připojení URL odkazu na *Widget* implementovaný pomocí *DSW Integration Widget SDK*.

2.2.1.6 Chapters

Poslední a velmi důležitou součástí znalostního modelu jsou *Chapters* neboli kapitoly, které seskupují otázky podobných témat. *Chapters* jsou definovány identifikátorem `uuid`, dále mají:

- `title` slouží k označení kapitoly.
- `text` slouží pro krátký popis kapitoly, ve kterém je možné použít značkovací jazyk *Markdown*.
- `questions` představuje seznam otázek patřících do kapitoly.
- `annotations` slouží pro anotace.

2.2.1.7 Question

Question neboli otázka je prvek díky kterému jsou získávány informace a data od uživatelů. *Question* je definována také pomocí `uuid` a uvádí se u ní:

- `title` pro uvedení otázky.
- `type` slouží k určení typu otázky. V DSW momentálně existuje pět typů otázek, ze kterých uživatel při definici otázky zvolí v *dropdown* výběru.

- `text` je určen pro krátký popis otázky, ve kterém je možné použít značkovací jazyk *Markdown*.
- `tags` představuje množinu štítků *Tag*, kterými je možné otázku označit. Na výběr jsou definované štítky, které lze vybrat pomocí zaškrtávacího políčka.
- `requiredPhase`, určuje fázi neboli *Phase* vývoje DMP, ve které je doporučeno zodpovědět otázku.
- `required` určuje, zda je zodpovězení této otázky povinné.
- `references` představuje seznam nápomocných odkazů neboli *Reference*.
- `annotations` slouží k dodatečným informacím.
- `expert` představuje kontakt na experta, který je schopen v oblasti této otázky poradit.

Existuje pět typů otázek a to:

- ***Value*** představuje otevřenou otázku, u které je potřeba definovat datový typ hodnoty `valueType`, kterou bude uživatel vyplňující dotazník DMP vkládat. Existuje několik předdefinovaných datových typů a to: `string`, `number`, `date`, `datetime`, `time`, `text`, `email`, `url` a `color`.
- ***Integration*** slouží k vyplnění informací za pomoci integrované externí služby. Zvolenou externí službu je třeba definovat v atributu s názvem `integration`. Tato otázka se uživateli zobrazí jako pole pro zadání hodnoty s našeptávačem z určené služby. V případě, že zvolená integrace má z její definice předurčené proměnné (jako je tomu například u *FAIRsharing* [21] integrace), pak se uživateli zobrazí další pole `props` pro specifikaci dotazů na službu.
- ***Multi-Choice*** je určena pro výběr jedné nebo více z předdefinovaných odpovědí. DSW zobrazí otázku s výčtem možností neboli *Choice* se zaškrtávacími poli. Atribut `choices` představuje seznam se všemi předdefinovanými možnostmi *Choice*.
- ***Options*** slouží pro výběr právě jedné předdefinované odpovědi. DSW zobrazí otázku s výběrem možností čili *Answers*. Atribut `answers` představuje seznam možných odpovědí *Answer*. Tento typ otázek umožňuje vkládat navazující otázky na základě odpovědi primární (rodičovské) otázky. Navazující otázky se zobrazí pouze v případě kladné odpovědi u možnosti na kterou jsou připojeny. Díky tomuto typu otázek je možné neomezeně prohlubovat strom znalostí.

- ***List Of Items*** je typ otázky s možností neomezeného vkládání odpovědí uskupených do seznamu. Každá položka obsahuje tzv. *Item Template* složený z podotázek. Proto je u tohoto typu otázek nutné stanovit v atributu `followups` seznam otázek, které budou obsaženy v každé položce primární otázky. Podotázky pak mohou být jakéhokoli typu.

2.2.1.8 Choice

Choice je předdefinovaná odpověď na otázku typu *Multi-Choice*. Tato odpověď je také identifikována pomocí `uuid` a dále je u ní uveden:

- `label` představuje znění volby.
- `annotations` slouží k doplňujícím informacím.

2.2.1.9 Answer

Answer je odpověď speciálně na otázku typu *Options*. Tato odpověď je také identifikována pomocí `uuid` a dále se u ní uvádí:

- `label` představuje znění odpovědi.
- `advice` slouží k zachycení nápovědy ke zvolení odpovědi, kde je možné užít značkovací jazyk *Markdown*.
- `metricMeasures` představuje seznam metrických měr.
- `followups` obsahuje seznam navazujících otázek na tuto odpověď.
- `annotations` slouží k dodatečným informacím.

2.2.1.10 Metric Measure

Metric Measure čili míry metriky jsou klasifikace odpovědí *Answer*. Každá *Metric Measure* má:

- `metric` představuje vymezenou metriku podle které je odpověď hodnocena.
- `measure` říká v jaké míře odpověď metriku ovlivňuje. Je definována číslem mezi 0 a 1.
- `weight` slouží k definici důležitosti odpovědi pro metriku. Je definována číslem mezi 0 a 1.

2.2.1.11 Reference

Reference neboli odkaz slouží k propojení s pomocným URL odkazem, nebo odkazem na určitou kapitolu knihy *Data Stewardship for Open Science: Implementing FAIR Principles* [22]. *Reference* je identifikována pomocí `uuid` a dále se určuje:

- `type` slouží k definici typu odkazu, zda se jedná o odkaz na kapitolu knihy nebo o URL odkaz.
- `label` je název definovaný u reference typu URL, který se zobrazuje u otázky ve které se reference vyskytuje.
- `url` nese odkaz pro referenci typu URL.
- `shortUuid` se udává u typu reference odkazující na kapitolu knihy *Data Stewardship for Open Science: Implementing FAIR Principles* [22]. Představuje identifikátor pro konkrétní určení odkazované kapitoly.

2.2.1.12 Expert

Expert se uvádí u otázek a představuje kontakt na experta, který je schopen v kontextu otázky poradit. Je identifikován pomocí `uuid` a má:

- `name` jeho jméno, či název.
- `email` jeho kontaktní e-mail.

2.2.2 Struktura instance dotazníku

Dotazník představuje hlavní část DSW díky které jsou seskupena data. Konkrétní podoba instance dotazníku závisí na použitém znalostním modelu. Tato oblast prvků struktury DSW zahrnuje dotazník, konkrétní odpovědi na otázky dotazníku a uživatele.

2.2.2.1 Questionnaire

Questionnaire neboli dotazník je formulář reprezentující vybraný znalostní model. Jeho cílem je shromáždit informace týkající se sběru a práce s daty v rámci projektu. *Questionnaire* je identifikován pomocí `uuid`, dále má:

- `name` definuje jméno dotazníku i celého projektu.
- `description` slouží k popisu.
- `projectTags` představují štítky neboli *Tag* kterými lze označit celý projekt. Uživatel je sám definuje a může jich přidat neomezeně mnoho.

- `phase` slouží k určení momentální fáze projektu. Za pomoci zvolené příslušné fáze v průběhu projektu dokáže DSW zvýraznit části dotazníku, které je doporučeno mít ve vybrané fázi zodpovězené.
- `replies` odkazuje na seznam odpovědí zanesených do dotazníku.
- `version` slouží k odkazu na aktuální verzi dotazníku.
- `versions` je určen k odkazu na všechny existující verze dotazníku.
- `createdBy` představuje uživatele čili *User*, který dotazník vytvořil.

2.2.2.2 Questionnaire Version

Entita *Questionnaire Version* představuje konkrétní verzi dotazníku. Také se identifikuje pomocí `uuid` a dále se u ní uvádí:

- `createdBy` definuje autora dané verze dotazníku.
- `createdAt` určuje, kdy byla verze vytvořena.
- `updatedAt` definuje, kdy byla verze aktualizována.
- `name` představuje jméno verze dotazníku. Pojmenovat či přejmenovat se mohou i verze předcházející té nejnovější.
- `description` slouží k zachycení popisu verze.

2.2.2.3 Reply

Reply je entita představující odpověď na otázku bez ohledu na její typ. Odpovědi jsou identifikovány pomocí `path`, která je složena z `uuid` kapitoly, otázek a případně *Answer*, které k dané odpovědi vedli. U odpovědi je uvedeno:

- `createdBy` představuje autora odpovědi.
- `createdAt` definuje kdy byla odpověď vytvořena.
- `question` slouží k propojení s konkrétní otázkou.
- `replyType` definuje typ odpovědi. Existuje pět typů odpovědí, které závisí na typu otázky.

Typ odpovědi je určen typem otázky, se kterou je spojena. Existuje tedy pět typů odpovědí a to:

- ***StringReply*** odpovídá otázce typu *Value*. U tohoto typu odpovědi se ukládá hodnota, kterou zadal uživatel, do atributu `value`.

- ***ItemListReply*** je odpověď pro otázku typu *List Of Items*. Tento typ odpovědi má `value` obsahující identifikátory `uuid` odkazující na položky v seznamu odpovědí.
- ***IntegrationReply*** odpovídá otázce typu *Integration*. Obsahuje atribut `value`, ve kterém je hodnota odpovědi z externího zdroje a `itemId`, představující odkaz na položku externího zdroje. Dále dva atributy s názvem `isPlain` a `isIntegration` označující, zda byla požadovaná odpověď v externím zdroji nalezena, či nikoli.
- ***MultiChoiceReply*** je odpověď na otázku typu *Multi-Choice*, obsahující `value` odkazující na zvolené *Choice*.
- ***AnswerReply*** odpovídá otázce typu *Options*. Tento typ otázky obsahuje `value` odkazující na zvolenou *Answear*.

2.2.2.4 User

User je entita reprezentující uživatele DSW. Uživatel je identifikován pomocí `uuid` a dále se u něj uvádí:

- `firstName` neboli křestní jméno uživatele.
- `lastName` čili příjmení uživatele.
- `email` je atribut s e-mailem uživatele.
- `role` obsahuje jednu ze tří předdefinovaných rolí uživatele. Existující role jsou: *Admin*, *Researcher*, *Data Steward*.
- `imageUrl` je atribut obsahující URL profilového obrázku.
- `affiliation` představuje přidružení k předem definovaným afilacím v nastavení organizace.
- `premissions` představují oprávnění uživatele.
- `createdAt` definuje kdy byl uživatelský profil vytvořen.
- `updatedAt` definuje kdy byl uživatelský profil naposledy upraven.

2.2.2.5 Simple Author

Simple Author je entita, která slouží k reprezentaci uživatele, který je autorem buď verze dotazníku *Questionnaire Version*, nebo odpovědi *Reply*. Je identifikován pomocí `uuid`, který odpovídá `uuid` identifikátoru entity *User*. V případě, že uživatel vytvoří novou verzi dotazníku, nebo odpoví na dotaz, pak je dle totožného `uuid` vytvořena instance entity *Simple Author*. Dále obsahuje `firstName`, `lastName`, `imageUrl`, které jsou totožné s hodnotami atributů entity *User*. Navíc pak obsahuje `gravatarHash` odkazující na *Gravatar*.

2.2.3 Struktura doplňujících informací

Aby mohly být data z dotazníku exportována do dokumentu je zapotřebí dalších doplňujících informací, díky kterým je umožněno vygenerovat kompletní výsledný dokument.

2.2.3.1 Document Context

Document Context čili kontext dokumentu je hlavní entitou představující souvislosti výsledného dokumentu a je propojen jak s prvky z oblasti znalostního modelu tak s prvky z instance dotazníku a jeho odpovědí. Skládá se z *Context Configuration*, *Document*, *Organization*, *Package*, *Report* a je propojen i s *Questionnaire*, *Phase* a *Knowledge Model*.

2.2.3.2 Context Configuration

Context Configuration je entita představující konfiguraci instance DSW. Obsahuje atribut `clientURL` definující URL instance DSW.

2.2.3.3 Document

Document je entita reprezentující exportovaný dokument. Je definována pomocí `uuid` a dále obsahuje:

- `createdAt` definuje kdy byl dokument vytvořen.
- `updatedAt` definuje kdy byl dokument naposledy upraven.

2.2.3.4 Organization

Organization představuje organizaci spravující DSW instanci. Organizace je definována svým `id` identifikátorem, který slouží k identifikaci znalostních modelů vytvořených v instanci a má:

- `name` definuje jméno organizace spravující instanci.
- `description` slouží pro popis organizace.
- `affiliation` slouží pro definici afilací, ke kterým se přiřazují uživatelé z organizace.

2.2.3.5 Package

Package je entita seskupující metadata o znalostním modelu. Je identifikována pomocí `id` identifikátoru a obsahuje:

- `organizationId` představuje identifikátor organizace.
- `kmId` je identifikátor znalostního modelu.

- `version` je verze znalostního modelu, která tvoří základ dotazníku.
- `versions` slouží pro výčet všech existujících verzí znalostního modelu.
- `name` představuje název znalostního modelu.
- `description` představuje popis znalostního modelu.
- `createdAt` definuje kdy byl znalostní model vytvořen.

2.2.3.6 Report

Report představuje report z odpovědí dotazníku, který obsahuje informace o výsledcích metrik v rámci celého dotazníku i jednotlivých kapitol a počtu zodpovězených otázek. Jeho části se skládají z *Report Item*. Je identifikován pomocí `uuid` a dále obsahuje:

- `totalReport` představuje výsledky metrik v rámci celého dotazníku.
- `chapterReports` představuje výsledky jednotlivých kapitol.
- `createdAt` definuje kdy byl report vytvořen.
- `updatedAt` definuje kdy byl report naposledy upraven.

2.2.3.7 Report Item

Report Item je entita představující jednu část reportu, ať už se jedná výsledek z celého dotazníku, nebo z jedné jeho kapitoly. Skládá se z:

- `indications` představují indikace, které byly zohledněny při výpočtu a ty jsou vyjádřeny pomocí entity *Report Indication*.
- `metrics` odkazují na *Report Metric*, představující výsledek určité metriky.
- `chapter` slouží k určení kapitoly v případě, že se jedná o report z jedné konkrétní kapitoly.

2.2.3.8 Report Indication

Report Indication je entita sloužící k zachování indikací části reportu. Obsahuje:

- `answered` určuje počet zodpovězených odpovědí.
- `unanswered` určuje počet nezodpovězených odpovědí.
- `indicationType` říká, zda hodnoty v atributech `answered` a `unanswered` reprezentují všechny odpovědi dotazníku, nebo pouze ty které je doporučeno zodpovědět ve zvolené fázi.

2.2.3.9 Report Metric

Report Metric představuje výslednou míru dané metriky na základě odpovědí dotazníku. Obsahuje:

- `measure` představuje výsledek, jedná se o číslo mezi 0 a 1.
- `metric` definuje metriku výsledku.

2.2.3.10 Replies Container

Replies Container je pomocná struktura, díky které je umožněno přistupovat k odpovědím *Reply*.

2.2.3.11 Knowledge Model Entites

Knowledge Model Entites je, stejně tak jako *Replies Container*, pomocná struktura pro přístup ke komponentám znalostního modelu *Knowledge Model*.

2.3 Vývoj šablon pro export dotazníků

Pomocí úpravy šablony pro export dotazníku je možné dosáhnout libovolného formátu výsledného dokumentu. Všechny šablony, i již existující, jsou definované pomocí jazyka *Jinja2* [23], který je dále popsán v 3.7. Jak je popsáno v [24] vývoj i úpravu šablon lze provést lokálně v textovém editoru či v integrovaném vývojovém prostředí anebo lze využít *Document Template Editor* přímo v nástroji DSW. Lokální vývoj šablony vyžaduje mimo jiné *DSW Template Development Kit* neboli nástroj příkazové řádky.

Vývoj šablon v DSW je v části *Document Template Editors*, kde lze vytvářet jednotlivé editory *Document Template Editor*. V nich je možné definovat či upravovat šablonu i její obecné nastavení jako je například její název, popis či verze. Při úpravě kódu je možné nahlédnout jak bude výsledný dokument vypadat pro zvolený projekt (čili zanesená data) a jeho formát.

Veškeré potřebné informace k vytvoření výsledného dokumentu založeného na dotazníku obsahuje objekt `Document Context`. Obsah tohoto objektu je popsán v části 2.2.3.

Jednou z již existujících šablon je *Questionnaire Report* [7], která přímo reprodukuje obsah dotazníku do dokumentu přičemž zachovává jeho strukturu. Díky tomu ji lze použít jako referenční řešení pro zacházení s datovými objekty.

Kapitola 3

Terminologie a technologie

V této kapitole jsou blíže popsány termíny a technologie, které byly využity při návrhu a tvorbě ontologie struktur nástroje DSW i při tvorbě exportní šablony. Jsou popsány existující známé ontologie, standardy, jazyky, formáty, syntaxe a nástroje, které jsou využívány při návrhu a tvorbě ontologie.

3.1 Ontologie a slovníky

Slovník je soubor termínů, které jsou přesně definované a jejich výklad není závislý na kontextu [25].

Ontologie původně označuje filosofickou disciplínu, pocházející z Aristotelovy První filosofie, která zkoumá otázku bytí [26]. Dnes ale dle [27] význam tohoto konceptu v souvislosti s informačními technologiemi, spočívá v co nejúplnější definici pojmů a vztahů mezi nimi. Ontologie může pokrývat pojmy z vybrané oblasti, či problematiky nebo obecné koncepty nezávislé na doméně.

Využití ontologií i slovníků je dle [27] široké, hlavně z důvodu uchování znalostí a jednoduchého zachování doménové struktury. Například v oblasti znalostního inženýrství jsou dobře strukturované znalostní domény opakovaně využívány při řešení různých problémů. V oblasti sémantického webu se ontologie vyskytují při anotaci webových zdrojů, což umožňuje vyšší sémantickou interoperabilitu a lepší porozumění obsahu mezi různými systémy a uživateli.

3.1.1 Známé ontologie

V této sekci je přiblíženo několik ontologií a slovníků, které se týkají domény DMP. Použití již existujících ontologií a slovníků přispívá ke znovupoužitelnosti a propojitelnosti dat.

3.1.1.1 Dublin Core

Dublin Core (DC) [28] je ontologie pro popis elektronických zdrojů skládající se z minimální množiny termů. Ty byly sestaveny tak, aby byly použitelné v různých oblastech i jazycích a bylo možné je jednoduše doplnit o další požadované termy.

Pro DC se nejčastěji užívá prefix `dc` pro základní elementy a `dct` pro rozšířenou sadu termů.

3.1.1.2 Data Catalog Vocabulary

Data Catalog Vocabulary (DCAT) [29] je slovník určený k popisu datových sad v katalozích. Tento slovník byl zkonstruován za účelem propojitelnosti a dohledatelnosti datových sad napříč katalogy. Pro DCAT se nejčastěji užívá prefix `dcat`.

3.1.1.3 Friend of a Friend

Friend of a Friend (FOAF) [30] je ontologie, která se zaměřuje na virtuální identitu osob, jejích rolí, aktivit a vztahů k ostatním osobám. Zároveň slouží k popisu skupin, organizací a jejich aktivit. Pro tuto ontologii je nejčastěji užíván prefix `foaf`.

3.1.1.4 Schema.org

Schema.org ontology (Schema.org) [31] je sdílený slovník, který je spravován za účelem vytvoření, udržení a propagace schémat pro popis strukturovaných dat. Byl založen předními technologickými společnostmi, jako je například *Google* či *Microsoft*, a je spravován otevřenou komunitou. K označení Schema.org se nejčastěji užívá prefix `schema`.

3.1.1.5 Simple Knowledge Organization System

Simple Knowledge Organization System (SKOS) [32] představuje datový model, který slouží k popisu různých slovníků a taxonomií za účelem jejich možného propojení a vzájemného využití. Pro tuto ontologii se nejčastěji užívá prefix `skos`.

3.2 RDF

RDF [3] je standard pro meta-datový popis zdrojů, který byl vyvinut World Wide Web Consortium (W3C). Využívá se pro popis jak zdrojů dostupných na internetu, tak těch které nemusí ani fyzicky existovat.

Základní struktura popisu zdroje se skládá z takzvaných tripletů, obsahující subjekt, predikát a objekt. Tato trojice značí, že existuje jednosměrný

vztah mezi zdroji, definovaný pomocí predikátu, které jsou určené jako subjekt a objekt. Z množiny tripletů sestává tzv. *RDF Graf*, tedy graf definovaný jako množina uzlů s orientovaný hranami.

Dle [33] existují tři možnosti, jak definovat komponenty tripletů a to International Resource Identifier (IRI), literál a tzv. *blank node* společně nazývané *RDF termy*. Ne všechny komponenty smí být definovány pomocí kteréhokoli RDF termu. Subjektem může být IRI nebo blank node, predikátem pouze IRI a objektem jakýkoli RDF term.

IRI [33, 34] je zobecnění Uniform Resource Identifier (URI) ve smyslu rozšíření množiny Unicode znaků. IRI tak může obsahovat jakékoli znaky mimo těch rezervovaných. Znamená to, že každá URI je IRI, ale není tomu tak vždy i obráceně. IRI slouží k přesné definici zdroje a zajišťuje tak jejich nezaměnitelnost a jednoznačnost.

Blank nodes [3] neboli prázdné uzly se využívají v případě, kdy není možné nebo chtěné daný zdroj přesně a globálně identifikovat. Jedná se o uzly, které neobsahují informaci a slouží primárně k propojení s dalšími částmi RDF grafu. V rámci jednoho RDF grafu se může vyskytovat více těchto prázdných uzlů, proto mají vlastní identifikaci, která je platná pouze v lokálním rozmezí.

Literál [3] je užíván pro hodnoty, může se jednat o čísla, textové řetězce, pravdivostní hodnoty, data, či znaky a další. Literál je definován svou hodnotou, data typem a nepovinně jazykem. Slouží tedy jako nosič definované hodnoty s určeným data typem a případně i jazykem hodnoty. RDF využívá mnoho datových typů, které jsou definované v Extensible Markup Language Schema Definition (XSD) a dále definuje dva dodatečné typy a to `rdf:HTML` a `rdf:XMLLiteral` [33].

XSD [35] je jazyk, který se řadí mezi Extensible Markup Language (XML) schémata, sloužící k formálnímu popisu elementů jazyka XML. XSD byl vyvinut v roce 2001 *XML Schema Working Group* pod W3C. S XSD lze definovat elementy, atributy, vztahy a datové typy, kterým je povoleno se vyskytovat v XML dokumentu. Poskytuje množinu již předdefinovaných datových typů, na které lze odkazovat pomocí URI a jména datového typu. Například pro *Integer* je to `http://www.w3.org/2001/XMLSchema#integer`.

Předdefinované datové typy se dělí na primitivní a odvozené. Mezi primitivní se řadí například: *String*, *dateTime*, nebo *float*. Do odvozených spadá *Integer*, který je odvozen od typu *decimal* neboli desetinných míst, *Name*, nebo *Language* představující identifikátor jazyka.

V RDF existuje několik již předdefinovaných konstruktů, které se využívají při tvorbě ontologie a popisu v RDF. Mezi ně patří například třída *rdf:Property*, do které spadají všechny vlastnosti neboli *properties*. Nebo vlastnost *rdf:type*, která říká, že zdroj je instancí nějaké třídy.

Díky popisům metadat je realizovatelné zpracovávat komplikovaně strukturovaná data různých formátů a zároveň je možné machine-understandable data sdílet mezi vícero stranami bez ztráty znalosti struktur a bez požadavku na určitou technologii. [36]

RDF je pouze standard, který říká jakým způsobem metadata zapsat, ale nemá definovanou svoji konkrétní syntaxi. Pro jeho zápis lze využít různé jazyky, ale XML je díky jeho všestrannosti nejvíce využívaný. W3C vymezilo specifikaci RDF/XML, což je syntax pro vyjádření RDF za použití formátu XML. [33] Vybrané druhy formátů RDF dat jsou popsány v 3.5.

3.3 RDFS

RDFS [37] je ontologický jazyk RDF, který poskytuje aparát k rozlišení a popisu skupin souvisejících zdrojů, vztahů mezi nimi a jejich vlastnostmi. Jedná se o standard W3C, který byl vyvinut v roce 2004. RDFS poskytuje několik konstruktů pro skupiny zdrojů neboli *tříd* či *class* a jejich vlastnosti čili *property*.

Třída neboli *class* představuje množinu zdrojů, které na základě definice spadají do stejné skupiny. Může se jednat o třídu reprezentující například osobu, zvíře, nebo nemovitost. Zdroje patřící do dané třídy se nazývají její instance a mohou být v jednom okamžiku instancemi několika tříd najednou. RDFS definuje několik typů tříd, z nichž nejdůležitější jsou dále stručně popsané:

- *rdfs:Resource* je definována jako třída všeho. Všechny zdroje jsou její instancí a zároveň všechny třídy jsou jí hierarchicky podřazené.
- *rdfs:Class* je třída všech tříd. Spadají do ní všechny zdroje, které jsou RDF třídou. Z toho vyplývá, že její instancí je i *rdfs:Resource*.
- *rdfs:Literal* je třída všech RDF literálů čili hodnot.
- *rdfs:Datatype* je podobná *rdfs:Class* jen s tím rozdílem, že se jedná o množinu datových typů. Může se jednat o datatypy představující například integer číselné hodnoty, boolean hodnoty nebo znakové hodnoty.

Zdroje představující určité vlastnosti či charakteristiky se nazývají *property*. Všechny definované vlastnosti jsou automaticky instancemi třídy *rdf:Property*. V RDFS existuje několik předdefinovaných typů vlastností a mezi nimi jsou:

- *rdfs:domain* je určen k definici množiny zdrojů čili třídy, které mohou disponovat nějakou vlastností. Definuje nositele vztahu, takže v RDF tripletu o daném vztahu je třída na pozici objektu.
- *rdfs:range* naopak slouží k určení třídy, jejíchž instancí, smí vztah nabývat.
- *rdfs:subClassOf* určuje hierarchický vztah mezi dvěma třídami. Plyne z toho, že všechny instance podřazené třídy jsou zároveň instancemi

třídy nadřazené a nabývají tak i jejích vlastností. Příkladem může být třída `Zvíře` a třída `Pes`. Třída `Pes` je konkrétnější a je tedy podtřídou třídy `Zvíře`.

- *rdfs:subPropertyOf* vyjadřuje hierarchický vztah mezi vlastnostmi. Znamená to, že zdroje mezi kterými je vazba podřazená jsou spojeny i vazbou nadřazenou. Může se jednat například o properties `je_rodičem` a `je_otcem`. Property `je_otcem` je specifičtějším vztahem, než je property `je_rodičem`. Proto jakékoli dva zdroje mezi kterými se nachází vztah `je_otcem` mají mezi sebou i nadřazený vztah `je_rodičem`.
- *rdfs:label* je vlastnost sloužící k popisu zdrojů lidem srozumitelnou formou.
- *rdfs:comment* je určen k dodatečným či zpřesňujícím informacím o zdroji.

RDFS je jednoduchý jazyk, a tak s ním nelze vyjádřit všechny okolnosti a stavy zdrojů a jejich vlastností. Není možné zachytit multiplicity vztahů, tedy určit kolikrát zdroj určitého vztahu smí nabývat. Nelze definovat množiny jako disjunktní či kompletní. A chybí možnost stanovit ekvivalenci tříd či vlastností, nebo určit inverzní poměr mezi vztahy. Prostředky pro přesnější zachycení pak nabízejí pokročilejší jazyky s bohatším slovníkem jako například OWL.

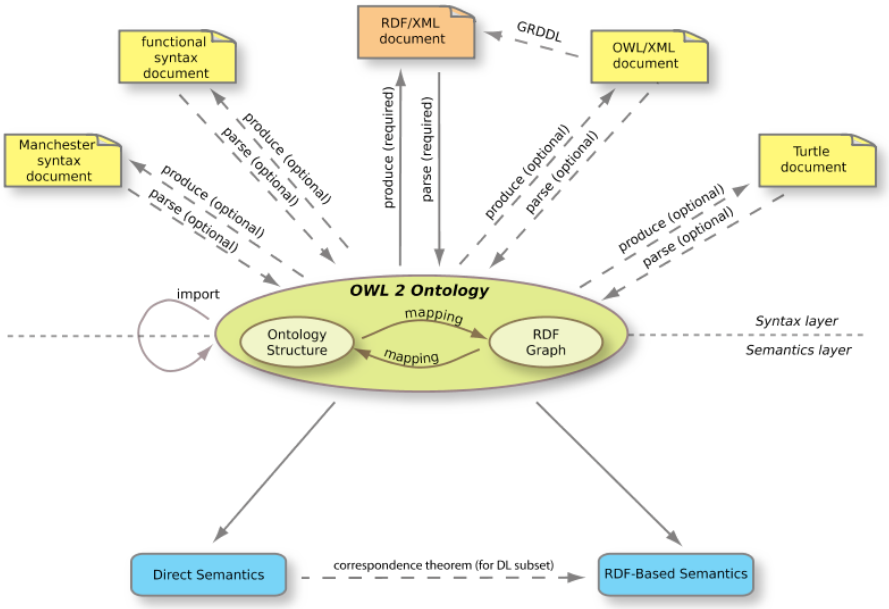
3.4 OWL

Web Ontology Language (OWL) [38] je jazyk, který se využívá k popisu ontologií RDF. *OWL 1* byl vyvinut skupinou *W3C Web Ontology group* v roce 2004, nyní existuje rozšíření *OWL 2*, které spravuje navazující skupina *W3C OWL Working Group* od roku 2009 [39].

OWL [38] je založený na výpočetní logice, díky tomu je strojově čitelný a je tak možné ověřit správnost zápisu nebo odvodit poznatky vedoucí ze zapsaných znalostí.

Dle [39] lze jeho strukturu použití pomyslně rozdělit do tří částí, které jsou spolu provázané, což je zobrazeno na 3.1. Uprostřed je znázorněna ontologie, tedy konstrukty které je možné využít k uspořádání znalostí a pojmů za účelem jejich uchování. Ontologie může být vyjádřena abstraktní strukturou, v jazyce OWL, bez nutnosti syntaxe nebo jako RDF Graf. Tyto dvě zobrazení jsou navzájem převoditelné. Specifikace konstruktů OWL je dále popsána v části 3.4.1.

V horní syntaktické části jsou zobrazeny konkrétní syntaxe, kterými lze ontologii zapsat a primárně slouží k výměně či serializaci znalostí. Nejdůležitějším formátem pro znalosti zachycené v OWL je RDF/XML, popsán v 3.5.1. Tento syntax musí být podporován v každém nástroji pro vývoj OWL ontologií. Ale existují i další formáty, které jsou vhodné pro různá využití, například *Turtle*, blíže popsaný v 3.5.2, *TRiG* nebo *Manchester Syntax*.



Obrázek 3.1: Struktura OWL 2 [40]

V dolní části se nachází sémantické specifikace, které určují význam konstruktů ontologie. Přidaný význam využívají nástroje k ověření správnosti a konzistence ontologie, k odvození znalostí nebo zodpovězení dotazů. [39]

3.4.1 Specifikace OWL

Strukturální specifikace OWL 2 je nezávislá na použití konkrétního formátu pro výměnu znalostí, ale je definována za pomocí jazyka Unified Modeling Language (UML) [41].

Dle OWL 2 specifikace [39, 41] se struktura skládá ze tří kategorií a to entit, výrazů neboli *expressions* a axiomů, které jsou vyloženy dle definované sémantiky a dohromady tak tvoří logický základ ontologie.

OWL rozšiřuje jazyk RDFS a přináší do něj další prostředky pro přesnější vyjádření. To znamená, že jsou k dispozici nejen prostředky a konstrukty definované v OWL, ale také prostředky poskytované samotným RDFS. Specifikace OWL proto popisuje i prostředky z jednoduššího jazyka RDFS.

3.4.1.1 Entity

Entita je základní prvek OWL, který odkazuje na myslitelnou či reálnou věc. Každá entita je identifikována pomocí IRI, ta zajišťuje její jednoznačnost a nezaměnitelnost. Do kategorie entit se řadí třídy, vlastnosti, datové typy a instance nazývané *individuals*. Pokud z ontologie vybereme pouze prvky, které spadají do kategorie entit, pak se jedná o tzv. *znak ontologie*, protože entity

představují nejvýznamnější termíny a koncepty v doméně, kterou ontologie popisuje.

Třída neboli *class* představuje množinu jednotlivců, kteří na základě definice spadají do stejné skupiny. Třídy jsou již definované v RDFS.

Datové typy jsou podobné třídám jen s tím rozdílem, že se jedná o množinu hodnot, ne instance. I ty jsou již definované v RDFS .

Properties obecně vyjadřují vztah a dále se specifikují dle jeho druhu. *Object property* představuje vztah mezi dvěma instancemi, nebo celými třídami. Může se jednat například o vyjádření vztahu manželství mezi mužem a ženou. *Data property* představuje vztah mezi třídou (tedy jejími instancemi) a hodnotou, jako například osoba a její jméno nebo její věk. U všech vlastností se definuje *domain* a *range*.

Annotation properties se využívají pro upřesnění významu ontologie, axiomu nebo IRI identifikátoru. Existují již předdefinované anotace, mezi které patří třeba upřesnění člověkem čitelného označení `rdfs:label`, komentář `rdfs:comment`, nebo zpřesnění definice `rdfs:isDefinedBy`.

Instance neboli individuals reprezentují existující objekty ze zkoumané domény. V OWL existují tzv. *Named Individuals* a *Anonymous Individuals*. *Named Individuals* jsou instance, které je možné globálně určit, protože mají přiřazený identifikátor IRI. Naopak *Anonymous Individuals* mají pouze lokální identifikátory, a tak nejsou globálně jednoznačné. Může se jednat o znalosti, které nejsou potřebné či možné přesněji specifikovat, jako například přesná adresa osoby v případě, že není zjistitelná nebo potřebná pro účel ontologie.

3.4.1.2 Výrazy

Expressions neboli výrazy se v OWL používají k definici podmínek, za kterých je objekt instancí třídy. Výraz pro třídu `Rodič` může být například `"Osoba, která má alespoň jedno dítě"`, pakliže existuje jedinec s naplněným vztahem `"má_dítě"`, pak je instancí dané třídy.

3.4.1.3 Výroky

Axiomy jsou výroky, sloužící ke specifikaci znalostí domény. Říkají co v doméně, a potažmo i v ontologii, je pravda a je tedy validní. OWL poskytuje množinu již předdefinovaných axiomů, které se rozdělují dle toho jaké konstrukty upřesňují.

Class axioms upravují vztahy mezi třídami na které je axiom aplikován. V této skupině existují čtyři definované axiomy: *SubClassOf*, *EquivalentClasses*, *DisjointClasses* a *DisjointUnion*.

Axiom *SubClassOf* říká, že třída `A` je podtřídou třídy `B`, tedy že třída `A` je specifičtější než třída `B`. Příkladem může být třída `Zvíře` a třída `Pes`. Třída `Pes` je konkrétnější a je tedy podtřídou třídy `Zvíře`. Zároveň třída `Zvíře` je nadtřídou, *SuperClassOf*, třídy `Pes`. *EquivalentClasses* axiom značí

sémantickou ekvivalenci mezi třídami. Axiom *DisjointClasses* znamená, že mezi třídami je disjointní vztah, a tedy žádná instance nemůže být instancí těchto tříd zároveň. *DisjointUnion* axiom je vztah pokrytí mezi množinou tříd, která je vzájemně disjunktní a jednou třídou jejíž instance jsou nutně instancí právě jedné třídy z množiny disjunktních tříd. Například třídy `Žena`, `Muž` jsou vzájemně disjunktní, zároveň všechny instance třídy `Osoba` musí být instancí jedné ze tříd `Žena`, nebo `Muž`.

Property axiomy specifikují vztahy mezi vlastnostmi a dělí se na *Object property axiomy*, *Data property axiomy* a *Annotations property axiomy* v závislosti na tom, mezi kterými konstrukty vztah upravují.

Mezi *Object Property axiomy* spadají *SubObjectPropertyOf*, *EquivalentObjectProperties*, *DisjointObjectProperties*, které mají význam užití stejný jako v případě *Class axiomů* jen přenesený na *Object property*. Dále *InverseObjectProperties axiom*, který o dvou *Object properties* říká, že jsou navzájem inverzní. Může jít například o vztah mezi rodičem a dítětem, kdy `"být_rodičem"` je inverzní vztahu `"být_dítětem"`.

Další axiomy jsou *ObjectPropertyDomain* a *ObjectPropertyRange*, které specifikují ze které třídy může být nositel vztahu a z jaké třídy pochází hodnoty vztahu. Například pro vztah `"být_vdaná_za"` je *domain* třída `Manželka` a *range* je třída `Manžel`. Dále existují axiomy specifikující zda je vztah (i)reflexivní, (anti)symetrický, či tranzitivní.

Mezi *Data Property axiomy* patří *SubDataPropertyOf*, *EquivalentDataProperties*, *DisjointDataProperties*, *DataPropertyDomain*, *DataPropertyRange*, jejichž význam je stejný jako v případě *Class* či *Object property* axiomů s přenesením na *Data property*.

Dále existují *Assertions axiomy* neboli tvrzení o instancích kterým se také říká fakta. Patří mezi ně *SameIndividual*, který znázorňuje ekvivalenci instancí. *DifferentIndividuals*, který naopak určuje jejich odlišnost a *ClassAssertion* axiom, který značí, že jednotlivec je instancí dané třídy.

3.5 Formáty RDF

Existuje několik formátů pro zápis a reprezentaci RDF dat, které vznikly z důvodů různých potřeb, preferencí a omezení a liší se způsobem zápisu tripletů. Zde jsou popsány vybrané z nich:

3.5.1 RDF/XML

Jak již bylo zmíněno, RDF/XML [42] je standard W3C, který slouží pro vyjádření RDF za použití formátu XML. XML [43] je jednoduchý textový formát, který byl vybudován *XML Working Group* pod W3C. Jedná se o jazyk primárně využívaný pro uložení a sdílení strukturovaných dat, který je

strojem i člověkem čitelný. XML je složen ze stromové struktury tzv. *elementů*, které představují základní komponenty XML dokumentu. Elementy jsou značkovány pomocí slovních *tags* neboli značek či pojmenování jejichž konstrukt začíná „<“ a končí „>“. Tyto tagy nejsou předdefinované a jejich určení je zcela na autorovi dokumentu. Díky tomu lze XML označit za samopopisný.

K zamezení konfliktů jmen elementů slouží tzv. *Namespaces* [44] neboli jmenné prostory. Často se totiž může stát, že pojmenování jednoho elementu je kolizní s jiným, například při spojení částí různých XML dokumentů. Namespace je možný definovat pomocí atributu `xmlns` v počátečním tagu elementu. Každý element je pak možné přiřadit do jeho unikátního namespace, čímž se i stejně pojmenované elementy odliší a takovýmto konfliktům je předejito.

XML *root element* je v RDF/XML označen jako `rdf:RDF`. Tímto je definováno, že soubor obsahuje RDF data. V `rdf:RDF` jsou definovány i XML namespace. Každý RDF triplet je reprezentován pomocí XML elementu. Subjekt, jak je vidět na ukázce 3.1, je zachycen pomocí `<rdf:Description>`, představující tzv. *node element*. IRI definující subjekt je pak zaznamenaná pomocí jeho atributu `rdf:about`.

Predikáty jsou zachyceny pomocí tzv. *property element*, které jsou subelementy node elementu představující subjekt, což je patrné na 3.1. Pokud má subjekt vícero predikátů, pak je možné zachytit tuto skutečnost zkrácenou verzí zápisu tak, že *node element* má více *property element* jako své subelementy. K zachycení IRI objektu predikátu se využívá `rdf:resource`.

Ukázka kódu 3.1: Příklad RDF/XML

```
<rdf:Description
↪   rdf:about="http://example.com/persons/john-Smith">
    <name>John Smith</name>
    <age>35</age>
  </rdf:Description>
```

V případě zachycení *blank node* se užívá `rdf:nodeID` pro identifikaci v rámci dokumentu. Užívá se jak na místě `rdf:about` pro *node element* tak `rdf:resource` pro *property element*.

Pro vyjádření datového typu objektu, který je literál, se využívá atribut elementu `rdf:datatype` s URI vybraného datatypu.

3.5.2 Turtle

Terse RDF Triple Langugage (Turtle) [45] je další možný syntax pro reprezentaci RDF dat. Jedná se o jednoduše čitelný a pochopitelný zápis, který

je obohacen o několik funkčních výhod, díky kterým lze zachytit data expresivněji.

Primitivní trojice (subjekt, predikát, objekt) se zapisuje v jednom řádku. V případě, že jeden subjekt disponuje vícero predikáty, je možné zápis stále stejného subjektu vynechat, což je vidět na 3.2. Tomuto typu tvrzení se říká *Predicate List*.

V případě, že jeden predikát disponuje vícero objekty, tak jako subjekt disponoval vícero predikáty, se dané tvrzení nazývá *Object List*. Jedná se o trojici, kde subjekt i predikát jsou totožné jako na předešlé řádce, což je vidět na ukázce 3.2 v případě predikátu `:knows`.

Pro zápis RDF termů umožňuje Turtle něolik různých způsobů. IRI je možné zapsat jako relativní, absolutní či pomocí prefixu. Relativní i absolutní IRI je vložena do špičatých závorek. Relativní IRI, na rozdíl od absolutní, je vztažena k tzv. *base* IRI, která musí být v dokumentu definována. Zápis IRI pomocí prefixového jména je složen z prefixu a lokální části, oddělenými dvojtečkou „:“. Prefixové jméno je potřeba na začátku dokumentu definovat pomocí anotace `@prefix` spolu se zvoleným jménem a IRI, kterou bude reprezentovat. Díky využití prefixů je dokument čitelnější, přehlednější a je jednoduší na úpravu.

Literály se v Turtle, což je vidět na ukázce 3.2, zapisují do uvozovek jako textové řetězce a lze u nich určit jejich datový typ či jazyk, pokud to u daného literálu má smysl. Jazyk je případně označen za literálem pomocí „@“ a zkratkou jazyka. Datový typ je označen také za literálem, použije se oddělovač „^^“, za kterým se definuje IRI datového typu. V případě, že datový typ není definován jedná se o textový řetězec.

Ukázka kódu 3.2: Příklad Turtle

```
<http://example.com/persons/john-Smith>
    :name   "John Smith"@en ;
    :age    "35"^^xsd:integer ;
    :knows <http://example.com/persons/jane-smith> ,
    ↪  <http://example.com/persons/mark-johnson> .
```

Blank nodes neboli prázdné uzly se v Turtle zapisují pomocí hranatých závorek s vynecháním identifikace.

3.5.3 N-Triples

N-Triples [46] je jednoduchý formát pro zápis RDF dat. Jednotlivé trojice jsou zapsány na odděleném řádku zakončeným tečkou a termy jsou odděleny mezerami, což je viditelné na 3.3.

IRI jsou v tomto formátu zapisovány do špičatých závorek. Literály jsou zapisovány stejně tak jako ve formátu Turtle a to do horních uvozovek a lze u nich definovat jejich datový typ a případně jazyk.

Ukázka kódu 3.3: Příklad N-Triples

```
<http://example.com/persons/john-smith>
↪  <http://example.com/ontology/name> "John Smith" .
<http://example.com/persons/john-smith>
↪  <http://example.com/ontology/age> "30"^^xsd:integer .
```

Blank nodes, neboli prázdné uzly, se vyjadřují pomocí prefixu „_:“ před znakovým řetězcem sloužícím pro označení subjektu. Ačkoli prefixy, takové jako využívá Turtle, v *N-Triples* neexistují.

3.6 Nástroje pro tvorbu ontologií

Při návrhu i tvorbě ontologie byly využity známé nástroje, primárně po kontrolu správnosti obsahu i validaci zápisu.

3.6.1 Protégé

Protégé [47] je open-source nástroj, který slouží pro vývoj a správu ontologií. Jedná se o nástroj rozšiřitelný pomocí přídavných modulů, které umožňují různé vizualizace, import ontologií v různých formátech z webového i z lokálního zdroje a export do různých formátů. Vývojáři mohou využít API za účelem propojení s programy s cílem využití ontologií a manipulace s nimi. V případě importu ontologie nástroj dokáže validovat vstup a v případě nepřesností odhalí u jednotlivých tříd a vlastností chyby v jejich definici. Existuje i několik přídavných modulů pro validaci ontologie. Jedním z nich je například *OntoDebug* [48], který odhaluje nekonzistence.

3.6.2 IDLab Turtle Validator

IDLab Turtle Validator [49] je jednoduchý nástroj pro validaci RDF dat zapsaných ve formátu Turtle nebo *NTriples*.

3.6.3 Ontology Pitfall Scanner

Ontology Pitfall Scanner (OOPS!) [50] je nástroj pro detekci chyb v navržených ontologiích. Je založen na seznamu známých charakteristik, které často představují problém vedoucí k chybě v ontologii. Tento výčet potenciálních chyb je

založen jak na výzkumech zabývajících se validací ontologií, tak na manuální kontrole několika desítek ontologií.

Do on-line nástroje je možné nahrát ontologii ve formátu RDF/XML, která je následně validována. Všechny nalezené chyby jsou označeny důležitostí, zároveň ne všechny nalezené chyby musí nutně znamenat opravdovou chybu v ontologii.

3.6.4 Shape Expressions Validata

Shape Expressions (ShEx) [51] je jazyk pro popis a validaci RDF datových struktur. Slouží ke specifikaci, jak by měla vypadat data a jaká pravidla by měla dodržovat. Díky tomu, je možné popsat očekávanou strukturu dat a tím data validovat. Nástroj *ShEx Validata* [52] představuje validátor RDF dat na základě předdefinovaných schémat v jazyce ShEx.

3.6.5 Nástroje pro tvorbu dokumentace ontologií

Pro vytvoření strukturované dokumentace ontologie existuje několik nástrojů. Zde jsou představeny tři z nich:

3.6.5.1 Live OWL Documentation Environment

Live OWL Documentation Environment (LODE) [53] je služba, která automaticky extrahuje podstatné části ontologie v jazyce OWL a generuje interaktivní dokumentaci v Hyper Text Markup Language (HTML) formátu. Výsledná dokumentace zahrnuje informace o třídách, vlastnostech, axiomech a dalších prvcích ontologie. Interaktivní výsledek usnadňuje porozumění a navigaci.

3.6.5.2 Wizard for Documenting Ontologies

Wizard for Documenting Ontologies (WIDOCO) [54] je nástroj, který generuje dokumentaci ontologie. Využívá automatické extrakce klíčových částí ontologie na základě technologie LODE. WIDOCO poskytuje pokročilé funkce, včetně validace ontologie pomocí nástroje OOPS! a možnosti přidání popisu licence ontologie.

3.6.5.3 OntoDoc

OntoDoc [55] je nástroj pro generování statické dokumentace. Využívá anotací tříd a vztahů, jejich vzájemné propojení i jejich typy.

3.7 Jinja2

Jinja2 [23] je jazyk pro tvorbu šablon se související knihovnou jazyka Python. Jeho název je odvozen od japonského slova *Jinja* čili chrám, který v angličtině zní podobně jako anglický výraz *template*.

Jinja2 je charakterizován svou jednoduchostí, expresivitou a rozšiřitelností. Tento nástroj umožňuje definovat šablonu výsledného dokumentu a po přidání dat je vyobrazena kompletní a finální verze dokumentu. Nejčastěji je využíván při tvorbě dynamických webových stránek s daty z objektů v jazyce Python.

Syntaxe obsahuje několik základních konstrukcí. Vypsání hodnoty proměnné se provádí pomocí dvojitých složených závorek, do kterých se vloží název proměnné, jako je uvedeno v příkladu 3.4 pro proměnou `user`.

Řídící struktury neboli *tagy* slouží pro kontrolu toku dat ve výsledném dokumentu. Jedná se o konstrukty představující `if-else` podmínky, cykly `for` a makra. Tyto konstrukce jsou zapisovány do složených závorek specifikovaných pomocí procenta. Díky těmto řídícím strukturám je možné určit podmínky vypsání dat, jejich procházení či definovat opakovatelně použitelné funkce. Na ukázce 3.4 je znázorněn zápis makra, uvnitř kterého je `if` podmínka i cyklus `for`.

Ukázka kódu 3.4: Příklad Jinja2

```
{#- makro vypisující všechny uživatele se jménem Jan -#}
{%- macro renderUserJan(users) -%}
  {%- for user in users %}
    {%- if user.firstName == "Jan"  -%}
        {{user}}
    {%- endif -%}
  {%- endfor -%}
{%- endmacro -%}
```

Při práci s tagy je důležité brát v potaz kontrolu bílých znaků. Výchozí nastavení zaručuje odstranění nového řádku a ponechání všech ostatních bílých znaků. V konfiguraci Jinja2 lze nastavit *trim_blocks*, který odstraňuje první nový řádek po definici tagu a *lstrip_blocks*, který zamezuje výskyt všech bílých znaků od začátku řádku až k definici tagu. Obě nastavení lze manuálně zakázat a to pomocí vložení znaku „+“ před tag v konstruktu v případě *lstrip_blocks*, nebo za tag v případě *trim_blocks*.

Pokud tyto nastavení nejsou v globální konfiguraci, je možné je definovat manuálně u jednotlivých konstruktů řídících struktur pomocí znaku „-“ jak je znázorněno na ukázce 3.4. Komentáře v Jinja2 se zapisují do složených závorek doplněných o znak křížek „#“, což je patrné z ukázky 3.4.

Kapitola 4

Návrh ontologie struktur DSW

V této kapitole jsou popsány jednotlivé aspekty návrhu ontologie pro nástroj DSW i výsledná ontologie. Při návrhu ontologie struktur nástroje DSW bylo primárně vycházeno z analýzy nástroje a také z výsledného dokumentu exportní šablony *Questionnaire Report* [7] ve formátu JavaScript Object Notation (JSON). Tato exportní šablona generuje data strukturovaná, tak jak jsou uložena. Díky tomu je zřejmá přesná struktura uložení dat pro jakýkoli vytvořený projekt.

Celý návrh ontologie v prvotní fázi probíhal v grafické podobě, kdy pomocí vytvořené notace byla celá ontologie reprezentována Class diagramem. Jeho notace je popsána v části 4.6. Model ontologie vznikl úpravou analytického modelu, který byl vytvořen během analýzy nástroje DSW 2.2. Nejdříve byl ontologický model značně upraven za účelem výběru důležitých částí pro tvorbu ontologie. Dále byl sjednocen způsob pojmenování jak tříd, tak i jejich vlastností. Poté byly určeny vlastnosti, které mají jednotlivé třídy společné a bylo vytvořeno několik abstraktních tříd za účelem zamezení redundance. Následně bylo přidáno několik inverzí k již definovaným vlastnostem, které doplňují celkový pohled na data. Po vytvoření ustáleného návrhu ontologie v podobě modelu bylo vzato v úvahu propojení s již existujícími ontologiemi, které zapříčinilo několik nutných změn. Nakonec byla dle modelu vytvořena výsledná ontologie.

Pro označení ontologie nástroje DSW je využívána i preferována zkratka `dsw` jako prefix. Ve výsledné ontologii tento prefix slouží k jednoznačné identifikaci tříd a vlastností v rámci ontologie.

Ontologii tohoto nástroje je možné navrhnout z několika různých pohledů v závislosti na chtěném výsledku. Tato ontologie byla navrhována z pohledu výsledného dokumentu, obsahujícího transformovaná data. Dalším možným přístupem by mohlo být zaměření z pohledu dotazníku a jím využívaných částí spolu se zahrnutím příslušnosti odpovědí k dotazníku s existující ale nedefinující vazbou na související otázku.

4.1 Navržené změny pro ontologii struktur DSW

Při tvorbě návrhu ontologie bylo potřeba vybrat důležité entity a atributy DSW, které budou součástí ontologie. Struktury DSW se skládají i z několika technických a pomocných entit či atributů, které nejsou pro účel zachycení znalostí relevantní. Jejich zařazení by způsobilo rozšíření rozsahu zaměření ontologie o nevýznamné části, což by ji učinilo složitější a méně pochopitelnou. Technické a pomocné aspekty navíc nepřináší pro ontologii důležité informace a zároveň se často jedná o nejednoznačná data s vícero možnými interpretacemi.

Při návrhu byly některé entity i vlastnosti vynechány či přesunuty za účelem dosažení konzistentní a jasné ontologie. Většinou se jedná o struktury, které jsou v částech kontextu dotazníku nebo jeho konkrétní instance. Dále jsou vypsány změny a jejich odůvodnění.

4.1.1 Navržené změny ve strukturách znalostního modelu

Struktury znalostního modelu hrají klíčovou roli v ontologii DSW. Jak plyne z provedené analýzy, tyto struktury byly navrženy již s ohledem na ontologické zásady, a tak nebylo nutné provádět významné změny při návrhu ontologie. Ve strukturách znalostního modelu se vyskytuje značné množství redundantních vztahů, které byly abstrahovány, což je popsáno v 4.3.

Pro uspořádání prvků znalostního modelu je klíčová struktura uspořádání prvků ve smyslu rodičovského vztahu a zároveň pořadí, v jakém se potomci vyskytují v rámci svých rodičovských prvků. Například pořadí kapitol v rámci znalostního modelu, pořadí otázek v rámci kapitoly, či podotázek v rodičovské otázce. Díky této znalosti lze uspořádat posloupnost prvků v dotazníku a rekonstruovat jeho průběh.

Každá entita, která má vztah k jiné entitě prostřednictvím rodičovské vazby, byla obohacena o vlastnost `dsw:order`. Tato vlastnost určuje pořadí instance nositele v rámci rodičovské entity a umožňuje tedy správné uspořádání prvků v ontologii.

Zároveň vztah *hasFollowup* ať už od otázky typu *ListItemQuestion* nebo od odpovědi *Answer* byl vyhodnocen jako vztah mezi rodičem a potomkem, a byl jím v návrhu ontologie i nahrazen.

Pro vlastnosti `dsw:annotation` a `dsw:props` bylo nutné vytvořit třídu reprezentující datový typ. Data, která odpovídají těmto vlastnostem, jsou uložena v datové struktuře *dictionary* neboli slovníku a tomu neodpovídá žádný doporučený datový typ. Z tohoto důvodu byla vytvořena další třída `dsw:KeyValueEntry` s vlastnostmi `dsw:key` a `dsw:value`.

4.1.2 Navržené změny ve strukturách doplňujících informací

Doplňující informace obsahují nezbytné znalosti pro tvorbu plnohodnotného DMP dokumentu. Při návrhu ontologie se vyskytly prvky, které nebyly ontologicky přesné nebo způsobovaly redundanci. Tyto prvky byly buď přepracovány nebo úplně vynechány, důsledkem čehož bylo dosaženo ontologicky přesnějšího a jasnějšího návrhu.

4.1.2.1 Report Indication

Entita *Report Indication* disponuje několika pomocnými atributy, které jsou dopočítané z ostatních atributů této entity. Konkrétně atribut *total* představuje celkový počet odpovědí a vypočítává se jako součet hodnot v atributech *answered* a *unanswered*. Atribut *percentage*, který vyjadřuje procentuální poměr zodpovězených odpovědí, lze vypočítat jako podíl hodnoty atributu *answered* a hodnoty atributu *total*. Atributy *isForPhase* a *isOverall* označují, zda se jedná o indikaci ze všech odpovědí dotazníku nebo pouze těch, které jsou povinné pro danou fázi dotazování. Tuto skutečnost lze ale vyvodit i z atributu *indicationType*, a tak byly atributy *isForPhase* a *isOverall* vynechány.

4.1.2.2 Report Item

Entita *Report Item* představuje jak reporty celého dotazníku, tak i jeho kapitol a to se odlišuje pomocí hodnoty atributu *chapter*. V případě, že je atribut vyplněn, jedná se o report určené kapitoly a pokud není vyplněn jedná se o report celého dotazníku. Z tohoto důvodu byla entita zachována jako ontologická abstraktní třída `dsw:ReportItem` a byly vytvořeny dvě podtřídy `dsw:ChapterReportItem` a `dsw:TotalReportItem`, které představují konkrétní typy reportu.

4.1.2.3 Document Context, Document

Veškerá data pro tvorbu výsledného DMP dokumentu jsou obsažena v kořenovém objektu *Document Context*. Tento objekt odkazuje na další entity jako například znalostní model, dotazník nebo nastavení organizace, ale samotný objekt nepřináší žádné nové informace. Kromě toho existuje entita *Document*, která reprezentuje výsledný exportovaný dokument a obsahuje všechny části výsledného dokumentu stejně jako entita *Document Context*. Z tohoto důvodu byly entity *Document Context* a *Document* spojeny do jedné ontologické třídy `dsw:Document`. Ta je nositelkou *properties*, které odkazují na jednotlivé části výsledného dokumentu tak jako *Document Context*. Třída `dsw:Document` je dále charakterizována okamžikem vytvoření či aktualizací dokumentu jako byl objekt *Document*.

Jediný atribut, který je z *Document Context* vynechán a nepřenesen do *Document* je atribut *currentPhase*, který obsahuje stejnou hodnotu jako atribut entity *Phase*.

4.1.2.4 Replies Container, Knowledge Model Entities

Pomocné struktury *Replies Container* a *Knowledge Model Entites* slouží pouze k přístupu k jednotlivým odpovědím, respektive ke složkám znalostního modelu. Tyto entity obsahují sice potřebné konkrétní odpovědi a komponenty znalostního modelu, ale zachycení pomocných entit sloužících pro přístup není pro účel ontologie relevantní.

4.1.2.5 Package

Entita *Package* slouží k seskupení metadat souvisejících se znalostním modelem. Z ontologického hlediska tato metadata patří ke třídě `dsw:KnowledgeModel`, která představuje samotný znalostní model. K `dsw:KnowledgeModel` byly přesunuty některé atributy, konkrétně `kmId`, `version`, `name`, `description` a `createdAt`. Atribut `organizationId` byl vynechán, neboť jeho hodnota se shoduje s atributem `id` entity *Organization*, a tudíž nepřináší žádnou novou informaci.

Identifikátor `id` entity *Package* se skládá z identifikátorů organizace, znalostního modelu a verze tohoto modelu. Představuje pomocnou dopočítanou hodnotu, která nerozšiřuje znalosti, a tak byl atribut vynechán.

Atribut `versions` v entitě *Package* reprezentuje seznam všech existujících verzí daného znalostního modelu. Nicméně z ontologického hlediska není tento výčet klíčkový, jelikož neodkazuje na konkrétní instanci verze znalostního modelu, ale definuje pouze její číselné označení.

4.1.3 Navržené změny ve strukturách instance dotazníku

Mezi prvky, které patří do skupiny struktur instancí dotazníků, se primárně řadí dotazníky a konkrétní odpovědi. V této skupině nebyly úmyslně prováděny významné úpravy na třídách `dsw:Questionnaire` (dotazník) nebo `dsw:QuestionnaireVersion` (verze dotazníku). V důsledku změn provedených v propojených třídách došlo ke drobným úpravám. Tyto druhotné změny nejsou v rámci práce popsány. Výjimkou jsou entity *Simple Author* a *User*, které byly spojeny a tato změna je podrobně popsána v této části.

V případě třídy *Reply* (odpověď) byly provedeny změny tak, aby se jejich vlastnosti co nejvíce zjednodušily a zároveň dostatečně dobře vypovídaly o jejich obsahu.

4.1.3.1 Replies

Z analýzy vyplývá, že odpovědi (*Replies*) se rozlišují v závislosti na typu otázky, na kterou odpovídají. Proto i v návrhu ontologie vznikly konkrétní typy odpovědí a to: `dsw:ValueReply`, `dsw:ItemListReply`, `dsw:IntegrationReply`, `dsw:MultiChoiceReply` a `dsw:AnswerReply`. Díky vzniku konkrétních tříd typů odpovědí mohl být vynechán atribut *replyType* udávající typ odpovědi. Do abstraktní třídy `dsw:Reply`, která je nadtřídou jednotlivých typů odpovědí, byla přidána vlastnost `dsw:replyValue`. Ta představuje hodnotu odpovědi ve formě textového řetězce a přesná hodnota záleží na typu odpovědi.

4.1.3.2 ValueReply

Třída `dsw:ValueReply` představuje původní *StringReply*. Prvním důvodem k přejmenování byla skutečnost, že se jedná o odpověď na otázku typu *ValueQuestion*. Druhým důvodem bylo to, že původní *StringReply* může mít jako hodnotu odpovědi různé datové typy včetně typu *string*. Původní název tak vedl k zavádějícímu předpokladu, že hodnota odpovědi je pouze typu *string*. Ve skutečnosti ale může být hodnota odpovědi také číslo, datum, čas, nebo URL adresa. Proto bylo navrženo několik nových vlastností pro reprezentaci příslušných datových typů odpovědí a to `dsw:urlReplyValue`, `dsw:numberReplyValue`, `dsw:dateReplyValue`, `dsw:dateReplyValue`, `dsw:dateTimeReplyValue`, `dsw:timeReplyValue`. Tyto vlastnosti jsou nepovinné a k jejich vyplnění dojde jen v případě, že odpověď je daného typu.

Pro odpověď typu *string* nebyla vytvořena žádná nová vlastnost, jelikož vlastnost `dsw:replyValue` nadtřídy `dsw:Reply` již obsahuje hodnoty odpovědí v datovém typu *string*. Pokud by byla vytvořena nová vlastnost pro odpovědi typu *string*, došlo by k redundantnímu vyjádření hodnot. Proto jsou odpovědi s hodnotami typu *String*, *Text*, *Email* a *Color* reprezentovány pouze s pomocí `dsw:replyValue`.

4.1.3.3 IntegrationReply

Z datového souboru je patrné, že *IntegrationReply* se dále dělí na základě toho, zda obsahuje odkaz na existující prvek v externím zdroji. V případě otázek typu *Integration* může být odpověď nalezena v externím zdroji a v takovém případě je vedle textové odpovědi uveden i odkaz. Pokud externí zdroj neobsahuje požadovanou odpověď, je odpověď uložena pouze v textové formě.

Původně se tento rozdíl v reprezentaci odpovědí typu *Integration* zaznamenával pomocí atributů *isPlain* a *isIntegration*, což však vedlo k nedostatečně přesnému ontologickému vyjádření, neboť odpověď typu *integration* musí mít validní odkaz na položku externího zdroje, ale odpověď typu *plain* takový odkaz mít nemusí.

Tento rozdíl v reprezentaci odpovědí typu *Integration* je zajištěn pomocí ontologické nadtřídy `dsw:IntegrationReply`, která je nadřazena dvěma třídám: `dsw:IntegrationTypeIntegrationReply` představující odpověď s validním odkazem a `dsw:PlainTypeIntegrationReply` reprezentující odpovědi pouze v textové formě.

Třída `dsw:IntegrationTypeIntegrationReply` obsahuje vlastnost `dsw:integrationReplyValue` s validním URL odkazem, který je poskládán z využité integrace a identifikátoru položky externího zdroje. Dále obsahuje vlastnost `dsw:integrationReplyItem` obsahující identifikátor konkrétní položky externího zdroje.

4.1.3.4 MultiChoiceReply, AnswerReply

Jak odpověď typu *MultiChoiceReply*, tak *AnswerReply* ve vlastnosti `dsw:replyValue` obsahují identifikátory zvolených odpovědí či voleb ve formě textového řetězce. U odpovědi typu *MultiChoiceReply* jde o zvolenou instanci třídy `dsw:Choice`. Obdobně u odpovědi typu *AnswerReply* se jedná o zvolenou instanci třídy `dsw:Answer`.

Pro přímé propojení odpovědi s konkrétní volbou či odpovědí byly vytvořeny příslušné vlastnosti. U odpovědi *MultiChoiceReply* se jedná o vlastnost `dsw:multiChoiceReplyValue`, která propojuje odpověď s konkrétní volbou třídy `dsw:Choice`. U odpovědi typu *AnswerReply* se pak jedná o vlastnost `dsw:answerReplyValue`, která propojuje odpověď s konkrétní odpovědí třídy `dsw:Answer`.

4.1.3.5 Simple Author, User

Z datového souboru je zřejmé, že entity *Simple Author* a *User*, reprezentující uživatele, využívají stejná identifikační čísla `uuid` pro stejné uživatele. Kromě toho jsou atributy entity *Simple Author* podmnožinou atributů entity *User*, s výjimkou atributu `gravatarHash`, který slouží pro implementaci a byl proto vynechán. V důsledku toho byly tyto entity sloučeny do jedné třídy `dsw:User` s nepovinnými atributy, které patřili k původní entitě *User*. Díky této konsolidaci je reprezentace uživatelů v DSW přehlednější. Eliminovala se redundantní reprezentace totožné znalosti a zároveň se zvýšila úroveň abstrakce a sémantická přesnost.

4.2 Názvy tříd a vlastností

Při návrhu ontologie bylo nutné zvolit konzistentní, dostatečně popisné a přesné názvy, jak pro třídy tak pro vlastnosti. Pojmenování tříd bylo voleno primárně podle názvů struktur DSW tak, aby co nejvíce odpovídalo skutečnému objektu a bylo jednoznačné. Při nazývaní vlastností bylo důležité

vystihnout dostatečně přesně vztah či atribut, který představují. Názvy v ontologiích by měly podléhat jmenným konvencím, aby se docílilo čitelnosti ontologie i její znovupoužitelnosti.

4.2.1 Zvolené notace

Pro názvy tříd byla zvolena tzv. *PascalCase* notace. Jelikož je jednoduchá ke čtení, jednoznačná a zároveň je to nejčastěji užívaná notace při pojmenování ontologických tříd.

Další možná často využívaná notace je tzv. *snake_case* neboli hadí notace. Ta ale může být nechtěně zaměněna za název vlastnosti, protože začíná malým písmenem a navíc, díky podtržítkům oddělující slova, vznikají delší názvy.

Pro názvy vlastností byla zvolena velbloudí notace *camelCase*, která disponuje malým počátečním písmenem prvního slova. Malé počáteční písmeno v názvu vlastnosti je zavedené pravidlo, díky kterému je snadněji rozeznatelné zda se jedná o název třídy nebo vlastnosti.

V názvech vlastností je zvykem užívat mimo podstatných jmen i slovesa, přídavná jména či předložky ke zpřesnění názvu. V každém případě by se mělo jednat o výraz v přítomném čase (pokud obsahuje sloveso) a v jednotném čísle.

Při návrhu ontologie bylo cílem vztah pomocí názvu vyjádřit jasně a zároveň stručně. Proto byly formulovány *Data properties* pouze z podstatného jména a *Object properties* doplněny o sloveso. *Data properties* vyjadřují atributy charakterizující entitu, které lze vyjádřit pomocí podstatných či přídavných jmen. Naopak *Object properties* představují vztahy mezi dvěma entitami, a tak je díky slovesu možné zpřesnit povahu tohoto vztahu.

V návrhu například existuje třída `dsw:Answer` reprezentující předdefinované odpovědi na otázky typu *OptionsQuestion*. Existují dva různé vztahy mající instanci `dsw:Answer` na straně objektu. Prvním je vyjádření obsažení odpovědi ve znalostním modelu, které je formulováno pomocí `dsw:containsAnswer`. Druhým je vztah užití v nabídce odpovědí otázky *OptionsQuestion*, ten je definován jako `dsw:usesAnswer`.

Dalším příkladem užitím sloves k rozlišení nuancí mezi vztahy může být vyjádření souvislosti s instancemi třídy `dsw:Phase`. Stejně tak jako v případě `dsw:Answer` se jedná o obsah znalostního modelu a je tak potřeba `dsw:containsPhase`. Dále se u dotazníku `dsw:Questionnaire` určuje momentální fáze pomocí `dsw:hasPhase`. A nakonec je u otázek potřeba uvést v jaké fázi (pokud vůbec) je otázka povinná. Tato souvislost se značí pomocí vlastnosti `dsw:isDesiredAtPhase`.

Díky slovesům je tak možné rozlišit zdánlivě podobné vztahy a zároveň ujasnit směr i význam vztahu. Přidanou hodnotou jsou i na první pohled rozeznatelné typy vlastností.

V návrhu ontologie bylo například potřeba odlišit i vyjádření verze určité entity DSW. Třída `dsw:KnowledgeModel` neboli znalostní model je charakterizován mimo jiné i verzí, která je uložena v datové podobě a jedná se tak pouze

o číslo verze například `"4.7.1"`. Naopak `dsw:Questionnaire` neboli dotazník je nositelem propojení s konkrétní verzí dotazníku `dsw:QuestionnaireVersion`, která je momentálně využívána a zároveň všemi existujícími verzemi. Tyto vztahy by mohly být vyjádřeny jednoduše jen pomocí názvu `"version"`, ale tím by se smazala rozdílnost mezi nimi. Proto byly navrhnuty vlastnosti `dsw:version` pro reprezentaci verze znalostního modelu, `dsw:hasCurrentVersion`, která představuje momentálně užívanou verzi dotazníku a `dsw:hasVersion` pro všechny existující verze dotazníku.

Výjimkou použití sloves v návrhu ontologie jsou *Object properties* u jednotlivých typů odpovědí, například `dsw:multiChoiceReplyValue` pro `dsw:MultiChoiceReply`, která odpovědi propojuje s `dsw:Choice`. I bez užití slovesa se totiž jedná o jasný význam vztahu, který je navíc podpořen tím, že jeho subjektem je konkrétní typ odpovědi. Přidání slovesa by navíc prodloužilo už tak dlouhý název vztahu.

4.2.2 Sjednocení názvů vlastností

Z provedené analýzy vyplývá existence několika synonym pro pojmenování stejných vlastností prvků DSW. Pokud by se výskyt synonym nezredukoval, mohl by tento fakt vést k nejednoznačnosti a nepřesnosti vyjádření v ontologii.

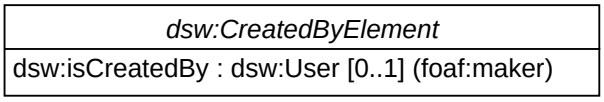
Konkrétně jsou to synonyma pro atributy představující *název* a *popis* prvků. Atribut představující *název* má, dle datového souboru, několik pojmenování, jako jsou *label*, *title* a *name*. Například entita *Chapter*, nebo *Question* používá pojmenování *title*, zatímco *Choice* nebo *Answer* používá pojmenování *label* a *Integration* nebo *Tag* používají pojmenování *name*. Z toho plyne, že použití pojmenování nepodléhá pravidlům výskytu v elementech, což může vést k nejasnostem. Proto bylo navrženo sjednocení pojmenování vlastnosti `dsw:title`.

Atribut představující *popis* má dle datového souboru také několik pojmenování a to *text* a *description*. Například entita *Chapter* používá pojmenování *text*, zatímco *Phase* nebo *Metric* používají pojmenování *description*. Proto bylo navrženo sjednocení pojmenování na `dsw:description`, aby se zajistila jednoznačnost a přesnost vyjádření v ontologii.

4.3 Popis vzniku abstraktních tříd

K vyjádření vztahů a vlastností, kterými disponuje vícero různých tříd, bylo využito abstraktních tříd. Díky nim je zajištěna jednoznačnost a konzistence často vyskytujících se vztahů, Tím, že se definice vyskytuje pouze jedenkrát, je možné ontologii jednodušeji upravovat a udržovat. Pokud třída nabývá dané vlastnosti, pak je definovaná jako podtřída abstraktní třídy představující vlastnost.

Abstraktní třídy jsou v návrhu ontologie vyobrazeny s názvem vztahu či vlastnosti, které představují, doplněné o *Element*. Jako vlastnost abstraktní



Obrázek 4.1: Abstraktní třída *CreatedByElement*

třídy je pak ta daná vlastnost, kterou abstraktní třída reprezentuje, což je vidět na 4.1.

Mezi vlastnosti, které byly vyjádřeny pomocí abstraktní třídy patří: `uuid`, `title`, `hasParent`, `order`, `annotation`, `createdAt`, `updatedAt`, `isCreatedBy`, `affiliation`, `email`, `hasMetric`, `prop` a `description`.

Pro zjednodušení byla mezi abstraktními třídami vytvořena hierarchie podle výskytu vlastností ve třídách představující struktury DSW. V 4.1 jsou zachycené vlastnosti a třídy ve kterých se vyskytují. Všechny názvy tříd i vlastností jsou uvedeny bez prefixu `dsw`. Barevně jsou označeny ty, které byly při návrhu ontologie propojeny či spojeny.

Mezi abstraktními třídami reprezentujícími vlastnosti `dsw:uuid` a `dsw:title` byl vytvořen vztah dědičnosti. Třída `dsw:UuidElement` jako nadtřída `dsw:TitleElement`, jelikož všechny třídy nabývající vlastností `dsw:title` nabývají taktéž `dsw:uuid`.

Vlastnosti `dsw:hasParent` a `dsw:order` jsou reprezentovány jednou abstraktní třídou, jelikož se ve třídách vždy vyskytují současně. Vztah `dsw:hasParent` představuje vazbu v rámci znalostního modelu mezi elementem představujícím potomka a elementem představujícím rodiče. Vlastnost `dsw:hasParent` byla přidána jako inverzní k vlastnosti `dsw:containsChild`. Ta představuje vazbu od rodiče k potomkovi, tedy z druhé strany. Všechny třídy dědící vlastnost `dsw:containsChild` ji dále specifikují podle toho k jakým třídám se ve smyslu tohoto vztahu vážou. Například třída `dsw:KnowledgeModel` má několik těchto vlastností, jelikož znalostní model má několik potomků. Z tohoto důvodu existují ve třídě `dsw:KnowledgeModel` vlastnosti jako `dsw:containsChapter` nebo `dsw:containsMetric`.

Order neboli pořadí je pak umístění dané entity potomka v množině u rodičovského elementu. Abstraktní třída představující vlastnosti `dsw:hasParent` a `dsw:order` se nazývá `dsw:ChildElement`, jelikož jsou to entity potomků, které nabývají těchto vlastností. Naopak abstraktní třída vlastnosti `dsw:containsChild` má název `dsw:ParentElement`. Tyto dvě abstraktní třídy tak porušují pravidlo složení názvů a nejmenují se dle vlastností které představují, ale podle určení role v rámci vztahu mezi rodičem a potomkem.

Vlastnost `dsw:annotation` se vyskytuje také u elementů znalostního modelu, ale navíc se může vyskytovat i u znalostního modelu samotného. Z tohoto důvodu abstraktní třída představující vlastnost `dsw:annotation` je nadtřídou `dsw:ChildElement`. Protože je `dsw:annotation` ve všech výskytech nepo-

vinná, je název nositelské abstraktní třídy v podmiňovacím způsobu `dsw:AnnotatableElement`.

Mezi abstraktními třídami vlastností `dsw:createdAt` a `dsw:updatedAt` je vztah dědičnosti, jelikož všechny třídy nabývající `dsw:updatedAt` nabývají i `dsw:createdAt`. Obráceně tomu tak není, protože u některých entit není uvedení času aktualizace potřebné.

Zbylé vytvořené abstraktní třídy mezi sebou nemají žádný vztah dědičnosti, jelikož nebylo možné vytvořit hierarchii tak, aby byl výskyt vlastností a vztahů reprezentovaných abstraktními třídami v entitách dodržen.

Tak jako `dsw:AnnotatableElement` i ostatní abstraktní třídy obsahující nepovinné vlastnosti mají název v podmiňovacím způsobu. Jedná se o třídy s vlastnostmi `dsw:affiliation`, `dsw:prop`, `dsw:description` a `dsw:email`.

Dále byla zvažována abstraktní nadtřída pro všechny identifikátory, do kterých by spadala jak abstraktní třída pro `dsw:uuid` tak vlastnosti představující identifikátory jako jsou například `dsw:knowledgeModelId`, `dsw:shortUuid` nebo třeba `dsw:integrationId`. Od tohoto návrhu bylo upuštěno, jelikož vlastnosti reprezentující speciální identifikátory přímo neodpovídají identifikaci popisovaného zdroje. Například vlastnost `dsw:shortUuid` odkazuje na jiný zdroj a nejedná se tak o identifikaci popisovaného zdroje. V případě vlastnosti `dsw:knowledgeModelId` se nejedná o primární identifikaci popisovaného zdroje, jelikož znalostní model je primárně identifikován pomocí `dsw:uuid`.

4.4 Využití existujících ontologií

V rámci návrhu ontologie bylo zváženo využití již existujících ontologií a slovníků. Jejich využití přispívá k propojitelnosti dat různých zdrojů, čímž je umožněno jednodušší sdílení. Zároveň je zajištěna konzistence, která snižuje možné chyby či nepřesnosti v reprezentaci dat.

V potaz byly brány hlavně často využívané a stabilní ontologie vzhledem k obrovskému počtu a různorodosti dostupných ontologií. Při vyhledávání pojmů bylo využito existujícího modelu, který propojuje nejčastěji využívané ontologie v oblasti správy dat [56], kde je možné najít většinu použitých termů z externích ontologií. Dále bylo využito Schema.org [31] za účelem vyhledání přesnějších a konkrétnějších reprezentací.

Celkem bylo využito 24 pojmů představujících vlastnosti i třídy. Všechna propojení jsou hierarchického typu, kde převzatý pojem je nadtypem pojmu z navrhované ontologie DSW. Je to z toho důvodu, že pojem z DSW ontologie vždy představuje konkrétnější definici. Pro vlastnosti by bylo možné využít ekvivalentního vztahu, ale všechny vlastnosti z navržené ontologie DSW jsou omezeny na třídy DSW a ty jsou, v případě že jsou propojeny, `rdfs:subClassOf` jiné třídy. Proto se i v případě vlastností jedná o specifikaci, a tak je využito `rdfs:subPropertyOf`.

Tabulka 4.1: Výskyt vlastností u tříd představující struktury DSW

Třída/Property	uuid	title	hasParent	annotation	order	createdAt	updatedAt	isCreatedBy	affiliation	email	hasMetric	prop	descritpion
Metric	x	x	x	x	x								x
Chapter	x	x	x	x	x								x
Tag	x	x	x	x	x								x
Phase	x	x	x	x	x								x
Expert	x	x	x	x	x					x			
Question	x	x	x	x	x								x
ListOfItemQuestion	x	x	x	x	x								
IntegrationQuestion	x	x	x	x	x							x	
Answer	x	x	x	x	x								
Integration	x	x	x	x	x							x	
SimpleUser	x												
QuestionnaireVersion	x	x				x	x	x					x
Questionnaire	x	x						x					x
User	x					x	x		x	x			
Package						x							x
Organization									x				x
Document	x					x	x						
KnowledgeModel	x			x									
Report	x					x	x						
Choice	x	x	x	x	x								
ResourcePageReference	x		x	x	x								
URLReference	x	x	x	x	x								
Reply						x		x					
ReportMetric											x		
MetricMeasure											x		

4.4.1 Propojení na úrovni vlastností

Na úrovni vlastností existuje několik propojení s obecně užívanými ontologiemi. Vždy se jedná o vztah `rdfs:subPropertyOf`, kde vlastnost z `dsw:` je podtypem jiné.

Vlastnost `dsw:uuid` je definována jako podtyp vlastnosti `dct:identifier`, která slouží k identifikaci zdroje. Vlastnost `dsw:description` je propojena s vlastností `dct:description`, která poskytuje popis zdroje.

Vlastnost `dsw:title` je propojena s vlastností `dct:title`, ta označuje název zdroje. Dále byly propojeny vlastnosti `dsw:createdAt` a `dsw:updatedAt` s vlastnostmi `dct:created` a `dct:modified`, které určují čas vzniku a aktualizace zdroje. V rámci návrhu ontologie jsou všechny tyto `dsw:` vlastnosti opakovaně využívány.

Dalším využitým pojmem je `foaf:logo`, který představuje logo zdroje. Tento pojem je propojen s vlastností `dsw:logo` třídy `dsw:Integration`. Dále je vlastnost `dsw:abberviation` propojena s vlastností `skos:altLabel`, která slouží k reprezentaci alternativního názvu zdroje.

Nadřazený pojem pro vlastnosti `dsw:hasVersion` a `dsw:hasCurrentVersion` je vlastnost `dct:hasVersion`, která vyjadřuje vztah ke zdroji, který představuje verzi daného popisovaného zdroje. Dále je využívána i inverzní vlastnost k `dct:hasVersion` a to vlastnost `dct:isVersionOf`, která je nadřazenou pro inverzní vlastnosti `dsw:hasVersion` a `dsw:hasCurrentVersion`.

Vlastnosti `dsw:clientUrl`, `dsw:referenceUrl`, `dsw:imageUrl` a `dsw:integrationReplyValue` jsou definovány jako podtypy vlastnosti `schema:url`. Tato vlastnost představuje URL popisovaného zdroje a má jako *range* svých hodnot `schema:URL`. Proto i výše vyjmenované vlastnosti mají totožný *range*.

4.4.2 Propojení na úrovni tříd

Na úrovni tříd se také vyskytuje využití již existujících ontologií. Tato propojení, na rozdíl od těch na úrovni vlastností, však ovlivňují i propojení právě na úrovni vlastností. Tato propojení jsou realizována pomocí vztahu `rdfs:SubClassOf`, kde třída z `dsw:` je podtypem jiné třídy. Níže jsou popsána všechna tato propojení.

Třída `dsw:User` je definována jako podtřída třídy `foaf:Person`, která představuje osobu a je podtřídou třídy `foaf:Agent`. Přestože existuje třída `sioc:User` ontologie Semantically-Interlinked Online Communities (SIOC), tak není využita neboť je definována ve smyslu uživatele v rámci on-line komunity a neodpovídá tak potřebnému nadtypu pro `dsw:User`. Díky tomu, že třída `dsw:User` je tranzitivně podtypem `foaf:Agent`, je možné pro vlastnost `dsw:isCreatedBy` a její inverzní vlastnost `dsw:creatorOf` definovat nadřazené vlastnosti a to `foaf:maker` a její inverzní vlastnost `foaf:made`. Tyto využité vlastnosti mají jako *range* (reps. *domain* pro inverzní vlastnost) právě `foaf:Agent`, a proto je možné je určit jako nadřazené vlastnosti jen ve chvíli, když je třída `dsw:User` podřazená třídě `foaf:Agent`, potažmo specifičtěji `foaf:Person`.

Pro vlastnosti třídy `dsw:User` byly využity vlastnosti `foaf:firstName` pro `dsw:firstName`, `foaf:surname` pro `dsw:lastName` a `foaf:mbox` pro vlastnost `dsw:email`.

Třída `schema:ContactPoint` představuje kontaktní bod. Tato třída byla definována jako nadřazená ke třídě `dsw:Expert`. Ta totiž také představuje kontakt konkrétně na odborníka, který je schopný zodpovědět a být nápomocný v oblasti otázky dotazníku. Tento vztah umožňuje využívat některé vlastnosti třídy `schema:ContactPoint` jako nadřazené vlastnosti. Konkrétně se jedná o vlastnost `dsw:email`, která je podtypem `schema:email`. A dále vlastnosti `dsw:title`, `dsw:uuid` a `dsw:description`, které třída `dsw:Expert` dědí z abstraktních tříd představující tyto vlastnosti. Definované vlastnosti v abs-

traktních třídách jsou již samy o sobě podtypy vlastností z ontologie DC, které jsou ekvivalentní s těmito vlastnostmi definovanými v ontologii Schema.org. Z tohoto důvodu mohla třída `dsw:Expert` zůstat potomkem příslušných abstraktních tříd.

Jedinou komplikací se zdála být vlastnost `dsw:email`, která je v návrhu sdílena také s třídou `dsw:User` za pomocí abstraktní třídy `dsw:EmailableElement` a to kvůli tomu, že třída `dsw:User` dědí od třídy `foaf:Person` a třída `dsw:Expert` dědí od třídy `schema:ContactPoint`. Jelikož ale existuje vlastnost `schema:email`, která má jako *domain* třídu `schema:ContactPoint` i třídu `schema:Person`, která je definována jako ekvivalentní třídě `foaf:Person`, bylo možné abstraktní třídu `dsw:EmailableElement` zachovat.

Třída představující znalostní model `dsw:KnowledgeModel` byla definována jako podtyp třídy `schema:CreativeWork`. Tato třída slouží k označení libovolného výsledku tvůrčí činnosti. Díky tomuto navrženému vztahu mezi třídami bylo možné vlastnost `dsw:version` znalostního modelu definovat jako podtyp vlastnosti `schema:version`. Ta na rozdíl od `dct:hasVersion` má range `xsd:string`, což je v případě `dsw:version` třídy `dsw:KnowledgeModel` žádoucí.

Dále je třída `dsw:Organization` definována jako podtyp třídy `schema:Organization`, která představuje organizaci v širokém pojetí.

4.4.3 Nevyužité propojení s existujícími ontologiemi

Při vyhledávání pojmů z existujících ontologií vhodných k propojení se vyskytlo několik návrhů, které nakonec nebyly zapracovány. Zde je jejich výčet i s odůvodněním, proč se tak nestalo.

Bylo zvažováno propojit vlastnost `dsw:hasReference` s `dct:reference`. Tato vlastnost představuje vazbu na citovaný objekt, kdežto `dsw:hasReference` reprezentuje vazbu na referenci v podobě třídy `dsw:Reference`, ne na konkrétní objekt.

V Schema.org byly nalezeny třídy `schema:Question` a `schema:Answer`, které by mohly reprezentovat nadtypy tříd `dsw:Question` a `dsw:Answer`. Způsobilo by to ale změnu celé struktury ontologie, která by pak neodpovídala složení DSW. Nicméně, postavit ontologii komplexně na existujících pojmech by bylo možné.

Dále bylo zváženo definovat jako nadřazenou třídu `foaf:Organization` pro třídu `dsw:Organization`, tato třída je ale považována za nestabilní, a tak nakonec byla využila třída `schema:Organization`.

V Schema.org byly nalezena třída `schema:Chapter`, ta je ale úzce definována pouze jako kapitola knihy, což použití neodpovídá.

Existuje vlastnost `schema:affiliation`, která užitím odpovídá vlastnosti `dsw:affiliation`. Nalezená vlastnost, má ale *range* konkrétní organizace, do kterých může osoba patřit. V případě `dsw:affiliation` se ale, dle dat, jedná pouze o názvy organizací, či skupin, které ani nemusí reálně existovat.

Dále existuje vlastnost `schema:parentItem`, která byla zvažována jako nadřazená vlastnost pro vlastnost `dsw:hasParent`. Taktéž se jedná o vyjádření vztahu mezi rodičem a potomkem, ale pouze mezi instancemi `schema:Comment`, čili komentářů. To však neodpovídá elementům znalostního modelu, mezi kterými se tento druh vztahu vyskytuje, a proto vlastnost `schema:parentItem` nebyla využita.

4.5 Inverzní vlastnosti

Aby bylo docíleno úplné a srozumitelné ontologie, byly přidány inverzní *object properties* mezi třídami. Toto obohacení umožňuje dotazovat se na data i z opačného směru, odvozovat další vztahy a zároveň umožňuje rozsáhlejší kontrolu konzistence dat.

Celkem bylo vytvořeno 22 inverzních vztahů, mezi kterými jsou například `dsw:desiresQuestion` inverzní k `dsw:isDesirableAtPhase`, který říká které všechny otázky je v dané fázi vyžadováno mít zodpovězené. Nebo třeba vlastnost `dsw:IntegrationOf` inverzní k `dsw:hasIntegration`, která určuje všechny otázky typu *Integration Question*, které využívají danou integraci.

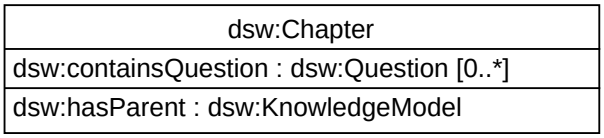
Pro *object properties* třídy `dsw:Document` byly vytvořeny také inverzní vlastnosti. V případě, že jsou data získána z jednoho dotazníku, je očekáváno, že všechny jeho inverzní vlastnosti budou mít stejnou hodnotu, která odkazuje na jediný dokument. Nicméně, při sloučení několika souborů dat, inverzní vlastnosti umožňují jednoznačně určit, ke kterému exportovanému dokumentu přísluší daný znalostní model, dotazník, organizace, report nebo související konfigurace. To přispívá ke zvýšení přesnosti a konzistence, což je důležité zejména při integraci dat.

4.6 Model ontologie a jeho notace

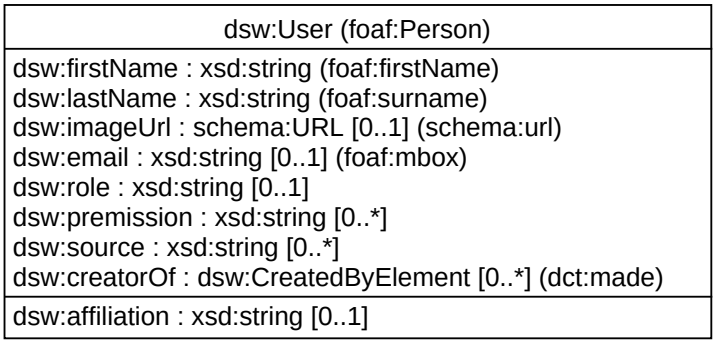
Návrh ontologie probíhal v grafické podobě ve formě class diagramu s vlastní notací. Kompletní model návrhu ontologie je dostupný v příloze B.

Tento model obsahuje klíčové informace o vztazích v rámci reprezentací objektů DSW včetně propojení s existujícími ontologiemi a slovníky. Při jeho tvorbě byl kladen důraz na co nejefektivnější zachycení všech potřebných informací, ale zároveň bylo dbáno na minimalizaci redundancí a eliminaci možné víceznačnosti. Cílem bylo zachytit vztahy a vlastnosti v ucelené a přehledné formě, která poskytuje dostatečnou orientaci.

Třídy jsou reprezentovány klasicky pomocí boxů. V horní části je název třídy a následuje výčet vlastností třídy spolu s definicí *range* i jejich multiplicitou definovanou v hranatých závorkách. V případě, že se jedná o multiplicitu přesně jedna, je její definice vynechána. Dále jsou u některých tříd uvedeny pod čarou vlastnosti, které třída dědí od jiné, ale jako vlastnosti dané třídy jsou modifikovány. Může se jednat například o modifikaci v oblasti multiplicit



Obrázek 4.2: Třída `dsw:Chapter`



Obrázek 4.3: Třída `dsw:User`

či *range*. Na 4.2 je vidět modifikace vlastnosti `dsw:hasParent`, kterou třída `dsw:Chapter` dědí od abstraktní třídy `dsw:ChildElemet` a je nutné ve třídě `dsw:Chapter` upravit její *range*.

Vazby plynoucí z *object properties* jsou znázorněny vazbou se šipkou ve směru vztahu. Každý takový vztah je popsán názvem a opatřen multiplicitami. Inverzní vazby jsou v modelu zachycené pomocí čárkované čáry za účelem viditelného oddělení. Pro lepší pochopitelnost jsou všechny uměle vytvořené abstraktní třídy barevně odlišené včetně vazeb dědičnosti k jejich podtřídám. Naopak ostatní abstraktní třídy, které vychází původem z nástroje DSW a představují více obecný koncept objektu, jako je například `dsw:Reply`, barevně označeny nejsou.

Dále jsou v modelu znázorněny i propojení s existujícími ontologiemi a slovníky. Využitý externí pojem je zanesen do závorky za pojem v navrhované ontologii. Jak již bylo zmíněno, všechna propojení jsou hierarchického typu, kdy externí pojem je nadtypem pojmu z navrhované ontologie DSW. Díky tomu nebylo potřeba definovat druh vztahu mezi nimi. Na 4.3 je viditelné znázornění vztahu mezi externími ontologiemi či slovníky. Není potřeba ani dále specifikovat zda se jedná o vztah mezi třídami či mezi vlastnostmi, protože je jasné kdy se jedná o název vlastnosti a kdy o název třídy.

4.7 Výsledná ontologie

Výsledná ontologie byla zapsána pomocí standardu RDF ve formátu Turtle. Pro zápis byl využit textový editor *Visual Code Studio* (verze 1.78.2) [1] spolu s rozšířením pro syntaxi Turtle nazvaným *Turtle Language Server* (verze 0.2.1) [2]. Toto rozšíření poskytuje užitečné funkce, jako je kontrola syntaxe, automatické předpovídání názvů a zvýraznění syntaxe. Díky Turtle bylo možné použít zjednodušený zápis, který umožňuje vynechání subjektů a predikátů.

Ve výsledné ontologii byly zachyceny všechny třídy, jejich vlastnosti i propojení s existujícími ontologiemi a slovníky, jak bylo specifikováno v návrhu. Kompletní ontologie je dostupná v příloze B. V této sekci je popsán způsob zápisu jednotlivých prvků ontologie.

4.7.1 Definice tříd a vlastností

Každý ontologický prvek, který definuje určitou třídu nebo vlastnost, je vyznačen identifikátorem IRI reprezentujícím definovaný subjekt následovaný predikátem `rdf:type`. Pokud je objekt této vlastnosti `rdfs:Class`, jedná se o třídu, zatímco pokud je objektem `owl:ObjectPropety` nebo `owl:DataProperty`, jedná se o vlastnost, a to buď *object property* nebo *data property*. Dále se v ontologickém prvku upřesňuje název třídy či vlastnosti a poskytuje se přesnější popis v podobě komentáře, což je patrné na ukázce 4.1 definice abstraktní třídy `dsw:CreatedByElement`.

Ukázka kódu 4.1: Definice třídy `dsw:CreatedByElement`

```
dsw:CreatedByElement
    rdf:type rdfs:Class ;
    rdfs:comment "An abstract class carrying the CreatedBy
    ↪  property."@en ;
    rdfs:label "Created By Element"@en .
```

Při definici ontologického prvku, který popisuje vlastnost, bylo nezbytné specifikovat také jeho *range* a *domain* ať už se jedná o vlastnost typu *object property* nebo *data property*. Na ukázce 4.2 je definice `dsw:usesChoice`.

4.7.2 Určení hierarchie

V návrhu ontologie se vyskytuje hierarchické řazení nejen v rámci navrhované ontologie, ale i mimo ni s využitím existujících ontologií a slovníků. Pro

[1] https://code.visualstudio.com

[2] https://marketplace.visualstudio.com/items?itemName=stardog-union.vscode-langserver-turtle

Ukázka kódu 4.2: Definice vlastnosti `dsw:usesChoice`

```
dsw:usesChoice rdf:type owl:ObjectProperty ;
               rdfs:domain dsw:MultiChoiceQuestion ;
               rdfs:range dsw:Choice .
```

vyjádření situace, kdy třída je podtypem jiné třídy, ať už v rámci navrhované ontologie nebo mimo ni, bylo využito predikátu `rdfs:subClassOf`. Na ukázce 4.3 je vidět použití v případě dědičnosti třídy `dsw:ValueQuestion` od třídy `dsw:Question`. Jak bylo zmíněno v 4.4, všechny převzaté pojmy jsou nadtypy pojmů z navrhované ontologie, takže i v případě pojmů jiných ontologií bylo využíváno stejného predikátu. Pro určení hierarchie vlastností bylo využito paralelního predikátu `rdfs:subPropertyOf`.

Ukázka kódu 4.3: Definice vlastnosti `dsw:usesChoice`

```
dsw:ValueQuestion rdf:type owl:Class ;
                  rdfs:subClassOf dsw:Question .
```

4.7.3 Specifikace vlastností třídy

V ontologickém prvku, který definuje vlastnost, bylo nutné specifikovat její *range* a *domain*. Z toho plyne, které třídy tyto vlastnosti mají a případně instance kterých tříd může vlastnost nabývat. V případě, že bylo potřebné specifikovat konkrétnější *range* pro využití vlastnosti třídou než je v její definici, pak bylo potřeba definovat tzv. *restriction* neboli omezení. Na ukázce 4.4 je příklad užití restrikce ve výsledné ontologii, konkrétně v případě vlastnosti `dsw:hasParent` třídy `dsw:Chapter`. Vlastnost v kontextu této třídy může nabývat pouze instancí třídy `dsw:KnowledgeModel`.

Výchozí multiplicity definovaných vlastností jsou shora i zdola neomezené. Proto bylo v některých případech nutné pomocí restrikcí omezit násobnosti vlastností v *domain* třídách. V 4.5 je ukázka ontologického prvku definice třídy `dsw:Questionnaire` s restirkcí násobnosti vlastností `dsw:hasCurrentVersion` a `dsw:hasPhase`.

4.7.4 Validace výsledné ontologie

Kontrola správnosti zápisu ontologie byla provedena v nástroji *Protégé* [47]. Tento software disponuje integrovaným validátorem syntaxe, který umožňuje

Ukázka kódu 4.4: Specifikace *range*

```
dsw:Chapter rdf:type owl:Class ;
            rdfs:subClassOf [ rdf:type owl:Restriction ;
                              owl:onProperty dsw:hasParent ;
                              owl:allValuesFrom
                              ↪  dsw:KnowledgeModel
                            ] .
```

Ukázka kódu 4.5: Specifikace násobnosti

```
rdfs:subClassOf [ rdf:type owl:Restriction ;
                  owl:onProperty dsw:hasCurrentVersion ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger
                ] ,
                [ rdf:type owl:Restriction ;
                  owl:onProperty dsw:hasPhase ;
                  owl:maxCardinality "1"^^xsd:nonNegativeInteger
                ] .
```

kontrolovat správnost zápisu ontologie i ve formátu Turtle. V případě detekce chyb v ontologickém zápisu byly zdrojové soubory ontologie upraveny a následně byla provedena kontrola zda byly odstraněny všechny zjištěné chyby.

Pro kontrolu konzistence a koherence ontologie byl využit přídavný modul *OntoDebug* [48], který je možné importovat do nástroje *Protégé*. Při kontrole nově vytvořené ontologie nebyly identifikovány žádné nekonzistence. Nicméně vzhledem k využití již existujících ontologií došlo k mylnému zjištění několika nekonzistencí. Důvodem byla nemožnost importu všech ontologií do nástroje *Protégé*, a tedy modul nesprávně signalizoval chyby v definicích použitých ontologií.

Dále byla ontologie validována v on-line nástroji OOPS! [50]. Tento nástroj detekoval několik chyb, které byly opraveny a zároveň detekoval i několik méně závažných indikací, které opraveny nebyly. Jedná se primárně o absenci definic inverzních vztahů. Nástroj například považoval za chybné, že neexistuje definice inverzního vztahu pro vlastnost `dsw:containsIntegration`. Tato vlastnost je ale podtypem vlastnosti `dsw:containsChild`, která má inverzní vlastnost `dsw:hasParent` a ta se dále do podtypů nedělí. Z tohoto důvodu tato indikovaná chyba nebyla vyřešena. Další indikace poukazovala na nedostatečnou definici. Týkalo se to ale pouze tříd a vlastností, které jsou importované z ex-

terních ontologií a tak přesná definice není nutná, neboť se vyskytuje v jejich primárním zdroji.

4.7.5 Způsoby publikace ontologií

Zveřejnění ontologie je podstatným krokem v procesu její tvorby. Díky prezentaci ontologie širší komunitě se stává dostupná ostatním potenciálním uživatelům, kteří ji mohou využít či dále sdílet. Tímto krokem je ontologická znalost dále rozvíjena a přispívá tím k inovaci v dané oblasti. Z tohoto důvodu byl analyzován přístup tvůrců známých ontologií k jejich publikaci.

V základu existuje několik způsobů publikace ontologií. Jedním z nich je publikace v odborných časopisech, které se specializují na danou problematiku. Touto cestou se ontologie dostanou primárně k odborníkům, a tak je možné získat zpětnou vazbu.

Dalším způsobem publikace jsou konference a workshopy, během kterých je ontologie představena. Tento kanál umožňuje téměř okamžitou zpětnou vazbu i diskuzi.

Oba výše vyjmenované způsoby představují spíše publikaci ve smyslu prezentace vytvořené ontologie. Ta je sice v rámci procesu tvorby ontologie také velmi důležitá, ale o něco podstatnější je publikace ve smyslu dostupného umístění ontologie pro její možné další využití a sdílení.

Pro publikování ontologie ve smyslu dostupného umístění existují dle provedené analýzy dva přístupy. Prvním je využití repozitáře a druhým je publikace v rámci vlastního on-line zdroje. V každém případě je důležitý perzistentní URL identifikátor používaný pro odkazování na danou ontologii. Proto bylo analyzováno i přesné schéma těchto perzistentních identifikátorů včetně způsobu vypořádání se s verzemi ontologií. Dále byl zkoumán způsob správy verzí a pod jakou licencí byly ontologie publikovány. V provedené analýze byly zohledněny známé existující ontologie a jejich přístup k publikaci.

4.7.5.1 Dublin Core Ontology

DC [28] využívá webovou stránku projektu [57], kde je publikována dokumentace ontologie a její specifikace včetně historických verzí. Její URL adresa se v případě historické verze doplňuje o určení verze pomocí data vydání, například `/dcmi-terms/2010-10-11/`.

Perzistentní URL adresa vždy odkazuje na aktuální verzi. Například v případě *Dublin Core Terms* má podobu `http://purl.org/dc/terms/`. DC je licencována *Creative Commons Attribution 4.0 International License* [58].

4.7.5.2 Data Cite Vocabulary

DCAT [29] je publikována na stránkách konsorcia *W3C*. URL neaktuálních verzí se liší podle názvu dané verze. Aktuální verze má v cestě URL pouze

název verze `/vocab-dcat-3/`, naopak historické verze mají celý název doplněný o datum vydání verze, například `/WD-vocab-dcat-3-20220111/`. DCAT je licencována *W3C Document License* [59], kterou užívají všechny dokumenty konsorcia *W3C*.

4.7.5.3 Friend of a Friend

FOAF [30] ontologie využívá stránku projektu pro publikování specifikace i dokumentace. Ačkoliv její URL adresa obsahuje označení verze, existuje verze pouze jedna. FOAF je licencována *Creative Commons Attribution 1.0 Generic* [60].

4.7.5.4 Semantically-Interlinked Online Communities

SIOC ontologie má hlavní stránku projektu [61], ta ale odkazuje na ontologický repozitář, který představuje centrální místo sdílení. Verze této ontologie buď nejsou přístupné nebo se nenachází na stránce projektu spravujícího tuto ontologii. SIOC ontologie je sdílena pod licencí *Creative Commons Attribution 1.0 Generic* [60].

4.7.5.5 EDAM ontologie

EDAM ontologie [62] má oficiální stránky [63], kde jsou dostupné jednotlivé verze ke stažení. Dále využívá nástroj *GitHub* [64] pro uložení a správu verzí ontologie i dokumentace. Verze jsou uloženy ve struktuře složek s cestou `tree/main/releases`. Dokumentace je generována za pomocí open-source platformy *Read the Docs* [3]. EDAM je sdílena pod licencí *Creative Commons Attribution Share Alike 4.0 International* [58].

4.7.5.6 Data Use Ontology

Data Use Ontology (DUO) [65] využívá nástroj *GitHub* [64] pro správu verzí ontologie. Neaktuální verze jsou uloženy ve struktuře složek s cestou `/master/src/ontology/`. Poslední aktuální verze je uložena ve složce `/master/` a odkazuje se na ni persistentní URL `http://purl.obolibrary.org/obo/duo.owl`. DUO je registrováno v *OBO Foundry* [66] a její dokumentace je dostupná například v *Ontobee* [67]. DUO je publikována pod licencí *Creative Commons Attribution 4.0 International* [58].

4.7.5.7 DataCite Ontology

DataCite [68] ontologie využívá nástroj *GitHub* [64] pro správu verzí ontologie. Všechny verze jsou uloženy ve struktuře složek s cestou `/docs/`, která je

[3] https://readthedocs.org/

v případě aktuální verze doplněna o `/current` a v případě historických verzí o datum publikace například `/2016-01-21`.

Dokumentace, i její historické verze, jsou dostupné s perzistentní URL. Aktuální verze má `http://purl.org/spar/datacite` a historické verze mají cestu doplněnou o datum publikace například `/2021-09-24`. DataCite je publikována pod licencí *Creative Commons Attribution 4.0 International* [58].

4.7.5.8 Software Ontology

Software Ontology (SWO) [69] využívá nástroj *GitHub* [64]. Na aktuální verzi ontologie odkazuje perzistentní URL adresa `http://purl.obolibrary.org/obo/swo.owl`. SWO je publikována pod licencí *Creative Commons Attribution 4.0 International* [58].

4.7.5.9 Publikace ontologie DSW

Vytvořená ontologie byla publikována v *GitHub* [64] repozitáři organizace *Data Stewardship Wizard* s názvem *rdf-report-template*. Tento repozitář obsahuje dokument `dsw-ontology` s aktuální navrženou ontologií. Dále obsahuje dokument `README` se stručným popisem, dokument s názvem `LICENSE` s definicí licence, pod kterou je ontologie publikována a dokument `CHANGELOG` [70] s informacemi o verzích a jednotlivých změnách. Dále byl do repozitáře přidán model ontologie pro jednodušší orientaci, ten se nachází ve složce nazvané `ontology-model`.

Ontologie je publikována pod licencí *Creative Commons Attribution 4.0 International* [58]. Dokumentace je tvořena pomocí nástroje WIDOCO [54] přímo z *GitHub* repozitáře.

Dokumentace vytvořené DSW ontologie je publikována na webové stránce `https://ontology.ds-wizard.org/dsw-ontology#`. Byla rezervována perzistentní URL adresa `http://purl.org/ds-wizard/dsw-ontology/`, odkazující na tuto stránku. V její cestě se vyskytuje název `ds-wizard`, který je již registrovaný pro nástroj DSW.

Tato URL odkazuje přímo na aktuální verzi dokumentace. Díky využití *URL redirection* je možné doplnit *path* nebo *fragment* této perzistentní URL. Doplněním *fragmentu* prohlížeč přímo nalezne část dokumentu s tímto identifikátorem, což lze využít v případě hledání konkrétní třídy ontologie. Doplněním *path* je možné se odkázat přímo na konkrétní historickou verzi ontologie.

Pro označení verzí byl využit způsob *Calendar Version* [71], který definuje verze podle data publikace konkrétně ve formátu `YYYY-0M-0D`. Tedy kompletní zápis roku, číslo měsíce doplněné o nulu a číslo dne v měsíci doplněné o nulu. Aktuální je verze „2023-06-19“.

Kapitola 5

Návrh RDF reprezentace dle navržené ontologie

Dle DSW ontologie byla navržena RDF reprezentace dat. Při návrhu bylo důležité rozhodnout v jakém formátu bude výsledek reprezentován, přesné použití notace a jakým způsobem budou strukturovaná data z dotazníku procházena za účelem jejich výpisu. Kompletní návrh RDF reprezentace, která byla vytvořena z dat existující instance nástroje DSW, je v příloze B. Existující instance byla vytvořena pro účely návrhu a testování. Jedná se o velmi jednoduchý znalostní model na základě kterého byl vytvořen dotazník, vyplněný pouze testovacími daty.

5.1 Návrh syntaxe

V kontextu uvedeném v 3.5 je patrné, že existuje několik syntaxí, které jsou standardem konzorcia W3C a aktivně se používají pro reprezentaci dat ve formátu RDF. Mezi všemi zaujímá významné místo syntaxe RDF/XML. Byť je tento syntax hojně užívaný a rozšířený, je i hodně komplexní. V důsledku toho, že je založen na formátu XML, je složen ze stromové struktury, využívá vnoření a značkování pomocí *tags*. Z tohoto důvodu se tak stává lidem hůře čitelným i zapisovatelným, zejména při práci s velkým množstvím dat, jako je tomu například při zpracování dat z dotazníku.

Mezi další využitelné syntaxe patří například *N-Tripes*, která ale vyžaduje, aby každá trojice byla reprezentována na novém řádku. Navíc nepodporuje identifikaci komponent za pomocí prefixů, což ztěžuje zápis IRI. Z těchto důvodů není *N-Triples* vhodný syntax pro toto použití.

Pro reprezentaci dat byl zvolen formát Turtle, jelikož se jedná o syntaxi, která je snadno čitelná i zapisovatelná. Navíc poskytuje možnosti využití různých zjednodušení jak bylo zmíněno v 3.5.2. Tato volba byla motivována také popularitou formátu *Turtle*, který je podporován mnoha nástroji a široce

používán v praxi.

Při návrhu RDF reprezentace dat byl využit nástroj *Visual Code Studio* (verze 1.78.2)[4] spolu s rozšířením pro syntax Turtle nazvaným *Turtle Language Server* (verze 0.2.1)[5]. Toto rozšíření poskytuje užitečné funkce jako je kontrola syntaxe, automatické předpovídání názvů a zvýraznění syntaxe, které usnadňují proces návrhu a editace reprezentace RDF dat.

5.2 Návrh zápisu RDF reprezentace

Výsledná podoba výpisu RDF dat byla navržena s ohledem na jednoduchost, kompaktnost i dobrou čitelnost. Organizovaný výpis RDF dat přispívá k lepší orientaci v konečné reprezentaci. Díky použití zjednodušeného výpisu v syntaxi Turtle, je možné informace zapsat zkráceně a eliminovat jejich redundanci. Přestože je standard RDF navržen hlavně pro strojovou čitelnost, při návrhu reprezentace dat bylo usilováno o zvýšení lidské čitelnosti.

Všechny instance jedné třídy mohou být vypisovány v reprezentaci u sebe, což přispívá ke zlepšení orientace v celém dokumentu. Při reprezentaci vlastností jednotlivých instancí tříd byly zapsány v takové posloupnosti, aby odpovídaly pořadí zápisu v rámci modelu navržené ontologie.

Kromě toho byla využita možnost jednoduššího zápisu díky využití syntaxe Turtle. Pro zápis IRI byly využity označení `@prefix` s cílem zlepšit přehlednost a usnadnit úpravy. Těch bylo navrženo celkem 12 s ohledem na používané IRI. Při jejich návrhu se využívaly identifikace konkrétních instancí v nástroji DSW. Na ukázce kódu 5.1 jsou uvedeny některé nich.

Ukázka kódu 5.1: Příklady definovaných prefixů

```
@prefix dsw: <http://ds-wizard.org/ontology#> .
@prefix ctxConfig: <{{ ctx.config.clientUrl }}/contextConfig> .
@prefix user: <{{ ctx.config.clientUrl }}/users/> .
@prefix document: <{{ ctx.config.clientUrl }}/documents/> .
@prefix questionnaire: <{{ ctx.config.clientUrl }}/projects/> .
@prefix km: <{{ ctx.config.clientUrl }}/knowledge-models/{{
↪  ctx.package.id }}/> .
```

Prefix `dsw:` slouží k označení IRI celé navrhované ontologie. Další prefixy jsou použity k reprezentaci IRI jednotlivých instancí tříd v ontologii nástroje DSW. Pro sestavení IRI je obvykle nutné ji doplnit o tzv. *clientUrl* neboli URL

[4] https://code.visualstudio.com

[5] https://marketplace.visualstudio.com/items?itemName=stardog-union.vscode-langserver-turtle

adresy instance nástroje DSW. V některých případech je nutné využít dalších proměnných, které mají specifickou hodnotu v závislosti na instanci nástroje, konkrétním znalostním modelu nebo třeba dotazníku. Například prefix `km:` je specifikován s využitím identifikátoru objektu *package*, protože jednotlivé prvky znalostního modelu i samotný znalostní model jsou v nástroji odkazovány tímto způsobem.

Prefix může buď označovat konkrétní instanci, jako je tomu například u prefixu `ctxConfig:`. Nebo je zakončen lomítkem „/“ a za lomítko je při generování výsledného dokumentu dosazen identifikátor konkrétního zdroje, jako je tomu například u prefixu `km:`.

Dále byly využity další možnosti syntaxe Turtle a to konkrétně vynechání subjektu a predikátu v RDF zápisu. V případě, že jeden subjekt má vícero predikátů, je možné jej při zápisu dalšího tripletu vynechat. Stejně tak pokud predikát má více objektů je možné jej při zápisu další trojice vynechat, což je popsáno v 3.5.2. Tento způsob zápisu vede k čitelnější reprezentaci. Na ukázce 5.2 je návrh použití tohoto zjednodušení na třídě `dsw:Chapter`. Subjekt je zapsaný pouze jedenkrát a taktéž predikát, v návrhu nazvaný `dsw:containsQuestion`, je zapsán pouze jednou, přičemž pro uvedení příkladu má objekty čtyři. Hodnoty na ukázce 5.2 pochází ze znalostního modelu, existující instance, vytvořeného pro testovací účely.

Ukázka kódu 5.2: Example Chapter RDF representation

```
km:8ce36f08-4fbd-4861-9691-a28c36d6dcdb a dsw:Chapter ;
  dsw:uuid "8ce36f08-4fbd-4861-9691-a28c36d6dcdb" ;
  dsw:title "Example Chapter" ;
  dsw:description "Description of Example Chapter" ;
  dsw:containsQuestion km:b96b957b-e2af-4ac8-a410-6275578b8d5f,
  ↪  km:26b70241-777c-4495-8a7b-2778e20b0568,
  ↪  km:be1d05a8-4e62-4897-aa1f-0cf355ed54db,
  ↪  km:f9917b32-8727-44ed-bb17-96b827f74aaa ;
  dsw:order "1"^^xsd:integer ;
  dsw:hasParent km:9f017486-71d3-42d9-a83a-053f10a1132d .
```

Pro případ zápisu objektů bez unikátního identifikátoru byl využit tzv. *blank node* neboli prázdný uzel. Toho se využívalo, v případě zápisu vlastnosti `dsw:annotation` či `dsw:prop`. Ty mají jako *range* třídu `dsw:KeyValueEntry`, která byla vytvořena přímo pro toto použití. Jak je vidět na ukázce 5.3, objekt vlastnosti `dsw:annotation` je blank node a obsahuje predikáty a objekty s obsahem položky anotace, přičemž uvedení subjektu je vynecháno.

Ukázka kódu 5.3: Zápis vlastnosti `dsw:annotation` pomocí blank node

```
km:0a18d9fc-abb1-46ed-b65b-ec302e94a1b3 a dsw:Metric ;
  dsw:uuid "0a18d9fc-abb1-46ed-b65b-ec302e94a1b3" ;
  dsw:annotation
    [ a dsw:KeyValueEntry ;
      dsw:key "Annotation of example metric" ;
      dsw:value "value of annotation of example metric" ] ;
```

5.3 Návrh zpracování dat

V této sekci je popsán způsob zpracování dat za účelem jejich výpisu v RDF reprezentaci. Při návrhu bylo vycházeno z exportní šablony *Questionnaire Report* [7], která vypisuje všechna data obsažená v dotazníku.

Zpracování dat závisí na struktuře, ve které jsou data uložena. Velkou část tříd lze vypsat přímo, například `dsw:Questionnaire`, `dsw:KnowledgeModel`, nebo například `dsw:Report`. Tyto třídy v návrhu ontologie totiž nejsou nositeli vlastností, které příslušné datové objekty neobsahují. Naopak většina tříd spadajících do oblasti znalostního modelu jsou nositeli například vlastnosti `dsw:hasParent`, která představuje odkaz na jejich rodičovské třídy. Tato informace ale není u příslušných datových objektů explicitně zapsaná, naopak jsou zaneseny odkazy na třídy potomků. Proto je potřeba speciálního přístupu, který je popsán v části 5.3.2.

Nejsložitější však je zpracování stromové struktury a výpis otázek znalostního modelu a přiřazených odpovědí, který je popsán v části 5.3.1.

5.3.1 Návrh zpracování otázek

Při návrhu zpracování RDF reprezentace otázek bylo zvažováno, zda by měly být vypsány všechny existující otázky znalostního modelu nebo pouze ty, které jsou v dotazníku přítomné, tedy relevantní otázky na základě předchozích odpovědí. Nakonec byl navržen přístup zahrnující všechny otázky ve znalostním modelu. Je to z toho důvodu, že z této verze je úprava na výpis pouze relevantních otázek jednodušší, jelikož je možné omezit zanoření průchodu otázek dle zvolených odpovědí *Answer* i *Choice*, na které jsou podotázky navázány.

Při zpracování otázek se vychází z jednotlivých kapitol, které slouží k seskupení otázek. Průchod stromovou strukturou otázek začíná vnořeným *cyklem for*, jak je viditelné v algoritmu 1. Pro každou otázku je volána obecná procedura, nazvaná v návrhu *QuestionType*, společně s aktuálně zpracovávanou otázkou. Tato procedura slouží k rozdělení otázek podle jejich typu pro další zpracování.

Algorithm 1 Algoritmus zpracování otázek

```
function QuestionProcessing
    for chapter in chapters do
        for question in chapter.questions do
            QuestionType(question)
        end for
    end for
end function
```

Původní návrh zahrnoval při průchodu stromu znalostního modelu konstrukci kompletního identifikátoru odpovědí s názvem *path*. Ten se totiž skládá z identifikátorů *uuid* jednotlivých otázek, odpovědí a kapitoly předcházející této konkrétní odpovědi. Tento přístup by byl přínosný, neboť poskytuje úplnou kontrolu správného průchodu otázek znalostního modelu. Avšak poslední identifikátor obsažený v *path* představuje identifikaci otázky, na kterou odpověď je. Z toho plyne, že při průchodu znalostním modelem není nutné uchovávat celou *path*, ale k odkazu na odpověď stačí pouze *uuid* otázky, na kterou hledáme odpovědi. Byť by tedy konstrukce kompletní *path* přinesla další validaci, tak se nakonec nevyužila, neboť díky *List Of Items* otázce by bylo velmi komplikované správně zkonstruovat všechny *path* odpovědí na její podotázky. Je to z toho důvodu, že k jednotlivým podotázkám se řadí všechny odpovědi k ní patřící napříč položkami seznamu odpovědí. A proto by bylo potřeba přenášet několik různých *path*, které by se postupným zanořováním pouze rozšiřovaly.

Na algoritmu 2 je viditelný způsob hledání odpovědi na právě zpracovávanou otázku. Jsou vyhledány odpovědi, které mají shodnou poslední část *path* s identifikátorem *uuid* právě zpracovávané otázky.

Algorithm 2 Algoritmus vyhledání odpovědi na otázku

```
function FindReply(question)
    for reply in replies do
        if reply.endswith('.'+question.uuid) then
            searchReply is reply
        end if
    end for
end function
```

Dále je podrobně popsán způsob zpracování otázek typu *List Of Item* a *Options*, jelikož právě díky nim je možné prohlubovat strom otázek znalostního modelu.

5.3.1.1 List Of Items otázka

Pro otázku typu *List Of Items* je důležité projít i její jednotlivé podotázky. Pokud tento typ otázky obsahuje odpověď, pak hodnoty takové odpovědi slouží pouze jako identifikátory. Ty označují položky, složené vždy z totožných otázek, v seznamu odpovědí. To pro průchod znamená větvení podle otázek v položce seznamu a dohledání všech odpovědí k nim patřících napříč všemi položkami.

Z algoritmu 3 je patrné, že je důležité zda má otázka *List Of Items* navázané podotázky neboli *followup*. Z povahy použití tohoto typu otázky by takové podotázky existovat měly, editor znalostních modelů ale umožňuje je nedefinovat.

Jelikož podotázky mohou být různých typů, je pro každou novou podotázku volána obecná procedura, která je v návrhu algoritmu nazvána *QuestionType*. Tato funkce slouží k rozdělení otázek podle jejich typu pro další zpracování.

Algorithm 3 Algoritmus zpracování otázky typu *List Of Items*

```
procedure ListOfItemsQuestion(question)
    if question.followups > 0 then
        for followup in question.followups do
            QuestionsType(followup)
        end for
    end if
end procedure
```

5.3.1.2 Options otázka

Pro otázky typu *Options* vzniká také povinnost zpracování jejich nepřímých podotázek. Tento typ otázky nemá přímo navazující podotázky, ale jejich předdefinované odpovědi neboli *answers* mohou být nasledovány definovanými podotázkami.

Z algoritmu 4 je patrné, že záleží na přítomnosti definovaných odpovědí *answers* pro danou otázku a dále na definovaných navazujících otázkách neboli *followup* pro danou *answer*. U tohoto typu otázky je nezbytné přistoupit k následným otázkám prostřednictvím odpovědí, protože jsou definované jen pro ně.

Pro novou podotázku je volána obecná metoda *QuestionsType*, která slouží k rozdělení otázek podle jejich typu pro další zpracování.

5.3.2 Uložení rodičovského elementu a pořadí v něm

V návrhu existuje několik tříd, které nelze zpracovat přímočaře pouze z informací, které jsou obsažené v příslušném datovém objektu. Jedná se o vlastnosti `dsw:hasParent` a `dsw:order` tříd v části znalostního modelu. Vlast-

Algorithm 4 Algoritmus zpracování otázky typu *Options*

```
OptionsQuestion (question)
for answer in question.answers do
    for followup in answer.followups do
        Questions(followup)
    end for
end for
```

nost `dsw:hasParent` představuje vztah k rodičovskému elementu a `dsw:order` představuje pořadí v množině prvků rodičovského elementu.

Třídy `dsw:Metric`, `dsw:Chapter`, `dsw:Tag`, `dsw:Phase` a `dsw:Integration` jsou nositelé vlastností `dsw:hasParent` a `dsw:order`. Jejich rodičovský element patří do třídy `dsw:KnowledgeModel`. Díky tomu, že se vždy zpracovává v jednu chvíli pouze jeden datový soubor představující jeden dotazník, je přítomný jen jeden znalostní model. Z toho plyne, že je možné jednoduše označit jako rodičovský element jediný existující znalostní model, který se v datech nachází. Hodnota `dsw:order` odpovídá pořadí zpracování objektu. Pro třídy `dsw:Question`, `dsw:Expert`, `dsw:Answer`, `dsw:Choice` a `dsw:Reference` to není možné, neboť jejich rodičovské elementy jsou instance, které se v datovém souboru vyskytují zpravidla ve větším množství.

Rodičovský element třídy `dsw:Question` smí být buď `dsw:Answer`, nebo `dsw:Chapter`, anebo `dsw:ListOfItemsQuestion`. Identifikátor rodičovského elementu je ale možné jednoduše přenést v parametru při průchodu stromovou strukturou znalostního modelu. Při volání procedury, v návrhu nazvané *QuestionsType*, bude přidán parametr *parentUuid*, do kterého bude vložen identifikátor aktuálního elementu, ať už se jedná o kapitolu, odpověď, nebo předcházející otázku. Identifikátor rodiče procedura *QuestionsType* předá dále proceduře, která zpracovává konkrétní typ otázky. Hodnota `dsw:order` odpovídá pořadí průchodu objektů rodičovského elementu, což je díky způsobu zanořování zachováno.

Pro třídy `dsw:Answer`, `dsw:Choice`, `dsw:Expert` a `dsw:Reference` bylo nutné navrhnout jiný přístup. Jelikož pro vypsané třídy jsou rodičovské elementy ze třídy `dsw:Question`, pro `dsw:Answer` konkrétně ze třídy `dsw:OptionsQuestion` a pro třídu `dsw:Choice` jsou ze třídy `dsw:MultiChoiceQuestion`. Jelikož bylo navrhnuto vypisovat ve výsledné RDF reprezentaci jednotlivé instance tříd u sebe, bylo nutné si pro instance výše vyjmenovaných tříd zapamatovat rodičovský element i jejich pořadí v něm už při průchodu stromovou strukturou otázek.

Pro uložení pořadí a identifikátoru rodičovského elementu byly navrhnuty proměnné typu *dictionary*, tedy seznam dvojic hodnota a klíč, zvlášť pro každou třídu. Při průchodu stromovou strukturou otázek jsou ukládány jednotlivé identifikátory instancí výše vyjmenovaných třídy do prvku klíče a identifikátor právě zpracovávané otázky, která představuje rodiče, do prvku

hodnoty. Zároveň do proměnných pro zachycení pořadí je uložen identifikátor právě zpracovávaného objektu a jeho pořadí v rámci rodičovského elementu, který je ve chvíli průchodu stromovou strukturou jednoznačný. Při zpracování datových objektů, představující instance výše vypsaných tříd, je prohledán odpovídající seznam podle identifikátoru zpracovávané instance. Tím je vyhledán identifikátor odpovídající instanci `dsw:Question`, potažmo jejím podtřídám `dsw:OptionsQuestion` či `dsw:ListOfItemsQuestion`, pro získání hodnoty `dsw:hasParent`. Stejně tak je prohledána proměnná zachycující pořadí za účelem získání hodnoty `dsw:order`.

5.3.3 Uložení uživatelů

Aby bylo možné vypisovat ve výsledné RDF reprezentaci jednotlivé instance tříd u sebe, bylo nutné navrhnout proměnnou *dictionary* pro uložení instancí třídy `dsw:User`. Tento návrh byl nezbytný z toho důvodu, že uživatelé, kteří vytvořili odpověď, jsou uloženi pouze u dané instance odpovědi a nelze se k nim ve struktuře dat dostat jinak, než skrz tyto odpovědi. Z tohoto důvodu byla navržena proměnná pro uložení autorů odpovědí v průběhu jejich zpracování. Díky tomu jsou následně k dispozici všichni uživatelé a je možné je vypsat samostatně v oddělené části. Zároveň je zamezeno redundantnímu výpisu uživatelů, neboť v proměnné jsou na místě klíče uloženy *uuid* uživatelů, a tak není možné mít vícero totožných uživatelů v seznamu.

5.3.4 Umělé identifikátory pro report

Pro instance reportů bylo nutné navrhnout umělé identifikátory, protože samy o sobě identifikaci nemají. V rámci reprezentace RDF je ale vhodné na ně odkazovat a nevypisovat je pouze jako *blank node* neboli prázdné uzly. Pro instance třídy `dsw:TotalReportItem` se jedná o řetězec `totalReport` a pro `dsw:ChapterRepotyItem` se identifikátor skládá z řetězce `chapterReport-` doplněný o *uuid* kapitoly, ke které se report vztahuje. Díky tomu, že jejich identifikátor je doplněn prefixem, což je vidět na 5.4, je možné se na ně konkrétně odkazovat i v případě, že by bylo sloučeno několik různých exportů.

Ukázka kódu 5.4: Prefix report:

```
@prefix report: <{{ctx.config.clientUrl}}/projects/
↪  {{ctx.questionnaireUuid}}/metrics/>.
```

Kapitola 6

Implementace šablony pro export dat z dotazníků

Dle návrhu ontologie struktur nástroje DSW a návrhu RDF reprezentace dat byla implementována šablona pro export dat v RDF. Šablona byla vyvíjena přímo v nástroji DSW pomocí tzv. *Document Template Editor*. Při vytváření nového editoru, kde je šablona implementována, je možné vybrat, zda bude editor prázdný nebo již bude obsahovat soubory i konfiguraci vybrané šablony dokumentů.

Původně byla šablona implementována spolu s *Questionnaire Report* [7]. Tato šablona představuje výchozí transformaci ze strukturovaných dat do výsledného dokumentu v několika různých formátech. Následně byl vytvořen samostatný editor pro exportní šablonu do RDF za pomoci Template Development Kit (TDK) [72].

Při implementaci šablony bylo vycházeno jak z vytvořeného návrhu reprezentace tak i z výsledného dokumentu exportní šablony *Questionnaire report* [7] ve formátu JSON. Tato exportní šablona generuje data strukturovaná tak, jak jsou uložena. Ke konkrétnímu objektu se přistupuje pomocí tečkové notace a k přesné implementaci tak bylo možné vycházet ze skladby výsledného dokumentu této exportní šablony.

Prvotně byla šablona implementována v jednom souboru, který generoval všechna data bez ohledu na jejich kontext v rámci nástroje DSW. Po dokončení a validaci byla šablona rozdělena do tří samostatných částí, které odpovídají pomyslným oblastem struktury DSW a to znalostní model, instance dotazníku, odpovědí a uživatelů a oblast doplňujících informací. Pro používané prefixy byl vytvořen samostatný soubor za účelem jednodušší úpravy. Pro každou oblast byl vytvořen soubor obsahující makra a příkazy pro zpracování dané oblasti dat a také šablona, zahrnující pomocí příkazu `include` soubor prefixů a soubor pro zpracování dané oblasti. Na ukázce 6.1 je viditelný kód šablony pro výpis znalostního modelu, ve kterém je zahrnuta šablona pro výpis prefixů a zároveň šablona obsahující makra pro výpis této části nástroje DSW.

Tím vznikly šablony, které umožňují uživatelům zvolit jakou oblast dat chtějí generovat.

Ukázka kódu 6.1: Šablona pro výpis znalostního modelu

```
{#- ------------------- -#}
{#-  INCLUDE PREFIXES   -#}
{#- ------------------- -#}

{%- include 'src/prefixes.ttl.j2' -%}

{#- ------------------- -#}
{#-  INCLUDE MACROS     -#}
{#- ------------------- -#}

{%- include 'src/macros/macro-knowledge-model.ttl.j2' -%}
```

Popis implementace je rozdělen na hlavní část implementace a popis využitých funkcí a procedur. Podrobněji jsou popsány specifické funkce či přístupy v implementaci.

6.1 Hlavní část implementace

V hlavní části implementace jsou postupně zpracovávány instance tříd. V případě, že instance třídy představuje jednodušší datovou strukturu a nevyskytuje se v datech vícekrát, je vypsána rovnou v hlavní části kódu. Tomu tak je v případě tříd `dsw:Document`, `dsw:Organization`, `dsw:ContextConfig`, `dsw:Questionnaire`, `dsw:Report` a `dsw:KnowledgeModel`. Na ukázce 6.2 je kód třídy `dsw:Document`, která se nachází v hlavní části implementace. Vlastnosti instance třídy `dsw:Document` jsou přímo vypsány bez pomoci dodatečné procedury této třídy.

Instance ostatních tříd jsou v hlavní části kódu zpracovány v cyklu *for* a pro jednotlivé instance jsou volány příslušné procedury, které je vypisují. Tomu je tak v případě tříd `dsw:QuestionnaireVersion`, `dsw:Chapter`, `dsw:Metric`, `dsw:Phase`, `dsw:Tag`, `dsw:Question`, `dsw:Expert`, `dsw:Choice`, `dsw:Answer`, `dsw:Reference`, `dsw:Integration`, `dsw:Reply` a `dsw:ReportItem`. Na ukázce kódu 6.3 je část, která v cyklu *for* prochází všechny existující instance třídy `dsw:Choice` a volá vypisující proceduru `renderChoice`.

V případě abstraktních tříd, které se dále dělí do specifičtějších podtypů, se v hlavní části kódu instance rozdělují podle jejich typu a jsou volány procedury až pro konkrétní podtřídu. Tomu tak je v případě tříd `dsw:Reference`, `dsw:Integration` a `dsw:Reply`. Na ukázce kódu 6.4 je zpracování instancí

Ukázka kódu 6.2: Zpracování třídy `dsw:Document`

```
document:{{ctx.uuid}} a dsw:Document ;
  dsw:uuid "{{ctx.uuid}}" ;
  dsw:createdAt "{{ctx.createdAt}}"^^xsd:dateTime ;
  dsw:updatedAt "{{ctx.updatedAt}}"^^xsd:dateTime ;
  dsw:hasKnowledgeModel km:{{ctx.knowledgeModel.uuid}} ;
  dsw:hasOrganization
↪   organization:{{ctx.organization.organizationId}} ;
  dsw:hasContextConfig ctxConfig:contextConfig ;
  dsw:hasQuestionnaire questionnaire:{{ctx.questionnaireUuid}}
↪   ;
  dsw:hasReport report:{{ctx.report.uuid}} .
```

Ukázka kódu 6.3: Zpracování třídy `dsw:Choice`

```
{%- for choiceUuid in ctx.knowledgeModel.entities.choices -%}
  {%- set choice =
↪   ctx.knowledgeModel.entities.choices[choiceUuid] %}

{{ renderChoice(choice, loop.index) }}
{%- endfor -%}
```

třídy `dsw:Reply`. Zde se instance dělí podle hodnoty v atributu s názvem `reply.value.type` a dle něj jsou odpovědi dále zpracovány příslušnou procedurou.

Pro instance podtříd abstraktní třídy `dsw:Question` je způsob zpracování odlišný, jelikož je v rámci průchodu jejich stromovým uspořádáním potřeba každou otázku zpracovat podle jejího typu. To znamená, že v hlavní části kódu se jednotlivé instance nerozdělují, ale volá se procedura `QuestionType`, která rozlišuje otázky dle typu pro následné zpracování. V případě zanoření ve struktuře do nové otázky se vždy volá tato procedura, což by v případě rozdělení v hlavní části implementace nebylo možné.

Původně návrh zahrnoval výpis všech instancí jedné třídy u sebe, což je v implementaci dodrženo až na instance jednotlivých typů otázek a odpovědí. Je to z toho důvodu, že všechny typy otázek jsou procházeny v jejich stromové struktuře tak, jak jsou uloženy ve znalostním modelu. Odpovědi jsou v datové struktuře také uloženy bez rozdělení. Proto se ve výsledné reprezentaci vyskytují v jedné oblasti bez ohledu na jejich typ.

Ukázka kódu 6.4: Zpracování třídy `dsw:Reply`

```
{%- for replyPath in ctx.questionnaireReplies -%}
  {%- set reply = ctx.questionnaireReplies[replyPath] -%}
  {%- if reply.value.type == "StringReply" %}

  {{renderStringReply(replyPath, reply, loop.index)}}
  {%- elif reply.value.type == "ItemListReply"%}

  {{renderItemListReply(replyPath, reply, loop.index)}}
  {%- elif reply.value.type == "IntegrationReply"%}

  {{renderIntegrationReply(replyPath, reply, loop.index)}}
  {%- elif reply.value.type == "MultiChoiceReply"%}

  {{renderMultiChoiceReply(replyPath, reply, loop.index)}}
  {%- elif reply.value.type == "AnswerReply"%}

  {{renderAnswerReply(replyPath, reply, loop.index)}}
  {%- endif %}
{%- endfor %}
```

Dále nebyla dodržena posloupnost vlastností tříd tak, jak jsou zanesené v modelu návrhu ontologie. Je to z toho důvodu, že jsou využity zjednodušené zápisy syntaxe Turtle a ty vyžadují určitá znaménka na konci řádku, což je popsáno v 3.5.2. Jelikož se ve struktuře vyskytují vlastnosti, které nejsou povinné, bylo užitečnější přidat na konec výpisu instance třídy vždy vlastnost povinnou, která bude zakončena tečkou.

6.2 Makra

Mimo hlavní část byly implementovány i funkce a procedury neboli makra, které obsahují opakovaně využívané bloky kódu. V implementaci šablony se makra, která jsou více popsány v částech 6.2.1 a 6.2.2, většinou pojí s konkrétní existující třídou.

Bylo definováno i obecné makro *makeList*, které se používá při výpisu instancí různých tříd. Jak plyne z analýzy a je patrné i z návrhu ontologie, mnoho objektů disponuje atributy s datovým typem *list*. V RDF reprezentaci dat pro tyto atributy vznikne predikát s vícero objekty. Aby nebylo nutné vytvářet cyklus *for* pro procházení hodnot ve všech výskytech těchto vlastností, bylo vytvořeno makro *makeList*.

Makro *makeList* 6.5 přijímá jako vstupní hodnotu seznam objektů, které je třeba zpracovat a dále vstupní textový řetězec `strBefore` udávající co by mělo být dosazeno před hodnotu objektu, a `strAfter`, který určuje co by mělo být vloženo za ní. Ve vstupním seznamu se totiž mohou vyskytovat dva typy hodnot. V prvním případě se jedná o odkazy IRI v formě `uuid` instancí jiných tříd, jako je například vlastnost `dsw:containsQuestion` třídy `dsw:Chapter`. V takovém případě je nutné přidat před hodnotu prefix `km:`. Na druhou stranu jiné seznamy, také využívající *makeList*, mohou obsahovat pouze literály, které musí být opatřeny uvozovkami a případně definicí datového typu. V takovém případě proměnné `strBefore` a `strAfter` obsahují znak uvozovek a případně `strAfter` může obsahovat i definici datového typu.

Ukázka kódu 6.5: Makro pro konstrukci seznamu hodnot

```
{%- macro makeList(list, strBefore, strAfter) -%}
  {%- set resultList = [] -%}
    {%- for item in list -%}
    {%- do resultList.append(strBefore+item+strAfter) -%}
    {%- endfor -%}
    {{resultList|join(", ") }}
{%- endmacro -%}
```

V této funkci je pomocí cyklu *for* každá položka vstupního seznamu opatřena textovými řetězci, které mají být vloženy před a za každou hodnotu. Následně je položka vložena do výsledného seznamu. Na konci funkce je vrácen výsledný seznam s hodnotami oddělenými čárkami.

Volání této funkce je tedy nejčastěji ve dvou případech. Ta první je ve chvíli, když je zapotřebí konstrukce seznamu hodnot *object property*, které před každou hodnotou mají odpovídající prefix. Na ukázce kódu 6.6 je část makra pro `dsw:Chapter` a volání funkce `renderList` za účelem konstrukce seznamu všech otázek, které kapitola obsahuje. Jedná se o hodnoty *object property* `dsw:containsQuestion`. Makru `makeList` je předán seznam všech obsažených otázek `chapter.questionUuids` a dále textový řetězec, který se má vyskytovat před každou hodnotou, což je `km`. Textový řetězec za hodnotu není třeba neboť se jedná o *object property*.

Makro třídy `dsw:Chapter` ukládá výstup makra `makeList`, otestuje zda v seznamu jsou nějaké hodnoty a pokud ano, tak je vypíše spolu s predikátem `dsw:containsQuestion`.

Druhá možnost užití je v případě, kdy se jedná o seznam hodnot *data property* a tam je potřeba hodnoty ohraničit uvozovkami a případně doplnit i jejich datový typ. Na ukázce 6.7 je volání uvnitř makra pro zpracování třídy `dsw:MultiChoiceReply`. Makru `renderList` je předán seznam všech možností

Ukázka kódu 6.6: Volání makra `renderList` uvnitř makra pro `dsw:Chapter`

```
{%- set questions = makeList(chapter.questionUuids, "km:", "")
↪  -%}
  {%- if questions|length > 0 %}
  dsw:containsQuestion {{ questions.split(',')|join(", ")}} ;
  {%- endif %}
```

`dsw:Choice`, které jsou v odpovědi zvoleny. Jedná se o *data property* obsahující identifikátory zvolených `dsw:Choice` ve formě textového řetězce. Z tohoto důvodu se hodnoty doplňují o uvozovky.

Ukázka kódu 6.7: Volání makra `renderList` uvnitř makra pro `dsw:MultiChoiceReply`

```
{%- set choicesUuid = makeList(reply.value.value, '"', '"') -%}
  {%- if choicesUuidPrefix|length > 0 %}
  dsw:replyValue {{choicesUuid.split(',')|join(", ")}} .
  {%- endif %}
```

6.2.1 Makra tříd

Ostatní makra se vážou ke konkrétním existujícím třídám či vlastnostem, jejichž instance se vyskytují v množině, a tak je využití maker vhodné. Tato makra jsou definována na začátku souboru šablony s názvem *render* doplněným o název třídy, kterou vypisují. Na ukázce 6.8 je uvedeno makro vypisující třídu `dsw:Choice`.

V ukázce je viditelný výpis jednotlivých vlastností třídy i získání jejich hodnot za pomoci tečkové notace ze struktury uložených dat. Pro vlastnost `dsw:annotation` je viditelné využití dalšího makra *renderAnnotation* vypisujícího anotace dané `dsw:Choice`. Pro výpis hodnot vlastností `dsw:hasParent` a `dsw:order` jsou využívány proměnné obsahující identifikaci rodičovského elementu a pořadí uvnitř něj, což je popsáno v návrhu RDF reprezentace v části 5.3.2.

6.2.2 Makra abstraktních tříd

Některé abstraktní třídy dále rozdělené do několika podtříd mají také svá makra zprostředkující výpis jejich vlastností. Jedná se o abstraktní třídy

Ukázka kódu 6.8: Makro pro třídu `dsw:Choice`

```
{%- macro renderChoice(choice, index) -%}
km:{{choice.uuid}} a dsw:Choice ;
  dsw:uuid "{{choice.uuid}}" ;
  dsw:title {{choice.label|tojson}} ;
  {%- if choice.annotations %}
  {{ renderAnnotation(choice.annotations) }} ;
  {%- endif %}
  dsw:hasParent
↪   km:{{multiChoiceQuestionChoiceUuidParent[choice.uuid]}} ;
  dsw:order "{{choiceOrder[choice.uuid]}}"^^xsd:integer .
{%- endmacro -%}
```

`dsw:Question`, `dsw:Report`, `dsw:Reference` a třídu `dsw:Integration`. Každá z těchto abstraktních tříd má vlastnosti, které jsou děděny do jejich podtříd. Proto je vhodné využít společných procedur za účelem zpracování těchto společných vlastností, aby se zabránilo opakování stejných příkazů v kódu.

Na ukázce 6.9 je uvedena procedura abstraktní třídy `dsw:Reference` i jejích podtříd `dsw:URLReference` a `dsw:ResourcePageReference`. V rámci hlavní části kódu, která není zahrnuta v ukázce, dochází k procházení referencí pomocí cyklu *for*. Pro každou referenci je poté volána procedura dle jejího typu, buď `URLReference`, nebo `ResourcePageReference`. Tyto procedury nejdříve vypisují označení objektu a určují jejich typy, následně pak volají proceduru `renderReference`, která vypisuje sdílené vlastnosti. Po návratu do makra podtřídy jsou vypsány vlastnosti specifické pro danou podtřídu. Tento postup umožňuje snadné zpracování referencí a minimalizaci opakovaných úkonů.

6.3 Specifické komponenty

V rámci implementace byly vytvořeny specifické komponenty buď v souladu s požadavky na reprezentaci RDF, nebo s cílem zlepšit přehlednost či jednoduchost kódu. Tato sekce popisuje tyto komponenty a jejich klíčové charakteristiky.

6.3.1 URL Integration Reply

Odpovědi typu *Integration*, konkrétně *IntegrationTypeIntegrationReply* mají vlastnost `dsw:integrationReplyValue` představující přesnou URL zvolené odpovědi. Ta ale v datech není nikde specificky definována. Je třeba ji složit

Ukázka kódu 6.9: Makra třídy `dsw:Reference` i jejích podtříd

```
{%- macro renderReference(reference) -%}
  dsw:uuid "{{reference.uuid}}" ;
  {%- if reference.annotations -%}
  {{ renderAnnotation(reference.annotations) }} ;
  {%- endif %}
  dsw:hasParent
↪   km:{{questionReferenceUuidParent[reference.uuid]}} ;
  dsw:order "{{referenceOrder[reference.uuid]}}"^^xsd:integer ;
{%- endmacro -%}

{%- macro renderURLReference(reference, index) -%}
km:{{reference.uuid}} a dsw:URLReference ;
  {{ renderReference(reference)}}
  dsw:title {{reference.label|tojson}} ;
  dsw:referenceUrl "{{reference.url}}"^^schema:URL .
{%- endmacro -%}

{%- macro renderResourcePageReference(reference, index) -%}
km:{{reference.uuid}} a dsw:ResourcePageReference ;
  {{renderReference(reference)}}
  dsw:shortUuid "{{reference.shortUuid}}" .
{%- endmacro -%}
```

z URL využívané integrace, tedy externího zdroje, a z identifikátoru konkrétní položky externího zdroje. Identifikátor položky je uložen v datovém objektu hodnoty odpovědi *reply.value.value.id*. Za pomoci vestavěné funkce *replace* je možné nahradit proměnnou `$id` v adrese externího zdroje za identifikátor zvolené položky, což je viditelné na ukázce kódu 6.10.

Ukázka kódu 6.10: Konstrukce odpovědi typu *Integration*

```
dsw:integrationReplyValue "{{integration.itemUrl |
↪   replace("${id}", reply.value.value.id )}}"^^schema:URL ;
```

6.3.2 User

Třída `dsw:User` v návrhu vznikla spojením entit *User* a *Simple Author*. Tímto spojením vzniklo ve třídě `dsw:User` několik nepovinných vlastností, které patřili původně entitě *User*, ale pro instance vycházející z entity *SimpleAuthor* nejsou definovány. Z tohoto důvodu by v případě vytvoření pouze jedné procedury vypisující instance `dsw:User` přineslo zbytečně několik *if-else* podmínek. Navíc v rámci jedné instance dotazníku znalostního modelu existuje pouze jedna instance původní entity *User* a to uživatel, který projekt vytvořil. Všichni ostatní uživatelé jsou řazeni k *Simple Author*.

Z tohoto důvodu byl implementován výpis třídy `dsw:User` podle toho, zda se jedná o uživatele, který vytvořil projekt čili instance bývalé *User*, anebo se jedná o uživatele spadajícího k původní *Simple Author*. Na ukázce kódu 6.11 je viditelná část pracující s uživateli, kteří nevytvořili projekt, tedy instance bývalé *Simple Author*. Na řádce 2 je podmínka, zamezující zpracování uživatele patřící k bývalé *User*, který disponuje více vlastnostmi.

V kódu, který není součástí ukázky, je následně zpracován uživatel, který vytvořil projekt. Tedy ten, který je instancí bývalé *User* se všemi jeho vlastnostmi.

Ukázka kódu 6.11: Výpis instancí třídy `dsw:User` původně patřící k *Simple Author*

```
{%- for simpleAuthorsItem in simpleAuthors -%}
{%- if simpleAuthorsItem != ctx.createdBy.uuid -%}
{%- set simpleAuthor = simpleAuthors[simpleAuthorsItem] -%}
user:{{simpleAuthor.uuid}} a dsw:User;
  dsw:uuid "{{simpleAuthor.uuid}}" ;
  dsw:firstName {{simpleAuthor.firstName|tojson}} ;
  dsw:lastName {{simpleAuthor.lastName|tojson}} ;
  {%- if simpleAuthor.imageUrl %}
  dsw:imageUrl "{{simpleAuthor.imageUrl}}"^^schema:URL ;
  {%- endif -%}
{%- endif -%}
{%- endfor -%}
```

6.4 Validace RDF exportu

Výsledná reprezentace dat testovací instance v RDF byla validována za účelem dosažení plně korektního zápisu RDF. Z důvodů velkého objemu nebyla kompletně testována data pocházející z rozsáhlých znalostních modelů a dotazníků.

Pro účely testování byl vytvořen jednodušší znalostní model, který ale obsahuje všechny druhy komponent. Prvotně byl výsledek exportu validován v nástroji *Apache Jena Fuseki* [73]. Avšak jeho užití pro kontrolu validace není zcela vhodné, neboť umístění chyby zobrazí pouze jako číslo řádku a neuvede její přesnější popis.

Z tohoto důvodu bylo od tohoto nástroje upuštěno a byl využit on-line nástroj *RDFShape* [74]. Tento nástroj validuje zapsaná RDF data oproti schématu RDF zapsanému v jazyce ShEx. S jeho pomocí byly opraveny všechny nepřesnosti v zápisu.

6.5 Způsob publikace šablony

Všechny dostupné šablony dokumentů i znalostní modely se nachází v registru [6] nástroje DSW. Tento registr poskytuje možnost vyhledávání, uložení a následného importu do nástroje DSW. Tyto šablony i jejich verze jsou spravovány prostřednictvím nástroje *GitHub* [64]. Na této platformě je registrována organizace *Data Stewardship Wizard*, která slouží jako centrální místo pro uchování a správu relevantních repozitářů. Jedním z těchto repozitářů je například *questionnaire-report-template*, ve kterém je spravována šablona dokumentu s názvem *Questionnaire Report* [7].

Pro implementovanou šablonu byl vytvořen repozitář *rdf-report-template*. Tento repozitář obsahuje kód šablony ve složce `src`, kde jsou publikovány jednotlivé šablony generující data do RDF podoby s ohledem na oblast dat v rámci nástroje DSW. Repozitář obsahuje dokument `README` s popisem a dokument s názvem `LICENSE`, který obsahuje celé znění licence, pod kterou je šablona publikována.

Všechny dostupné šablony dokumentů nástroje jsou publikovány pod licencí *Apache License 2.0* [75], která umožňuje sdílení, modifikaci a použití jak pro osobní tak komerční účely. Proto i implementovaná šablona dokumentů byla publikována pod touto licencí.

6.6 Nalezené chyby a nepřesnosti v nástroji DSW

Během procesu návrhu ontologie struktur DSW a následné implementace šablony bylo zjištěno několik chyb v nástroji DSW. Primárně se jednalo o chyby v oblasti reprezentace a uložení dat, což je oblast, která byla nejvíce prozkoumána. Některé chyby v aktuální verzi (3.21.0) mají přímý vliv i na výsledek exportní šablony, a proto jsou zde blíže popsány.

V datovém souboru, ve formátu JSON, se vyskytují objekty typu *Answer*, které ale mají existovat, dle jejich obsahu, pouze jako objekty typu *Choice*. Tyto mylné objekty nemají příslušnou otázku typu *OptionsQuestion* a navíc jejich název je totožný se správně existující *Choice*. V datovém souboru znalostního modelu *Common DSW Knowledge Model* bylo identifikováno celkem

21 takových objektů. Tento problém má významný dopad na správnost exportovaných dat v RDF.

Další chybou, která má významný vliv na správnost exportovaných dat ve formátu RDF, je přetrvávající definice fáze (`dsw:hasPhase`), ve které se momentálně nachází dotazník (`dsw:Questionnaire`). I po smazání dané fáze (`dsw:Phase`) hodnota atributu `dsw:hasPhase` třídy `dsw:Questionnaire` stále přetrvává a tím pádem RDF data odkazují na neexistující instanci. I tato chyba má tedy vliv na korektnost datové reprezentace.

Poslední nepřesnost v návrhu znalostního modelu nelze s ohledem na charakter editoru považovat za chybu nástroje. Nicméně je nutné tuto nepřesnost vzít v potaz, neboť může mít vliv na správnost exportovaných dat. Konkrétně se jedná se o několik atributů, které jsou nepovinné a jsou v ontologii vyjádřeny jako `dsw:usesIntegration` třídy `dsw:IntegrationQuestion`, `dsw:title` a `dsw:url` třídy `dsw:URLReference` a většina vlastností třídy `dsw:Integration` i jejích podtříd. V souladu s principy tvorby znalostních modelů v editoru jsou tyto atributy správně definovány jako nepovinné. Pokud nejsou vyplněny, je uživatel pouze upozorněn, ale může i nadále uložit a publikovat znalostní model. Problém nastává v datech, kde neúplná definice může vést k nekorektním výstupům. Například pro instanci třídy `dsw:IntegrationQuestion` pak není definovaná integrace, kterou využívá. V exportní šabloně byl i tak ponechán výpis těchto vlastností, které se tak vypisují prázdné. Výjimkou je zmíněná vlastnost `dsw:usesIntegration` třídy `dsw:IntegrationQuestion`, která by vedla na neexistující integraci, což by způsobilo nekorektnost dat. V případě prázdné `dsw:usesIntegration` se tato vlastnost nevypisuje.

Kapitola 7

Přínosy a zhodnocení využití RDF reprezentace dat

Tato kapitola se zaměřuje na zhodnocení přínosů využití RDF reprezentace dat z dotazníku a to jak z pohledu uživatelů nástroje DSW, tak z pohledu následného vyhodnocení dat.

Oproti již existující šabloně *Questionnaire Report* [7] je nově implementovaná šablona odstíněna od skutečné struktury uložení dat. Šablona *Questionnaire Report* generuje data v totožné struktuře jako jsou uložena ve formátu JSON. Při změně struktury uložení se tak změní i struktura exportovaných dat, což může zapříčinit další nutné modifikace. Naopak export dat v RDF, který je založen na stálé ontologii, je abstrahován od struktury uložení.

RDF přináší mnoho výhod, data jsou strojově zpracovatelná, propojitelná a je možné nad nimi vytvářet dotazy a analyzovat je. Zásadní přínosy i objevené nevýhody jsou dále popsány.

7.1 Sémantický význam

RDF reprezentace dat přináší významné výhody a přínosy z pohledu interpretace a porozumění datům. Jedna z klíčových výhod je možnost obohatit data o sémantický význam pomocí odkazů na existující ontologie. To přináší jednoznačnou interpretaci dat, což eliminuje možnost nesprávného vyhodnocení či určení významu. Nemělo by se tím pádem stávat, že během vyhodnocení je datový objekt mylně považován významově za odlišný, než skutečně je.

Pro nástroj byla vytvořena kompletní ontologie jednotlivých struktur znalostního modelu i částí potřebných pro tvorbu dotazníku i jeho výsledného dokumentu. Tím je zajištěno, že význam jednotlivých datových objektů je v souladu s jejich zamýšleným sémantickým významem a minimalizuje se riziko nedorozumění.

Vytvořená ontologie i ontologický model navíc může usnadnit porozumění

rolím jednotlivých prvků v rámci nástroje i znalostního modelu. Ontologie poskytuje kompletní znalosti o prvcích nástroje, což pomáhá uživatelům při orientaci ve struktuře znalostních modelů a dalších prvcích potřebných pro tvorbu výsledného dokumentu dotazníku.

Je však nutno říci, že se jedná pouze o zachycené sémantické významy a znalosti z hlediska uživatele nástroje. Struktura a přesný způsob uložení dat nemusí přesně odpovídat ontologii nebo ontologickému modelu.

7.2 Propojitelnost a otevřenost dat

Díky tomu, že byla implementována exportní šablona dat do reprezentace RDF s využitím odpovídající ontologie, která navíc byla obohacena o propojení s existujícími ontologiemi, jsou data v propojitelné i otevřené formě. Propojitelnost dat umožňuje jejich spojení s dalšími zdroji informací, anebo s dalšími exporty dat z nástroje DSW. To je přínosné zejména v případě rozsáhlejšího dotazování nad různými soubory dat s cílem provést komplexní analýzu a vyhodnocení. Propojitelnost je zajištěna pomocí jedinečných identifikátorů jednotlivých datových objektů, díky kterým při propojování nebude docházet ke kolizím v rámci identifikace.

Otevřenost výsledné formy dat je zaručena použitím otevřených standardů a specifikací, jako je RDF, RDFS, OWL i formát Turtle. Díky tomu není výsledná reprezentace dat závislá na konkrétních technologiích a umožňuje snadné sdílení dat mezi různými nástroji a systémy za účelem jejich vyhodnocení.

To spolu s přesně definovaným významem dat podporuje interoperabilitu ve smyslu možnosti spolupráce různých systému nad jednou reprezentací dat. Různé systémy či nástroje mohou dohromady komunikovat a pracovat s daty, bez rizika nepřesné interpretace a bez nutnosti zohledňovat specifické technologie a platformy, které využívají.

Tento přínos využití RDF pro reprezentaci dat je zejména výhodný při dotazování nad daty, jejich analýzou, vyhodnocením a například i jejich validací. Díky tomu, že pro nástroj byla vytvořena ontologie, je přesně definováno jaká data splňují požadavky na validní znalostní model a která nikoliv. Byť nástroj DSW upozorní na nepřesnost ve znalostním modelu, i tak je možné jej publikovat a exportovaná RDF data tak nemusí plně odpovídat navržené ontologii. Na příklad se může jednat o definici integrace v případě otázky typu *Integration*. V DSW ontologii je vyžadována, ale editor znalostního modelu jí dovoluje i bez její definice publikovat. Z tohoto důvodu by bylo vhodné využít jazyk ShEx [51] pro specifikaci očekávané struktury dat plynoucí z ontologie za účelem následné validace exportovaných RDF dat oproti této specifikaci. Díky tomu, že byly využity otevřené standardy a specifikace, by bylo možné data takto kompletně validovat.

7.3 Dotazování

Reprezentace dat v RDF přináší možnosti dotazování a vyhledávání v datech na základě jejich vlastností a vztahů, které jsou definované ontologií. Jedná se o proces získávání informací, ať už ve formě vyhledávání trojic, nebo vyhledávání za použití ontologií, kde je možné využít sémantických vazeb, vlastností i typů.

Existuje dotazovací jazyk SPARQL Protocol and RDF Query Language (SPARQL) [76] pro reprezentaci RDF, který umožňuje efektivní zpracování dotazů a získání relevantních informací. S jeho pomocí lze formulovat dotazy, které vyhledávají konkrétní vzory trojic pro nalezení přesných záznamů RDF. Je možné kombinovat podmínky vyhledávání, jako například přesné hodnoty, kterých má vlastnost hledané instance nabývat. Přináší to možnost definovat filtry nebo data agregovat. Dotazování přináší rozsáhlejší využití uložených znalostí.

7.4 Udržitelnost a rozšiřitelnost

RDF reprezentace umožňuje flexibilně zařadit změny jak v ontologii, tak i ve výsledné reprezentaci. Zároveň je ale třeba vzít v potaz, že změnu je nutné zařadit hned do několika objektů.

V případě změny ve struktuře uložení dat se jednoduše modifikuje pouze příslušná část v exportní šabloně a další objekty to nezasáhne. Pokud je ale změna na úrovni ontologie, je potřeba dílčích úprav několik. Nejdříve je potřeba upravit samotnou ontologii DSW, kterou je následně vhodné ověřit jak ve validátoru syntaxe Turtle, tak například v nástroji OOPS! [50].

Následuje úprava ontologického modelu tak, aby zapsané ontologii odpovídal. Nakonec je nezbytné zařadit změnu i do exportní šablony. Výsledná RDF data je také vhodné validovat za účelem ověření správnosti zápisu.

Kapitola 8

Zhodnocení možnosti tvorby šablon pro DSW

Tvorba exportních šablon pro nástroj DSW je možná dvěma způsoby. První možností je *Document Template Editor*, který se nachází přímo v nástroji. Druhou možností je implementovat šablonu lokálně v textovém editoru, či vývojovém prostředí, a následně ji nahrát do instance nástroje. V průvodci DSW [24] existuje podrobný popis, jak nastavit lokální vývojové prostředí i editor přímo v nástroji.

Průvodce [24] poskytuje poměrně širokou škálu příkladů kódu v jazyce Jinja2, neboť jsou dostupné již vytvořené exportní šablony v registru [6]. Pro pochopení a ukázku je nejvhodnější šablona *Questionnaire Report* [7], jak je i v průvodci doporučeno, jelikož není spojena s žádným konkrétním znalostním modelem a exportuje všechny informace. Lze si na ní prohlédnout možné postupy zpracování dat.

Průvodce také popisuje strukturu uložení dat, a tedy jakým způsobem lze data zpracovávat a přistupovat k nim. Struktura uložení znalostního modelu na první úrovni obsahuje podobjekt zvaný *entities* a pouze `uuid` identifikátory prvků. Prvky jsou rozdělené do seznamů podle jejich typů, kterým je znalostní model rodičovský element. Obsahuje tedy například kapitoly, integrace, fáze nebo metriky. Konkrétní datové objekty se pak nacházejí až v objektu *entities*, kde se ale nacházejí i prvky, které nemají jako rodičovský prvek znalostní model. Nachází se tam i otázky nebo například experti, ale i kapitoly nebo třeba fáze. To na první pohled může působit chaoticky, neboť to vypadá, že jsou využity dva přístupy ke struktuře uložení. Po bližším průzkumu je ale zřejmé, že identifikátory prvků se vždy vyskytují u rodičovského elementu, čímž znalostní model pro značnou část prvků je. Tím, že je ale do struktury identifikátorů přimíchaná část *entities*, obsahující všechny prvky bez ohledu na jejich rodičovský element, to působí nekonzistentně. Z důvodů složitosti propojenosti prvků znalostního modelu je těžké navrhnout srozumitelnější strukturu. Jasnější struktura by mohla obsahovat objekt *entities*

speciálně pro každou skupinu prvků stejného typu, které mohou figurovat jako rodičovské elementy a v tomto objektu mít plnohodnotné prvky, které jsou potomci prvků tohoto typu. Například *Answer* neboli odpověď má rodičovský prvek konkrétně typu *OptionsQuestion* a ne obecné otázky. Proto by bylo potřeba se vypořádat i s rozdělením otázek dle jejich typů, což by přineslo jen větší složitost struktury uložení.

V průvodci je popsáno i několik předdefinovaných Jinja2 filtrů, které ulehčí implementaci exportní šablony. Například filtr pro poskládání `path` odpovědi z vložených `uuid` identifikátorů, což velmi zjednoduší samotnou implementaci.

Pro rychlejší implementaci exportní šablony by bylo vhodné zvážit vytvoření základního a jednoduchého testovací projektu, jehož data by bylo možné využít v začátcích implementace. Vytvořit dostatečně složitý znalostní model, jen pro účely testování implementované šablony, může být poměrně časově náročné. Lze sice využít existující znalostní modely, ze kterých je možné vytvořit dotazník, do kterého se vyplní testovací data, ale všechny dostupné znalostní modely obsahují velké množství prvků, což pro počáteční validaci nemusí být vhodné. Samozřejmě každá šablona může mít různá využití, a tedy i požadavky na testovací data. Ale pro prvotní ověření správnosti exportní šablony by existující testovací podklady byly výhodou.

Dále jsou zhodnoceny existující možnosti tvorby exportních šablon.

8.1 Tvorba šablony v nástroji DSW

Přímo v nástroji existuje část *Document Template Editor*, ve které je možné implementovat či upravovat exportní šablony. Každý editor obsahuje část pro vývoj šablony, která obsahuje implementovaný kód. Editor zvýrazňuje syntaxi Jinja2 a díky tomu je zapsaný kód lépe čitelný.

Další část editoru je náhled, ve kterém je možné zvolit z jakého konkrétního projektu data v náhledu budou pocházet a jaký konkrétní formát bude v náhledu použit. V případě, že je v implementaci šablony chyba, tak se zobrazí hláška právě v této části s náhledem. Chybová hláška vypíše stručný popis chyby a číslo řádku, na kterém se chyba nachází a to v případě že se nachází na konkrétním řádku. Tento přístup není zcela přívětivý, neboť uživatel je nucen přejít zpět do části kódu a řádek s chybou manuálně vyhledat. Přínosnější by byla možnost validace přímo v části s implementací a případně i automatické nalezení chybové řádky, pokud se chyba týká konkrétního řádku.

Poslední částí editoru je nastavení obsahující jak obecné informace, tak definované formáty. Postup pro správné vyplnění je i s vysvětlením dobře popsán v průvodci nástroje.

Dalším zlepšením by mohlo být zachování přesného místa zobrazení kódu. Při implementaci totiž uživatel přeskakuje z jedné části editoru, kde je implementovaný kód, k části náhledu, aby provedl kontrolu správnosti chtěného výstupu. Díky tomu, že v část implementace je vždy zobrazen začátek souboru,

tak je uživatel nucen zpětně nalézt místo, kde momentálně kód upravoval či psal. Případným řešením by mohlo být spojení části náhledu a implementace na jednu obrazovku.

Při implementaci kódu přímo v nástroji za využití *Google Chrome* (verze 113.0.5672.127)[6] prohlížeče, je generování náhledu pro velký objem dat pomalé. Daleko lepší výsledky v tomto ohledu poskytuje *Microsoft Edge* (verze 113.0.1774.57)[7] prohlížeč. Největší rozdíl v délce trvání generování náhledu mezi zmíněnými prohlížeči dosahuje až 3 vteřiny, v průměru se jedná přibližně 1 vteřinu. Délka trvání byla měřena pomocí *devconsole* a v prohlížeči *Google Chrome* (verze 113.0.5672.127) doba generování dosahuje 3-6 vteřin, přičemž v prohlížeči *Microsoft Edge* (verze 113.0.1774.57) do 3 vteřin.

8.2 Tvorba šablony pomocí TDK

TDK [72] je nástroj příkazové řádky, který je vyvinut pro lokální implementaci exportní šablony. Všechny požadavky i přesná instalace a nastavení je popsáno v průvodci nástroje včetně podrobného video-návodu. V průvodci se též vyskytuje popis postupu lokálního vývoje šablony. V příkazové řádce lze vytvořit novou šablonu, stáhnout šablonu z existující instance DSW nebo ji tam naopak nahrát.

Mezi výhody využití TDK je zcela jistě možnost implementace kódu bez nutnosti internetového připojení a možnost vytváření verzí implementovaného kódu i správa těchto verzí za pomoci příslušných nástrojů, jako je například *GitHub* [64].

Další výhodou je vytvoření nové exportní šablony v příkazové řádce, neboť na základě komunikace nástroje jsou zároveň popsány chtěné formáty a metadata, které se v on-line editoru nástroje nachází v sekci *Formats*.

Nicméně, pro zobrazení náhledu šablony s reálnými daty, je potřeba implementovanou šablonu nahrát do existující instance nástroje DSW za pomoci příkazové řádky. Následně ve webovém rozhraní nástroje je třeba dohledat importovanou šablonu a v editoru přejít na část náhledu. Až zde je pak možné zjistit výsledek exportu šablony, případně zda šablona neobsahuje nějakou chybu.

I v tomto případě by bylo užitečné z pohledu uživatele mít možnost validovat správnost zápisu implementované šablony lokálně. I jednoduchý náhled za pomoci velmi obecné a testovací instance dat by přispěl k jednoduššímu i rychlejšímu vývoji šablony.

[6] https://www.google.com/chrome/
[7] https://www.microsoft.com/cs-cz/edge

Závěr

Cílem této práce bylo navrhnout a implementovat exportní šablonu nástroje DSW, která generuje data z vyplněných dotazníků do navržené RDF reprezentace, přičemž je obohacena o sémantický význam z navržené ontologie struktur nástroje DSW, která je také součástí této práce.

Nejdříve byla provedena analýza nástroje DSW se zaměřením na oblast znalostních modelů, dotazníků a možností tvorby exportních šablon. Následovalo obeznámení s technologií RDF a jejich nejvyužívanějších formátů, stejně tak seznámení se specifikacemi jazyků RDFS a OWL pro tvorbu ontologií. Byly popsány známé nástroje a technologie, které se využívají v procesu tvorby ontologie a její dokumentace, i seznámení s jazykem Jinja2 pro tvorbu šablon, který je v nástroji DSW využíván.

Dle provedené analýzy byla navrhnuta ontologie struktur nástroje, která byla nejdříve tvořena v grafické podobě. Bylo navrhnuto několik změn oproti stávající podobě struktur nástroje za účelem dosažení sémanticky jasné a konzistentní ontologie. Také byla diskutována volba použitých notací pro třídy i pro vlastnosti a sjednocení několika nekonzistentních názvů plynoucích z analýzy. Byly vytvořeny abstraktní třídy ontologie za účelem abstrahování a zamezení redundance často využívaných vlastností a vztahů. Po vytvoření stabilního návrhu bylo v ontologii DSW využito celkem 24 pojmů z externích ontologií, na úrovni tříd i vlastností, což přispívá k propojitelnosti s různými zdroji. Zároveň byla ontologie doplněna o inverzní vlastnosti. Výsledná ontologie ve formátu Turtle byla validována v nástrojích *Protégé* [47] spolu s modulem *OntoDebug* [48] a v on-line nástroji OOPS! [50].

V práci bylo analyzováno devět známých ontologií za účelem zjištění jejich způsobů publikací a správy verzí. Výstupy z této omezené analýzy dopomohly k volbě způsobu publikace navržené ontologie DSW.

Byla navržena RDF reprezentace dat získaných z dotazníků nástroje DSW. Nejdříve byla diskutována volba syntaxe reprezentace, která byla nakonec navržena v Turtle. Dále je popsán způsob zápisu v RDF a využití zjednodušení sytnaxe Turtle. Bylo důležité navrhnout jakým způsobem budou data z do-

tazníků zpracována s ohledem na jejich strukturu uložení. Nejdůležitějšími částmi pro podrobný návrh zpracování byla data znalostního modelu, konkrétně popis průchodu stromové struktury otázek dotazníku a vztah rodič-potomek mezi elementy znalostního modelu.

Na základě návrhu byla implementována exportní šablona nástroje DSW v jazyce Jinja2. Šablona byla rozdělena na dílčí části na základě kontextu nástroje a to na oblast znalostního modelu, oblast dotazníku, jeho odpovědí a uživatelů a oblast doplňujících informací dokumentu. V práci jsou popsána využitá makra a konkrétní implementační části. Následně byla výsledná reprezentace dat testovací instance validována v on-line nástroji *RDFShape* [74].

V práci jsou také zmíněny zjištěné chyby v nástroji DSW, na které se narazilo během procesu návrhu ontologie struktur a následné implementace šablony. Byly popsány chyby v aktuálně užívané verzi (3.21.0), které mají přímý vliv na správnost výsledku exportní šablony.

Součástí práce je také zhodnocení přínosů reprezentace dat v RDF, mezi které se řadí obohacení dat o jejich sémantický význam, možnost komplexní analýzy dat a dotazování nad nimi nebo například otevřenost a propojitelnost dat s externími zdroji. Jako jedna z nevýhod byla zmíněna nutnost postupu zařazení změny v rámci ontologie, která vyvolá další změny v modelu ontologie, jejím zápisu i exportní šabloně.

V práci je taktéž část hodnotící možnosti tvorby exportních šablon pro nástroj DSW, která je doplněna o možná zlepšení z pohledu uživatele jak editoru šablon, tak i TDK.

Všechny cíle práce byly naplněny, navíc práce obsahuje analýzu způsobů publikací a správy verzí devíti známých ontologií. V budoucnu by bylo přínosné zvážit tvorbu ontologie nástroje DSW z pohledu dotazníku a jím využívaných částí, spolu se zahrnutím příslušnosti odpovědí přímo k dotazníku. Tím by bylo docíleno možnosti lepší integrace například mezi verzemi dotazníků a jejich obsahu.

Dalším důležitým krokem by mohla být možnost validace exportovaných dat v RDF reprezentaci oproti specifikaci navržené ontologie DSW zapsané v jazyce ShEx [51]. Díky tomu by mohl být plně ověřen výsledný RDF dokument a správnost jednotlivých zapsaných znalostní podle vytvořené ontologie.

Literatura

[1] Smale, N.; Unsworth, K.; Denyer, G.; aj.: The History, Advocacy and Efficacy of Data Management Plans. *bioRxiv*, 2018, doi:10.1101/443499. Dostupné z: https://www.biorxiv.org/content/early/2018/10/17/443499

[2] Pergl, R.; Hooft, R. W. W.; Suchánek, M.; aj.: "Data Stewardship Wizard": A Tool Bringing Together Researchers, Data Stewards, and Data Experts around Data Management Planning. *Data Science Journal*, ročník 18, 2019: str. 59, doi:10.5334/dsj-2019-059.

[3] Manola, F.; Miller, E.; McBride, B.: RDF 1.1 Primer. [online], červen 2014, [cit. 2023-02-23]. Dostupné z: https://www.w3.org/TR/rdf11-primer/

[4] Hooft, R. W. W.: Data Stewardship Mindmap. červenec 2019, doi:10.5281/zenodo.3351949. Dostupné z: https://doi.org/10.5281/zenodo.3351949

[5] DSW Team: Common DSW Knowledge Model. [online], [cit. 2023-03-19]. Dostupné z: https://registry.ds-wizard.org/knowledge-models/dsw:root:latest

[6] DSW Team: DSW Registry. [online], [cit. 2023-05-28]. Dostupné z: https://registry.ds-wizard.org/

[7] DSW Team: Questionnaire Report. [online], [cit. 2023-05-15]. Dostupné z: https://registry.ds-wizard.org/document-templates/dsw:questionnaire-report:latest

[8] DSW Team: Horizon Europe DMP. [online], [cit. 2023-05-15]. Dostupné z: https://registry.ds-wizard.org/document-templates/dsw:horizon-europe-dmp:latest

[9] DSW Team: Horizon 2020 DMP. [online], [cit. 2023-05-15]. Dostupné z: https://registry.ds-wizard.org/document-templates/dsw:h2020-dmp:latest

[10] DSW Team: Science Europe DMP. [online], [cit. 2023-05-15]. Dostupné z: https://registry.ds-wizard.org/document-templates/dsw:science-europe:latest

[11] Wilkinson, M.; Dumontier, M.; Aalbersberg, I. J.; aj.: The FAIR Guiding Principles for Scientific Data Management and Stewardship. *Scientific Data*, ročník 3, březen 2016, doi:10.1038/sdata.2016.18.

[12] GO FAIR: FAIR Implementation Profile. [online], [cit. 2023-04-25]. Dostupné z: https://www.go-fair.org/how-to-go-fair/fair-implementation-profile/

[13] Schultes, E.; Magagna, B.; Hettne, K. M.; aj.: Reusable FAIR Implementation Profiles as Accelerators of FAIR Convergence. In *Advances in Conceptual Modeling*, editace G. Grossmann; S. Ram, Cham: Springer International Publishing, 2020, ISBN 978-3-030-65847-2, s. 138–147.

[14] VODAN Africa: About VODAN Africa. [online], 2023, [cit. 2023-04-25]. Dostupné z: https://www.vodan-totafrica.info/vodan-africa.php?i=1&a=about-vodan-africa

[15] VODAN: VODAN in a BOX documentation. 2020, [cit. 2023-04-25]. Dostupné z: https://docs.vodan.fairdatapoint.org/en/latest/

[16] Basajja, M.; Suchánek, M.; Taye, G. T.; aj.: Proof of Concept and Horizons on Deployment of FAIR Data Points in the COVID-19 Pandemic. *Data Intelligence*, ročník 4, č. 4, 2022: s. 917–937, doi:10.1162/dint_a_00179.

[17] DSW Team: SM Wizard. [online], [cit. 2023-05-15]. Dostupné z: https://smw.ds-wizard.org/

[18] DSW Team: Elixir Software Management. [online], [cit. 2023-05-15]. Dostupné z: https://elixir-europe.github.io/elixir-smp-lesson/

[19] DSW Team: DS Wizard. [online], [cit. 2023-05-15]. Dostupné z: https://ds-wizard.org/

[20] DSW Team: DS Wizard Guide. [online], [cit. 2023-05-15]. Dostupné z: https://guide.ds-wizard.org/en/latest/

[21] Sansone, S.-A.; McQuilton, P.; Rocca-Serra, P.; aj.: FAIRsharing as a community approach to standards, repositories and policies. *Nature biotechnology*, ročník 37, č. 4, duben 2019: s. 358–367, doi:https://doi.org/10.1038/s41587-019-0080-8.

[22] Mons, B.: *Data stewardship for open science: Implementing FAIR principles*. Chapman and Hall/CRC, 2018, ISBN 978-1498753173.

[23] Pallets Project: Jinja2 Template Engine (3.1.x). [online], březen 2022, [cit. 2023-03-26]. Dostupné z: https://jinja.palletsprojects.com/en/3.1.x/

[24] DSW Team: Document Template Development. [online], [cit. 2023-05-15]. Dostupné z: https://guide.ds-wizard.org/en/latest/more/development/document-templates.html

[25] Linked Data Tools Comunity: Tutorial 3: Semantic Modeling. [online], [cit. 2023-02-22]. Dostupné z: https://www.linkeddatatools.com/semantic-modeling/

[26] Šmajs, J.; Krob, J.: *Úvod do ontologie: Bytí, prostor, čas, pohyb, evoluce, struktura, systém, řád, informace, vesmír, kultura, člověk : (Skriptum filoz. fak. MU)*. Masarykova Univerzita Brno, 1994, ISBN 80-210-0879-2. Dostupné z: https://www.phil.muni.cz/fil/eo/skripta/uvod_do_ontologie.pdf

[27] Guizzardi, G.: *Ontological Foundations for Structural Conceptual Models*. Dizertační práce, Centre for Telematics and Information Technology, University of Twente, Enschede, Nizozemsko, leden 2005. Dostupné z: https://ris.utwente.nl/ws/portalfiles/portal/6042428/thesis_Guizzardi.pdf

[28] DCMI Usage Board: DCMI metadata terms. leden 2020, [cit. 2023-02-26]. Dostupné z: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/

[29] Albertoni, R.; Browning, D.; Cox, S.; aj.: Data Catalog Vocabulary (DCAT)-Version 2. [online], 2020, [cit. 2023-02-26]. Dostupné z: https://www.w3.org/TR/owl2-overview/

[30] Brickley, D.; Miller, L.: FOAF Vocabulary Specification. [online], září 2004, [cit. 2023-02-26]. Dostupné z: http://xmlns.com/foaf/0.1/

[31] Schema.org Community Group: Schema.org. [online], [cit. 2023-05-21]. Dostupné z: https://schema.org

[32] Miles, A.; Bechhofer, S.: SKOS simple knowledge organization system reference. [online], srpen 2009, [cit. 2023-05-21]. Dostupné z: https://www.w3.org/TR/skos-reference/

[33] Klyne, G.; Carroll, J. J.; McBride, B.: RDF 1.1 Concepts and Abstract Syntax. [online], únor 2014, [cit. 2023-02-23]. Dostupné z: https://www.w3.org/TR/rdf11-concepts/

[34] Dürst, M. J.; Suignard, M.: Internationalized Resource Identifiers (IRIs). *RFC*, ročník 3987, 2005: s. 1–46, doi:10.17487/RFC3987, [cit. 2023-02-2]. Dostupné z: https://doi.org/10.17487/RFC3987

[35] Peterson, D.; Gao, S.; Malhotra, A.; aj.: W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. [online], květen 2012, [cit. 2023-02-29]. Dostupné z: https://www.w3.org/TR/rdf11-primer/

[36] Vochozka, J.: Resource Description Framework. *Zpravodaj ÚVT MU*, ročník 4, č. 4, 2001: s. 10–14, ISSN 1212–0901. Dostupné z: http://webserver.ics.muni.cz/bulletin/articles/214.html

[37] Brickley, D.; Guha, R.: RDF Schema 1.1. [online], únor 2014, [cit. 2023-05-16]. Dostupné z: https://www.w3.org/TR/rdf-schema/

[38] W3C OWL Working Group: Web Ontology Language (OWL). [online], prosinec 2012, [cit. 2023-02-25]. Dostupné z: https://www.w3.org/OWL/

[39] W3C OWL Working Group: OWL 2 Web Ontology Language Document Overview (Second Edition). [online], prosinec 2012, [cit. 2023-02-2]. Dostupné z: http://www.w3.org/TR/owl2-overview/

[40] W3C OWL Working Group: The Structure of OWL 2 [figure]. [online], [cit. 2023-03-19]. Dostupné z: https://www.w3.org/TR/owl2-overview/OWL2-structure2-800.png

[41] Parsia, B.; Patel-Schneider, P.; Motik, B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). [online], prosinec 2012, [cit. 2023-02-26]. Dostupné z: https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

[42] Gandon, F.; Schreiber, G.: RDF 1.1 XML Syntax. [online], červen 2014, [cit. 2023-02-23]. Dostupné z: https://www.w3.org/TR/rdf11-primer/

[43] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; aj.: Extensible Markup Language (XML) 1.0 (Fifth Edition). [online], 2008, [cit. 2023-03-07]. Dostupné z: https://www.w3.org/TR/xml/

[44] Bray, T.; Hollander, D.; Layman, A.; aj.: Namespaces in XML 1.0 (Third Edition). [online], 2009, [cit. 2023-03-07]. Dostupné z: https://www.w3.org/TR/REC-xml-names/

[45] Beckett, D.; Berners-Lee, T.; Prud'hommeaux, E.; aj.: RDF 1.1 Turtle. [online], únor 2014, [cit. 2023-02-23]. Dostupné z: https://www.w3.org/TR/turtle/

[46] Carothers, G.; Seaborne, A.; Beckett, D.: RDF 1.1 N-Triples. [online], únor 2014, [cit. 2023-02-23]. Dostupné z: https://www.w3.org/TR/n-triples/

[47] Musen, M. A.: The protégé project: a Look Back and a Look Forward. *AI Matters*, ročník 1, č. 4, 2015: s. 4–12, doi:10.1145/2757001.2757003. Dostupné z: https://doi.org/10.1145/2757001.2757003

[48] Schekotihin, K.; Rodler, P.; Schmid, W.; aj.: OntoDebug. [online], srpen 2009, [cit. 2023-05-22]. Dostupné z: https://protegewiki.stanford.edu/wiki/OntoDebug

[49] IDLab - Ghent University: Turtle Validator. [online], 2014–2015, [cit. 2023-05-22]. Dostupné z: https://github.com/IDLabResearch/TurtleValidator

[50] Poveda-Villalón, M.; Suárez-Figueroa, M. C.; Ángel García-Delgado, M.; aj.: OOPS! (OntOlogy Pitfall Scanner!): supporting ontology evaluation on-line. 2009, [cit. 2023-05-22]. Dostupné z: https://www.semantic-web-journal.net/system/files/swj989.pdf

[51] Prud'hommeaux, E.; Boneva, I.; Gayo, J. E. L.; aj.: Shape Expressions Language 2.1. [online], [cit. 2023-05-22]. Dostupné z: http://shex.io/shex-semantics/

[52] Beveridge, A.; Hansen, J. B.; Val, J.; aj.: Validata: RDF Validator using Shape Expressions. [online], [cit. 2023-05-22]. Dostupné z: https://www.w3.org/2015/03/ShExValidata/

[53] Peroni, S.: Live OWL Documentation Environment. [online], [cit. 2023-05-22]. Dostupné z: https://essepuntato.it/lode/

[54] Garijo, D.: WIDOCO: A Wizard for Documenting Ontologies. In *International Semantic Web Conference*, Springer, Cham, 2017, s. 94–102, doi:10.1007/978-3-319-68204-4_9. Dostupné z: http://dgarijo.com/papers/widoco-iswc2017.pdf

[55] Pasin, M.: OntoDoc. [online], [cit. 2023-05-22]. Dostupné z: https://github.com/semanticarts/ontodoc

[56] Martínková, J.; Suchánek, M.: Mapping of Ontologies and Vocabularies Used in Research Data Management and Open Science, leden 2023.

[57] The Dublin Core Metadata Initiative: Dublin Core Specifications. [online], 2023, [cit. 2023-05-27]. Dostupné z: https://www.dublincore.org/specifications/

[58] Creative Commons: Attribution 4.0 International. [online], [cit. 2023-05-27]. Dostupné z: https://creativecommons.org/licenses/by/4.0/

[59] W3C: Software license. [online], 2023, [cit. 2023-05-27]. Dostupné z: https://www.w3.org/Consortium/Legal/2023/software-license.html

[60] Creative Commons: Attribution 1.0 Generic. [online], [cit. 2023-05-27]. Dostupné z: https://creativecommons.org/licenses/by/1.0/

[61] SIOC initiative: SIOC Project. [online], [cit. 2023-05-27]. Dostupné z: http://sioc-project.org/

[62] Ison, J.; Kalaš, M.; Jonassen, I.; aj.: EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*, ročník 29, č. 10, 03 2013: s. 1325–1332, ISSN 1367-4803, doi:10.1093/bioinformatics/btt113, [Accessed 21-Jan-2023]. Dostupné z: https://doi.org/10.1093/bioinformatics/btt113

[63] Ison, J.; Ménager, H.; Kalaš, M.: EDAM - Ontology of bioscientific data analysis and data management. [online], [cit. 2023-05-27]. Dostupné z: http://edamontology.org/page#

[64] Preston-Werner, T.; Wanstrath, C.; Hyett, P. J.; aj.: GitHub. [online], [cit. 2023-05-28]. Dostupné z: https://github.com/

[65] Lawson, J.; Cabili, M. N.; Kerry, G.; aj.: The Data Use Ontology to streamline responsible access to human biomedical datasets. *Cell Genomics*, ročník 1, č. 2, 2021: str. 100028, [cit. 2023-02-23]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2666979X21000355

[66] Jackson, R.; Matentzoglu, N.; Overton, J. A.; aj.: OBO Foundry in 2021: operationalizing open data principles to evaluate ontologies. *Database*, ročník 2021, říjen 2021, ISSN 1758-0463, doi:10.1093/database/baab069, baab069. Dostupné z: https://doi.org/10.1093/database/baab069

[67] Xiang, Z.; Mungall, C.; Ruttenberg, A.; aj.: Ontobee: A linked data server and browser for ontology terms. In *Proceedings of the 2nd International Conference on Biomedical Ontologies (ICBO)*, červenec 2011. Dostupné z: http://ceur-ws.org/Vol-833/paper48.pdf

[68] Barton, A. J.; Gramsbergen, E.; Ashton, J.; aj.: The DataCite Ontology. [online], září 2022, [cit. 2023-05-31. Dostupné z: http://purl.org/spar/datacite

[69] Malone, J.; Brown, A.; Lister, A. L.; aj.: The Software Ontology (SWO): a resource for reproducibility in biomedical data analysis, curation and digital preservation. *Journal of Biomedical Semantics*, ročník 5, č. 1, červen 2014, doi:10.1186/2041-1480-5-25. Dostupné z: http://dx.doi.org/10.1186/2041-1480-5-25

[70] Lacan, O.: Keep a changelog. [online], únor 2019, [cit. 2023-06-02]. Dostupné z: https://keepachangelog.com/en/1.1.0/

[71] Hashemi, M.; LaPorte, S.; Williams, M.; aj.: Calendar Versioning. [online], květen 2022, [cit. 2023-06-02]. Dostupné z: https://calver.org/

[72] DSW Team: Template Development Kit. [online], [cit. 2023-06-03]. Dostupné z: https://github.com/ds-wizard/engine-tools/tree/develop/packages/dsw-tdk

[73] Apache Software Foundation: Apache Jena. [online], 2021, [cit. 2023-02-28]. Dostupné z: https://jena.apache.org/

[74] Gayo, J. E. L.; Fernández-Álvarez, D.; Garcıa-González, H.: RDFShape: An RDF playground based on Shapes. In *Proceedings of ISWC*, 2018.

[75] The Apache Software Foundation: APACHE LICENSE, VERSION 2.0. [online], 2023, [cit. 2023-05-28]. Dostupné z: https://www.apache.org/licenses/LICENSE-2.0#apache-license-version-20

[76] Harris, S.; Seaborne, A.: SPARQL 1.1 Query Language. [online], březen 2013, [cit. 2023-06-03]. Dostupné z: https://www.w3.org/TR/sparql11-query/

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface.

DC Dublin Core.

DCAT Data Catalog Vocabulary.

DMP Data Management Plan.

DSW Data Stewardship Wizard.

DUO Data Use Ontology.

eCRF Electronic Case Report form.

FAIR Findable, Accessible, Interoperable, Reusable.

FIP FAIR Implementation Profile.

FIP Wizard FAIR Implementation Profile Wizard.

FOAF Friend of a Friend.

HTML Hyper Text Markup Language.

IRI International Resource Identifier.

JSON JavaScript Object Notation.

LODE Live OWL Documentation Environment.

OOPS! Ontology Pitfall Scanner.

OWL Web Ontology Language.

RDF Resource Description Framework.

RDFS Resource Description Framework Schema.

Schema.org Schema.org ontology.

ShEx Shape Expressions.

SIOC Semantically-Interlinked Online Communities.

SKOS Simple Knowledge Organization System.

SMP Software Management Plan.

SPARQL SPARQL Protocol and RDF Query Language.

SWO Software Ontology.

TDK Template Development Kit.

Turtle Terse RDF Triple Langugage.

UML Unified Modeling Language.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

UUID Universal Unique Identifier.

ViB VODAN in a Box.

VODAN Virus Outbreak Data Network.

W3C World Wide Web Consortium.

WHO World Health Organization.

WIDOCO Wizard for Documenting Ontologies.

XML Extensible Markup Language.

XSD Extensible Markup Language Schema Definition.

ČVUT České vysoké učení technické v Praze.

Příloha B

Obsah přiloženého média

```
README.md ..... stručný popis obsahu média
ontology/
    dsw-ontology.ttl ..... ontologie DSW
    ontology-model.uxf ..... model ontologie DSW
    ontology-model.svg ..... model ontologie DSW
rdf-representation.ttl ..... návrh RDF reprezentace dat DSW
rdf-report-template/
    template.json ..... metadata exportní šablony
    src/
        all.ttl.j2 ..... šablona všech dat
        context.ttl.j2 ..... šablona doplňujících informací
        knowledge-model.ttl.j2 ..... šablona znalostního modelu
        questionnaire-replies.ttl.j2 ..... šablona dotazníku
        prefixes.ttl.j2 ..... šablona prefixů
        macros/
            macro-context.ttl.j2 ..... makra doplňujících informací
            macro-knowledge-model.ttl.j2 ... makra znalostního modelu
            macro-questionnaire-replies.ttl.j2 ..... makra dotazníku
text/
    src/ ..... zdrojová forma práce ve formátu LaTeX
    DP_Martínková_Jana_2023.pdf ..... text práce ve formátu PDF
```