MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Digitální repozitář Fedora

DIPLOMOVÁ PRÁCE

Stanislav Novotný

Brno, 2006

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Miroslav Bartošek, CSc.

Poděkování

Děkuji svému vedoucímu této diplomové práce, RNDr. Miroslavu Bartoškovi, CSc., za trpělivost a pečlivé vedení po celou dobu řešení a Mgr. Vlastimilu Krejčířovi za cenné připomínky a rady při realizaci. Poděkování patří též Bohumile Ručkové za neustálé povzbuzování, užitečné postřehy a jazykovou korekturu textu.

Shrnutí

V dnešní době velkého množství informací v digitální podobě se ukazuje jako nezbytné řešit rozumným způsobem jejich uchovávání, popis a nakládání s nimi. V posledních letech se proto začala rozvíjet teorie týkající se digitálních knihoven, které umožňují společně s daty uchovávat i jejich metadata, vztahy mezi objekty a zpřístupňovat služby.

Jednou z konkrétních implementací je systém Fedora, jenž si dal za cíl stát se univerzálním flexibilním snadno rozšiřitelným repozitářem. Tato práce popisuje současný stav vývoje a usnadňuje vývojářům a správcům získání přehledu o možnostech a omezeních použití.

Jelikož zatím neexistuje oficiální uživatelsky použitelný klient, je součástí práce také ukázání jeho realizovatelnosti. Za tímto účelem byla vytvořena webová aplikace, která umožňuje přístup a správu digitálních objektů v repozitáři. Navržené řešení je dostatečně obecné a lze ho snadným způsobem rozšiřovat o podporu dalších typů digitálních objektů, jejich pohledů a akcí. Použitelnost byla otestována na vytvořené sbírce ukázkových dat.

Klíčová slova

digitální knihovna, digitální objekt, repozitář, metadata, identifikátor, diseminátor, *Fedora*

Sem bude v papírové podobě diplomové práce vložena kopie oficiálního zadání.

Obsah

1 **Úvod** 4
2 **Teorie digitálních knihoven** 5
2.1 Definice 5
2.2 Kahn-Wilenského architektura 6
3 **Technologie a standardy** 7
3.1 Dublin Core metadata 7
3.2 PURL 7
3.3 OAI Protocol for Metadata Harvesting 8
3.4 Webové služby 8
3.5 REST 9
3.5.1 Základní charakteristiky 9
3.5.2 Příklad 9
3.6 RDF 10
4 **Digitální repozitář Fedora** 12
4.1 Historie 12
4.2 Přednosti a nevýhody Fedory 13
4.3 Architektura systému 13
4.4 Model digitálních objektů 16
4.5 Typy digitálních objektů 16
4.6 PID 17
4.7 Vlastnosti objektu 18
4.8 Datové proudy 18
4.9 Diseminátory 19
4.10 Časové známky 21
4.11 Fedora Object XML (FOXML) 21
4.11.1 Vlastnosti objektu 22
4.11.2 Datové proudy 23
Typ X 23
Typy M, R a E 24
4.11.3 Diseminátory 24
4.12 Datové proudy se zvláštním významem 25
4.12.1 DC 25

4.12.2 RELS-EXT . . . 25
4.12.3 AUDIT . . . 26
4.12.4 POLICY . . . 26
4.13 Index zdrojů . . . 27
4.14 Uživatelé . . . 30
4.15 Politiky přístupu . . . 32
4.15.1 Struktura zápisu XACML politiky . . . 34
4.16 Struktura distribuce Fedory . . . 35
4.17 Klient Fedory . . . 37
5 **Služby poskytované repozitářem Fedora** . . . 39
5.1 Fedora Access Service . . . 39
5.1.1 Informace o repozitáři . . . 40
5.1.2 Vyhledávání objektů . . . 40
5.1.3 Práce s konkrétním digitálním objektem . . . 41
5.2 Fedora Management Service . . . 41
5.2.1 Informace o uživateli . . . 42
5.2.2 Vkládání a odstraňování objektů . . . 42
5.2.3 Změna vlastností objektu . . . 42
5.2.4 Práce s datovými proudy . . . 43
5.2.5 Práce s diseminátory . . . 44
5.2.6 Export objektu . . . 45
6 **Implementace webového rozhraní** . . . 46
6.1 Licence . . . 46
6.2 Konfigurace . . . 47
6.3 Repozitář . . . 47
6.4 Uživatel . . . 47
6.5 Typ objektu . . . 48
6.6 Pohledy . . . 49
6.7 Akce . . . 50
6.8 Zpřístupnění dat přes URL . . . 52
6.9 Soubory s nastaveními aplikace . . . 52
6.9.1 objectTypes.xml . . . 52
6.9.2 config.xml . . . 53
6.10 Konkrétní typy objektů . . . 54
6.10.1 Adresáře . . . 54
6.10.2 Obrázek . . . 54
6.10.3 Článek . . . 55
7 **Zkušenosti a postřehy** . . . 56
7.1 První krůčky . . . 56
7.2 První zjištění . . . 57

7.3 Navrhované architektury implementace klienta 58
7.3.1 Fedora jako repozitář 58
7.3.2 Přístup pomocí diseminátorů 58
7.3.3 Implementace typů uložena přímo v digitální knihovně 58
7.4 Vývoj Fedory 59
7.5 Na co je třeba si dát pozor 59
7.6 Dostupné systémy 60
8 **Závěr** 62
A **Instalace serveru Fedory** 67
B **Instalace webového klienta** 71

Kapitola 1

Úvod

V dnešním světě Internetu, v němž může téměř kdokoliv cokoliv publikovat, panuje velký zmatek a je obtížné jednoduše a rychle najít požadovanou informaci. Pomoci vyřešit tyto problémy, zejména prostřednictvím kategorizace, popisu pomocí metadat, služeb nad daty a povolením přístupu jen oprávněným uživatelům, se snaží digitální knihovny.

Jednou z volně dostupných digitálních knihoven je Fedora, jejíž vývoj v poslední době vypadá nadějně a vzniká z ní mocný mnohostranný nástroj. V současnosti však dosud neobsahuje rozhraní, které by poskytovalo přijatelný přístup pro koncového uživatele. Součástí této práce je implementace webové aplikace, jež umožňuje prohlížení obsahu repozitáře a správu v něm obsažených digitálních objektů.

Tato práce nejprve popisuje obecně digitální knihovny, jejich teorii, používané techniky a standardy. V další části je rozebrána Fedora (verze 2.1, respektive 2.1.1, která byla uvolněna během psaní této práce), její možnosti a omezení. Třetí oddíl se věnuje popisu realizovaného klienta, způsobům jeho nastavení a postupům k dodefinování vlastních tříd, reprezentujících typy digitálních objektů. V poslední části jsou sepsány mé vlastní zkušenosti s používáním Fedory, postřehy a náměty k vylepšení.

Při práci jsem čerpal ze zdrojů uvedených v Literatuře.

Kapitola 2

Teorie digitálních knihoven

V této kapitole budou vysvětleny základní pojmy týkající se digitálních knihoven.

2.1 Definice

Pojem digitální knihovna nemá jednotnou definici. Lze se na něj dívat z různých úhlů pohledu a vyvíjí se též v čase s rozvojem a možnostmi výpočetní techniky.

Pohled knihovníka z „kamenné knihovny“:

> Digitální knihovny jsou organizace, které poskytují zdroje (včetně specializovaného personálu) umožňující provádět výběr, strukturování a zpřístupnění sbírek digitálních prací, tyto práce dále distribuovat, udržovat jejich integritu a dlouhodobě uchovávat – to vše s ohledem na snadné a ekonomické využití určitou komunitou nebo množinou komunit uživatelů.
>
> Waters, Donald J.: *What Are Digital Libraries?*, [1]

Pro tuto práci je důležitý zejména pohled z počítačového hlediska:

> Digitální knihovna je spravovaná sbírka informací spolu s odpovídajícími službami, přičemž informace jsou uloženy v digitální podobě a jsou dostupné prostřednictvím sítě.
>
> Arms, W. Y., [1]

Z uvedených definic se dá vyčíst, že digitální knihovna se skládá z repozitáře, v němž jsou uchovávána data, a množiny služeb nad uloženými informacemi. Důležité je především jednoduché, efektivní, rychlé a uživatelsky přijatelné vyhledávání požadovaných objektů. Sbírky jsou určeny vždy pouze omezené komunitě uživatelů, pro niž jsou náležitě přizpůsobeny. Vytvořit obecnou, všeobjímající knihovnu je prakticky nemožné.

2.2 Kahn-Wilenského architektura

Nejpropracovanější teorii, na níž je postavena i digitální knihovna Fedora, nabízí ve své práci Robert Kahn a Robert Wilensky (viz [22]). Systém by podle jejich návrhu měl vypadat následovně:

Původce převede svůj materiál, který chce zpřístupnit, na *digitální objekt*. To je struktura, jejíž součástí jsou digitální materiály nebo data, metadata a *jednoznačný identifikátor* (handle). Ten je objektu přiřazen *generátorem identifikátorů* (handle generator). Uživatel potom může zpřístupnit daný digitální objekt přes jeden nebo více *repozitářů* a následně jej registrovat ve veřejně dostupném systému identifikačních serverů (handle server). Tento objekt je zpravidla určen široké veřejnosti a nazývá se pak *registrovaný digitální objekt*.

Každý repozitář musí umět komunikovat s ostatními úložišti prostřednictvím protokolu RAP (Repository Access Protocol). Repozitář může obsahovat i neregistrované digitální objekty a zpřístupnit je jiným protokolem, než je RAP.

Kapitola 3

Technologie a standardy

3.1 Dublin Core metadata

Dublin Core Metadata Initiative (DCMI, [6]) je otevřené fórum, které se zabývá popisnými metadaty Dublin Core a vytvářením jejich standardů.

Dublin Core (dále jen DC) je množina vlastností, jež se vztahují k danému objektu a mají za úkol umožnit a usnadnit jeho vyhledávání. Počet údajů pro kteroukoliv z vlastností je libovolný, může být nulový, nebo se naopak informace mohou opakovat podle potřeby.

V současné době má Dublin Core 15 metadatových vlastností. Těmi jsou *název* (title), *tvůrce* (creator), *předmět* (subject), *popis* (description), *vydavatel* (publisher), *přispěvatel* (contributor), *datum* (date), *typ* (type), *formát* (format), *identifikátor* (identifier), *zdroj* (source), *jazyk* (language), *vztah* (relation), *pokrytí* (coverage) a *práva* (rights).

Každá z těchto vlastností může obsahovat tzv. *kvalifikátor*, který blíže specifikuje, co daná položka popisuje. Na příklad jméno ilustrátora knihy můžeme uvést v položce CREATOR.Illustrator. Význam však musí být zachován i pro systémy, jež kvalifikátory nepodporují. Ty ho pak budou považovat za blíže nespecifikovaného autora.

3.2 PURL

Persistent Uniform Resource Locators (PURL, [16]) je způsob, kterým lze zajistit, že daný zdroj na Internetu bude mít v čase neustále neměnný identifikátor, podle něhož půjde vyhledat a nalézt. Klasická URL adresa tento úkol neřeší, protože v sobě obsahuje přímo identifikaci serveru, na kterém se hledaná data nacházejí. Při používání Internetu se neustále setkáváme s chybovými hlášeními, že námi hledané objekty byly přesunuty a že tedy mají jiné umístění. Běžného uživatele ale nezajímá, odkud požadovaná data přijímá, hledá je kvůli jejich obsahu. Toto hlášení jej obtěžuje a zdržuje.

Na druhé straně vystavovatelé informací při takovéto změně přicházejí o své někdy dosti složitě získané návštěvníky, kteří mají adresy uloženy

ve svém prohlížeči nebo jsou na ně zvyklí a mají je „uložené v hlavě“.

Princip OCLC PURL je jednoduchý. Vystavovatel registruje svoji URL adresu a dostane přidělenu PURL adresu. Ta není ve skutečnosti ničím jiným než URL adresou, která odkazuje na server registrátora. Při přístupu k ní je prohlížeč automaticky přesměrován na skutečnou lokaci zdroje. Pokud dojde k přemístění dat, PURL zůstává pořád stejné, stačí pouze změnit místo odkazu na novou hodnotu.

3.3 OAI Protocol for Metadata Harvesting

Open Archives Initiative (OAI, [8]) byla založena za účelem vytvoření a rozšíření spolupráce mezi jednotlivými repozitáři. Její protokol OAI-PMH (OAI Protocol for Metadata Harvesting) popisuje způsob výměny informací mezi nezávislými repozitáři a systémem, který jejich metadata sklízí a dále zpracovává (například zpřístupňuje pro vyhledávání). Hlavními úkoly tohoto protokolu je celosvětové propojení vědeckých archivů, volný přístup alespoň k jejich metadatům, konzistentní rozhraní mezi knihovnami a zprostředkovateli služeb a snadno implementovatelný bezbariérový přístup. Celý postup komunikace je založený na protokolech HTTP, XML a DC.

3.4 Webové služby

Webové služby jsou technologie umožňující vzdálené spouštění funkcí. Komunikující strany mohou být používány v různých prostředích na webu, služby implementovány v rozdílných programovacích jazycích, aniž o tom protějšek má podrobnější informace. Technologie je postavená na protokolu *SOAP* (Simple Object Access Protocol, [32]). Server, který nabízí webovou službu, publikuje ve formátu *WSDL* (Web Services Description Language, [31]) formální popis rozhraní, tj. nabízených funkcí a jejich parametrů. Pomocí takto získaných informací může klient automaticky vygenerovat SOAP požadavek, který zašle serveru. Ten po zpracování vrátí v SOAP odpovědi návratovou hodnotu.

WSDL, požadavek i odpověď jsou kódovány v jazyce XML. „Ruční“ zpracovávání použitých XML, vyvinuté každým vývojářem pro svoji potřebu, by bylo zbytečné a namáhavé. Lze tedy aplikovat nástroje, které manipulace usnadní. Mezi ty nejčastěji používané patří např. Apache Axis ([10]), některé z nich jsou integrovány přímo do vývojových prostředí, mezi něž patří i NetBeans ([43]). Ty potom umožní vytvoření webové služby z javovské třídy nebo naopak vygenerování třídy podle popisu WSDL. Je-li

v ní pak zavolána metoda, použitý framework vygeneruje z parametrů příslušný SOAP požadavek, který odešle druhé straně. Návratovou hodnotu vrátí volajícímu kódu, popř. upozorní na výjimku při neúspěšném pokusu.

3.5 REST

Representational State Transfer (REST) je styl architektury, který byl poprvé popsán v doktorské dizertační práci Roye Thomase Fieldinga (viz [38]). Postaven je na standardech XML (nebo HTML) a HTTP.

3.5.1 Základní charakteristiky

- *architektura klient-server*
- *bezstavovost*: server si neuchovává žádné údaje o stavu, ve kterém se nachází, všechny parametry musí být součástí požadavku
- *vyrovnávací paměti*: každá odpověď musí být označena, je-li možno ji uložit do cache
- *jednotné rozhraní*: všechny zdroje jsou přístupné přes jednotné rozhraní (např. prostřednictvím HTTP GET, POST, PUT, DELETE)
- *pojmenované zdroje*: systém se skládá ze zdrojů, každý z nich je pojmenovány unikátní URL adresou
- *použití médií provázaných odkazy*: reprezentace zdrojů obsahují jak vlastní informace, tak také odkazy, pomocí nichž je umožněn přechod k ostatním zdrojům nebo z jednoho stavu do druhého
- *vrstvené komponenty*: prostředníci, jakými jsou např. proxy servery nebo vyrovnávací paměti, mohou být libovolně vloženy mezi klienta a zdroje k zajištění vyššího výkonu, bezpečnosti atd.

Reprezentace v REST systémech jsou obvykle kódovány ve formě HTML nebo XML.

3.5.2 Příklad

Pro názornost uvádím příklad, který jsem převzal z Wikipedie (viz [39]). Znázorněn je rozdíl přístupu REST vůči klasickému vzdálenému volání procedur (Remote Procedure Call, RPC). V něm se klade důraz na různost funkcí, které se označují slovesy. `getUser()`,`addUser()`,`removeUser()`,

`updateUser()`,
`getLocation()`,`addLocation()`,`removeLocation()`,
`updateLocation()`,
`listUsers()`,`listLocations()`,`findLocation()`,`findUser()`.

Naproti tomu REST se zaměřuje na samotné zdroje (podstatná jména). Výše popsaný systém by měl tedy dva typy – `User` a `Location`.

Každý konkrétní zdroj má svůj vlastní jedinečný identifikátor, např. `http://example.cz/locations/us/ny/new_york_city` reprezentující město New York City ve státě New York ve Spojených státech. Uživatelé se zdroji pracují prostřednictvím běžných HTTP operací – GET pro získání, PUT pro přidání nebo změnu, DELETE pro odstranění. POST je vyhrazeno pro funkce s vedlejšími efekty, tj. těmi, které mění stavy.

Záznam o uživateli může vypadat následovně.

```
<user>
 <name>Jane User</name>
 <gender>female</gender>
 <location href=
   "http://example.cz/locations/us/ny/new_york_city">
          New York City, NY, US
  </location>
</user>
```

Postup při změně údajů bude následovný: Klient prostřednictvím požadavku GET stáhne aktuální instanci, provede v ní změny a pomocí PUT nový stav odešle serveru.

Za povšimnutí stojí, že mezi požadavky HTTP není žádný pro vyhledávání a výpis seznamu. Neexistuje HTTP FIND nebo HTTP LIST, které by odpovídalo funkcím `find*` a `list*`. Z tohoto důvodu je každá kolekce sama zdrojem. Pro výpis seznamu měst ve státě New York (včetně odkazů na ně) tedy přistoupíme k adrese `http://example.cz/locations/us/ny/`. Při vyhledávání předáváme hledané hodnoty prostřednictvím HTTP parametrů, seznam odkazů na uživatele s příjmením Michaels tedy získáme pomocí dotazu `http://example.cz/users?surname=Michaels`.

3.6 RDF

Resource Description Framework (RDF) zajišťuje přiřazování metadat, tj. dat o datech, ke zdrojům. Zapisuje se ve formě XML. Specifikaci udržuje World Wide Web Consortium (dále WWWC) a můžeme ji najít v [13]. Popis můžeme též nalézt v [34] a [35].

Cílem je popsat metadata ve formě subjekt-predikát-objekt, nazývané v terminologii RDF *trojice* (triple). Subjekt je zdroj, který popisujeme, predikát potom vlastnost nebo aspekt subjektu. Ve většině případů popisuje predikát vztah subjektu k objektu. Objektem je předmět vztahu nebo hodnota vlastnosti.

Zápis RDF výrazů definuje orientovaný graf. V něm lze vyhledávat pomocí různých jazyků. Některé z nich jsou založeny na XML – z těchto jmenujme DQL a RDFQ. Další jsou odvozeny od SQL – RDQ, RDQL, SeRQL a SPARQL. Poslední jmenovaný prochází v současné době procesem standardizace WWWC (viz [36]).

Blíže je tato technologie ukázána přímo v kontextu Fedory na modulu pro indexaci zdrojů (sekce 4.13 na straně 27).

Kapitola 4

Digitální repozitář Fedora

Název digitální knihovny Fedora je zkratkou z anglického *Flexible and Extensible Digital Object Repository Architecture*. Tato charakteristika dobře vystihuje poslání a funkce tohoto systému. Od současné verze 2.1 je šířena pod licencí Educational Community License 1.0 (ECL, viz [4]), ke které přešla od Mozilla Public License 1.1 (viz [5]).

K dispozici je server, jenž poskytuje základní operace s digitálními objekty. Mezi nejdůležitější z nich patří vkládání objektů do repozitáře, správa jednotlivých verzí datových proudů či informací o nich, získání uložených dat a diseminací, vztahy mezi objekty, zajištění řízení přístupu a vyhledávání. Knihovna je však i tak dostatečně obecná a lze v ní bez problémů uchovávat data nejrozmanitějších formátů.

Dále je k dispozici jednoduchý klient. Je to aplikace napsaná v programovacím jazyce Java, pomocí níž je možné vkládat a vytvářet nové datové digitální objekty, definice a mechanismy chování. Lze též vyhledávat a prohlížet údaje o skladovaných objektech, ale prostředí není určené pro koncového uživatele. Tento nástroj naopak ocení správci, neboť jim umožňuje manipulovat se všemi daty a získat o nich přehled.

V současné době je vyvíjen uživatelsky přívětivější klient Fire, což je opět zkratka, tentokrát z Fedora Institutional Repository Environment, a nástroje pro procházení grafů závislostí RDF. K dispozici by měly programy být do příštího roku.

4.1 Historie

Projekt Fedora byl zahájen v roce 1997 na Cornell University. Dotovaly jej organizace DARPA a NSF. Prvotní referenční implementace stavěla na technologii CORBA. První praktickou aplikací byl prototyp vyvinutý v University of Virginia v roce 1999. Ten byl přepracován pro webové prostředí a postaven na relační databázi pro zvýšení výkonu. Testovala se škálovatelnost na deseti miliónech objektů.

Vývoj v plném rosahu podpořilo získání dotace od Andrew W. Mellon Foundation v roce 2002. Úkolem bylo vytvořit univerzální flexibilní digitální knihovnu postavenou na technologii webových služeb a XML. Verze 1.0 vyšla v květnu 2003.

Nyní probíhá druhá fáze vývoje, zahájená v roce 2004 získáním dalšího, tentokrát tříletého grantu znovu od Andrew W. Mellon Foundation.

4.2 Přednosti a nevýhody Fedory

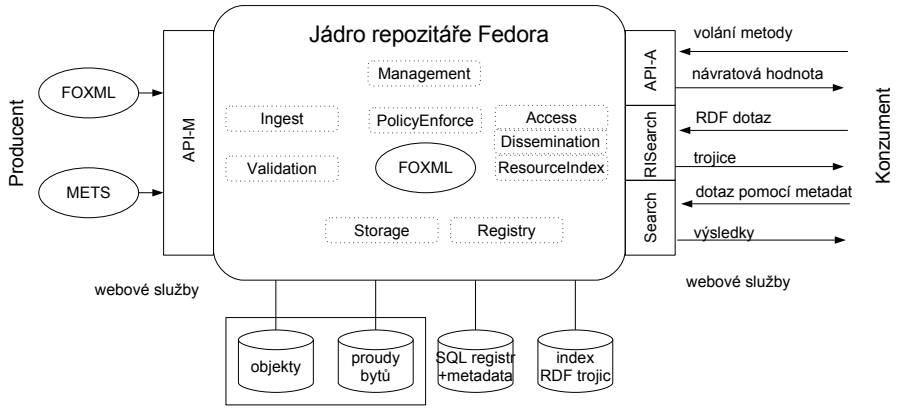
Ze způsobu vývoje vyplývají i výhody Fedory. Mezi ně lze zařadit fakt, že jde o systém s volně šiřitelnými zdrojovými kódy. Programátor v nich může provést přizpůsobení konkrétním podmínkám a požadavkům, jež jsou důležité pro komunitu koncových uživatelů. Výběr programovacího jazyka Java umožňuje provoz ve všech rozšířených operačních systémech a automatické vygenerování dokumentace (nástrojem JavaDoc), na niž jsou vývojáři zvyklí. Kvalitu zajišťuje dodržování standardů a používání osvědčených balíků třetích stran. Vývoj pokračuje a dá se předpokládat, že neskončí dříve než v září 2007, kdy vyprší grant na jeho druhou fázi. Nalezené chyby a nedostatky programátoři pravidelně odstraňují a přidávají nové nástroje a moduly. Tohoto procesu se účastní i jiné organizace, které mají zájem o další funkčnost. Se zkušenějšími správci je možné se poradit prostřednictvím e-mailové diskuze.

Další přednosti se týkají samotných vlastností systému. Snahou je udržet jej co nejobecnější a rozšiřitelný, což naznačuje již jeho název. Lze v něm skutečně vytvořit a udržovat téměř libovolnou sbírku dat. Nevýhodou tohoto přístupu se jeví to, že před nasazením je nutné provést přizpůsobení a nastavení konkrétnímu prostředí. Výkon není žádnému z nich předem přizpůsoben. Proto je třeba zvážit nasazení Fedory podle požadavků, které na výsledný systém klademe. V určitých případech výběr specializovaného softwaru zvýší výkon na stejném hardwarovém zařízení a sníží pracnost implementace klienta a nastavení aplikace.

4.3 Architektura systému

Schéma 4.1 ukazuje architekturu jádra Fedory s rozhraními, jež poskytuje prostřednictvím webových služeb.

Producent má k dispozici správcovské funkce API-M (viz 5.2 na straně 41). Pomocí nich může vkládat objekty do repozitáře ve formátu FOXML (viz 4.11 na straně 21) či METS. Fedora uloží informace o objektech a jejich



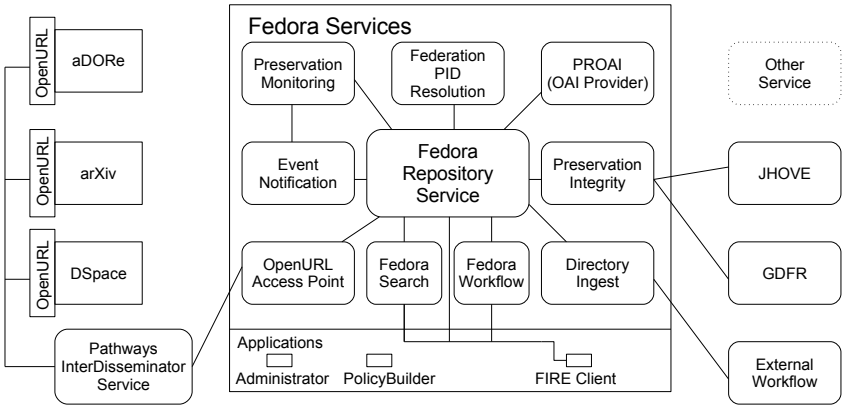
Obrázek 4.1: Architektura jádra Fedory (převzato z [27]).

datové proudy, které jsou pod její správou (viz 4.8 na straně 18), v souborovém systému. Metadata navíc vkládá do registru realizovaného prostřednictvím relační databáze. Trojice RDF jsou uchovávány a indexovány modulem RISearch (viz 4.13 na straně 27), je-li zapnutý.

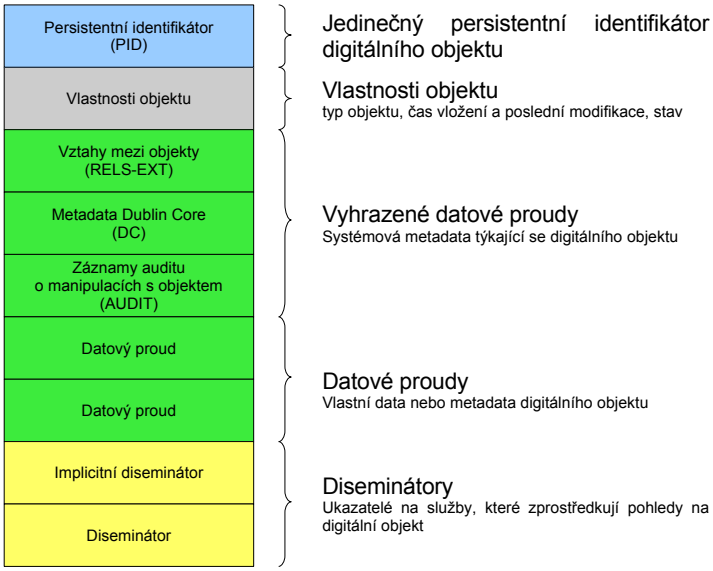
Uvnitř jádra jsou obsaženy moduly zajišťující skladování digitálních objektů, jejich datových proudů typů M a X (viz 4.8 na straně 18), správu dat v registru a indexu trojic RDF a vyhledávání podle nich, zajištění bezpečnosti prostřednictvím přístupových práv a provádění diseminací.

Konzument přistupuje k datům, metadatům a diseminacím pomocí rozhraní webových služeb API-A (viz 5.1 na straně 39), procházení v grafu závislostí RDF zprostředkuje přístup k modulu RISearch. Vyhledávání digitálních objektů je možné zatím pouze podle jejich metadat, ne podle obsahu datových proudů.

Obrázek 4.2 ukazuje zamýšlené uspořádání služeb, jež jsou vyvíjeny v průběhu let 2005–2007. Jedná se o samostatné služby, které mohou spolupracovat s jádrem systému a mezi sebou navzájem. V současné době je k dispozici modul OAI Provider, pomocí něhož mohou být sklízena metadata systémem vně digitální knihovny (viz 3.3 na straně 8), a Directory Ingest pro vkládání více digitálních objektů najednou ze speciálního komprimovaného zip nebo jar souboru. V příští verzi 2.2, jež by měla být vydána ve třetím čtvrtletí roku 2006, by se mělo objevit vyhledávání podle obsahů datových proudů (Search), o němž lze získat bližší informace na [41].



Obrázek 4.2: Plánované služby Fedory (převzato z [27].)



Obrázek 4.3: Model digitálního objektu ve Fedoře (přepracováno podle [27])

4.4 Model digitálních objektů

Obrázek 4.3 znázorňuje základní komponenty modelu digitálního objektu v repozitáři Fedora. Těmi jsou:

PID – persistentní jedinečný identifikátor (viz sekce 4.6 na straně 17)

Vlastnosti objektu – množina systémem definovaných popisných vlastností objektů, které jsou nezbytné pro správu a vyhledání objektu v repozitáři (viz sekce 4.7 na straně 18)

Datové proudy – data, jejichž typ je identifikován prostřednictvím MIME

Každý digitální objekt má jeden nebo více datových proudů. Ty jsou buď přímo spravovány repozitářem, nebo je na ně udržována URL adresa. Každý objekt má povinně proud obsahující metadata Dublin Core. Tématu datových proudů bude věnována sekce 4.8 na straně 18.

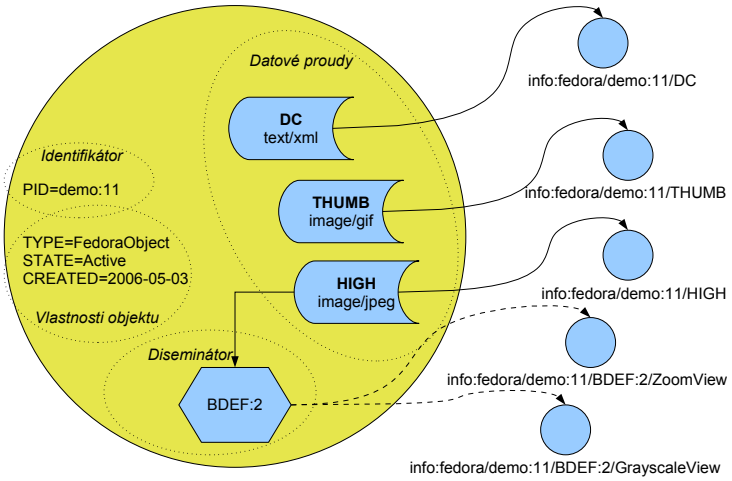
Diseminátory – přiřazení externích služeb k objektům (viz sekce 4.9 na straně 19)

K identifikaci digitálního objektu může být použito označení „`info`“ URI, jmenný prostor „`fedora`“ je registrován (viz [9]). Označení je tedy ve tvaru `info:fedora/`*pid*, kam je nutné za *pid* dosadit PID objektu (viz sekce 4.6 na straně 17).

Na obrázku 4.4 vidíme jiný pohled na digitální objekt. V tomto případě máme objekt s PID demo:11 a podle hodnoty vlastnosti TYPE se jedná o datový digitální objekt (viz sekce 4.5), jenž reprezentuje obrázek s náhledem. Obsahuje datové proudy DC (v něm jsou zakódována v XML metadata Dublin Core), THUMB (náhled ve formátu GIF) a HIGH (JPEG soubor v plném rozlišení). Jejich data můžeme získat pomocí *přímých reprezentací* (ve schématu jsou označeny spojitými šipkami). Dále si můžeme všimnout diseminátoru, který obsahuje popis služeb nad digitálním objektem a způsob jejich volání. Použitím získáme *nepřímé reprezentace* (naznačeny čárkovanými šipkami). Ty nejsou nikde v repozitáři uloženy, ale jsou získány uvedenými službami. V našem příkladu umožňují převod do stupňů šedi a přiblížení.

4.5 Typy digitálních objektů

Fedora rozlišuje 3 typy digitálních objektů, přestože jsou všechny uloženy stejně, podle výše popsaného modelu. Rozdíly jsou pouze ve způsobu, jakým systém nakládá s jejich údaji.



Obrázek 4.4: Další pohled na digitální objekt Fedory (přepracováno podle [27])

Datové digitální objekty Jsou určeny pro uchovávání digitálního obsahu. Mohou obsahovat obrázky, texty, hudbu, video a mnoho dalších dat. S daty lze pracovat pomocí diseminátorů, které jsou popsány následujícími dvěma typy objektů.

Definice chování objektu Tyto objekty definují operace, které je možné vykonávat na objektech, k nimž jsou přiřazeny. Jde ale pouze o abstraktní popis volání a typy parametrů. Pro dokonalejší pochopení je možné představit si je jako rozhraní v Javě.

Mechanismus chování objektu Zde jsou popsány přímo vazby operací, uvedených v definici chování, na konkrétní služby. Každý objekt tohoto typu je svázán s definicí chování a určuje konkrétní implementaci.

Nejdůležitějšími metadaty jsou informace o způsobu volání daných služeb. Uloženy jsou ve formě WSDL. Podle nich je zkonstruován požadavek a pomocí webové služby zavolána určitá metoda.

4.6 PID

PID (Persistent IDentifier) je vnitřní označení digitálního objektu uloženého v repozitáři Fedora. Tvoří jej předpona jmenného prostoru, která je složena

ze znaků a-z, A-Z, 0-9, pomlčky a tečky, a jednoduchý řetězcový identifikátor, jenž kromě znaků anglické abecedy může obsahovat též číslice, pomlčky, tečky, vlnovky, podtržítka a zápisy pomocí tvaru %XY, kde za X a Y dosadíme šestnáctkové číslice. Obě části jsou odděleny dvojtečkou a každá z nich má alespoň 1 znak. Záleží zde také na velikosti písmen. Maximální délka celého PID je 64 znaků.

Jako příklady, jak mohou vypadat identifikátory, mohu uvést `image:4`, `demo:SmileyStuff`, `fi-muni-cz:243` nebo `fi.muni.cz:%5C_1354`.

4.7 Vlastnosti objektu

Vlastnostmi objektu rozumíme charakteristiky, které se týkají objektu jako celku. Nelze od nich vytvářet různé verze.

Každý objekt má definovány *typ objektu* a jeho *stav*. Typ udává, zda se jedná o datový digitální objekt, definici chování, anebo mechanismus chování (viz sekce 4.5 na straně 16). Stavem rozlišujeme, jde-li o aktivní, neaktivní, nebo smazaný objekt.

Nepovinnými vlastnostmi jsou *pojmenování* a *identifikátor modelu objektu*. Obě z nich jsou uživatelem definované řetězce. Druhá z jmenovaných charakteristik určuje typ objektu z pohledu klienta. Označuje, které datové proudy a diseminátory jsou přítomny, aby přistupující aplikace věděly, jak s daným objektem nakládat.

Posledními Fedorou definovanými vlastnostmi, které spravuje sám server, jsou *čas vytvoření* a *čas poslední změny*. Jejich formát je popsán v sekci 4.10 na straně 21.

Všechny výše uvedené vlastnosti a jejich hodnoty jsou serverem indexovány a lze podle nich vyhledávat pomocí funkce `findObjects`, jež je přístupná prostřednictvím webové služby.

Kromě nich však mohou být definovány i *rozšiřující vlastnosti*. Jejich význam a použití je ponecháno přímo na uživateli.

Je-li zapnutá indexace pomocí modulu RISearch (viz sekce 4.13 na straně 27), jsou všechny vlastnosti, tentokrát i včetně těch rozšiřujících, indexovány a vyhledatelné pomocí jeho rozhraní.

4.8 Datové proudy

Datový proud obsahuje digitální data, jako jsou například obrázky, texty a dokumenty v PDF. Jejich typ je určen pomocí MIME. Uložena může být i historie změn, která je identifikována pomocí času vložení. Metadata všech

proudů jsou umístěna přímo ve FOXML (Fedora Object XML), což je dokument, v němž jsou uchovávána všechna systémem používaná metadata o digitálním objektu.

Představme si případ, že chceme pomocí digitálního objektu reprezentovat článek ve Zpravodaji ÚVT. Potřebujeme mít k dispozici dokument ve formě HTML, PDF, PS a zdrojového kódu pro TEX. Vytvoříme tedy objekt, který bude obsahovat proudy DC (text/xml) typu X, HTML (text/html), PDF (application/pdf), PS (application/postscript) a TeX (text/plain) typu M. Klient potom podle způsobu zobrazení vybere nejvhodnější z nich.

Podle umístění vlastních dat rozlišuje Fedora následující 4 typy datových proudů.

Interní XML metadata (Internal XML metadata, dále X)

Tento typ lze použít jenom pro validní XML dokumenty, protože obsah je uložen přímo jako součást FOXML. Takto je veden i proud DC, který obsahuje povinně každý objekt a v němž jsou umístěna ve formě XML dokumentu metadata Dublin Core.

Spravovaný obsah (Managed Content, dále M)

Data jsou uložena a spravována přímo serverem Fedory.

Externě odkazovaný obsah (External Referenced Content, dále E)

Repozitář si uchovává pouze metadata datového proudu, která obsahují URL adresu, z níž je v případě použití stažen serverem Fedora a poskytnut uživateli.

Přesměrovaný (Redirect, dále R)

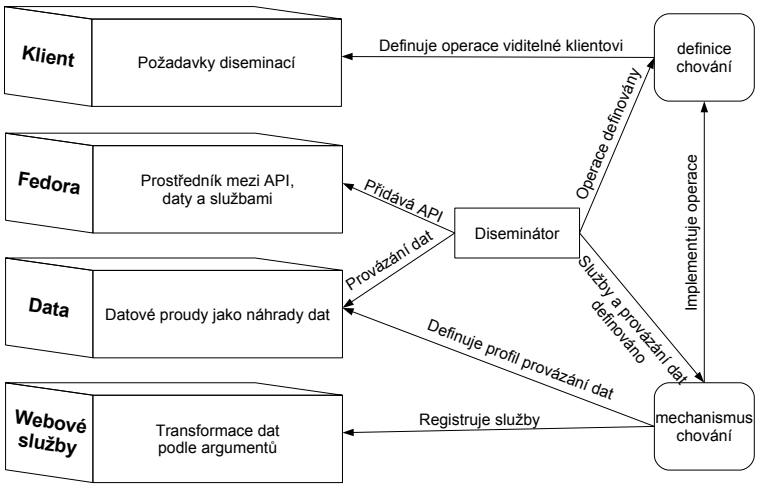
Server si v tomto případě, podobně jako v předchozím, uchovává metadata s uloženou adresou. Rozdíl je ve způsobu získání dat. Tentokrát však bude uživatel při použití přímo přesměrován na udané umístění, data nebudou stahována přes server Fedory.

Identifikátor datového proudu je řetězec skládající se z XML NCName znaků (viz [37]).

4.9 Diseminátory

Vraťme se nyní k výše popsanému příkladu s článkem ve Zpravodaji ÚVT. Představme si, že bychom nechtěli uchovávat v repozitáři příspěvky ve formátech PDF a PS, ale v případě potřeby bychom je nechali přeložit z přítomných zdrojových kódů pro TEX. Tento postup, ačkoliv není vhodný

z hlediska výkonu, lze uskutečnit pomocí diseminátorů. Vytvoříme webovou službu, která bude překládat dodané dokumenty a vracet je převedené do PDF, respektive PS, a spustíme ji na J2EE (Java 2 Enterprise Edition) serveru. V repozitáři vytvoříme definici chování (přiřadíme jí PID `uvt:convertBDef`), která bude obsahovat funkce `getPDF` a `getPS`. Jako parametr `src` převezmou dokument typu text/plain a vrátí data ve formátu *application/pdf*, resp. *application/postscript*. Nyní musíme tyto abstraktní definice spojit s běžící webovou službou. K tomu slouží mechanismus chování. Označme jej PID `uvt:convertBMech`. Ten bude implementovat služby definice `uvt:convertBDef` a volání metody převede na požadavek webové služby. Parametr `src` sváže s datovým proudem `TeX`.



Obrázek 4.5: Schéma fungování diseminátorů (převzato z [27])

Popišme si fungování diseminátorů obecně, jak je znázorňuje schéma 4.5. Diseminátor svazuje k sobě definici chování, jež určuje, jaké funkce s jakými parametry budou definovány, s mechanismem chování. Ten ukazuje na konkrétní implementaci a způsob volání. Fedora funguje jako prostředník, který vyhledá podle požadavku správný diseminátor, podle deklarovaného svázání datových proudů s parametry funkce dosadí správné hodnoty a zavolá příslušnou webovou službu. Na té je již samotné provedení uvedené transformace.

4.10 Časové známky

Časové známky jsou použity jednak k označení časového okamžiku, jako je tomu u času vytvoření nebo poslední modifikace objektu, tak také pro verzi datového proudu, která v uvedenou dobu byla aktuální. Rozlišení této jednotky je milisekunda. Zapisujeme ji *YYYY*-*MM*-*DD*T*HH*:*mm*:*SS*.*XXX*Z, kam dosadíme za *YYYY*, *MM*, *DD*, *HH*, *mm*, *SS* a *XXX* postupně rok, měsíc, den, hodinu, minutu, sekundu a počet milisekund. Každá z předchozích hodnot se doplní levostrannými nulami na požadovaný počet znaků. Jako názorný příklad uvádím `2006-04-14T00:33:33.132Z`.

4.11 Fedora Object XML (FOXML)

Fedora Object XML (dále jen FOXML) je XML dokument popisující digitální objekt. V tomto formátu jsou data uchovávána přímo repozitářem, ačkoliv pro vkládání a export je možné použít také formát Fedora METS a v dalších verzích je počítáno též s METS 1.4 a MPEG21/DIDL.

Kořenový element se nazývá digitalObject[1]. Soubor lze pomyslně rozdělit na 3 různé části. V první jde o definici vlastností objektu, ve druhé pak o datové proudy a v poslední jsou deklarovány diseminátory. Strukturu zanoření elementů lze zjednodušeně vyjádřit takto:

```
<digitalObject PID="jedinečný identifikátor">
    <!-- vlastnosti objektu -->
    <objectProperties>
        <property/>
        <property/>
        ...
        <extproperty/>
        ...
    </objectProperties>
    <!-- jeden nebo více datových proudů -->
    <datastream>
        <datastreamVersion/>
        <datastreamVersion/>
        ...
    </datastream>
    <!-- žádný nebo více diseminátorů -->
```

[1] Implicitní jmenný prostor v této sekci je `info:fedora/fedora-system:def/foxml#`.

```
    <disseminator>
        <disseminatorVersion/>
        <disseminatorVersion/>
        ...
    </disseminator>
</digitalObject>
```

Jednotlivé části budou nyní popsány blíže.

4.11.1 Vlastnosti objektu

Každý objekt musí mít při vkládání definovány vlastnosti *typ objektu* (`type`)[2] a *stav* (`state`)[3]. Typ nabývá právě jedné z hodnot `FedoraObject` pro datový digitální objekt, `FedoraBDefObject` pro definici chování, anebo `FedoraBMechObject` pro mechanismus chování. Stav bývá označován `A` (aktivní), `I` (neaktivní), nebo `D` (smazáno).

Dále můžeme upřesnit nepovinné vlastnosti *pojmenování* (`label`)[4] a *identifikátor modelu objektu* (`contentModel`)[5]. Obě z nich jsou uživatelem definované řetězce.

Poslední Fedorou definované vlastnosti by při vkládání nemělo obsahovat FOXML, doplní je sám server. Jsou jimi *čas vytvoření* (`createdDate`)[6] a *čas poslední změny* (`lastModifiedDate`)[7], jejich formát je popsán v sekci 4.10 na straně 21.

Předchozí vlastnosti jsou zapsány jako element ve tvaru

`<foxml:property NAME="..." VALUE="..."/>.`

Jako hodnoty atributů se dosadí příslušné údaje.

Způsob zápisu rozšiřujících vlastností je obdobný:

`<foxml:extproperty NAME="..." VALUE="..."/>.`

Nyní uvádím jednoduchý příklad použití.

```
<objectProperties>
  <property
    NAME="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
```

[2] Jméno je `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`.
[3] Jméno je `info:fedora/fedora-system:def/model#state`.
[4] Jméno je `info:fedora/fedora-system:def/model#label`.
[5] Jméno je `info:fedora/fedora-system:def/model#contentModel`.
[6] Jméno je `info:fedora/fedora-system:def/model#createdDate`.
[7] Jméno je `info:fedora/fedora-system:def/view#lastModifiedDate`.

```
    VALUE="FedoraObject"/>
  <property
    NAME="info:fedora/fedora-system:def/model#state"
    VALUE="A"/>
  <property
    NAME="info:fedora/fedora-system:def/model#label"
    VALUE="Digitální repozitář Fedora"/>
  <property
    NAME="info:fedora/fedora-system:def/model#contentModel"
    VALUE="FI_THESIS"/>
  <extproperty
    NAME="http://fi.muni.cz/~xnovot10/fi-thesis.xsd#year"
    VALUE="2006"/>
</objectProperties>
```

4.11.2 Datové proudy

Typ X

Datové proudy typu X obsahují svá data přímo uvnitř FOXML. Ta jsou uzavřená v elementu `xmlContent`.

Následuje ukázka uložení metadat DC.

```
<datastream ID="DC" STATE="A" CONTROL_GROUP="X"
  VERSIONABLE="true">
  <datastreamVersion ID="DC.0" MIMETYPE="text/xml"
    LABEL="Default Dublin Core Record">
    <xmlContent>
      <oai_dc:dc
        xmlns:oai_dc=
          "http://www.openarchives.org/OAI/2.0/oai_dc/"
          xmlns:dc="http://purl.org/dc/elements/1.1/">
        <dc:title>Digitální repozitář Fedora</dc:title>
        <dc:title>Fedora</dc:title>
        <dc:creator>Stanislav Novotný</dc:creator>
        <dc:subject>Diplomové práce</dc:subject>
        <dc:description>
          Diplomka na téma digitální knihovna Fedora
        </dc:description>
        <dc:identifier>fi.muni.cz:thesis-243</dc:identifier>
      </oai_dc:dc>
    </xmlContent>
  </datastreamVersion>
</datastream>
```

Typy M, R a E

Datové proudy typů M, R a E mají shodnou strukturu. O jaký typ se jedná v konkrétním případě, je upřesněno pomocí atributu `CONTROL_GROUP`, který nabývá příslušné hodnoty. Atribut `TYPE` elementu `contentLocation` může obsahovat hodnotu `URL` nebo `INTERNAL_ID` pro proud typu M, u něhož je správa svěřena repozitáři Fedory. Význam ostatních atributů není třeba podrobněji zmiňovat, je zřejmý z jejich názvů.

V uvedeném příkladu je ukázán typ M, který obsahuje 2 verze souboru ve formátu PDF.

```
<datastream CONTROL_GROUP="M" ID="PDF"
  STATE="A" VERSIONABLE="true">
  <datastreamVersion ID="PDF.0" MIMETYPE="application/pdf"
    LABEL="Thesis in PDF"
    CREATED="2006-02-10T12:12:12.012Z">
    <contentLocation
      REF="http://www.fi.muni.cz/~xnovot10/dp.pdf"
      TYPE="URL"/>
  </datastreamVersion>
  <datastreamVersion ID="PDF.1" MIMETYPE="application/pdf"
    LABEL="New version of thesis in PDF"
    CREATED="2006-05-12T23:56:57.587Z">
    <contentLocation
      REF="http://www.fi.muni.cz/~xnovot10/dp_latest.pdf"
      TYPE="URL"/>
  </datastreamVersion>
</datastream>
```

4.11.3 Diseminátory

V poslední části FOXML souboru se nachází seznam diseminátorů, jež je možno použít na daný objekt a jež tím určují možná chování. Deklaraci, které funkce s jakými vstupy diseminátor obsahuje, najdeme v definici chování. To je digitální objekt, jehož PID je uvedeno v `BDEF_CONTRACT_PID`. Podobně jako u datových proudů i zde je možné pracovat s více verzemi. Každá z nich mapuje definici chování na konkrétní realizaci, mechanismus chování (viz PID v atributu `BMECH_SERVICE_PID`). Dále je nutné definovat provázání datových proudů (atribut `DATASTREAM_ID`) na parametry chování (atribut `KEY`).

Následuje krátký příklad, jak by mohlo vypadat navázání diseminátoru, který má za úkol převést zdrojový text pro LaTeXdo PDF, na datový digitální objekt.

```
<disseminator ID="TO_PDF" STATE="A"
  BDEF_CONTRACT_PID="bdef:TeX2PDF" VERSIONABLE="true">
  <disseminatorVersion ID="TO_PDF.0"
    BMECH_SERVICE_PID="bmech:TeX2PDF"
    LABEL="Compile TeX source with pdfcslatex."
    CREATED="2006-04-10T13:45:23.082Z">
    <serviceInputMap>
      <datastreamBinding DATASTREAM_ID="TeX" ORDER="0"
        LABEL="TeX source to be compiled." KEY="TeX_SRC"/>
    </serviceInputMap>
  </disseminatorVersion>
</disseminator>
```

4.12 Datové proudy se zvláštním významem

4.12.1 DC

Datový proud s označením DC obsahuje ve formě XML zakódované údaje *nekvalifikovaných Dublin Core metadat* (viz sekce 3.1 na straně 7). Jako jediný je *povinně přítomný* v každém digitálním objektu. Pokud bychom chtěli používat kvalifikátory, museli bychom si vytvořit vlastní datový proud a v nich tyto uchovávat, jelikož Fedora zatím jejich podporu neobsahuje. Kódování je velmi jednoduché a intuitivní, příklad je uveden v podsekci 4.11.2 na straně 23. Každou z vlastností lze libovolněkrát opakovat. Identifikátor (vlastnost `identifier`) Fedora po vložení do repozitáře automaticky nastaví na přidělené PID objektu.

Všechny položky tohoto proudu jsou indexovány a lze podle nich efektivně vyhledávat. Taktéž jsou indexovány modulem RISearch, je-li zapnutý.

4.12.2 RELS-EXT

Pomocí tohoto datového proudu, jehož daty je RDF soubor, se vkládají vztahy k objektu, v němž se nachází. Ten je subjektem vztahu a jeho PID musí být obsaženo v atributu `about` elementu `Description`. Uvnitř vidíme definice relací. Ty jsou buď vztaženy k jinému objektu, nebo mohou obsahovat libovolnou textovou hodnotu.

Z následující ukázky lze vyčíst, že se datový proud nalézá v digitálním objektu s PID `demo:99` (atribut `about` elementu `Description`) a obsahuje 3 vztahy. První je typu `isMemberOfCollection` přímo definovaného Fedorou[8] a jeho cílem je `demo:c1`. Druhý určuje vztah k celku (objekt s PID

[8] Jmenný prostor je `info:fedora/fedora-system:def/relations-external#`.

mystuff:100) a poslední definuje jméno vlastníka.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:fedora=
     "info:fedora/fedora-system:def/relations-external#"
 xmlns:myns="http://www.nsdl.org/ontologies/relationships#">
 <rdf:Description rdf:about="info:fedora/demo:99">
    <fedora:isMemberOfCollection
        rdf:resource="info:fedora/demo:c1"/>
    <myns:isPartOf rdf:resource="info:fedora/mystuff:100"/>
    <myns:owner>Jane Doe</myns:owner>
 </rdf:Description>
</rdf:RDF>
```

Při vložení, změně či odstranění tohoto proudu dojde k promítnutí nových údajů do indexu uvedených vztahů, je-li aktivní modul RISearch. Pomocí něho je možné číst vztahy a vyhledávat podle nich požadované údaje (viz sekce 4.13 na straně 27).

4.12.3 AUDIT

Zde spravuje systém údaje o provedených změnách nad tímto objektem.

Pro názornost uvádím ukázku informace, která vznikne po změně datového proudu DC.

```
<auditTrail xmlns="info:fedora/fedora-system:def/audit#">
  <record ID="AUDREC1">
    <process type="Fedora API-M"/>
    <action>modifyDatastreamByValue</action>
    <componentID>DC</componentID>
    <responsibility>fedoraAdmin</responsibility>
    <date>2006-04-14T10:08:03.330Z</date>
    <justification>Změna titulu DC.</justification>
  <record>
</auditTrail>
```

4.12.4 POLICY

V tomto datovém proudu mohou být umístěny XACML politiky, které se uplatní při manipulaci s digitálním objektem. Nemusí mít specifikovány PID objektu, ke kterému se vztahují, Fedora za něj vždy dosadí ten objekt, v němž se nachází. Výhodou použití těchto lokálních politik na rozdíl

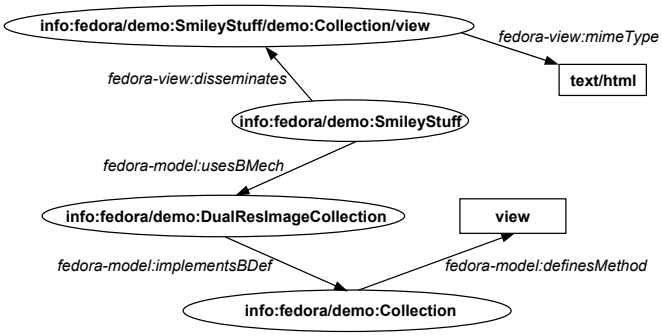
od globálních je to, že nejsou načteny při startu serveru a neplýtvají tím jeho zdroji, ale použijí se až při přístupu k objektu.

Realizovat lze dvěma možnostmi:

- **jako datový proud typu X nebo M** – v tomto případě se nachází zcela pod správou repozitáře, je tedy vždy dostupný, při exportu a přesunu jej obsahuje rodičovský objekt
- **jako datový proud typu E nebo R** – výhodou je, že více digitálních objektů může odkazovat na stejnou adresu a sdílet tak stejnou politiku přístupu

4.13 Index zdrojů

V posledních verzích obsahuje Fedora index zdrojů (modul RISearch), pomocí něhož lze mimo jiné uchovávat vztahy mezi objekty, např. vztah část-celek (u objektů článek-časopis). Realizace je provedena pomocí RDF (viz sekce 3.6 na straně 10) systémem Kowari.



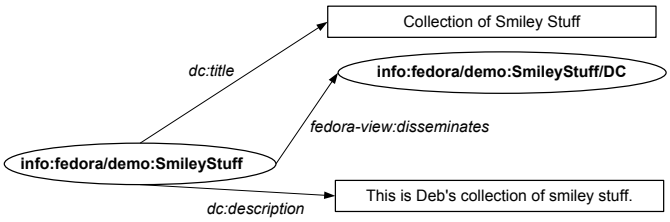
Obrázek 4.6: Část objektového modelu jako orientovaného grafu (přepracováno podle [27])

Na objektový model Fedory lze pohlížet jako na orientovaný graf, jenž se skládá z *vnitřních vztahů* mezi uzly digitálních objektů a uzly jejich diseminací a z *vnějších vztahů* mezi uzly digitálních objektů. Index zdrojů udržuje tento graf a umožňuje provádět nad ním dotazy. Aktualizace je provedena automaticky při každém přidávání a při všech změnách digitálních objektů.

Na obrázku 4.6 je znázorněna část objektového modelu (uvedené digitální objekty jsou vybrány z ukázkových dat, která jsou obsažena v klientovi Fedory). Datový objekt s PID `demo:SmileyStuff` používá mechanismus

chování (`demo:dualResImageCollection`), který implementuje definici chování (`demo:Collection`), v níž je deklarována metoda `view`. Zároveň je ukázáno, že diseminací pomocí této metody vzniknou data typu *text/html*.

Automaticky jsou též indexována metadata Dublin Core, uvedená v datovém proudu DC, jak ukazuje schéma na obrázku 4.7.



Obrázek 4.7: Indexace metadat DC (přepracováno podle [27])

Vztahy jsou uchovávány ve formě `<subjekt> <vztah> <cíl>`. Vlastníkem relace je subjekt. Jeho vztahy jsou definovány ve zvláštním datovém proudu RELS-EXT (viz 4.12.2 na straně 25) ve formátu RDF. Cílem vztahu nemusí být jen konkrétní datový objekt, ale může jím být libovolný řetězec. Tím lze subjektu přidávat hodnoty určitých vlastností, podle nichž lze modulem RISearch vyhledávat.

Pokládat dotazy lze prostřednictvím rozhraní, jež je přístupné na adrese serveru Fedory pod `/fedora/risearch`. Lze k němu přistupovat běžným WWW prohlížečem.

Jako první způsob se nabízí *vyhledávání dvojic*. Vrácenými položkami jsou pojmenované hodnoty. Z dotazovacích jazyků jsou podporovány iTQL a RDQL.

Uvádím ukázku zápisu dotazu v jazyce iTQL, který vychází z SQL. Vrácenými hodnotami jsou PID všech definic chování (pojmenováno `object`) a čas jejich poslední modifikace (pojmenováno `lastModified`).

```
select $object $lastModified from <#ri>
where $object <rdf:type> <fedora-model:FedoraBDefObject>
and $object <fedora-view:lastModifiedDate> $lastModified
```

Formu odpovědi lze vybrat z následujících možností. U každé z nich je zaznamenán výsledek výše uvedeného příkladu.

CSV – čárkami oddělený seznam hodnot

```
"object","lastModified"
info:fedora/demo:8,2006-02-25T18:56:40.85
info:fedora/demo:1,2006-02-25T18:56:06.983
```

```
info:fedora/demo:Collection,2006-02-25T18:56:03.378
```

Simple – snadno čitelný formát, který obsahuje informace o typu, jsou-li přítomny

```
object       : <info:fedora/demo:8>
lastModified : ”2006-02-25T18:56:40.85”
            ^^http://www.w3.org/2001/XMLSchema#dateTime

object       : <info:fedora/demo:1>
lastModified : ”2006-02-25T18:56:06.983”
            ^^http://www.w3.org/2001/XMLSchema#dateTime

object       : <info:fedora/demo:Collection>
lastModified : ”2006-02-25T18:56:03.378”
            ^^http://www.w3.org/2001/XMLSchema#dateTime
```

Sparql – XML formát navržený RDF Data Access Working Group

```
<?xml version=”1.0” encoding=”UTF-8”?>
<sparql xmlns=
    ”http://www.w3.org/2001/sw/DataAccess/rf1/result”>
  <head>
    <variable name=”object”/>
    <variable name=”lastModified”/>
  </head>
  <results>
    <result>
      <object uri=”info:fedora/demo:8”/>
      <lastModified datatype=
          ”http://www.w3.org/2001/XMLSchema#dateTime”>
        2006-02-25T18:56:40.85
      </lastModified>
    </result>
    <result>
      <object uri=”info:fedora/demo:1”/>
      <lastModified datatype=
          ”http://www.w3.org/2001/XMLSchema#dateTime”>
        2006-02-25T18:56:06.983
      </lastModified>
    </result>
    <result>
      <object uri=”info:fedora/demo:Collection”/>
      <lastModified datatype=
          ”http://www.w3.org/2001/XMLSchema#dateTime”>
        2006-02-25T18:56:03.378
```

```
        </lastModified>
      </result>
    </results>
  </sparql>
```

TSV – hodnoty oddělené tabulátorem

```
object                        lastModified
info:fedora/demo:8            2006-02-25T18:56:40.85
info:fedora/demo:1            2006-02-25T18:56:06.983
info:fedora/demo:Collection 2006-02-25T18:56:03.378
```

Druhým způsobem máme možnost *vyhledávat trojice*. Výsledkem jsou trojice ve smyslu RDF (subjekt-predikát-objekt). Rozhraní se podobá vyhledávání dvojic. Liší se jinými formáty odpovědí (N-Triples, Notation 3, RDF/XML, Turtle), dotazovacími jazyky jsou iTQL a RDQL se šablonou převádějící dvojice na trojice.

Příklad použití iTQL ukazuje způsob vyhledání všech vztahů mezi datovými digitálními objekty.

Dotaz

```
select $a $r $b from <#ri>
where $a <rdf:type> <fedora-model:FedoraObject>
and $a $r $b
and $b <rdf:type> <fedora-model:FedoraObject>
```

Šablona

```
$a $r $b
```

Navíc je přidán velmi jednoduchý jazyk SPO. V něm se zapisuje dotaz ve formě *subjekt predikát objekt*, přičemž každá z položek může být nahrazena hvězdičkou `*`, znamenající libovolnou hodnotu. Dotaz `* * *` najde všechny trojice, `* * <info:fedora/fi-muni-cz:1>` pak ty, jejichž objektem je objekt s PID `fi-muni-cz:1`, a `<info:fedora/demo:1> * <info:fedora/demo:2>` vrátí vztahy, jejichž subjektem je `demo:1` a objektem `demo:2`.

4.14 Uživatelé

Fedora 2.0 nabízela minimální možnosti nastavení bezpečnosti přístupu k serveru – byl definován jediný superuživatel (nastaven spolu s heslem ve

`fedora.fcfg`), přístup k API-A[9] byl neomezený, k API-M[10] byl povolen prostřednictvím IP adresy pouze lokálnímu stroji a nebylo možné použít SSL.

Od verze 2.1 je podporována autentikace, SSL a autorizace pomocí XACML politik. Distribuce obsahuje 4 základní konfigurace bezpečnosti:

- **ssl-authenticate-apim**
  - API-M – základní autentikace a SSL, volání omezeno přes IP na lokální stroj
  - API-A – neomezeno, volný přístup
  - minimální zabezpečení koncových služeb

  Určeno pro repozitáře, které chtějí mít pod kontrolou a zabezpečený přístup k administraci a údržbě systému, ale volný přístup k jejich obsahu.

- **ssl-authenticate-all**
  - API-M – základní autentikace a SSL, volání omezeno přes IP na lokální stroj
  - API-A – základní autentikace a SSL
  - komunikace koncových služeb pouze z určených IP adres

  Zde je zabezpečen jak přístup k rozhraní služeb pro údržbu repozitáře, tak také pro získávání dat z něj. Také je možné nastavit způsob komunikace vzdálených koncových služeb s digitální knihovnou. Jelikož pro komunikaci využívají adresu určenou parametrem `fedoraServerHost`, musí jít o plně kvalifikované doménové jméno, nestačí tedy `localhost` nebo `127.0.0.1`.

- **no-ssl-authenticate-apim**
  - API-M – základní autentikace, ale bez SSL, volání omezeno přes IP na lokální stroj
  - API-A – neomezeno, volný přístup
  - minimální zabezpečení koncových služeb

[9] API-A je rozhraní webových služeb určených pro přístup k informacím a objektům repozitáře. Bližší informace jsou uvedeny v podkapitole 5.1 na straně 39.

[10] API-M je označení webové služby zpřístupňujících správcovcké funkce objektů v repozitáři. Více informací je uvedeno v podkapitole 5.2 na straně 41.

Toto zabezpečení se nedoporučuje pro žádné repozitáře, které vyžadují bezpečné prostředí. Jelikož nedochází k autentikaci uživatele při přístupu k API-A, nepoužívají se zde žádné XACML politiky, jež se vztahují přímo k uživateli. Uživatelské jméno a heslo se při přihlašování odesílá v čisté textové podobě, není šifrováno. Toto nastavení simuluje chování Fedory verze 2.0.

- **no-ssl-authenticate-all**
  - API-M – základní autentikace, ale bez SSL, volání omezeno přes IP na lokální stroj
  - API-A – základní autentikace, ale bez SSL
  - komunikace koncových služeb pouze z určených IP adres

  Toto nastavení je určeno pro repozitáře, které správci potřebují udržovat v bezpečném prostředí. Je nabízeno pro případ, že by klienti neumožňovali přístup k serveru přes SSL zabezpečení nebo pro experimenty s nastaveními XACML politik. Uživatelská jména a hesla jsou odesílána v čisté textové podobě, nešifrována. Ani zde, ze stejných důvodů jako u ssl-authenticate-all, nesmí parametr `fedoraServerHost` obsahovat hodnotu `localhost` nebo `127.0.0.1`.

V současné verzi Fedory není žádné z uvedených nastavení implicitní, před prvním spuštěním serveru je nutné skriptu `fedora-setup` předat jako parametr jedno z výše uvedených jmen. Dojde k nakopírování souborů `fedora.fcfg`, `web.xml` a `beSecurity.xml` do adresářů k tomu určených. Nastavení lze kdykoliv, kdy server neběží, stejným způsobem změnit, je třeba ovšem znovu provést úpravu zmíněných souborů.

Uživatelská jména, hesla a role, které správci mohou později využít při tvorbě politik, je možné zapsat do souboru uživatelů pro server Tomcat nebo získat prostřednictvím LDAP, popřípadě možnosti kombinovat.

4.15 Politiky přístupu

Fedora využívá od verze 2.1 novou bezpečnostní architekturu postavenou na XACML politikách (eXtensible Access Control Markup Language). Modul, jenž zajišťuje jejich dodržování, je postaven na Sun XACML engine.

Každá politika obsahuje

- cíl, na který se vztahuje (objekty, datové proudy, operace,. . .)

- jedno nebo více pravidel, jež určují, zda bude operace povolena nebo zakázána

Fedora obsahuje jak globální politiky, které jsou načteny při startu serveru, tak také lokální, které jsou uvedeny přímo jako datový proud objektu, na nějž se vztahují (viz podsekce 4.12.4 na straně 26).

Implicitně jsou po instalaci nastavena globální pravidla, která omezují přístup k funkcím API-M jenom pro administrátora, dovolují volný přístup k API-A, přístup k API-M povolují jen z lokálního stroje,. . .

Autorizaci zajišťuje modul `Authorization`.

Zatím pouze zkušebně byly přidány zástupné politiky (surrogate policies). Ty jsou určeny pro případy, v nichž se serverem komunikuje aplikace nebo služba na vyšší úrovni (webové rozhraní, middleware nebo webové služby). Ta se potom může autentikovat svojí identitou a předat identitu koncového uživatele (přes HTTP hlavičku `From`), který se přihlásil již na této vyšší vrstvě.

Aby příchozí požadavek služby byl povolen, politiky musí vyslovit *explicitní povolení a žádný zákaz*, absence zákazu není dostatečná pro vyjádření souhlasu. Pokud dojde při zpracování k chybě nebo nejsou poskytnuty všechny povinné atributy, výsledkem pravidla je nerozhodnuto. Následující tabulka ukazuje jednotlivá rozhodnutí modulu (ano znamená existenci jednoho a více typů daného pravidla):

zákaz	nerozhodnuto	povolení	rozhodnutí
ne	ne	ne	zakázán
ano	ne	ne	zakázán
ano	ne	ano	zakázán
ne	**ne**	**ano**	**povolen**
ne	ano	ano	zakázán
ne	ano	ne	zakázán

Povolení přístupu ke zdroji vyžaduje, aby *všechny následující podmínky byly pravdivé*:

- alespoň jedna politika musí vyhodnotit povolení
- žádná politika nesmí vyslovit zákaz
- žádná politika nesmí určit nerozhodnutý stav
- SUN XACML modul nevrátí chybu ani neznámý výsledek

K *zákazu* však stačí, aby byla *aspoň jedna z následujících podmínek pravdivá*:

- alespoň jedna politika vrátila zákaz
- alespoň jedna politika vrátila nerozhodnutý stav
- žádná politika nevrátila povolení
- modul SUN XACML vrátil chybu nebo neznámý výsledek

4.15.1 Struktura zápisu XACML politiky

Slovník URN specifických pro Fedoru, jehož položky lze použít pro zápis operací API, atributů objektů a prostředí Fedory při zápisu politik, lze najít v souboru `vocabulary.txt`[11]. Základní struktura je zachycena zde:

```
<Policy PolicyId="deny-example"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
                 rule-combining-algorithm:first-applicable"
  xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>...</Description>
  <Target>
    <Subjects>
      ...
    </Subjects>
    <Resources>
      ...
    </Resources>
    <Actions>
      ...
    </Actions>
  </Target>

  <Rule RuleId="1" Effect="Deny">
    <Target>
      ...
    </Target>
    <Condition>
      ...
    </Condition>
  </Rule>
</Policy>
```

Každá politika má identifikátor, popis a algoritmus spojení pravidel. Ten je použit pro výpočet výsledku, obsahuje-li politika více definovaných

[11] `$FEDORA_HOME/server/fedora-internal-use/vocabulary.txt`.

pravidel. Je zcela nezávislý na stejných algoritmech v ostatních politikách i na způsobu určení výsledné hodnoty ze všech politik.

Tělo dokumentu se skládá z *cíle politiky* (Policy Target) a jednoho nebo více *pravidel* (Rule).

V elementu Resources je určeno, kterých zdrojů se daná politika týká (objekty, datové proudy, diseminace atd.). Je-li uvedeno více elementů Resources, bere se jejich logický součet, logický součin je realizován elementy ResourceMatch uvnitř Resources. Actions udává, na které operace se pravidlo uplatňuje, a Subjects se týká uživatele nebo agenta, jenž se snaží vykonat operaci nad daným zdrojem. Logické operace součet a součin na akcích a subjektech se provádí analogicky jako u zdrojů. Standard XACML definuje ještě element Environments, který dosud není implementován v SUN XACML, tudíž není ani použitelný ve Fedoře.

Element Rule definuje efekt, jehož hodnota je buď Permit pro povolení nebo Deny pro zákaz provedení operace. Je-li uvnitř uveden cíl Target, použije se v tomto pravidle, v opačném případě bude uplatněn cíl z kořenového elementu Policy. Podmínka Condition je predikát, který musí být splněn, aby bylo pravidlo bráno v úvahu. Ten může být sestaven z XACML funkcí.

4.16 Struktura distribuce Fedory

V této sekci je popsána struktura distribuce při použití implicitního nastavení. V adresáři `$FEDORA_HOME` jsou následující položky:

- **client** – pouze v případě instalace klienta
  - **bin** – spustitelné skripty pro prostředí UNIX (bez přípony a *.sh) a Windows (*.bat)
  - **demo** – ukázkové datové digitální objekty, definice a mechanismy chování ve formátech FOXML a METS pro lokální přístup (`local-server-demos`), tak i využívající prostředí Internetu (`open-server-demos`)
  - **lib** – knihovny používané klientem
  - **license** – licenční podmínky použití klienta i podmínky použití obsažených knihoven
  - **logs** – záznamy provedených operací
  - *client.jar* – klient
  - *fedora-admin.properties* – poslední použité přihlášení

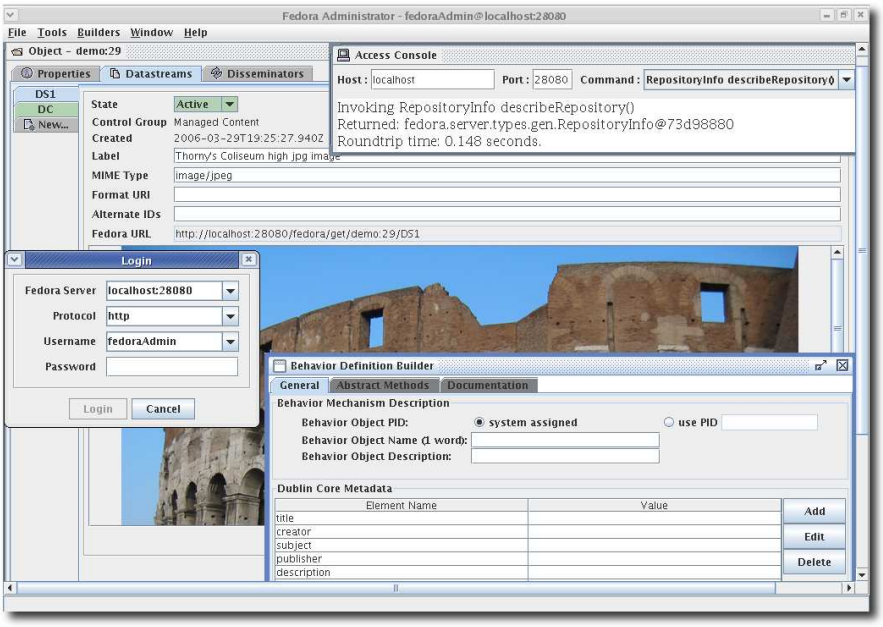
  - *truststore* – certifikát Fedory potřebný pro přístup přes SSL

- **data**
  - **datastreams** – zde jsou uloženy jednotlivé verze datových proudů, které jsou spravovány repozitářem, struktura jejich umístění je *YYYY/MMDD/HH/mm/PID+DS+DSV*, kam za jednotlivé proměnné dosadíme postupně rok, měsíc, den, hodinu, minutu vložení, PID, název datového proudu a název jeho verze (znaky, které nemohou být obsaženy v názvu souboru, jsou nahrazeny podtržítkem)
  - **fedora-xacml-policies** – nastavení politik řízení přístupu
  - **objects** – zde jsou uloženy FOXML soubory jednotlivých digitálních objektů, struktura je *YYYY/MMDD/HH/mm/PID*, kam za jednotlivé proměnné dosadíme postupně rok, měsíc, den, hodinu, minutu vložení a PID (znaky, které nemohou být obsaženy v názvu souboru, jsou nahrazeny podtržítkem)
  - **resourceIndex** – udržuje index zdrojů, velikost souborů v tomto adresáři při zapnutí indexování naroste na cca 250 MB

- **server**
  - **access** – styly XSL transformací používané pro prezentaci výsledků webových služeb API-A
  - **bin** – spustitelné skripty pro prostředí UNIX (bez přípony a *.sh) a Windows (*.bat)
  - **config** – nastavení serveru
  - **fedora-internal-use** – šablony různých druhů nastavení, používáno skripty
  - **jakarta-tomcat-5.0.28** – webový kontejner Jakarta Tomcat
  - **license** – podmínky použití Fedory a licenční ujednání obsažených knihoven
  - **logs** – záznamy činnosti serveru
  - **management** – použito pro služby API-M
  - **mckoi1.0.3** – databázový server použitelný pro testování Fedory
  - **schematron** – schémata FOXML a METS souborů
  - **userdocs** – uživatelská dokumentace
  - **utilities** – utility usnadňující nastavení

– **xsd** – schémata XML
– *status* – záznam o startu serveru
– *truststore* – certifikát Fedory potřebný pro přístup přes SSL

4.17 Klient Fedory



Obrázek 4.8: Vzhled aplikace pro správu digitálních objektů ve Fedoře.

K přístupu k digitálním objektům a jejich správě je možné použít klienta, který je rovněž k dispozici na [27]. Je to aplikace napsaná v programovacím jazyce Java a jako taková potřebuje pro svůj běh nainstalované javovské běhové prostředí. Svými funkcemi a vlastnostmi se však hodí pouze pro správce, nelze ji považovat za uživatelsky přívětivé a intuitivní prostředí z hlediska koncového uživatele.

Po spuštění je zobrazeno přihlašovací okno, kde je nutné specifikovat adresu serveru digitální knihovny a údaje o uživateli. Poté se aplikace připojí a stáhne informace o nastavení Fedory a zalogovaném subjektu.

Jedna z nejpoužívanějších funkcí je určitě *vyhledání* digitálních objektů. Po vyplnění podmínky hledání se zobrazí seznam výsledků vyhovujících těmto vlastnostem. Dvojitým kliknutím na vybraný řádek zpřístupníme okno s informacemi o vybraném objektu repozitáře. Na první záložce jsou

uvedeny jeho vlastnosti a pomocí dole přítomných tlačítek lze zobrazit jeho FOXML, exportovat ho nebo smazat. Další ze záložek obsahuje seznam datových proudů, jejichž atributy se po klepnutí myší zobrazí a umožní jejich změny. K prohlédnutí obsahu vybrané verze je nutné použít tlačítko View. Spodní údaj v seznamu datových proudů nese označení New... a je určen pro přidání nového datastreamu. Poslední záložka je přítomna pouze u datových digitálních objektů (označení Object v titulku okna) a obsahuje seznam diseminátorů a položku pro přidání nového. U každého z nich musí uživatel specifikovat pomocí definice chování, jaké funkce bude zpřístupňovat, a pomocí mechanismu chování určit způsob jejich volání. Pak už jen zbývá nastavit provázání datových proudů a parametrů metod.

Užitečnou funkcí je určitě *pomocník pro vytváření nových definicí chování a jejich mechanismů*, pomocí něhož je možné intuitivně „naklikat" požadované údaje. Rozhodně je škoda, že takto lze pouze vytvářet nové objekty a nelze editovat ty, co již jsou obsaženy v repozitáři.

Dalším usnadněním jsou *dávky*, pomocí nichž lze uložit do repozitáře najednou více objektů.

Jako zajímavý nápad se jeví zpřístupnění *rozhraní k funkcím API-M a API-A*. Před každým zavoláním je uživateli nabídnuta možnost nastavit parametry. Velmi mě zklamaly výsledky typu

```
Invoking RepositoryInfo describeRepository()
Returned: fedora.server.types.gen.RepositoryInfo@73d98880
Roundtrip time: 0.127 seconds.
```

Pro vývojáře by jistě nebyl žádný problém implementovat zobrazení údajů z vráceného objektu (např. název repozitáře). V současném stavu je výsledek čitelný pouze tehdy, je-li řetězcem nebo jejich polem, a to dosti ubírá na použitelnosti této konzole.

K dalším funkcím klienta již není třeba nic dodávat. K nim patří vytvoření nového objektu, jeho smazání, zobrazení FOXML, export a import ze souboru či adresáře.

Kapitola 5

Služby poskytované repozitářem Fedora

Tato kapitola popisuje možnosti a způsoby komunikace mezi serverem digitální knihovny a okolím. Tím mohou být aplikace, které přistupují k jejím datům, klienti a další služby, které nejsou přímo součástí jádra sytému.

Fedora poskytuje v současné verzi 2.1.1 volání svých funkcí ve dvou formách. První z nich je použití *webových služeb* (viz sekce 3.4 na straně 8). Ty jsou realizovány na straně Fedory použitím knihovny Apache Axis ([10]). Jako druhý možný způsob se nabízí použití *REST* (viz sekce 3.5 na straně 9). Tento druhý typ služeb nese označení LITE.

Funkce se dále dělí podle své funkčnosti. První skupinu tvoří služby pro přístup k digitálním objektům, informacím o nastavení repozitáře nebo o přihlášeném uživateli, tzv. *Fedora Access Service*. Jejich SOAP verze je označována jako *API-A*[1]. REST verze nese označení *API-A-LITE*. Druhou množinu tvoří volání, která lze použít pro správu digitálních objektů, například vkládání, mazání a úpravy, tzv. *Fedora Management Service*. SOAP verze je pak pojmenována *API-M*, REST potom *API-M-LITE*[2].

Konkrétní názvy funkcí, vysvětlení významu jednotlivých parametrů a vrácené hodnoty se nachází v dokumentaci na stránkách Fedory ([27]). Při použití webových služeb z programovacího jazyka Java doporučuji použít nástroje Apache Axis ([10]), které z dodaného WSDL vytvoří třídy zpřístupňující jejich volání, zajistí veškeré kódování požadavků a dekódování odpovědí zaslaných zpět serverem.

5.1 Fedora Access Service

Nejdůležitějším úkolem této služby je poskytovat informace o digitálních objektech a jejich diseminátorech, o přístupu k nim a o možnostech jejich

[1] WSDL je umístěno na adrese `http://www.fedora.info/definitions/1/0/api/#apia`

[2] WSDL lze nalézt na adrese `http://www.fedora.info/definitions/1/0/api/#apim`

použití. Webové služby komunikují se spodními vrstvami architektury systému, jež vyhodnocují, která chování jsou asociována s daným digitálním objektem, a vyřizují požadavky volání.

5.1.1 Informace o repozitáři

Zavoláním funkce `describeRepository` získáme základní informace o repozitáři, k němuž přistupujeme. Mezi dosaženými údaji jsou název repozitáře, základní adresa, na níž je spuštěn, ukázkové URL adresy a e-maily správců.

5.1.2 Vyhledávání objektů

Metoda `findObjects` zajišťuje vyhledávání. Prvním parametrem je pole obsahující názvy položek, které chceme získat ve výsledcích. Ty vybíráme z následujícího seznamu:

PID – pid

Vlastnosti objektu – label, fType, cModel, state, ownerId, cDate, mDate, dcmDate

Diseminátory – bDef, bMech

Metadata DC – title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage, rights

Druhým parametrem je maximální počet výsledků, jež chceme získat. Může mít hodnotu **null**. Skutečně vrácených položek však nikdy není více, než je nastaveno v konfiguračním souboru serveru.

Naposledy předáváme vyhledávací dotaz. Ten může mít 2 formy – prostou textovou nebo podrobnější, v níž lze specifikovat zvlášť hodnoty jednotlivých vlastností a metadat. Jsou-li definovány obě varianty, uplatní se prostý text. V obou možnostech lze používat hvězdičkovou konvenci[3].

Při definici podrobnějšího vyhledávacího dotazu použijeme stejné položky jako v prvním parametru (viz předchozí seznam). Ke každé z nich definujeme jeden z následujících vztahů: *obsahuje* (kdekoli v položce je přítomna, používá se hvězdičková konvence), *je rovno* (jde o přesnou shodu, nepoužívá se hvězdičková konvence) a u číselných hodnot a dat také *je*

[3] Otazník `?` zastupuje právě jeden libovolný znak, hvězdička `*` potom libovolný počet opakování znaků (včetně nulového).

menší, *je menší nebo rovno*, *je větší*, *je větší nebo rovno*. Nakonec určíme hodnotu, se kterou proběhne porovnávání. Mezi jednotlivými položkami dotazu je proveden logický součin, tj. budou vráceny objekty, které splňují všechny zadané podmínky.

Výsledek obsahuje pole informací o vyhovujících digitálních objektech. Vyplněny jsou položky, jež byly specifikovány v prvním parametru volání. U diseminátorů a metadat DC jsou vrácena pole, neboť hodnot může být více.

Bylo-li vyhovujících objektů příliš mnoho a nebyla vrácena data o všech z nich, obsahuje výsledek *token* s hodnotu různou od **null**. Ten je po určitou omezenou dobu, kterou lze nastavit v konfiguračním souboru serveru, možné předat jako parametr `resumeFindObjects` a získat další sadu výsledků.

5.1.3 Práce s konkrétním digitálním objektem

Profil vybraného objektu získáme pomocí `getObjectProfile` s parametry PID objektu a časovou známkou v tomto pořadí. Bude-li místo ní předána hodnota **null**, vrátí se momentální stav.

Pokud potřebujeme určit, ve kterých časech probíhaly změny objektu, použijeme metodu `getObjectHistory`, jež pro definovaný identifikátor PID vrátí pole časových známek.

Seznam datových proudů obdržíme zavoláním `listDatastreams` s parametry stejnými jako u `getObjectProfile`. Vlastní data z nich přečteme pomocí `getDatastreamDissemination`.

Předáním opět stejných hodnot parametrů jako u `getObjectProfile` funkci `listMethods` získáme seznam všech možných diseminací uvedeného objektu, kterých můžeme dosáhnout použitím `getDissemination`. Parametry jsou zapsány v tomto pořadí: PID objektu, PID definice chování, název metody, pole parametrů a časová známka, pomocí níž je určen stav v minulosti, který chceme získat. Pokud si přejeme vidět aktuální stav, necháme v posledním parametru hodnotu **null**.

5.2 Fedora Management Service

Pod tímto označením se vyskytují správcovské služby, jimiž jsou především přidání objektu, jeho datových proudů a diseminátorů a jejich změny.

5.2.1 Informace o uživateli

Pomocí funkce `describeUser`, jíž předáme přihlašovací jméno, získáme základní informace o uživateli. Bohužel však obsahuje pouze logickou hodnotu, zda-li se jedná o administrátorský účet, což je vzhledem k velikým možnostem definování politik dosti nepostačující.

5.2.2 Vkládání a odstraňování objektů

Nový objekt můžeme vložit do repozitáře zavoláním `ingest`. Jako parametry obsahuje XML s definicí, jeho formát, který nabývá hodnoty `foxml1.0` pro FOXML (viz sekce 4.11 na straně 21) nebo `metslikefedora1` pro METS, a zprávu o vložení. Stav digitálního objektu bude nastaven na aktivní. V kořenovém elementu můžeme definovat atribut `OBJID`, jehož hodnota obsahuje PID, které si přejeme novému objektu přiřadit. Vyhověno bude pouze v případě, že jsou splněny všechny následující podmínky:

- zápis odpovídá syntaxi PID
- je použit jmenný prostor, který máme obsažen v seznamu hodnot v parametru `retainPIDs` v konfiguračním souboru Fedory
- objekt s tímto identifikátorem v repozitáři neexistuje

Navrácená hodnota je rovna PID nově přidaného digitálního objektu.

Kvůli kompatibilitě s dřívějšími verzemi Fedory je definována i funkce `ingestObject`, nyní již zavrhovaná. V jejích parametrech není uveden formát XML, automaticky se předpokládá METS. Ve verzích 1.x byl totiž jedinou možností.

Volné hodnoty identifikátorů zjistíme pomocí `getNextPID`. Zadáme požadovaný počet a jmenný prostor, vráceno bude pole řetězců obsahujících přiřazené identifikátory. Příští zavolání vrátí hodnoty další v pořadí, tyto jsou rezervovány a přiřazeny budou pouze v případě, že o toto bude požádáno při vkládání nového objektu (viz popis výše).

Pokud si přejeme trvale objekt odstranit, poslouží nám k tomu metoda `purgeObject`. Ta pracuje s parametry PID objektu, záznam o odstranění a logická hodnota určující, má-li dojít ke smazání i v případě, že by došlo k porušení závislostí mezi objekty v repozitáři.

5.2.3 Změna vlastností objektu

U existujícího objektu lze provést změnu jeho stavu a pojmenování. K tomuto účelu slouží `modifyObject`. Předáváme jí PID měněného objektu,

nové hodnoty zmiňovaných charakteristik v uvedeném pořadí a logovací zprávu o provedené operaci. Navrácena bude časová známka okamžiku nahrazení hodnot.

5.2.4 Práce s datovými proudy

Datové proudy již objekt může obsahovat při svém vytváření. Deklarují se přímo ve FOXML (bližší popis viz 4.11.2 na straně 23). Pozdější manipulaci umožňují funkce popsané v této sekci.

Vkládání nového datového proudu do existujícího objektu se provede zavoláním `addDatastream` s těmito parametry:

PID – identifikátor digitálního objektu, do něhož bude nový proud přidán

identifikátor datového proudu – řetězcové označení s délkou maximálně 64 znaků, při hodnotě **null** Fedora vhodné jméno vygeneruje sama

alternativní identifikátory – pole identifikátorů, které budou také odkazovat na tento datový proud

název datového proudu – „lidsky čitelný“ text

udržovat verze – logická hodnota, jež určuje, zda bude udržována historie verzí či dojde při změně obsahu k přepsání

typ MIME – formát dat

URI formát – další přístup k popisu typu uložených dat, tato položka je volitelná, může nabývat hodnoty **null**

umístění dat – z této pozice bude získán obsah datového proudu (pro typy X a M) nebo na ni bude odkazováno (typy R a E)

typ datového proudu – hodnota `X`, `M`, `E`, či `R`

stav datového proudu – hodnota `A`, `I`, nebo `D`

logovací záznam – informace o provedené akci

Navrácen bude identifikátor přidaného datového proudu.

Provedení změny v datovém proudu lze uskutečnit dvěma různými způsoby. V prvním funkci `modifyDatastreamByReference` předáme stejné parametry jako v předchozím případě, pouze vynecháme typ datového proudu a na konec přidáme logickou hodnotu určující, zda má dojít

k žádané změně i v případě porušení datové dohody. Druhou variantu nabízí `modifyDatastreamByValue`. V jejích parametrech jenom zaměníme umístění dat přímo za jejich hodnotu.

Informace o existujícím datastreamu získáme prostřednictvím metody `getDatastream`. Předáme PID objektu, identifikátor datového proudu a časovou známku, definující verzi.

Pole s časovými známkami okamžiků všech změn proudu obstarává `getDatastreamHistory`, pole metadat datových proudů objektu určité verze v definovaném čase `getDatastreams`, změny stavu docílíme předáním této nové hodnoty `setDatastreamState` a jeho trvalé odstranění zajistí `purgeDatastream`.

5.2.5 Práce s diseminátory

Stejně jako datové proudy, tak také diseminátory mohou být součástí FOXML, definujícího objekt, už při jeho vkládání do repozitáře (podrobněji viz 4.11.3 na straně 24). Pokud se rozhodneme vložit nový diseminátor až posléze, využijeme k tomu `addDiseminator` s těmito parametry:

PID – PID objektu, do kterého dojde ke vložení

definice chování – PID objektu s definicí chování, jejíž metody diseminátor implementuje

mechanismus chování – PID objektu s mechanismem chování, jenž odkazuje na umístění služeb

název diseminátoru – „lidsky čitelný“ název diseminátoru

mapa provázání datových proudů – svázání proměnných z definice chování se skutečně existujícími datovými proudy v datovém digitálním objektu

stav diseminátoru – aktivní, neaktivní, či smazaný, stejný význam jako u digitálních objektů

logovací záznam – záznam o provedené akci

Vrácen bude identifikátor vytvořeného diseminátoru.

Pro změnu zavoláme `modifyDisseminator` s výše definovanými parametry, k nimž přidáme logickou hodnotu, jež určí, zda se má modifikace provést i za cenu porušení datové dohody.

S analogickým významem jako u datových proudů fungují i metody `getDisseminator`, `getDisseminatorHistory`, `getDisseminators`, `purgeDisseminator` a `setDisseminatorState`.

5.2.6 Export objektu

Exportovat objekt můžeme ve třech možných způsobech. *Veřejný* je vytvořen se zřetelem na možnost jeho použití v prostředí mimo repozitář. Obsahy všech datových proudů jsou přístupné přes veřejné URL adresy. *Migrační* lze použít pro přenos digitálního objektu z jednoho repozitáře Fedora do druhého. *Archivní* zatím není k dispozici, ale přidání jeho podpory je plánováno v příštích verzích. Tato možnost bude generovat nezávislý soubor obsahující všechna data. Do něho budou vloženy všechny datové proudy – XML přímo a ostatní budou nejprve překódovány do base64.

Získání dat je zpřístupněno přes funkci `export`. Ta přijímá hodnoty následujících parametrů:

PID – identifikátor exportovaného objektu

formát – `foxml1.0` pro FOXML, `metslikefedora1` pro METS

způsob exportu – `public` pro veřejný, `migrate` pro migrační (a v budoucnu též `archive` pro archivní, viz výše)

Za zmínku ještě stojí existence funkce `exportObject`, která je přítomna z důvodu kompatibility s předchozími verzemi. V současnosti patří mezi zavrhované.

Kapitola 6

Implementace webového rozhraní

Součástí mé práce bylo též navrhnout a implementovat webové rozhraní pro prohlížení a správu digitálních objektů v repozitáři. Pro realizaci jsem si vybral programovací jazyk Java a v prostředí NetBeans zhotovil webovou aplikaci, která umožňuje v uživatelsky příjemném a intuitivním prostředí přístup k digitálním objektům v nastaveném repozitáři Fedora. V současném stavu obsahuje lokalizace pro český a anglický jazyk, doplnění dalších překladů je umožněno snadným způsobem.

Aplikace rozlišuje digitální objekty na jednotlivé typy pomocí vlastnosti digitálního objektu **contentModel**. Každý z nich má přiřazenou třídu, která zajišťuje veškeré nakládání s objektem. Zejména poskytuje pohledy, jež mají na starosti vypisování a zobrazování objektů, a akce, které jsou určeny pro manipulace s objekty.

Použitý prohlížeč musí podporovat cookies a mít je zapnuté. Aplikace komunikuje v jazyce odeslaném prostřednictvím hlavičky HTTP požadavku, pokud danou lokalizaci podporuje. Je též doporučeno mít povoleno skriptování v jazyce JavaScript a otevírání nových oken, ale není to podmínkou pro bezproblémový přístup, pouze to zvyšuje pohodlnost pro návštěvníka.

Na začátku přístupu k aplikaci si uživatel zvolí repozitář, který se chystá navštívit, a přihlásí se pomocí svého přihlašovacího jména a hesla, jež má přiděleno do systému Fedora. Po přihlášení bude ihned přesměrován na základní stránku, která obsahuje kořenový objekt. Pomocí oken umístěných v levém pruhu je možné vyhledávat objekty, nastavit použité třídění nebo spouštět akce. V hlavním okně se zobrazuje aktuální objekt, popř. průběh jednotlivé akce apod.

6.1 Licence

Aplikace a její zdrojové kódy mohou být šířeny pod GPL. Software obsahuje následující knihovny třetích stran – Java Server Faces 1.1, Java Standard Tag

Library 1.1, Apache Axis, DOM4J, Commons FileUpload a nástroje, které potřebují ke své práci.

6.2 Konfigurace

Základní nastavení aplikace je přístupné skrze třídu **Configuration**[1]. Instanci získáme její statickou metodou `getConfiguration()`. Všem voláním během jednoho spuštění je vrácena stejná instance, změny se tedy projeví v celé aplikaci.

Nyní se podrobněji zaměřme na údaje, které nám třída poskytuje. Pomocí `getUrl()` zjistíme adresu, na níž klient běží. Pokud předáme metodě `getGetObjectUrl` jako parametry PID objektu a logickou hodnotu, zda chceme náhled, získáme URL adresu, na níž klient zobrazí požadovaný objekt. Pole nadefinovaných repozitářů, jejichž volba je umožněna uživateli, vrací `getRepositories()`. Funkce `getRcDir` vrátí cestu do adresáře s nastavením klienta.

6.3 Repozitář

Během procesu přihlašování si uživatel může zvolit repozitář, k němuž si přeje přistupovat a pracovat s ním. Údaje o něm jsou uchovávány ve třídě, která implementuje rozhraní **Repository**. Tou je v současné verzi aplikace **FedoraRepository**[2]. Pomocí ní lze získat URL adresu serveru Fedory (**`getURL()`**), URL adresu, na které jsou přístupny webové služby API-A (**`getFedoraAPIAUrl()`**) a API-M (**`getFedoraAPIMUrl()`**), kořenový objekt (**`getRootDirectory()`**) a registr typů digitálních objektů (**`getDigitalObjectTypesRegister()`**). Ten k dodanému řetězci, který je uchováván ve vlastnosti objektu contentModel, vrátí javovskou třídu, jež reprezentuje udaný typ digitálního prostoru z pohledu této aplikace.

6.4 Uživatel

Pro uchování informací o přihlášeném uživateli, který právě pracuje prostřednictvím této webové aplikace s digitální knihovnou, je použita třída **FedoraRepositoryUser**, která implementuje rozhraní **RepositoryUser**. K aktuálnímu repozitáři (viz sekce 6.3) můžeme přistoupit prostřednictvím me-

[1] V balíku *cz.muni.fi.xnovot10.diplomka.config*.

[2] Rozhraní **Repository** i třída **RepositoryUser** jsou umístěny v balíku *cz.muni.fi.xnovot10.diplomka.repository*.

tody **`getRepository()`**. Instance uživatele obsahuje přihlašovací jméno a heslo, autorizační hlavičku, která je nutná pro autorizaci vůči serveru Fedory a lze ji získat zavoláním **`getAuthorizationHeader()`**. Nabízí se možnost využívat veškerá volání webových služeb prostřednictvím metody **`getFedoraAPIA()`** pro API-A, resp. **`getFedoraAPIM()`** pro API-M. Pro každého uživatele je také připravena cache paměť, v níž jsou uchovávány objekty získané v poslední době, obvykle pouze během vyřizování jednoho požadavku. Po něm je pak tato paměť zcela vyprázdněna. Byli v minulosti změněn u některého typu objektu pohled, bude zachován i pro příští zobrazení objektu stejného druhu (**`getView(`*`contentModel`*`)`**). Pro každý z pohledů je uložena informace o předešlé změně komparátoru (**`getComparator(`*`viewClassName`*`)`**). Poslední prováděná vyhledávání a již zobrazená část výsledků je dočasně uchovávána, aby bylo umožněno procházení položek a vracení se k nim před vypršením jejich životnosti (**`getSearchCache()`**).

6.5 Typ objektu

Pro vytvoření typu objektu je potřeba rozšířit abstraktní třídu **AbstractDigitalObject** v balíčku *cz.muni.fi.xnovot10.diplomka.repository.digitalObject* a implementovat její abstraktní metody, popřípadě překrýt metody nevyhovující. Pro vytvoření instance musí být přítomen jednak bezparametrický konstruktor, jednak konstruktor s parametry v pořadí **RepositoryUser** `user`, **String** `pid` a **String** `asOfDate`.

Získání konkrétní instance digitálního objektu zajišťuje statická metoda `getDigitalObject` třídy **AbstractDigitalObject**. Ta se nejprve podívá do cache paměti, není-li již konkrétní objekt přítomen. Pokud ne, prostřednictvím webových služeb získá informace o objektu. Podle vlastnosti contentModel vyhledá třídu, jež je pro daný typ registrovaná, a vytvoří její instanci. Tu potom vrátí.

Pomocí této třídy máme k dispozici vlastnosti objektu, metadata DC zpřístupníme zavoláním `loadDC()`. Dostupné jsou přes intuitivně pojmenované get metody.

Seznam potomků vrací `getChildren`, předků `getParentObjects`.

Okamžik, jehož stav daná instance třídy právě reprezentuje, skrývá `getObjectTimestamp`. Je-li výsledkem **null**, jedná se o aktuální stav.

Chceme-li objekt uložit do repozitáře, poslouží nám k tomu metoda `ingest`. Té předáme buď pouze logovací zprávu, nebo též PID objektu, který se stane rodičovským.

Naopak smazání zajišťuje `purge`. Hodnota `deep` určí, zda jsou odstraněni i všichni potomci, ke kterým neexistuje jiný rodič, nebo jen aktuální objekt.

O výše uvedené metody se nemusíme starat, jsou implementovány již abstraktní třídou.

Mezi nejdůležitější věci, které je potřeba provést, abychom získali funkční implementaci nového typu, patří vytvoření jeho pohledů a akcí.

Pokud potřebujeme uchovávat pouze vlastnosti objektu a metadata DC (např. zobrazujeme jenom položky adresáře), můžeme k tomu využít třídu **DigitalObjectInfo**, jež ale neumožňuje manipulace s objekty.

6.6 Pohledy

Pohled reprezentuje způsob, jakým probíhá zobrazení objektu daného typu v klientovi. Pro lepší představu uvedu příklad možné implementace zobrazení obsahu adresáře. Mezi nabízené varianty patří prosté vypsání názvů digitálních objektů v něm obsažených, jejich vypsání včetně údajů o vytvoření, poslední změně a typu objektu, nebo možnost nechat jednotlivé objekty, aby samy vygenerovaly své náhledy. Samozřejmostí všech variant je, že po kliknutí na vybranou položku dojde k jejímu zobrazení.

Pohledem rozumíme třídu rozšiřující **DigitalObjectAbstractView** z balíku *cz.muni.fi.xnovot10.diplomka.repository.digitalObject.view*. Nutností je existence konstruktoru, kterému se prostřednictvím parametru předává instance třídy **AbstractDigitalObject**, jež reprezentuje konkrétní digitální objekt z repozitáře. Odtud můžeme získat potřebné údaje. Z nejdůležitějších jmenujme repozitář, z něhož získáváme objekty, jazyk lokalizace, který by měl být pokud možno použit, rodiče a potomky, vlastnosti objektu, datové proudy, metadata DC a uživatele, který s ním momentálně pracuje. Pomocí posledně jmenovaného máme rovněž přístup ke všem metodám webových služeb, jež Fedora nabízí.

Každý pohled musí vracet prostřednictvím metody `getName()` svůj *identifikátor* složený ze znaků anglické abecedy. Ten musí být jednoznačný mezi názvy všech pohledů v rámci objektu, ve kterém je použit.

Zobrazování probíhá ve dvou rozdílných polohách. V první z nich může rodičovský objekt požadovat, aby pohled vrátil zobrazení svého *náhledu*. Ten lze použít například ve výpisu adresáře, v němž mají možnost obrázky zobrazit svůj náhled, články krátký výtah, zvukové soubory odkaz na ukázku atp., aniž by rodič tušil, jaké objekty obsahuje.

Druhou polohu představuje vlastní zobrazení *údajů o objektu*, který je re-

prezentován pohledem. K tomuto kroku patří i vypsání informací o potomcích, je-li to žádoucí. Můžeme využít toho, že každý potomek umí pomocí svého základního pohledu vygenerovat svůj náhled.

Pro realizaci vypisování dostaneme jako parametr k dispozici kontext JSF, z něhož získáme instanci objektu rozšiřujícího abstraktní třídu **ResponseWriter**. Jeho použití umožňuje tvorbu úseku XHTML kódu, který bude vložen do výsledné stránky. „Obalen" bude ovládacími prvky.

Nejdříve je nutné vytvořit *definici kaskádových stylů* CSS (Cascading Style Sheets, [40]). Ta bude vložena do hlavičky stránky, na níž bude použit tento pohled. Název každé třídy by měl začínat řetězcem, který vrátí příkaz `getDigitalObject().getName()`, aby nemohlo docházet ke kolizím způsobeným stejným pojmenováním. Tyto záležitosti má na starosti metoda `getCSSStyle()`, resp. `getPreviewCSSStyle()` pro náhledy. Ta musí vrátit řetězec, v případě nezájmu použití tříd kaskádových stylů může být prázdný, nesmí však být **null**.

Pokud chceme, aby měl uživatel možnost výběru, jakým způsobem chce setřídit zobrazení potomků, např. položek adresáře, článků v časopise nebo obrázků v galerii, použijeme k tomu *komparátory*. Těmi jsou objekty rozšiřující třídu **AbstractViewComparator**, která implementuje rozhraní *java.util.***Comparator**. Použití řazení při zobrazování je zcela ponecháno na pohledu, jenž zjistí aktuálně vybraný komparátor pomocí metody `getActualViewComparator()`. Pokud nebyla doposud provedena žádná volba, dojde k návratu hodnoty `getDefaultViewComparator()`. Při vstupu klienta na daný objekt se totiž uživateli vypíše v okně seznam možných třídění a umožní se mu jejich výběr. Naposledy vybrané třídění si aplikace zapamatuje v rámci jednoho přihlášení a při příštím použití stejného pohledu jej znovu uplatní. Není třeba definovat komparátory pro třídění vzestupné i sestupné, systém nabízí obrácení pořadí pomocí speciálního adaptéru.

Implicitní pohled vrací typ objektu metodou `getDefaultView`. Pokud si přejeme nabídnout uživateli volbu, kterým pohledem si přeje konkrétní instanci zobrazit, vrátíme seznam pohledů prostřednictvím `getViews`. V tomto případě musí fungovat překlad z identifikátoru pohledu na instanci pohledu v `getViewByName`.

6.7 Akce

Každý typ objektu též obsahuje seznam akcí, které nad ním lze provádět. *Akcí* rozumíme jednak třídu rozšiřující **AbstractAction**, tzv. *definici akce*, tak

také *vykonavatele* rozšiřujícího **AbstractActionProcessor**[3]. Definice akce nese informace nutné pro vypisování názvu a instrukce upřesňující, pro koho je určena. Po zavolání její metody `getActionProcessor` (jejímž parametrem je objekt, na kterém je vyvolána) se vrátí samotný vykonavatel, který se stará o vlastní provádění a vypisování údajů o průběhu.

Během procesu akce je pro usnadnění práce programátorům kontrolováno, zda-li nebylo použito obnovení stránky prohlížečem nebo vracení se zpět. V tom případě aplikace ohlásí chybu. Není ovšem zakázáno postup kdykoliv přerušit nebo spustit znovu.

Každá akce se skládá z *kroků*. Aktuální krok lze zjistit pomocí zavolání metody `getPhase()`. Z programátorského hlediska je krokem libovolný řetězec znaků anglické abecedy, jenž akci pomáhá určit, ve které fázi se momentálně nalézá. Po spuštění nové akce je automaticky nastaven na **start**. Ukončena je určením hodnoty příštího kroku (pomocí metody `setNextActionPhase(`**String** `phase)`), pro kterou je zavoláním `isFinalPhase` s příslušným parametrem vrácena hodnota **true**. Uživatel bude přesměrován na adresu `getURL()`, implicitně na zobrazení objektu, na němž byla akce volána.

Pokud se nacházíme v kroku, jenž není koncovým, je zavolána metoda `performAction`, v níž máme možnost vykonat požadované činnosti. Následně se ocitneme v `encodeBegin`, kde prostřednictvím JSF kontextu vygenerujeme kód pro zobrazení kroku v aplikaci a nastavíme název dalšího kroku.

Seznam akcí, které se týkají daného objektu, vrací instance typu metodou `getActions`. Objekt také může definovat tzv. *univerzální akce*. Těmi jsou ty, které vrátí prostřednictvím `getUniversalActions` a jsou většinou určeny pro větší počet typů objektů. Jejich seznam je udržován pro každý repozitář zvlášť[4]. Každý objekt by měl od každé z nich získat svoji vlastní instanci pro aktuální objekt[5] a přidat ke svým akcím ty, které vrátí souhlas s konkrétním objektem[6] a uživatelem[7].

Analogicky jako u pohledů i zde musí fungovat překlad identifikátorů na konkrétní instance akcí metodou `getActionByName`, a to včetně univerzálních akcí.

[3] Obě zmíněné abstraktní třídy jsou obsaženy v balíčku *cz.muni.fi.xnovot10.diplomka.repository.digitalObject.action*.

[4] `Repository.getDigitalObjectTypesRegister().getUniversalActions()`

[5] `getInstance(object)`

[6] `isForThisObject()`

[7] `isForThisUser()`

6.8 Zpřístupnění dat přes URL

Ve spoustě případů potřebují typy, pohledy nebo akce zpřístupnit nějaká svá data vnějšku prostřednictvím URL adresy. K tomuto účelu je implementován registr takovýchto informací třídou **ContentHandler**[8]. Všechny její metody jsou statické. Poskytovatel zaregistruje vstupní proud dat spolu s udaným typem MIME[9] a získá identifikátor. Z něho dostane jednoduchým způsobem URL adresu[10], na níž je objekt dosažitelný prostřednictvím obsluhujícího servletu. Životnost je omezena nastavenou dobou a pouze jedním možným přístupem. Potom dojde k odstranění zdroje a uvolnění prostředků, jež používal.

6.9 Soubory s nastaveními aplikace

V adresáři s nastaveními aplikace[11] jsou přítomny konfigurační soubory `objects/objectTypes.xml` s přiřazením modelů digitálních objektů k implementaci javovských tříd, jež jsou jejich reprezentanty z pohledu klienta, `config/config.xml` s vlastnostmi aplikace jako celku a certifikát serveru Fedora `truststore`. Ten je nutný při přístupu přes šifrovaný protokol SSL k ověření identity.

6.9.1 objectTypes.xml

Jedná se o soubor v XML formátu, jehož kořenovým element je pojmenován `objectTypesConfig`. Uvnitř něho je pro každý typ definován `repositoryObjectTypeConfig`. Ten ukrývá v elementu `name` hodnotu vlastnosti `contentModel` a jako obsah elementu `fullyQualifiedName` plně kvalifikované jméno třídy, která se stará o manipulaci s digitálním objektem tohoto typu.

Pro snadnější pochopení vkládám ukázku.

```
<objectTypesConfig>
  <repositoryObjectTypeConfig>
    <name>directory</name>
    <fullyQualifiedName>
      cz.muni.fi.xnovot10.diplomka.digitalObject.type.directory.Directory
    </fullyQualifiedName>
```

[8] V balíku *cz.muni.fi.xnovot10.diplomka.sendingContent*

[9] **ContentInputStream**

[10] `getAddress`

[11] Adresář s nastaveními aplikace se nachází v umístění, na nějž odkazuje parametr `rc2006directory` v deskriptoru `web.xml` v distribuci klientské aplikace. Implicitně jím je `~/.rc2006`.

```
  </repositoryObjectTypeConfig>
  <repositoryObjectTypeConfig>
    <name>picture</name>
    <fullyQualifiedName>
      cz.muni.fi.xnovot10.diplomka.digitalObject.type.picture.Picture
    </fullyQualifiedName>
  </repositoryObjectTypeConfig>
  <repositoryObjectTypeConfig>
    <name>article</name>
    <fullyQualifiedName>
      cz.muni.fi.xnovot10.diplomka.digitalObject.type.magazine.Article
    </fullyQualifiedName>
  </repositoryObjectTypeConfig>
</objectTypesConfig>
```

6.9.2 config.xml

V tomto souboru jsou ve formátu XML uvedena nastavení aplikace jako celku.

V první řadě uvádí soubor seznam repozitářů, k nimž je možné se připojit. Uživatel dostane možnost výběru během procesu přihlašování. Pro každou definici přístupu k serveru Fedory je určen zvláštní element `repository`. V něm je uvedena přístupová adresa (element `URL`) a název, jenž se ukáže v seznamu repozitářů (element `name`).

Dále je uvedeno umístění souboru s certifikátem serveru Fedory a heslo k přístupu k němu. Tento soubor je možno obstarat přímo v distribuci serveru nebo klienta.

Opět uvádím příklad.

```
<config>
  <repository>
    <URL>http://localhost:28080/</URL>
    <name>Moje Fedora (bez SSL)</name>
  </repository>
  <repository>
    <URL>https://localhost:8443/</URL>
    <name>Moje Fedora (SSL)</name>
  </repository>
  <truststore>~/.rc2006/truststore</truststore>
  <truststorePassword>tomcat</truststorePassword>
</config>
```

6.10 Konkrétní typy objektů

6.10.1 Adresáře

Prvním úkolem bylo implementovat typ, jenž by umožnil uchovávat adresářovou strukturu. K tomu slouží třída **Directory**. Umožňuje libovolnou hloubku zanořování.

Vztahy mezi adresářem a jeho položkami jsou definovány pomocí datového proudu RELS-EXT a predikátu `isMemberOf`. Jeden nese označení *kořenový adresář*[12]. K němu je přistoupeno v případě, že není požadován žádný konkrétní existující objekt.

V počátcích jsem měl problémy s výkonem. Pomocí indexu zdrojů jsem si nechal vyhledat všechny identifikátory objektů, které jsem chtěl zobrazit. Ty jsem pak získával jednotlivě voláním přes rozhraní webových služeb. Po několika neúspěšných pokusech jsem přešel na model, kdy všechna data, která zobrazuji, získám již z indexu zdrojů a uchovávám je v instancích třídy **DigitalObjectInfo**.

Pomocí tohoto typu jsou přidány univerzální akce pracující obecně s jakýmkoli typem – zobrazení a editace metadat DC, volání libovolného diseminátoru, získání vybraného datového proudu, zobrazení historie objektu a jeho odstranění.

6.10.2 Obrázek

Dalším úkolem bylo přenesení dat Digitální knihovny fotografií ÚVT Masarykovy univerzity (DKF, viz [54]) do repozitáře Fedory. Metadata jsem dostal uložená v XML souboru. Obrázků bylo více než 800.

Definoval jsem typ **Picture**. Ten uchovává v jednom datovém proudu obrázek v plném rozlišení a ve druhém zmenšený náhled. Ten je při vkládání vytvořen automaticky. Používá se při zobrazování náhledu objektu (např. při výpisu položek adresáře), kde je kromě pojmenování zobrazena i zmenšená ukázka. Dále je vytvořen další datový proud, jenž nese údaje o instanci přečtené z dodaného XML souboru.

Vložení celé galerie DKF trvalo více než půl hodiny a mazání nebylo o mnoho kratší. Částečně je tento průběh způsoben nastavením logování na nejjemnější úroveň.

[12] Predikát `http://fi.muni.cz/ xnovot10/diplomka/frc#rootDirectory` má hodnotu `root`.

6.10.3 Článek

Poslední prací bylo vytvořit též možnost uchovávání článků časopisu, konkrétně Zpravodaje ÚVT Masarykovy univerzity (viz [55]). Za tímto účelem byla vytvořena hierarchie *titul* (**Title**), *ročník* (**Year**), *číslo časopisu* (**Magazine**), *článek* (**Article**) a *stránka článku* (**Page**). První tři jmenované typy vychází z implementace adresáře.

Článek a stránka jsou definovány obdobně. Obsahují datový proud s fragmentem HTML kódu, který je použit pro přímé zobrazení v klientu, a volitelně i reprezentaci ve formátu PDF a PS.

Kapitola 7

Zkušenosti a postřehy

V této kapitole blíže popíši průběh mé práce s Fedorou, moje postřehy a nápady pro použití jednotlivých vlastností této digitální knihovny.

7.1 První krůčky

Poté, co jsem si vybral téma diplomové práce, jsem se pustil do studia informací uvedených na stránkách Fedory (viz [27]) a pročítání uživatelských informací. Problematikou digitálních knihoven jsem se do té doby podrobněji nezabýval, takže pro mě byla spousta pojmů nových.

Jelikož patřím k lidem, kteří rychleji pochopí problém, když si s ním mohou pohrát a „osahat" ho, stáhnul jsem si distribuci serveru i klienta Fedory a pustil se do konfigurování. Začínal jsem s beta verzí 2.1. Systém jsem provozoval a testoval na notebooku[1] v linuxovém operačním systému Fedora Core 3. Zpočátku jsem měl problémy s přístupovými právy a vytvořením příslušné adresářové struktury (ta byla v pozdějších vydáních přepracována). Nepříjemné bylo to, že pokus o spuštění serveru skončil vždy prostým vypsáním OK nebo ERROR bez udání bližší příčiny.

Po úspěšném zdolání problémů s instalací a nastavením a po dosažení kýženého OK při spuštění serveru jsem se začal ptát, k čemu tento systém slouží. Na první pohled totiž nebylo nic „vidět". Pustil jsem se tedy do instalace klienta. S tou naštěstí žádné komplikace nebyly, javovská aplikace se spustila bez obtíží, ale já ještě stále nic neměl „v ruce", repozitář byl pořád prázdný. Přidal jsem tedy do něho dodávanou sadu ukázkových objektů, jejich definic a mechanismů chování a konečně mohl začít zkoušet pracovat s věcmi, o nichž jsem četl v dokumentaci.

Nepříjemné je to, že při použití standardního diseminátoru a přístupu přes WWW prohlížeč systém vrací při různých chybách stále stejnou hlášku[2]. Jejich pravou příčinu je obtížné dohledat i v logovacích záznamech, zvláště

[1] Procesor Intel Pentium 4 2.4 GHz, 1 GB operační paměti.
[2] Fedora Security Error, authorization failed

pro správce, který se zatím se systémem seznamuje.

Dodávané webové prostředí obsahuje pouze funkci vyhledávání digitálních objektů, vypsání vlastností, datových proudů a seznamu jejich diseminátorů. Z nich lze po zadání potřebných parametrů získat výsledné diseminace.

Bližší objasnění používaných pojmů mi nabízela již zmiňovaná klientská aplikace, v níž jsem měl k dispozici všechny parametry digitálního objektu v jednom okně a mohl jsem zkoušet jejich změny a následné reakce. Další nezanedbatelnou službu mi poskytli pomocníci pro vytváření definic chování a jejich mechanismy. V nich lze daleko snáze zjistit, které údaje s čím souvisí, než z výsledné hromady různých XML datových proudů, jejichž „ruční“ vytváření by dalo práci.

7.2 První zjištění

Při používání dodaného klienta jsem si všiml, jak dlouho trvají jednotlivé operace, které jsou volány přes webové služby. To mě docela zarazilo. Začal jsem tedy v prostředí NetBeans vyvíjet aplikaci. Nejprve jsem neúspěšně zkoušel vytvořit rozhraní pro přístup k volání webových služeb přímo z vývojového prostředí, byla mi vždy vrácena chyba. Použil jsem poté Apache Axis, které mi dané třídy a rozhraní vytvořilo bez problémů. Mé první testy vypadaly spíše jako chaotické servlety, které spolu nespolupracovaly. Později jsem si uvědomil, že musím vytvořit určitý model, z něhož budou jednotlivé požadavky těžit informace. Typy souborů jsem navrhl jako komponenty Java Server Faces, zobrazování je realizováno pomocí tagů z Java Server Pages stránek.

První typ, který jsem vytvořil, byl adresář. Hierarchie vnořování je zajištěna pomocí vztahů mezi objekty. Zpočátku bylo zobrazování příliš zdlouhavé i při nízkém počtu obsažených položek, jež jsem získával přes webové služby. To se ukázalo jako neschůdné, musel jsem tedy získání informací předat službě RISearch. Ta mi nyní vrací i použité vlastnosti objektu a metadata DC. Vracené výsledky ale obsahují tu nectnost, že při výskytu více položek stejného typu vrací kombinace každý s každým. Máme-li tedy v digitálním objektu definovány v metadatech DC 2 tituly, 3 tvůrce a 4 popisy, vrácený výsledek bude obsahovat 24 záznamů, což způsobuje problémy nejen z hlediska pracnosti zpracování, ale též výkonu.

Každý typ má v každém pohledu způsob, kterým se vypíší informace o konkrétním objektu, je-li právě aktivní. Součástí toho je i zobrazení údajů o položkách, jež jsou v něm obsaženy (např. typ číslo článku obsahuje články,

adresář položky). K tomu může využít náhledu, který též každý pohled umí sám o sobě vygenerovat.

Dalšími typy, jež jsem implementoval, jsou články a obrázky.

7.3 Navrhované architektury implementace klienta

V této sekci popíši své návrhy různých přístupů k tvorbě klientů, kteří umožní zobrazovat a spravovat údaje z digitální knihovny.

7.3.1 Fedora jako repozitář

Podstatou tohoto typu, který jsem též použil ve své webové aplikaci, je to, že Fedoru využívám zejména jako datový sklad. Z něho získávám data pro klienta, jenž je zpracovává a odesílá ve formě HTML stránek uživateli. Způsob zobrazení je tedy zcela ponechán na klientovi, který s objekty nakládá podle identifikátoru typu, jenž je uložen ve vlastnosti contentModel. Nevyužívám tedy diseminátory. Tento typ jsem zvolil z důvodu, že jsem chtěl ulehčit serveru Fedory, který je již tak na mém počítači dost pomalý.

7.3.2 Přístup pomocí diseminátorů

Každý zobrazitelný objekt by obsahoval diseminátor, nazvěme jej třeba view. Ten by byl realizován pomocí mechanismů chování, které by byly pro každý typ objektu vytvořeny a které by všechny byly navázány na jedinou definici chování. Ta by určovala základní funkce společné všem objektům (např. ono view). Typ objektu by byl tedy dán navázáním na příslušný mechanismus chování. Klient by byl realizován jednoduše tak, že by používal jednotlivé diseminace a ty zprostředkovával uživateli. Výhodou je, že tento přístup může být využíván více různorodými aplikacemi, implementovaných v různých programovacích jazycích, a při změně dané metody bude úprava dostupná ze všech přístupových míst.

7.3.3 Implementace typů uložena přímo v digitální knihovně

Tento přístup je založen na myšlence, že by každý typ objektu měl implementován svůj plugin, který by s ním prováděl veškerou manipulaci. Ten by byl uložen ve formě JAR archivu přímo v repozitáři jako datový digitální objekt a pomocí vztahů mezi objekty navázán přímo na své instance. Klient by podle potřeby připojil daný plugin a využil jeho funkce. Výhodou je, že může být uchovávána historie verzí pluginů a objekt může mít více typů

(na to je ale potřeba myslet při vývoji klienta). Nevýhodou se jeví to, že aplikace, jež chce využít prostředky a výhody tohoto způsobu, musí být implementována v programovacím jazyce Java.

7.4 Vývoj Fedory

Při zahájení studia Fedory jsem pracoval s verzí 2.1 beta, asi po čtyřech měsících byla uvolněna 2.1 a po čtvrt roce 2.1.1, která přinesla hlavně vyladění výkonu. Mohl jsem tedy sledovat průběh vývoje systému. Podle dokumentace byla verze 2.1 přelomová. Z nejdůležitějších vlastností jmenujme alespoň tyto.

Nyní je možný výběr ze čtyř různých nastavení bezpečnosti, přidán byl systém politik, jež dokáží lepší konfiguraci přístupových práv k jednotlivým objektům, datovým proudům, diseminacím, atd. Doplněn byl také nový způsob autentikace a framework pro služby spojené s repozitářem. Z nich jsou zatím dostupné Directory Ingest pro přidávání více objektů ze zip nebo jar archivu do repozitáře a OAI Provider, který může být nastaven pro sklízení libovolných datových proudů nebo diseminací digitálních objektů v digitální knihovně).

7.5 Na co je třeba si dát pozor

Zejména ze začátku se čas od času stávalo, že *přestal fungovat index zdrojů*. Podle informací vývojářů je to způsobeno chybou v Kowari Triplestore, pomocí něhož je implementován. V tomto případě je nutné vybudovat index znovu projitím všech digitálních objektů. K tomuto účelu je přítomen skript `fedora-rebuild`. Problém ve verzi 2.1 (ve 2.1.1 je už opraven) byl ten, že při spuštění se indexace provedla z aktuálního adresáře, ne z $FEDORA_HOME. Dojde k projití všech souborů a vytvoření celého skladiště trojic znovu. Stejným způsobem lze také obnovit data v databázi.

V konfiguračním souboru serveru lze též udat *stupeň logování*, od něhož se odvíjí počet uložených záznamů. Nastavení na `finest`, při němž dochází k zápisu všech vykonaných operací, znatelně snižuje výkon systému.

Velmi podstatným problémem je fakt, že *efekty změn provedených pomocí webových služeb se projeví až po chvíli*. Při manipulaci s jednotlivými objekty na mém notebooku jsou to řádově sekundy od okamžiku, kdy skončila volání služeb. Při vložení většího počtu (více než 800 obrázků Ústavu výpočetní techniky Masarykovy univerzity) se již zpoždění počítalo v minutách. Po tuto chvilku je ještě „vidět“ starý stav. Matoucí je to v situacích, kdy

nemůžeme najít nově vytvořený objekt nebo zaznamenáme přítomnost již smazaných položek.

K modulu *indexu zdrojů* smí v jednom okamžiku přistupovat pouze *omezený počet klientů* (lze ovlivnit nastavením v souboru `fedora.fcfg`). Každému dalšímu pokusu o vyhledávání bude vrácena chyba.

V současné verzi není součástí distribuce služba, která by dokázala *indexovat obsahy datových proudů*, např. text v PDF souboru, a umožnit jejich fulltextové vyhledávání. Počítá se s jejím přidáním v příští verzi, informace o dosavadním vývoji a verzi ke stažení lze nalézt na [41].

Každý datový digitální objekt může obsahovat *diseminátory*, přes něž jsou zpřístupněny funkce libovolných služeb. V rámci realizace svého klienta jsem nenašel příležitost, při níž lze tuto možnost využít. Dynamické přepočítávání velikosti obrázků či konverzi do jiných formátů, jak je ukázáno v příkladech, přináší zbytečnou zátěž na systém a domnívám se, že je to již na knihovnu velký luxus. Uživatel si může konkrétní data stáhnout a k transformacím použít jiné programy nebo služby. V „kamenných knihovnách“ mi také nenabízejí výběr, jakým písmem má být mnou vypůjčená knihu vysázena a jak velké obrázky má obsahovat. Nejsem proti možnosti definovat diseminátory, jenom varuji před nadbytečným využíváním. Kdyby došlo při každém přístupu ke zmenšení obrázku na náhled, výrazně by vzrostly nároky na systém. Naopak zajímavým využitím, jež by stálo za vyzkoušení, by mohl být přístup popsaný v 7.3.2 na straně 58.

Zvláště zpočátku jsem měl problémy s výkonem na mém notebooku. Operace trvaly dle mého názoru dlouho. Při sledování síťového provozu mě zaskočilo, že server pokládal databázi několikrát po sobě stejný dotaz.

Velkou překážkou při práci s digitálními objekty je *nemožnost zjistit* prostřednictvím webových služeb ani jiného dostupného rozhraní, jaká *přístupová práva* k nim přihlášený uživatel vlastní.

Distribuce obsahuje též nástroj pro editaci nastavení politik *Policy Editor*. Ten se mi však v žádném prostředí Javy nepodařilo spustit, aplikace vždy skončila výjimkou nebo narušila běh samotného běhového prostředí. Chybu se mi nepodařilo odhalit.

7.6 Dostupné systémy

Podle poznámek a odkazů ze stránek Fedory (viz [27]), lze vytušit existenci velkého množství digitálních knihoven postavených na Fedoře, jež jsou dostupné prostřednictvím celosvětové sítě Internet. Jen velmi málo z nich však tuto informaci zveřejňuje a tak není jisté, která část (jestli vůbec nějaká) je

pomocí Fedory realizována. U jiných byl naopak zablokován přístup z míst, z nichž jsem byl připojený, nebo byl omezen na určitou skupinu uživatelů, zejména studentů a zaměstnanců univerzit. Bohužel se mi nepodařilo získat vyjádření oslovených vývojářů k jejich zkušenostem s touto digitální knihovnou a k implementaci jejich klienta.

Kapitola 8

Závěr

V této práci jsem se snažil popsat aktuální verzi digitálního repozitáře Fedora. Nebylo možné věnovat se všem podrobnostem do detailů, neboť jde o velmi rozsáhlý systém. Zaměřil jsem se tedy na nejdůležitější charakteristiky a snažil se představit novému uživateli, jak je lze využít.

Součástí práce byla i praktická ukázka možnosti, jak implementovat webového klienta, jenž by v uživatelsky příjemném a intuitivním prostředí zobrazoval objekty v „lidsky čitelné“ formě. Na uvedené webové aplikaci se mi podařilo ukázat poměrně nenáročnou realizovatelnost tohoto rozhraní.

Otevřenou otázkou zůstává, který způsob tvorby klienta poskytuje nejvýhodnější výsledky. Tři možné přístupy jsem naznačil v podkapitole 7.3 na straně 58. Zajímavé bude jistě porovnání se systémem FIRE, který vyvíjí přímo tvůrci Fedory.

Při realizaci jsem zjistil, že Fedora je dostatečně flexibilní a mocný nástroj, který má budoucnost. Jeho vývoj není zatím u konce, s každou novou verzí přichází nové nástroje, služby a zvýšení výkonu. Doporučuji vývojářům, kteří chtějí vybudovat systém pro správu rozsáhlých sbírek různorodých objektů, aby věnovali tomuto systému pozornost.

Literatura

[1] Miroslav Bartošek. *Digitální knihovny*. Proc. Datakon 2001. http://www.ics.muni.cz/mba/dl-datakon01.pdf

[2] Jaroslav Pokorný. *Digitální knihovny: Principy a problémy*. Automatizace knihovnických procesů (8), duben 2001. http://knihovny.cvut.cz/akp/clanky/03.pdf

[3] Digital Library Definitions. http://web.simmons.edu/~schwartz/462-defs.html

[4] Fedora License. http://fedora.info/download/2.1/userdocs/distribution/license/license.html

[5] Mozilla Public License 1.1. http://www.mozilla.org/MPL/MPL-1.1.html

[6] Dublin Core Metadata Initiative. http://dublincore.org

[7] Dublin Core – Czech homepage. http://www.ics.muni.cz/dublin_core/

[8] Open Archives Inititative. http://www.openarchives.org/

[9] „info“ URI Scheme. http://info-uri.info/registry/OAIHandler?verb=ListRecords&metadataPrefix=oai_dc

[10] Apache Axis. http://ws.apache.org/axis/

[11] Dublin Core. http://dublincore.org/

[12] Library of Congress, Core Metadata Elements. http://www.loc.gov/standards/metadata.html

[13] RDF, Resource Description Framework. http://www.w3.org/RDF/

[14] METS, Metadata Encoding & Transmission Standard. http://www.loc.gov/standards/mets/.

[15] URIs, URLs, and URNs: Clarifications and Recommendations 1.0. http://www.w3.org/TR/uri-clarification/

[16] PURL, Persistent URL. http://purl.oclc.org/

[17] CNRI, Corporation for National Research Initiatives. Handle System. http://www.handle.net/

[18] DOI, The Digital Object Identifier System. http://www.doi.org/

[19] ARK, Archival Resource Key. http://www.cdlib.org

[20] OAI-PMH, Open Archives Initiative – Protocol for Metadata Harvesting. http://www.openarchives.org/OAI/openarchivesprotocol.html

[21] OpenURL. http://library.caltech.edu/openurl/

[22] Robert Kahn, Robert Wilensky. *A Framework for Distributed Digital Objects Services*. May 13, 1995. cnri.dlib/tn95-01. http://www.cnri.reston.va.us/home/cstr/arch/k-w.html

[23] William Y. Arms. *Key Concepts in the Architecture of the Digital Library*. July 1995. cnri.dlib/july95-arms http://www.dlib.org/dlib/July95/07arms.html

[24] DNS, Domain Name System. http://www.dns.net/dnsrd/

[25] Vannevar Bush. *As We May Think*. Atlantic Monthly, July 1945 http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm

[26] Digital Library Initiative. http://www.dli2.nsf.gov/

[27] Fedora, Flexible Extensible Digital Object and Repository Architecture. http://www.fedora.info/

[28] Sandra Payette, Carl Lagoze. *Flexible and Extensible Digital Object and Repository Architecture.* http://www.cs.cornell.edu/payette/papers/ECDL98/FEDORA.html

[29] CORBA, Common Object Request Broker Architecture. http://www.corba.org

[30] Web Services. http://www.w3.org/2002/ws/

[31] Web Services Description Language. http://www.w3.org/TR/wsdl

[32] SOAP, Simple Object Access Protocol. http://www.w3.org/TR/soap12-part1/

[33] Building Web Services the REST Way. http://www.xfront.com/REST-Web-Services.html

[34] XML.com: What Is RDF. http://www.xml.com/pub/a/2001/01/24/rdf.html?page=1

[35] Resource Description Framework - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Resource_Description_Framework

[36] SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/

[37] Namespaces in XML. http://www.w3.org/TR/REC-xml-names/

[38] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[39] Representational State Transfer - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Representational_State_Transfer

[40] Cascading Style Sheets, level 2 revision 1. http://www.w3.org/TR/CSS21/

[41] Fedora Generic Search Service Development. http://defxws2006.cvt.dk/fedoragsearch/

[42] REST Client Demo of Fedora Generic Search Service. http://defxws2006.cvt.dk/fedoragsearch/rest

[43] NetBeans. http://www.netbeans.org/

[44] Java. http://java.sun.com/

[45] MySQL Database Machine http://www.mysql.com/

[46] Apache Ant. http://ant.apache.org/

[47] Jakarta Tomcat. http://jakarta.apache.org/tomcat/

[48] MIME Media Types. http://www.iana.org/assignments/media-types/

[49] Marty Hall. *Java servlety a stránky JSP*. Neocortex, 2001. ISBN 80-86330-06-0

[50] FOP, Formatting Objects Processor. http://xml.apache.org/fop/

[51] XSL-FO, XSL Formatting Objects. http://www.w3.org/Style/XSL/

[52] ImageJ, Image Processing and Analysis in Java. http://rsb.info.nih.gov/ij/

[53] The Fedora Project, An Open-source Digital Object Repository Management System http://www.dlib.org/dlib/april03/staples/04staples.html

[54] DKF, Digitální knihovna fotografií na ÚVT MU. http://dkf.ics.muni.cz/

[55] Zpravodaj ÚVT MU http://www.ics.muni.cz/zpravodaj/

Příloha A

Instalace serveru Fedory

V této kapitole jsou popsány kroky, které je potřeba vykonat pro instalaci serveru digitální knihovny Fedora verze 2.1.1 tak, aby byla použitelná s mou implementací webové aplikace, jejíž vytvoření je součástí mé diplomové práce.

Fedora je napsána v programovacím jazyku Java, proto je nutné mít nainstalovanou J2SDK (Java 2 Software Development Kit) verze alespoň 1.4.2. Ta je k dispozici ke stažení na stránkách firmy Sun [44].

Binární distribuci serveru digitální knihovny lze stáhnout ze stránek Fedory [27] (k dispozici jsou též zdrojové kódy, překlad je řízen systémem Ant – je též přítomen v balíku). Její verze 2.1.1, která je popisována v této práci, je šířena pod licencí Educational Community License (viz [4]).

Prvním krokem je rozbalení získaného archivu (k dispozici jsou formáty tar.gz a zip) do určeného adresáře (např. `/opt`). Adresář `fedora-2.1.1`, který vznikne, označme *domovským adresářem* Fedory. Prostřednictvím prostředků použitého operačního systému nastavíme proměnné prostředí:

- `JAVA_HOME` – základní adresář instalace J2SDK, pokud ji používáme pro jiné programy a nechceme nebo nemůžeme ji změnit, lze ji nahradit `FEDORA_JAVA_HOME` (to neplatí pro průběh instalace ze zdrojových kódů).
- `FEDORA_HOME` – absolutní cesta do domovského adresáře Fedory (např. `/opt/fedora-2.1.1`)
- `PATH` – musí obsahovat adresář `$FEDORA_HOME/server/bin`

Skriptům v adresáři `$FEDORA_HOME/server/bin` je potřeba v prostředí unixového operačního systému nastavit právo spouštění.

Fedora používá pro uchovávání dat relační databázi. Samotná distribuce obsahuje přímo čistě javovský databázový server McKoi, má však podporu také pro MySQL a Oracle9i. Lze použít i jiné databáze, je však třeba je ručně nakonfigurovat.

Pro nastavení McKoi postačuje spustit příkaz `mckoi-init mckoiUser mckoiPasswd`, v němž parametry mají po řadě význam uživatelského jména a hesla pro přístup k databázi. Ty je potom nutné nastavit v konfiguračním souboru Fedory. Databáze se spouští příkazem `mckoi-start`, vypíná `mckoi-stop`.

Pro MySQL je potřeba vybrat uživatele s právy vytvářet databáze, popř. vytvořit uživatele nového. Příkaz `mysql-config installDir dbaUser dbaPass fedoraUser fedoraPass dbName mysql41_flag`. Jednotlivé parametry mají následující významy:

- *installDir* – adresář, v němž je naistalováno MySQL
- *dbaUser* – vybraný uživatel MySQL
- *dbaPass* – heslo uživatele MySQL
- *fedoraUser* – jméno uživatele Fedory (fedoraAdmin)
- *fedoraPass* – heslo uživatele Fedory (fedoraAdmin)
- *dbName* – jméno databáze, která bude vytvořena
- *mysql41_flag* – tento parametr může nabývat libovolné hodnoty a jeho přítomnost určuje, že verze MySQL je 4.1.x, příkaz nastaví kódování na utf-8

Pokud v budoucnosti dojde k poškození databáze nebo k instalaci nové verze, je možné vytvořit v ní znovu potřebná data projitím uložených XML definic objektů příkazem `fedora-rebuild`.

Nyní se rozhodneme pro jednu z možností nastavení bezpečnosti – ssl-authenticate-apim, ssl-authenticate-all, no-ssl-authenticate-apim, či no-ssl-authenticate-all. Bližší popis je uveden v sekci 4.14 na straně 30. Tuto hodnotu předáme jako parametr skriptu `fedora-setup`. Při pozdějších spuštěních musíme počítat s tím, že dojde k přepsání konfigurace serveru implicitními hodnotami a budeme muset znovu provést přizpůsobení konkrétním podmínkám.

Nyní přejděme ke změně souboru s nejdůležitějšími nastaveními. Nachází se v `$FEDORA_HOME/server/config/fedora.fcfg`, kam byl skriptem nakopírován v minulém kroku. Soubor je přehledný a obsahuje popis jednotlivých položek nastavení. Z nejdůležitějších pro nás jmenujme následující.

repositoryName – název repozitáře

adminUsername – uživatelské jméno administrátora, implicitní hodnotou je `fedoraAdmin`

adminPassword – heslo administrátora, implicitně `fedoraAdmin`

fedoraServerPort – číslo portu, na němž poběží server

fedoraShutdownPort – číslo portu pro ukončení běhu serveru

fedoraRedirectPort – číslo portu přesměrování

fedoraServerHost – jméno počítače, na němž server běží, které bude vidět z vnějšího světa, v případě použití v síti je potřeba nastavit na správnou hodnotu, ne na `localhost` nebo `127.0.0.1`.

logLevel – úroveň logování, podstatně ovlivňuje výkon systému

Modul fedora.server.storage.DOManager – Rozhraní k systému skladování digitálních objektů

- **pidNamespace** – implicitní jmenný prostor PID, z něhož budou přiřazovány nové identifikátory, nebudou-li specifikovány vkladatelem
- **retainPIDs** – jmenné prostory PID, z nichž může vkladatel vybírat, ostatní budou zamítnuty
- **storagePool** – pojmenované spojení na databázový server

Modul fedora.server.search.FieldSearch – zajišťuje jednoduché i pokročilé vyhledávání

- **maxResults** – maximální počet záznamů, který je vrácen najednou jako výsledek vyhledávání
- **maxSecondsPerSession** – doba v sekundách, po kterou lze získat další záznamy vyhledávání

Modul fedora.server.resourceIndex.ResourceIndex – udržuje data RDF

- **level** – určuje úroveň indexace, *pro funkčnost mého webového klienta musí být nastavena aspoň na 1*
  - **0** – indexace je vypnuta
  - **1** – systémová metadata, RELS-EXT, diseminace
  - **2** – navíc permutace metod a jejich parametrů

Při každé změně (mimo nastavení na 0) je nutno znovu vytvořit index skriptem `fedora-rebuild`.

Datové sklady – pro každé spojení existuje vlastní element `datastore`

dbUsername – uživatelské jméno do databáze

dbPassword – heslo do databáze

Zejména v adresářích
`$FEDORA_HOME/data`, `$FEDORA_HOME/server/logs`,
`$FEDORA_HOME/server/jakarta-tomcat-5.0.28/logs`,
`$FEDORA_HOME/server/jakarta-tomcat-5.0.28/temp`,
`$FEDORA_HOME/server/jakarta-tomcat-5.0.28/work`
a jejich podadresářích a v souboru `$FEDORA_HOME/server/status` musí mít aplikace práva zapisovat.

Nyní již stačí mít zapnutý databázový server a spustit Fedoru příkazem `fedora-start`. Po skončení práce uživatel vypne digitální knihovnu pomocí `fedora-stop`.

Příloha B

Instalace webového klienta

Instalace mnou vytvořeného klienta je snadná. Pro svůj běh potřebuje webový kontejner a nainstalované běhové prostředí Javy (viz [44]). Aplikace byla vytvářena a testována v prostředí Jakarta Tomcat, kterého se budu držet i v následujícím návodu. Pro instalaci v jiném webovém kontejneru se řiďte příslušnou dokumentací.

Nejprve je nutné nainstalovat Jakarta Tomcat. Ten je šířen pod open-source licencí a jeho distribuci lze získat na stránkách [47].

Po rozbalení příslušného archivu přejdeme do adresáře `conf` a v textovém editoru otevřeme soubor `server.xml`. Nás zajímá element zejména `Connector` definující požadované spojení – buď obyčejné HTTP[1], nebo šifrované HTTPS[2]. Jedinou změnu, kterou je nutno provést, je přidat mezi atributy `URIEncoding` s hodnotou `UTF-8`. Jsou-li obsazeny implicitní porty, je však potřeba též změnit je na volná čísla. Pro účely tohoto návodu předpokládejme využití standardních hodnot – 8080 pro HTTP a 8443 pro HTTPS. Chceme-li použít šifrované spojení, musíme vytvořit klíč šifry. Využít k tomu můžeme příkaz z distribuce Javy `keytool -genkey -alias tomcat -keyalg RSA`, jako heslo zadáme `changeit`. Pokročilejší způsoby konfigurace jsou popsány v dokumentaci Tomcatu.

Nyní musíme přidat uživatele s rolí manager. Ve stejném adresáři otevřeme soubor `tomcat-users.xml`. Mezi definice uživatelských účtů přidáme následující řádek.

```
<user username="manager" password="tajneheslo"
                              roles="manager"/>
```

Nyní ze svého WWW prohlížeče přistoupíme k manažeru Tomcatu na adrese[3] `http://localhost:8080/manager/html`, resp. `https://`

[1] Element za komentářem `<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->`.

[2] Element za komentářem `<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->`. Je nutné odkomentovat ho.

[3] Pokud jsme změnili porty, je nutno je v uvedených adresách nahradit.

`localhost:8443/manager/html` pro HTTPS. Přihlásíme se jako uživatel, kterého jsme vytvořili v předchozím kroku[4].

Vyhledejme sekci *Deploy* a ve druhém formuláři *War file to deploy* u položky *Select WAR file to upload* vybereme soubor `Diplomka.war` z distribuce klienta. Mezi spuštěnými aplikacemi by se měl objevit *Fedora Repository Client* pod umístěním `/Diplomka`.

Před samotným přístupem ke klientovi je nutno nahrát jeho nastavení. Adresář, do něhož musíme zkopírovat adresářovou strukturu, je určen parametrem v souboru `web.xml` v samotné aplikaci. Implicitní hodnota je `~/.rc2006`[5]. V adresáři `objects` jsou definovány typy digitálních objektů, které umí klient obsloužit. Soubor `truststore` obsahuje informace potřebné k přístupu k serveru Fedory přes šifrování SSL. V souboru `config/config.xml` musíme nadefinovat přístupové body k digitální knihovně. Těch může být více, budou nabídnuty uživateli při procesu přihlašování. V kořenovém elementu `config` přidáme pro každý tento bod element `repository` a v něm `URL` s URL adresou, na níž běží Fedora, a `name` s názvem.

Nyní spustíme server Fedory a zadáme ve WWW prohlížeči adresu klienta[6] `http://localhost:8080/Diplomka`, respektive `https://localhost:8443/Diplomka`. Přihlásíme se pod administrátorským účtem Fedory[7]. Aplikace ohlásí chybu, že v repozitáři chybí kořenový objekt, a umožní ho definovat. Po jeho vytvoření (pokud je znovu ohlášen chybějící kořenový objekt, je nutné počkat několik sekund, než proběhne indexace, a poté ho zkusit znovu vytvořit) je instalace dokončena a můžeme zahájit práci.

[4] Přihlašovací jméno: `manager`, heslo: `tajneheslo`

[5] Je-li prvním znakem vlnovka ~, nahradí se za domovský adresář právě přihlášeného uživatele.

[6] Pokud jsme změnili porty, je nutno je v uvedených adresách nahradit.

[7] Implicitní administrátor Fedory je `fedoraAdmin` s heslem `fedoraAdmin`.