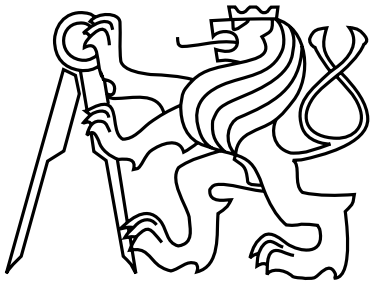
Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Ruby - Access Control List

Jan Širl

Vedoucí práce: Ing. Pavel Strnad

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

15. května 2012

Poděkování

Chtěl bych především poděkovat panu Ing. Pavlu Strnadovi

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 25. 5. 2012 .....

Abstract

This paper presents the design and implementation of library management access rights in the programming language Ruby.

The library is designed for object database CellStore and is solved using the Access Control List. Library used for storing and querying the database itself, which communicates with the protocol XML-RPC (Remote Procedure Call) XQuery and XQuery Update Facility. The functionality of the library was developed and tested on eXistDB.

Abstrakt

Tato práce prezentuje návrh a realizaci knihovny pro správu přístupových práv v programovacím jazyku Ruby.

Knihovna je určena pro objektovou databázi Cellstore a je řešena pomocí Access Control List. Knihovna využívá pro ukládání a dotazování samotnou databázi, se kterou komunikuje pomocí protokolu XML-RPC (Remote Procedure Call) technologií xQuery a xQuery Update Facility. Funkčnost knihovny byla vyvíjena a testována na eXistDB.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Úvod do problematiky

Tato bakalářská práce navazuje na můj semestrální projekt. Zabývá se správou - administrací řízení přístupu, jakožto procesu autorizování uživatelů, skupin a počítačů pro přístup k objektům. Tento proces pracuje s pojmy: oprávnění, uživatelská práva a audit objektů. Tato jednotlivá přiřazená oprávnění začleňuje jako položky řízení přístupu (ACE) a jejich celé sady začleňuje do seznamu přístupových práv (ACL). Úkolem bakalářské práce bylo vytvořit, navrhnout a realizovat v jazyce Ruby model uživatelských přístupových práv určenou pro objektovou xml databázi.

Součástí práce byl návrh knihovny. Navržený modul realizuje správu řízení přístupu pomocí ACL (Access Control List). Protože moje bakalářská práce je prací implementační, včetně testů navrženého modulu, zaměřil jsem se na specifikaci rozhraní knihovny a na příklady jejího použití.

Výsledkem bakalářské práce je nejen samotná realizace knihovny, ale i podrobná programátorská dokumentace.

V druhé části úvodu je vysvětlení nejdůležistějších pojmů.

Chcete-li zabezpečit zdroje a jeho prostředky, je nutné vzít v úvahu, jakými právy budou disponovat ti, kteří k nim budou přistupovat. Zabezpečit objekt, například dokument či kolekci, lze přidělením oprávnění, která umožňují uživatelům nebo skupinám provádět u tohoto objektu určité akce. Řízení přístupů je činnost zabývající se přidáváním a zamítáním oprávnění přistupujícím.

Ruby je interpretovaný skriptovací programovací jazyk. Díky své jednoduché syntaxi je poměrně snadný, přesto však dostatečně výkonný, aby dokázal konkurovat známějším jazykům jako je Python a Perl. Je objektově orientovaný.

ACL je zkratka pro access control list, což v překladu znamená seznam pro řízení přístupů. ACL je v oblasti počítačové bezpečnosti seznam oprávnění připojený k nějakému objektu. Seznam určuje, kdo nebo co má povolení přistupovat k objektu a jaké operace s ním může nebo nesmí provádět. V typickém ACL specifikuje každý záznam v seznamu uživatele a operaci.

1.2 Motivace

Zaujala mě problematika práv, databází a pro mě neznámého jazyka. Jádro celé mé motivace, bylo projít si vývojem softwaru, v tomto případě knihovny, od návrhu přes implementaci k testování a dokumentaci a ověřit si získané poznatky z předmětu softwarového inženýrství. Soustředil jsem se na vlastní nápady. Nechtěl jsem skládat a kopírovat polotvary a "lepit"je dohromady.

Kapitola 2

Popis problému, specifikace cíle

- Popis řešeného problému, vymezení cílů DP/BP a požadavků na implementovaný systém.
- Popis struktury DP/BP ve vztahu k vytyčeným cílům.
- Rešeršní zpracování existujících implementací, pokud jsou známy.

2.1 Popis řešeného problému

Určená XML databáze v současné době nemá žádný model uživatelských přístupových práv. Bylo potřeba tento nedostatek vyřešit knihovnou implementovanou v jazyce Ruby. Jazyk Ruby byl vybrán proto, že Ruby je rozšířený jazyk a z důvodu kompatibilty s budoucími částmi databáze, které budou taktéž naimplementované v Ruby. Mnou naimplementovaná knihovna RubyACL řeší problém se spravováním přístupových práv.

2.2 Vymezení cílů a požadavků

Cílem bakalářská práce bylo navrhnout, realizovat a otestovat knihovnu v jazyce Ruby, která bude spravovat uživatelská přístupová práva pro objektovou XML databázi. Vytyčil jsem si vytvořit co nejjednodušší knihovnu, která by splňovala všechna kritéria zadání. Nechtěl jsem jsem používat a "lepit"dohromady existující moduly a knihovny, které danou problematiku řeší, protože jsem si za cíl dal vytvořit něco svého a projít si vývojem softwaru od požadavků, analýzu, návrh přes realizaci k testování a dokumentaci. I když Ruby ACL je primárně určena pro XML databázi chtěl jsem, aby byla použitelná i pro jiné databáze a reálné přístupy do budov apod. Určil jsem si, že by bylo pěkné, kdyby knihovna umožňovala jemně nastavit přístupy (Tzv.„fine-grained“).

Seznam požadavků je popsán v tabulce 2.1

2.3 Popis struktury bakalářské práce ve vztahu k vytyčeným cílům

V kapitole Analýza a návrh řešení je analýza a návrh tak jsem je vytvořil před implementací. Zato kapitola realizace popisuje realizaci a stav po realizaci a dokumentační část popisující pravidla rozhodování apod.

V analýze je analýza... V realizaci je realizace... V testování je testování...

Co se týče vytyčených cílů, nejpodstatnější kapitolu a části bakalářské práce je kapitola Analýza sekce Rozhraní a sekce Příklady užití a dále kapitola Testování.

Kapitola 1 nás uvádí do Bakalářské práce. Vysvětluje, co vlastně řízení práv je, popisuje jeho význam a vysvětluje nejdůležitější pojmy týkající se bakalářské práce a řízení práv.

V kapitole 2 se hovoří o důvodu implementace RubyACL, vytyčují se cíle a prezentují požadavky. Dále je stručný popis existujících řešeních.

Kapitola 3 je nejpodstatnější z celé práce. Jedná se o Analýzu, ve které bylo popsáno z čeho jsem vycházel při návrhu. Nejdůležitější část je Rozhraní a Ukázka použití, protože přímo splňují zadání/požadavky práce. Sekce Popisu logiky vyhodnocování je důležitá pro uživatele knihovny. Nejen že vysvětluje pojmy jako ACL objekt, principal, ale hlavně popisuje jakým způsobem knihovna Ruby-ACL rozhoduje, které pravidlo má přednost.

Kapitola 4 Realizace se zaměřuje na postup vývoje a problémy při implementaci.

Kapitole 5 je rozdělená na dvě části. Jedna se zabývá pomocným komunikačním rozhraním eXistAPI a druhá samotnou knihovnou Ruby-ACL. Obě části jsou však zaměřeny vysvětlení způsobu testování a jejich výsledky.

Kapitola 6 se zaměřuje na zhodnocení splnění cílů Bakalářské práce a rozebírá možné nedostatky a případné pokračování v práci na knihovně.

V příloze se nacházejí diagramy, které nebyly potřeba pro vysvětlení funkcionality knihovny.

2.4 Existující řešení

Existující řešení jsem rozdělil na dvě části. První jsou podnikové a druhé jsou knihovny v Ruby zabývající se stejnou nebo podobnou problematikou.

2.4.1 Podnikové řešení

Vybral jsem tři nejukázkovější řešení. Oracle má nejpropracovanější model řízení práv. Podporuje integrování LDAP a DAV. PhpGACL je bezplatný jednodušší systém spravující přístupy ke zdrojům. Obecné řešení se skládá z dvojrozměrné tabulky, kde jeden rozměr je tvořen všemi, kdo vyžadují přístup a druhý rozměr obsahuje objekty, ke kterým je vyžadován přístup. Oprávnění je v buňce, která se nachází na prusečíku os zminěných dvou rozměrů. Podrobnější zpracování se nachází v sekci 3.1.1 Existující řešení kapitoli Analýza.

id	Specifikace požadavků na software	priorita
FUNKČNÍ POŽADAVKY		
0	RubyACL bude umožňovat řízení přístupů pomocí ACL	povinný
1.0	RubyACL bude umožňovat definovat, editovat a mazat zmocnitele (principals)	povinný
1.1	RubyACL bude umožňovat definovat, editovat a mazat oprávnění (privileges)	povinný
1.2	RubyACL bude umožňovat definovat, editovat a mazat zdrojové objekty (resource objects)	povinný
1.3	RubyACL bude umožňovat definovat, editovat a mazat pravidla přístupu (ACE)	povinný
2.0	RubyACL bude umožňovat vytvářet ACL	povinný
2.1	RubyACL bude umožňovat načítat a ukládat ACL z a do XML souboru	povinný
2.2	XML soubor bude definovaný pomocí DTD or XML Schema	povinný
2.3	XML soubor bude "well formated"podle W3C doporučení	povinný
3.0	RubyACL bude nabízet pouze Default-Deny politiku	povinný
4.0	RubyACL bude testována na eXist DB	povinný
OBECNÉ POŽADAVKY		
1.0	RubyACL bude naprogramována v jazyce Ruby	povinný
2.0	RubyACL bude vydaná jako RubyGem	volitelný
3.0	RubyACL potřebuje databázi podporující xQuery, xPath technologie	povinný

Tabulka 2.1: Tabulka funkčních a obecných požadavků. priorita = (povinný, volitelný, nepovinný)

2.4.2 Dostupné knihovny

Kdybych si za cíl nedal vytvořit RubyACL sám, nejspíše bych čerpal z jedné z těchto knihoven.

TODO pridat reference, ze bylo prevzato z webu.

2.4.2.1 Acl9

Acl9 je další řešení autorizace založené na rolích v Rails. Skládá se ze dvou subsystémů, které mohou být použity samostatně. Subsystém kontroly rolí umožňuje nastavovat a dotazovat se na uživatelské role pro různé objekty. Subsystém řízení přístupu umožňuje zadat uvnitř řadiče různá přístupová pravidla podle rolí.

2.4.2.2 iq-acl

Cílem tohoto rubygemu je poskytnout serii tříd, které umí zacházet s bežnýma požadavkama na řízení práv. V současné době poskytuje třídu IQ::ACL::Basic, která přestože je velmi jednoduchá je také velmi schopná.Více o použití se můžete dočíst zde. TODO dělat odkaz, kde lze více precist.

2.4.2.3 ActiveACLPlus

Plugin ActiveAclPlus realizuje flexibilní, rychlý a snadno použitelný obecný systém kontroly přístupu. Dosud nejsou žádná skutečná měřítka. Systém je založen na phpgacl.sourceforge.net, přidání objektu orientace, polymorfismu a dvou úrovní cache. PhpGacl tvrdí, že v reálné pracovní verzi s mnoha přidanými vrstvami složitosti podporuje více než 60.000 účtů, 200 skupin a 300 ACO."Testy provedené na vyvojářském notebooku ukazují 10 - 30 krát zlepšení výkonnosti ve srovnání s `active _rbac`. Plugin používá ukládání do mezipaměti. Používá instanční cache a případě potřeby ukládá výsledky oprávněních do memcached použitím časového omezení.

Kapitola 3

Analýza a návrh řešení

Analýza a návrh implementace (včetně diskuse různých alternativ a volby implementačního prostředí).

Tato kapitola se dělí na analýzu a návrh. V analýze se zaměřilo na prostudování tří existujících řešení. Z informací získaných z dokumentace jsem sestavil návrh pro Ruby ACL.

3.1 Analýza

Protože práce byla velmi přesně zadána, nezbylo příliš prostoru na různé alternativy. Ruby bylo zadáno jako implementační prostředí. Způsob zpracování byl zadán pomocí ACL - Access Control List. Hlavním úkol bylo zjistit, jakým způsobem implementovat samotné ACL a řádně implementaci zdokumentovat a otestovat.

3.1.1 Exitující řešení

Při řešení vlastního návrhu na model řízení přístupových práv jsem vycházel ze dvou zdrojů – známých řešení a jednoho obecného řešení.

3.1.1.1 Obecné řešení

Obecným řešením je držet si tabulku, kde ve sloupcích budou objekty, ke kterým je možno přistupovat a v řádcích jsou přistupující. V poli pak jsou hodnoty boolean, které vyjadřují buď allow nebo deny. Příklad tabulkového řešení je v tabulce 3.1.

Who / Where	Surgeryroom	ambulance	Patient's room
Doctors	1	1	1
Nurses	X	1	1
Patients	X	X	1

Tabulka 3.1: Obecné řešení modelu práv pomocí matice

3.1.1.2 Oracle

Jedním z řešení je podnikové řešení prezentované v Oracle® XML DB Developer's Guide, 11g Release 1 (11.1), Part Number B28369-04 na stránkách Oracle dokumentace [1]. Ve stati Access Control Lists and Security Classes je popsán koncept firmy ORACLE.

Text popisuje několik podmínek a pojetí řízení přístupu. Každá z popsaných entit, uživatel, role, privilegia, bezpečnostní třídy, Access Control List (ACL) a Access Control Entry (ACE), je realizována deklarativně jako XML dokument nebo fragment.

Bezpečnostní autorizace vyžaduje definovat, kteří uživatelé, aplikace nebo funkce mohou mít přístup k jakým datům nebo jaké druhy operací mohou provádět. Existují tedy tři dimenze: (1) kteří uživatelé mohou, (2) vykonávat jaké činnosti, (3) na jakých datech. V souvislosti s každou jednotlivou dimenzí hovoříme o (1) principals - zmocnitelích, (2) privileges - oprávněných, a (3) objektech, které korespondují s těmito třemi dimenzemi. Principals mohou být uživatelé nebo role/skupiny.

Principals a privileges (dimenze 1 a 2) jsou deklarativním způsobem spojeni v definovaných seznamech řízení přístupu - ACL. Ty jsou pak spojené s třetí dimenzí - daty, různými způsoby. Například úložiště zdrojů nebo tabulky dat Oracle XML DB mohou být ochráněny pomocí PL / SQL procedury DBMS_XDB.setACL nastavením jeho řídícího ACL.

3.1.1.3 phpGACL

Druhým ze zdrojů, z nichž jsem vycházel, je řešení prezentované v Generic Access Control List with PHP - phpGACL na [2].

Nástroj phpGACL je sada funkcí, která umožňuje použít řízení přístupu na libovolné objekty (webové stránky, databáze, atd.), jiným libovolným objektům (uživatelé, vzdálené počítače, atd.). Stejně jako Oracle nabízí jemně nastavitelnou kontrolu přístupu s jednoduchou a velmi rychlou správou. Je napsán v populárním dynamickém skriptovacím jazyku PHP.

Nástroj phpGACL vyžaduje relační databáze pro ukládání informací k řízení přístupu. Přistupuje k databázi prostřednictvím tzv. abstraktního obalu ADOdb. Je kompatibilní s databázemi, jako PostgreSQL, MySQL a Oracle.

Nástroj phpGACL používá pojmy jako ACO a ARO:

- Access Control Objects (ACO), jsou věci, ke kterým chceme ovládat přístup (např. webové stránky, databáze, pokoje, atd.).
- Access Request Objects (ARO), jsou věci, které žádají o přístup (např. osoby, vzdálené počítače, atd.)
- ARO stromy definují hierarchii skupin a ARO. Skupiny mohou obsahovat jiné skupiny a ARO.
- Výchozí "catch-all"politikou stromu ARO je vždy "DENY ALL ".
- Chceme-li přiřadit přístupovou politiku ve stromu směrem dolů, explicitně přiřazujeme oprávnění skupinám a ARO pro každou ACO, pro kterou je potřeba.

3.2 Návrh implementace

Nechal jsem se inspirovat jak Oraclem tak phpGACL. Oba modely řízení přístupových práv mají podobnou strukturu nebo stejnou s jiným pojmenováním. Z Oraclu jsem převzal koncept a pojmenování dimenzí: principals, privileges, objects, ze kterých jsem vytvořil hlavní třídy.

Jemně nastavitelných přístupových práv se docílí pomocí Access Control Listu. ACL obsahuje seznam pravidel jednotlivých přístupů. Pravidlo se nazývá Access Control Entry (ACE). V ACE je uloženo kdo, nebo co má jaká práva přistupovat k jakým objektům. Těmto třem rozměrům se v problematice přístupových práv říka: principals, privileges, resource objects.

3.2.1 Rozhraní

S knihovnou RubyACL je zapotřebí nějak komunikovat. Za účelem vytvoření a nastavení zmoctnitelů, oprávněních, zdrojových objektů a pravidel musí uživatel předat knihovně potřebné vstupy popsané v tabulce. Hlavní účel knihovny je rozlišit jestli někdo nebo něco má oprávnění na nějaký cílový objekt. Pokud zmocnitel má nebo nemá přístup se uživatel dozví podle výstupu. Výstupem je true nebo false hodnota. 3.2 TODO přidat do tabulky i popis Model rozhraní se nachází v příloze jako obrázek **??**

3.2.1.1 Rozhraní mezi Ruby-ACL a databází

RubyACL bylo testováno s databází eXist-db, se kterou komunikovalo pomocí eXistAPI. K používání RubyACL s jinou XML databází než eXist je zapotřebí komunikační rozhraní, které nahradí eXistAPI. V ukázkové třídě API je výčet všech potřebných metod, které RubyACL používá, včetně parametrů a výstupů. Je potřeba podle vzorové třídy API naimplementovat nové komunikační rozhraní.

3.2.1.2 Aplikace > RubyACL

Jedná se o výčet public metod třídy RubyACL. ...přidat výčet Na této části záleží bezpočenost. Potencionálně nebezpečné místo k útoku.

jméno	**typ**
Vstupy	
principal	string
privilege	string
resource object	string
Výstupy	
access	boolean

Tabulka 3.2: Vstupy a výstupy knihovny

3.2.2 Ukázka použití

Tato sekce prezentuje stručné ukázky použití

3.2.2.1 Příklad nastavení práv

Příklad tohoto kódu v podstatě říká: Pokud metoda `ac_check` vratí hodnotu true, přístup povolen a aplikace provede co zamýšlela provést. Pokud vrátí hodnotu false, tak je přístup zamítnut. Tento postup názorně vysvětluje obrázek 3.2 zobrazující komunikaci a průběh jednoho dotázení na pravidlo.

```
require 'eXistAPI'    #must require 'eXistAPI' to comunicated with eXist-db
#create instance of ExistAPI
@db = ExistAPI.new("http://localhost:8080/exist/xmlrpc", "admin", "password")
@col_path = "/db/test_acl/"          #sets the collection where you want to have ACL in db
@src_files_path = "./src_files/"     #path to source files
@my_acl = RubyACL.new("my_acl", @db, @col_path, @src_files_path, true)
#it's good to create some principals at the begging
@my_acl.create_principal("Sheldon")
@my_acl.create_privilege("SIT")
@my_acl.create_resource_object("seat", "/livingroom/couch/Sheldon's_spot", "Sheldon")
@my_acl.create_ace("Sheldon", "allow", "SIT", "seat", "/livingroom/couch/Sheldon's_spot")
```

3.2.2.2 Příklad kontroly práv

Druhý příklad naznačuje použití metody pro vytvoření pravidla. Pro vytvoření pravidla musí existovat instance Ruby ACL. Metodě create_ace aplikace předá uživatelské jméno zmocnitele, typ přístupu (allow nebo deny), oprávnění a požadovaný objekt.

```
#Next method in if returns deny
if(@my_acl.check("Penny", "SIT", "seat", "/livingroom/couch/Sheldon's_spot"))
  puts "Access allowed. You may retrive resource object."
else
  puts "Access denied."
```

3.2.3 Use Case Scénáře

3.2.3.1 Ověřování oprávnění k objektu

Uživatel má vytvořenou instanci RubyACL, která pracuje s pravidly. Hlavní úspěšný scénář:

1. Uživatel zavolá metodu acl_check. Přes tuto metodu se dotáže knihovny, jestli uživatel/skupina (ne)mají oprávnění ke zdrojovému objektu.

2. a) Systém vrátí true v případě, že uživatel/skupina má specifikované nebo vyšší oprávnění.

3. b) Systém vrátí false v případě, že uživatel/skupina nemají specifikované nebo vyšší oprávnění.

Rozšíření:

0a) Pokud neexistuje žádné pravidlo, systém vrátí false, protože nenašel, žádné vyhovující pravidlo.

3.2.3.2 Zadávání pravidla (ACE)

Uživatel má vytvořenou instanci RubyACL, která obsahuje pravidla. Hlavní úspěšný scénář:

1. Uživatel zavolá metodu set_new_ace a specifikuje údaje (Uživatel/skupina, typ přístupy (allow/deny), oprávnění, zdrojový objekt

2. Knihovna nastaví pravidlo a vrátí 0, když vše proběhlo v pořádku, a 1 když nastala chyba.

Poznámka: Jde vlastně o přiřazování oprávnění uživatelům na objekt

3.2.4 Data flow diagram

TODO smazat starý obrázek. Okomentovat nový Data flow diagram (viz "Obrázek 2") znázorňuje tok dat mezi jednotlivými funkcemi aplikace. Popisuje funkce a jejich vazby. Uložiště pro ACL je používaná databáze. Pro každou databázi bude jedna instance Ruby-ACL.

3.2.5 Class diagram

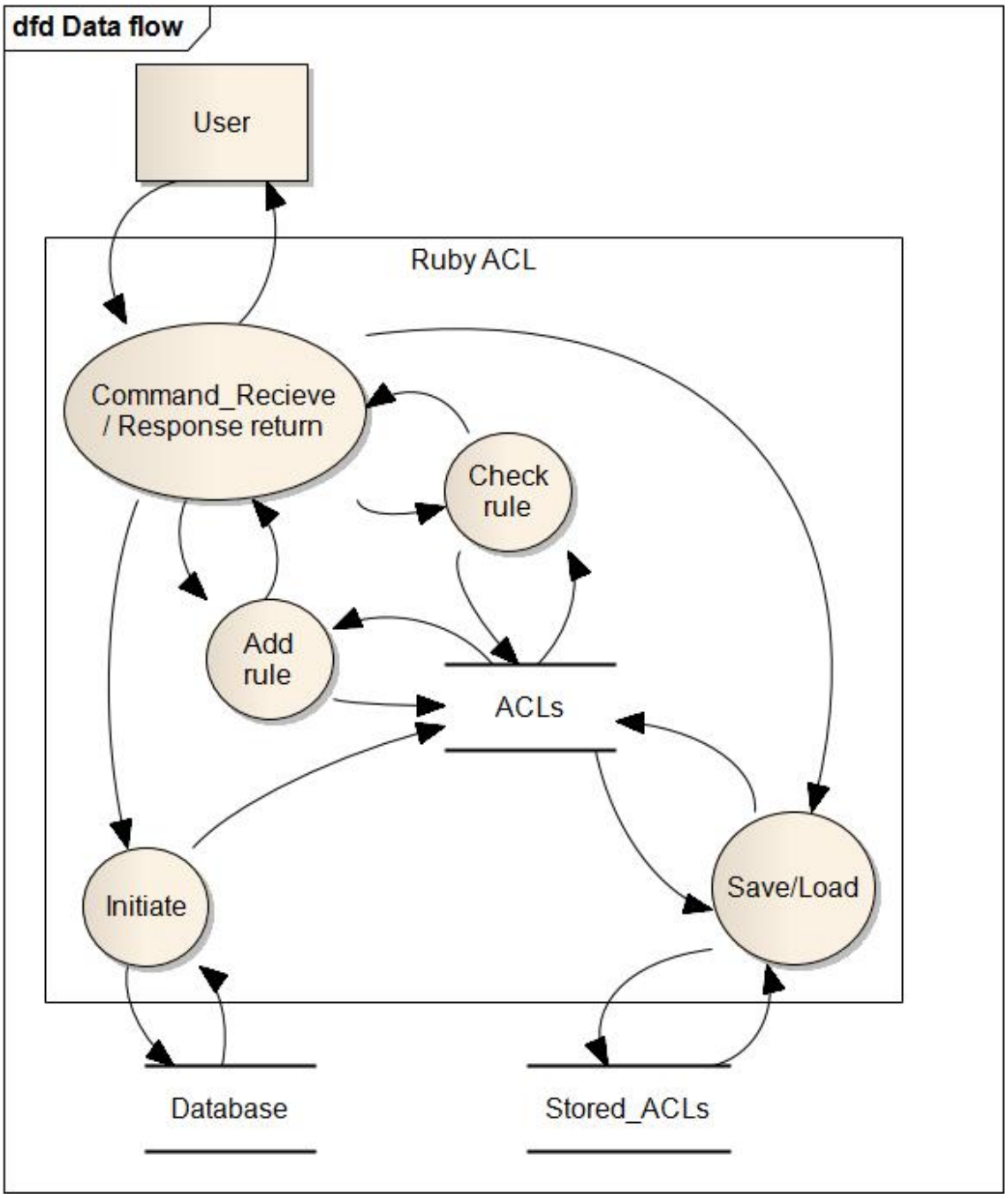
Class diagram (viz "Obrázek 3") znázorňuje základní třídy aplikace a jejich vazby.

3.3 Popis logiky vyhodnocování pravidel

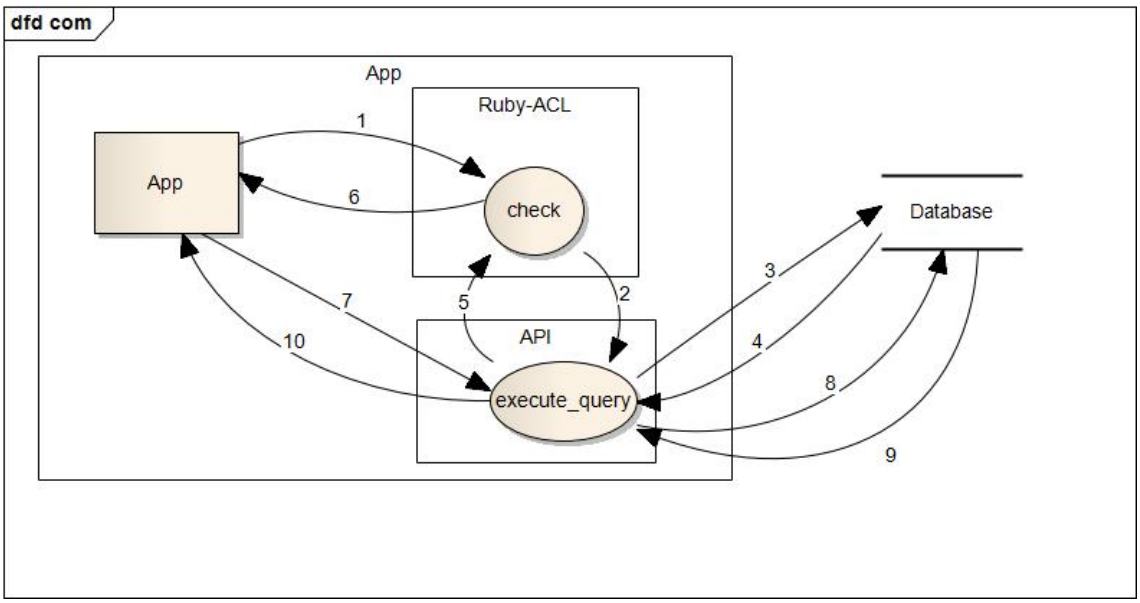
V této kapitole je popsáno jakým způsobem RubyACL rozhoduje o přidělení přístupu. Nejkonkrétněji se tímto zabývá sekce "Pravidla rozhodování", nicméně k pochopení celku je potřeba vědět vlastnosti jednotlivých objektů, které jsem nazval ACL objekty. O nich se dočtete ve stejnojmenné sekci. Vše se odvijí od priority rozhodování.

3.3.1 ACL Objekty

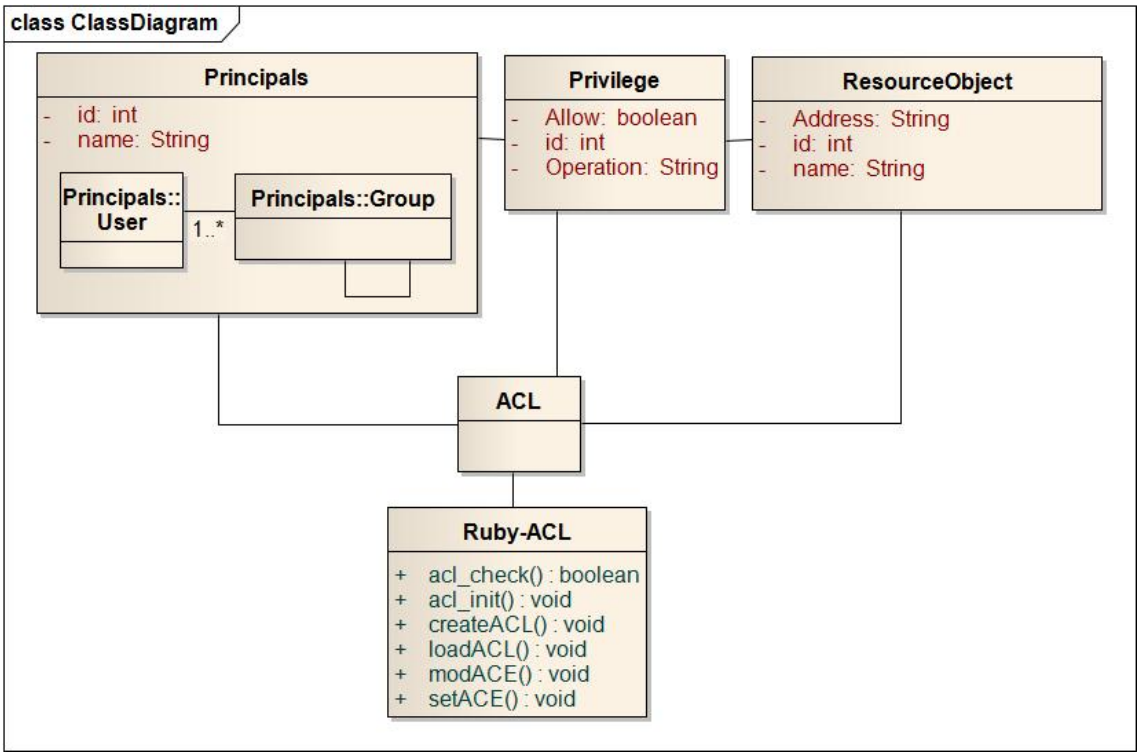
ACL objekt je principal, který se dělí na individual a group, privilege



Obrázek 3.1: Data flow diagram



Obrázek 3.2: Diagram znázorňující komunikaci



Obrázek 3.3: Class diagram

3.3.1.1 Zmocnitel (Principal)

Principal = [Individual, Group]

```
<Group id="Administrators">
    <membership>
        <mgroup idref="ALL"/>
    </membership>
</Group>
<Individual id="sirljan">
    <membership>
        <mgroup idref="Users" />
        <mgroup idref="Developers" />
    </membership>
</Individual>
```

3.3.1.2 Oprávnění (Privilege)

RubyACL obsahuje základní oprávnění Oracle a MySQL. Nevím ale, jestli to bude mít nějaký význam, když XML databáze má jinou formu dotazování. Pravidla lze jednoduše vytvořit a seskupovat. Ruby-ACL nabídne vlastní přidané privilegia a základní sadu privilegií a převzatých z Oracle potažmo MySQL. Jedná se o privilegia 'ALL PRIVILEGES', 'ALTER', 'CREATE', 'DELETE', 'DROP', 'FILE', 'INDEX', 'INSERT', 'PROCESS', 'REFERENCES', 'RELOAD', 'SELECT', 'SHUTDOWN', 'UPDATE' a 'USAGE'.

```
<Privilege id="SELECT">
    <membership>
        <mgroup idref="ALL_PRIVILEGES" />
    </membership>
</Privilege
```

3.3.1.3 Zdrojový objekt (Resource object)

Skládá se ze tří položek:

1. typ
2. adresa
3. vlastník

Při zadávání adresy je potřeba dodržet jediné pravidlo. V adrese oddělovat každý jednotlivý objekt dopředným lomítkem. (/)

Pokud je zdrojový objekt typu doc, adresa může obsahovat klauzuli ve formátu ("/koren/vetev/list-soubor_s_příponou") a pokud je zapotřebí jemnější řízení přístupu uvnitř dokumentu následuje klauzule /koren/vetev. Nicméně do databáze se ukládá sloučená adresa bez ("") Příklad: ("/db/data/cities.xml")/cities/city V databázi je uložen pod adresou: /db/data/-cities.xml/cities/city

Klauzule /* (lomítko hvězdička) vyjadřuje všechny podřadné objekty Příklad: /db/data/* Nyní jsou vybrány všechny podřadné objekty objektu data. /* nemá význam u vlastníka, protože vlastník zdrojového objektu vlastní i všechny podřazené zdrojové objekty.

Vlastník / Owner Owner může být jednotlivec, množina jednotlivců i skupina. Množina jednotlivců je v případě pokud se od kořene zdrojových objektů k listu mnění vlastník. Vlastník nejnadřazenějšího zdroje má největší práva. Vlastník může být o /* nemá význam, protože vlastník zdrojového objektu vlastní i všechny podřazené zdrojové objekty.

3.3.1.4 Pravidlo (ACE)

```
<Ace id="a894">
    <Principal idref="Users"/>
    <accessType>allow</accessType>
    <Privilege idref="SELECT"/>
    <ResourceObject idref="852"/>
</Ace>
```

3.3.2 Složitost rozhodování

Slozitost je e$\hat{2}$.

3.3.3 Pravidla rozhodování

3.3.3.1 Priorita rozhodování

Nejnižší má největší prioritu.

1. Owner
2. Explicit Deny
3. Explicit Allow
4. Inherited Deny
5. Inherited Allow
6. If not found > Deny

Kapitola 4

Realizace

Popis implementace/realizace se zaměřením na nestandardní části řešení.

4.1 Průběh realizace

Fáze 1 Po analýze a návrhu byla naimplementována zkušební část, kvůli ověření návrhu tříd a rozhraní. Na této verzi byla napsáno první velká část unit testů. Verze ukládala pouze do instancí nebo polí. Fáze 2 Zprovoznění komunikace s databází. Hledání alternativních možností(sedna apod). Napsání eXistAPI. Fáze 3 Předělání první verze implementace RubyACL z polové/instanční na databázovou. Přepsání tříd z instančních na jakési singlotonské managery. Všechny informace jsou uloženy v databázi. RubyACl pouze vkládá nové záznamy, upravuje a maže staré. Na existujících provádí dotazy a rozhodování o přístupu. Fáze 4 Návrh tříd byl trochu chybný. Implementace předchozích verzí se odchylovala návrhu. Předělání tříd. Vytvoření nadřazené třídy ACL_Object a dědění z této třídy. Intenzivní testování. Usnadnilo odhalit spousty chyb. Fáze 5 Dokončení rozhodovací metody. Dotahování funkcionality. Opravování chyb a nedostatků. Testování v této fázi nejintenzivnější. Dokumentace. Fáze 6 Finalizování - rakefile, rubygem. Dokumentace

4.2 Programy použité při vývoji

- NetBeans s Ruby platformou od Thomase Enebo http://blog.enebo.com/ http://plugins.netbeans.or and-rails
- GIT https://github.com/sirljan/Ruby-ACL
- eXist-db http://www.exist-db.org
- Enterprise Architect
- MikTex

4.3 Databáze

Protože XML databáze, která má RubyACL používat, není plně funkční, nebylo možné implementaci testovat přímo na ní. Za tímto učelem se musela vybrat jiná podobná databáze.

4.3.1 Sedna

Sedna knihovna v Ruby poskytuje rozhraní pro databázi Sedna. Klient je Ruby rozšíření, které používá ovladač jazyka C, který je dodáván jako součást distribuce Sedna. To má být jednoduché a snadno použitelné. Ovšem zprovoznění knihovny Sedna neproběhlo tak hladce, jak se tvrdí.

4.3.2 eXist-db

eXist-db je open source systém pro správu databáze vytvořena pomocí technologie XML. Ukládá XML data podle datového modelu XML a nabízí efektivní a XQuery zpracování založené na indexování. Podporuje velké množství technologií. Uvedu zde pouze ty, které se týkají RubyACL nebo eXistAPI: XPath 2.0(odkaz), XQuery(odkaz), XML-RPC(odkaz), XQuery Update Facility 1.0(odkaz). ExistAPI je komunikační rozhraní, které jsem byl nucený naimplementovat k lazení a testování RubyACL na eXist-db.

eXist-db se podobá XML databázi, která má RubyACL používat, a byla doporučena vedoucím práce jako ideální. Přesto proběhly komplikace s XUpdate navzdory přesné interpretace dokumentace. Z tohoto důvodu bylo nutné přejít na xQuery Update Facility

4.3.2.1 eXistAPI

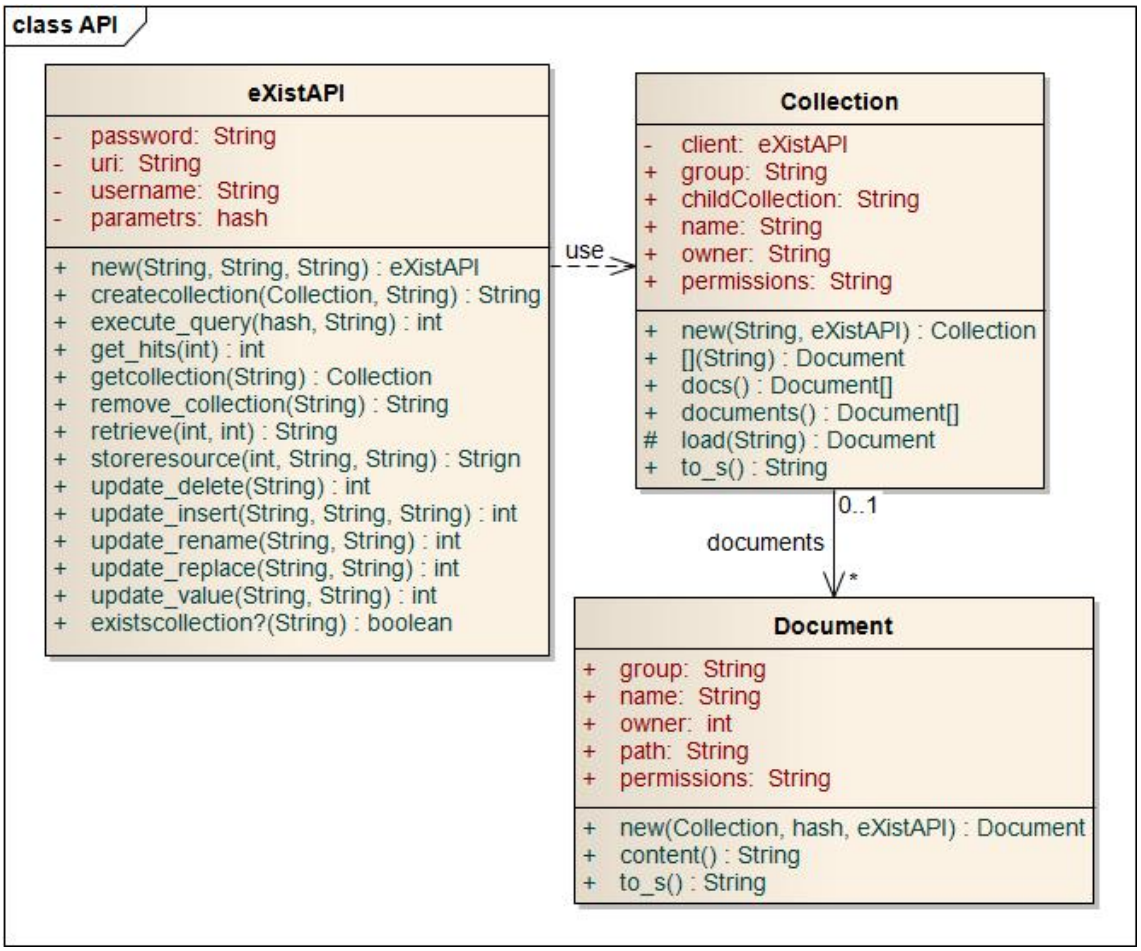
Trochu nestandartní část realizace. Bylo potřeba si vytvořit komunikační rozhraní mezi databází eXist-db a RubyACL. ExistAPI je komunikační rozhraní, které bylo potřeba naimplementovat k ladění a testování RubyACL na eXist-db. Komunikace s db pomocí eXistAPI. Použití xQuery Update Facility místo xUpdate. (Možná vložím ukázku implementace) Komunikace založená na protokolu XML-RPC. K dotazování využívá technologie xQuery a xPath. xQuery Update Facility je zapotřebí k editování dat v databázi. TODO předělat v obrázku api na eXistAPI

4.4 Implementace

4.4.1 pracovni poznamky

- Je třeba se věnovat uvolňování paměti, protože aplikace bude spuštěna dny až měsíce. Špatné zacházení s pamětí by, proto bylo kritické pro server, na kterém aplikace s Ruby-ACL knihovnou bude spuštěna.

- Ušetření času db serveru vhodným uspořádáním dotazů. Místo iterace dotazů jeden dotaz s vhodnou podmínkou. (tvoreni clenstvi)



Obrázek 4.1: Diagram tříd TODO

- Pro identifikaci a propojení jsem uvažoval mezi xLink a idref. Idref nabízí jednoduchý systém, ale xLink nabídl podrobné specifikace W3C, velké množství návodů a turtoriálů a především se jedná o novější technologii, jejíž osvojení jsem považoval za výhodné. Rozhodl jsem se proto implementovat xLink. Problém nastal při některých vkládání textů a dotazování. EXist DB měla problém s jmeným prostorem xLink i přes zkutečnost, že jmený prostor byl uveden. Troufám si říct, že byl uveden správně, protože při většině použití fungoval. Než abych se zabýval mnoho času proč použití xLink nejde, raději jsem přešel na jednoduchý idref. Implementace idref proběhla bez problémů.
- Občas je kód zdvojený, ale má to svoje opodstatnění. Například addmembership a addmembershippriv. Kdyby nebylo zdvojeno, šlo by přidávat privilige k principal a naopak.
- predelavam structuru z tridniho volani na singleton nebo jak se to jmenuje.
- predelavam structuru z tridniho volani na singleton nebo jak se to jmenuje.
- kdyz smazu nadrazenou skupinu tak smazu vsechny jeji clenstvi

4.5 Zdrojové soubory

zdrojové soubory jsou v xml formátu. Jejich strukturu popisují dtd soubory.

Kapitola 5

Testování

- Způsob, průběh a výsledky testování.
- Srovnání s existujícími řešeními, pokud jsou známy.

Testování bylo zaměřeno pouza na funkcionalitu. Vynecháno bylo testování rychlosti, práce s operační pamětí, spotřeby procesorového času.

5.1 eXistAPI

5.2 RubyACL

Kapitola 6

Závěr

- Zhodnocení splnění cílů DP/BP a vlastního přínosu práce (při formulaci je třeba vzít v potaz zadání práce).
- Diskuse dalšího možného pokračování práce.

Možné zlepšení: rychlost - Jednoznačně časově nejdražší operací je komunikace s databází. Zrychlení by se dalo docílit optimalizací dotazování tak, aby se dotazovalo, co nejméně. Daní by byla větší paměťová náročnost aplikace, která by knihovnu RubyACL používala. Současný kód knihovny obsahuje opakované dotazování na stejnou věc místo ukládání výsledku do paměti pro případné následující použití. Tento model byl zvolen kvůli jednoduchosti a faktu, že vykonání dotazu a vrácení výsledku trvá řadově milisekundy. Je ale těžké odhadnout, kolik času by zabralo dotazování při plné databázi a použití více uživatelů ve stejný čas. Navíc eXist si po určitou dobu v paměti uchovává výsledky předchozích dotazů, čímž se snižuje naročnost opakovaných dotazů. Vše je ale závislé na lince mezi knihovnou a databází.

Kapitola 7

Moje pracovni poznamky

- id u resob. Pokud to bude poradi, mohlo byt snaze napadnutelne nez kdyby id nedavalo smysl.
- eXist xpath chyba. Na dotaz

  ```
  doc("/db/test_acl/ResourceObjects.xml")/ResourceObjects/descendant::*[type="Rybnik"]
  ```

  ale na dotaz

  ```
  doc("/db/test_acl/ResourceObjects.xml")/ResourceObjects/descendant::*[type="Rybnik"]
  ```

- res_ob se muzou vytvorit dopredu v seznamu res_obs. Res_ob musi byt kazdopadne v seznamu (vytvori se pri vytvoreni pravidla).
- Use “all” or “*” to match any address.
- vysledek rozhodnuti jde nechranenou cestou. pokud ma napadajici pristup ke kanalu mezi db a rubyacl potazmo db aplikaci, tak muze zmenit vysledek dotazu a tim i rozhodnuti.
- to_s jsem vynechal, protoze knihovna bude pouzita aplikaci a ne clovekem, ktery by si mohl precist stringovy obsah.
-

Kapitola 8

Literatura

Literatura

[1] web:oracle. ORACLE — Oracle® XML DB Developer's Guide, 11g Release 1 (11.1), Part Number B28369-04.
`http://docs.oracle.com/cd/B28359_01/appdev.111/b28369/xdb21sec.htm`, stav z 15. 5. 2012.

[2] web:phpGACL. phpGACL — Generic Access Control List with PHP.
`phpgacl.sourceforge.net/manual.pdf`, stav z 15. 5. 2012.

Příloha A

Pokyny a návody k formátování textu práce

Tato příloha samozřejmě nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.

Používat se dají všechny příkazy systému LaTeX. Existuje velké množství volně přístupné dokumentace, tutoriálů, příruček a dalších materiálů v elektronické podobě. Výchozím bodem, kromě Googlu, může být stránka CSTUG (Czech Tech Users Group) [**?**]. Tam najdete odkazy na další materiály. Vetšinou dostačující a přehledně organizovanou elektronikou dokumentaci najdete například na [**?**] nebo [**?**].

Existují i různé nadstavby nad systémy TeX a LaTeX, které výrazně usnadní psaní textu zejména začátečníkům. Velmi rozšířený v Linuxovém prostředí je systém Kile.

A.1 Vkládání obrázků

Obrázky se umísťují do plovoucího prostředí `figure`. Každý obrázek by měl obsahovat **název** (`\caption`) a **návěští** (`\label`). Použití příkazu pro vložení obrázku `\includegraphics` je podmíněno aktivací (načtením) balíku graphicx příkazem `\usepackage{graphicx}`.

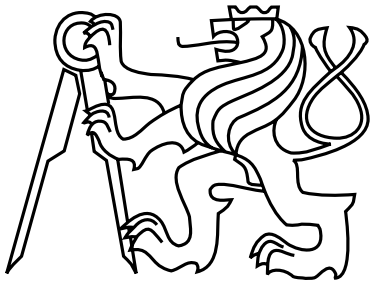
Budete-li zdrojový text zpracovávat pomocí programu `pdflatex`, očekávají se obrázky s příponou `*.pdf`[1], použijete-li k formátování `latex`, očekávají se obrázky s příponou `*.eps`.[2]

Příklad vložení obrázku:

```
\begin{figure}[h]
\begin{center}
\includegraphics[width=5cm]{figures/LogoCVUT}
\caption{Popiska obrazku}
\label{fig:logo}
```

[1] pdflatex umí také formáty PNG a JPG.

[2] Vzájemnou konverzi mezi snad všemi typy obrazku včetně změn vekostí a dalších vymožeností vám může zajistit balík ImageMagic (http://www.imagemagick.org/script/index.php). Je dostupný pod Linuxem, Mac OS i MS Windows. Důležité jsou zejména příkazy convert a identify.



Obrázek A.1: Popiska obrázku

DTD	construction	elimination
|	`in1|A|B a:sum A B` `in1|A|B b:sum A B`	`case([_:A]a)([_:B]a)ab:A` `case([_:A]b)([_:B]b)ba:B`
+	`do_reg:A -> reg A`	`undo_reg:reg A -> A`
$*, ?$	the same like | and + with `emtpy_el:empty`	the same like | and + with `emtpy_el:empty`
R(a,b)	`make_R:A->B->R`	`a: R -> A` `b: R -> B`

Tabulka A.1: Ukázka tabulky

```
\end{center}
\end{figure}
```

A.2 Kreslení obrázků

Zřejmě každý z vás má nějaký oblíbený nástroj pro tvorbu obrázků. Jde jen o to, abyste dokázali obrázek uložit v požadovaném formátu nebo jej do něj konvertovat (viz předchozí kapitola). Je zřejmě vhodné kreslit obrázky vektorově. Celkem oblíbený, na ovládání celkem jednoduchý a přitom dostatečně mocný je například program Inkscape.

Zde stojí za to upozornit na kreslící programe Ipe [**?**], který dokáže do obrázku vkládat komentáře přímo v latexovském formátu (vzroce, stejné fonty atd.). Podobné věci umí na Linuxové platformě nástroj Xfig.

Za pozornost ještě stojí schopnost editoru Ipe importovat obrázek (jpg nebo bitmap) a krelit do něj latexovské popisky a komentáře. Výsledek pak umí exportovat přímo do pdf.

A.3 Tabulky

Existuje více způsobů, jak sázet tabulky. Například je možno použít prostředí `table`, které je velmi podobné prostředí `figure`.

Zdrojový text tabulky 2.1 vypadá takto:

```
\begin{table}
\begin{center}
\begin{tabular}{|c|l|l|}
\hline
\textbf{DTD} & \textbf{construction} & \textbf{elimination} \\
\hline
$\mid$ & \verb+in1|A|B a:sum A B+ & \verb+case([_:A]a)([_:B]a)ab:A+\\
&\verb+in1|A|B b:sum A B+ & \verb+case([_:A]b)([_:B]b)ba:B+\\
\hline
$+$&\verb+do_reg:A -> reg A+&\verb+undo_reg:reg A -> A+\\
\hline
$*,?$& the same like $\mid$ and $+$ & the same like $\mid$ and $+$\\
& with \verb+emtpy_el:empty+ & with \verb+emtpy_el:empty+\\
\hline
R(a,b) & \verb+make_R:A->B->R+ & \verb+a: R -> A+\\
 & & \verb+b: R -> B+\\
\hline
\end{tabular}
\end{center}
\caption{Ukázka tabulky}
\label{tab:tab100}
\end{table}
\begin{table}
```

A.4 Odkazy v textu

A.4.1 Odkazy na literaturu

Jsou realizovány příkazem `\cite{odkaz}`.

Seznam literatury je dobré zapsat do samostatného souboru a ten pak zpracovat programem bibtex (viz soubor `reference.bib`). Zdrojový soubor pro `bibtex` vypadá například takto:

```
@Article{Chen01,
  author  = "Yong-Sheng Chen and Yi-Ping Hung and Chiou-Shann Fuh",
  title   = "Fast Block Matching Algorithm Based on
             the Winner-Update Strategy",
  journal = "IEEE Transactions On Image Processing",
  pages   = "1212--1222",
  volume  =  10,
  number  =   8,
  year    = 2001,
}

@Misc{latexdocweb,
```

```
  author  = "",
  title   = "{\LaTeX} --- online manuál",
  note    = "\verb|http://www.cstug.cz/latex/lm/frames.html|",
  year    = "",
}
...
```

Pozor: Sazba názvů odkazů je dána BibTEX stylem (`\bibliographystyle{abbrv}`). BibTEX tedy obvykle vysází velké pouze počáteční písmeno z názvu zdroje, ostatní písmena zůstanou malá bez ohledu na to, jak je napíšete. Přesněji řečeno, styl může zvolit pro každý typ publikace jiné konverze. Pro časopisecké články třeba výše uvedené, jiné pro monografie (u nich často bývá naopak velikost písmen zachována).

Pokud chcete BibTEXu napovědět, která písmena nechat bez konverzí (viz `title = "{\LaTeX} --- online manuál"` v předchozím příkladu), je nutné příslušné písmeno (zde celé makro) uzavřít do složených závorek. Pro přehlednost je proto vhodné celé parametry uzavírat do uvozovek (`author = "..."`), nikoliv do složených závorek.

Odkazy na literaturu ve zdrojovém textu se pak zapisují:

```
Podívejte se na \cite{Chen01},
další detaily najdete na \cite{latexdocweb}
```

Vazbu mezi soubory `*.tex` a `*.bib` zajistíte příkazem `\bibliography{}` v souboru `*.tex`. V našem případě tedy zdrojový dokument `thesis.tex` obsahuje příkaz `\bibliography{reference}`.

Zpracování zdrojového textu s odkazy se provede postupným voláním programů `pdflatex <soubor>` (případně `latex <soubor>`), `bibtex <soubor>` a opět `pdflatex <soubor>`.[3]

Níže uvedený příklad je převzat z dříve existujících pokynů studentům, kteří dělají svou diplomovou nebo bakalářskou práci v Grafické skupině.[4] Zde se praví:

```
...
j) Seznam literatury a dalších použitých pramenů, odkazy na WWW stránky, ...
 Pozor na to, že na veškeré uvedené prameny se musíte v textu práce
 odkazovat -- [1].
Pramen, na který neodkazujete, vypadá, že jste ho vlastně nepotřebovali
a je uveden jen do počtu. Příklad citace knihy [1], článku v časopise [2],
stati ve sborníku [3] a html odkazu [4]:
[1] J. Žára, B. Beneš;, and P. Felkel.
     Moderní počítačová grafika. Computer Press s.r.o, Brno, 1 edition, 1998.
     (in Czech).
```

[3] První volání `pdflatex` vytvoří soubor s koncovkou `*.aux`, který je vstupem pro program `bibtex`, pak je potřeba znovu zavolat program `pdflatex` (`latex`), který tentokrát zpracuje soubory s příponami `.aux` a `.tex`. Informaci o případných nevyřešených odkazech (cross-reference) vidíte přímo při zpracovávání zdrojového souboru příkazem `pdflatex`. Program `pdflatex` (`latex`) lze volat vícekrát, pokud stále vidíte nevyřešené závislosti.

[4] Několikrát jsem byl upozorněn, že web s těmito pokyny byl zrušen, proto jej zde přímo necituji. Nicméně příklad sám o sobě dokumentuje obecně přijímaný konsensus ohledně citací v bakalářských a diplomových pracích na KP.

```
[2] P. Slavík. Grammars and Rewriting Systems as Models for Graphical User
     Interfaces. Cognitive Systems, 4(4--3):381--399, 1997.
[3] M. Haindl, Š. Kment, and P. Slavík. Virtual Information Systems.
     In WSCG'2000 -- Short communication papers, pages 22--27, Pilsen, 2000.
     University of West Bohemia.
[4] Knihovna grafické skupiny katedry počítačů:
     http://www.cgg.cvut.cz/Bib/library/
```

... abychom výše citované odkazy skutečně našli v (automaticky generovaném) seznamu literatury tohoto textu, musíme je nyní alespoň jednou citovat: Kniha [**?**], článek v časopisu [**?**], příspěvek na konferenci [**?**], www odkaz [**?**].

Ještě přidáme další ukázku citací online zdrojů podle české normy. Odkaz na wiki o frameworcich [**?**] a ORM [**?**]. Použití viz soubor `reference.bib`. V seznamu literatury by nyní měly být živé odkazy na zdroje. V `reference.bib` je zcela nový typ publikace. Detaily dohledal a dodal Petr Dlouhý v dubnu 2010. Podrobnosti najdete ve zdrojovém souboru tohoto textu v komentáři u příkazu `\thebibliography`.

A.4.2 Odkazy na obrázky, tabulky a kapitoly

- Označení místa v textu, na které chcete později čtenáře práce odkázat, se provede příkazem `\label{navesti}`. Lze použít v prostředích `figure` a `table`, ale též za názvem kapitoly nebo podkapitoly.
- Na návěští se odkážeme příkazem `\ref{navesti}` nebo `\pageref{navesti}`.

A.5 Rovnice, centrovaná, číslovaná matematika

Jednoduchý matematický výraz zapsaný přímo do textu se vysází pomocí prostředí `math`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$`.

Kód `$ S = \pi * r^2 $` bude vysázen takto: $S = \pi * r^2$.

Pokud chcete nečíslované rovnice, ale umístěné centrovaně na samostatné řádky, pak lze použít prostředí `displaymath`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$$`. Zdrojový kód: `|$$ S = \pi * r^2 $$|` bude pak vysázen takto:

$$S = \pi * r^2$$

Chcete-li mít rovnice číslované, je třeba použít prostředí `eqation`. Kód:

```
\begin{equation}
  S = \pi * r^2
\end{equation}

\begin{equation}
  V = \pi * r^3
\end{equation}
```

je potom vysázen takto:

$$S = \pi * r^2 \tag{A.1}$$

$$V = \pi * r^3 \tag{A.2}$$

A.6 Kódy programu

Chceme-li vysázet například část zdrojového kódu programu (bez formátování), hodí se prostředí `verbatim`:

```
         (* nickname2 *)
Lego> Refine in1
             (do_reg (nickname1 h));
Refine by  in1 (do_reg (nickname1 h))
   ?4 : pcdata
   ?5 : pcdata
          (* surname2 *)
Lego> Refine surname1 h;
Refine by  surname1 h
   ?5 : pcdata
          (* email2 *)
Lego> Refine undo_reg (email1 h);
Refine by  undo_reg (email1 h)
*** QED ***
```

A.7 Další poznámky

A.7.1 České uvozovky

V souboru `k336_thesis_macros.tex` je příkaz `\uv{}` pro sázení českých uvozovek. „Text uzavřený do českých uvozovek.“

Příloha B

Seznam použitých zkratek

2D Two-Dimensional

ABN Abstract Boolean Networks

ASIC Application-Specific Integrated Circuit

⋮

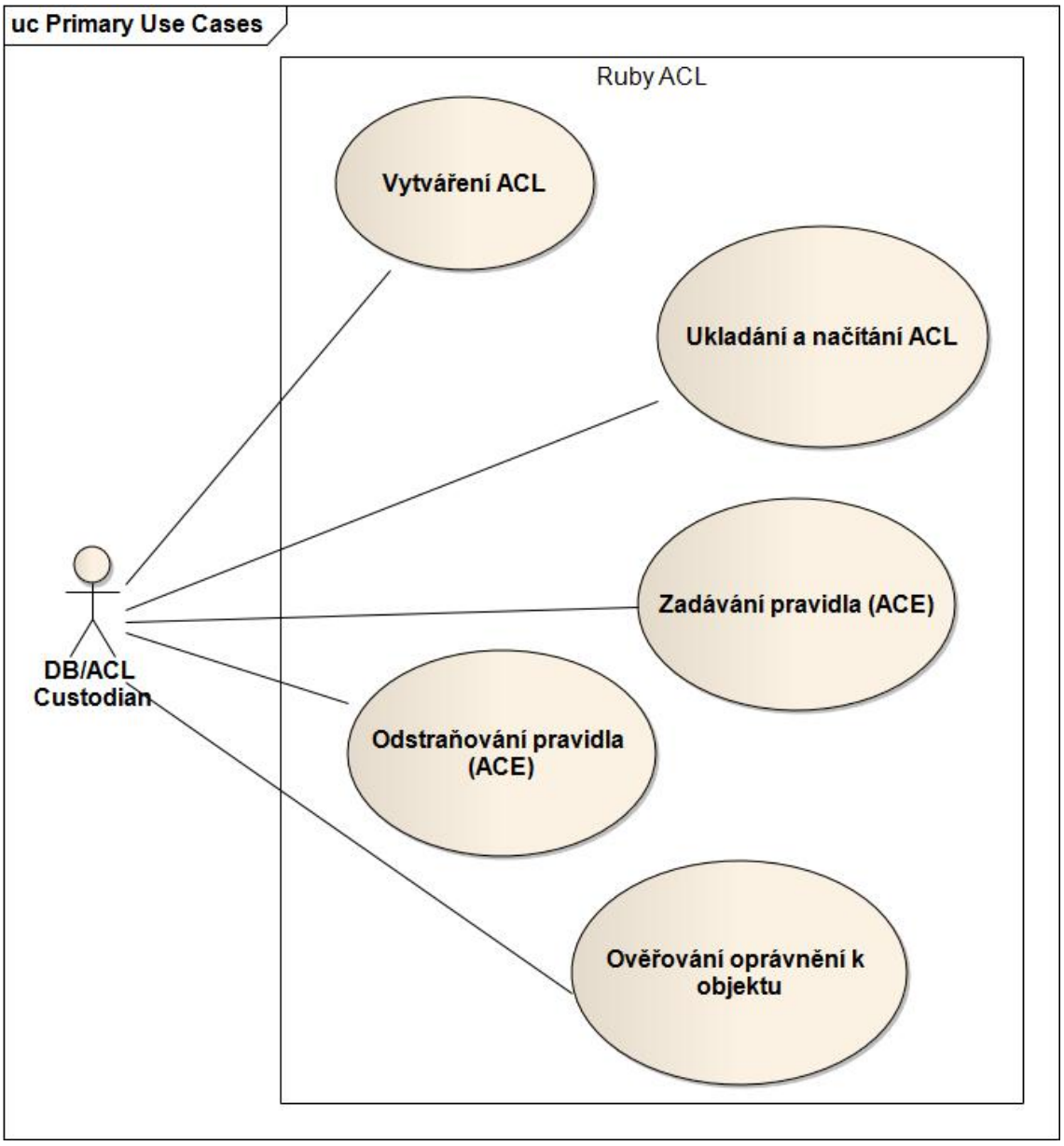
Příloha C

UML diagramy

Tato příloha není povinná a zřejmě se neobjeví v každé práci. Máte-li ale větší množství podobných diagramů popisujících systém, není nutné všechny umísťovat do hlavního textu, zvláště pokud by to snižovalo jeho čitelnost.

C.0.2 Use Case Diagram

TODO přesunout do přílohy. Znázorňuje roli uživatele vůči knihovně. Ruby ACL definuje jednoho aktéra, kterým je uživatel/administrátor ACL viz obrázek C.1. Jedná se vlastně o vývojáře DB Applikace. Všechny případy užití předpokládají vytvořenou instanci RubyACL, která má vytvořená nějaká pravidla.



Obrázek C.1: UseCases

Příloha D

Instalační a uživatelská příručka

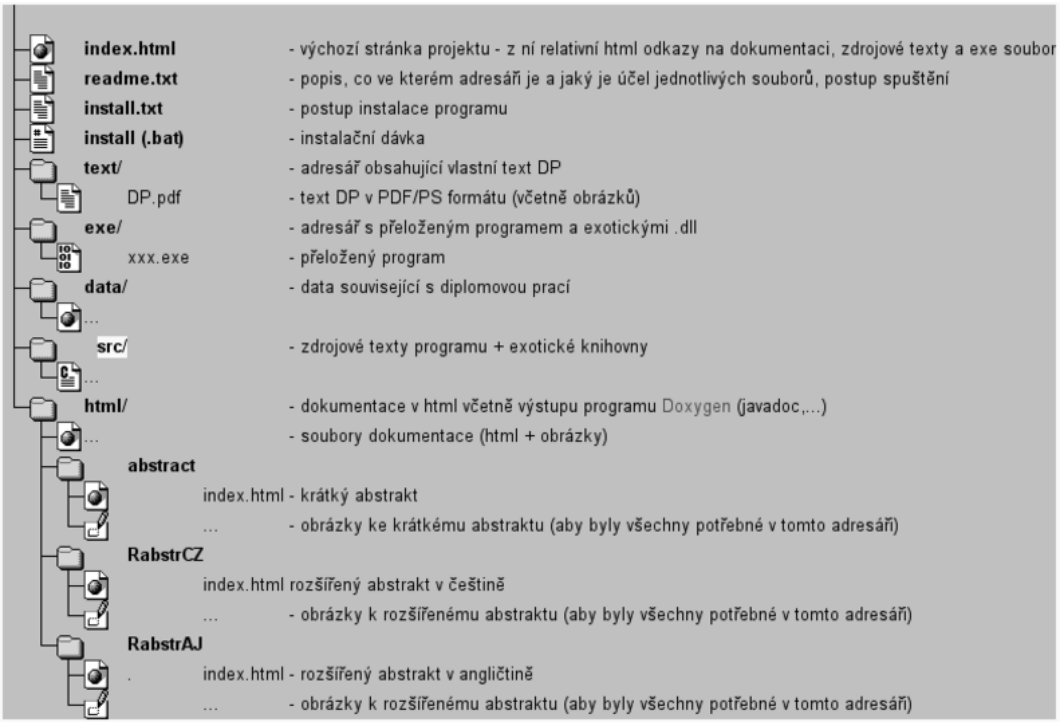
Tato příloha velmi žádoucí zejména u softwarových implementačních prací.

Příloha E

Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [**?**]):



Obrázek E.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případne index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.