Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky

DIPLOMOVÁ PRÁCE

Strojové zpracování dat z insolvenčního rejstříku

Plzeň, 2016 Jiří Kubeš

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

Plzeň, 2016

Jiří Kubeš

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Janu Valdmanovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce.

Abstrakt

Strojové zpracování dat z insolvenčního rejstříku

Tématem této práce je návrh a implementace řešení, které čerpá data z webové služby poskytované provozovatelem Informačního systému insolvenční rejstřík a tato data transformuje do podoby databáze. Tato databáze může dále sloužit jako zdroj informací pro další informační systémy, například účetní nebo spisové, a to za výrazně jednodušších podmínek, než právě zdrojová webová služba.

V analytické části této práce je rozebrána problematika elektronizace státní správy šířeji než jen z pohledu Informačního systému insolvenční rejstřík. Práce se zabývá přístupem státní správy k tomuto problému, pojmem e-government, a kriticky hodnotí jednotlivé projekty, které v minulosti byly na tomto poli realizovány.

Klíčová slova: insolvenční rejstřík, automatické zpracování, webová služba, ISIR, e-government

Abstract

The primary object of this thesis is analyse, design and implement solution for reading data from insolvency register web service. This data should be transform into relational (SQL) database, that servers as a source of information for other systems, e.g. accountings, and provide them more comfortable access to data than source web service does.

Analytical part of this thesis is concerned with computerization of government in general. It is focused on e-government in Czech Republic, its concept and realizations and critically evaluate them.

Key words: insolvency register, automatic data processing, web services, e-government

Obsah

Seznam obrázků

Seznam diagramů

Seznam tabulek

Seznam použitých zkratek

4G LTE	Mobilní síť 4. generace, Long Term Evolution
A4	formát papíru o velikosti 210 x 297 mm
ADSL	Asymmetric Digital Subscriber Line
AJAX	Asynchronous JavaScript and XML
ANSI	American National Standards Institute
API	Application Programming Interface
ARES	Administrativní registr ekonomických subjektů
ARIS	Automatizovaný rozpočtový informační systém
CAPTCHA	Completely automated public Turing test to tell computers and humans apart
CEDR	Centrální evidence dotací z rozpočtu
CEU	Centrální evidence úpadců
CKAN	Comprehensive Knowledge Archive Network
Czech POINT	Český podací ověřovací informační národní terminál
ČR	Česká republika
ČÚZK	Český úřad zeměměřičský a katastrální
DPH	Daň z přidané hodnoty
DVD	Digital Versatile Disc
ER diagram	Entity Relationship diagram
ERA diagram	Entity Relationship Attributes diagram
EU	Evropská unie
EZP	Evidence zemědělského podnikatele
FO	Fyzická osoba
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IČ	Identifikační číslo
ID	Identifikátor
IDE	Integrated Development Environment
IP	Internet Protocol
IR	Insolvenční rejstřík
ISDS	Informační systém datových schránek
ISIR	Informační systém insolvenční rejstřík
JAXB	Java Architecture for XML Binding
JAX-WS	Java API for XML Web Services
JDBC	Java Database Connectivity
KN	Katastr nemovitostí
OECD	Organisation for Economic Co-operation and Development
OR	Obchodní rejstřík
ORM	Object-relational mapping
OS	Operační systém
OSN	Organizace spojených národů
OSS	Seznam občanských sdružení a spolků

OVM ..... Orgán veřejné moci
PDF ..... Portable Document Format
PFO ..... Podnikající fyzická osoba
PHP ..... PHP: Hypertext Preprocessor
PO ..... Právnická osoba
PSH ..... Seznam politických stran a hnutí
RARIS ..... Účelový registr organizací systému ARIS
RCNS ..... Registr církví a náboženských společností
RES ..... Registr ekonomických subjektů
RŠ ..... Rejstřík škol a školských zařízení
RÚIAN ..... Registr územní identifikace, adres a nemovitostí
RZZ ..... Registr zdravotnických zařízení
RŽP ..... Registr živnostenského podnikání
SD ..... Spotřební daň
SJM ..... Společné jmění manželů
SMTP ..... Simple Mail Transfer Protocol
SOAP ..... Simple Object Access Protocol
SQL ..... Structured Query Language
SŘBD ..... Systém řízení báze dat
TV ..... Televize
UIR-ADR ..... Územně identifikační registr adres
UML ..... Unified Modeling Language
URI ..... Uniform Resource Identifier
USA ..... United States of America
VZP ..... Všeobecná zdravotní pojišťovna
WSDL ..... Web Services Description Language
XML ..... Extensible Markup Language
XSD ..... XML Schema Definition
ZČU ..... Západočeská univerzita

1 Úvod

V dnešní informační společnosti jsou informace artiklem, který je pro mnoho subjektů tím nejcennějším, co lze získat. Člověk, který má dostatek informací, se může lépe rozhodovat o svých budoucích krocích. V korporátním světě byla již tato skutečnost pochopena a téměř každá korporace investuje nemalé prostředky do rozvoje vnitřních informačních systémů, pomocí kterých koriguje tok informací uvnitř korporace, ale také navenek, a to jak k zákazníkům, tak zejména k obchodním partnerům.

Člověk moderní doby si postupně zvykl na to, že nejlepším zdrojem informací je dnes kyberprostor, tedy zejména celosvětová síť Internet. Pokud potřebujeme nějakou informaci získat, nejčastěji otevřeme počítač a začneme hledat právě na Internetu. Fenoménem poslední doby není jen pasivní přijímání informace, ale též aktivní interakce se vzdáleným subjektem. Zdárným příkladem jsou internetové obchody.

Na tuto skutečnost, byť s určitým zpožděním, reaguje i státní správa. I stát poskytuje svým občanům informace v kyberprostoru více či méně úspěšným způsobem. V dnešní době již snad neprobíhá vyhledávání informací v obchodním rejstříku jinak, než přes jeho elektronickou verzi, díky zavedení a uznávání elektronického podpisu lze komunikovat se státní správou elektronicky z domova od počítače a občan tak nemusí absolvovat mnohdy velice nepříjemnou návštěvu úřadu. I stát se snaží zjednodušit si svoje úlohy přesunem do kyberprostoru, a tak například vznikl koncept datových schránek, který umožňuje (nejen) státním institucím doručovat zásilky s účinky fyzického doručení prostřednictvím poštovního doručovatele.

Specifickou součást v tomto procesu hraje i automatické zpracovávání. Informace lze totiž konzumovat buďto v lidsky čitelné podobě, nebo ve strojově čitelné podobě. S rozmachem poskytování informací v kyberprostoru se samozřejmě rozvíjí možnosti automatického zpracování. Pomocí standardizovaných technologií lze data čerpat z různých zdrojů a následně je jakkoliv zpracovat, a to s efektivitou, kterou by člověk samotný nebyl nikdy schopen.

Tato práce se zabývá právě možnostmi komunikace s veřejnou správnou, tedy tzv. e-governmentem, a dále se zabývá analýzou, návrhem a implementací řešení pro automatické zpracování dat z insolvenčního rejstříku.

2 Problematika e-government

Tato kapitola se zabývá analýzou vnějšího prostředí (business analýzou) okolo elektronizace státní správy. Rozebírá pojem e-government a věnuje se tomu, jaké jsou cíle jeho implementace v sociálněprávním prostředí České republiky a jak jsou tyto cíle naplňovány.

S rozmachem informačních technologií a automatického zpracování se tyto pomalu dostávají i do života státních úřadů. Pryč už jsou doby, kdy si soudci, advokáti a úředníci každý rok kupovali kompletní vydání sbírky zákonů a při každé novelizaci text přelepovali a vpisovali nový. Stejně tak již lze pomocí datových schránek učinit prakticky jakékoliv podání kterémukoliv úřadu elektronicky se stejným účinkem a relevancí, jako by takové podání bylo učiněno pomocí poskytovatele poštovních služeb nebo přímo osobně na podatelně příslušného úřadu.

2.1 Pojem e-governmentu

Pojem e-government je fenoménem poslední doby. R. Heeks chápe e-government v širokém pojetí jako použití informačních a komunikačních technologií ve veřejném sektoru [1]. Podobně Evropská komise v akčním plánu eGovernmentu na roky 2011-2015 [2] vymezuje e-government jako použití nástrojů a systémů, které jsou zde díky informačním a komunikačním technologiím, pro poskytování lepších veřejných služeb občanům a podnikům. Podobně zní i definice, která říká, že e-government je použití informačních technologií vládními institucemi za účelem transformace vztahů s občany, podniky, ale i dalšími součástmi veřejné správy, kdy tato transformace může sloužit ke zkvalitnění doručování orgány veřejné moci soukromým subjektům, k posílení interakce s podnikatelskou sférou nebo k lepšímu přístupu k informacím pro občany a zvýšení efektivity veřejné správy [1]. V tuzemském pojetí se o definici e-governmentu pokusilo bývalé Ministerstvo informatiky, které jej chápalo jako proces transformace vnitřních a vnějších vztahů veřejné správy pomocí informačních a komunikačních technologií s cílem optimalizovat interní procesy. Jeho posláním je rychlejší, spolehlivější a levnější poskytování služeb veřejné správy ve vztahu k jejím uživatelům [1]. Problémem této definice je redukce pojmu pouze na oblast veřejné správy, což není přesné, neboť elektronizace probíhá např. i v oblasti justice, která pod veřejnou správu nespadá. Je tedy vidět, že nad jednoznačným obsahem pojmu e-government nepanuje celosvětově jednoznačná shoda, ale v zásadě se dá říci, že se jedná o elektronizaci komunikace občanů s úřady veřejné moci a komunikace těchto úřadů navzájem.

V současném světě je nutné e-government chápat jakou součást modelu „dobré správy“ (Good Governance), tj. procesu vládnutí a jeho kontroly a participace občanů na tomto procesu. Dlouhodobě je v rozvinutých civilizacích vidět trend v přesunu od vrchnostenské správy věcí veřejných k vnímání této správy jako služby občanům coby zákazníkům. Důležitost implementace postupů, které se pod pojem skrývají, je dále potrhávaná zájmem organizací jako je OSN, OECD nebo EU o míru jejich využití v jednotlivých členských státech. Lze říci, že přenesení komunikace mezi veřejnou mocí a ostatními subjekty je v zájmu všech zúčastněných. Pro veřejnou správu znamená elektronizace komunikace snížení nákladů a automatizaci postupů, které před tím vykonávali úředníci, a pro občany a podniky to znamená pohodlnější přístup k informacím a službám veřejné správy, což ve finále vede k jejich nižším výdajům na tyto úkony. Pomocí e-governmentu došlo v posledních letech, nejen v České republice, k výrazným úsporám. Pomocí informačních technologií se podařilo odstranit duplicity, případně i multiplicity, činností, které jednotlivé orgány veřejné moci prováděly. Dále došlo

ke zkvalitnění toků informací mezi institucemi. Skvělým příkladem z České republiky jsou základní registry (obchodní rejstřík atd.). Ne vždy však musí převedení činností do sféry informačních technologií nutně znamenat úspory a to jak finanční tak i časové. Ze strany veřejné moci je třeba vždy sledovat určitý cíl a zkoumat, zda právě využitím informačních technologií bude tohoto cíle nejefektivněji dosaženo. Na druhou stranu rozhodování by nemělo stát na ryze ekonomických kritériích, neboť pomocí e-governmentu lze dosahovat lepší občanské informovanosti a tudíž i vyšší kontroly a angažovanosti ze strany občanské společnosti, což jsou důležité činitele udržitelnosti každého demokratického zřízení.

Dalším úhlem pohledu na problematiku e-governmentu je jeho dostupnost pro běžného občana. V souvislosti s převedením činnosti veřejné správy do kyberprostoru může dojít k sociálnímu vyloučení občana, který k takové technologii nemá z jakéhokoliv důvodu přístup. Toto riziko se ale s plynutím času stále snižuje. Jelikož zavedení e-governmentu velice úzce souvisí s celosvětovou sítí Internet, která je podkladovým komunikačním médiem pro veškeré služby poskytované formou e-governmentu, dá se říci, že riziko sociálního vyloučení občana lze převést na obtížnost přístupu občana k síti Internet a jeho schopnosti pomocí tohoto média komunikovat. První problém je eliminován postupem času tím, že samotné informační technologie (myšleno ve smyslu zařízení, pomocí kterého může koncový uživatel přistupovat k síti Internet) se stávají dostupné pro široký počet obyvatelstva a stejně tak samotné připojení k síti Internet se stává dostupnějším. V této souvislosti je třeba mít na paměti nejen cenovou stránku věci, ale také technologickou (kvalitativní), kdy dochází k rozšiřování vysokorychlostního internetu bez omezování množství stažených dat a to nejen v oblasti pevného připojení k internetu (ADSL, kabelová televize atd.), ale v poslední době i pomocí mobilního připojení k internetu ze strany mobilních operátorů (4G LTE). Důležitou roli v tomto procesu hrají také menší, většinou lokální, poskytovatelé internetu, kteří jsou schopni za nižší ceny nabízet kvalitnější připojení k síti Internet, než velké celostátní společnosti, a to převážně na technologii WiFi. Nad rámec uvedeného téměř každá obec poskytuje možnost přístupu k Internetu zdarma a to buď přímo ze zařízení umístěných v prostorách obecního úřadu, popřípadě ve větších obcích jako doplňkovou službu knihoven apod.

Pod problematiku e-goverment by se dalo v jeho širokém pojetí také zařadit udělení právní relevance jednání mimostátních subjektů učiněného v elektronické podobě a jeho uznání a případná vymahatelnost orgány státní moci. Důležitým prvkem bylo v tomto ohledu zavedení pojmu elektronický podpis do českého právního řádu. Zákon č. 89/2012, občanský zákoník, stanovuje v § 562 rovnost písemného právního jednání a právního jednání učiněného elektronickými nebo jinými technickými prostředky a v § 561 staví na roveň vlastnoruční a elektronický podpis. Kombinací těchto dvou ustanovení se stává platnou a vymahatelnou jakákoliv smlouva uzavřená elektronickými prostředky mezi dvěma soukromými subjekty. Navíc lze požadavek úředně (notářsky) ověřeného podpisu nahradit státem uznávaným (kvalifikovaným) elektronickým podpisem.

2.2 Open data

S pojmem e-government také souvisí problematika otevřených dat (open data). „Otevřená data jsou informace a čísla bezplatně a volně dostupná na internetu ve strukturované a strojově čitelné podobě a jsou zpřístupněna způsobem, který jejich využití neklade zbytečné technické či jiné překážky.“ [3]. Aby data mohla být považována za otevřená, musí být tedy splněno několik podmínek. Mezi technické podmínky patří zejména to, že data by měla být strukturována v nějakém široce

používaném standardu tak, aby bylo možné je bez velkých obtíží hromadně strojově (počítačově) zpracovávat. Mezi ostatní podmínky patří zejména jejich volná přístupnost a nakládání s nimi. Jejich poskytovatel tedy musí data poskytnout zdarma a musí poskytnout právní svolení k jejich neomezenému využívání. Poskytovatelem dat nemusí být pouze státní subjekty, ale také nevládní organizace, univerzity nebo komerční subjekty. Otevřená data nesouvisí pouze s informovaností široké veřejnosti, ale mají pozitivní dopad na vývoj ekonomiky. Toho si jsou vědomi i zodpovědné úřady, a proto je strategie otevřených dat implementována mj. ve strategických dokumentech rozvoje e-governmentu v České republice [3].

Jak již bylo řečeno výše, data by měla být strukturována v široce používaném formátu. Jedním z nejpoužívanějších systémů je Comprehensive Knowledge Archive Network (CKAN) [4]. Jedná se o open source řešení používané pro ukládání dat a jejich publikaci ve standardizované struktuře. Back-end systému se vtěluje do webového serveru (nejčastěji Apache) a je napsán v Pythonu. Samotná data systém ukládá do databáze PostgreSQL. Uživatelé mohou data procházet pomocí webového rozhraní, které k systému náleží, nebo strojově zpracovávat pomocí Web API. Systém CKAN je široce využíván soukromým i veřejným sektorem, jako platformu pro poskytování dat jej využívají vlády Spojeného království Velké Británie a Severního Irska, Nizozemska, Austrálie nebo USA [5].

Ze státních institucí v České republice poskytuje nejvíce otevřených dat Ministerstvo financí na portálu http://data.mfcr.cz. Dále Ministerstvo vnitra ČR provozuje Portál veřejné správy, který na adrese https://portal.gov.cz/portal/obcan/rejstriky/data má snahu o zřízení centrálního místa pro zveřejňování otevřených dat i od jiných institucí. Tento portál ale nevyžaduje formátování dat do předem určených standardů, velké množství dat je zde poskytnuto formou Excel WorkSheet (.xls, .xlsx). Otevřená data poskytují i jiné instituce, ale je třeba hledat na internetových stránkách jednotlivých organizací nebo využít portály provozované soukromými subjekty. Mezi takové patří např. OpenData.cz, kde lze datasety nalézt na adrese http://linked.opendata.cz/organization/opendata-cz, nebo portál OtevřenáData.cz, který zveřejňuje rozcestník vedoucí na stránky jednotlivých tuzemských i mezinárodních organizací, na adrese http://www.otevrenadata.cz/otevrena-data/zdroje-dat.

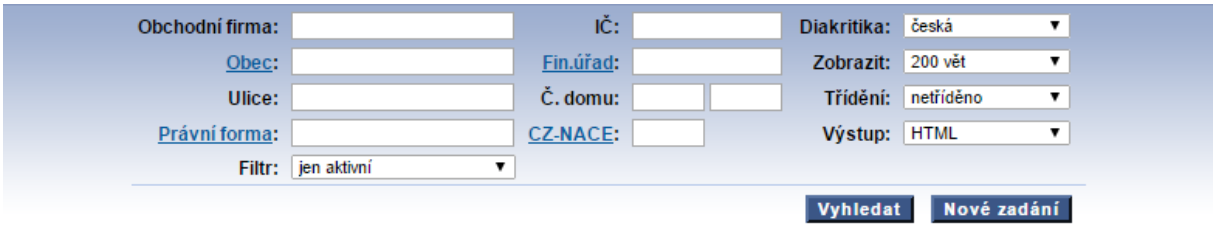
2.3 Administrativní registr ekonomických subjektů

Administrativní registr ekonomických subjektů je informační systém, který umožňuje vyhledávání nad ekonomickými subjekty registrovanými v České republice. Zprostředkovává zobrazení údajů vedených v jednotlivých registrech státní správy, ze kterých čerpá data (tzv. zdrojové registry) [6]. Provozovatelem registru je Ministerstvo financí ČR. Nejedná se tedy o primární zdroj dat, ale o sběrné místo, do kterého jsou importována data z registrů vedených ostatními státními institucemi. Podle [6] jsou těmito registry

a) Obchodní rejstřík (OR), vedený rejstříkovými soudy,
b) Živnostenský rejstřík (RŽP), vedený Ministerstvem průmyslu a obchodu ČR,
c) Registr ekonomických subjektů (RES), vedený Českým statistickým úřadem,
d) Registr církví a náboženských společností (RCNS), vedený Ministerstvem kultury ČR,
e) Registr zdravotnických zařízení (RZZ), vedený Ústavem zdravotnických informací a statistiky ČR,
f) Seznam občanských sdružení a spolků (OSS), vedený Ministerstvem vnitra ČR,

g) Evidence zemědělského podnikatele (EZP), která je vedena Ministerstvem zemědělství ČR,
h) Seznam politických stran a hnutí (PSH), vedený Ministerstvem vnitra ČR,
i) Rejstřík škol a školských zařízení (RŠ), vedený Ministerstvem školství a tělovýchovy ČR,
j) Registr plátců daně z přidané hodnoty (DPH, SkDPH, údaje o nespolehlivém plátci DPH), vedený Finanční správou ČR,
k) Registr plátců spotřební daně (SD), vedený Celní správou ČR,
l) Účelový registr organizací systému ARIS (RARIS), vedený Ministerstvem financí ČR,
m) Centrální evidence dotací z rozpočtu (CEDR), která je vedena Ministerstvem financí ČR,
n) Centrální evidence úpadců (CEU), která je vedena Ministerstvem spravedlnosti ČR,
o) Insolvenční rejstřík (IR), který je veden Ministerstvem spravedlnosti ČR,
p) Územně identifikační registr adres (UIR-ADR), vedený Ministerstvem práce a sociálních věcí ČR.

Z výše uvedeného vyplývá, že informace nalezené v tomto registru jsou komplexní a poměrně provázané. Výstupy z informačního systému registru jsou ve formátu XML, a to i v případě, že dochází k ručnímu vyhledávání ve webovém formuláři. V takovém případě dochází ke konverzi na HTML výstup. Z toho vyplývá, že celý systém je konstruován od počátku tak, aby jeho výstupy byly dále strojově zpracovatelné. Nejjednodušší způsob vyhledávání v registru je přes webový formulář.



Obrázek 2.1: Webový formulář pro vyhledávání v ARES

Jak zobrazuje Obrázek 2.1, vyhledávat lze podle různých kritérií. Nejčastěji uživatel vyhledává podle IČ, neboť se jedná o jednoznačný identifikátor jakéhokoliv ekonomického subjektu. Lze ale vyhledávat i podle firmy (laicky řečeno názvu), sídla atd. Výběrem *Výstup* lze vybrat, zda výsledek bude zobrazen v HTML nebo XML formě. Výsledkem vyhledávání je vždy seznam subjektů a odkazy na rejstříky, v kterých byl nalezen záznam o lustrované osobě. V případě obchodní korporace vypadá výsledek takto

ARES - přehled vybraných ekonomických subjektů — počet: 1

IČ	Obchodní firma Místo podnikání	Odkazy
26011590	PROXIMA 3000, s.r.o. Nasavrky, Ochoz 6	OR RES RŽP IR DPH

Strana : 1 / 1

Obrázek 2.2: Výsledek lustrace obchodní korporace v registru ARES

Obrázek 2.2 zobrazuje základní údaje o obchodní korporaci a dále hypertextové odkazy na záznamy jednotlivých registrů. V případě této konkrétní obchodní korporace byly nalezeny záznamy v Obchodním rejstříku, Registru ekonomických subjektů, Živnostenském rejstříku, Insolvenčním rejstříku a Registru plátců daně z přidané hodnoty. Pro zajímavost je uveden výpis lustrace pro Západočeskou univerzitu v Plzni.

ARES - přehled vybraných ekonomických subjektů počet: 1

IČ	Obchodní firma Místo podnikání	Odkazy	
49777513	**Západočeská univerzita v Plzni** Plzeň 3, Jižní Předměstí, Univerzitní 2732/8	RES RŽP RARIS CEDR DPH	

Strana : 1 / 1

Obrázek 2.3: Výsledek lustrace ZČU v registru ARES

Vzhledem k jinému zaměření právnické osoby, jakou je ZČU, je i struktura registrů, ve kterých byl nalezen záznam o subjektu, odlišná od obchodní korporace. Zajímavou funkcionalitou je porovnání údajů mezi registry (tlačítko), kdy systém zobrazí tabulku s porovnáním společných údajů v registrech.

IČ=49777513	Obchodní firma	PF	Jméno	Příjmení	Obec	Městská část	Část obce	Ulice	Číslo domu	Číslo orientační	Číslo do adresy
ARES	Západočeská univerzita v Plzni	601			Plzeň	Plzeň 3	Jižní Předměstí	Univerzitní	2732	8	
OR											
RES	Západočeská univerzita v Plzni	601			Plzeň	Plzeň 3	Jižní Předměstí	Univerzitní	2732	8	
RŽP	Západočeská univerzita v Plzni	331			Plzeň	Plzeň 3	Jižní Předměstí	Univerzitní	2732	8	
RZZ											
RCNS											
ISPOZ											
OSS											
EZP											
PSH											
RŠ											

Obrázek 2.4: Porovnání společných údajů napříč registry

V další části této podkapitoly se budeme zabývat možností strojového přístupu a zpracování dat poskytovaných registrem ARES. Komunikace probíhá pomocí XML výstupů. Pro získání dat lze provést dotaz pomocí metody HTTP GET, HTTP POST popřípadě využít jednu z poskytovaných webových služeb a komunikovat s registrem pomocí SOAP. Bohužel ani pro jednu metodu komunikace neexistuje unifikovaný způsob, jak získat veškerá data, která registr poskytuje. Nejvíce informací lze vytěžit pomocí přístupu přes SOAP a HTTP GET metodu. Filosofie poskytovaných služeb je taková, že pro každý konkrétní dotaz musí uživatel zvolit jinou HTTP GET metodu nebo webovou službu (SOAP). Uživatel může podat žádost typu Standard, Basic a pak speciálně na vybrané registry. Žádost typu Standard vrací základní informace o subjektu z jádra ARES. Jsou to údaje, které pocházejí z jednoho ze zdrojů, který byl pro subjekt určen jako prioritní. Základními údaji jsou IČ, firma, právní forma, datum vzniku a adresa [6]. Rozšířené vyhledávání v podobě žádosti Basic vrací informace o subjektu vybrané z více zdrojových registrů. Základní údaje (IČ, firma, právní forma, adresa) pocházejí z jádra ARES, tj. z jednoho ze zdrojů, který byl pro subjekt určen jako prioritní. Z prioritního zdroje pochází i datum vzniku a datum zániku subjektu. Některé elementy obsahují informaci o zdroji, ze kterého údaj pochází [6].

Co se týče programového zpracování, tak pro SOAP komunikaci je nabízeno 32 WSDL definic. Nutno dodat, že 12 ze zmíněných 32 definic je duplicitních. Každá poskytovaná služba nabízí metodu GetXmlFile pomocí které se provádí dotazování. Podle dokumentace jsou služby poskytovány ve dvou verzích pro zachování kompatibility s klienty využívajícími tyto starší verze služeb. Dále je poskytováno 16 přístupových adres pro získávání dat metodou HTTP GET. Každá přístupová adresa má jiné vstupní parametry, které závisí na typu dotazovaného registru. Společným parametrem je ale vždy IČ, podle kterého lze subjekt nalézt v kterémkoliv registru. Poslední možností je metoda HTTP POST, pro kterou je možno použít 8 přístupových adres. Koncepce je podobná jako u metody HTTP GET.

Pro úplnost budou uvedeny ještě rozdíly v použití jednotlivých metod. V případě použití webové služby (SOAP) je komunikace mezi klientem a informačním systémem registru definována pomocí SOAP standardu. Pokročilé programovací jazyky (Java, .NET, atd.) umožňují z WSDL definic vygenerovat třídy pro přístup ke službě a potřebné datové třídy. Ve zdrojovém kódu klienta je pak s webovou službou pracováno pomocí takto vygenerovaných tříd a programátor se dále nestará o samotnou komunikaci a převod XML dokumentů, pomocí kterých komunikace v SOAP probíhá, do podoby datových tříd a vice versa. U použití klasického protokolu HTTP a jedné z metod přenosu vstupních informací, tedy GET nebo POST, musí programátor klientské aplikace nejprve provést samotnou HTTP komunikaci a získat tak výstup ve formátu XML a tento dokument pak dále zpracovat (přečíst z něj potřebné informace, převést na datovou třídu atd.). Pokud tedy programovací jazyk umožňuje nativní využití webových služeb a komunikaci SOAP protokolem, je to nejjednodušší způsob, jak s daty pracovat.

Celá myšlenka centrálního místa pro výstupy z jednotlivých registrů je určitě dobrým počinem. Navíc většina registrů, které ARES sdružuje, neposkytuje rozhraní pro strojové zpracování jimi poskytovaných dat. Z tohoto hlediska má ARES své nezastupitelné místo v českém e-governmentu. Největší výtkou, a zároveň návrhem na vylepšení tohoto registru, je způsob implementace webových služeb. Webové služby jsou natolik komplexní entitou, že by mohla být poskytována pouze jediná (nikoliv 32) a následně dotazování jednotlivých registrů provádět metodami implementovaným do této jediné služby.

Bližší informace o registru ARES lze nalézt v [6], samotné WSDL definice a XSD schéma pro XML komunikaci pak na adrese [7].

2.4 Insolvenční rejstřík

Insolvence, nebo též úpadek, bankrot, je stav, při němž má dlužník závazky vůči řadě věřitelů a není v jeho reálných možnostech, aby je všechny splnil. Pokud se osoba, nehledě na to, zda se jedná o osobu právnickou, fyzickou nebo fyzickou osobu podnikatele, ocitne v takovémto stavu, existuje zde institut insolvence. Stěžejní úpravou pro toto řízení je zákon č. 182/2006 Sb., o úpadku a jeho řešení (insolvenční zákon), který v § 2 definuje insolvenční rejstřík jako informační systém, který obsahuje údaje podle tohoto zákona. Dle § 419 by měl insolvenční rejstřík obsahovat seznam insolvenčních správců, seznam dlužníků a insolvenční spisy. Dále stanovuje, že insolvenční rejstřík je veřejně přístupný, s výjimkou údajů, o kterých tak stanoví tento zákon, a že každý má právo do něj nahlížet a pořizovat si z něj kopie a výpisy. Zcela jasně a uceleně tedy definuje, co insolvenční rejstřík je, k čemu slouží a co vněm lze nalézt [8].

Provozovatelem insolvenčního rejstříku je Ministerstvo spravedlnosti ČR, ale jeho uživatelem jsou převážně krajské soudy, které jsou věcně příslušné pro vedení insolvenčního řízení, a dále též exekutoři, advokáti a obecně subjekty účastnící se obchodního styku. Pro ty jsou údaje z insolvenčního rejstříku důležité, neboť zákon s uveřejněním údajů spojuje následky, které svými důsledky přesahují osobu dlužníka, popřípadě věřitele, který podal návrh na zahájení insolvenčního řízení s dlužníkem. Např. účinky úkonů v insolvenčním řízení většinou nastávají okamžikem jejich zveřejnění v insolvenčním rejstříku. Takto nastává i účinek zahájení insolvenčního řízení, což je důležitá skutečnost, neboť proti osobě, se kterou je vedeno insolvenční řízení, nemůže být vedena exekuce nebo soudní výkon rozhodnutí na majetek ve vlastnictví dlužníka, nebo proti dlužníkovi nelze vést soudní spor o pohledávky, týkající se majetkové podstaty, pokud takové pohledávky lze

uplatnit přihláškou do insolvenčního řízení atd. [8]. Dalším naprosto stěžejním údajem, zveřejňovaným pomocí insolvenčního rejstříku, je informace o tom, zda byl u dlužníka zjištěn úpadek a pokud ano, tak jakou formou se bude řešit. To je velice důležitá informace pro dlužníkovy věřitele, neboť od tohoto okamžiku jim počíná běžet lhůta, ve které mají možnost svou pohledávku do insolvenčního řízení přihlásit. Pokud věřitel tak neučiní, stává se jeho pohledávka velice složitě dobytnou a v určitých případech osobního oddlužení dokonce nevymahatelnou. Z toho vyplývá, že informace obsažené v insolvenčním rejstříku jsou pro věřitele velice důležité a věřitel, který jedná s péčí řádného hospodáře, insolvenční rejstřík pravidelně kontroluje a lustruje v něm své dlužníky. Dále jsou informace obsažené v insolvenčním rejstříku cenné z hlediska obchodní strategie, neboť uzavírání obchodů se subjektem, se kterým je vedeno insolvenční řízení, je velice rizikové. Z obchodního hlediska je také relevantní zjišťovat informace o tom, zda protistrana opakovaně nezneužívá výhody, které jí zahájené insolvenční řízení nabízí (zejména pak zastavení a nemožnost vedení exekucí na majetek) a ztěžování tím postavení svých věřitelů.

Pro přístup k informacím, obsaženým v insolvenčním rejstříku provozuje ministerstvo spravedlnosti webový portál a dále též webové služby, pomocí kterých lze data z insolvenčního rejstříku strojově zpracovávat. Bližším popisem těchto služeb se zabývá kapitola 3.

2.5 Katastr nemovitostí

Tato podkapitola se zabývá službami, které v oblasti e-government poskytuje Český úřad zeměměřičský a katastrální. Základní službou poskytovanou široké veřejnosti je Nahlížení do KN. Tato služba umožňuje uživateli získat informace týkající se vlastnictví parcel, staveb, jednotek a práv stavby evidovaných v katastru nemovitostí. Kromě informace o vlastnictví výše uvedených nemovitých věcí lze vyhledat ještě základní údaje, jako jsou výměra nebo číselné označení a dále práva váznoucí na těchto věcech (například věcná břemena, zástavní právo apod.). Dále lze v aplikaci Nahlížení do KN vyhledat informace ohledně probíhajících řízení týkajících se změn zápisu v katastru nemovitostí. Jedná se o základní aplikaci, pomocí které lze vyhledávat pouze podle identifikačních čísel parcel, popřípadě staveb a spisových značek příslušných řízení. Ostatní formy vyhledávání např. podle IČ právnických osob nebo podle rodného čísla fyzických osob nejsou možné. Aplikace Nahlížení do KN je určena pro interaktivní práci uživatelů a nemá ekvivalent v podobě jakékoliv strojově čitelné verze. Naopak, provozovatel služby aktivně brání jakékoliv činnosti, která by směřovala k vytěžování dat z této služby pomocí automatizovaných prostředků. K tomuto bránění dochází formou kontroly IP adres a chování uživatele a dále blokací automatických robotů pomocí CAPTCHA kódů [9]. Podle [9] je aplikace určena pro bezplatné nahlížení do katastru nemovitostí do vybraných údajů souboru popisných informací a do vybraných údajů souboru geodetických informací v souladu s § 52 odst. 1 zákona č. 256/2013 Sb., o katastru nemovitostí (katastrální zákon), a do údajů v evidenci doručených návrhů v souladu s § 9 katastrálního zákona. Poskytované údaje lze užívat pouze k účelům uvedeným v § 1 odst. 2 katastrálního zákona.

Službou navazující na Nahlížení do KN je Dálkový přístup k údajům katastru nemovitostí České republiky. Jedná se o placenou službu, kdy si uživatel musí nejprve založit zákaznický účet. Jakékoliv vyhledávání je pak účtováno podle platného ceníku. Ten lze najít na adrese [10]. Je zajímavé, že ačkoliv se jedná o elektronické výpisy na obrazovku nebo ve formátu PDF je většina informací účtována v měrné jednotce stránka formátu A4, přičemž cena za jednu měrnou jednotku je povětšinou 50,- Kč (k datu sepsání této práce). Pomocí služby Dálkový přístup k údajům katastru nemovitostí

České republiky lze získat veškeré informace, které katastr nemovitostí poskytuje veřejnosti. Jedná se tedy o alternativu ke klasickému způsobu získávání výpisů a informací osobní návštěvou úřadu. Její funkcionalita je tedy o mnoho širší než u aplikace Nahlížení do KN. Informace o této službě jsou čerpány z veřejně přístupného popisu uvedeného v [11], neboť autor práce nemá přístup do placené části služby.

ČÚZK nabízí pro svoji placenou službu i verzi přístupnou pro strojové zpracování. Jedná se o webovou službu založenou na protokolu SOAP. Z dokumentace služby uvedené v [11] lze vyčíst, že služba je komplexně zpracována, měla by být plnohodnotným ekvivalentem služby Dálkový přístup k údajům katastru nemovitostí České republiky. I tato služba je zpoplatněna, platí pro ni stejný ceník jako pro její interaktivní verzi. Zajímavostí je opět účtování s měrnou jednotkou jedna stránka formátu A4, což je v případě XML odpovědí velice zvláštní. Provozní podmínky tuto zvláštnost vysvětlují tak, že za XML výstup bude účtováno tolik měrných jednotek, kolik by bylo třeba stránek formátu A4 pro vytištění vrácených informací v případě, že by uživatel žádal o výpis na přepážce úřadu. Stejně jako v případě přístupu přes grafické rozhraní, ani do webové služby nemá autor této práce přístup. Další zajímavostí ohledně placených služeb je také skutečnost, že z dokumentace k nim nevyplývá, že by provozovatel jakkoliv omezoval vytěžování dat z databáze. Zda tomu tak opravdu je, nebo zda provozovatel považuje omezení stanovené zpoplatněním služeb za dostatečné, se nepodařilo zjistit.

Další službou poskytovanou ČÚZK zdarma je přístup k mapovému podkladu. Tato služba je přístupná pomocí standardu Web Map Services. Pro bližší informace ohledně této služby odkážeme laskavého čtenáře na informace zveřejněné na adrese [12].

Poslední službou poskytovanou ČÚZK, kterou se budeme v této práci zabývat, je přístup k datům registru územní identifikace, adres a nemovitostí (RÚIAN). Jedná se o kompletní databázi všech územních celků od úrovně celého státu až po městské části u členěných měst. Dále tato databáze obsahuje mj. názvy všech ulic v ČR. Portál pro interaktivní vyhledávání lze nalézt na http://vdp.cuzk.cz. Kromě interaktivní části nabízí také data, která lze zpracovávat strojově. Data jsou poskytována ve formě komprimovaného XML (xml.gz). Na portálu [13] lze vygenerovat dataset s potřebnými údaji a ten poté strojově zpracovat. Pokud by uživatel potřeboval z nějakého důvodu provádět předem neurčité dotazy nad databází strojově, bylo by nutné nejprve celou databázi ve formě komprimovaného XML stáhnout a zpracovat dle potřeb uživatele (nejspíše import do SQL databáze) a poté takovou databázi stejným způsobem aktualizovat. Pro detailnější informace ohledně registru RÚIAN odkážeme laskavého čtenáře na informace zveřejněné v [13].

2.6 Další vybrané služby poskytované orgány veřejné moci

V této souhrnné podkapitole budou ukázány další služby, které orgány veřejné moci poskytují veřejnosti. Centrálním rozcestníkem pro celou škálu informací je Portál veřejné správy (http://portal.gov.cz) provozovaný Ministerstvem vnitra ČR. Jedná se o portál shromažďující velké množství různorodých dat a je koncipován tak, aby pokud možno odpovídal na většinu problémů, které jednotlivci mohou v interakci se státní správou vzniknout. Těmito informacemi se ale zabývat nebudeme, jedná se v podstatě o klasický informační web se specifickou tématikou. Do kategorie e-government ale patří služby poskytované portálem Czech POINT, který je na Portál veřejné správy úzce napojen. Czech POINT původně vznikl jako jednotné kontaktní místo veřejné správy, na kterém mohl občan získat výpisy ze základních registrů, aniž by musel navštívit několik institucí. V současné

době je možné pomocí služeb Czech POINT získat elektronické verze výpisů ze základních registrů. Tyto výpisy jsou opatřeny kvalifikovaným elektronickým podpisem příslušné instituce, jsou tedy postaveny na roveň písemné formě vyjádřené na papírovém podkladu. Pro získání těchto výpisů musí být žádost odeslána prostřednictvím datové schránky (o problematice datových schránek bude pojednávat následující podkapitola) a do datové schránky jsou následně také doručovány. Je tím zajištěno ověření oprávnění osoby k získání požadovaného výpisu a zároveň doručením do datové schránky je zajištěno také doručení výpisu oprávněné osobě. V současnosti lze touto cestou získat výpis z Registru obyvatel, Registru osob, Insolvenčního rejstříku, Veřejného rejstříku (obchodní rejstřík, spolkový rejstřík, atd.), Rejstříku trestů právnických osob, Živnostenského rejstříku a Seznamu kvalifikovaných dodavatelů [14]. Vzhledem k logice věci nemá smysl uvažovat o automatizovaném rozhraní této služby. Teoreticky si lze představit vyplnění formuláře a jeho odeslání automatizovaným skriptem a následné stažení výpisu doručeného do datové schránky.

Obdobou Portálu veřejné správy v oblasti soudnictví je portál Justice.cz. Z tohoto portálu lze lustrovat obchodní rejstřík (pouze interaktivní verze pro komunikaci s lidskou bytostí, strojově zpracovatelná verze je dostupná pouze přes ARES) a insolvenční rejstřík. Dále jsou z portálu Justice.cz přístupné informační služby infoDeska, infoSoud, infoJednání a infoData. InfoDeska je elektronickou verzí úřední desky na které publikují všechny soudy. InfoSoud je aplikace, pomocí které uživatel zjistí z jemu známé spisové značky informaci o vedeném soudním řízení, převážně stav, ve kterém se řízení nachází. Služba infoJednání je obdobnou službou jako infoSoud, pomocí které lze získat detaily o nařízeném soudním jednání a vůbec informace o soudních jednáních proběhlých v řízení. Aplikace infoData poskytuje přístup k statistickým datům ohledně soudnictví. Data jsou v této aplikaci většinou publikována ve formě Excel WorkSheet (.xls, .xlsx) nebo Adobe Acrobat PDF (.pdf). K těmto službám v současné době neexistují jejich ekvivalenty pro strojové zpracování. Bližší informace lze nalézt v [15].

2.7 Komunikace se státní správou

Předchozí podkapitoly se zabývaly poskytováním dat od státních orgánů veřejnosti. Komplexní e-government by ale měl obsahovat i možnost pro opačný tok informací, tedy od občana ke státu. Některé instituce poskytují specifické rozhraní pro podání žádostí (např. ePodatelna na portálu Justice.cz), ale většina komunikace mezi soukromými subjekty a veřejnou mocí probíhá přes datové schránky.

Datovou schránkou se v českém právním řádu rozumí elektronické uložiště zřízené podle zákona č. 300/2008 Sb., o elektronických úkonech a autorizované konverzi dokumentů. Na základě tohoto zákona byl dne 1. 7. 2009 spuštěn Informační systém datových schránek (ISDS). Tento informační systém je určen k doručování mezi orgány veřejné moci a fyzickými a právnickými osobami. Zřízení datové schránky je povinné pro všechny orgány veřejné moci a dále právnické osoby. Pro ostatní subjekty je jejich zřízení dobrovolné. Z toho vyplývá, že se rozlišuje několik typů datových schránek a to datová schránka orgánu veřejné moci (OVM), právnické osoby (PO), podnikající fyzické osoby (PFO) a fyzické osoby (FO) [16]. Každá z těchto kategorií obsahuje podkategorie, ale chování vůči systému se váže pouze k uvedeným. Všechny schránky v systému mohou bezplatně doručovat do schránek typu OVM. Do ostatních schránek (s výjimkou pokud je odesíláno ze schránky OVM) lze doručovat pouze za poplatek a příslušná schránka musí mít povoleno přijímání komerčních zpráv. Za komerční zprávu se označuje jakákoliv zpráva, jejíž jednou stranou není datová schránka typu OVM.

Základním rozdílem ISDS oproti klasickým elektronickým podatelnám zřízeným většinou formou emailové schránky je forma jistoty uživatelů ISDS. Již od roku 2001 totiž měly orgány veřejné moci povinnost takové elektronické podatelny zřídit. Tuto povinnost jim stanovovalo nařízení vlády č. 304/2001 Sb., kterým se provádí zákon č. 227/2000 Sb., o elektronickém podpisu [1]. ISDS ale oproti klasickému emailu poskytují širší portfolio služeb, než je prosté přenesení zprávy od odesílatele k příjemci tak, jak to dělá klasický systém elektronické pošty. V prvé řadě veškerá komunikace probíhá uvnitř jednoho systému, tudíž riziko ztráty zprávy z důvodu odmítnutí v rámci komunikace mezi zpracovávajícími servery je eliminováno. Dále tento systém má přehled o zprávě od jejího vzniku až do jejího přečtení a je schopen kdykoliv poskytnout informace o stadiu doručování zprávy. To v případě klasického emailu není možné, neboť zainteresované servery mají vždy informace pouze o činnostech, které samy provádějí. Server odesílatele ve chvíli, kdy předá pomocí protokolu SMTP zprávu doručujícímu serveru, o ni přestává mít zájem a i kdyby chtěl, nemá prostředky, jak se o osudu předané zprávy dozvědět bližší informace. A právě schopnost přesně určit v kterémkoliv okamžiku stádium doručování zprávy dělá z ISDS speciální, a pro právní důsledky nezastupitelný, systém. Díky němu lze vygenerovat elektronickou doručenku, která je ekvivalentem klasické doručenky vzniklé během doručování prostřednictvím pošty. Stejně tak lze v prostředí ISDS dosáhnout fikce doručení v případě, že si adresát datovou zprávu nepřečte. I to má význam v rámci právních účinků doručování. V neposlední řadě se také ISDS liší od klasického emailu mírou anonymity, respektive neanonymity, jeho uživatelů. Dohledat vlastníka emailové schránky může být někdy takřka nemožné a doručovat listiny, které jsou určené pouze do vlastních rukou výhradně jen adresáta, je pomocí klasického emailu nemyslitelné. Účty v systému ISDS se ale zřizují výhradně na jméno adresáta a tak je zajištěno vlastnictví takové schránky oprávněnou osobou. I v případě automaticky zřizovaných schránek právnických osob tomu tak je, neboť ty se zřizují na základě zápisu do veřejného rejstříku a přístupové údaje do nich se doručují listinnou zásilkou do vlastních rukou statutárnímu orgánu právnické osoby. Ověření totožnosti adresáta, tedy právnické osoby, a osob s právem přístupu do datové schránky, tedy členů statutárního orgánu, zde bylo provedeno v době zápisu do veřejného rejstříku. K jedné datové schránce, nehledě na její typ, lze zřídit přístup pro několik osob [17].

Datové zprávy jsou reprezentovány pomocí XML. Veškeré zprávy, které projdou ISDS jsou elektronicky podepsány systémem a označeny časovým razítkem. Podpisem zprávy se zde rozumí nikoliv nahrazení elektronického podpisu odesílatele v jednotlivých dokumentech, ale podepsání binární podoby celé datové zprávy. Takto podepsanou zprávu lze uložit mimo ISDS a byl pro ni zaveden souborový formát s příponou .zfo. Soubory s touto příponou umí nativně číst aplikace Software602 FormFiller, popřípadě i další aplikace pracující s datovými schránkami. ISDS ukládá přečtené datové zprávy po dobu 90 dnů[1], poté je automaticky smaže. Je tedy více než vhodné datové zprávy archivovat a k tomu slouží jejich ukládání mimo ISDS ve formátu .zfo. Takto staženou zprávu lze pak ověřit, a to buď pouhým faktem, že byla správně podepsána ISDS a od podpisu se nezměnila, nebo sofistikovaněji pomocí ISDS. Systém si totiž pamatuje několik digitálních otisků (hash) a další vlastnosti zprávy a to i v případě, že byla ze systému již smazána. Na základě těchto informací je ISDS schopen určit, zda předkládaná zpráva je pravá a zda byla někdy systémem doručována [18][2].

[1] V případě, že ve schránce není aktivována zpoplatněná služba Datový trezor, která umožňuje uložit ve schránce předem zaplacený počet zpráv bez časového omezení.

[2] Technická specifikace ISDS a poskytovaných webových služeb je dostupná pouze po zaregistrování na https://www.datoveschranky.info/dulezite-informace/rozhrani-pro-aplikace-tretich-stran/registracni-formular

Uživatel může k datové schránce přistupovat dvojím způsobem. Může využít oficiální portál https://www.mojedatovaschranka.cz, ze kterého lze schránku plně ovládat, ale jehož využití v případě intenzivního používání ISDS není příliš komfortní. Druhým způsobem je využití aplikací třetích stran, které s ISDS umí komunikovat. Za tímto účelem poskytuje ISDS webové služby, pomocí kterých lze vytvořit aplikaci, která plně nahradí oficiální portál[3]. WSDL definice těchto služeb, jejich dokumentace a fórum mezi vývojáři aplikací a ISDS jsou dostupné pouze registrovaným uživatelům. Pro tvůrce aplikací je také dostupné testovací prostředí ISDS, vůči kterému mohou svoje produkty testovat.

Spuštění služby ISDS bylo velkým zlomem v elektronizaci komunikace se státní správou. Za dobu své existence se sešlo velké množství kritiků, ale také uživatelů, které si je nemohou vynachválit. Celý systém má zajisté své neduhy, a to jak technického rázu, tak i skutečností souvisejících s jejich provozováním (cenová politika, provozní cena atd.). Několik postřehů z období jejich spuštění (článek z 18. 1. 2010) lze nalézt v [19], výmluvně hovoří i anticena udělena v rámci soutěže Křišťálová lupa za rok 2010 [20] a hojné kritice podrobují ISDS na serveru Lupa.cz (články z tohoto serveru s tématikou datových schránek lze nalézt na http://www.lupa.cz/n/datove-schranky). Pravdou ale zůstává, že ISDS šetří čas i peníze občanům i firmám. Autor práce sám aktivně využívá ISDS při komunikaci s úřady státní správy a ze svého subjektivního pohledu je tato komunikace bezesporu pohodlnější než osobní návštěva úřadu.

[3] Pomocí webových služeb nelze upravovat nastavení datové schránky a obnovit již expirované heslo.

3 Analýza služeb insolvenčního rejstříku

Tato kapitola analyzuje fungování insolvenčního rejstříku z technického úhlu pohledu. Zabývá se strukturou a podstatou dat, které insolvenční rejstřík poskytuje, a též způsoby, jak s těmito daty pracovat.

Zevrubným popisem Informačního systému insolvenční rejstřík (ISIR) se zabývala již kapitola 2.4. Tato kapitola se bude zabývat způsobem, jakým jsou data v insolvenčním rejstříku zveřejňována a jak s nimi pracovat. Nejprve ale bude pomocí Diagramu 3.1 ukázáno, jaké procesy okolo insolvenčního rejstříku probíhají a jaké subjekty do nich zasahují.

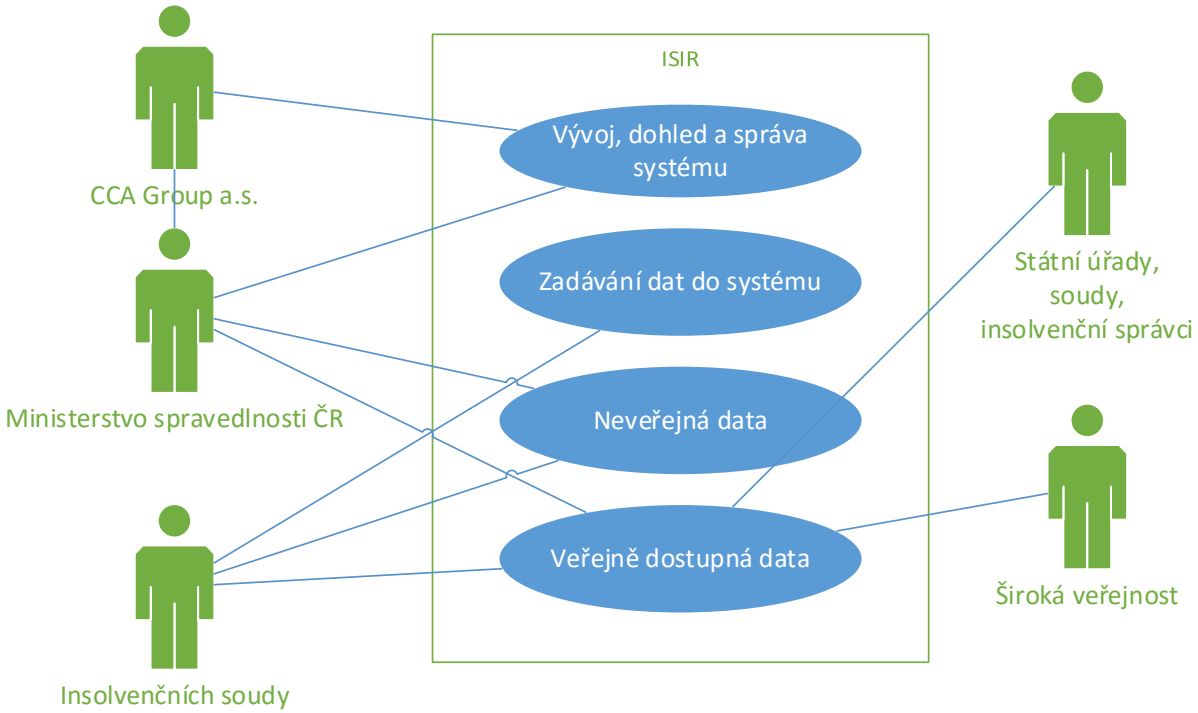


Diagram 3.1: Procesy a subjekty okolo insolvenčního rejstříku

Základními subjekty jsou Ministerstvo spravedlnosti ČR a insolvenční soudy. Oba tyto subjekty participují v majoritní části procesů, které se okolo insolvenčního rejstříku odehrávají a mají přístup k celému obsahu dat. Speciální postavení má firma CCA Group a.s., která celý systém vytvořila a dále na něm participuje v oblasti správy a vývoje. Na druhé straně stojí konzumenti dat, kteří data z insolvenčního rejstříku potřebují ke své činnosti. V současné chvíli se nebudeme zabývat způsobem, jakým data získávají. Největším konzumentem dat je široká veřejnost. Ta v majoritním případě přistupuje k datům jednotlivě a nejdůležitější informací pro ni je skutečnost, zda s jejím dlužníkem bylo zahájeno insolvenční řízení. Dále zde stojí státní úřady, které mohou data konzumovat podobným způsobem jako široká veřejnost, nebo na ně mít napojené svoje rejstříky a data do nich dále zapracovávat (příkladem je třeba Ministerstvo financí ČR s registrem ARES).

V dalším pokračování této práce budeme na insolvenční rejstřík nahlížet z pohledu široké veřejnosti. Ukažme nyní, jak vypadá struktura veřejných dat, jaké výstupy lze očekávat.

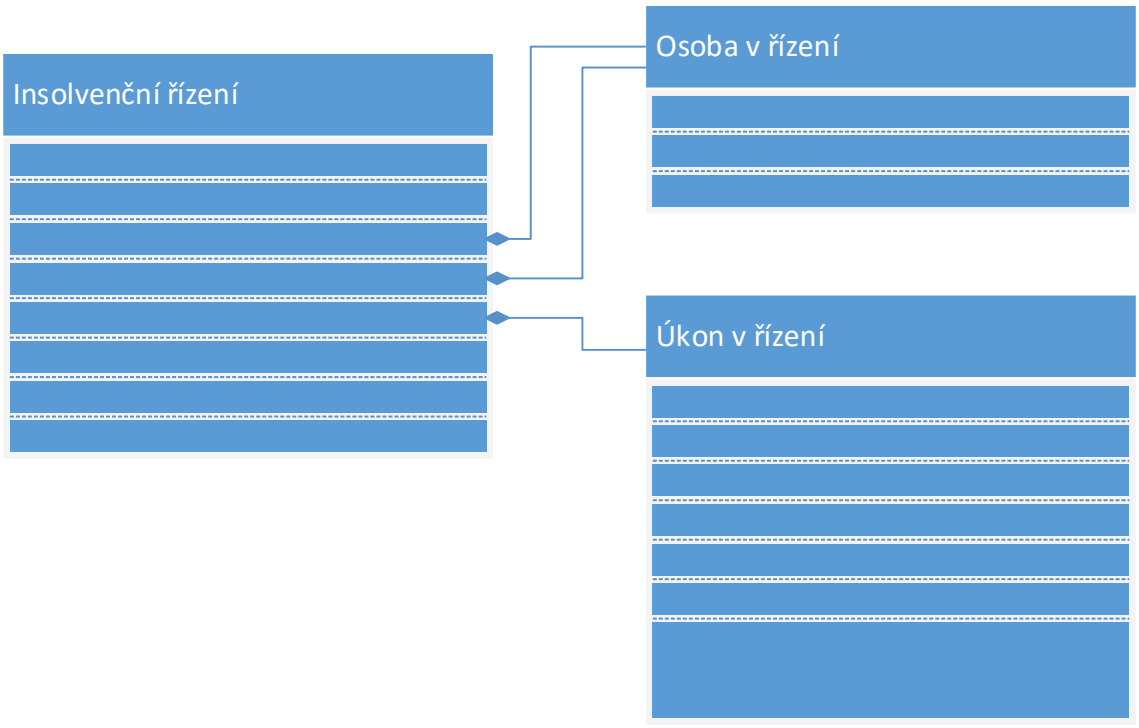


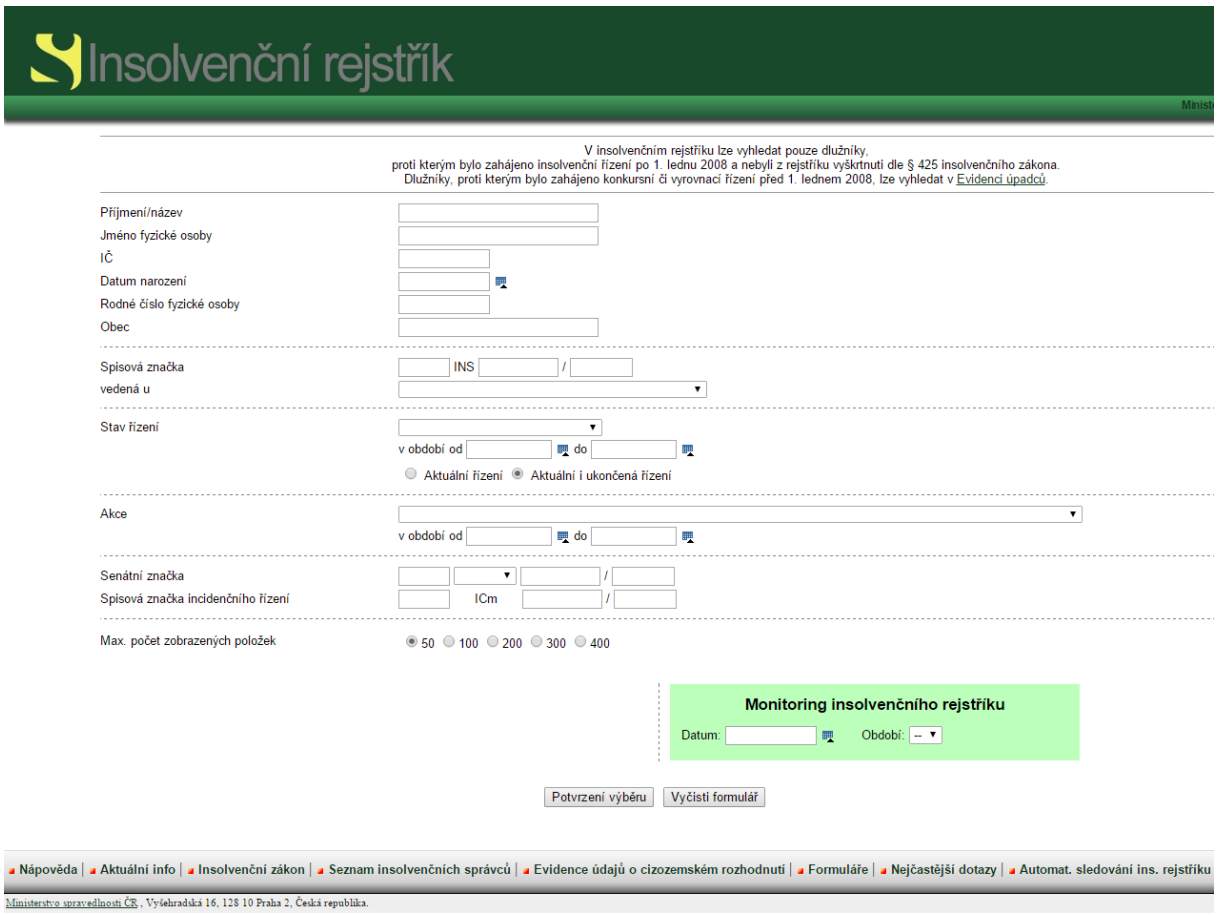
Diagram 3.2: Struktura veřejně dostupných dat

Diagram 3.2 zobrazuje základní datový prvek, kterým je Insolvenční řízení a podřízené datové prvky, které nesou další informace týkající se daného řízení. Atributy, pomocí nichž lze datový prvek jednoznačně identifikovat, jsou vypsány tučně a podtrženy. Pro podřízené prvky platí, že k jejich jednoznačné identifikaci je třeba propojit jejich identifikační atributy s identifikačními atributy nadřízeného prvku.

K datům v insolvenčním rejstříku lze přistupovat dvojím způsobem. Tím nejjednodušším je využít webové rozhraní na adrese https://isir.justice.cz. Pokud ale potřebujeme opakovaně lustrovat větší množství subjektů nebo data z insolvenčního rejstříku dále strojově zpracovávat, je praktické využít jednu z poskytovaných webových služeb.

3.1 Webové rozhraní

Jak již bylo řečeno, webové rozhraní je základní nástroj pro získávání dat z insolvenčního rejstříku. Po zadání adresy https://isir.justice.cz se uživateli zobrazí tato stránka.

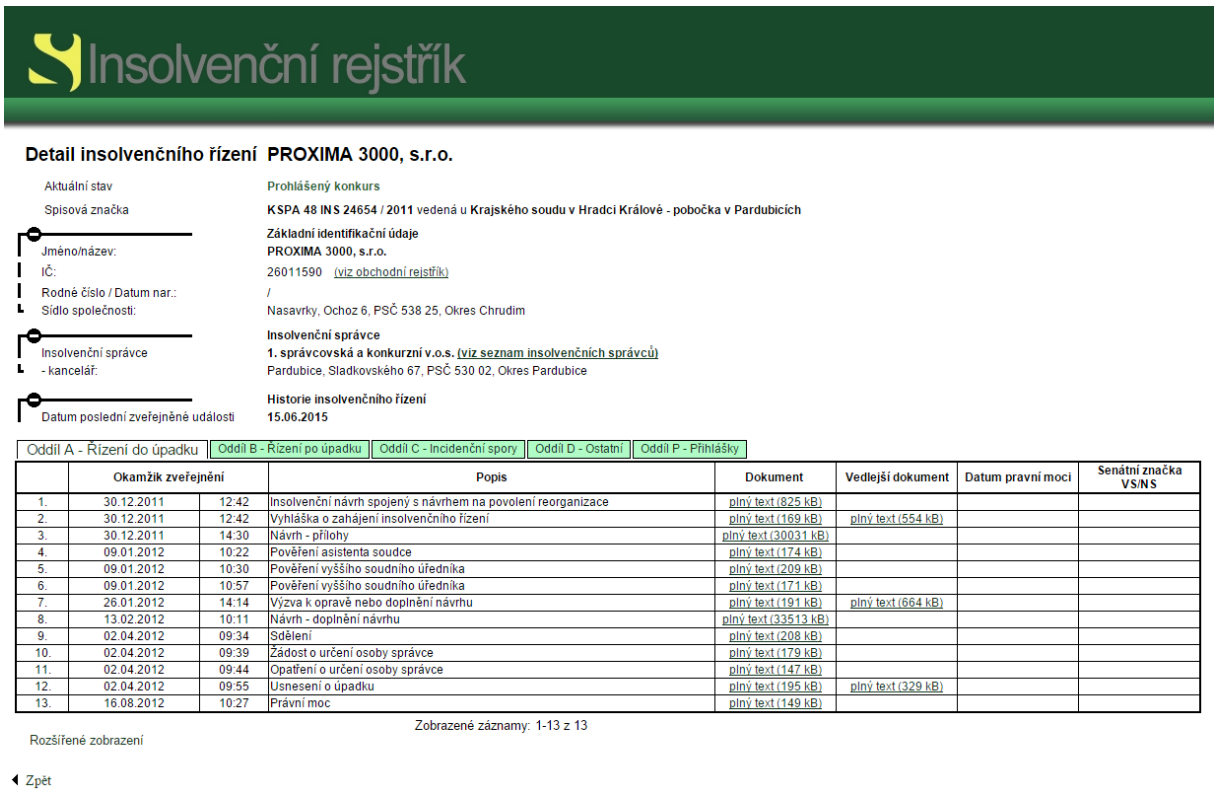


Obrázek 3.1: Vyhledávací formulář insolvenčního rejstříku

Obrázek 3.1 zobrazuje vyhledávací formulář pro lustraci v insolvenčním rejstříku a je z něho dobře patrné, že lze vyhledávat podle různých kritérií. Nejspolehlivější údaj, podle kterého lze vyhledávat, je u právnických osob identifikační číslo a u fyzických osob rodné číslo. S IČ u právnických osob není problém, neboť to lze bez problémů zjistit z jiných, veřejně dostupných databází, ale rodné číslo u fyzických osob nemusí věřitel vždy znát. V takovém případě má možnost vyhledávat podle jména a příjmení, data narození a obce, ve které má dlužník bydliště. Takto zadaná kritéria sice mohou nalézt více osob, ale z poskytnutých údajů by již věřitel měl být schopen identifikovat svého dlužníka. Na dalším obrázku je zobrazen výsledek vyhledávání v případě, že zadáme IČ osoby, která má záznam v insolvenčním rejstříku.

Obrázek 3.2: Výsledek vyhledávání v insolvenčním rejstříku

Obrázek 3.2 zobrazuje záznamy třech insolvenčních řízení, které byly nebo jsou s lustrovaným subjektem vedeny. V insolvenčním rejstříku jsou zaznamenány pouze subjekty, se kterými bylo nebo je vedeno insolvenční řízení. Pokud tedy zadáme vyhledávací kritéria, která vedou k subjektu, který nemá záznam v insolvenčním rejstříku (tj. není s ním vedeno insolvenční řízení), potom vyhledávací formulář nenalezne žádné výsledky. Na stránce s výsledky vyhledávání jsou vždy uvedeny jen základní údaje o nalezených dlužnících. Více informací lze zjistit po kliknutí na odkaz Detail. Zobrazený detail o spisu uvádíme na Obrázku 3.3.



Insolvenční rejstřík

Detail insolvenčního řízení PROXIMA 3000, s.r.o.

Aktuální stav: Prohlášený konkurs

Spisová značka: **KSPA 48 INS 24654 / 2011** vedená u **Krajského soudu v Hradci Králové - pobočka v Pardubicích**

Základní identifikační údaje

Jméno/název: **PROXIMA 3000, s.r.o.**

IČ: 26011590 (viz obchodní rejstřík)

Rodné číslo / Datum nar.: /

Sídlo společnosti: Nasavrky, Ochoz 6, PSČ 538 25, Okres Chrudim

Insolvenční správce

Insolvenční správce: **1. správcovská a konkurzní v.o.s.** (viz seznam insolvenčních správců)

- kancelář: Pardubice, Sladkovského 67, PSČ 530 02, Okres Pardubice

Historie insolvenčního řízení

Datum poslední zveřejněné události: **15.06.2015**

Oddíl A - Řízení do úpadku | Oddíl B - Řízení po úpadku | Oddíl C - Incidenční spory | Oddíl D - Ostatní | Oddíl P - Přihlášky

	Okamžik zveřejnění		Popis	Dokument	Vedlejší dokument	Datum právní moci	Senátní značka VS/NS
1.	30.12.2011	12:42	Insolvenční návrh spojený s návrhem na povolení reorganizace	plný text (825 kB)			
2.	30.12.2011	12:42	Vyhláška o zahájení insolvenčního řízení	plný text (169 kB)	plný text (554 kB)		
3.	30.12.2011	14:30	Návrh - přílohy	plný text (30031 kB)			
4.	09.01.2012	10:22	Pověření asistenta soudce	plný text (174 kB)			
5.	09.01.2012	10:30	Pověření vyššího soudního úředníka	plný text (209 kB)			
6.	09.01.2012	10:57	Pověření vyššího soudního úředníka	plný text (171 kB)			
7.	26.01.2012	14:14	Výzva k opravě nebo doplnění návrhu	plný text (191 kB)	plný text (664 kB)		
8.	13.02.2012	10:11	Návrh - doplnění návrhu	plný text (33513 kB)			
9.	02.04.2012	09:34	Sdělení	plný text (208 kB)			
10.	02.04.2012	09:39	Žádost o určení osoby správce	plný text (179 kB)			
11.	02.04.2012	09:44	Opatření o určení osoby správce	plný text (147 kB)			
12.	02.04.2012	09:55	Usnesení o úpadku	plný text (195 kB)	plný text (329 kB)		
13.	16.08.2012	10:27	Právní moc	plný text (149 kB)			

Zobrazené záznamy: 1-13 z 13

Rozšířené zobrazení

Zpět

Obrázek 3.3: Detail spisu lustrované osoby

Zde již vidíme veškeré veřejně přístupné informace týkající se spisu lustrované osoby. Vidíme zde aktuální stav insolvenčního řízení, identifikační údaje subjektu, údaje o insolvenčním správci a pět oddílů, do kterých se zapisují úkony účastníků řízení a insolvenčního soudu v průběhu jednotlivých fází insolvenčního řízení. U každého úkonu v každém oddílu je uvedeno jeho pořadí v takovém oddílu, datum a čas zveřejnění, popis úkonu, dokument, popřípadě vedlejší dokument, vztahující se k úkonu, datum právní moci pokud uveřejňovaný dokument této nabývá a senátní značku vrchního soudu / nejvyššího soudu. Ta je vyplněna v případě, že proti úkonu insolvenčního soudu bylo podáno odvolání, respektive dovolání. Ohledně významu jednotlivých oddílů a úkonů odkážeme laskavého čtenáře na zákon č. 182/2006 Sb., o úpadku a jeho řešení (insolvenční zákon), neboť jejich popis dalece přesahuje účel této práce. Navíc tyto nemají pro problematiku řešenou v této práci bližší význam.

3.2 Webová služba ISIR_WS

Další možností přístupu k datům v insolvenčním rejstříku jsou poskytované webové služby. V současné době ministerstvo spravedlnosti poskytuje veřejnosti přístup ke dvěma takovýmto službám. Obě služby využívají komunikaci pomocí SOAP protokolu. Z dokumentace k těmto službám lze také vyčíst, že celý systém je provozován na databázi Oracle 10g a zpřístupňován pomocí aplikačního serveru GlassFish.

Služba ISIR_WS je původní služba z roku 2008, ve kterém byl celý informační systém zprovozněn. Pomocí této služby lze strojově získat a zpracovat veškeré informace, které lze zobrazit pomocí webového rozhraní. Zákonitosti a skutečnosti popisované v této podkapitole vychází částečně z dokumentace [21] a částečně z analýzy akcí vrácených webovou službou. Konstrukce této služby je poněkud zvláštní. „Služba předpokládá vytvoření vlastních kopií databází na straně uživatelů.

Důvodem je požadavek vysoké dostupnosti.“ [21] Základním prvkem této služby je akce, což je datová entita identifikovaná jedinečným číselným identifikátorem. Pro pochopení toho, co je akce, dokumentace [21] ještě definuje událost. Událost v širším kontextu je ekvivalent úkonu v insolvenčním řízení, tj. jedné řádky v jednotlivých oddílech tak, jak zobrazuje Obrázek 7. Událost v užším kontextu pak jsou jednotlivé kroky uživatele vedoucí k zadání veškerých informací ohledně jednotlivých úkonů (událostí v širším kontextu) do systému. Např. pro úkon Usnesení o úpadku zadá uživatel do systému nejprve informaci o tom, že byl tento úkon proveden, připojí dokument ve formátu Adobe Acrobat PDF a zadá datum a čas kdy byl tento úkon proveden. Následně, mimo informační systém, úředník na insolvenčním soudě ten samý dokument, jaký byl připojen, vytiskne, vloží do obálky a odešle dlužníkovi (popřípadě k doručení použije ISDS). Ve chvíli, kdy je písemnost dlužníkovi doručena, nastává její právní moc. Doručení potvrdí dlužník doručujícímu orgánu na doručence a doručující orgán následně doručenku předá zpět soudu. Soud se tedy o tom, že nastala právní moc listiny, dozví až s určitým zpožděním. Jakmile se o této skutečnosti dozví, zadá úředník do informačního systému k již zveřejněnému úkonu informaci o nabytí právní moci. Tím provede další událost v užším kontextu. Lze tedy říci, že úkon (událost v širším kontextu) se skládá z několika událostí v užším kontextu. Dle dokumentace [21] je vygenerování akce reakcí informačního systému na událost v užším kontextu.

Webová služba poskytuje dvě metody a to getIsirWsPublicDatumDataRequest a getIsirWsPublicIdDataRequest. Jako vstupní proměnné akceptuje metoda getIsirWsPublicDatumDataRequest datum a jako výstup poskytuje pole s 1000 akcí bezprostředně následujících po tomto datu. Metoda getIsirWsPublicIdDataRequest akceptuje jako vstupní metodu číslo typu long (64 bit integer) a vrací opět vzestupně seřazené pole s 1000 akcí, které mají id následující po zadaném vstupním id.

3.2.1 Struktura datového typu akce

Dále se budeme zabývat XML strukturou datového typu akce, kterou udává následující tabulka (zdroj: [21])

Název elementu	**Popis**
Id	Unikátní sekvenční číslo události v ISIRu. Číslo bude generovat aplikace při zápisu události do aplikace.
datumZalozeniUdalosti	Datum a čas založení události v centrální databázi. S tímto datem je porovnáván vstupní parametr datumZalozeniPodnetu. Bude ve formátu dateTime – standard pro webové služby.
datumZverejenniUdalosti	Datum a čas zveřejnění události na příslušném soudě. Jedná se o datum, kdy byl podnět odeslán ke zveřejnění z lokální aplikace. Může se lišit od datumZalozeniUdalosti, kvůli době přenosu mezi lokální a centrální databází, nebo dalším vlivům . Bude ve formátu dateTime – standard pro webové služby.
dokumentUrl	Adresa dokumentu – služba nebude vracet obsah dokumentů. Pro ten bude vyhrazená jiná cesta. Element idDokument (sic!) bude obsahovat URI dokumentu.

Poznamka	Poznámka – obsahuje strukturované údaje k události – pro vybrané události zobrazuje formou XML dokumentu údaje např. o dlužníkovi. Struktura poznámky bude součástí definice číselníku událostí. Poznámka bude pouze u omezeného počtu událostí.
spisovaZnacka	Spisová značka řízení, ke kterému událost patří. Bude ve formátu <druh_věci> <běžné_číslo> / <ročník>, např. INS 56 / 2007. Spisová značka spojuje události, které k sobě patří.
typUdalosti	Typ události podle číselníku událostí [22].
popisUdalosti	Popis události podle číselníku událostí [22].
oddil	Oddíl, do kterého událost patří.
cisloVOddilu	Pořadové číslo v rámci oddílu, spolu se spisovou značkou a oddílem identifikuje událost.

Tabulka 3.1: Struktura datového typu akce

Popisy jednotlivých atributů v Tabulce 3.1 jsou v zásadě jasné. Je třeba si uvědomit, že kombinací atributů spisovaZnacka, oddil, cisloVOddilu a typUdalosti získáme jednoznačný identifikátor úkonu v insolvenčním řízení.

3.2.2 Struktura atributu Poznamka

Důležitým atributem u každé akce je Poznamka. Z té se uživatel dozví podrobnější údaje vztahující se ke konkrétní akci. „Aby byla webová služba co nejobecnější a přitom odolná proti častým změnám rozhraní, obsahuje samotné volání pouze základní údaje pro identifikaci události, které se akce týká. Samotná přenášená data obsahuje poznámka. Data jsou v poznámce strukturována formou XML dokumentu.“ [21]

Kompletní XSD schéma atributu Poznamka zobrazuje Diagram 3.3.

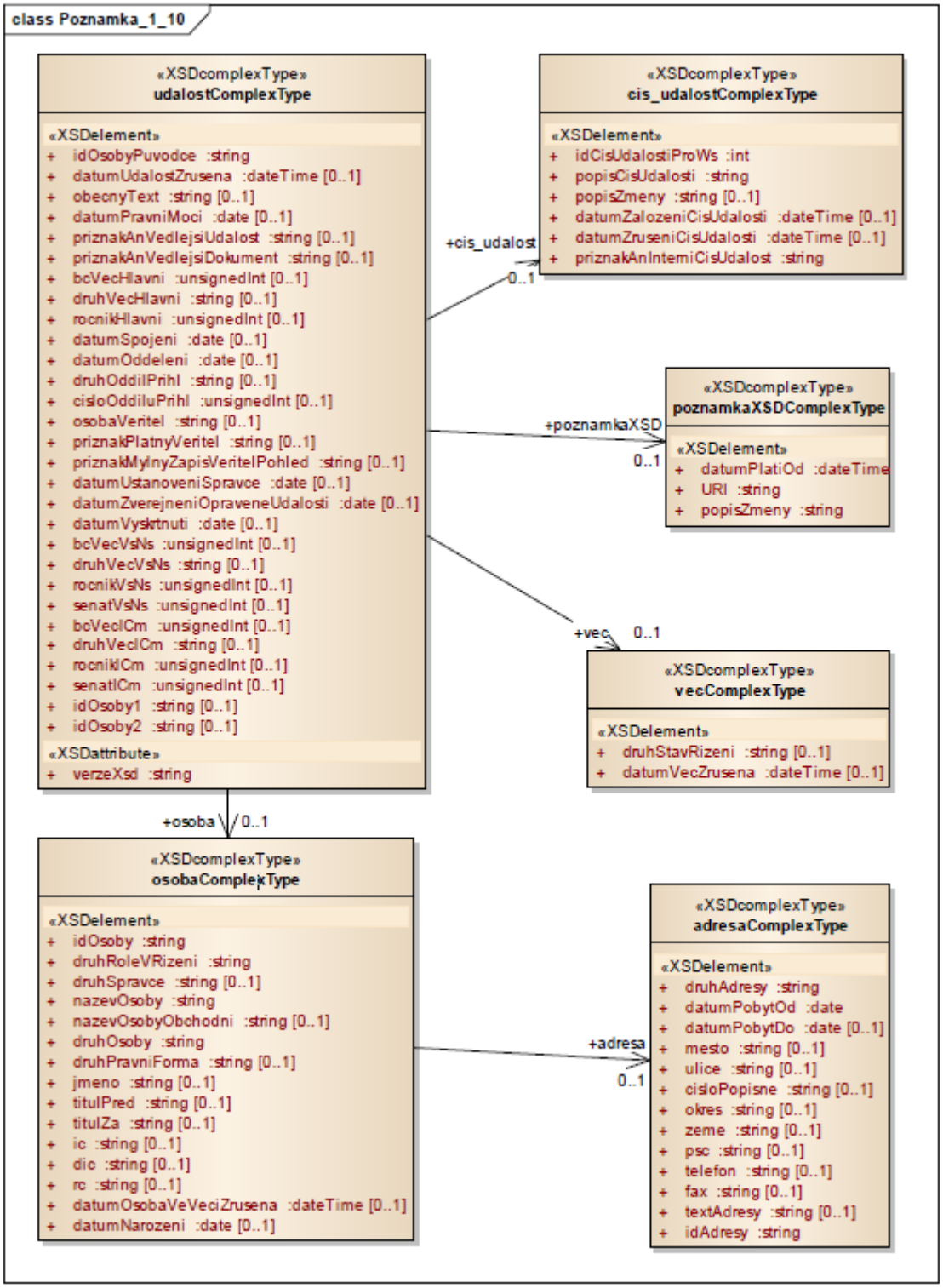


Diagram 3.3: XSD definice atributu Poznamka (zdroj: [21])

Struktura udalostComplexType udává informace o samotném úkonu. Při jejím detailnějším prohlédnutí lze zjistit, že její atributy jsou v zásadě totožné se sloupci tabulky z Obrázku 3.3. Celá struktura udalostComplexType může mít vnořené ještě další XML struktury. Struktura vecComplexType je uvedena v případě, že se úkonem mění stav řízení, tj. např. byl prohlášený úpadek dlužníka. Atributu druhStavRizeni odpovídá položka Aktuální stav z Obrázku 3.3. Struktura poznamkaXSDComplexType a cis_udalostComplexType jsou servisního charakteru. První jmenovaná je obsažena v případě, že došlo ke změně XSD definice XML struktury pro atribut

Poznamka. Druhá jmenovaná je obsažena v případě, že zaslaná akce provádí změnu v číselníku událostí [22]. Poslední strukturou je osobaComplexType, která v sobě ještě může mít vnořenou strukturu adresaComplexType. XML struktura osobaComplexType je v atributu Poznamka obsažena v případě, že se servisní akcí zavádí nebo mění u věci účastník. Pokud tato struktura obsahuje též vnořenou XML strukturu adresaComplexType znamená to, že se akcí zavádí nebo mění adresa u účastníka. Detailnější popis jednotlivých atributů a jejich významy nalezne laskavý čtenář v dokumentaci [21].

3.2.3 Servisní a informační akce

Z dokumentace [21] lze dále také zjistit, že akce jsou děleny do dvou kategorií, a to akce nesoucí informace o zveřejňovaných událostech (úkonech v insolvenčním řízení) a servisní akce, kterými se uživateli webové služby posílají podpůrné informace, které je třeba znát ke správné interpretaci akcí z první kategorie. Mezi takové patří např. akce definující účastníka, jeho adresu atd. Servisní akci od akce, která zveřejňuje událost, lze také rozeznat podle číselného id jejího typu. Dle číselníku událostí [22] existují tyto servisní akce[4].

Kód akce	Popis akce	Reakce klienta služby na akci
1	Změna osoby	Založit nebo zaktualizovat záznam o osobě
2	Změna adresy osoby	Založit nebo zaktualizovat adresu osoby
3	Změna věci	Změnit věc
4	Změna v číselníku událostí	Upravit lokální kopii číselníku
330	Změna XSD dokumentu poznámky	Změna zdrojového kódu klienta, který parsuje XML kód z atributu Poznamka
371	Informační zpráva	Informovat uživatele o textu zprávy, popřípadě nedělat nic
372	Změna stavu věci	Změnit záznam o stavu věci
405	Změna senátu věci	Změnit záznam o čísle senátu

Tabulka 3.2: Kódy servisních akcí

Většinu servisních akcí lze zpracovat bez zásahu obsluhy popřípadě tvůrce klienta. To neplatí pro kód akce 371, ve které se posílají předem neurčené zprávy bez bližší systematiky. Např. dne 4. 11. 2008 byla odeslána akce s id 192127 a s textem zprávy

> *„U insolvenčního řízení 4395/2008 byli po technické závadě chybně zveřejněni dva dlužníci, Chýlek Zdeněk a Investiční společnost FUTURUM, a.s. v likvidaci. Bylo provedeno zmylnění dlužníka Chýlek Zdeněk regulérním zasláním podnětu o jeho zmylnění ve věci. Tento dlužník bude zapsán dne 5.11.2008 pod novým číslem. Dále byla provedena oprava čísla senátu u věci 36INS2588/2008 z 36 na 89 regulérním zasláním podnětu o změně senátu.“*

[4] Pro jednoduchost uvádíme pouze kódy akcí, které se dle dokumentace mohou vyskytnout ve veřejné webové službě

Byť informace obsažené ve zprávě jsou pro rekonstrukci databáze důležité, lze tuto akci ignorovat, neboť (jak i její text naznačuje) byly opravy provedeny k tomu navrženými procesy a tudíž je tato zpráva nadbytečná. Stejně tak např. akce ze dne 25. 2. 2013 s id 6378278 nesoucí zprávu

> *„Dne 26.2.2013 bude vystaven nový číselník událostí. Bude založeno celkem 355 nových událostí týkajících se incidenčního řízení.“*

V případě, že klientská aplikace správně reaguje na akci s kódem 4, lze i tuto zprávu ignorovat, neboť všech 355 nových typů akcí bylo dne 26. 2. 2013 zasláno 355 akcemi s kódem 4. Podobně vypadají i ostatní akce s kódem 371, které jsou ve webové službě od data jejího zprovoznění registrovány.

Další servisní akcí, na kterou musí reagovat dokonce tvůrce klientské aplikace, je akce s kódem 330. Každá klientská aplikace musí pro správný běh parsovat XML strukturu, kterou obsahuje atribut Poznamka. O tom, jaké elementy se mohou v tomto atributu objevit a jak na ně reagovat, informuje tvůrce aplikace právě XSD definice. Pokročilejší programovací jazyky, jako např. Java nebo rodina .NET, dokonce programátorovi umožňují vygenerovat z XSD definice datovou třídu a následně z XML dokumentu vytvořit instanci takto vygenerované třídy. V případě, že se XSD definice změní, přestanou tyto techniky správně fungovat. Jedinou možnou reakcí na takovou situaci je aktualizace klientské aplikace. Z toho vyplývá, že samotná aplikace na akci s kódem 330 reagovat nemusí, ale tvůrce této aplikace by měl sledovat vývoj webové služby např. právě upozorněním v případě, že se akce s kódem 330 vyskytne. To ale není nic neobvyklého, tvůrce aktivně vyvíjené a podporované klientské aplikace by měl sledovat kompletně veškeré technické a implementační změny okolo webové služby a aplikaci jim přizpůsobovat.

Na akce nesoucí informaci by klientská aplikace měla reagovat tak, že správně rozpozná do kterého spisu (jednoznačně identifikovaného atributem spisovaZnacka), oddílu a pořadí (atributy oddil a cisloVOddilu) notifikovaná událost patří a tuto událost založit, popřípadě pozměnit dle informací, které aplikace z akce získá. Z tohoto pravidla ale existují výjimky, které jsou uvedeny v Tabulce 3.3.

Kód akce	**Popis akce**	**Reakce klienta služby na akci**
621	Usnesení o spojení věci k společnému projednávání	Standardně zařadit akci k události a navíc u dotčených spisů vyznačit spisovou značku druhého spisu
622	Oddělení spojené věci	
623	Oddělení spojené věci po PM	
924	Oddělení spojené věci po právní moci	
636	Znepřístupnění údajů o dlužníkovi dle § 425 IZ [8]	Vymazat, nebo jinak znepřístupnit, údaje o vedeném insolvenčním řízení

Tabulka 3.3: Kódy informačních akcií, na které je třeba speciálně reagovat

Na akce spojující nebo oddělující řízení není reakce nikterak složitá. Pouze je třeba kromě standardního zapracování akce ještě poznamenat u spisů, že došlo k oddělení, respektive sloučení, a poznamenat spisovou značku příbuzného spisu. Dále se budeme zabývat akcí s kódem 636. Ust. § 425 odst. 1 insolvenčního zákona uvádí: „Po uplynutí 5 let od nabytí právní moci rozhodnutí, jímž bylo skončeno insolvenční řízení, vyškrtne insolvenční soud dlužníka ze seznamu dlužníků a údaje o něm

v insolvenčním rejstříku znepřístupní. Skončí-li insolvenční řízení rozhodnutím podle § 142, vyškrtne insolvenční soud dlužníka ze seznamu dlužníků a údaje o něm v insolvenčním rejstříku znepřístupní do 15 dnů od doručení žádosti dlužníka; dlužník je oprávněn požádat o vyškrtnutí nejdříve po uplynutí 3 měsíců od právní moci rozhodnutí.“ [8]

Reakcí webové služby na takový úkon soudu je zaslání akce s kódem 636, na kterou by klientská aplikace měla reagovat znepřístupněním údajů ve své lokální databázi. Je logickým postupem dotčený spis, včetně všech úkonů, účastníků řízení a dalších informací s ním souvisejících, vymazat. Je tedy legitimní očekávat, že pokud se vyskytne akce s kódem 636, tak už žádná jiná nepřijde, spis je tím odstraněn a už se s ním nic dělat nebude a nemůže. Bohužel to tak není. Např. hned u prvního spisu v databázi insolvenčního rejstříku, INS 1/2008, se akce s kódem 636 objeví poprvé pod ID 6475455 dne 6. 3. 2013. Poté se objevuje znovu a to skoro každý den po dobu půl roku. Během této záplavy akcí s kódem 636 je dne 18. 3. 2013 vygenerována akce s ID 6609152 a kódem 5 (Insolvenční návrh) s příznakem na vedlejší dokument. Jako poslední je vygenerována akce s kódem 492 (Zápis v rejstříku ukončen) s ID 8791918 ze dne 25. 9. 2013. Bohužel nelze ani dedukovat, že akce s kódem 492 uzavírá celý spis a se smazáním počkat až na tuto událost, neboť u drtivé většiny spisů je akce s kódem 636 poslední zaslaná, což je i náš původní předpoklad. Neplatí tedy, že po akci 636 nemůže být zaslaná jiná akce pro stejný spis a správnou reakcí je pravděpodobně takovou akci ignorovat, neboť spis již stejně nemá být přístupný. O tom ale dokumentace [21] mlčí.

Dále ještě stojí za zmínku skutečnost, že neexistuje servisní ani informační akce, která by indikovala založení spisu. Dle dokumentace [21] se spis zakládá první zaslanou akcí náležející k danému spisu, nejčastěji akcí s kódem 5 (Insolvenční návrh), ale není to pravidlo, následovanou servisními akcemi s kódem 1 a 2, což je založení a přiřazení osoby ke spisu a adresy k této osobě.

Pro zajímavost je ještě v Tabulce 3.4 uveden počet poskytnutých akcí pro roky 2008 až 2015.

Rok	Počet akcí za rok	Průměrný počet akcí za den
2008	197 134	540,09
2009	500 630	1 371,59
2010	828 930	2 271,04
2011	1 573 003	4 309,60
2012	2 508 232	6 871,87
2013	3 782 135	10 362,01
2014	5 273 234	14 447,22
2015	4 857 205	13 307,41

Tabulka 3.4: Počet poskytnutých akcí v jednotlivých letech

3.3 Webová služba ISIR_CUZK_WS

Tato služba je doplňkovou službou k původní službě ISIR_WS. Informace v této kapitole jsou čerpány z dokumentace [23] a webu [24]. Tato služba umožňuje online dotazování na řízení vedená v insolvenčním rejstříku za pomoci identifikačních údajů o dlužníkovi. Webová služba neposkytuje

kompletní informace, které insolvenční rejstřík obsahuje. V zásadě se jedná o službu, pomocí které lze bez rekonstrukce databáze insolvenčního rejstříku tak, jak ji předpokládá služba ISIR_WS, odpovědět na otázku: Je můj dlužník v insolvenci? Dle dokumentace byla služba spuštěna na popud Českého úřadu zeměměřičského a katastrálního (odtud její název) a provozovatel se posléze rozhodl v roce 2014 službu zpřístupnit i veřejnosti. Implementace této služby není cílem této práce, neboť neposkytuje veškeré veřejně dostupné informace.

Dle dokumentace [23] se služba nedotazuje ostré databáze insolvenčního rejstříku, ale každou hodinu si vytváří vlastní kopii této databáze, nad kterou poté pracuje. Zřejmě se jedná o snahu zachování funkčnosti celého systému v případě přetížení ze strany této služby. Motivace této snahy zřejmě není zcestná, neboť jak již bylo uvedeno v kapitole 2.4, o informace z insolvenčního rejstříku má zájem velký okruh subjektů. Lze si představit, že pokud by subjekty, jako je například mobilní operátor, implementovaly tuto službu do svých informačních systémů a prověřovali periodicky své klienty, mohlo by dojít k obrovskému množství dotazů v krátkém časovém intervalu. Nicméně jsme nenalezli žádnou informaci o tom, že by došlo k přetížení a nedostupnosti této služby z důvodu přemíry dotazů ze strany klientských aplikací.

Služba poskytuje pouze metodu nazvanou getIsirWsCuzkData. Vstupní proměnné této metody uvádí Tabulka 3.5 (citace: [23]).

Parametr	**Typ**	**Popis**
ic	`VARCHAR2`	IČ osoby úpadce/dlužníka
rc	`VARCHAR2`	Rodné číslo osoby úpadce/dlužníka
druhVec	`VARCHAR2`	Druh věci – dílčí atribut spisové značky
bcVec	`INT`	Běžné číslo věci – dílčí atribut spisové značky
rocnik	`INT`	Ročník věci – dílčí atribut spisové značky
nazevOsoby	`VARCHAR2`	Název právnické osoby/příjmení fyzické osoby úpadce/dlužníka
jmeno	`VARCHAR2`	Jméno osoby úpadce/dlužníka
datumNarozeni	`DATE`	Datum narození osoby úpadce/dlužníka
maxPocetVysledku	`INT`	Maximální počet záznamů, které vrací WS
filtrAktualniRizeni	`VARCHAR2`	Určuje, zda výsledek bude obsahovat pouze probíhající řízení

Tabulka 3.5: Vstupní proměnné metody getIsirWsCuzkData

Ještě před tím, než webová služba začne vyhledávat podle zadaných kritérií, provede kontrolu, zda jsou zadány vyhledávací kritéria v minimální kombinaci pro spuštění vyhledávání. Tyto minimální kombinace uvádí Tabulka 3.6 (citace: [23]).

ic	rc	druhVec	bcVec	Rocnik	nazevOsoby	Jmeno	datumNarozeni	maxPocetVysledku	filtrAktualniRizeni
X									
	X								
					X				
			X	X					
					X	X			
					X		X		

Tabulka 3.6: Minimální kombinace vstupních parametrů pro metodu getIsirWsCuzkData

Metoda dále obsahuje jednu výstupní proměnnou, do které zapíše informace o proběhlém vyhledávání a případně informaci o tom, zda nastala chyba a jaká. Návratovou hodnotou metody je pak pole s nalezenými výsledky. Obecně z takto nalezených výsledků vyhledávání lze zjistit základní informace o insolvenčním řízení a o osobě dlužníka. Pro detailnější popis odkážeme laskavého čtenáře na dokumentaci [23].

3.4 Hodnocení poskytovaných služeb insolvenčního rejstříku

Jak bylo v této kapitole uvedeno, veřejně přístupné jsou tři způsoby čerpání dat z insolvenčního rejstříku. Prvním způsobem je webové rozhraní. Pomocí webového rozhraní lze získat poměrně jednoduchým způsobem veškeré informace, které věřitel o svém dlužníkovi potřebuje získat. Webové rozhraní je správně vykreslované v široce používaných prohlížečích (Internet Explorer, Firefox, Chrome). Design je poměrně konzervativní s absencí jakýchkoliv AJAX prvků, ale to v tomto případě nevadí. Konzervativní design lze ze strany státní moci chápat a uživatel nemá pocit, že pracuje s webovou stránkou vzniklou v prehistorických dobách Internetu. Stejně tak absence AJAX ovládacích prvků nemá za následek krkolomné nebo zdlouhavé proklikávání a načítání. Jediná výtka vůči webovému rozhraní by snad mohla směřovat z pozice uživatelů mobilních zařízení, zejména pak smartphonů, neboť stránka není pro tyto zařízení optimalizována a interakce na zařízení s malým dotykovým displejem není zrovna komfortní. Co se týče funkčnosti, tak lze pomocí webového rozhraní vyhledávat v insolvenčním rejstříku pomocí široké palety kritérií, výsledky jsou přehledně strukturované a poskytují komplexní informace.

Dalším možným způsobem čerpání informací je využití jedné z webových služeb. Primárně používanou je webová služba ISIR_WS. U té je největší výtkou samotná logika veřejné služby. Nelze sdílet názor autorů webové služby, kdy údajně z důvodu vysoké dostupnosti je každý, kdo chce s daty z insolvenčního rejstříku nějak dále pracovat, nucen vytvořit vlastní kopii databáze. Ke konstrukci

služby bylo patrně přistupováno s obavou, že by hrozilo takové množství dotazů na webovou službu, že podkladový hardware takovou zátěž nezvládne. Autoři se proto rozhodli distribuovat zátěž mezi uživatele toužící po automatickém zpracování a zodpovědnost za dostatečný výpočetní výkon pro jejich potřeby nechávají na nich. Avšak podle názoru autora této práce není příliš reálné, aby celá služba byla zásadně přetěžována takovým způsobem, aby v dnešní době dostupný hardware takovou úlohu nezvládl. Ostatně tomuto názoru svědčí i pozdější spuštění webové služby ISIR_CUZK_WS.

Konstrukce webové služby by správně měla být taková, že uživatel se zeptá webové služby na konkrétní informaci a tato mu ji vrátí, tedy obdobně jako již zmíněná později zavedená služba ISIR_CUZK_WS. Avšak webová služba ISIR_WS je konstruována na principu poskytování změn. Je to stejné, jako kdybychom chtěli znát cenu benzinu Natural 95 dne 1. 6. 2015 na jedné z čerpacích stanic v Plzni a webová služba by nám odpověděla, že dne 1. 1. 2008 na této stanici stál Natural 95 31,- Kč za 1 l a dále nám poskytovala informace o tom, o kolik se cena měnila každý den od 1. 1. 2008 až do 1. 6. 2015. Navíc by nám mezi těmito informacemi též poskytla stejné informace pro všechny čerpací stanice v celé ČR, což nás v tu chvíli naprosto nezajímá.

Místo toho, aby autoři webové služby nabídli rozhraní, které umožňuje naprosto jednoduchým způsobem provést různé běžné dotazy přímo na službu, např. „vypiš veškeré úkony u insolvenčního řízení INS AA/YYYY“, nutí uživatele webové služby vytvořit vlastní lokální databázi a tyto běžné dotazy pak provádět vůči ní. Tento handicap neodstraňuje ani dodatečně spuštěná webová služba ISIR_CUZK_WS, která sice již nepředpokládá rekonstrukci lokální databáze, ale poskytuje pouze základní údaje o insolvenčním řízení, tedy ji nelze považovat za plnohodnotnou náhradu služby ISIR_WS.

Další velkou výtkou je zmatenost navrženého řešení a nedostatečná dokumentace. Jen podle dokumentace [21] je velice obtížné poznat logickou strukturu celé služby a hlavně strukturu dat. Bez toho, aniž by uživatel detailně zkoumal vrácené výsledky z webové služby a měl určitou elementární znalost ohledně insolvenčního řízení, by byl návrh konzumačního algoritmu podstatně složitější až skoro nemožný.

Velice negativně lze též vnímat způsob začlenění atributu Poznamka do odpovědi webové služby. Samotná odpověď od webové služby je strukturována do XML dokumentu. Poznámka je opět XML dokument a tento je vložen přímo do atributu Poznamka v XML struktuře odpovědi. Pro ilustraci je uveden XML response z webové služby:

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header/>
  <soapenv:Body>
    <ns2:getIsirWsPublicDataResponse xmlns:ns2="http://isirpublicws.cca.cz/types/">
      <result>
        <cas>2015-03-27T18:05:18.000</cas>
        <id>17702964</id>
        <idDokument>
             https://isir.justice.cz:8443/isir_ws/doc/Document?idDokument=17935898
        </idDokument>
        <poznamka>
          <![CDATA[
            <?xml version="1.0" encoding="UTF-8"?>
            <tns:udalost xmlns:tns="http://www.cca.cz/isir/poznamka"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" verzeXsd="1_10"
            xsi:schemaLocation="http://www.cca.cz/isir/poznamka https://isir.jus-
            tice.cz:8443/isir_ws/xsd/poznamka_1_10.xsd">
            <idOsobyPuvodce>KSJICCB</idOsobyPuvodce>
            <vec>
              <druhStavRizeni>ODDLUŽENÍ</druhStavRizeni>
            </vec>
            <priznakAnVedlejsiDokument>T</priznakAnVedlejsiDokument>
          </tns:udalost>
          ]]>
        </poznamka>
        <spisZnacka>INS 14313/2012</spisZnacka>
        <typ>468</typ>
        <typText>Usnesení o uložení povinnosti</typText>
        <oddil>B</oddil>
        <poradiVOddilu>5</poradiVOddilu>
      </result>
    </ns2:getIsirWsPublicDataResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Předně je velice těžce pochopitelný důvod tohoto řešení. V dokumentaci [21] se jako důvod sice uvádí stabilita rozhraní a jeho ochrana před častými změnami, ale v případě, kdy dojde ke změně poznámky, je třeba i tak webovou službu i klientské aplikace aktualizovat. Tedy k žádné ochraně služby před změnami nedošlo, jen se problém převalil z WSDL definic webové služby do XSD definice XML struktury poznámka. Správnější by bylo řešení, kdy atribut Poznamka by prostě dále obsahoval další vnořené atributy tak, jak je u XML struktur běžné. Pokud už ale musí být věc řešena tak, jak je nyní, bylo by mnohem čistějším řešením, kdyby vnořený XML dokument byl alespoň nějakým způsobem serializován, např. pomocí base64 kódování apod.

Další velkou výtkou je bezesporu akce s ID 636 (Znepřístupnění údajů o dlužníkovi dle § 425 IZ). Dokumentace [21] ohledně této akce úplně mlčí a uživatel webové služby tak musí těžce vysledovávat vzorec chování webové služby před a po zaslání této akce. Logicky by se nabízela varianta, že toto je úplně poslední akce, která se u daného insolvenčního řízení může vyskytnout, ale to bohužel není pravda, byť to odporuje smyslu této akce.

Výtek k současnému řešení dané problematiky by bylo možno naleznout více, avšak v této podkapitole byly zmíněny pouze ty, které jsou shledávány nejzávažnější nebo nějakým jiným způsobem zdůraznění hodné. Celá implementace služby působí velice neprofesionálním dojmem a navozuje pochybnosti, zda nebylo politickým zadáním splnit povinnosti o zpřístupnění dat uložené zákonem, ale zároveň tak, aby zájemcům o tato data byla jejich práce maximálně ztížena. O

neprofesionalitě v implementaci webové služby hovoří velice výstižně názvy jednotlivých proměnných a metod. Až do verze služby 2.0 se obě hlavní metody nazývaly výstižně a popisně getIsirPub001 a getIsirPub0012. Stejně tak kombinace českého a anglického jazyka u názvů proměnných a metod působí poněkud komicky, avšak zde se velice promítá subjektivní názor každého jedince na problematiku používání českého jazyka v IT prostředí a autor této práce zastává názor jednotného použití jazyka (anglického) ve zdrojovém kódu. Dne 16. 9. 2015 došlo ke spuštění služby ve verzi 2.0, ve které došlo k refaktoringu a změně rozhraní do podoby, kterou popisuje tato práce.

4 Návrh datové pumpy

Tato kapitola obsahuje návrh prostředí, které komunikuje s webovou službou poskytovanou provozovatelem Informačního systému insolvenční rejstřík a to jak databáze, do které aplikace data ukládá, tak i samotnou aplikaci, která data z ISIR těží, zpracovává a ukládá do navržené databáze.

Datovou pumpou se v této práci rozumí algoritmus, který čerpá data z webové služby ISIR_WS, a která byla popsána v předchozí kapitole, takto získaná data zpracovává a ukládá pro budoucí využití. Výsledkem činnosti datové pumpy je databáze, ze které lze získat informace jednoduchým a konvenčním způsobem. Nejedná se tedy o ekosystém, který by byl navržen k použití koncovým uživatelem, ale o mezičlánek, který nalezne využití ze strany různých informačních systémů, tj. např. účetních systémů, spisových systémů atd. Zařazení datové pumpy a vzniklé databáze do celého procesu znázorňuje Diagram 4.1.

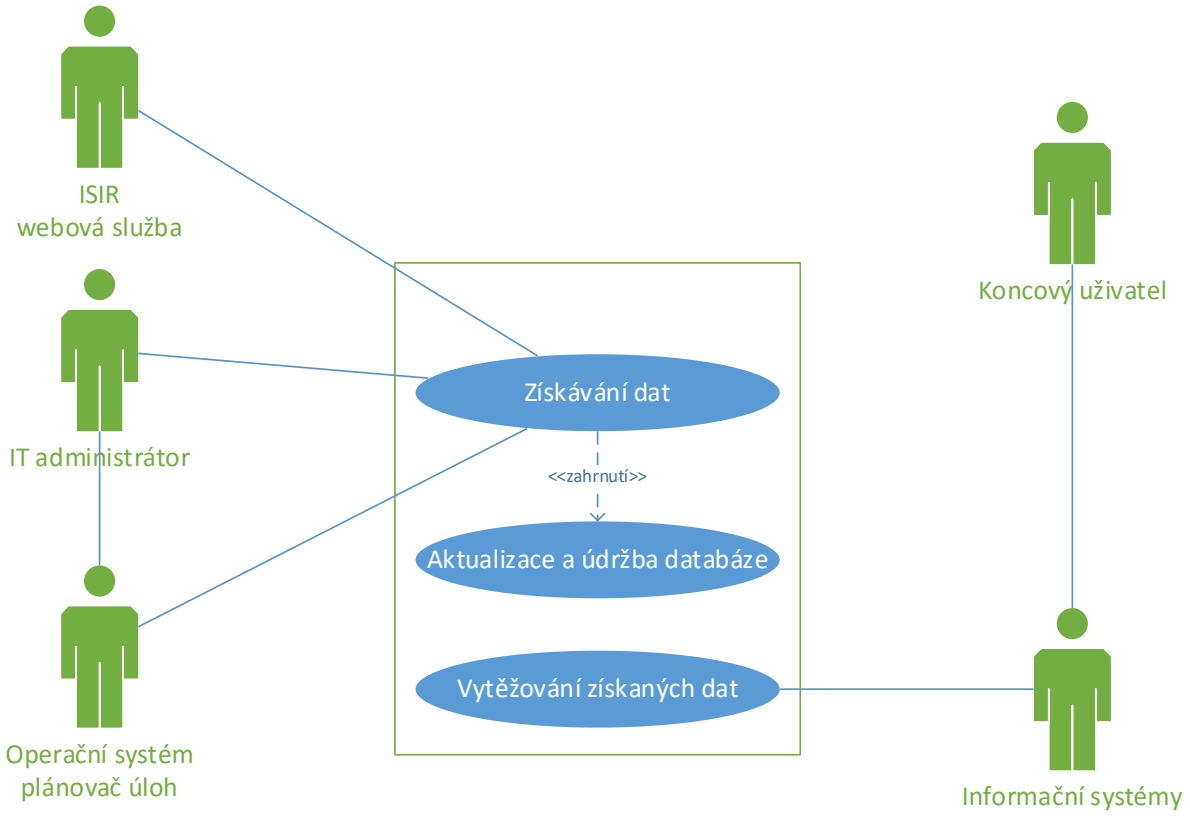


Diagram 4.1: UML use case diagram návrhu

4.1 Zpracování a způsob uložení dat

Jelikož data, která jsou získaná z webové služby, jsou vlastně inkrementálním popisem změny stavu celé databáze v čase, tedy strukturovanější transakční log, není vhodné taková data pouze stáhnout a nějakým způsobem uložit, neboť takto vzniklá databáze by nebyla sto posloužit pro vyhledávání dat a získávání informací. Primární funkcí datové pumpy je tedy na základě přijatých informací postupně rekonstruovat samotnou databázi do podoby, která odpovídá databázi provozovatele celého systému ISIR. Předchozí věta je lehce nadnesená, neboť taková databáze bude zajisté nést více informací a to už jen z faktu, že krom veřejně přístupných dat se v ní nachází i data veřejně nepřístupná, nicméně

cílem je vytvořit co možná nejpřesnější repliku databáze tak, aby stejný dotaz nad originální databází a nad replikou vrátil stejný výsledek.

Pro ukládání dat bude použita relační databáze. Relační databáze je zvolena z důvodu jejích výhod v podobě nástrojů pro zachování integrity dat, přirozenou reprezentací zpracovávaných dat, možností vazeb, indexů, primárních klíčů atd. Při vhodném nastavení všech těchto vlastností bude databáze pracovat svižně i při větším rozsahu uložených dat. Je třeba mít na paměti, že při zpracování bude muset datová pumpa nejen uložit novou změnu do databáze, ale také zjistit současný stav a rozhodnout se, jak změnu interpretovat a jak ji uložit. Například v případě založení spisu webová služba neposílá samostatnou akci značící založení nového spisu, ale spis se zakládá prvním úkonem. To znamená, že při každé došlé akci musí datová pumpa kontrolovat, zda je spis, na který akce odkazuje, již zaveden, nebo zda se jedná o onu první akci, která rovněž zakládá i spis. Tím pádem je vhodné zvolit takový způsob uložení dat, který je schopen po zadání konkrétní vlastnosti rychle vyhledat požadovanou informaci. Tuto výhodu následně použijeme i mimo datovou pumpu, kdy databáze bude sloužit svému primárnímu účelu, tedy získávání dat o insolvenčních řízeních.

Zpracováním dat se v případě datové pumpy myslí algoritmus, který rozumí logice a konstrukci akcí získávaných z webové služby a tyto umí správně interpretovat a pomocí nich rekonstruovat výše uvedenou databázi. Konstrukce takového algoritmu tedy vychází z poznatků získaných analýzou dat poskytovaných insolvenčním rejstříkem a z analýzy webové služby provedené v předchozí kapitole a dále též z konstrukce databáze, do které mají být data převedena.

4.2 Návrh databáze pro uložení dat

Jelikož cílem je vytvořit co možná nejpřesnější repliku originální databáze, jejíž struktura ovšem není známá, je třeba vycházet ze znaků a zákonitostí, které bylo možno vypozorovat jednak ze samotných dat a dále též z vlastností webové služby. Při návrhu databáze bylo rozhodnuto, že jako referenční klíče budou využívány identifikátory používané u webové služby, tzn., nebudou se vytvářet vlastní identifikátory a ty se následně přiřazovat jednotlivým atributům.

Základním elementem, na který se vážou veškeré další informace, je spis. Spis je reprezentovaný spisovou značkou. V databázi by tedy měl existovat seznam známých spisů s potřebnými informacemi o nich. Ve světě relačních databází naplníme tento požadavek existencí tabulky, kterou budeme nazývat `data_cases`, kde jednotlivé spisy budeme identifikovat pomocí unikátní spisové značky poskytnuté z webové služby. Pro návrh dalších tabulek se musíme zabývat vazbou mezi daty, která chceme uložit, a spisem. V případě, že se jedná o vazbu typu 1:1, budeme se důsledně snažit uvést tyto informace v tabulce `data_cases`. Takto budou uloženy veškeré informace týkající se spisu pouze jednou a navíc k nim přiřazeny informace, které nepotřebují další strukturování, respektive informace, které lze uložit v jednom z formátů podporovaných konkrétní implementací relační databáze (např. číslo, datum, text atd.).

Jak již bylo uvedeno výše, každý spis se skládá z úkonů, které učiní účastníci daného řízení nebo soud. Takových akcí je v každém spise více, nevystačíme si tedy pouze s jednou tabulkou a jejími sloupci. Vazba mezi spisem a úkony je vazba typu 1:N. Pro realizaci vazby 1:N v prostředí relačních databází použijeme novou tabulku, kterou budeme dále nazývat `data_actions`, a cizí klíč, který bude odkazovat na jednoznačný identifikátor do tabulky `data_cases`. Tímto cizím klíčem bude spisová značka. Každý úkon budeme identifikovat kombinací spisové značky, oddílu, pořadí v oddílu a typem

akce. Takovýto identifikátor je pro každý úkon jedinečný. Další informace poskytované webovou službou a vztahující se k jednotlivým úkonům již mají na úkon vazbu typu 1:1 a tedy budou uloženy v tabulce `data_actions`.

Dále se ke spisům vztahují jednotlivé osoby. Těmito osobami jsou dlužník, insolvenční správce a věřitelé. V implementaci webové služby se mezi těmito osobami rozlišuje jen pomocí atributu druhRoleVRizeni u entity osobaComplexType (viz Diagram 3.3). Ostatní vlastnosti mají všechny druhy osob více méně stejné, a tudíž lze zvolit možnost, že všechny druhy osob jsou uloženy ve stejné tabulce. Tím se ze vztahu osob vůči spisu stává vazba typu 1:N. Podobně jako v případě úkonů, zavedeme pro osoby novou tabulku, kterou budeme dále nazývat `data_persons`. Jelikož ale u jednoho spisu může být (a bude) více osob, ale i jedna osoba může být přiřazena k více spisům (zejména insolvenční správci) je výsledný vztah mezi tabulkou `data_cases` a `data_persons` typu M:N. Zde je třeba poznamenat, že jedna a tatáž osoba nemusí být jedinečná napříč celým systémem. Dokumentace [21] hovoří o tom, že atribut osoba je jedinečný ve vztahu k příslušnému soudu, tedy osoba může být v databázi zavedena vícekrát. Tato vlastnost má za následek duplicitu dat a bylo by vhodnější řešit daný problém jinak, nicméně snaha agregovat záznamy tak, aby nebyly duplicitní, by vedla k tomu, že bychom museli zkoumat jednotlivé identifikátory u spisů, poznamenat si je, a následně vždy odkazovat na příslušnou osobu jedinečnou napříč celou databází. To by ještě bylo možné vyřešit, ale dalším problémem je určitá nekonzistentnost v rámci různých záznamů reprezentujících stejnou osobu. Například Všeobecná zdravotní pojišťovna, IČ: 41197518, je v databázi zavedena (ke dni sepisu tohoto odstavce) celkem 328 krát, a to v pozici věřitele. To je v rozporu s tím, jak bychom měli navrhovat databázi, neboť každá informace by měla být uvedena pouze jednou a při dalším použití by se měl systém na tuto informaci pouze odkázat. Bohužel tyto záznamy nejsou naprosto totožné, protože byť se jedná o jeden subjekt reprezentovaný jedinečným IČ, ostatní vlastnosti tohoto subjektu se mění. Tak například již samotný název, který by neměl činit problém, se v databázi vyskytuje v 49 podobách (např. Všeobecná zdravotní pojišťovna České republiky, Všeobecná zdravotní pojišťovna ČR, VZP ČR Regionální pobočka-KP pro hl.m. Prahu a Středočeský kraj, atd.). Tento stav je dle názoru autora této práce dán dvěma faktory. Jednak tím, že pracovníci na jednotlivých soudech prostě zanášejí data do systému dle vlastního rozhodnutí, nerespektujíc žádné celosystémové konvence (ať již z důvodu nedostatečného proškolení obsluhy nebo z důvodu jejich prosté ignorace ze strany této obsluhy) a dále pravděpodobně nedostatečnou kontrolou vstupu do systému, kdy systém dovolí zadat takovéto duplicity, ba dokonce s nimi počítá. To má za následek nekonzistentnost dat, kdy v extrémním případě jeden jediný subjekt (identifikovaný stejným IČ), se v databázi nazývá různými názvy, má různá sídla atd., nehledě na překlepy a jiné obdobné chyby, které jsou obecně častým nešvarem v celé databázi. Avšak autor této práce nemá dostatek informací ohledně vzniku systému, tudíž mu ani nepřísluší hodnotit strukturu systému, neboť zde mohlo zásadní roli zahrát i politické zadání popřípadě nemožnost dohody napříč jednotlivými orgány justice, a tvůrce systému byl v těžké situaci a před rozhodnutím, zda vyhovět konvencím navrhování software anebo naprosto protichůdnému požadavku ze strany klienta – pak je pochopitelné, že přání klienta mělo přednost. Z těchto důvodů by případná agregace dat měla za následek nutnost vyřešit způsob, jak interpretovat změny u subjektů se stejným IČ a ať už by byla odpověď na tuto otázku jakákoliv, vždy by výsledky vrácené z replikované databáze byly rozdílné od těch z databáze originální, což by bylo v rozporu s původním cílem, tedy s totožností odpovědí na stejné dotazy. Ze všech těchto důvodů bylo tedy rozhodnuto následovat způsob, kterými jsou osoby vedené v systému ISIR, a tudíž každá osoba je v tabulce `data_persons` jednoznačně identifikována pomocí identifikátoru idOsoby. Pro přiřazení osob ke spisům bude použita samostatná tabulka `link_cases_persons`, která bude

realizovat vazbu M:N, a bude obsahovat spisovou značku řízení, identifikátor osoby a informace o účelu osoby v řízení.

K osobám jsou dále přiřazeny jejich adresy. Je možné, aby u jedné osoby bylo uvedeno více adres, typicky půjde o trvalou adresu, adresu pro doručování atd., a zároveň aby jedna adresa byla přiřazena více osobám (kupříkladu při oddlužení SJM), jedná se tedy o vazbu typu M:N. Pro ukládání adres zavedeme tabulku `data_addresses`. Vzhledem k výše popsanému způsobu zakládání nových osob a jejich přiřazování ke spisům, budeme i každou adresu přiřazovat k jednotlivým osobám u jednotlivých spisů. Každá adresa má identifikátor idAdresy, který bude jednoznačným identifikátorem v tabulce `data_addresses`. Pro přiřazení adresy k osobám bude použita samostatná tabulka `link_persons_addresses`, která bude realizovat vazbu M:N a bude obsahovat spisovou značku řízení, identifikátor osoby a identifikátor adresy. Ostatní atributy vztahující se k osobám mají již na osobu vazbu typu 1:1 a budou tedy uložené v tabulce `data_persons`. Takto navržená datová struktura kopíruje strukturu dat uvedenou v Diagramu 3.2. Dále pro představu uvádíme zjednodušený diagram takto navržené databáze obsahující pouze primární klíče – nejedná se o ER diagram, ten bude uveden později.

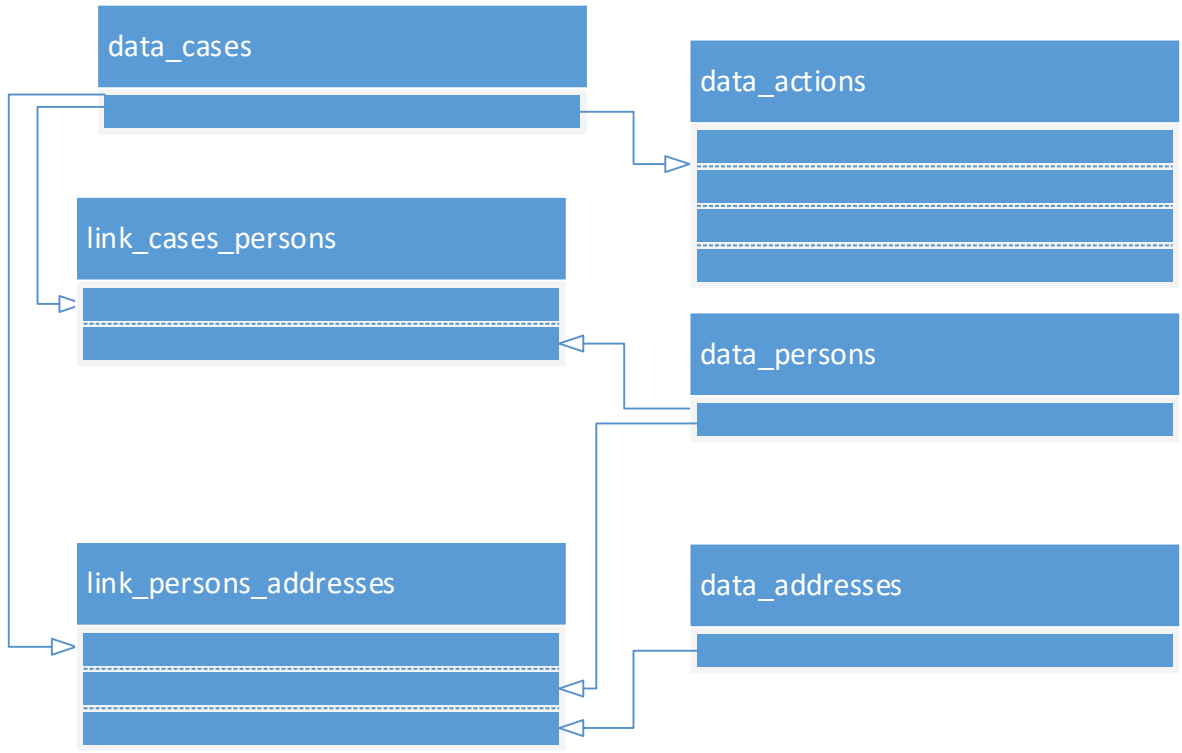


Diagram 4.2: Zjednodušený diagram databáze

4.3 Návrh algoritmu pro zpracování dat

Jak již bylo uvedeno výše, data z webové služby nejsou ve formátu, ve kterém by je bylo možno přímo uložit do databáze, ale musí být zpracována. Při procesu zpracování se rozhodne, o jakou akci se jedná a jak se má taková akce uložit do databáze.

Zpracování musí probíhat tak, že algoritmus nejprve stáhne pomocí webové služby dávku nových akcí. Tyto akce následně po jedné projde a dle jejich povahy zpracuje tak, aby vytvořil změnu výše navržené databáze v souladu se změnou provedenou zpracovávanou akcí. Následně takovouto změnu uloží do databáze. Celé zpracování lze tedy rozdělit do tří relativně samostatných činností, které znázorňuje Diagram 4.3.

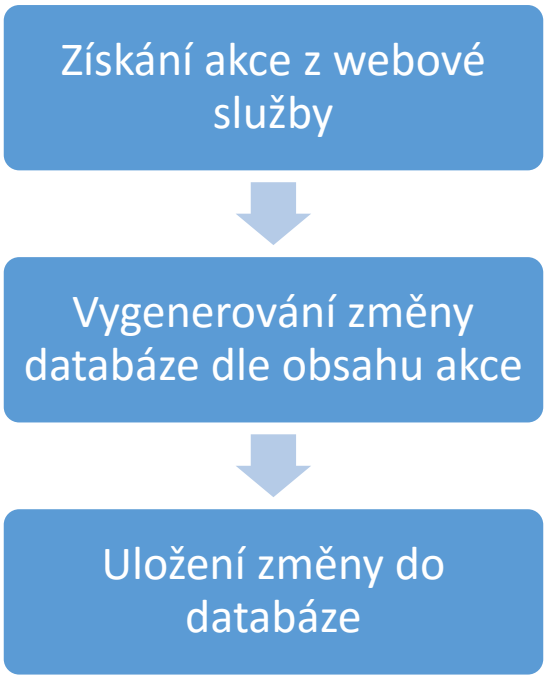


Diagram 4.3: Fáze zpracování dat

Získání dat z webové služby není náročnou činností, je pouze nutné kontaktovat webovou službu, zavolat jednu ze dvou nabízených metod a přečíst vrácený výstup. Stejně tak uložení změny do databáze je algoritmicky triviální akce, neboť tato fáze přebere od předchozí fáze vygenerovaný SQL příkaz a tento předá dále databázovému stroji, který se postará o jeho provedení. Dále se tedy v této kapitole budeme zabývat pouze druhou fází, která ze získané akce od webové služby vygeneruje pozměňovací SQL příkaz.

Od webové služby bude k dispozici vždy datová struktura akce (viz kap. 3.2.1). Z této datové struktury se lze dozvědět, který spis má být pozměněn. Dále je téměř vždy k dispozici položka Poznámka, tedy datový typ udalostComplexType (viz kap. 3.2.2), byť dokumentace k webové službě uvádí, že poznámka bude pouze u omezeného počtu událostí. Z datového typu udalostComplexType lze přečíst minimálně informaci o tom, kdo je původcem předané akce (který subjekt zanesl do systému změnu, která podnítila vygenerování právě zpracovávané akce). U každé akce se musí počítat s tím, že může být první akcí ve spisu, neboť není nikde uvedeno, zda tomu tak opravdu je. To lze zjistit dotazem na databázi, zda již přijatou spisovou značku zná, nebo ne. Pokud nikoliv, je třeba založit spis. Dále z poskytnuté akce nelze dovodit informaci, jestli je akcí zakládající událost, nebo zda je akcí měnící událost. To lze opět zjistit dotazem nad databází, zda událost s jednoznačným identifikátorem zkonstruovaným z atributů akce zná, či nikoliv. Od toho se pak odvíjí, zda se bude událost zakládat nebo aktualizovat. Následně je třeba rozlišovat, zda se jedná o servisní akci, kdy s takovou akcí jsou většinou spojeny speciální úkony na rozdíl od klasického zpracování. Akce, které nejsou servisního charakteru, budou zpracovány tak, že se přečtou veškeré poskytnuté atributy a zapíšou se do tabulky `data_actions`. Z tohoto pravidla existuje několik výjimek. První je existence

atributu datumUstanoveniSpravce. Pokud tento existuje, zapisuje se do tabulky `data_cases`, jelikož se jedná o vlastnost celého spisu, nikoliv jen události v něm. Další výjimky tvoří akce, které značí událost ve spise, ale s touto událostí je spjata nutnost další změny v databázi nad rámec klasického zpracování. Jedná se o následující události:

Kód události	Popis	Zpracování
621	Usnesení o spojení věci k společnému projednávání	Při výskytu této události se krom klasického zpracování ještě označí v tabulce `data_cases` spisová značka věci, se kterou bylo řízení spojeno.
622	Oddělení spojené věci	Při výskytu této události se krom klasického zpracování ještě označí v tabulce `data_cases` spisová značka nového řízení, do kterého byla věc oddělena.
623	Oddělení spojené věci po PM (právní moci)	Při výskytu této události se krom klasického zpracování ještě označí v tabulce `data_cases` spisová značka nového řízení, do kterého byla věc oddělena.
636	Znepřístupnění údajů o dlužníkovi dle § 425 IZ	Při výskytu této události se označí její výskyt tak, aby informace o tomto spise již nebyly dále předávány. Pravděpodobně by mělo dojít k vymazání všech záznamů týkajících se dané věci, nicméně o problémech s tím spojených pojednává kap. 3.2.3.
924	Oddělení spojené věci po právní moci	Duplikát pro akci 622[5]

Tabulka 4.1: Události s rozšířeným zpracováním

Jak bylo uvedeno výše, postup při zpracování servisní akce je odlišný. Počátek zpracování je stejný, nejprve se zjistí, zda událost zakládá spis, nebo zda jej aktualizuje. Společné je i to, že servisní události mohou zanést novou informaci do databáze, nebo aktualizovat již známou informaci. Reakce na jednotlivé akce je pak odvislá od povahy reprezentované události. Proto popis jednotlivých reakcí je pro přehlednost uveden v Tabulce 4.2.

Kód události	Popis	Zpracování
1	Změna osoby	Při výskytu této události dojde k založení nové osoby, popřípadě aktualizaci údajů o

[5] Jedná se pouze o názor autora, který je podložen názvem akce a zkoumáním vrácených XML struktur s kódem události 622 a 924. Avšak potvrzení této domněnky v dokumentaci [21] nenalezneme.

		již známé osobě, v tabulce `data_persons` a `link_cases_persons`. Zdrojem informací pro takové zpracování jsou hodnoty atributů v datové struktuře osobaComplexType.
2	Změna adresy osoby	Při výskytu této události dojde k založení nové adresy a přiřazení této adresy ke konkrétní osobě, nebo k aktualizaci již známé adresy v tabulce `data_addresses` a `link_persons_addresses`. Zdrojem informací pro takové zpracování jsou hodnoty atributů v datové struktuře adresaComplexType.
3	Změna věci	Při výskytu této události aktualizovat příslušnou položku v tabulce `data_cases`.
4	Změna v číselníku událostí	Aktualizovat číselník událostí. Jedná se o seznam číselných kódů a jejich textové interpretace ve stejném duchu, jako jsou první dva sloupce této tabulky. Jelikož u jednotlivých událostí uvedených v tabulce `data_actions` se ukládají pouze číselné kódy, je tento seznam potřebný pro následné zobrazení výstupů ve srozumitelné podobě.
330	Změna XSD dokumentu poznámky	V tomto případě je zapotřebí změnit zdrojový kód klienta, tudíž samotná klientská aplikace na tuto akci nikterak nereaguje.
371	Informační zpráva	Problematikou této akce jsme se zabývali již v kap. 3.2.3. Vzhledem k tomu, že takto poskytovaným informacím běžný uživatel pravděpodobně rozumět nebude a algoritmus též ne, neboť avizované změny pomocí informačních zpráv jsou následně provedeny pomocí postupů strojově srozumitelných (zaslání příslušných akcí atd.), rozhodli jsme se při návrhu algoritmické části tuto akci ignorovat.
372	Změna stavu věci	Při výskytu této události aktualizovat příslušné položky v tabulce `data_cases`, a to v závislosti na změně dle poskytnutých atributů v poznámce u datové struktury udalostComplexType a vecComplexType

405	Změna senátu věci	Při výskytu této události aktualizovat příslušnou položku v tabulce `data_cases`.

Tabulka 4.2: Zpracování servisních událostí

Vývojový diagram k výše popsanému algoritmu zobrazuje Diagram 4.4. Dále je třeba si uvědomit vztah fází zpracování uvedených v Diagramu 4.3 s podobou Diagramu 4.4. Na první pohled by se mohlo zdát, že si tyto odporují, avšak není tomu tak. Fázi *Získání akce z webové služby* reprezentuje vstup *Získej akci z webové služby*, fázi *Vygenerování změny databáze dle obsahu akce* reprezentuje logická struktura vývojového diagramu a částečně též jednotlivé dílčí kroky a fázi *Uložení změny do databáze* reprezentují jednotlivé dílčí kroky.

Diagram 4.4 uvádí zpracování jedné akce. Pro celkové zpracování je zapotřebí pouze doplnit, že uvedený algoritmus pracuje v cyklu, tedy dokud webová služba poskytuje nové akce, algoritmus je zpracovává. S tím souvisí problém časového plánování běhu algoritmu. Na tento problém existují v zásadě dvě odpovědi. Buď lze spustit algoritmus jako nekonečný cyklus, kdy pokud webová služba odpoví, že nová akce v systému neexistuje, algoritmus se na chvíli uspí a zeptá se znovu, nebo lze zvolit způsob, kdy při neexistenci nové akce v systému se algoritmus ukončí a za určitou dobu (např. hodina, den atd.) se znovu spustí. Řešení tohoto problému je úzce svázáno s potřebou aktuálnosti dat v replikované databázi. Pokud se konečný uživatel spokojí s tím, že data v databázi nejsou zcela aktuální, a pracuje s daty několik hodin či dnů starými, pak je výhodnější vydat se cestou ukončení algoritmu, neboť tato cesta méně zatěžuje hardware jak uživatele, tak i samotnou webovou službu.

V rámci dalšího návrhu a implementace se bude výše uvedený problém řešit ukončením algoritmu při neexistenci nové akce. Tento postup byl zvolen jednak z toho důvodu, že ve většině případů postačuje aktualizace databáze vždy pouze jednou za den, a také z toho důvodu, že vhodným nastavením plánovače úloh lze dosáhnout i u ukončovacího se algoritmu stejného výsledku jako u nekonečného.

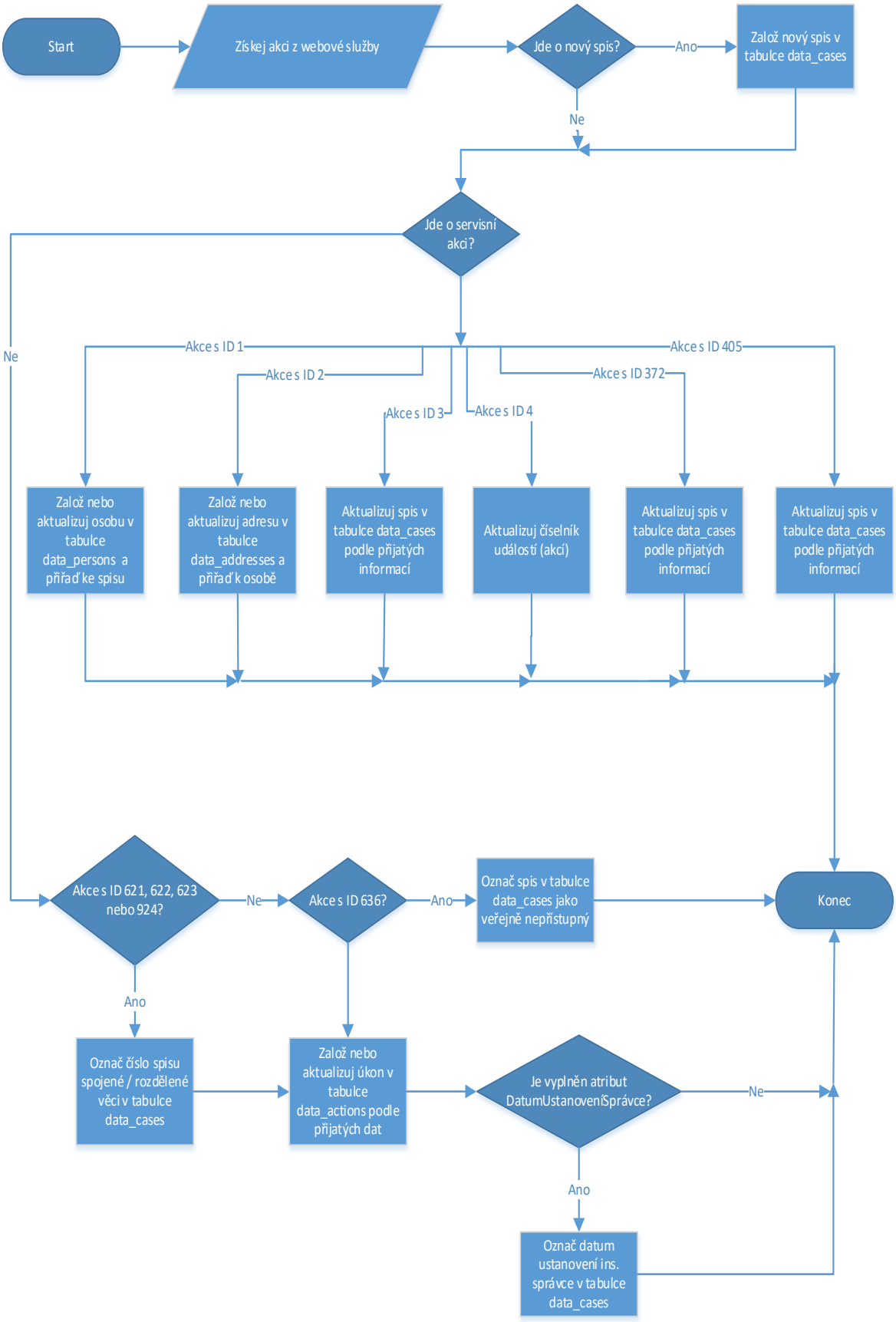


Diagram 4.4: Vývojový diagram algoritmu zpracování

4.4 Vylepšení algoritmu pro zpracování dat

Návrh uvedený v kapitole výše vychází z techniky sekvenčního programování. Jak již bylo uvedeno pomocí Diagramu 4.3, celou problematiku lze rozdělit na tři relativně samostatné fáze. Tyto fáze lze tedy do určité míry provádět samostatně a nezávisle na fázích ostatních. Zároveň je třeba říci, že fází, ve které je třeba výpočetního výkonu, je pouze fáze Vygenerování změny databáze dle obsahu akce. U ostatních dvou fází program pouze předá pokyn k provedení určité činnosti a pak čeká na výsledek tohoto pokynu. U fáze Získání akce z webové služby program pouze předá webové službě požadavek na poskytnutí akce a následně čeká na vrácení výsledku. Toto čekání může být poměrně dlouhé a je závislé především na vytíženosti webové služby a na kvalitě spojení mezi datovou pumpou a webovou službou. Stejně tak u fáze Uložení změny do databáze předá algoritmus pokyn (v tomto případě SQL příkaz) a čeká na jeho provedení. U této činnosti je doba jejího provedení závislá především na vytíženosti databázového stroje, jeho HW konfiguraci a také kvalitě spojení mezi datovou pumpou a databázovým strojem.

Ze všech výše uvedených důvodů se jeví jako vhodné vykonávat jednotlivé fáze paralelně s určitými restrikcemi. Prvně je třeba zachovat posloupnost akcí v tom pořadí, v jakém byly poskytnuty webovou službou. Jelikož akce na sebe navazují a může se stát, že pozdější akce pozměňuje událost modifikovanou akcí předchozí, musí být tyto akce zpracovány ve správném pořadí. Dalším omezením je skutečnost, že webová služba přejímá jako vstupní parametr datum nebo ID, od kterého má vracet 1000 následujících akcí. Dotaz na dalších 1000 akcí můžeme učinit až v době, kdy známe ID nebo datum poslední vrácené akce. Bohužel tomuto požadavku se nevyhneme ani využitím metody webové služby, která přejímá jako parametr ID, neboť číselná řada reprezentující ID akcí není nepřetržitá a neplatí, že zavoláme-li tuto metodu s ID X, bude ID poslední vrácené akce X+1000. Dalším omezením je možnost neaktuálnosti databáze při zpracování akce. Ta může nastat tím, že zpracujeme akci, předáme ji k uložení do databáze a začneme zpracovávat další akci. Pokud tato druhá v pořadí zpracovávaná akce bude pracovat s informacemi, které byly předmětem pozměnění databáze vygenerovaných zpracováním akce první v pořadí, je velice pravděpodobné, že tato ještě nebyla do databáze vložena. Tudíž zpracovávací fáze se nesmí při své činnosti dotazovat databáze na informace, musí pouze vygenerovat příkazy databázi, které provedou správnou změnu, až dojde k jejich zpracování.

4.4.1 Návrh paralelního zpracování

Při dodržení takto stanovených restrikcí lze celý proces paralelizovat tak, že jednotlivé fáze dle Diagramu 4.3 jsou vykonávány samostatně. Pak je třeba vyřešit problém předávání dat mezi jednotlivými fázemi. Jelikož paralelizace bude v tomto případě probíhat pomocí několika vláken, které sdílí fyzicky stejný hardware, lze data předávat pomocí bufferu v operační paměti. V takovém případě buffer mezi první fází a druhou fází bude udržovat jednotlivé akce a buffer mezi druhou a třetí fází vygenerované SQL příkazy. Abychom zajistili posloupnost zpracování tak, aby odpovídala

posloupnosti akcí ve webové službě, musí oba buffery být typu FIFO[6]. Tok dat od počáteční fáze až do uložení do databáze zobrazuje Diagram 4.5.

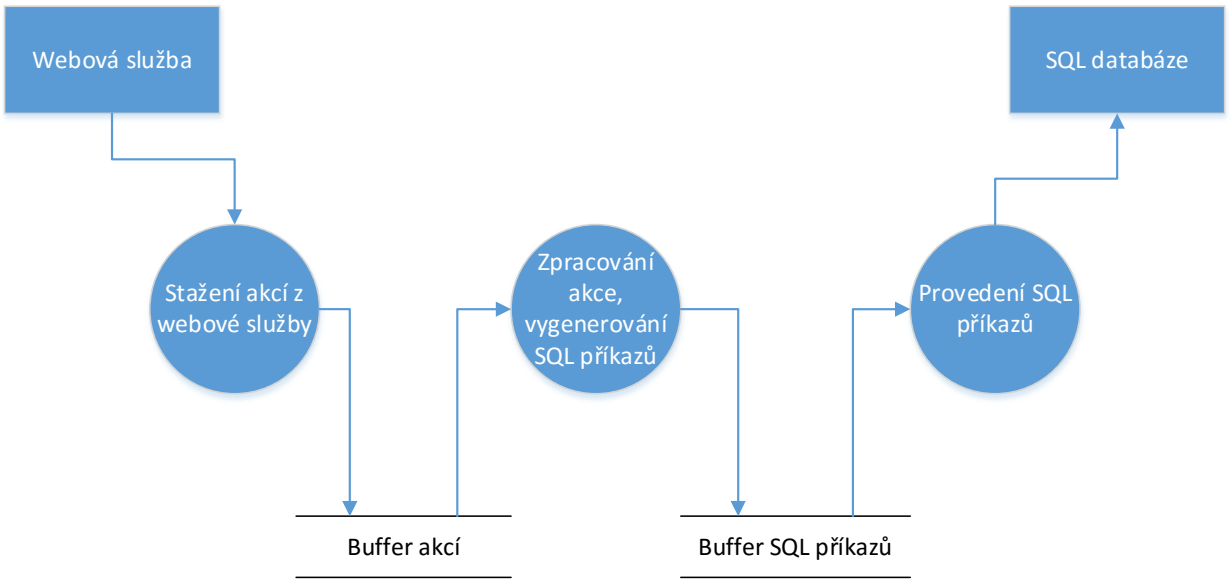


Diagram 4.5: Diagram toku dat při paralelním zpracování

Diagram 4.5 názorně zobrazuje, že v celém procesu paralelního zpracování se nám násobně vyskytuje problém producent – konzument. V učebnicovém popisu tohoto problému producenti produkují nějakou položku a ukládají ji do společného uložiště, konzumenti tuto položku ze společného uložiště odebírají a dále jí zpracovávají [25]. Podle tohoto popisu je pro buffer akcí producentem část stahující akce z webové služby a konzumentem část zpracovávající akce. Pro buffer SQL příkazů je část zpracovávající akce producentem a část provádějící SQL příkazy konzumentem. Aby takto navržený proces mohl správně fungovat, musíme zajistit splnění dvou základních podmínek [25]. Ve chvíli, kdy je do bufferu přidávána nebo odebírána položka, je buffer v nekonzistentním stavu a proces, který takovou akci provádí, musí mít k bufferu exkluzivní přístup. Druhou podmínkou je, že pokud konzument přistoupí k bufferu ve chvíli, kdy je buffer prázdný, zastaví činnost a čeká, dokud producent nepřidá novou položku do bufferu. K těmto dvěma podmínkám musíme přidat ještě omezenou velikost bufferu. Tzn. pokud je buffer plný, producent zastaví svou činnost a počká, dokud se neuvolní místo.

K řízení správného přístupu k bufferům budeme potřebovat dva mutexy[7]. Každý buffer má přiřazený svůj mutex, který hlídá splnění podmínky, že při přidání nebo odebrání položky v bufferu smí takovou akci provádět v jednu chvíli pouze jeden proces. Tím zajistíme konzistentní stav bufferu pro každý proces, který žádá přístup do kritického kódu, který v tomto případě představuje tu část kódu, která

[6] FIFO = first in, first out, je princip ukládání dat tak, že prvně vstupující do fronty je také prvně obsloužený. Zachovává se tím pořadí dat. V praxi mimo svět IT se princip FIFO uplatňuje například při čekání pacientů u lékaře atd.
[7] Mutex zabraňuje tomu, aby současně vstoupila dvě vlákna do kritické sekce nad stejným sdíleným prostředkem. Mutex může nabývat dvou stavů: volný a vlastněný. K tomu udržuje dva druhy informace. Prvním je identifikátor procesu, který mutex drží. Druhým je počet uzamknutí. Počet uzamknutí je vyjádřen číslem, které aktualizuje operace lock (získání mutexu) nebo unlock (uvolnění mutexu). Cílem je zabránění více procesům držet daný mutex. Při vícenásobném uzamknutí musí být i stejný počet odemknutí. [32].

provádí vložení, respektive odebrání, položky z/do bufferu. Dále budeme potřebovat čtyři semafory[8]. Tyto semafory budou určovat obsazenost bufferů a případně pozastaví proces, který nemůže dočasně vykonávat svoji činnost. Takto zkonstruujeme první semafor pro povolení získání dalších akcí z webové služby, druhý semafor pro povolení zpracování získaných akcí, třetí semafor pro povolení uložit SQL příkazy a čtvrtý pro povolení provést vygenerované SQL příkazy. První semafor udává počet volných míst v bufferu akcí a jeho počáteční hodnota se rovná velikosti tohoto bufferu. Při stanovení velikosti bufferu akcí je třeba mít na paměti, že webová služba vrací akce v dávce po 1000 a proto tento buffer by měl mít velikost rovnající se násobkům 1000. Druhý semafor udává počet akcí čekajících na zpracování. Jeho počáteční hodnota je 0. Čtvrtý semafor udává počet volných míst v bufferu SQL příkazů a jeho počáteční hodnota se rovná velikosti bufferu. Čtvrtý semafor udává počet vygenerovaných SQL příkazů a jeho počáteční hodnota je 0.

Takto navržený systém poté funguje následujícím způsobem:

a) Proces zodpovědný za získání akcí z webové služby (stahovací proces) nejprve tyto získá a následně požaduje povolení pro jejich uložení do bufferu akcí (volá nad prvním semaforem operaci down s parametrem 1000). Následně se tento proces pokusí získat vlastnictví mutexu nad bufferem akcí. Jakmile jej získá, uloží stažené akce do bufferu, uvolní jeho mutex a volá operaci up nad druhým semaforem (signalizuje, že v bufferu akcí jsou akce). Jakmile tyto operace dokončí, stáhne novou dávku 1000 akcí a celou činnost opakuje.
b) Proces zodpovědný za zpracování akcí a vygenerování příslušných SQL příkazů (zpracovávací proces) zahájí svoji činnost voláním operace down nad druhým semaforem (tedy ověřuje, že v bufferu existuje alespoň jedna akce ke zpracování). Následně se proces pokusí získat vlastnictví mutexu nad bufferem akcí. Jakmile jej získá, odebere jednu akci z bufferu akcí, uvolní jeho mutex a volá operaci up nad prvním semaforem (signalizuje uvolnění místa v bufferu akcí). Následně získanou akci zpracuje a vygeneruje SQL příkaz. Jakmile se tak stane, volá nad třetím semaforem operaci down (požaduje povolení uložit příkaz do bufferu) a poté se pokusí získat vlastnictví mutexu nad bufferem SQL příkazů. Po jeho získání proces vloží vygenerovaný příkaz do bufferu, uvolní mutex, volá operaci up nad čtvrtým semaforem (signalizuje, že v bufferu je SQL příkaz ke zpracování) a celou činnost opakuje znovu.
c) Proces zodpovědný za provedení SQL příkazů (ukládací proces) zahájí svoji činnost voláním operace down nad čtvrtým semaforem (tj. ověřuje existenci alespoň jednoho SQL příkazu v bufferu). Poté se proces pokusí získat vlastnictví mutexu nad bufferem SQL příkazů. Po jeho získání odebere jeden SQL příkaz z bufferu, uvolní vlastnictví mutexu a volá operaci up nad třetím semaforem (signalizuje uvolnění místa v bufferu SQL příkazů). Takto získaný příkaz provede a celou činnost opakuje znovu.

[8] Semafor je synchronizační primitivum, které obsahuje celočíselný čítač a které chrání přístup do kritické sekce, k čemuž používá dvojici operací *up* a *down*. Operace *down* otestuje stav čítače a v případě, že je nulový, zahájí čekání. Je-li nenulový, je čítač snížen o jedna a vstup do kritické sekce je povolen. Při výstupu z kritické sekce je vyvolána operace up, která odblokuje vstup do kritické sekce pro další (čekající) proces. Čítač je možné si představit jako omezení počtu procesů, které mohou zároveň vstoupit do kritické sekce nebo například jako počítadlo volných prostředků [33].

Je důležité, aby se proces pokusil získat vlastnictví mutexu vždy až poté, co projde semaforem. Pokud by nebyla tato zásada dodržena, mohlo by dojít k deadlocku[9], kdy konzumační proces získá vlastnictví mutexu a bude čekat na vložení nové položky do bufferu, kdežto produkční proces není schopen novou položku vložit, jelikož nemá možnost získat vlastnictví mutexu.

Každý ze čtyř semaforů má v celém procesu svůj nezastupitelný úkol.

a) První semafor hlídá volné místo v bufferu akcí, a pokud již volné místo není, zablokuje možnost vložit další akce do bufferu. Tímto se ošetří problém přetečení bufferu.
b) Druhý semafor hlídá, zda v bufferu akcí jsou akce, a pokud ano, povolí jejich zpracování.
c) Třetí semafor je obdobou prvního pro buffer SQL příkazů. Hlídá jeho obsazenost, a pokud je plný, nedovolí jeho přetečení.
d) Čtvrtý semafor je obdobou druhého. Hlídá, zda v bufferu SQL příkazů jsou příkazy, a pokud ano, povolí jejich provedení.

Takto navržený systém je odolný vůči rozdílné rychlosti každého zúčastněného procesu. Pokud nejpomalejším procesem v systému bude proces zodpovědný za získávání akcí z webové služby, bude proces zpracovávající akce odebírat položky z bufferu akcí rychleji, než je tam proces stahovací bude dodávat. Druhý semafor tak bude po většinu času blokovat zpracovávací proces. Jako důsledek tohoto bude po většinu času prázdný též buffer SQL příkazů a čtvrtý semafor bude po většinu času blokovat ukládací proces.

Pokud nejpomalejším procesem v systému bude zpracovávací proces, první semafor zastaví stahovací proces ve chvíli, kdy v bufferu akcí dojde místo. Tím předejde přetečení tohoto bufferu (popřípadě ke konzumaci veškeré dostupné operační paměti v případě dynamicky se zvětšujícího bufferu). Stahovací proces tak bude po většinu času čekat na možnost uložit nové akce do bufferu. Jako důsledek bude to, že buffer SQL příkazů bude po většinu času prázdný a čtvrtý semafor bude po většinu času blokovat ukládací proces.

Pokud nejpomalejším procesem v systému bude ukládací proces, třetí semafor zastaví zpracovávací proces ve chvíli, kdy v bufferu SQL příkazů dojde volné místo, a předejde tím jeho přetečení (popřípadě ke konzumaci veškeré dostupné operační paměti v případě dynamicky se zvětšujícího bufferu). Důsledkem toho bude, že stahovací proces zaplní buffer akcí a první semafor jej zablokuje, čímž ochrání přetečení bufferu akcí.

První a druhý semafor, stejně tak i třetí a čtvrtý, jsou navzájem inverzní. Součet jejich čítačů se vždy rovná velikosti bufferu akcí, respektive bufferu SQL příkazů. Pokud by implementace semaforu, jako synchronizačního primitiva, byla taková, že lze určit maximální hodnotu čítače a v případě jejího dosažení bude operace *up* blokovat volající proces stejně jako operace *down* v případě nulové hodnoty čítače, pak by bylo možné celý problém řešit pouze za pomoci dvou semaforů.

[9] Deadlock je výraz pro situaci, kdy úspěšné dokončení první akce je podmíněno předchozím dokončením druhé akce, přičemž druhá akce může být dokončena až po dokončení první akce. V počítači se jedná o zablokování procesů (případně vláken) způsobené křížovým čekáním na synchronizačních primitivech. K uváznutí dochází v důsledku chyby programu nebo není uváznutí v programu úmyslně řešeno, protože řešení by bylo příliš náročné. Pokud v takovém případě dojde k uváznutí, je nutný zásah uživatele, který může násilně ukončit jeden z procesů nebo v případě práce s databází může jednu transakci zrušit (příkazem rollback). [34]

4.4.2 Násobná paralelizace

Předchozí část této kapitoly se věnovala paralelizaci návrhu, nicméně předpokládala vždy pouze jeden proces svého druhu, tj. pouze jeden stahovací, zpracovávací a ukládací proces. Tato podkapitola se bude zabývat tím, zda by nešel celý proces zpracování zrychlit ještě tím, že budeme mít více stahovacích, zpracovávacích a ukládacích procesů. Celý problém je značně limitován požadavkem na zpracování akcí v takové posloupnosti, ve které je poskytla webová služba. K tomu je třeba uvést, že dva procesy, které vykonávají totožnou úlohu a které byly spuštěny s určitou prodlevou, nemusí skončit svoji činnost v tom pořadí, ve kterém byly spuštěny. Tato vlastnost v kombinaci s požadavkem na dodržení posloupnosti akcí značně limituje možnost využití násobné paralelizace.

Celá situace bude demonstrována na procesu zodpovědném za získání akcí z webové služby. Tento proces samozřejmě může běžet v několika instancích. První problém, na který narazíme, je skutečnost, že neznáme ID nebo datum poslední získané akce. Pokud by existovaly dva stahovací procesy, které započnou svoji činnost těsně po sobě (prakticky při spuštění programu, kdy spouštěcí metoda vytvoří a zahájí tyto dva procesy), prvnímu lze předat ID poslední zpracované akce. Nicméně v důsledku toho, že ID poslední vrácené akce nemusí být předané ID + 1000 (řada s ID není posloupností s po sobě jdoucími čísly), nelze druhému procesu v pořadí říci, s jakým ID má zahájit činnost. Musí se tedy počkat na dokončení činnosti prvního procesu a převzít si ID jeho posledně získané akce. Tu předá druhému procesu a započne svoji činnost. V tu chvíli se bude neznalost ID řešit u prvého procesu a bude se muset postupovat způsobem uvedeným výše. Z toho plyne, že jeden proces bude vždy čekat na dokončení činnosti svého sourozence, a výsledek bude stejný, jako když je stahovací proces pouze jeden. Pokud by se hypoteticky podařilo problém s neznalostí počátečního ID vyřešit, vyvstane požadavek na dodržení posloupnosti akcí. Jak již bylo uvedeno, dva procesy spuštěné po sobě nemusí skončit ve stejném pořadí, ve kterém byly spuštěny. Například by mohlo dojít k situaci, že první proces stahuje akce od ID 2000 a druhý proces od ID 3000. Pokud druhý proces dokončí svoji činnost dříve, uloží akce s ID 3000 a vyšší do bufferu akcí a notifikuje zpracovávací proces, aby zahájil svou činnost. Tím dojde k porušení posloupnosti. Aby posloupnost byla dodržena, musely by všechny později spuštěné procesy čekat s vložením akcí do bufferu akcí až na dobu, kdy akce vloží dříve spuštěné procesy. Stejný problém se bude řešit i u zpracovávacího procesu. I zde, pokud by zpracovávacích procesů bylo více, musí později spuštěný proces čekat na procesy dříve spuštěné a až poté vložit svou sadu vygenerovaných SQL příkazů do bufferu. V opačném případě dojde k porušení posloupnosti akcí a výsledkem bude nekonzistentní databáze. Ani v případě ukládacího procesu není situace příznivější. I zde musí být zajištěno, že SQL příkazy budou vykonávány v tom pořadí, ve kterém byly vloženy do bufferu. Opět se naráží na problém různé rychlosti stejných procesů, a jelikož zpracování SQL příkazů je závislé na externím subjektu, nelze je dále ohlídat. Jediným zaručeným postupem je počkat na dokončení činnosti dříve spuštěného procesu, nicméně při takové konstrukci bude činný vždy pouze jeden proces a výsledek bude totožný jako v případě, kdy bude ukládací proces pouze jeden.

V předchozím odstavci bylo ukázáno, že násobnou paralelizaci lze poměrně snadno implementovat v případě zpracovávacího procesu. Zde by se dal implementovat řídící mechanismus, např. pomocí číselné hodnoty procesu, který má vložit SQL příkazy do bufferu, a inkrementaci této hodnoty ve chvíli vložení příkazů příslušným procesem. Nicméně z logiky věci zpracovávací proces bude tím nejrychlejším ze všech tří uvažovaných. Stahovací proces je závislý na připojení k webové službě a jejím vytížení, zpracovávací proces na rychlosti procesoru a operační paměti, a ukládací proces na

spojení a vytíženosti SQL stroje. Zpracovávací proces je tedy závislý na nejrychlejších komponentách a tudíž i jeho činnost bude nejrychlejší. Nemá tedy smysl paralelizovat jeho činnost, neboť takové procesy by po většinu času pouze čekaly na činnosti ostatních. Paralelizovat činnost stahovacího procesu je obtížné z důvodu neznalosti počátečních ID akcí, od kterých mají násobné procesy započíst svoji činnost. Stejně tak nelze paralelizovat činnost ukládacího procesu, protože nelze zajistit správnou posloupnost vykonání SQL příkazů.

Paralelizovat daný problém tedy jde pouze na úrovni jednotlivých fází dle Diagramu 4.3 a takto navržený algoritmus bude dále implementován.

5 Realizace návrhu datové pumpy

Tato kapitola se zabývá implementací návrhu z předchozí kapitoly tak, aby vznikl funkční program. Pro implementaci je třeba vybrat technologie, které budou používány, tzn. v tomto případě konkrétní SW systému řízení báze dat (SŘBD) a programovací jazyk, ve kterém bude implementován příslušný algoritmus.

5.1 Výběr vhodných technologií

Základním požadavkem je multiplatformnost. Program a hlavně databáze, která činností programu vznikne, není prvkem, který by byl používán koncovým uživatelem. Jedná se spíše o mezistupeň, kdy další aplikace, které uživatel používá, budou k této databázi přistupovat a zjišťovat z ní informace. Lze si představit klientskou aplikaci, která přímo přistupuje do databáze a zjišťuje informace o subjektech, stejně tak i webové rozhraní, které bude umožňovat v databázi vyhledávat, nebo i strojové rozhraní pomocí např. SOAP protokolu atd. Z výše uvedeného tedy plyne, že databáze bude nejčastěji provozována v serverovém prostředí. Serverové prostředí bude nejčastěji provozováno na operačním systému Linux nebo Windows. V případě Linuxu je třeba řešit i snadné nasazení napříč jednotlivými distribucemi. V úvahu tedy přicházejí

a) Oracle 11g,
b) Firebird,
c) MySQL,
d) PostreSQL či
e) SQLite.

Tento výčet není samozřejmě vyčerpávající, existuje více implementací SŘBD, které by splňovaly nastavené podmínky, avšak uvedeny (a uvažovány) byly ty nejběžnější. Samotná databáze nebude natolik složitým komplexem, aby vyžadovala provozování dedikované instance, ale může být umístěna společně na sdílenou instanci s dalšími daty. Aby to bylo možné, musí být navržena v SŘBD, který je běžně rozšířený a tedy existuje velká pravděpodobnost, že bude již v dané organizaci nasazený. Do užší úvahy tedy spadá databázový stroj Oracle 11g a MySQL. Oracle 11g je proprietární software, ale existuje i verze, která je zdarma (Oracle 11g Express). MySQL je open source projekt a je široce implementovaný, byť existuje též placená verze. Určitou zajímavostí je, že za oběma produkty, tedy Oracle 11g a MySQL, stojí firma Oracle. V této práci zvolíme pro implementaci SŘBD MySQL, a to z důvodu

a) možnosti nasazení na různém spektru operačních systémů (Windows, Linux, Mac OS, FreeBSD),
b) jednoduchosti,
c) široké komunitní podpory,
d) rozšířenosti,
e) a v neposlední řadě zkušeností autora s touto databází.

Výběr konkrétního produktu je však velice subjektivní záležitostí a stejně tak by pro dané účely posloužil kterýkoliv jiný výše zmíněný SŘBD.

Podle podobných kritérií, pomocí kterých byl vybírán SŘBD, byl vybírán též programovací jazyk pro samotnou datovou pumpu. Uvažovány byly jazyky z rodiny .NET (C#), Java a PHP. Rodina .NET je souborem proprietárních jazyků z dílny firmy Microsoft, a je primárně určen pro systém Windows, ale lze jej s určitými omezeními spustit i v prostředí Linux (Mono nebo DotGNU) a Mac OS (Xamarin nebo Mono). Jazyk C# .NET je z uvažovaných programovacích jazyků nejmladší, vznikl v roce 2002, a dle subjektivního názoru autora je i programátorsky nejpřívětivější. Nabízí širokou komunitní podporu a je i hojně využíván. Avšak požadavek multiplatformního nasazení splňuje pouze v případě nasazení ještě dalšího speciálního softwaru, což bylo nakonec vyhodnoceno jako handicap, který tuto technologii vyřadil z dalšího uvažování. Další uvažovanou technologií je PHP, které vzniklo jako skriptovací jazyk na bázi procedurálního programování a objektově orientované programování bylo do tohoto jazyku přidáno až s verzí 5. Proti PHP hovoří též jeho primární určení – primárně je určen pro skriptování internetových stránek, nikoliv pro vývoj aplikací. Nakonec byl tedy vyhodnocen jako nevhodný pro danou problematiku. V případě Javy je plně splněn požadavek multiplatformního nasazení. Java je v dnešní době prakticky nejrozšířenějším programovacím jazykem, samozřejmostí je podpora operačních systémů Windows, Linux a Mac OS, avšak Javu dnes najdeme i v různých specializovaných zařízeních (TV, DVD přehrávače, automobily atd.) a dále nesmíme zapomenout na nejrozšířenější mobilní operační systém Android. Java je velice dobře zdokumentovaným programovacím jazykem se širokou komunitní základnou. Pro vývoj datové pumpy byl tedy nakonec zvolen programovací jazyk Java a použito vývojové prostředí NetBeans IDE.

Jelikož nástroje, vývojová prostředí i samotná syntaxe programovacího jazyku je v angličtině, je i při implementaci dodržován tento jazyk. Stejně tak i v této práci budou používány anglické výrazy dotýkající se popisu algoritmických struktur atd.

5.2 Implementace databáze

Implementace databáze vychází z analýzy provedené v kapitole 4.2. Diagram 5.1 zobrazuje ER diagram implementované databáze. Tento vychází z Diagramu 4.2. Oproti návrhu v předchozí kapitole obsahuje finální databáze ještě výčtové tabulky, pomocí kterých lze k identifikátorům přiřadit lidsky přívětivější vyjádření informace (např. místo KSZPCPM hodnotu Krajský soud v Plzni). Tyto výčtové tabulky nemají nakonfigurované vazby na nosné tabulky, ač by to tak mělo být, a to z toho důvodu, že data do těchto tabulek nejsou vkládána při zpracování událostí – poskytované události ani tyto popisné informace neobsahují. Pokud by v budoucnu došlo k rozšíření některého z číselníků o novou hodnotu, nebylo by možné takovou událost zpracovat, neboť by došlo k chybě v databázi. Po pečlivém zvážení tedy bylo rozhodnuto, že zachování funkčnosti programu má v tomto případě vyšší prioritu než konzistence číselníků s nosnými tabulkami.

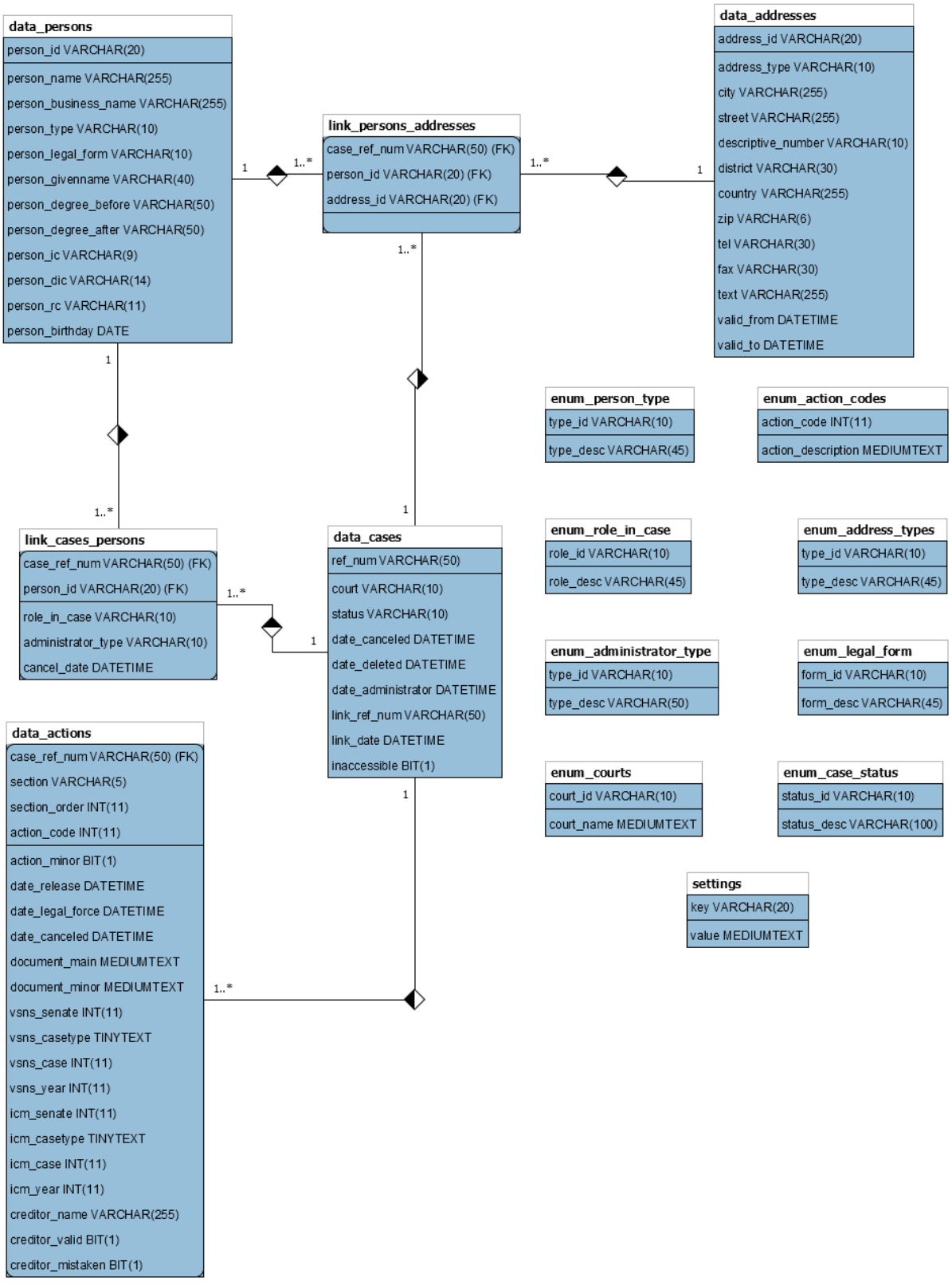


Diagram 5.1: ER diagram databáze

Tabulka 5.1 uvádí seznam výčtových tabulek a krátký popis, jaký druh výčtu obsahují.

Tabulka	Popis
`enum_action_codes`	Obsahuje číselník kódů jednotlivých událostí a jejich slovní vyjádření (např. 5 => Insolvenční návrh)
`enum_address_types`	Obsahuje číselník typů adres a jejich popisné vyjádření (např. SÍDLO FY => Sídlo firmy)
`enum_administrator_type`	Obsahuje číselník funkcí insolvenčního správce a jejich popisného vyjádření (např. PREDB.SPR. => Předběžný správce)
`enum_case_status`	Obsahuje číselník stavů spisu a jejich popisného vyjádření (např. REORGANIZ => Povolena reorganizace)
`enum_courts`	Obsahuje číselník soudů a jejich oficiální názvy (např. KSZPCPM => Krajský soud v Plzni)
`enum_legal_form`	Obsahuje číselník forem právnických osob a jejich popisné vyjádření (např. s.r.o. => společnost s ručením omezeným)
`enum_person_type`	Obsahuje číselník typů osob a jejich popisné vyjádření (např. F => Fyzická)
`enum_role_in_case`	Obsahuje číselník pozic v řízení a jejich popisného vyjádření (např. VĚŘIT-NAVR => Věřitel, který podal návrh na insolvenci)

Tabulka 5.1: Seznam výčtových tabulek

Dále databáze obsahuje tabulku `settings`, která se používá pro ukládání stavu databáze (např. pro uložení posledního zpracovaného ID události).

V tabulce `settings` je mimo jiné uložena i hodnota verze databáze. Verzí databáze se rozumí datum a čas, ke kterému je databáze aktuální.

Krom tabulek obsahuje databáze též několik views, které usnadňují vyhledávání v databázi a zobrazují data z tabulek v interpretační formě, tj. zobrazují se finální významová data a nikoliv identifikátory.

View	Popis
`view_actions`	Zobrazí akce z tabulky `data_actions` s tím, že nahradí identifikátory číselníků jejich významovou hodnotou.
`view_case_detail`	Zobrazí podrobnosti o spisech.
`view_case_status`	Zobrazí spisovou značku a korespondující stav řízení.

Tabulka 5.2: Seznam views v databázi

Databáze dále obsahuje uložené procedury, jejichž seznam je uveden v Tabulce 5.3.

Procedura	Popis
`getCasesForPerson`	Jedná se o proceduru, která vyhledává podle zadaných kritérií osobu a zobrazí všechny známé řízení, ve kterých osoba vystupuje jako dlužník. Procedura přebírá 5 parametrů, a to PName VARCHAR(50), GivenName VARCHAR(50), IC VARCHAR (9), RC VARCHAR(11), Birthday DATE. Může být zadána jakákoliv kombinace těchto parametrů. Pokud uživatel parametr nezná, nebo jej nechce zadat, zadá hodnotu NULL. Procedura skončí chybou, pokud by počet vrácených výsledků byl větší než 50.

Tabulka 5.3: Seznam procedur v databázi

5.3 Implementace algoritmické části

Tato část se zabývá podrobně způsobem, kterým byl implementován navržený algoritmus z kapitoly 4.3 a 4.4 v programovacím jazyce Java. Celý projekt je implementován v JDK 1.8.

5.3.1 Napojení na webovou službu

Jak již bylo několikrát uvedeno, data z insolvenčního rejstříku jsou poskytována pomocí webové služby a pomocí SOAP protokolu. Základem pro tvorbu klienta, komunikujícího s webovou službou, je WSDL definice, což je popis veškerých funkcí, které webová služba nabízí, včetně popisů datových struktur pro vstupy a výstupy těchto funkcí. WSDL vychází z formátu XML. Moderní programovací jazyky pak umí z WSDL definice vygenerovat zdrojový kód. V Javě k tomuto účelu slouží Java API for XML Web Services (JAX-WS).

WSDL definice webové služby insolvenčního rejstříku se nachází na internetové adrese portálu české justice https://isir.justice.cz:8443/isir_public_ws/IsirWsPublicService?wsdl. Vložením této WSDL definice do JAX-WS dojde k vygenerování balíčku `cz.cca.isirpublicws`, který ještě obsahuje dále podpoložku `types` (tj. `cz.cca.isirpublicws.types`). Balíček `cz.cca.isirpublicws` obsahuje interface `IsirWsPublicPortType` a třídu `IsirWsPublicService`.

Balíček `cz.cca.isirpublicws.types` pak obsahuje třídy

- `GetIsirWsPublicDataResponse`,
- `GetIsirWsPublicDatumDataRequest`,
- `GetIsirWsPublicIdDataRequest`,
- `IsirWsPublicData`,
- `IsirWsPublicStatus`,
- `ObjectFactory`,

a výčty

- `KodChybyType`,
- `StavType`.

Z těchto stojí za popis pouze třída `IsirWsPublicData`, která je datovou třídou reprezentující data vrácená z webové služby. Obsahuje, mimo jiné, tyto metody:

Typ	Název metody
`java.lang.Integer`	`getCisloVOddilu()`
`java.util.Date`	`getDatumZalozeniUdalosti()`
`java.util.Date`	`getDatumZverejneniUdalosti()`
`java.lang.String`	`getDokumentUrl()`
`long`	`getId()`
`java.lang.String`	`getOddil()`
`java.lang.String`	`getPopisUdalosti()`
`java.lang.String`	`getPoznamka()`
`java.lang.String`	`getSpisovaZnacka()`
`java.lang.String`	`getTypUdalosti()`

Tabulka 5.4:Tabulka vybraných metod třídy IsirWsPublicData

Jak si jistě pozorný čtenář povšiml, metody a jejich typy se shodují s XML strukturou akce, která je popsána v Tabulce 3.1. Komunikaci s webovou službou pak lze realizovat pomocí následujícího kódu:

```
import cz.cca.isirpublicws.*;
import cz.cca.isirpublicws.types.*;
---
IsirWsPublicService _service = new IsirWsPublicService();
IsirWsPublicPortType _port = _service.getIsirWsPublicPortType();
GetIsirWsPublicIdDataRequest _dreq = new GetIsirWsPublicIdDataRequest();
_dreq.setIdPodnetu(_lastIsirId);
GetIsirWsPublicDataResponse _dresp = _port.getIsirWsPublicPodnetId(_dreq);
List<IsirWsPublicData> _ret = _dresp.getData();
```

Dále stojí za zmínku, že metoda `IsirWsPublicData.getPoznamka()` navrátí textový řetězec, který obsahuje XML dokument poznámky. Tento XML dokument je definovaný schématem XSD. Aktuální verze definice poznámky je 1.10 a XSD schéma lze nalézt na internetové adrese https://isir.justice.cz/isir/help/poznamka_1_10.xsd. Pro další zpracování je vhodné převést XML dokument na nativní datový typ (datovou třídu) jazyka Java, neboť pak se s informacemi poskytnutými v XML dokumentu bude lépe pracovat. Jelikož existuje XSD schéma a XML dokument v atributu poznámka toto schéma respektuje, přičemž na tom nemění nic ani změna verze schémat, neboť historicky docházelo pouze k rozšiřování schémat, tedy schéma vyšší verze obsahuje vše, co schéma verze nižší a přidává vždy něco navíc, lze takový XML dokument převádět na datovou třídu a vice versa. V Javě k tomuto úkolu slouží rozhraní Java Architecture for XML Binding (JAXB).

Pomocí JAXB tak lze z XSD schéma vygenerovat datovou třídu podobně, jako JAX-WS generuje třídy z WSDL definic, a pak XML dokument mapovat vůči takové třídě. Pomocí JAXB byl vygenerován z XSD schéma poznámky verze 1.10 balíček `cz.cca.isirpublicws.note`, který obsahuje třídy

a) `Adapter1`,

b) `Adapter2`,

c) `AdresaComplexType`,

d) `CisUdalostComplexType`,

e) `ObjectFactory`,

f) `OsobaComplexType`,

g) `PoznamkaXSDComplexType`,

h) `UdalostComplexType`,

i) `VecComplexType`.

Vygenerované třídy jsou shodné se strukturami uvedenými v Diagramu 3.3, což je logické, neboť tento diagram popisuje právě strukturu XSD schématu, který byl použit pro vygenerování. Za povšimnutí stojí ještě třídy `Adapter1` a `Adapter2`, které v XSD schématu nefigurují. Tyto dvě třídy se starají o správnou konverzi datových typů mezi XML a Java. V tomto konkrétním případě o datové typy reprezentující datum, jelikož ve výchozím nastavení se XSD typ `date` převádí na Java typ `java.xml.datatype.XMLGregorianCalendar`. Ve zbytku programu ale pracujeme se třídou `java.util.Date` a proto je žádoucí, aby i při převodu XML dokumentu byla data reprezentována touto třídou. O to se starají právě vygenerované třídy `Adapter1` a `Adapter2`, které využívají ke konverzi třídu `cz.jirikubes.isirsync.DateAdapter` (bude popsána dále). Na Java datovou třídu pak lze XML dokument převést následujícím způsobem:

```
import javax.xml.bind.JAXBElement;
import javax.xml.bind.Unmarshaller;
import java.io;
import cz.cca.isirpublicws.note;
---
Unmarshaller _unmarshaller =
JAXBContext.newInstance(cz.cca.isirpublicws.note.ObjectFactory.class);
_unmarshaller.setSchema(null);
ByteArrayInputStream _xmlContentBytes = new
ByteArrayInputStream(_noteInXmlString.getBytes(StandardCharsets.UTF_8));
JAXBElement<UdalostComplexType> _je = (JAXBElement<UdalostComplexType>)
_unmarshaller.unmarshal(_xmlContentBytes);
UdalostComplexType _note = _je.getValue();
_xmlContentBytes.close();
```

Vyobrazený kód je pouze ukázkou celé situace. Pro úplnost by bylo třeba ještě kód uzavřít do `try-catch` bloku a také není dobré při každém převodu inicializovat `JAXBContext`. Tato úloha je náročná na zdroje a její časté opakování znamená výrazné zpomalení běhu programu. Je tedy nezbytné inicializovaný kontext uložit v nějaké globální proměnné a v případě jeho potřeby provést převod tímto již inicializovaným kontextem. Samotný převod pak probíhá rychle.

5.3.2 Napojení na MySQL databázi

Minulá podkapitola se zabývala způsobem, kterým program data získá, následující text se bude zabývat tím, jak program zpracovaná data uloží. Pro napojení programu na databázi slouží Java Database Connectivity (JDBC) API. Toto API zahrnuje též balíček `java.sql`. Tento balíček obsahuje interface `Driver`, které specifikuje rozhraní pro implementaci jednotlivých tříd, které pak specificky komunikují s konkrétním produktem (tj. např. MySQL, MS SQL, Oracle 11g atd.). Pro MySQL je společností Oracle, vývojářem tohoto SŘBD, poskytován MySQL JDBC Driver, který implementuje interface `Driver`, a umožňuje tak komunikaci mezi Javou a MySQL. Spojení za pomoci JDBC se inicializuje URL řetězcem, ve kterém se specifikují veškeré potřebné informace pro bezproblémovou komunikaci. Pro připojení JDBC na MySQL server se používá URL v tomto tvaru `jdbc:mysql://hostname:port/database`.

Jelikož databáze, ke které se program připojuje, není nikterak obsáhlá, není využíván žádný z frameworků, které mapují tabulky z databáze na datové objekty (ORM) v Javě, jako je např. Hibernate. Toto rozhodnutí dále podporuje skutečnost, že s databází není v rámci celého programu nikterak složitě pracováno. Program po startu zjistí z databáze některá nastavení a pak pouze provádí vkládání dat. Tedy ani z hlediska čistoty zdrojového kódu nemá využití ORM frameworku opodstatnění. Pro práci s databází tedy budou využity pouze prostředky, které poskytuje JDBC, tedy zejména

a) `java.sql.Connection`,
b) `java.sql.Statement`,
c) `java.sql.ResultSet`,
d) `java.sql.PreparedStatement`.

Při ukládání dat bude docházet k vkládání velkého množství záznamů během krátké doby. Pro vkládání dat do databáze slouží příkaz `INSERT`, jehož zjednodušená syntaxe je

```
INSERT INTO tbl_name (col1, col2, col3) VALUES (val1, val2, val3).
```

Tímto způsobem lze jedním zavoláním SQL příkazu vložit jeden řádek do jedné tabulky. Ve většině případů je taková syntaxe dostatečná, avšak pokud dochází k hromadnému vkládání dat do jedné tabulky, je volání velkého počtu `INSERT` příkazů během krátké doby neefektivní. Místo toho je vhodnější využít vložení více řádků jedním příkazem, tedy upravit syntaxi do podoby

```
INSERT INTO tbl_name
(col1, col2, col3, ..., colN)
VALUES
(val_11, val_12, val_13, ..., val_1N),
(val_21, val_22, val_23, ..., val_2N),
...,
(val_M1, val_M2, val_M3, ..., val_MN).
```

Tímto způsobem můžeme vložit do *N* sloupců tabulky *M* řádků dat a to vše jedním příkazem. Oproti vkládání hodnot *M* příkazy `INSERT` zde dochází k ušetření systémových prostředků na komunikační režii, tj. na síťové komunikaci mezi programem a MySQL serverem, parsování příkazu ze strany MySQL serveru atd. Při hromadném `INSERT` příkazu se komunikační režie provádí pouze jednou, což znamená efektivnější alokaci systémových prostředků. Množství řádků, které lze takto vložit, není v definici SQL jazyka omezeno, nicméně k omezení dochází při implementaci, kdy konkrétně MySQL skýtá omezení v podobě maximální velikosti paketu, tj. v tomto případě dotazu. Maximální velikost paketu lze nastavit v konfiguraci MySQL serveru pomocí vlastnosti *max_allowed_packet* [26].

Z návrhu provedeného v kapitole 4 vyplývá, že při zpracování akce nevíme, zda událost, kterou akce reprezentuje, již je v databázi nějak zavedena a dochází k aktualizaci, nebo zda jde o novou událost a tedy data mají být vložena. Podobně například nevíme, jestli již spis existuje, nebo nikoliv, neboť spis se do databáze zakládá zasláním první akce patřící ke spisu. Taková konstrukce je poměrně nešťastná ve vztahu k SQL, neboť nové události bychom měli vkládat pomocí příkazů `INSERT`, ale aktualizace událostí bychom měli provádět příkazem `UPDATE`. Jelikož z dat, které nám webová služba poskytne, netušíme, jestli událost v databázi existuje či nikoliv, je třeba tuto informaci zjistit přímo z databáze. MySQL má pro takovou úlohu skvělý nástroj v podobě `ON DUPLICATE KEY UPDATE` části `INSERT` příkazu. Jedná se v podstatě o kombinaci `INSERT` a `UPDATE` příkazu v jednom. Pokud MySQL zjistí, že v tabulce již existuje záznam se stejnými hodnotami v primárním klíči, a u klasického `INSERT` příkazu by tedy vrátil chybu ohledně duplicity primárního klíče, provede `UPDATE` příkaz u požadovaných hodnot.

```
INSERT INTO tbl_name
(col1,col2,col3)
VALUES
(val_1,val_2,val_3)
ON DUPLICATE KEY UPDATE
col3 = val_3
```

Správným nastavením primárních klíčů v databázi a použitím `INSERT` příkazu s `ON DUPLICATE KEY UPDATE` částí vyřešíme snadno problém s nevědomostí ohledně existence záznamu v databázi, aniž bychom se museli databáze dotazovat při generování příkazů, zda se má vygenerovat příkaz `INSERT` nebo `UPDATE`.

V programu se tedy pro vkládání dat do databáze bude využívat příkaz `INSERT` v této syntaxi:

```
INSERT INTO tbl_name
(col1, col2, col3, ..., colN)
VALUES
(val_11, val_12, val_13, ..., val_1N),
(val_21, val_22, val_23, ..., val_2N),
...,
(val_M1, val_M2, val_M3, ..., val_MN)
ON DUPLICATE KEY UPDATE
col2 = IFNULL(VALUES (col2),col2),
col3 = IFNULL(VALUES (col3),col3),
...,
colN = IFNULL(VALUES (colN),colN)
```

U příkazu se ještě věnujme konstrukci `UPDATE` části. Přestože datové typy, které vrátí webová služba, obsahují velké množství atributů, většinou je vyplněno pouze malé množství položek a velká část jich

má nenastavenu hodnotu (`null`). Konstrukce `IFNULL(VALUES (col),col)` otestuje hodnotu poskytnutou na pozici pro vložení do sloupce `col` a pokud je `NULL` tak zachová původní hodnotu sloupce `col` uloženou v databázi.

Dále je při implementaci využíváno třídy `java.sql.PreparedStatement`. Ta umožňuje nevkládat hodnoty přímo do textu SQL příkazu, ale pomocí parametrů přikládat k příkazu. To přináší bezpečnější vykonávání příkazů (zejména ochranu proti SQL injection) a také možnost specifikace datových typů pro jednotlivé parametry.

5.3.3 Třída QueryBuilder pro konstrukci INSERT a UPDATE příkazů

Pro zjednodušení práce s JDBC byla napsána třída `QueryBuilder`. Motivace k vývoji této třídy byla dána několika skutečnostmi

a) `INSERT` příkazy budou generovány v jednom vláknu, ale v jiném vykonány,
b) práce s `PreparedStatement` není v Javě nikterak programátorsky přívětivá, hodnoty se nahrazují znakem `?` a pozice se nečíslují ani nepojmenovávají, tedy programátor musí vložit parametry ve správném pořadí,
c) agregování tvorby SQL příkazu v Java kódu do jednoho místa.

Třída `QueryBuilder` tvoří spolu s dalšími pomocnými třídami balíček `cz.jirikubes.isirsync.SQLUtils`. Třída umí vygenerovat SQL příkaz typu prostého `INSERT`, `INSERT` s `UPDATE ON DUPLICATE` částí a prostý `UPDATE`. Třída poskytuje následující metody:

Typ	Název metody a popis
`void`	**`addReferenceColumn`**`(java.lang.String ColumnName, java.lang.Object ColumnValue)` Vkládá referenční sloupce pro vykonávaný SQL příkaz.
`void`	**`addUpdateColumn`**`(java.lang.String ColumnName, java.lang.Object ColumnValue)` Vkládá sloupce pro vykonávaný příkaz, které v případě `INSERT` budou vloženy, v případě `INSERT_UPDATE_ON_DUPLICATE` budou vloženy nebo aktualizovány a v případě `UPDATE` budou aktualizovány.
`void`	**`addWhereStatement`**`(java.lang.String ColumnName, java.lang.Object ColumnValue, java.lang.String Operator)` Vkládá podmínky pro vykonání příkazu (`WHERE` část SQL příkazu).
`void`	**`buildQuery`**`()` Ze zadaných dat vygeneruje SQL příkaz a vloží do něj data v podobě parametrů.
`void`	**`confirmRow`**`()` Potvrdí zadání parametrů pro aktuální záznam a započne vkládání parametrů pro nový záznam.
`int`	**`executeQuery`**`()` Vykoná SQL příkaz.

`int`	**`getAvailableParametersCount`**`()` Navrací počet parametrů, které lze ještě použít.
`int`	**`getSqlType`**`(java.lang.Object Value)` Získat SQL datový typ ze zadaného Java datového typu
`java.lang.String`	**`getTable`**`()` Navrátí jméno tabulky, se kterou bude SQL příkaz pracovat.
`boolean`	**`isSpaceForRow`**`()` Metoda zjišťuje počet potřebných parametrů pro nový záznam, kdy tento počet je roven množství sloupců v příkazu, a porovnává jej s počtem volných parametrů.
`void`	**`setSqlConnection`**`(java.sql.Connection SqlConnection)` Přejímá přednastavenou proměnou `Connection`, tedy připojení k MySQL serveru, pomocí kterého následně umí vykonat vygenerovaný SQL příkaz.
`void`	**`setTable`**`(java.lang.String Table)` Nastaví jméno tabulky, se kterou bude SQL příkaz pracovat.

Tabulka 5.5: Metody třídy QueryBuilder

Práce se třídou je jednoduchá. Před použitím je třeba vytvořit dvě pole typu `TableColumn`, kdy jedno pole reprezentuje referenční sloupce v tabulce (sloupce tvořící primární klíč) a druhé pole reprezentuje ostatní sloupce v tabulce, které budou vloženy nebo aktualizovány. Poté lze vytvořit instanci třídy. Následně se přidávají pomocí metod `addReferenceColumn` a `addUpdateColumn` hodnoty sloupců k prvnímu záznamu, který bude příkazem vložen. Jakmile jsou přidány veškeré požadované sloupce, potvrdí se řádek metodou `confirmRow` a postup se opakuje pro všechny záznamy, které mají být příkazem vloženy do SQL. Nakonec se vygeneruje a provede požadovaný příkaz pomocí metody `executeQuery`.

Následující fragment kódu, který demonstruje možnosti třídy `QueryBuilder`, vkládá do tabulky `tbl_name`, která má jeden sloupec tvořící primární klíč a čtyři ostatní sloupce, hodnoty z proměnné `_valuesList`. Třída `QueryBuilder` se postará o kontrolu, zda všechny povinné sloupce byly zadány, o vygenerování SQL příkazu, o přiřazení hodnot na správné pozice parametrů a o provedení příkazu na MySQL serveru.

```
QueryBuilder _query;
private void initQuery()
{
    TableColumn[] _refCols = new TableColumn[1];
    _refCols[0] = new TableColumn("ref_col",Types.VARCHAR);
    TableColumn[] _updateCols = new TableColumn[8];
    _updateCols[0] = new TableColumn("update_col_1",Types.VARCHAR);
    _updateCols[1] = new TableColumn("update_col_2",Types.VARCHAR,true);
    _updateCols[2] = new TableColumn("update_col_3",Types.FLOAT,true);
    _updateCols[3] = new TableColumn("update_col_4",Types.INT,true);
    _query = new QueryBuilder
    (QueryType.INSERT_UPDATE_ON_DUPLICATE,_refCols,_updateCols);
    _query.setTable("tbl_name");
}
private void executeQuery()
{
    _query.setSqlConnection(_sqlConnection);
    _query.executeQuery();
    initQuery();
}
private void addRows()
{
    List<GenericDataObject> _valuesList
    initQuery();
    for (GenericDataObject _row : _valuesList)
    {
        if (!_query.isSpaceForRow()) executeQuery();
        _query.addReferenceColumn("ref_col", _row.getRefColValue());
        _query.addUpdateColumn("update_col_1", _row.getUpdateCol1Val());
        _query.addUpdateColumn("update_col_2", _row.getUpdateCol2Val());
        _query.addUpdateColumn("update_col_3", _row.getUpdateCol3Val());
        _query.addUpdateColumn("update_col_4", _row.getUpdateCol4Val());
        _query.confirmRow();
    }
    executeQuery();
}
```

5.3.4 Implementace paralelizace úloh

Nyní, když bylo ukázáno, jak program získává data z webové služby a jak následně zpracovaná data uloží na MySQL server, je čas nastínit způsob, kterým byla implementována paralelizace činností tak, jak byla navržena v kapitole 4.4.1. K tomu slouží třídy v balíčku `cz.jirikubes.isirsync.Threads`.

Třída	Popis
`ActionDownloader`	Třída implementující získávání akcí.
`ActionProcessor`	Třída implementující část algoritmu, která převádí získané akce na SQL příkazy pozměňující databázi do stavu zamýšleného akcí.
`ActionStorer`	Třída implementující část algoritmu, která provádí vygenerované SQL příkazy.
`ThreadKiller`	Vlákno pro násilné ukončení jiného vlákna
`WatchDog`	Třída implementující hlídací algoritmus pro řízení činnosti zpracovávacích vláken a jejich případné obnově.

Tabulka 5.6: Obsah balíčku cz.jirikubes.isirsync.Threads

Všechny třídy v balíčku `cz.jirikubes.isirsync.Threads` rozšiřují třídu `Thread` a jsou navrženy pro paralelní běh jako samostatné vlákno. Návrh dle kapitoly 4.4.1 předpokládá ještě buffer stažených akcí a buffer vygenerovaných SQL příkazů. Ty jsou implementovány následovně

```
LinkedList<IsirWsPublicData> _downloadedActions = new LinkedList<>();
LinkedList<QueryList<QueryBuilder>> _preparedQueries = new LinkedList<>();
```

Celá paralelizace následně probíhá tak, že instance třídy `ActionDownloader` stáhne způsobem popsaným v kapitole 5.3.1 sadu akcí z webové služby a tuto vloží do bufferu `_downloadedActions`. Proces stahování opakuje do té doby, dokud nedojde ke stažení všech akcí, které webová služba k danému času poskytuje. Jakmile dojde ke stažení poslední akce z webové služby, dojde ke vložení zastavovací akce s ID -10 do bufferu `_downloadedActions`.

Z tohoto bufferu si akce postupně vyzvedává instance třídy `ActionProcessor`, z akce vygeneruje SQL příkaz v podobě instance třídy `QueryBuilder`. Třída `ActionProcessor` drží instanci třídy `QueryBuilder` pro každou tabulku v databázi, která se zpracováním aktualizuje. Akce zpracovává, a tedy vkládá záznamy do jednotlivých instancí třídy `QueryBuilder`, tak dlouho, dokud v některé z instancí nedojde počet volných parametrů, popřípadě do doby než se v bufferu `_downloadedActions` objeví zastavovací akce s ID -10. V okamžiku, kdy v jedné z instancí třídy `QueryBuilder` dojde počet volných parametrů, nebo dojde k zastavení zpracování, třída `ActionProcessor` vloží jednotlivé instance třídy `QueryBuilder` do seznamu a to v pořadí

1. instance pro tabulku `data_cases`,
2. instance pro tabulku `data_persons`,
3. instance pro tabulku `data_addresses`,
4. instance pro tabulku `data_actions`,
5. instance pro tabulku `link_cases_persons`,
6. instance pro tabulku `link_persons_addresses`,
7. instance pro tabulku `enum_action_codes`.

Vkládání v tomto pořadí je velice důležité, neboť následně je nutné provádět příkazy ve stejném pořadí, aby nedošlo k chybě v podobě neexistujícího cizího klíče v závislých tabulkách. Jinými slovy musí dojít nejprve k vložení hodnot do referenčních tabulek a až následně k vložení hodnot do

odkazujících tabulek. Seznam příkazů pak `ActionProcessor` vloží do bufferu `_preparedQueries`. Pokud dojde ke zpracování poslední akce, vloží `ActionProcessor` do bufferu neinicializovaný seznam (hodnotu `null`).

Z tohoto bufferu si sadu příkazů vyzvedne instance třídy `ActionStorer`, která po celou dobu svého života drží otevřené spojení k MySQL databázi. Toto spojení přiřadí ke všem instancím třídy `QueryBuilder`, a začne volat metodu `QueryBuilder.executeQuery()` pro každou instanci přesně v tom pořadí, ve kterém byla vložena do seznamu. Pokud všechny příkazy byly provedeny úspěšně, `ActionStorer` provede finální `COMMIT` a změny jsou tím pádem zapsány do databáze. V opačném případě provede `ROLLBACK` a zastaví běh celého programu s chybou. Pokud `ActionStorer` narazí na neinicializovaný seznam (`null`) ukončí činnost a signalizuje úspěšné ukončení programu.

Aby vše fungovalo správně, je nutné řídit činnost vláken pomocí dvou mutexů pro přístup k bufferům

```
Lock _downloadedActionsLock = new ReentrantLock(true);
Lock _preparedQueriesLock = new ReentrantLock(true);
```

a

pomocí čtyř semaforů

```
int _bufferSizeAction = 3000;
int _bufferSizeQuery = 20;
Semaphore _allowDownloadSemaphore = new Semaphore (_bufferSizeAction);
Semaphore _allowProcessingSemaphore = new Semaphore (0);
Semaphore _allowQueryInsertSemaphore = new Semaphore (_bufferSizeQuery);
Semaphore _allowStoreToSQLSemaphore = new Semaphore (0);
```

Pokud chce vlákno `ActionDownloader` vložit stažené akce do bufferu `_downloadedActions`, musí nejprve získat 1000 permits na semaforu `_allowDownloadSemaphore`, poté získat vlastnictví mutexu `_downloadedActionsLock` a až poté může vložit stažené akce do bufferu. Poté provede 1000 release na semaforu `_allowProcessingSemaphore` a uvolní vlastnictví mutexu `_downloadedActionsLock`.

Na to naváže vlákno `ActionProcessor`, které se od svého spuštění pokouší získat 1 permit na semaforu `_allowProcessingSemaphore`. Jakmile se mu to povede, pokusí se získat vlastnictví mutexu `_downloadedActionsLock` a vstoupit tak do kritické sekce, kdy si bude moci z bufferu `_downloadedActions` vyzvednout jednu akci. Po vyzvednutí zavolá 1 release na semaforu `_allowDownloadSemaphore` a uvolní vlastnictví mutexu `_downloadedActionsLock`. Vyzvednutou akci následně zpracuje a vyzvedávání akcí opakuje do té doby, dokud nevyvstane potřeba vložit seznam instancí tříd `QueryBuilder` do bufferu `_preparedQueries`. Při ukládání do bufferu `_preparedQueries` se `ActionProcessor` nejprve pokusí získat 1 permit na semaforu `_allowQueryInsertSemaphore`. Jakmile se mu to podaří, pokusí se získat vlastnictví mutexu `_preparedQueriesLock`. Po jeho získání vstoupí do kritické sekce a vloží seznam do bufferu. Poté zavolá 1 release na semaforu `_allowStoreToSQLSemaphore` a uvolní vlastnictví mutexu `_preparedQueriesLock`.

Na naplnění bufferu `_preparedQueries` naváže instance `ActionStorer` a pokusí se získat 1 permit na semaforu `_allowStoreToSQLSemaphore`. Po jeho získání `ActionStorer` potřebuje získat vlastnictví mutexu `_preparedQueriesLock` a vstoupit tak do kritické sekce. V té vyzvedne jeden

seznam instancí třídy `QueryBuilder`, signalizuje 1 release na semaforu `_allowQueryInsertSemaphore` a uvolní vlastnictví mutexu `_preparedQueriesLock`.

Aby nedocházelo k zbytečnému zastavení vláken `ActionProcessor` a `ActionStorer` ihned při jejich spuštění, je při započetí se stahováním akcí spuštěno pouze vlákno `ActionDownloader`. Ostatní vlákna jsou inicializovaná, ale nejsou spuštěná. Pokaždé, když `ActionDownloader` vkládá nové akce do bufferu `_downloadedActions`, kontroluje, v jakém je vlákno `ActionProcessor` stavu. Pokud zjistí, že `Thread.State.NEW` spustí vlákno `ActionProcessor` voláním metody `start()`. Obdobně se postupuje i v případě, kdy vlákno `ActionProcessor` vkládá seznam instancí `QueryBuilder` do bufferu `_preparedQueries` a spustí tak vlákno `ActionStorer`.

5.3.5 Dohled nad běžícími vlákny

Během vývoje a testování aplikace docházelo poměrně často k situaci, kdy jedno z vláken zamrzlo a tím pádem zastavilo celý stahovací a zpracovávací proces. Nejčastěji se tak dělo u vlákna `ActionDownloader`, které zamrzlo při komunikaci s webovou službou. Bohužel celý portál insolvenčního rejstříku, tedy nejen webová služba, trpí výpadky, které nejsou ojedinělé, a tak bylo nutno na tuto situaci reagovat. V případě komunikace s webovou službou byl nastaven timeout na odeslání a vrácení žádosti webové službě a dále byl implementován kontrolní mechanismus v podobě třídy `WatchDog`, která běží jako další samostatné vlákno. Každé vlákno pravidelně signalizuje svoji činnost. Třída `WatchDog` pak hlídá, zda k takové signalizaci dochází u všech vláken. Tuto kontrolu provádí každých 200 milisekund. Pokud kterékoliv zpracovávací vlákno nesignalizuje činnost v mezidobí kontrolního cyklu, je umístěno do karantény. Tam dostane dalších 60 vteřin k tomu, aby zahlásilo činnost. Pokud se tak stane, opustí karanténu. Pokud nikoliv, znamená to, že čeká na jednom ze semaforů, nebo je zamrzlé. Třída `WatchDog` ověří stavy bufferů, a pokud je vlákno zamrzlé, zahájí resetování všech vláken. To provede tak, že nad všemi vlákny zavolá metodu `interrupt()`, a pokud se vlákna sama neukončí, inicializuje jejich násilné ukončení pomocí třídy `ThreadKiller`. Následně zresetuje oba buffery a vytvoří nové instance tříd `ActionDownloader`, `ActionProcessor` a `ActionStorer` a spustí vlákno `ActionDownloader` od ID akce, která byla jako poslední uložena do databáze. Ukončit i ostatní běžící vlákna je v tomto případě nutné k zachování konzistence dat.

Třídy `ActionDownloader`, `ActionProcessor` a `ActionStorer` jsou implementované tak, aby před každou důležitou činností (např. vložení dat do bufferu) testovaly, zda nebyla volána metoda `interrupt()`. Pokud byla, danou akci neprovedou. Tím je zajištěno, že jakmile se vlákno `WatchDog` jednou pokusí vlákna ukončit, tak i kdyby nebylo úspěšné, tak zpracovávací vlákna již nezasáhnou do znovu inicializovaného prostředí.

Třída `ThreadKiller` implementuje algoritmus, který dohlíží nad ukončením vláken, pokud tato neskončila svou činnost okamžitě po volání metody `interrupt()` z třídy `WatchDog`. Tedy typicky u zamrzlých vláken. Třída `ThreadKiller` se pokusí znovu volat metodu `interrupt()` a vyčká 30 vteřin. Pokud nedojde k ukončení vlákna tak pokus znovu opakuje a pokud ani tak nedojde k ukončení, zavolá metodu `Thread.stop()`. Tato metoda sice byla označena jako zastaralá a není doporučené ji používat, a to hlavně z důvodů velkého rizika nekonzistentnosti dat, pokud by vlákno bylo ukončeno ve chvíli, kdy by provádělo činnost v kritické sekci a bylo držitelem mutexů [27]. Taková situace v tomto konkrétním případě nastat nemůže, neboť prostředí bylo obnovené, a tak případná přítomnost vlákna v kritické sekci nemá vliv na konzistenci dat.

Po implementaci tohoto kontrolního mechanismu došlo ke zvýšení stability programu, kdy se již nestává, že v extrémním případě program zamrzne každých pět minut po spuštění. Navíc výhodou je i skutečnost, že pokud by došlo k deadlocku, hlídací algoritmus obnoví prostředí a tak program deadlocku zbaví.

5.3.6 Architektura programu

Předchozí podkapitoly se věnovaly jednotlivým částem programu, kdy vždy některá část má na starosti určitý úkol. Tato podkapitola se podívá na celý program, tj. na to jak jsou jednotlivé části spojeny do uceleného celku.

Předpokládaný způsob nasazení programu je jako automatický skript spouštěný plánovačem úloh v operačním systému. Z toho důvodu bylo zvoleno, že program bude mít podobu konzolové aplikace. Ovládání programu tedy probíhá pomocí přepínačů z příkazové řádky. Kromě již dříve zmíněných balíčků obsahuje program ještě balíček `cz.jirikubes.isirsync`.

Třída	**Popis**
`DateAdapter`	Třída pro převod proměnných reprezentujících datum z XML na Java typ `java.util.Date`
`ISIR`	Hlavní třída zodpovědná za inicializaci a běh programu.
`Lists`	Statická třída obsahující různé výčty potřebné pro běh programu
`LogFormatter`	Třída zajišťující formátování výstupu pro log.
`LoggingOutputStream`	Třída pro přesměrování `outputstream` do log souboru.
`LogPrintStream`	`PrintStream` do log souboru
`QueryList<T>`	Třída rozšiřující třídu `LinkedList` o další prvky. Používá se k ukládání SQL příkazů do bufferu příkazů.
`RuntimeData`	Statická datová třída poskytující ukládací prostor pro veškerá data, která program potřebuje během své činnosti. Třída obsahuje pouze metody `get` a `set` a veškeré metody jsou psané jako thread safe.

Tabulka 5.7: Obsah balíčku cz.jirikubes.isirsync

Třída `ISIR` je hlavní spouštěcí třídou a obsahuje metodu `public static void main(String[] args)`, tedy spouštěcí metodu volanou při spuštění programu. Metoda zjistí, jaké parametry byly při spuštění programu zadány, a podle toho upraví činnost programu. Pokud podle zadaných parametrů má dojít k získávání a zpracovávání akcí z webové služby, vytvoří metoda instance vláken `ActionDownloader`, `ActionProcessor`, `ActionStorer` a `WatchDog`, spustí první a poslední jmenované a předá řízení programu vláknům tak, jak bylo popsáno v kapitole 5.3.4. Činnosti, kdy ke spuštění vláken nedojde, jsou minoritní a vyřeší je sama metoda `main`. Jedná se zejména o uložení konfigurace, výpis konfigurace na obrazovku atd.

Pro práci s parametry z příkazové řádky byla implementována knihovna Apache Commons CLI [28]. Tato knihovna umožňuje jednoduchou práci s parametry uvnitř programového kódu a zároveň poskytuje uživateli prostředí a způsob práce, na který je běžně zvyklý.

Pro komunikaci mezi běžícími vlákny slouží statická třída `RuntimeData`. Tato třída obsahuje veškeré potřebné informace, které se při běhu programu generují. Namátkou obsahuje informace potřebné k připojení k MySQL serveru, oba buffery, ID posledně stažené akce apod. Celá třída je konstruovaná jako statická a přístup ke všem informacím je ošetřen mutexy, tedy její použití napříč několika běžícími vlákny je bezpečné.

Program umožňuje několik způsobů, jakými zobrazuje svoji činnosti. Primární je výstup na obrazovku, který může být jednořádkový, detailní typu log. Jednořádkový výstup znamená, že program zobrazuje pouze několik základních údajů své činnosti a ty průběžně aktualizuje na jedné řádce po celou dobu své činnosti. Při detailním výstupu zobrazuje program informace na dvanácti řádcích, které opět průběžně aktualizuje po celou dobu své činnosti. Při výstupu typu log vypisuje program záznamy o své činnosti a to buď na obrazovku, nebo do logovacího souboru.

Jednořádkový výstup je realizován pomocí metody `System.out.print()` a průběžné aktualizace zajišťuje znakem `\r` (tedy posunutím kurzoru na začátek řádku) při každém vytisknutí nové řádky. U detailního výstupu, který používá více řádků, se již takto jednoduše pohybovat nelze a je třeba sofistikovanějšího postupu. K úpravě několika řádků v terminálu slouží ANSI escape codes [29]. Použitím těchto speciálních znaků lze dosáhnout požadovaného výsledku. Většina současných terminálů je schopna správně interpretovat ANSI escape codes znaky (např. Linux shell). Velice významnou výjimkou je Win32 console, tedy příkazový řádek v operačním systému Windows. V Javě tento problém řeší knihovna Chalk [30], která rozvíjí knihovnu jansi [31]. Pomocí těchto dvou knihoven lze ANSI escape codes používat i v terminálu Win32 console. Aktualizace informací v těchto dvou výstupech provádí periodicky vlákno `WatchDog` při každém svém kontrolním cyklu.

Výstup typu log je realizován pomocí nástrojů poskytovaných v balíčku `java.util.logging`. Výstup může být zobrazován na obrazovce nebo zapisován do souboru. Vzhledem k velkému množství řádků, které tento výstup vygeneruje, je vhodné použít zápis do souboru. Výstup podporuje několik úrovní detailnosti logování tak, jak jsou implementovány v balíčku `java.util.logging`. Při výstupu typu log se dále používají třídy `LogFormatter`, která formátuje text, který bude do logu přidán, a před každý záznam přidá též datum a čas jeho vygenerování, a třídy `LogOutputStream` a `LogPrintStream`, pomocí kterých lze přesměrovat výchozí `System.out` a `System.err` na `java.util.logging.Logger`. Jelikož se jedná o standardní a doporučený způsob generování log souboru v Javě, získáme touto konstrukcí informace i o činnostech ostatních tříd z Java frameworku (např. z JAX-WS při komunikaci s webovou službou). O inicializaci zápisu záznamu do logu se stará vždy ta část programu, která daný záznam vygenerovala.

UML package diagram a UML class diagramy jsou uvedeny v příloze. Pro rozsáhlost jsou class diagramy uvedeny podle balíčků, ve kterých se třída nachází, a vztahy mezi třídami jsou vyobrazeny pouze do první úrovně. Úplný UML class diagram je pro svou obsáhlost přiložen pouze v elektronické podobě.

5.4 Vývojové prostředí

Konfigurace a tvorba databáze probíhala v programu MySQL Workbench, který je poskytován společností Oracle a pomocí něhož lze MySQL server spravovat. Jako elektronická příloha je k této práci přiložen soubor 2_db_model.mwb, který reprezentuje schéma databáze. Dále je přiložen soubor 3_create_db.sql, pomocí kterého lze vytvořit databázi tak, aby s ní mohl program začít okamžitě pracovat. Skript vytvoří prázdnou databázi, do které je následně potřeba nasynchronizovat data pomocí programu. Skript vytvoří, a v případě, že existuje, tak nejprve odstraní, databázi s názvem isir. Pokud chce uživatel vytvořit databázi s jiným jménem, je zapotřebí skript upravit. Na řádcích 1 až 11 se nachází tyto příkazy:

```
-- -----------------------------------------------------
-- Schema isir
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `isir`;

-- -----------------------------------------------------
-- Schema isir
-- -----------------------------------------------------
CREATE DATABASE IF NOT EXISTS `isir` DEFAULT CHARACTER SET utf8 COLLATE utf8_czech_ci;
USE `isir`;
```

Pokud chceme vytvořit databázi s jiným názvem, stačí v těchto třech případech nahradit název databáze, tj. text `` `isir` ``, jakýmkoliv jiným požadovaným názvem.

Aby uživatel nemusel provádět synchronizaci databáze od počátku, tvoří elektronickou přílohu též skript 4_data_import.sql. Tento skript naimportuje data do databáze vytvořené skriptem 3_create_db.sql . Skript 4_data_import.sql byl vygenerován pomocí funkce Data Export nástroje MySQL Workbench. Tato funkce je grafický wrapper nad konzolovou aplikací mysqldump. Pro import dat lze tedy použít opět MySQL Workbench a funkci Data Import, popřípadě použít přímo konzolovou aplikaci mysql, kdy funkce Data Import je opět grafický wrapper nad takovou aplikací.

Pro vývoj aplikace bylo použito IDE NetBeans 8.0.2. Projekt datové pumpy je zabalen a zkomprimován formátem ZIP a do prostředí NetBeans jej lze naimportovat pomocí menu File -> Import project -> From ZIP. Uživateli se po importu otevře projekt se všemi potřebnými soubory, tvořící knihovny a zdrojové kódy, pro zkompilování aplikace. Ihned po importu budou třídy, obsahující odkazy na automaticky generovaný kód (tj. kód z JAX-WS a JAX-B), označeny červeným křížkem, který značí chybu v kódu. To je dáno tím, že projekt byl před exportem vyčištěn a neobsahuje žádné binární ani další automaticky generované soubory. Problém lze jednoduše vyřešit zkompilováním projektu (klávesa F11), kdy při kompilaci dojde též k vygenerování potřebných souborů. Konzole prostředí NetBeans má bohužel problém se zobrazováním výstupu programu. ANSI escape codes neumí zobrazit vůbec, tedy výstup typu VERBOSE vygeneruje v konzoli řadu nesrozumitelných znaků, u výstupu INLINE je výstup poněkud srozumitelnější, ale NetBeans konzole trpí velkým zpožděním a navozuje dojem, že program zamrzl. Pokud tedy není potřeba program krokovat (debug), je vhodnější jej spustit z konzole operačního systému, kde již výstupy fungují správně. Pokud je přeci jen zapotřebí spustit program v prostředí NetBeans, je vhodné nastavit pomocí spouštěcích konfigurací výstup programu typu INLINE, tj. kliknout pravým tlačítkem na projekt, menu Properties a v otevřeném dialogovém okně vybrat možnost Run a do políčka Arguments zapsat –o INLINE. Stejným způsobem lze předávat programu i ostatní parametry příkazové řádky, které budou popsány v následující kapitole.

5.5 Používání programu, uživatelská dokumentace

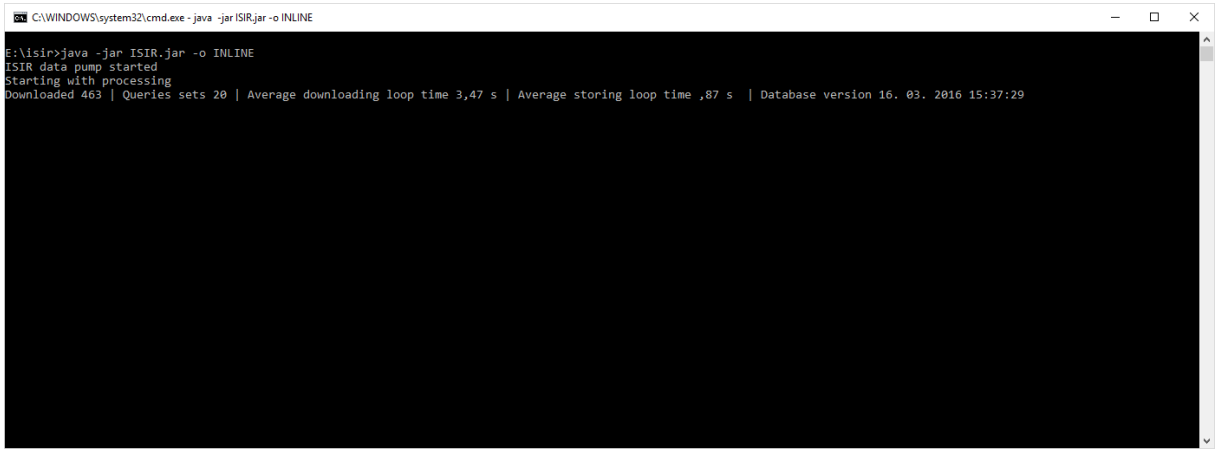
Uživatel může program ovládat pomocí parametrů z příkazové řádky. Program rozumí a reaguje na následující parametry

Parametr	**Popis**
-C <Property>	Vymaže hodnotu uloženého parametru specifikovaného v argumentu.
-clearsettings	Vymaže veškerou uloženou konfiguraci.
-database <MySqlDatabase>	Jméno databáze na MySQL serveru, do které budou ukládána data.
-h,--help	Zobrazí nápovědu – výpis možných příkazů.
-host <MySqlHost>	IP adresa, FQDN nebo název serveru, na kterém je zprovozněn MySQL server
-logfile <FileName>	Název souboru, do kterého má být zapisován výstup v případě LOG výstupu.
-logleveldetail <LivelDetail>	Úroveň logování, ve výchozím nastavení je INFO
-o,--output <OutputType>	Způsob výstupu. Možné hodnoty jsou INLINE, VERBOSE a LOG. Ve výchozím nastavení je výstup typu VERBOSE.
-password <MySqlPassword>	Heslo pro přístup k MySQL serveru.
-port <MySqlPort>	Port pro připojení k MySQL serveru. Ve výchozím nastavení se použije hodnota 3306.
-rm	Odstraní z databáze data patřící spisům, které byly znepřístupněny.
-s,--save	Uloží všechny poskytnuté parametry do konfiguračního souboru. Při dalším spuštění se již parametry zadávat nemusí a program se spustí s požadovanou konfigurací.
-showsettings	Vypíše konfiguraci, se kterou by byl program spuštěn, pokud by nebyl poskytnut tento parametr.
-user <MySqlUser>	Uživatel, který má oprávnění k MySQL serveru.

Tabulka 5.8: Seznam parametrů pro konfiguraci programu

Parametry, u kterých je uvedeno za názvem parametru <arg>, očekávají zadání hodnoty, a pokud k němu nedojde, skončí program s chybou.

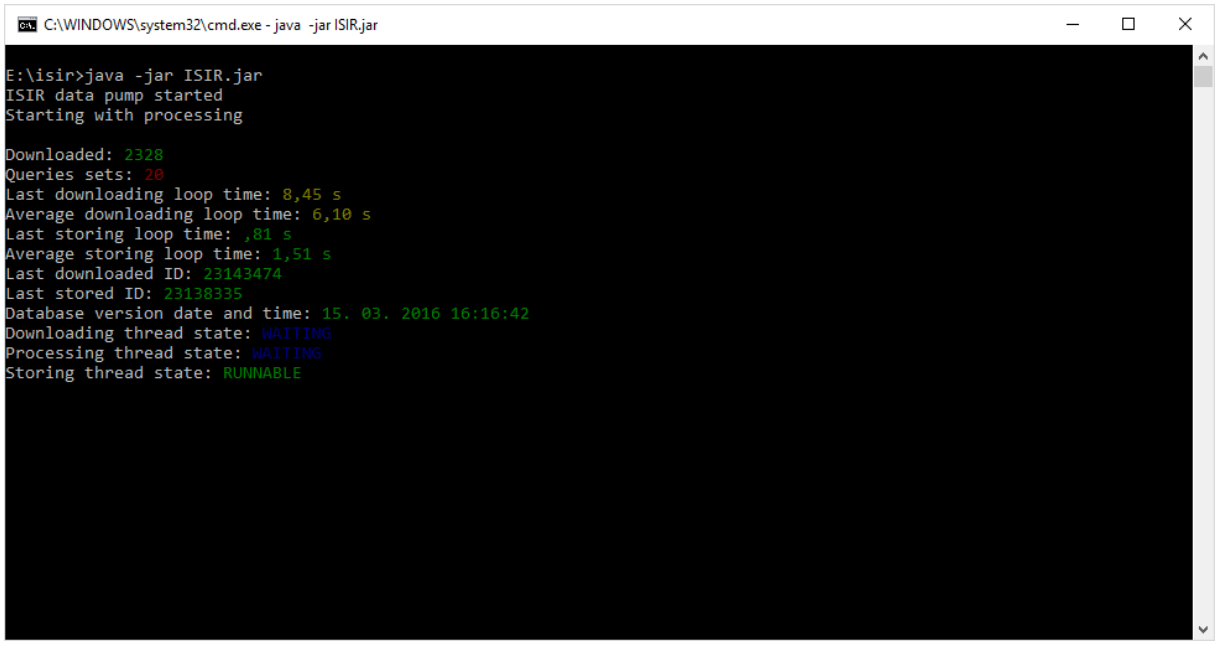
Obrázek 5.1 uvádí ukázku výstupu, pokud je výstup nastaven na hodnotu INLINE.



Obrázek 5.1: Výstup programu v režimu INLINE

V režimu INLINE zobrazuje program pouze základní informace. Uvádí počet položek v bufferu stažených akcí, počet položek v bufferu SQL příkazů, průměrnou dobu stažení 1000 akcí z webové služby, průměrnou dobu uložení jedné sady SQL příkazů na SQL server a verzi databáze.

Obrázek 5.2 zobrazuje výstup typu VERBOSE. Krom výše uvedeného se ještě zobrazuje doba posledního stažení 1000 akcí a doba posledního uložení sady SQL příkazů na SQL server, ID posledně stažené a posledně uložené akce a stav zpracovávacích vláken. Jelikož výstup VERBOSE využívá ANSI escape codes, můžou být hodnoty v tomto výstupu barevně rozlišovány.



Obrázek 5.2: Výstup programu v režimu VERBOSE

Dále je uvedena část log souboru v případě, kdy výstup je nastaven na režim LOG a úroveň logování je nastavena na INFO.

"17. 03. 2016 15:30:02";"ISIR";"INFO";"ISIR data pump started"

"17. 03. 2016 15:30:04";"ISIR";"INFO";"Starting with processing"

"17. 03. 2016 15:30:04";"WatchDog";"INFO";"Thread WatchDog started"

"17. 03. 2016 15:30:04";"ActionDownloader";"INFO";"Thread IsirDownloadingThread0 started"

"17. 03. 2016 15:30:09";"ActionProcessor";"INFO";"Thread IsirProcessingThread0 started"

"17. 03. 2016 15:30:09";"ActionStorer";"INFO";"Thread IsirStoringThread0 started"

Jak je ze zobrazené ukázky vidět, logovací soubory jsou formátovány do podoby CSV souboru a tak mohou být dále jednoduše strojově zpracovávány. Sloupce jsou řazeny v pořadí datum a čas vzniku události, původce události (název Java třídy), vážnost (severity) daného záznamu a samotná zpráva.

5.6 Ukázka dotazů do databáze pro zjišťování vybraných informací o dlužnících

Běžný uživatel bude nejčastěji vyhledávat v databázi dlužníka podle jména a příjmení, popřípadě podle obchodní firmy, IČ nebo data narození. Ke zjednodušení tohoto vyhledávání je v databázi procedura `getCasesForPerson`. Vyhledání podle jména a příjmení fyzické osoby lze provést následujícím způsobem

```
CALL isir.getCasesForPerson('Kubeš', 'Jiří', NULL, NULL, NULL);
```

Tento dotaz vrátí výsledek se sloupci

a) court,
b) reference_number,
c) case_status,
d) person_fullname,
e) person_ic,
f) person_rc,
g) person_birthday,
h) person_address.

Pomocí takto vráceného výsledku lze identifikovat spisovou značku insolvenčního řízení a ověřit si, že nalezený dlužník je opravdu tou osobou, kterou uživatel vyhledává. Při znalosti spisové značky lze následně získat informace o samotném spisu pomocí příkazu

```
SELECT * FROM isir.view_case_detail WHERE ref_num = 'INS 1565/2013';
```

s výsledkovou datovou strukturou

a) court_name,
b) ref_num,
c) status_desc,
d) administrator,
e) administrator_address,
f) date_administrator,
g) link_ref_num,
h) link_date,
i) date_canceled,

j) date_deleted.

Další častou činností bude požadavek na zjištění veškerých úkonů patřících do konkrétního spisu. Takovou informaci lze zjistit SQL příkazem

```
SELECT * FROM isir.view_actions WHERE case_ref_num = 'INS 1564/2013';
```

Výše uvedený příkaz vrátí datovou strukturu se sloupci

a) case_ref_num,
b) section,
c) section_order,
d) action_description,
e) action_minor,
f) date_release,
g) date_legal_force,
h) date_canceled,
i) document_main,
j) document_minor,
k) vsns_senate,
l) vsns_casetype,
m) vsns_case,
n) vsns_year,
o) icm_senate,
p) icm_casetype,
q) icm_case,
r) icm_year,
s) creditor_name,
t) creditor_valid,
u) creditor_mistaken.

V případě, že uživatel již zná spisovou značku insolvenčního řízení a pouze chce ověřit jeho stav, lze tak učinit následujícím příkazem

```
SELECT * FROM isir.view_case_status WHERE ref_num = 'INS 1564/2013';
```

Tento dotaz vrátí poměrně stručnou datovou strukturu se sloupci

a) ref_num,
b) status.

Ve většině případů pomocí výše uvedených příkazů získá uživatel veškeré informace, které potřebuje.

6 Závěr

V této práci se povedlo seznámit čtenáře s problematikou e-governmentu, seznámit jej s pojmem open data a představit několik stěžejních projektů, kterými Česká republika naplňuje závazek poskytování služeb e-governmentu.

Dále byla hlouběji rozebrána jedna z takto poskytovaných služeb, Insolvenční rejstřík, a to jak z pohledu získávání informací uživatelem, tak i z pohledu strojového zpracování. Byly analyzovány webové služby pro strojové získávání dat z insolvenčního rejstříku a na základě této analýzy byl navržen a implementován počítačový program, který data z webové služby získává, zpracovává a ukládá pro další potřebu.

Navržený a realizovaný program má jistě svůj potenciál, a to hlavně jako middleware pro komplexní informační systémy, které potřebují získávat a interpretovat data z insolvenčního rejstříku. Vývojáři takových informačních systémů pak nemusejí implementovat celý proces rekonstrukce databáze insolvenčního rejstříku, ale mohou číst přímo z databáze, kterou jim datová pumpa připraví.

Samotný program by se dal ještě rozšířit o další výstupy pro uživatele nebo například o možnosti různých způsobů zpracování informací a různých způsobů jejich ukládání. Lze si představit možnost různých zpracovávacích vláken a ukládacích vláken, které by spojovalo stejné rozhraní (interface) a pomocí přepínačů by uživatel zvolil, jakým způsobem chce akce zpracovávat a ukládat. Takto by program mohl být ještě více univerzálním a umožnit ukládání informací na jiné SŘBD, než jen MySQL, popřípadě zvolit i jiný způsob ukládání dat (např. NoSQL databáze, nebo XML struktury). Další možností vývoje celého projektu je poskytnout nástroje, jak data z databáze číst. Nabízí se webový portál, propracovanější systém views v databázi, nebo například webová služba, který by poskytovala informace z databáze tak, jak by ji uživatel očekával, a nikoliv v podobě v podstatě transakčního logu databáze.

Tím se dostáváme k samotné webové službě ISIR_WS, která zaostává za potencionálem, který by mohla naplnit. Výtky k této službě jsou shrnuty v podkapitole 3.4, za ty nejvýznamnější jmenujme samotnou logiku webové služby, která navozuje dojem cíleného ztěžování jejího používání, nelogičnost v její implementaci a v neposlední řadě nepřehlednou a nedostatečnou dokumentaci.

7 Reference

[1] MATES, Pavel a Vladimír SMEJKAL. *E-government v České republice: právní a technologické aspekty*. 2., podstatně přeprac. a rozš. vyd., V nakl. Leges vyd. 1. Praha: Leges, 2012, 464 s.

[2] EUROPEAN COMMISSION, . The European eGovernment Action Plan 2011-2015. *EUR-Lex: Access to European Union law* [online]. b.r. [cit. 2015-06-27]. Dostupné z: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2010:0743:FIN:EN:PDF

[3] *OtevřenáData.cz* [online]. 2014 [cit. 2015-06-27]. Dostupné z: http://www.otevrenadata.cz/

[4] *CKAN - The open source data portal software* [online]. b.r. [cit. 2015-06-28]. Dostupné z: http://ckan.org/

[5] CKAN. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2015 [cit. 2015-06-28]. Dostupné z: https://en.wikipedia.org/wiki/CKAN

[6] MINISTERSTVO FINANCÍ ČR. *Administrativní registr ekonomických subjektů* [online]. 2013 [cit. 2015-06-28]. Dostupné z: http://wwwinfo.mfcr.cz/ares

[7] ARES - jmenné prostory. *Administrativní registr ekonomických subjektů* [online]. b.r. [cit. 2016-06-28]. Dostupné z: http://wwwinfo.mfcr.cz/ares/xml_doc/schemas/index.html

[8] *ČESKO. Zákon č. 182 ze dne 30. března 2006, o úpadku a způsobech jeho řešení (insolvenční zákon)*. 2006, (182).

[9] *Nahlížení do katastru nemovitostí* [online]. 2004-2015 [cit. 2015-06-28]. Dostupné z: http://nahlizenidokn.cuzk.cz

[10] Účtování výstupů z katastru nemovitostí poskytovaných dálkovým přístupem a webovými službami dálkového přístupu. *ČÚZK* [online]. b.r. [cit. 2015-06-28]. Dostupné z: http://www.cuzk.cz/Katastr-nemovitosti/Poskytovani-udaju-z-KN/Dalkovy-pristup/Uctovani-vystupu-z-KN-poskytovanych-DP-A-WSDP.aspx

[11] *Český úřad zeměměřičský a katastrální* [online]. 2013 [cit. 2015-06-28]. Dostupné z: http://www.cuzk.cz

[12] *Webové mapové služby pro katastrální mapy (WMS KN)* [online]. b.r. [cit. 2015-06-28]. Dostupné z: http://www.cuzk.cz/Katastr-nemovitosti/Poskytovani-udaju-z-KN/Webove-mapove-sluzby-pro-katastralni-mapy-(WMS-KN).aspx

[13] *Veřejný dálkový přístup k datům registru územní identifikace, adres a nemovitostí* [online]. 2015 [cit. 2015-06-28]. Dostupné z: http://vdp.cuzk.cz

[14] CzechPOINT@home. *Portál veřejné správy* [online]. 2015 [cit. 2015-06-28]. Dostupné z: http://portal.gov.cz/portal/cph/index.html

[15] *Justice.cz* [online]. 2015 [cit. 2015-06-28]. Dostupné z: http://portal.justice.cz

[16] *Datové schránky* [online]. 2015 [cit. 2015-06-29]. Dostupné z: https://www.datoveschranky.info

[17] Provozní řád informačního systému datových schránek (ISDS). *Datové schránky* [online]. 2015 [cit. 2015-06-29]. Dostupné z: https://www.datoveschranky.info/documents/1744842/2717964/provozni_rad_isds.pdf

[18] *ISDS – Webové služby rozhraní ISDS pro manipulaci s datovými zprávami* [online po zaregistrování]. 2014. Dostupné také z: http://team.smartadministration.cz

[19] NYGRÝN, Pavel. Datové schránky: Skvělý nápad s mizernou realizací. In: *Živě.cz* [online]. b.r. [cit. 2015-06-28]. Dostupné z: http://www.zive.cz/clanky/datove-schranky-skvely-napad-s-mizernou-realizaci/sc-3-a-150533

[20] Datová schránka. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2015-06-29]. Dostupné z: https://cs.wikipedia.org/wiki/Datová_schránka

[21] MINISTERSTVO SPRAVEDLNOSTI ČR, a CCA GROUP. *Webová služba aplikace ISIR: Popis způsobu používání webové služby* [online]. 2016 [cit. 2016-03-16]. Dostupné z: https://isir.justice.cz/isir/help/Popis_WS_1_v2_3.pdf

[22] MINISTERSTVO SPRAVEDLNOSTI ČR, a CCA GROUP. *Číselník událostí insolvenčního rejstříku* [online]. 2015 [cit. 2015-06-23]. Dostupné z: https://isir.justice.cz/isir/help/Cis_udalosti.xls

[23] MINISTERSTVO SPRAVEDLNOSTI ČR, a CCA GROUP. *Webová služba ISIR_CUZK_WS: Popis způsobu používání webové služby* [online]. b.r. [cit. 2015-06-23]. Dostupné z: https://isir.justice.cz/isir/help/Popis_WS_2_v1_3.pdf

[24] MINISTERSTVO SPRAVEDLNOSTI ČR, . Automatické sledování insolvenčního rejstříku. MINISTERSTVO SPRAVEDLNOSTI ČR. *Insolvenční rejstřík* [online]. b.r. [cit. 2015-06-23]. Dostupné z: https://isir.justice.cz/isir/common/stat.do?kodStranky=SLEDOVANIWS

[25] DOWNEY, Allen *The Little Book of Semaphores: Second Edition* [online]. 2.1.5. Needham, MA, 2005 [cit. 2016-02-13]. Dostupné z: http://www.greenteapress.com/semaphores/downey08semaphores.pdf

[26] Server System Variables. *MySQL Documentation* [online]. Oracle Corporation, b.r. [cit. 2016-03-16]. Dostupné z: http://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html#sysvar_max_allowed_packet

[27] Java Thread Primitive Deprecation. *Java SE Documentation* [online]. Oracle Corporation, b.r. [cit. 2016-03-16]. Dostupné z: https://docs.oracle.com/javase/8/docs/technotes/guides/concurrency/threadPrimitiveDeprecation.html

[28] *Apache Commons CLI library* [online]. b.r. [cit. 2016-03-17]. Dostupné z: https://commons.apache.org/proper/commons-cli/

[29] ANSI escape code. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2016-03-17]. Dostupné z: https://en.wikipedia.org/wiki/ANSI_escape_code

[30] *GitHub - tomas-langer/chalk: Terminal string styling done right (in java)* [online]. b.r. [cit. 2016-03-17]. Dostupné z: https://github.com/tomas-langer/chalk

[31] *GitHub - fusesource/jansi* [online]. b.r. [cit. 2016-03-17]. Dostupné z: https://github.com/fusesource/jansi

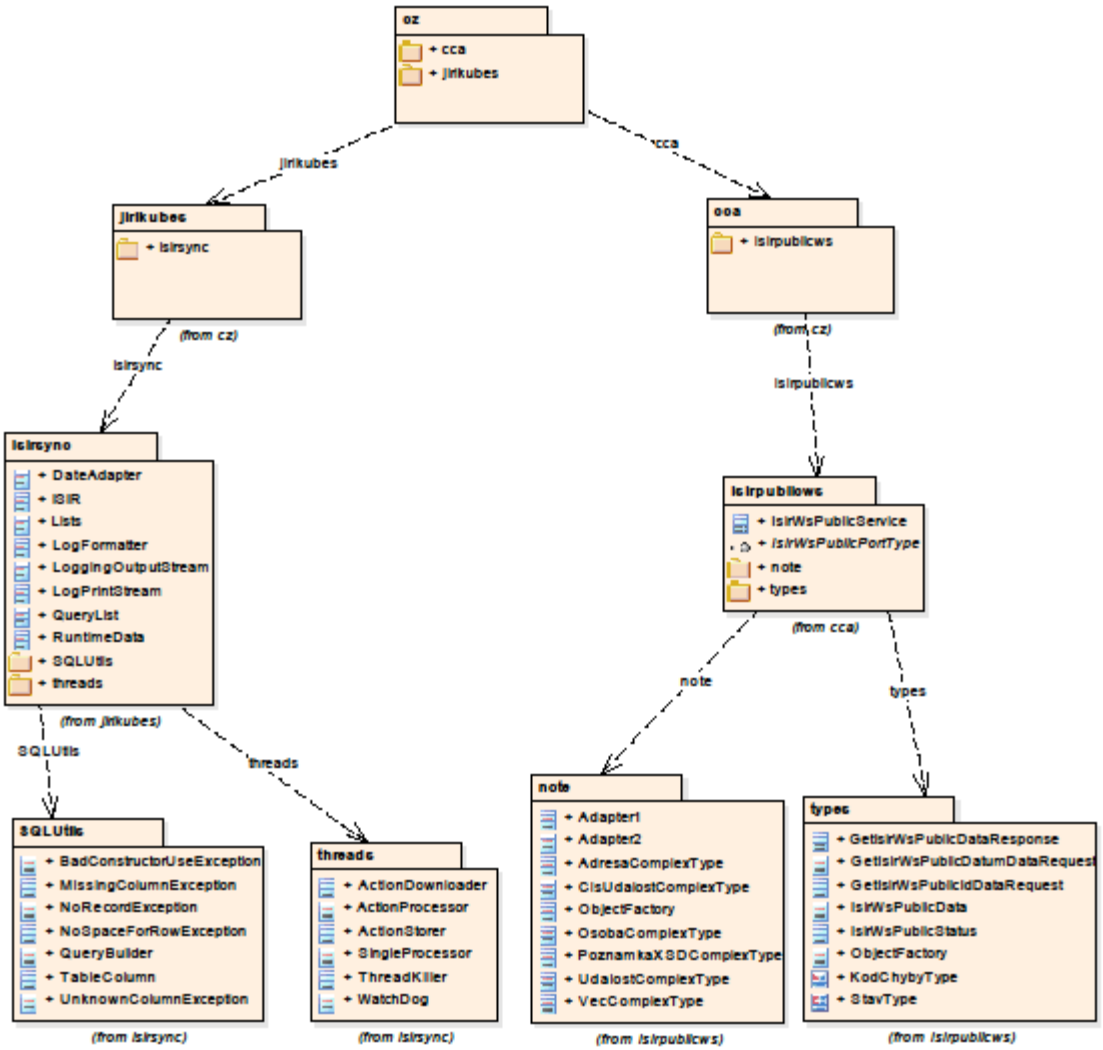
[32] Vzájemné vyloučení. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2016-02-13]. Dostupné z: https://cs.wikipedia.org/wiki/Vzájemné_vyloučení

[33] Semafor (synchronizace). *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2016-02-13]. Dostupné z: https://cs.wikipedia.org/wiki/Semafor_(synchronizace)

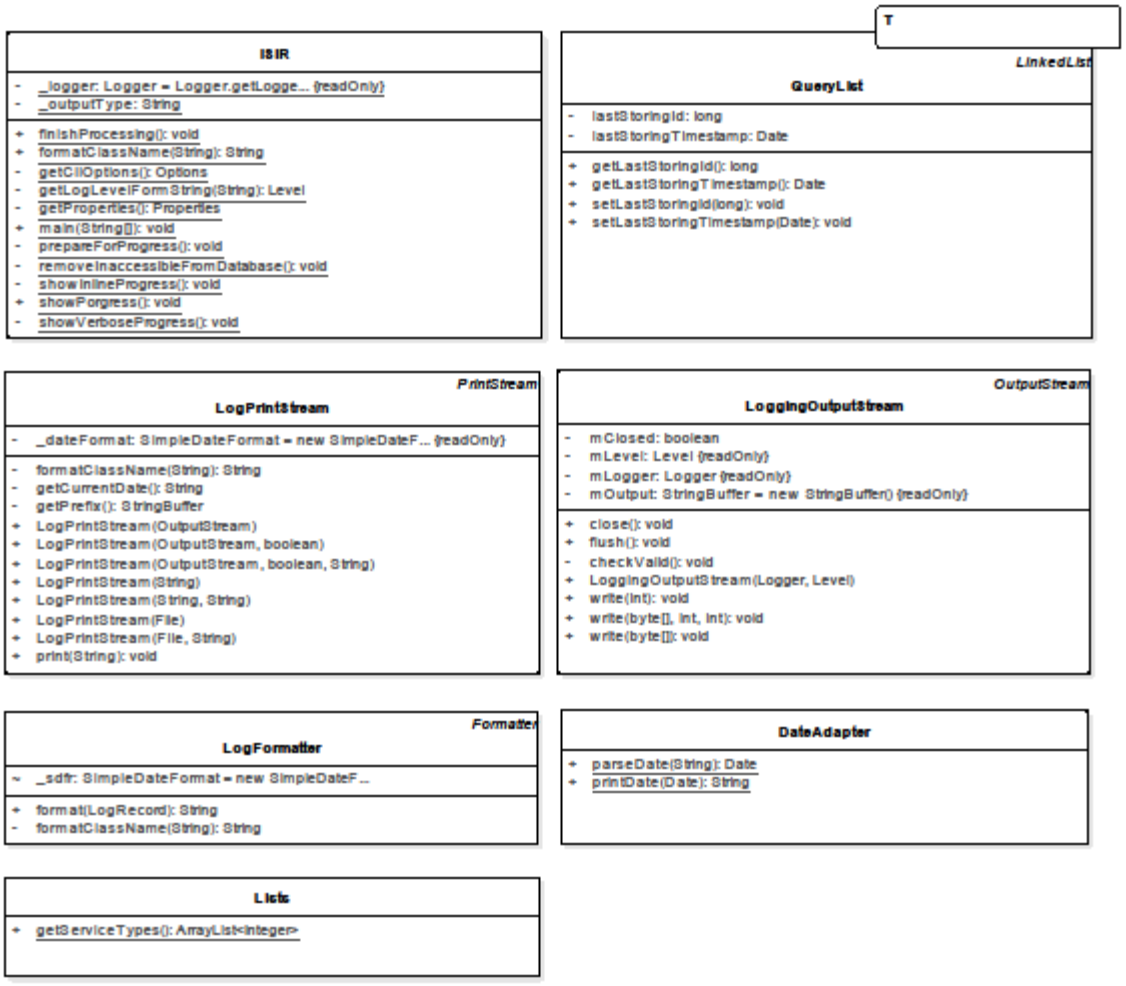
[34] Deadlock. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2016-02-13]. Dostupné z: https://cs.wikipedia.org/wiki/Deadlock

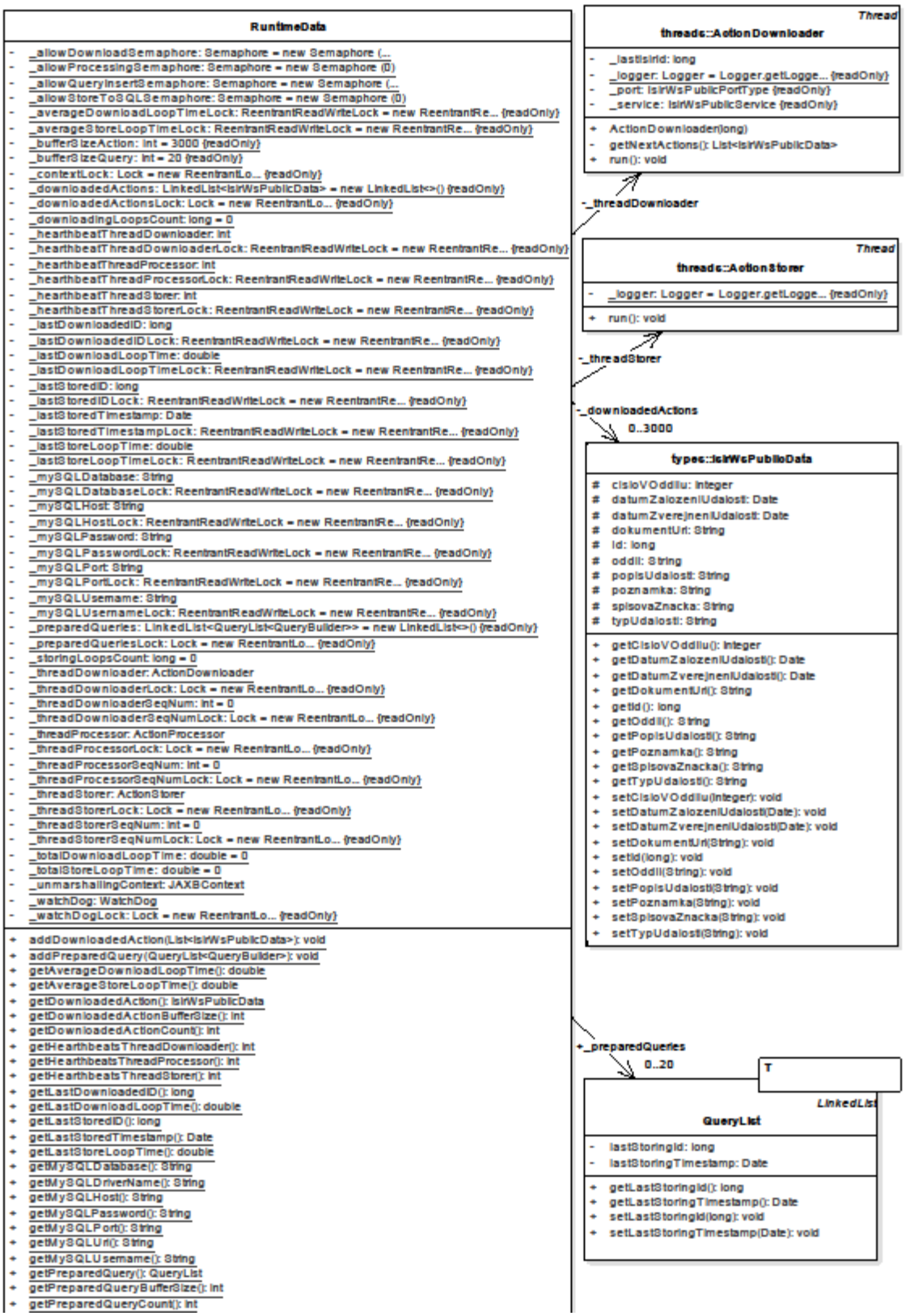
8 Přílohy

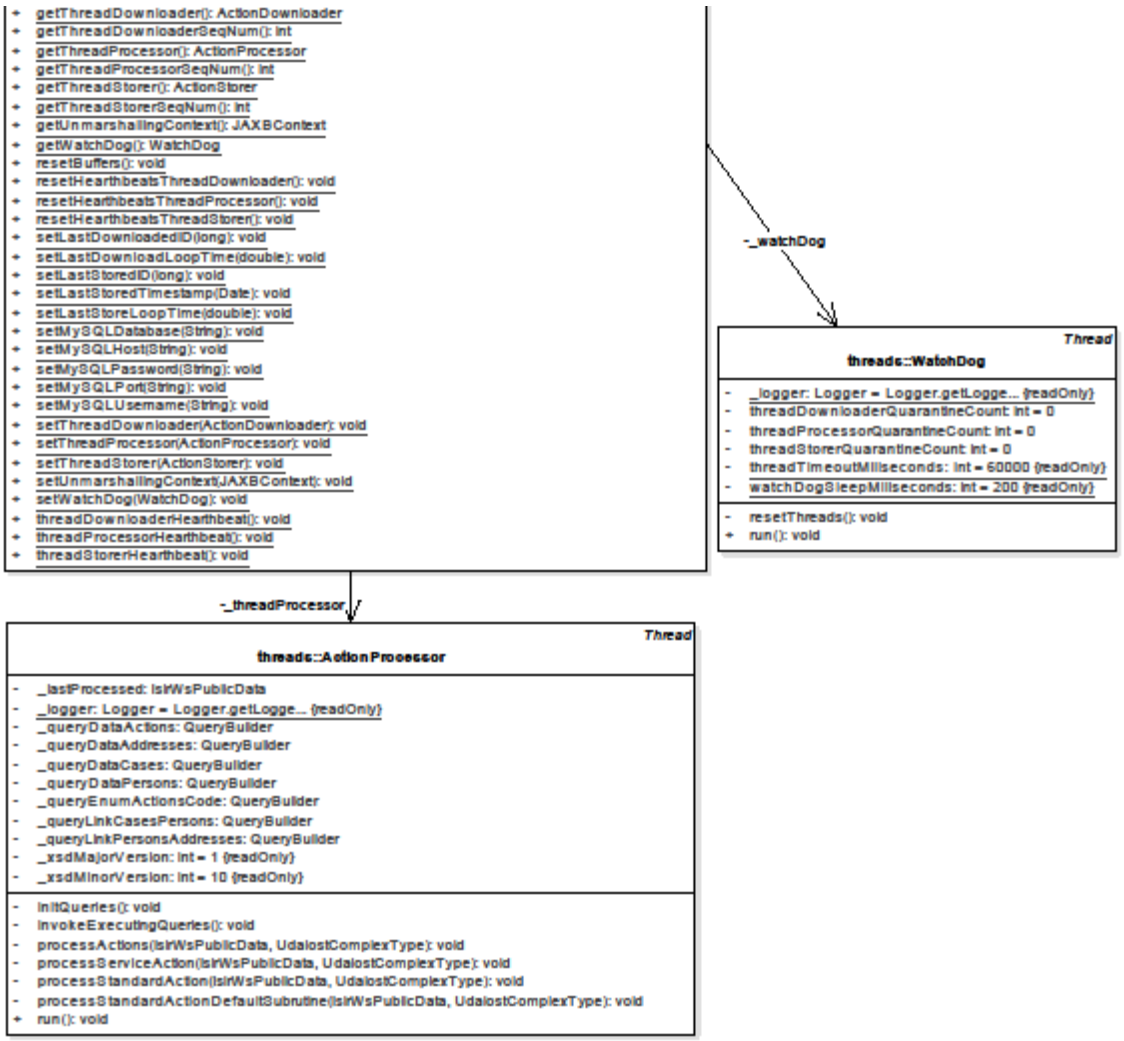
8.1 UML package diagram



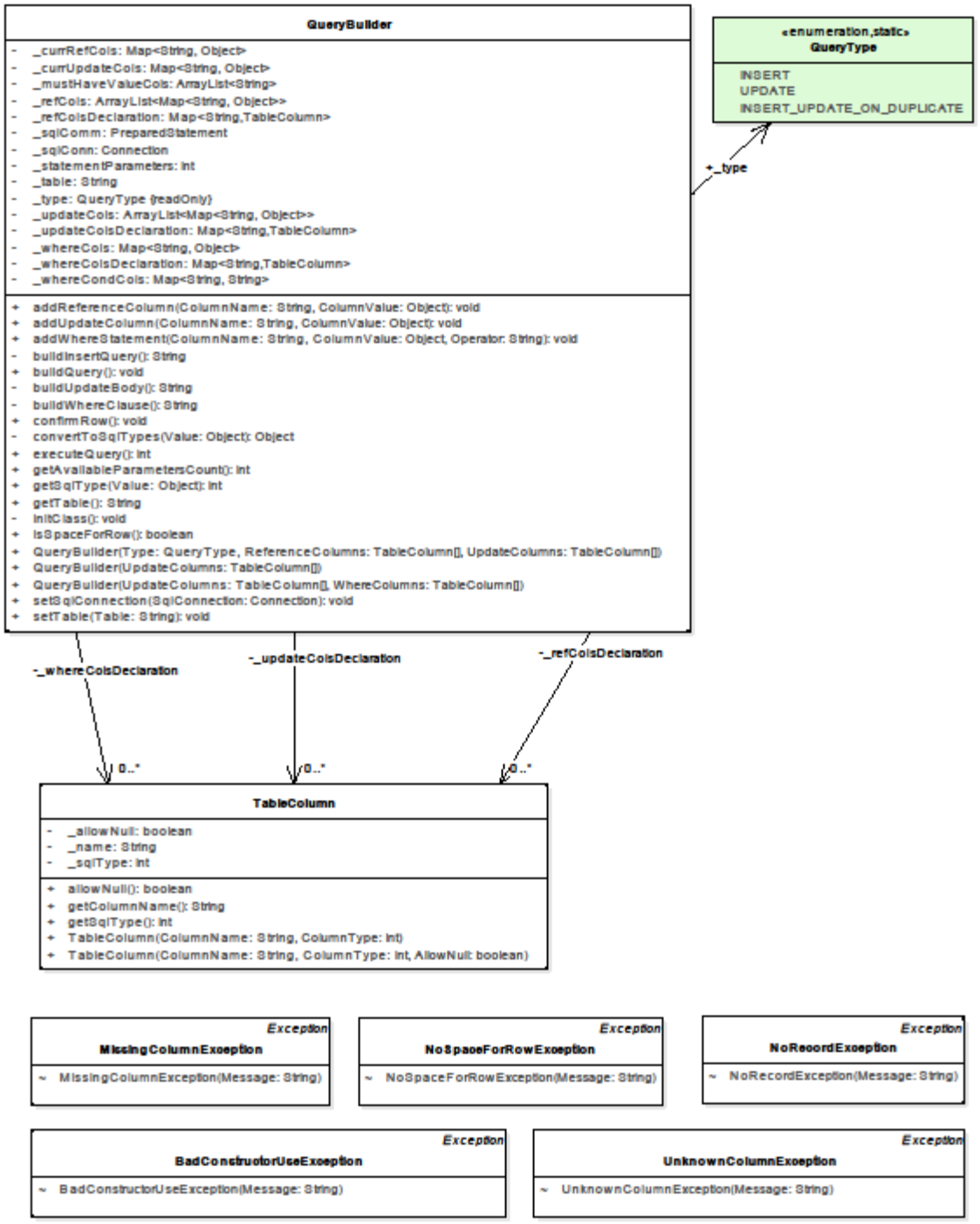
8.2 UML class diagram balíčku cz.jirikubes.isirsync



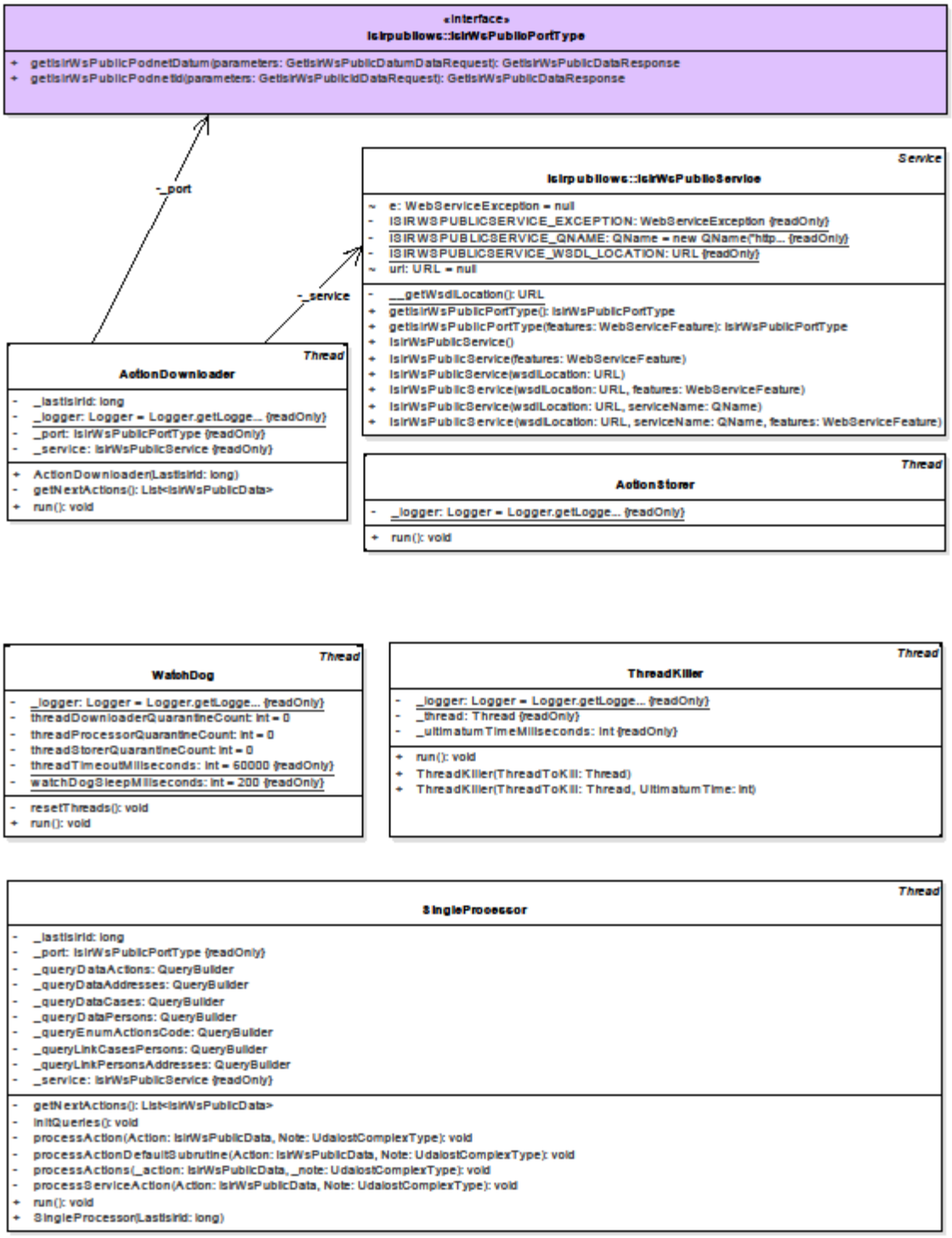

RuntimeData
- _allowDownloadSemaphore: Semaphore = new Semaphore (...
- _allowProcessingSemaphore: Semaphore = new Semaphore (0)
- _allowQueryInsertSemaphore: Semaphore = new Semaphore (...
- _allowStoreToSQLSemaphore: Semaphore = new Semaphore (0)
- _averageDownloadLoopTimeLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _averageStoreLoopTimeLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _bufferSizeAction: int = 3000 {readOnly}
- _bufferSizeQuery: int = 20 {readOnly}
- _contextLock: Lock = new ReentrantLo... {readOnly}
- _downloadedActions: LinkedList<IsirWsPublicData> = new LinkedList<>() {readOnly}
- _downloadedActionsLock: Lock = new ReentrantLo... {readOnly}
- _downloadingLoopsCount: long = 0
- _hearthbeatThreadDownloader: int
- _hearthbeatThreadDownloaderLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _hearthbeatThreadProcessor: int
- _hearthbeatThreadProcessorLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _hearthbeatThreadStorer: int
- _hearthbeatThreadStorerLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _lastDownloadedID: long
- _lastDownloadedIDLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _lastDownloadLoopTime: double
- _lastDownloadLoopTimeLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _lastStoredID: long
- _lastStoredIDLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _lastStoredTimestamp: Date
- _lastStoredTimestampLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _lastStoreLoopTime: double
- _lastStoreLoopTimeLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _mySQLDatabase: String
- _mySQLDatabaseLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _mySQLHost: String
- _mySQLHostLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _mySQLPassword: String
- _mySQLPasswordLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _mySQLPort: String
- _mySQLPortLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _mySQLUsername: String
- _mySQLUsernameLock: ReentrantReadWriteLock = new ReentrantRe... {readOnly}
- _preparedQueries: LinkedList<QueryList<QueryBuilder>> = new LinkedList<>() {readOnly}
- _preparedQueriesLock: Lock = new ReentrantLo... {readOnly}
- _storingLoopsCount: long = 0
- _threadDownloader: ActionDownloader
- _threadDownloaderLock: Lock = new ReentrantLo... {readOnly}
- _threadDownloaderSeqNum: int = 0
- _threadDownloaderSeqNumLock: Lock = new ReentrantLo... {readOnly}
- _threadProcessor: ActionProcessor
- _threadProcessorLock: Lock = new ReentrantLo... {readOnly}
- _threadProcessorSeqNum: int = 0
- _threadProcessorSeqNumLock: Lock = new ReentrantLo... {readOnly}
- _threadStorer: ActionStorer
- _threadStorerLock: Lock = new ReentrantLo... {readOnly}
- _threadStorerSeqNum: int = 0
- _threadStorerSeqNumLock: Lock = new ReentrantLo... {readOnly}
- _totalDownloadLoopTime: double = 0
- _totalStoreLoopTime: double = 0
- _unmarshallingContext: JAXBContext
- _watchDog: WatchDog
- _watchDogLock: Lock = new ReentrantLo... {readOnly}
+ addDownloadedAction(List<IsirWsPublicData>): void
+ addPreparedQuery(QueryList<QueryBuilder>): void
+ getAverageDownloadLoopTime(): double
+ getAverageStoreLoopTime(): double
+ getDownloadedAction(): IsirWsPublicData
+ getDownloadedActionBufferSize(): int
+ getDownloadedActionCount(): int
+ getHearthbeatsThreadDownloader(): int
+ getHearthbeatsThreadProcessor(): int
+ getHearthbeatsThreadStorer(): int
+ getLastDownloadedID(): long
+ getLastDownloadLoopTime(): double
+ getLastStoredID(): long
+ getLastStoredTimestamp(): Date
+ getLastStoreLoopTime(): double
+ getMySQLDatabase(): String
+ getMySQLDriverName(): String
+ getMySQLHost(): String
+ getMySQLPassword(): String
+ getMySQLPort(): String
+ getMySQLUri(): String
+ getMySQLUsername(): String
+ getPreparedQuery(): QueryList
+ getPreparedQueryBufferSize(): int
+ getPreparedQueryCount(): int
Thread
threads::ActionDownloader
- _lastIsirId: long
- _logger: Logger = Logger.getLogge... {readOnly}
- _port: IsirWsPublicPortType {readOnly}
- _service: IsirWsPublicService {readOnly}
+ ActionDownloader(long)
- getNextActions(): List<IsirWsPublicData>
+ run(): void
-_threadDownloader
Thread
threads::ActionStorer
- _logger: Logger = Logger.getLogge... {readOnly}
+ run(): void
-_threadStorer
-_downloadedActions
0..3000
types::IsirWsPublicData
cisloVOddilu: Integer
datumZalozeniUdalosti: Date
datumZverejneniUdalosti: Date
dokumentUri: String
id: long
oddil: String
popisUdalosti: String
poznamka: String
spisovaZnacka: String
typUdalosti: String
+ getCisloVOddilu(): Integer
+ getDatumZalozeniUdalosti(): Date
+ getDatumZverejneniUdalosti(): Date
+ getDokumentUri(): String
+ getId(): long
+ getOddil(): String
+ getPopisUdalosti(): String
+ getPoznamka(): String
+ getSpisovaZnacka(): String
+ getTypUdalosti(): String
+ setCisloVOddilu(Integer): void
+ setDatumZalozeniUdalosti(Date): void
+ setDatumZverejneniUdalosti(Date): void
+ setDokumentUri(String): void
+ setId(long): void
+ setOddil(String): void
+ setPopisUdalosti(String): void
+ setPoznamka(String): void
+ setSpisovaZnacka(String): void
+ setTypUdalosti(String): void
+_preparedQueries
0..20
T
LinkedList
QueryList
- lastStoringId: long
- lastStoringTimestamp: Date
+ getLastStoringId(): long
+ getLastStoringTimestamp(): Date
+ setLastStoringId(long): void
+ setLastStoringTimestamp(Date): void

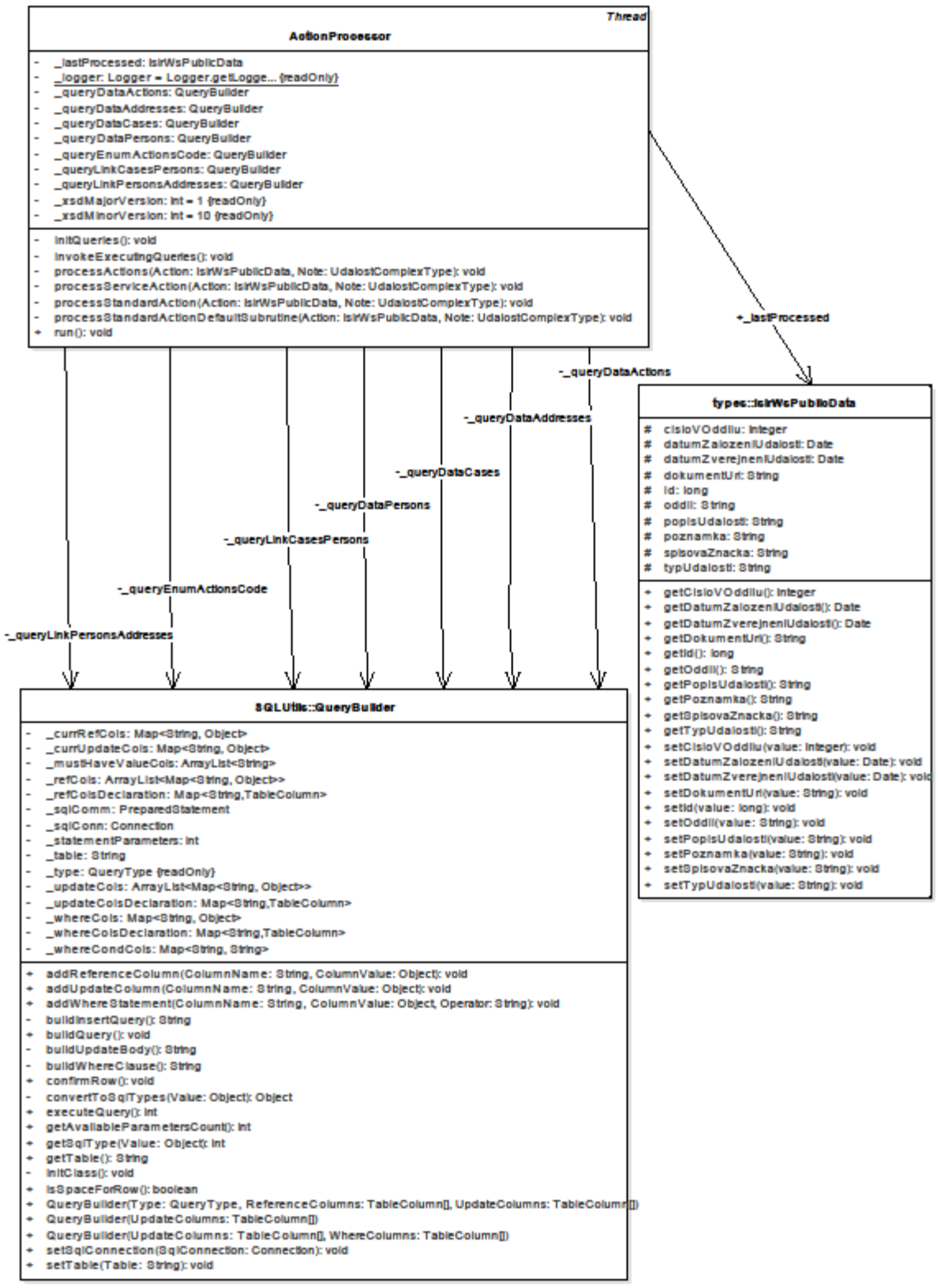

+ getThreadDownloader(): ActionDownloader
+ getThreadDownloaderSeqNum(): int
+ getThreadProcessor(): ActionProcessor
+ getThreadProcessorSeqNum(): int
+ getThreadStorer(): ActionStorer
+ getThreadStorerSeqNum(): int
+ getUnmarshallingContext(): JAXBContext
+ getWatchDog(): WatchDog
+ resetBuffers(): void
+ resetHearthbeatsThreadDownloader(): void
+ resetHearthbeatsThreadProcessor(): void
+ resetHearthbeatsThreadStorer(): void
+ setLastDownloadedID(long): void
+ setLastDownloadLoopTime(double): void
+ setLastStoredID(long): void
+ setLastStoredTimestamp(Date): void
+ setLastStoreLoopTime(double): void
+ setMySQLDatabase(String): void
+ setMySQLHost(String): void
+ setMySQLPassword(String): void
+ setMySQLPort(String): void
+ setMySQLUsername(String): void
+ setThreadDownloader(ActionDownloader): void
+ setThreadProcessor(ActionProcessor): void
+ setThreadStorer(ActionStorer): void
+ setUnmarshallingContext(JAXBContext): void
+ setWatchDog(WatchDog): void
+ threadDownloaderHearthbeat(): void
+ threadProcessorHearthbeat(): void
+ threadStorerHearthbeat(): void
-_watchDog
Thread
threads::WatchDog
- _logger: Logger = Logger.getLogge... {readOnly}
- threadDownloaderQuarantineCount: int = 0
- threadProcessorQuarantineCount: int = 0
- threadStorerQuarantineCount: int = 0
- threadTimeoutMilliseconds: int = 60000 {readOnly}
- watchDogSleepMilliseconds: int = 200 {readOnly}
- resetThreads(): void
+ run(): void
-_threadProcessor
Thread
threads::ActionProcessor
- _lastProcessed: IsirWsPublicData
- _logger: Logger = Logger.getLogge... {readOnly}
- _queryDataActions: QueryBuilder
- _queryDataAddresses: QueryBuilder
- _queryDataCases: QueryBuilder
- _queryDataPersons: QueryBuilder
- _queryEnumActionsCode: QueryBuilder
- _queryLinkCasesPersons: QueryBuilder
- _queryLinkPersonsAddresses: QueryBuilder
- _xsdMajorVersion: int = 1 {readOnly}
- _xsdMinorVersion: int = 10 {readOnly}
- initQueries(): void
- invokeExecutingQueries(): void
- processActions(IsirWsPublicData, UdalostComplexType): void
- processServiceAction(IsirWsPublicData, UdalostComplexType): void
- processStandardAction(IsirWsPublicData, UdalostComplexType): void
- processStandardActionDefaultSubrutine(IsirWsPublicData, UdalostComplexType): void
+ run(): void

8.3 UML class diagram balíčku cz.jirikubes.isirsync.SQLutils

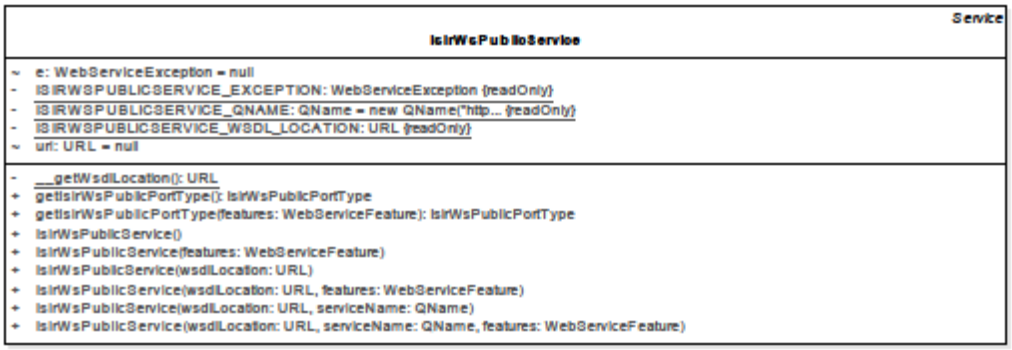


8.4 UML class diagram balíčku cz.jirikubes.isirsync.threads

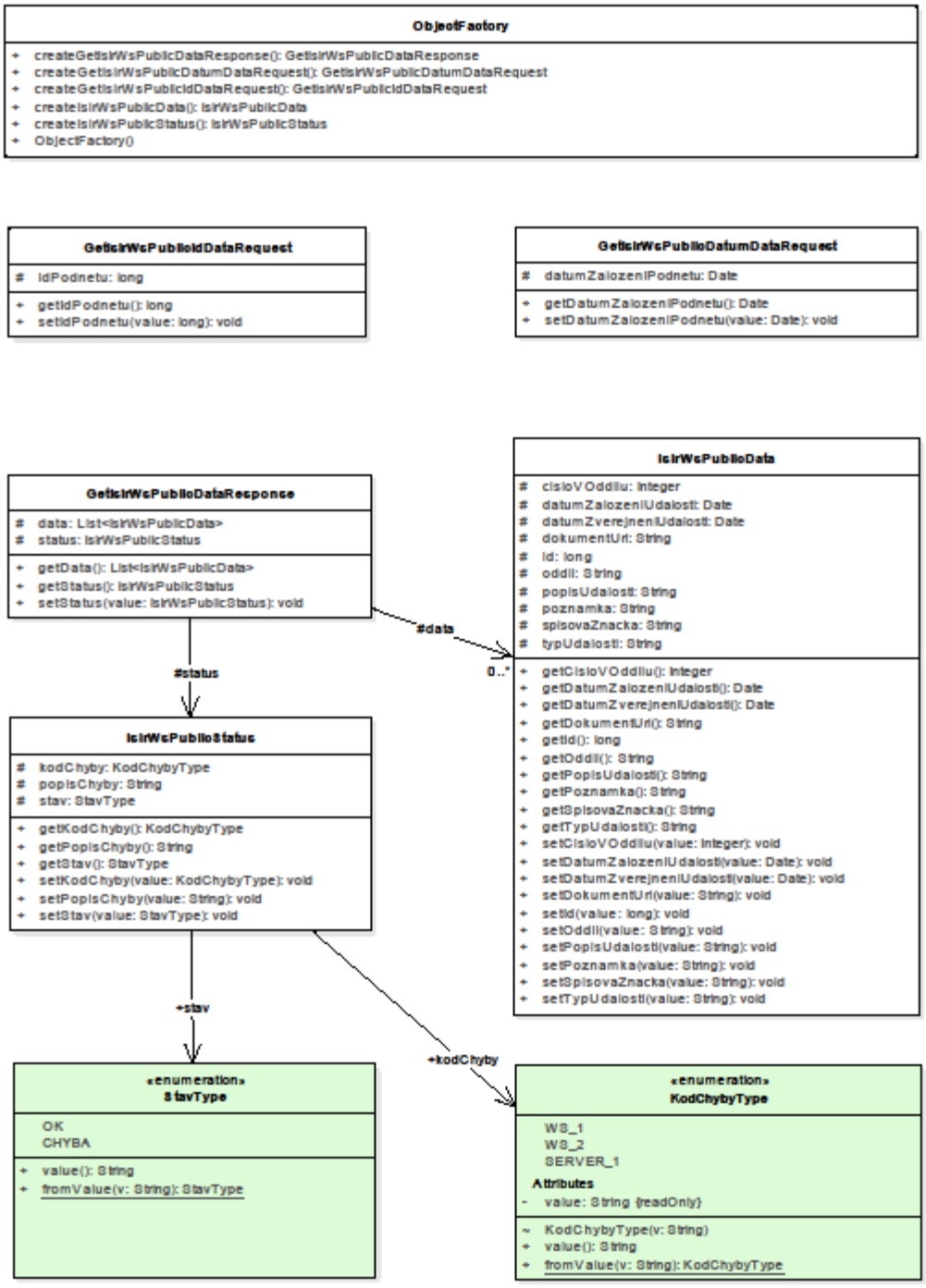



Thread
ActionProcessor
- _lastProcessed: IsirWsPublicData
- _logger: Logger = Logger.getLogge... {readOnly}
- _queryDataActions: QueryBuilder
- _queryDataAddresses: QueryBuilder
- _queryDataCases: QueryBuilder
- _queryDataPersons: QueryBuilder
- _queryEnumActionsCode: QueryBuilder
- _queryLinkCasesPersons: QueryBuilder
- _queryLinkPersonsAddresses: QueryBuilder
- _xsdMajorVersion: Int = 1 {readOnly}
- _xsdMinorVersion: Int = 10 {readOnly}
- initQueries(): void
- invokeExecutingQueries(): void
- processActions(Action: IsirWsPublicData, Note: UdalostComplexType): void
- processServiceAction(Action: IsirWsPublicData, Note: UdalostComplexType): void
- processStandardAction(Action: IsirWsPublicData, Note: UdalostComplexType): void
- processStandardActionDefaultSubrutine(Action: IsirWsPublicData, Note: UdalostComplexType): void
+ run(): void
+-_lastProcessed
-_queryDataActions
-_queryDataAddresses
-_queryDataCases
-_queryDataPersons
-_queryLinkCasesPersons
-_queryEnumActionsCode
-_queryLinkPersonsAddresses
types::IsirWsPublicData
cisloVOddilu: Integer
datumZalozeniUdalosti: Date
datumZverejneniUdalosti: Date
dokumentUri: String
id: long
oddil: String
popisUdalosti: String
poznamka: String
spisovaZnacka: String
typUdalosti: String
+ getCisloVOddilu(): Integer
+ getDatumZalozeniUdalosti(): Date
+ getDatumZverejneniUdalosti(): Date
+ getDokumentUri(): String
+ getId(): long
+ getOddil(): String
+ getPopisUdalosti(): String
+ getPoznamka(): String
+ getSpisovaZnacka(): String
+ getTypUdalosti(): String
+ setCisloVOddilu(value: Integer): void
+ setDatumZalozeniUdalosti(value: Date): void
+ setDatumZverejneniUdalosti(value: Date): void
+ setDokumentUri(value: String): void
+ setId(value: long): void
+ setOddil(value: String): void
+ setPopisUdalosti(value: String): void
+ setPoznamka(value: String): void
+ setSpisovaZnacka(value: String): void
+ setTypUdalosti(value: String): void
SQLUtils::QueryBuilder
- _currRefCols: Map<String, Object>
- _currUpdateCols: Map<String, Object>
- _mustHaveValueCols: ArrayList<String>
- _refCols: ArrayList<Map<String, Object>>
- _refColsDeclaration: Map<String,TableColumn>
- _sqlComm: PreparedStatement
- _sqlConn: Connection
- _statementParameters: Int
- _table: String
- _type: QueryType {readOnly}
- _updateCols: ArrayList<Map<String, Object>>
- _updateColsDeclaration: Map<String,TableColumn>
- _whereCols: Map<String, Object>
- _whereColsDeclaration: Map<String,TableColumn>
- _whereCondCols: Map<String, String>
+ addReferenceColumn(ColumnName: String, ColumnValue: Object): void
+ addUpdateColumn(ColumnName: String, ColumnValue: Object): void
+ addWhereStatement(ColumnName: String, ColumnValue: Object, Operator: String): void
- buildInsertQuery(): String
+ buildQuery(): void
- buildUpdateBody(): String
- buildWhereClause(): String
+ confirmRow(): void
- convertToSqlTypes(Value: Object): Object
+ executeQuery(): Int
+ getAvailableParametersCount(): Int
+ getSqlType(Value: Object): Int
+ getTable(): String
- initClass(): void
+ isSpaceForRow(): boolean
+ QueryBuilder(Type: QueryType, ReferenceColumns: TableColumn[], UpdateColumns: TableColumn[])
+ QueryBuilder(UpdateColumns: TableColumn[])
+ QueryBuilder(UpdateColumns: TableColumn[], WhereColumns: TableColumn[])
+ setSqlConnection(SqlConnection: Connection): void
+ setTable(Table: String): void

8.5 UML class diagram balíčku cz.cca.isirpublicws



8.6 UML class diagram balíčku cz.cca.isirpublicws.types



8.7 UML class diagram balíčku cz.cca.isirpublicws.note


+ setIdOsoby1(value: String): void
+ setIdOsoby2(value: String): void
+ setIdOsobyPuvodce(value: String): void
+ setObecnyText(value: String): void
+ setOsoba(value: OsobaComplexType): void
+ setOsobaVeriteI(value: String): void
+ setPoznamkaXSD(value: PoznamkaXSDComplexType): void
+ setPriznakAnVedlejsiDokument(value: String): void
+ setPriznakAnVedlejsiUdalost(value: String): void
+ setPriznakMylnyZapisVeriteIPohled(value: String): void
+ setPriznakPlatnyVeritel(value: String): void
+ setRocnikHlavni(value: Long): void
+ setRocnikICm(value: Long): void
+ setRocnikVsNs(value: Long): void
+ setSenatICm(value: Long): void
+ setSenatVsNs(value: Long): void
+ setVec(value: VecComplexType): void
+ setVerzeXsd(value: String): void
#osoba
OsobaComplexType
adresa: AdresaComplexType
datumNarozeni: Date
datumOsobaVeVeciZrusena: Date
dic: String
druhOsoby: String
druhPravniForma: String
druhRoleVRizeni: String
druhSpravce: String
ic: String
idOsoby: String
jmeno: String
nazevOsoby: String
nazevOsobyObchodni: String
rc: String
titulPred: String
titulZa: String
+ getAdresa(): AdresaComplexType
+ getDatumNarozeni(): Date
+ getDatumOsobaVeVeciZrusena(): Date
+ getDic(): String
+ getDruhOsoby(): String
+ getDruhPravniForma(): String
+ getDruhRoleVRizeni(): String
+ getDruhSpravce(): String
+ getIc(): String
+ getIdOsoby(): String
+ getJmeno(): String
+ getNazevOsoby(): String
+ getNazevOsobyObchodni(): String
+ getRc(): String
+ getTitulPred(): String
+ getTitulZa(): String
+ setAdresa(value: AdresaComplexType): void
+ setDatumNarozeni(value: Date): void
+ setDatumOsobaVeVeciZrusena(value: Date): void
+ setDic(value: String): void
+ setDruhOsoby(value: String): void
+ setDruhPravniForma(value: String): void
+ setDruhRoleVRizeni(value: String): void
+ setDruhSpravce(value: String): void
+ setIc(value: String): void
+ setIdOsoby(value: String): void
+ setJmeno(value: String): void
+ setNazevOsoby(value: String): void
+ setNazevOsobyObchodni(value: String): void
+ setRc(value: String): void
+ setTitulPred(value: String): void
+ setTitulZa(value: String): void
#adresa
AdresaComplexType
cisloPopisne: String
datumPobytDo: Date
datumPobytOd: Date
druhAdresy: String
fax: String
idAdresy: String
mesto: String
okres: String
psc: String
telefon: String
textAdresy: String
ulice: String
zeme: String
+ getCisloPopisne(): String
+ getDatumPobytDo(): Date
+ getDatumPobytOd(): Date
+ getDruhAdresy(): String
+ getFax(): String
+ getIdAdresy(): String
+ getMesto(): String
+ getOkres(): String
+ getPsc(): String
+ getTelefon(): String
+ getTextAdresy(): String
+ getUlice(): String
+ getZeme(): String
+ setCisloPopisne(value: String): void
+ setDatumPobytDo(value: Date): void
+ setDatumPobytOd(value: Date): void
+ setDruhAdresy(value: String): void
+ setFax(value: String): void
+ setIdAdresy(value: String): void
+ setMesto(value: String): void
+ setOkres(value: String): void
+ setPsc(value: String): void
+ setTelefon(value: String): void
+ setTextAdresy(value: String): void
+ setUlice(value: String): void
+ setZeme(value: String): void

8.8 Výčet elektronických příloh

1) Text této práce (složka 1_text)
2) MySQL model pro vytvoření databáze pomocí nástroje MySQL Workbench (soubor 2_db_model.mwb)
3) SQL skript pro vytvoření databáze (soubor 3_create_db.sql)
4) Sada SQL skriptů pro import dat do databáze (složka 4_data_import)
5) Zdrojové kódy v podobě projektu pro IDE NetBeans 8 (soubor 5_source.zip)
6) Zkompilovaný program z přiloženého zdrojového kódu (složka 6_bin)
7) UML class diagram zobrazující vazby mezi všemi třídami (soubor 7_ISIR_UML.jpg)
8) Dokumentace zdrojového kódu v podobě JavaDoc (složka 8_javadoc)
9) Poster (složka 9_poster)