Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Systém správy identit pro malé a střední firmy

Diplomová práce

Bc. Karel Maxa

Vedoucí práce: Ing. Martin Čížek, MBA

České Budějovice 2014

Bibliografické údaje

Maxa Karel, 2014: Systém správy identit pro malé a střední firmy. [Identity Management Solution for Small and Medium Businesses. Mgr. Thesis, in Czech.] – 106 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tématem práce je vytvoření systému pro správu identit u malých a středních firem. Práce obsahuje čtyři hlavní části. První část obsahuje nezbytný teoretický základ jako je popsání RBAC modelu nebo namodelování vztahu mezi prakticky používanými objekty (uživatelská identita, role, pozice, oprávnění, účet...). V rámci druhé části byla provedena analýza fungování a potřeb cílených organizací. Třetí část popisuje návrhovou část vyvíjené aplikace. Čtvrtá část se zabývá samotnou implementací aplikace.

Hlavním výstupem práce je hotová aplikace schopná nasazení u prací definovaných organizací. Aplikace obsahuje veškeré funkčnosti požadované v rámci první fáze projektu.

Anotation

The topic of this master's thesis is development of identity management solution for small and medium business. The thesis is divided into four major parts. The first part contains theoretical background as description of RBAC model or model with relationships between practically used objects (user identity, role, position, permission, account...). Analysis of functioning and needs of targeted organizations was carried out in the second part. The third part describes the design of the developed application. The fourth part discusses actual implementation of the application.

The main outcome of the thesis is implemented application that can be deployed at thesis defined organizations. The application includes all the functionality required in the first phase of the project.

Prohlašuji, že svoji diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V ..... dne ..... Podpis autora

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu diplomové práce Ing. Martinu Čížkovi, MBA za cenné rady, připomínky a dále za umožnění zpracování tohoto tématu. Dále bych rád poděkoval všem kolegům ze společnosti Orchitech Solutions a učitelům Ústavu aplikované informatiky, kteří se jakýmkoliv způsobem postarali o rozvoj mých znalostí a dovedností. Rovněž děkuji své rodině za neutuchající podporu v celém průběhu studia.

Obsah

Kapitola 1

Úvod

V dnešní době se neustále zvyšuje penetrace IT systémů do organizací, kde dané systémy mohou hrát klíčovou roli ve fungování organizace. Vzhledem k rostoucímu důrazu na zajištění autenticity a důvěrnosti dat spravovaných pomocí aplikací je nezbytné, aby byla prováděna autentizace a autorizace uživatelských identit. V praxi to znamená, že pokud chtějí uživatelé přistupovat k aplikacím, musí mít zřízené účty s patřičnými oprávněními. Se zvyšujícím se počtem používaných aplikací se zvyšuje počet účtů, které musí být uživatelům zřízeny, což vyplývá z heterogenity používaných aplikací. V důsledku toho je správa uživatelských účtů a jejich oprávnění a řízení přístupu uživatelů k aplikacím stále složitějším a méně přehledným procesem. Aby organizace měla jasný přehled o tom, kdo a kam má zřízený přístup, je potřeba provádět správu z jednoho místa. Proto se do organizací zavádí systém pro správu identit (dále jen IDMS), který řeší procesy identity managementu (dále jen IDM).

IDM je sada podnikových procesů a podpůrné infrastruktury pro vytváření, údržbu a používání digitálních identit v souladu s politikou organizace[1]. IDMS je systém, který řeší kompletní životní cyklus uživatelských účtů napříč všemi připojenými cílovými systémy. Zdrojem uživatelských identit může být např. HR systém[1]. Mezi hlavní přednosti takového systému patří přehledné zobrazení účtů a oprávnění, kterými uživatel aktuálně disponuje a zjednodušení jejich správy, kdy je mnoho operací prováděno automaticky. Mezi další přednosti takového systému patří podpora uživatelského self-service nebo podpora schvalovacích workflow. Podporou uživatelského self-service se rozumí funkčnosti, které uživateli umožňují provádět elementární operace, jako je změna hesla k účtu či zažádat o přiřazení oprávnění. Schvalovacím workflow se rozumí proces, kdy např. přiřazení oprávnění uživateli musí být schváleno definovanými schvalovateli, jako je nadřízený žadatele či garant požadovaného oprávnění. Vše uvedené rezultuje ve snížení nákladů na lidskou obsluhu a vyšší konzistenci dat mezi všemi zapojenými systémy.

Vzhledem k tomu, že v portfoliu organizace (dále zadavatel), kde je autor práce zaměstnán, chyběl produkt, který by řešil správu identit u malých a středních firem, bylo rozhodnuto, že by bylo vhodné takový produkt vytvořit. Tato diplomová práce se zabývá nezbytným teoretickým základem, návrhem a implementací takového systému. Systém je zkráceně označován jako MIDM.

[1]Systém, který obsahuje funkce pro řízení lidských zdrojů v organizaci (např. evidence zaměstnanců).

1.1 Cíle práce

Prvním cílem diplomové práce je provést teoretický popis modelu RBAC a non-RBAC modelů. Dále namodelovat vztah mezi prakticky používanými objekty u modelu RBAC (uživatelská identita, role, pozice, sada, oprávnění, účet, skupina apod.).

Hlavním cílem je analyzovat, navrhnout a vytvořit řešení správy identit, které bude jednoduše uplatnitelné u malých a středních firem. Řešení bude umožňovat integraci se zdrojovým systémem a obousměrnou integraci s cílovými systémy, kde v první fázi se očekává integrace s *Active Directory* a *Google Apps*. Řešení bude dále obsahovat uživatelské rozhraní pro administrátory a manažery, uživatelský self-service a mělo by být připraveno na budoucí implementaci schvalovacích workflow. Při implementaci řešení mohou být využity vhodné komunitní projekty, nicméně výsledné řešení musí být na těchto projektech nezávislé.

1.2 Definice cílených organizací

Jak již bylo zmíněno v předchozích kapitolách práce, vytvořené řešení správy identit je cílené pro organizace spadající do kategorie SMB[2]. Malá firma je definována jako firma, která zaměstnává maximálně 50 zaměstnanců a zároveň má roční obrat menší než 10 milionů euro, zatímco střední firma je definována jako firma, která zaměstnává maximálně 250 zaměstnanců a zároveň má roční obrat menší než 50 milionů euro[2]. U zmíněných typů organizací se předpokládá, že díky jejich velikosti dává nasazení IDMS smysl kromě jiného také kvůli značné fluktuaci zaměstnanců a objemu uživatelských požadavků. Zároveň se předpokládá, že jsou využívány aplikace s autentizací/autorizací uživatelů a je využíván minimálně jeden z podporovaných cílových systémů.

1.3 Metodika vývoje softwaru

Vzhledem k tomu, že praktická část práce se zabývá kompletním vývojem IDMS od úvodní studie až po nasazení je nezbytné zvolit vhodnou metodiku vývoje. V rámci kompletního procesu vývoje bude nezbytné provést tyto kroky:

- Provést řízené rozhovory s osobami z modelových organizací,
- Provést řízené rozhovory se zadavatelem,
- Definovat funkční a nefunkční požadavky na IDMS,
- Analyzovat dostupné komponenty a určit jejich vhodnost pro použití,
- Vést iterativní vývoj pomocí agilních metodik vývoje software.

Z výše uvedených bodů lze pozorovat, že kompletní proces vývoje lze rozdělit na dvě hlavní části. První část odpovídá metodice vodopád[3], zatímco druhá část odpovídá ně-

[2]Zkratka pro *Small and Medium Business*, jedná se o běžně používané formální označení pro malé a střední firmy.

[3]ROYCE, Winston. Managing the Development of Large Software Systems. [online]. 1970 [cit. 2014-06-27]. Dostupné z: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

které z agilních metodik vývoje. Vzhledem k tomu, že zadavatel má pozitivní zkušenosti s metodikou Scrum[4], byla zvolena právě tato agilní metodika vývoje software.

[4]Scrum Alliance: Transforming the World of Work. [online]. [cit. 2014-06-27]. Dostupné z: http://www.scrumalliance.org/)

Kapitola 2

Teorie

Cílem této kapitoly je popsat nezbytný teoretický základ související s IDM. Popsaná teorie bude následně prakticky využita v návrhové části práce.

2.1 Pojmy IDM

V této podkapitole je krátce popsáno několik pojmů používaných v kontextu IDM. Zcela nejobecnějším pojmem je *entita*. Typickými entitami z reálného světa jsou lidé nebo organizace. Pojem entity je důležitý, neboť se objevuje v samotné definici *identity*. Identita je reprezentace entity ve specifickém prostředí[3]. Jiná definice říká, že identita je informace o entitě, která je dostačující k identifikaci entity v konkrétním kontextu[4]. Například shromážděné osobní údaje o zákazníkovi banky tvoří identitu v prostředí banky. Každá entita může mít více identit. Podle [4] se informace každé identity skládají ze tří skupin:

- *Identifikátory* (identifiers) - jednoznačný identifikátor jako je např. ID nebo email,
- *Údaje pověření* (credentials) - údaje umožňující autentizaci identity (např. heslo nebo digitální certifikát),
- *Atributy* (attributes) - data charakterizující entitu (např. jméno, příjmení nebo práva).

Na obrázku 2.1 je možné spatřit namodelované vztahy mezi entitami, identitami a informacemi identit. V kontextu IDMS tedy uživatelské identity představují konkrétní osoby, kterým jsou přiřazována práva a uživatelské účty.

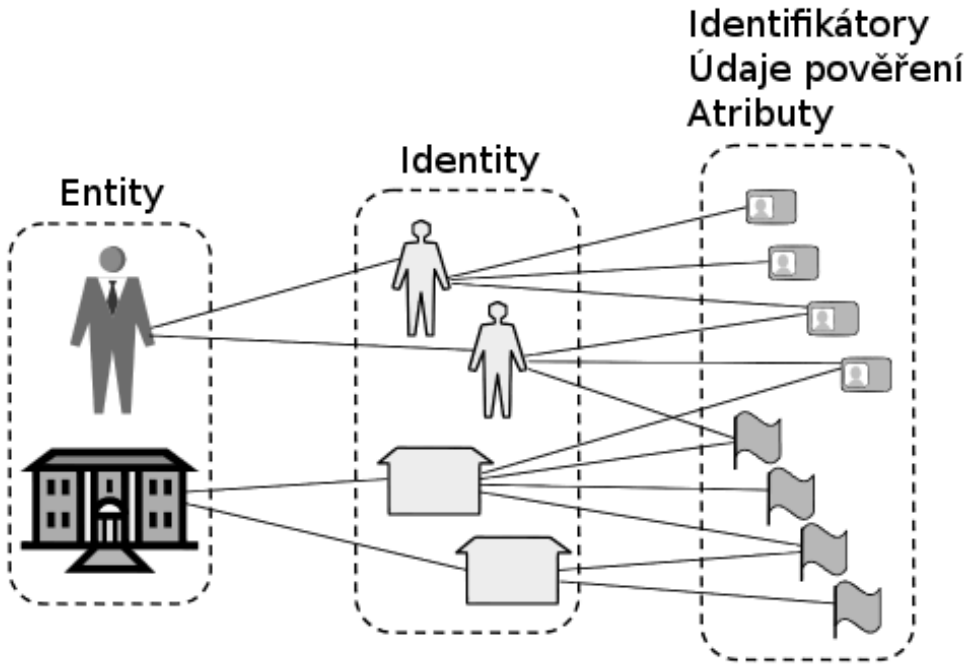
Dále je v rámci IDM zmiňován takzvaný *provisioning*. V rámci uživatelského provisioningu dochází k automatickému vytváření a údržbě uživatelských účtů a jejich atributů napříč všemi systémy. Pokud tedy například přijde do organizace nový zaměstnanec, díky provisioningu je zpropagována jeho identita do nezbytných systémů, kde jsou vytvořeny nezbytné uživatelské účty. Zaměstnanec poté může vykonávat jeho funkci. Procesem opačným je *deprovisioning*, v rámci kterého dochází k odebrání nebo zneplatnění uživatelských účtů ve všech nezbytných systémech.

Dalším pojmem je takzvaná *rekonciliace*. Během procesu rekonciliace dochází ke srovnání hodnot atributů objektů mezi IDMS a cílovými systémy. V rámci tohoto procesu může probíhat i přidělování vlastníků uživatelským účtům z různých cílových systémů. V rámci tohoto procesu jsou používána tzv. *korelační pravidla*, která definují

způsob určení vlastníka. Příkladem může být přidělování vlastníků na základě emailu či přihlašovacího jména.

Uživatelé IDMS mohou provádět úkony v kontextu tzv. uživatelského *self-service*. Jedním z nejčastěji prováděných úkonů je změna hesla uživatelského účtu ať z důvodu toho, že jej uživatel zapomněl nebo v případě, že to vyžaduje politika organizace.

Zdrojovým systémem se rozumí systém, z kterého jsou přebírány uživatelské identity. Cílovým systémem se rozumí systém, který je v organizaci používán pro autentizaci/autorizaci uživatelů a vůči němu IDMS synchronizuje spravované objekty.



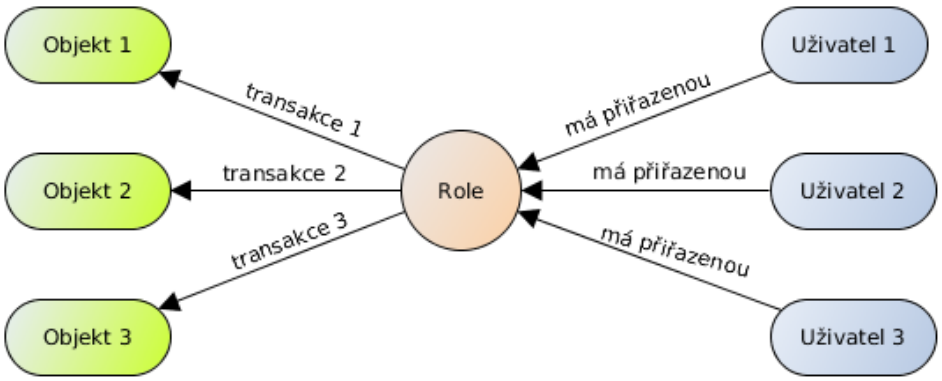
Obrázek 2.1: Vztahy mezi entitami, identitami a informacemi identit. Předloha [3].

2.2 RBAC model

Myšlenka řešení přístupu na základě uživatelských rolí není nikterak inovativní. Již před několika desítkami let využívaly organizace ve svých počítačových systémech omezení přístupu na základě role uživatele v dané organizaci. Zmíněné omezení přístupu na základě rolí bylo v systémech implementováno specifickým způsobem pro danou organizaci. Bylo to způsobeno tím, že v té době neexistovala žádná obecná specifikace, jak řídit přístup na základě uživatelských rolí. Model, který by byl vhodný pro všeobecné použití řízení přístupu na základě rolí definovali v roce 1992 David Ferraiolo a Richard Kuhn, kteří z tehdejších specifických přístupů vytvořili jeden obecný model. Tento model se nazývá RBAC neboli *role base access control*.

2.2.1 Popis

Role base access control lze chápat jako nezávislou komponentu kontroly přístupu, která může být použita spolu s MAC[1] nebo DAC[2], pokud je to vhodné[5]. Hlavním principem u RBAC je to, že oprávnění jsou přiřazována do rolí a uživatelům jsou přiřazovány vhodné role. Tím se výrazně zjednodušuje správa oprávnění, jelikož uživatelům nejsou přiřazována individuálně ale uživatel zdědí veškerá oprávnění, která jsou přiřazena do dané role. Role jsou v organizaci vytvářeny pro různé pracovní funkce, kde uživatelům jsou následně tyto role přiřazeny na základě jejich pozice a povinností. Členství uživatelů v rolích může být velmi snadno měněno s tím, jak jsou měněny povinnosti daného uživatele. Do rolí mohou být přiřazována nová oprávnění, která budou zároveň přiřazena všem uživatelům, kteří mají danou roli přiřazenou a stejně tak mohou být i odebírána. Z uvedeného vyplývá, že role by mohla být definována jako sémantický konstrukt pro spojení uživatelů a oprávnění. Jiná definice říká, že role může být chápána jako soubor transakcí, které může uživatel nebo skupina uživatelů provádět v rámci organizace[6]. Transakcí se rozumí jedna operace, která může být prováděna nad nějakým objektem. Příkladem transakce je např. schopnost pro lékaře zapsat diagnózu do zdravotní dokumentace pacienta nebo předepsat léky. Na obrázku 2.2 lze spatřit schematické znázornění vztahu mezi jednotlivými uživateli, rolemi, transakcemi a systémovými objekty. Z obrázku je jasně patrné, že znázornění uživatelé mohou provádět operace s objekty díky přiřazeným rolím.



Obrázek 2.2: Vztahy mezi objekty u role. Předloha [6].

Ve formálních definicích RBAC modelu se zpravidla používají následující konvence pro názvosloví (tyto konvence budou využívány v následujících podkapitolách práce):

- **A** (Access - přístup) - interakce mezi subjektem a objektem rezultující ve vzájemný tok informací,
- **AC** (Access control - řízení přístupu) - proces řízení přístupu k O, kdy přístup je povolen pouze pro autorizované S,
- **O** (Object - objekt) - pasivní entita obsahující data,
- **P** (Permission - oprávnění) - popis typu interakce, kterou S může mít s O,

[1]Zkratka pro *Mandatory access control* (viz 2.3.1).
[2]Zkratka pro *Discretionary Access Control* (viz 2.3.2).

- **PA** (Permission assignment - přiřazení oprávnění do rolí) - relace mezi množinou R a množinou P.
- **R** (Role) - pracovní pozice v organizaci definující odpovědnost a pravomoc uživatele přiřazeného do této role,
- **RH** (Role hierarchy - hierarchie rolí) - uspořádaná hierarchie mezi rolemi,
- **S** (Subject - subjekt) - aktivní entita jako je člověk, proces nebo zařízení,
- **SS** (Session - relace) - relace mezi U a množinou R, které má U přiřazené,
- **T** (Transaction - transakce) - operace, která je prováděna nad O,
- **U** (User - uživatel) - jakákoliv osoba, která přímo manipuluje se systémem,
- **UA** (User assignment - přiřazení rolí uživatelům) - relace mezi množinou U a množinou R.

V obecné rovině platí následující tvrzení:

- **S** může mít přiřazených více **R**,
- **R** může být přiřazená ve více **S**,
- **R** může mít přiřazených více **P**,
- **P** může být přiřazené ve více **R**,
- **P** může mít přiřazených více **T**,
- **T** může být přiřazená ve více **P**.

Pro formální popsání vztahů mezi objekty a řízení přístupu u RBAC je nejprve nezbytné uvést následující definice, tak jak jsou definovány v [6]:

- Pro každý **S** je aktivní **R** ta, kterou právě používá: *AR(S) = {aktivní R pro subjekt S}*,
- Každý **S** může mít autorizovaných více **R** k přiřazení: *RA(S) = {autorizované R pro subjekt S}*,
- Každá **R** může autorizovat k provedení více **T**: *TA(R) = {autorizované T pro roli R}*,
- **S** může provádět **T** pokud platí, že výraz *exec(S, T)* je pravdivý.

Na základě zmíněných definicí jsou poté definována tři základní pravidla:

1. Přiřazení role: **S** může provést **T** pouze pokud má přiřazenou roli. Matematicky vyjádřeno jako $\forall(S)\ \forall(T) : exec(S, T) \Rightarrow AR(S) \neq \emptyset$. Veškerá činnost uživatele v systému se provádí přes transakce, což znamená, že všichni aktivní uživatelé musejí mít přiřazenou nějakou aktivní roli.
2. Autorizace role: **S** může mít pouze takovou aktivní roli, která je pro tento **S** autorizována. Matematicky vyjádřeno jako $\forall(S) : AR(S) \subseteq RA(S)$.

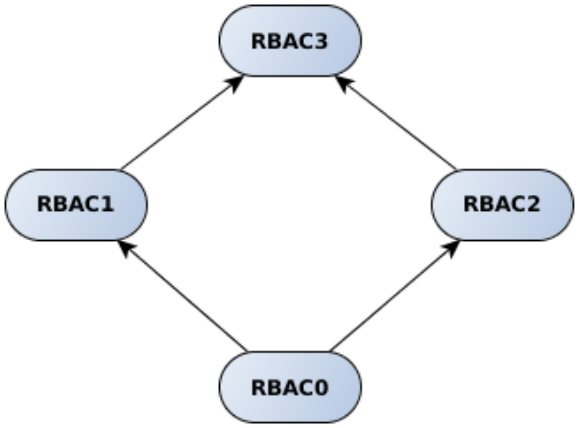
3. Autorizace transakce: **S** může provést **T** pouze za předpokladu, že provedení **T** povoluje aktivní role přiřazená **S**. Matematicky vyjádřeno jako $\forall(S)\forall(T) : exec(S,T) \Rightarrow T \in TA(AR(S))$.

Smyslem výše definovaných pravidel je formálně popsat situaci, kdy uživatel může provádět transakce (**T**). Na základě třetího pravidla je jasně patrné, že uživatelé mohou provádět pouze takové transakce, které jsou pro ně povolené na základě přiřazené role.

Transakce (**T**) byla již dříve definována jako operace nad nějaký objektem (**O**). Definovaná pravidla nezahrnují kontrolu, zdali uživatel vlastní oprávnění pro manipulaci s daným objektem (**O**). Pokud bychom chtěli do pravidel zahrnout kontrolu, zdali uživatel má povolen přístup k danému objektu (**O**), je nutné definovat další pravidlo, které bude matematicky vyjádřeno jako $\forall(S)\forall(T)\forall(O) : exec(S, T) \Rightarrow access(AR(S), T, O, x)$. Uvedené pravidlo definuje funkci k řízení přístupu (R, T, O, x), která určuje, zdali je přípustné, aby subjekt (**S**) s rolí (**R**) přistupoval k objektu (**O**) v režimu x pomocí transakce (**T**), kde x specifikuje požadovanou operaci (čtení, zápis).

2.2.2 Druhy RBAC modelů

Celý RBAC model je ve skutečnosti rozdělen do čtyř úrovní ($RBAC_0$, $RBAC_1$, $RBAC_2$ a $RBAC_3$). Základní myšlenkou je, že každá úroveň zavádí nějakou novou funkcionalitu. Vztahy modelů jsou definovány hierarchickou strukturou. Díky této struktuře mezi zmíněnými modely platí, že nadřazený model vždy obsahuje funkcionality podřazeného modelu. Hierarchickou strukturu modelů je možné spatřit na obrázku 2.3.

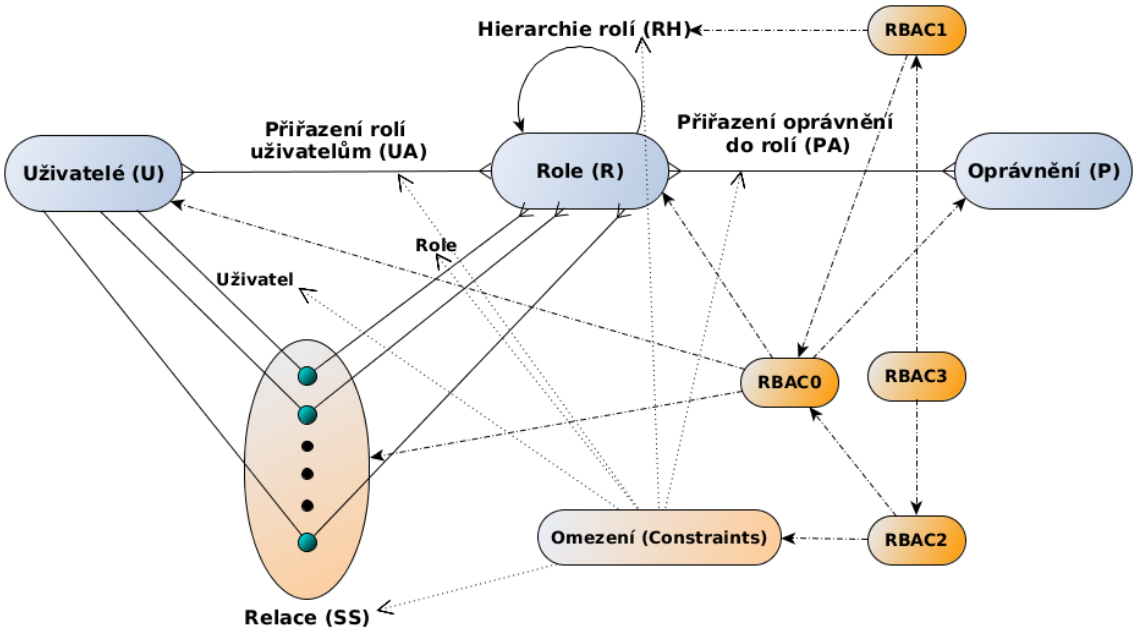


Obrázek 2.3: Struktura RBAC modelů. Předloha [5].

Z předchozího obrázku je jasně patrné, že základním modelem je model $RBAC_0$. Pokročilé modely $RBAC_1$ a $RBAC_2$ obsahují $RBAC_0$ a jsou vzájemně nezávislé, což znamená, že je lze využít samostatně. Nejvýše je umístěn model $RBAC_3$, který obsahuje $RBAC_1$, $RBAC_2$ a díky tranzitivitě také $RBAC_0$. Na obrázku 2.4 je možné spatřit znázornění obsahu jednotlivých úrovní RBAC modelu.

2.2.2.1 $RBAC_0$

Jak již bylo řečeno, $RBAC_0$ je základní RBAC model. Tento model zavádí tři různé množiny entit (uživatelé, oprávnění a role). Kromě zmíněných základních množin obsahuje základní model množinu relací. Obrázek 2.4 znázorňuje vztahy mezi množinou



Obrázek 2.4: Popis úrovní RBAC modelu. Předloha [5].

uživatelů a množinou rolí (UA) a množinou rolí a množinou oprávnění (PA). Oba tyto vztahy jsou typu *M:N*. To znamená, že uživatel může být členem ve více rolích a role může obsahovat více uživatelů. Podobné je to u druhého vztahu, kde role může obsahovat více oprávnění a oprávnění může být přiřazené ve více rolích.

Každá relace slouží jako spojovací prvek mezi jedním uživatelem a množinou rolí. Relace vzniká právě tehdy, když uživatel aktivuje určitou podmnožinu rolí, ve kterých je členem. Aktivací se rozumí operace, v rámci které je daná role pro daného uživatele převedena ze stavu neaktivního do aktivního. Poté množina oprávnění, která bude mít uživatel přiřazená, je dána sjednocením všech oprávnění z celé množiny aktivních rolí dané relace. Tato asociace mezi uživatelem a množinou rolí zůstává konstantní po celou dobu životního cyklu relace. Každý uživatel může mít otevřených více relací v jeden okamžik, kde každá relace může obsahovat rozdílnou množinu rolí. Tato funkce RBAC_0 podporuje princip minimálních oprávnění (viz 2.2.3). Uživatel, který je členem určité množiny rolí může mít aktivní vždy pouze určitou podmnožinu rolí, které jsou nezbytné pro plnění úkolů v dané organizaci.

Komponenty popsané v RBAC_0 mohou být formálně shrnuty do čtyř matematických výrazů:

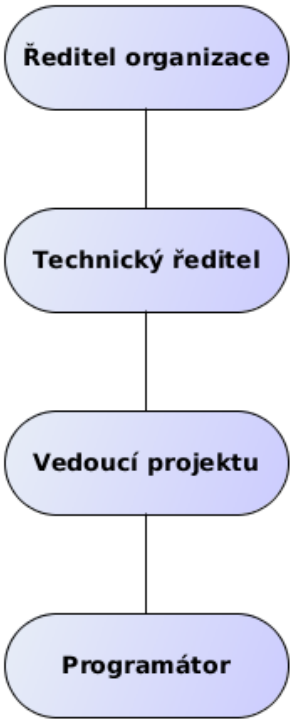
- $PA \subseteq P \times R$ - M:N relace mezi rolemi a oprávněními,
- $UA \subseteq U \times R$ - M:N relace mezi rolemi a uživateli,
- $User : SS \rightarrow U$ - funkce mapující každou relaci SS_i na uživatele $user(SS_i)$,
- $Roles : SS \rightarrow \{R\}$ - funkce mapující každou relaci SS_i na množinu rolí $roles(SS_i) \subseteq \{R \mid (user(SS_i), R) \in UA\}$, kde relace má oprávnění $R \in roles(SS_i)$ $\{P \mid (P, R) \in PA\}$.

2.2.2.2 $RBAC_1$

Pokročilý model $RBAC_1$ obsahuje veškerou funkcionalitu modelu $RBAC_0$ a navíc zavádí hierarchii rolí (RH). Hierarchie rolí je přirozeným prostředkem pro strukturování rolí tak, aby byly v souladu s pozicemi v organizaci. Dle konvencí se nadřazená role nazývá *senior* role a podřazená role se nazývá *junior* role. Na obrázku 2.5 lze spatřit příklad hierarchie rolí, kde jsou znázorněny čtyři různé role. Příkladem *senior* role je role *technický ředitel* a související *junior* role je role *vedoucí projektu*. Mezi souvisejícími rolemi platí dědičnost oprávnění. Díky tomu má každá role přiřazenu množinu oprávnění, která se skládá z oprávnění přímo přiřazených a oprávnění zděděných z *junior* role. Dále je potřeba zmínit, že dědičnost oprávnění mezi rolemi je tranzitivní. Ve výsledku to znamená, že např. role *technický ředitel* zdědí oprávnění jak od role *vedoucí projektu*, tak od role *programátor*.

Hierarchie rolí, která byla popsána v této podkapitole, může být formálně shrnuta do tří matematických výrazů:

- $RH \subseteq R \times R$ - uspořádaná M:N relace mezi rolemi,
- $Roles : SS \rightarrow \{R\}$ - funkce mapující každou relaci SS_i na množinu rolí (včetně podřízených rolí) $roles(SS_i) \subseteq \{R \mid \exists R' \geq R\ [(user(SS_i), R') \in UA]\}$, kde $\geq$ implikuje vztahy mezi rolemi,
- Oprávnění relace (včetně zděděných) jsou dána $R \in roles(SS_i)\ \{P \mid \exists\ R' \geq R\ [(P, R') \in PA]\}$.



Obrázek 2.5: Příklad hierarchie rolí u RBAC.

2.2.2.3 $RBAC_2$

Pokročilý model $RBAC_2$ obsahuje veškerou funkcionalitu modelu $RBAC_0$ a navíc zavádí koncept omezení (anglicky *constraints*). Koncept omezení je jedním z velmi důležitých aspektů RBAC, který slouží k restrikci vytváření vztahů mezi objekty. Typickým příkladem je situace, kdy v organizaci existují dvě exkluzivní role. V takovém případě není přípustné, aby měla nějaká osoba přiřazené obě role. Toho lze za pomocí konceptu omezení jednoduše dosáhnout pomocí vytvoření pravidla, které nedovolí přiřazení role, pokud daná osoba má již přiřazenou roli podmíněnou. Z toho vyplývá, že koncept omezení hraje klíčovou roli při uplatňování principu rozdělení pravomocí (viz 2.2.4).

Model $RBAC_2$ obsahuje veškeré prvky modelu $RBAC_0$. V důsledku toho je možné aplikovat omezení na všechny tyto prvky, jak je patrné z obrázku 2.4. Nicméně vzhledem k tomu, že modely $RBAC_1$ a $RBAC_2$ jsou vzájemně nezávislé, nelze provádět restrikce hierarchie rolí. Mezi nejpoužívanější omezení patří:

- *Prerekvizita rolí* - uživateli může být přiřazena role A pouze za předpokladu, že má již přiřazenou roli B,
- *Exkluzivita rolí* - uživatel může mít přiřazenou maximálně jednu roli z množiny vzájemně exkluzivních rolí,
- *Kardinalita vztahů* - počet objektů, které zahrnuje daný vztah může být omezen (např. uživatel může mít přiřazeny maximálně tři role).

2.2.2.4 $RBAC_3$

Konsolidovaný model $RBAC_3$ kombinuje oddělené modely $RBAC_1$ a $RBAC_2$. To ve výsledku znamená, že tento model obsahuje veškerou výše zmíněnou funkcionalitu, což lze spatřit na obrázku 2.4. Vzhledem k tomu, že model $RBAC_3$ je kompozicí pokročilých modelů, umožňuje vytvářet restrikce v souvislosti s hierarchií rolí. Typické použití restrikcí v souvislosti s hierarchií rolí je následující:

- *Omezení počtu přiřazených rolí* - omezení počtu *junior* nebo *senior* rolí, které mohou být do dané role přiřazeny,
- *Exkluzivita rolí* - dvě nebo více rolí nemají žádnou společnou *junior* nebo *senior* roli.

2.2.3 Princip minimálních oprávnění

Princip minimálních oprávnění (anglicky *Principle of Least Privilege*) je velmi důležitou součástí RBAC. Definice říká, že v rámci tohoto principu je vyžadováno, aby uživatel měl přiřazena přesně taková oprávnění, která jsou nezbytná k výkonu jeho zaměstnání[7]. To znamená, že uživatel by neměl mít přiřazena oprávnění, které nutně nepotřebuje. Před uplatněním principu minimálních oprávnění je vždy nutné identifikovat pracovní pozici uživatele a následně vydefinovat minimální množinu oprávnění, která jsou nezbytná pro vykonávání dané pozice. Tím, že uživatel nebude mít přiřazena oprávnění, která nejsou nezbytná pro plnění jeho povinností bude zajištěno, že nebude docházet k obcházení bezpečnostní politiky v organizaci.

2.2.4 Princip rozdělení pravomocí

Princip rozdělení pravomocí (anglicky *Separation of Duties*) je dalším principem uplatňovaným u RBAC. Jedná se o bezpečností politiku, ve které je odpovědnost za dokončení jednotlivých úloh a procesů rozdělena mezi více různých osob[8]. Účelem tohoto principu je snaha o případné zabránění podvodného jednání skrze účet jednotlivce tím, že odpovědnost za provedení dané operace je rozptýlena mezi více osob. To znamená, že k vykonání dané operace je nutná součinnost všech definovaných osob. Pro ilustraci lze použít příklad z reálného světa, kde máme dvě transakce. Jedna slouží k zadání platby, druhá ke schválení zadané platby. Při použití principu rozdělení pravomocí může každou z transakcí provést definovaná množina osob, ale žádná osoba nesmí být schopna provádět obě transakce.

Rozdělení pravomocí může být buď statického nebo dynamického typu. K vysvětlení rozdílů mezi těmito typu uvažujme, že máme proces platby, kde k jejímu provedení je nutné vykonat dvě různé transakce (zadání platby a schválení platby). Dále uvažujme, že pro každou transakci bude existovat samostatná role. U statického typu není možné, aby jedna osoba měla přiřazené obě role zároveň. Dynamický typ přináší větší flexibilitu, jelikož umožňuje, aby jedna osoba měla přiřazené obě role zároveň. To v tomto případě znamená, že jedna osoba může platby zadávat i schvalovat. Rozdělení pravomocí je zde dosaženo pomocí toho, že osoba nesmí schvalovat platbu, kterou sama vytvořila.

2.2.5 Vztah mezi složitostí RBAC modelu a složitostí organizace

Složitost RBAC modelu v dané organizaci je především vyjádřena mohutností množiny rolí. Proto je prováděn proces optimalizace, kdy je tento počet minimalizován na skutečně nezbytný počet rolí, které budou pokrývat veškeré přiřazené oprávnění uživatelům a zároveň bude minimalizován počet přiřazení takových rolí. Je nutné poznamenat, že v rámci této optimalizace mohou vzniknout role, které v *business* rovině nedávají smysl.

Nechť P je množina všech oprávnění, U je množina všech uživatelů a UP je relace mezi množinou U a množinou P popisující přiřazená oprávnění uživatelům. Za těchto podmínek může být definována taková množina rolí R, pro kterou platí následující výrok: $\forall(u \in U)\ \exists\ (R' \subseteq R) : RP(R') = UP(u)$, kde RP je relace mezi množinou R a množinou P. Pokud budeme uvažovat nejhorší možný případ, bude mohutnost množiny R rovna mohutnosti množiny U, matematicky zapsáno jako $|R| = |U|$. Tento případ nastane právě tehdy, když každý uživatel bude mít přiřazenu jedinečnou kombinaci oprávnění, pro kterou musí být vždy vytvořena samostatná role.

Proces vytváření množiny rolí v organizaci je nazýván jako *role engineering*. Tento proces zahrnuje extrahování rolí z existující infrastruktury organizace a pokud je prováděn manuálně, je poměrně časově náročný. Alternativním přístupem je zautomatizování tohoto procesu, kdy je množina rolí vytvořena pomocí grafové optimalizace (anglicky je tento proces označován jako *Role Engineering using Graph Optimisation*)[9]. Pomocí tohoto procesu je vytvořena minimální množina nezbytných rolí s využitím hierarchické struktury mezi samotnými rolemi, kde je zároveň minimalizován počet přiřazení. Jak již bylo řečeno dříve, během samotného procesu mohou vznikat role, které v *business* rovině nedávají smysl.

Nechť P je množina všech oprávnění, U je množina všech uživatelů, R je množina všech rolí, UP je relace mezi množinou U a množinou P popisující přiřazená oprávnění uživatelům, RP je relace mezi množinou R a množinou P popisující přiřazené oprávnění do rolí a UA je relace mezi množinou U a množinou R popisující přiřazené role uživatelům. Algoritmus procesu pro vytváření rolí pomocí grafové optimalizace lze popsat pomocí následujícího pseudokódu:

1: *pro množinu oprávnění každého uživatele vytvoř roli* ($\forall(u \in U)$ *vytvoř* $(r \in R)$: $RP(r) = UP(u)$)
2: *spočti ohodnocení vytvořené struktury jako součet počtu hran představujících přiřazení a počtu rolí (ohodnocení* $= |RP| + |UA| + |R|$)
3: **while** *může nastat optimalizace struktury rolí* **do**
4: *najdi dvě role*
5: *proveď rozdělení nebo sloučení rolí v závislosti na překrytí oprávnění*
6: *spočti ohodnocení vytvořené struktury jako součet počtu hran představujících přiřazení a počtu rolí (ohodnocení* $= |RP| + |UA| + |R|$)
7: **if** *nové ohodnocení* $>$ *staré ohodnocení* **then**
8: *vrať změny provedné v rámci operace z bodu 5*
9: **else**
10: *staré ohodnocení = nové ohodnocení*
11: **end if**
12: **end while**

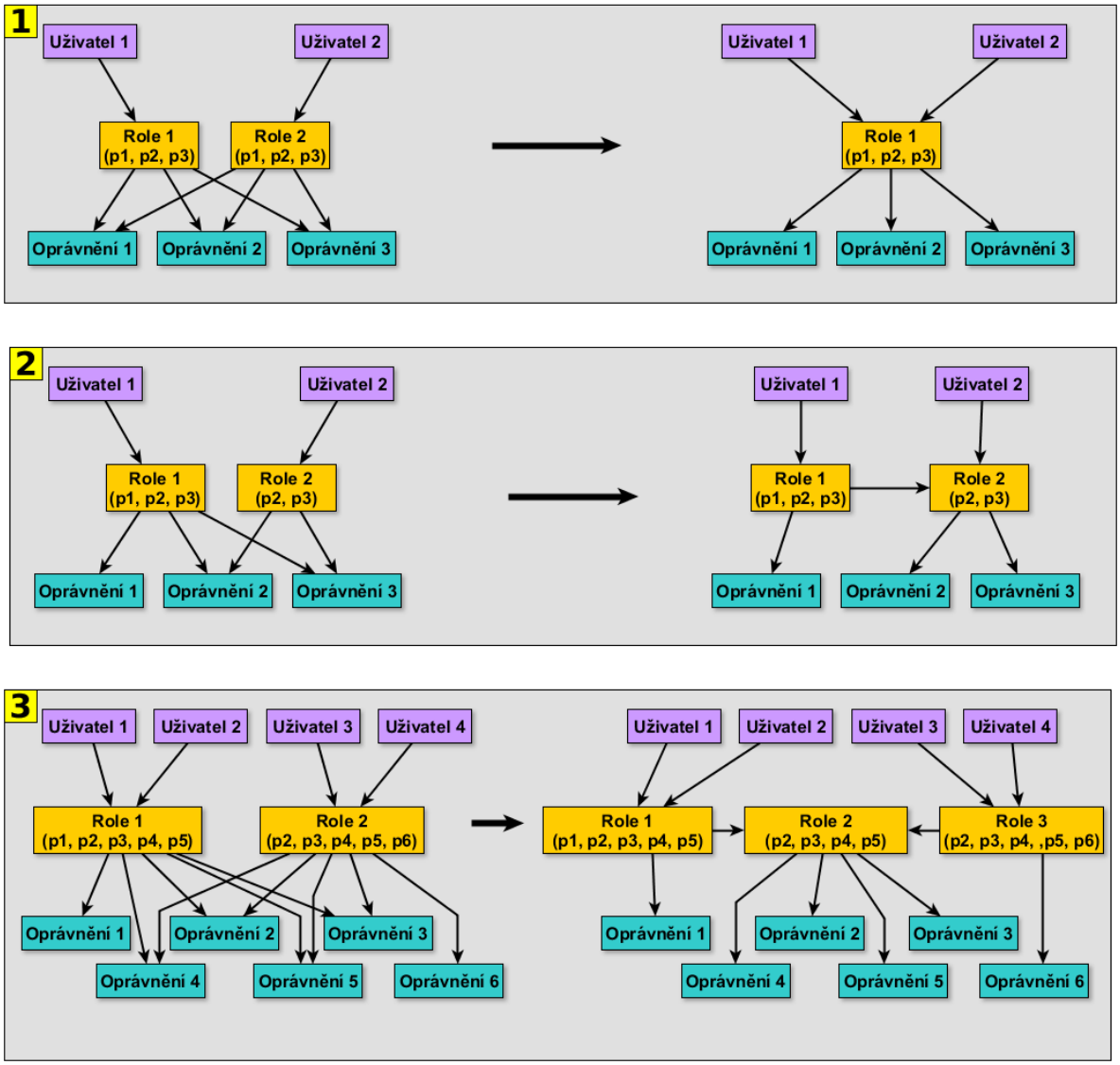
V rámci prvních kroků algoritmu je vytvořena iniciální struktura rolí a spočteno ohodnocení celé struktury, která je v následujících krocích optimalizována. Během provádění optimalizace jsou vždy identifikovány dvě role, u kterých je provedeno buď sloučení nebo rozdělení tak, aby bylo zlepšeno ohodnocení celé struktury. Typické situace, které nastávají během optimalizace, je možné spatřit na obrázku 2.6.

Situace označená jako *1* nastává právě tehdy, když jsou ve dvou rolích přiřazena stejná oprávnění. V tomto případě dojde ke sloučení rolí do jedné, která bude mít stejná oprávnění a bude přiřazena všem uživatelům, kteří měli přiřazené původní role. V rámci této optimalizace dochází ke snížení celkového počtu rolí a zároveň i ke snížení počtu přiřazení.

Situace označená jako *2* nastává právě tehdy, když jedna z rolí má přiřazenu podmnožinu oprávnění role druhé. V tomto případě dojde k vytvoření hierarchické struktury mezi rolemi, kdy role s přiřazenou podmnožinou bude *junior* role a druhá bude *senior* role. V rámci této optimalizace nedochází ke snížení celkového počtu rolí, ale dochází ke snížení počtu přiřazení. Za předpokladu, že jedna z rolí bude mít přiřazené jen a pouze jedno oprávnění, bude ohodnocení upraveného modelu totožné, jelikož dojde k vytvoření přiřazení mezi rolemi a k odstranění přiřazení mezi rolí a oprávněním.

Situace označená jako *3* je nejsložitější a nastává právě tehdy, když u obou rolí nastává překryv oprávnění, nicméně ani jedna z přiřazených množin oprávnění není podmnožinou druhé. V tomto případě dojde k vytvoření nové role, která bude mít přiřazeny překrývající se oprávnění. Individuální oprávnění budou ponechány v původních rolích. V rámci této optimalizace dochází ke zvýšení počtu rolí, nicméně taktéž dochází ke snížení počtu přiřazení. Tato optimalizace je prováděna právě tehdy, když obě role mají přiřazeny alespoň tři překrývající se oprávnění, jelikož přesně v takových případech se tato optimalizace vyplatí.

Samotný proces optimalizace je ukončen právě tehdy, když je již vytvořená struktura stabilní a nelze provést žádné další optimalizace. Tato situace nastane například



Obrázek 2.6: Typické případy nastávající během grafové optimalizace. Předloha [9].

tehdy, když budou analyzovány veškeré dvojice rolí.

Složitost modelu je nepochybně závislá na složitosti organizace. V nejhorším možném případě bude počet rolí roven počtu uživatelů a tedy složitost modelu bude přímo úměrná složitosti organizace. Ve chvíli kdy je vydefinována množina rolí a množina přiřazení existuje nenulová pravděpodobnost, že může být provedena optimalizace za účelem snížení mohutností zmíněných množin. Z tohoto důvodu nelze přesně specifikovat vztah mezi složitostí organizace a složitostí modelu, jelikož v každé organizaci bude po provedení optimalizace vytvořen jedinečný model respektující strukturu dané organizace.

2.2.6 Model vztahů mezi objekty

Cílem této podkapitoly je namodelovat vztahy mezi prakticky používanými objekty u RBAC. Mezi zmíněné objekty patří:

- *Uživatelská identita* - jednoznačně identifikovatelný subjekt, tj. osoba načtená ze

zdrojového systému,

- *Pozice* - pracovní pozice v organizaci, která je zpravidla vytvořená pro každé pracovní místo,
- *Sada* - množina rolí a oprávnění přiřazená k pozici,
- *Role* - funkce v organizaci zastupující množinu oprávnění,
- *Oprávnění* - popis interakce, kterou bude moci držitel vykonávat,
- *Skupina* - představuje množinu uživatelů v cílovém systému,
- *Účet* - uživatelský účet v cílovém systému.

Namodelování vztahů všech objektů lze spatřit na obrázku 2.7. Tento obrázek v podstatě znázorňuje mapování uživatelské identity na účty v cílových systémech. Role a oprávnění se obecně označují jako práva. Veškeré vazby mezi objekty jsou typu *M:N*, kromě vazby mezi sadou a pozicí, která je typu *1:N*. Vazba mezi uživatelskou identitou a pozicí je typu *M:N* proto, jelikož zaměstnanec může mít více pozic (například ředitel a předseda představenstva) a zároveň na jedné pozici může být přiřazeno více zaměstnanců. Díky tranzitivitě jsou uživatelským identitám přiřazována práva pomocí přiřazení zmíněných pozic, kde pozice mají práva přiřazené prostřednictvím sad. Nicméně práva mohou být uživatelským identitám přiřazována i individuálně. U rolí je umožněno vytváření hierarchické struktury (viz 2.2.2.2). Skupiny jsou přiřazovány do oprávnění a zastupují množinu uživatelských identit, které mají dané oprávnění přiřazené. Skupiny jsou umístěny v příslušných cílových systémech, kde jsou do těchto skupin shlukovány vytvořené uživatelské účty.

Čárkovaně jsou označeny vazby, které nevznikají přímo, ale na základě jiných asociací. Příkladem je vazba mezi uživateli a skupinami, která vzniká na základě přiřazených oprávnění.

2.3 Non-RBAC přístupy v modelování zabezpečení

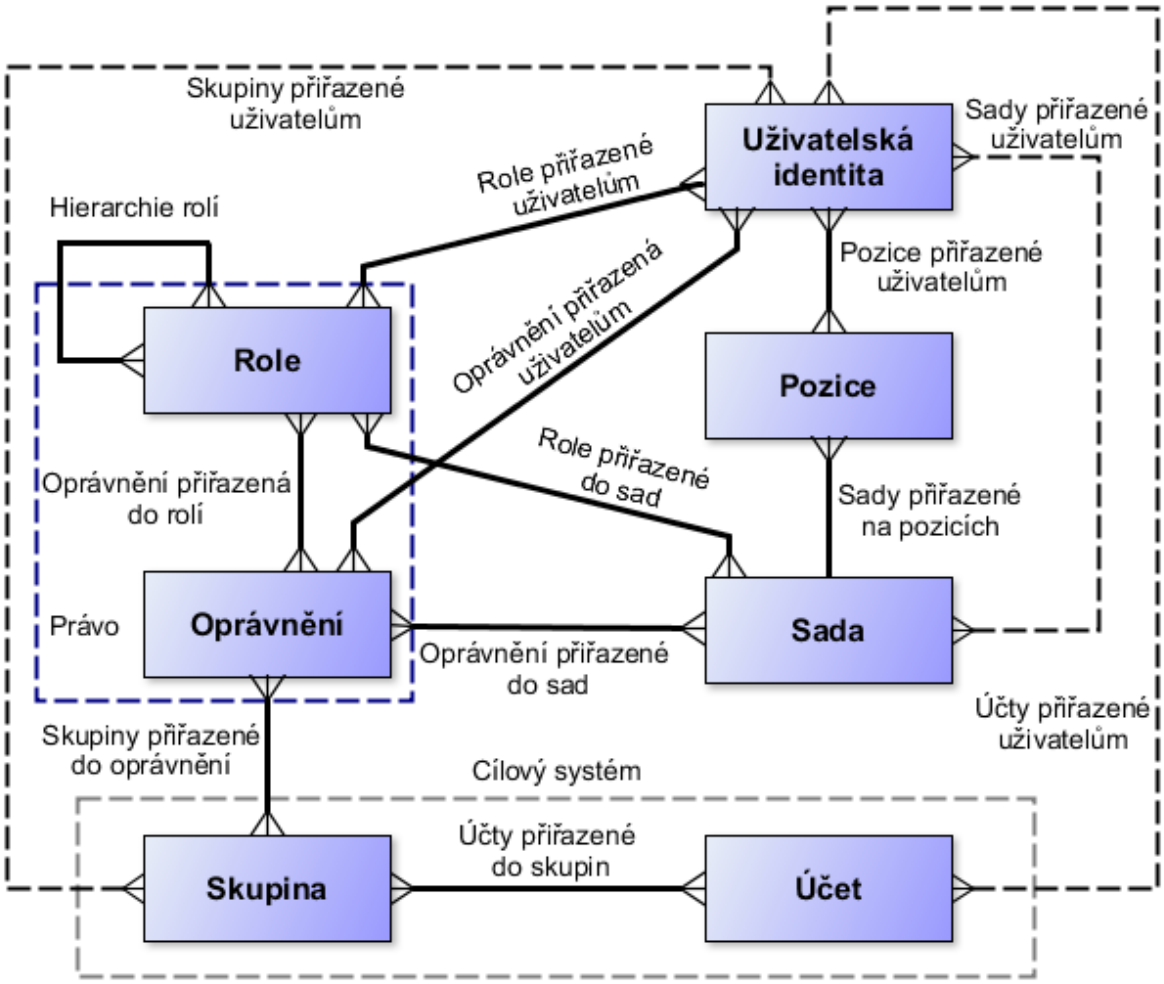
Zatímco v kapitole 2.2 je detailně rozebrán RBAC model, v této podkapitole budou v krátkosti popsány alternativní přístupy v modelování zabezpečení.

2.3.1 MAC

MAC je zkratka pro *mandatory access control*, česky doslova *mandatorní řízení přístupu*. Jedná se o bezpečnostní přístup používaný pro řízení přístupu k objektům v operačních systémech. Tento bezpečnostní přístup je používán hlavně u vládních či vojenských organizací, kde se klade velký důraz na bezpečnost[10].

U *MAC* platí, že jednotliví vlastníci objektů nemohou udělit nebo odepřít přístup k danému objektu dalším uživatelům. Přístupová politika všech objektů v systému je definována správcem systému a nemůže být měněna koncovými uživateli. Přístupová politika se definuje pomocí následujících dvou atributů:

- *Klasifikace* - určuje klasifikaci daného objektu (veřejný, důvěrný...),
- *Kategorie* - určuje, kterým uživatelským kategoriím bude objekt přístupný.



Obrázek 2.7: Model vztahů mezi prakticky používanými objekty u RBAC.

Zmíněné atributy jsou přiřazeny každému objektu a taktéž každému uživateli. Pokud chce uživatel provést nějakou operaci nad objektem, je zkontrolováno, zdali má příslušnou klasifikaci a je členem požadované kategorie. Pokud ano, bude mu umožněno provedení dané operace, v opačném případě bude provedení zamítnuto. Je důležité zmínit, že pokud bude vyhovující pouze jeden atribut uživatele, přístup mu nebude umožněn. Například uživateli s klasifikací přísně tajné nebude povolen přístup k objektu, pokud není členem požadované kategorie. Výhodou *MAC* je, že se jedná o velmi bezpečný přístup. Na druhou stranu správa je velmi náročná kvůli nutnosti vytvářet přístupové politiky pro každý objekt zvlášť.

2.3.2 DAC

DAC je zkratka pro *discretionary access control*. Opět se jedná o bezpečnostní přístup používaný pro řízení přístupu k objektům v operačních systémech. Na rozdíl od *MAC* (viz 2.3.1) je zde umožněno uživatelům řídit přístup k objektům, které vlastní. *DAC* je implicitním bezpečnostním přístupem pro řízení přístupu u operačních systémů. Pro řízení přístupu se využívá *ACL*.

ACL je zkratka pro *access control list*, česky doslova *seznam pro řízení přístupu*.

Koncept *ACL* je poměrně jednoduchý. Definice říká, že každý objekt, ke kterému by měl být řízen přístup, má přiřazenou množinu informací pro mapování subjektů, kteří chtějí přistupovat k danému objektu, a množinou operací, které může daný subjekt s objektem provádět[11]. Subjektem se rozumí uživatel nebo skupina uživatelů. To znamená, že na základě *ACL* lze vždy rozhodnout, zdali uživateli bude povoleno či zamítnuto provedení určité operace (čtení, zápis, modifikace, exekuce nebo mazání). Hlavní výhodou tohoto řešení je jednoduchost a podpora v převážné většině operačních systémů. Mezi nevýhody patří především fakt, že kontrola přístupu musí být prováděna vždy při přístupu k objektu. Mezi další nevýhody patří velmi špatná správa oprávnění, kde veškeré modifikace je nutné provádět individuálně nad každým objektem.

Na obrázku 2.8 lze spatřit reálné využití *DAC* pro řízení přístupu k souboru v linuxu. Z obrázku je patrné, že uživatelé *mysql* a *postgres* mají pro tento soubor povoleno provádět operace čtení (r) a zápis (w). Ostatní uživatelé, kromě vlastníka, mají povoleno provádět pouze operaci čtení.

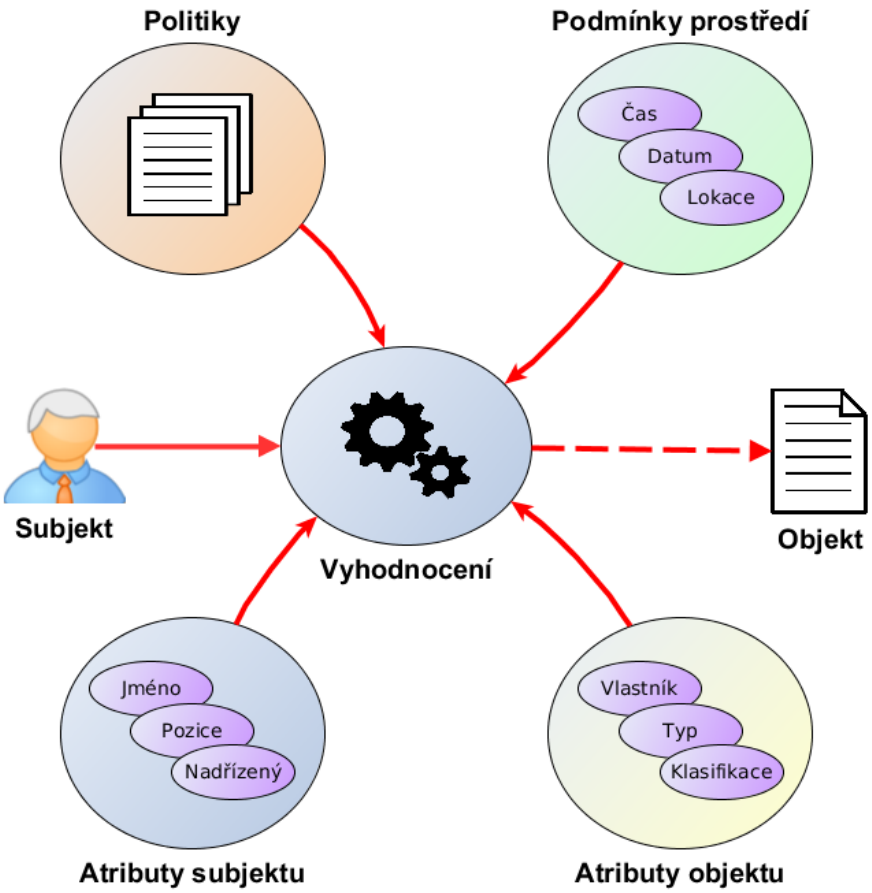
```
# file: test.dat
# owner: root
# group: root
user:mysql:rw-
user:postgres:rw-
group::r--
mask::rw-
other::r--
```

Obrázek 2.8: Příklad DAC pro soubor v linuxu.

2.3.3 ABAC

ABAC je zkratka pro *attribute based access control*, česky doslova *řízení přístupu na základě atributů*. Jedná se o takový bezpečnostní model, kde rozhodování o povolení nebo zamítnutí přístupu subjektu k nějakému objektu se provádí na základě vyhodnocení hodnot atributů, které mají daný subjekt a objekt přiřazené. Jedna z definicí říká, že *ABAC* je taková metoda pro řízení přístupu, kde žádosti subjektů na provádění operací nad objekty jsou schvalovány nebo zamítány na základě přiřazených atributů subjektu, přiřazených atributů objektu, podmínek prostředí a politik, které jsou specifikovány s ohledem na zmíněné atributy a podmínky[12]. Atributem může být v podstatě cokoliv, co může mít přiřazenou nějakou hodnotu (definováno jako pár *název* = *hodnota*).

Na obrázku 2.9 lze spatřit schematické znázornění bezpečnostního modelu *ABAC*. Z tohoto obrázku je patrné, že vyhodnocení přístupu je prováděno na základě třech skupin atributů. Způsob vyhodnocení atributů je definován příslušnou politikou. Politika může například definovat, že operace nad objektem může provádět pouze jeho vlastník (hodnota atributu *jméno* u subjektu se musí rovnat hodnotě atributu *vlastník* u objektu) pouze v úterý (hodnota atributu *datum* v proměnných prostředí musí mít odpovídající hodnotu). Z výše uvedeného vyplývá, že každý objekt musí mít přiřazenou alespoň jednu politiku. Politiky často bývají odvozeny z pravidel, které popisují procesy v organizaci.

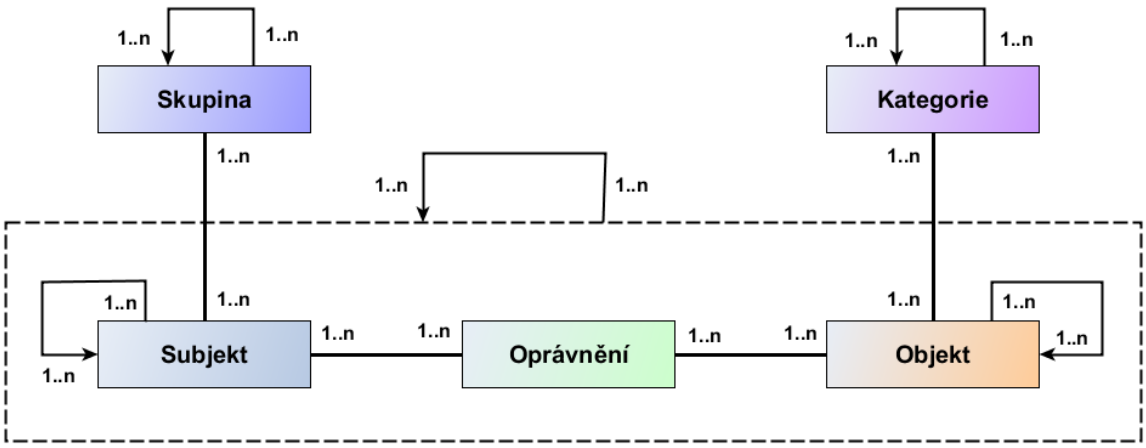


Obrázek 2.9: Schematické znázornění modelu ABAC. Předloha [12].

Hlavní výhoda modelu *ABAC* spočívá v tom, že v systému nemusí být definována množina uživatelů, kteří mají mít přístup k danému objektu. Přístup k objektu bude udělen všem žadatelům, kteří splňují požadovaná kritéria. Tato schopnost povolit přístup bez potřeby existence předem definovaného seznamu osob, které mají povolen přístup, hraje zásadní roli ve velkých organizacích se značnou fluktuací lidí.

2.3.4 RelBAC

RelBAC je zkratka pro *relation-based access control*, česky doslova *řízení přístupu na základě relací*. U tohoto modelu je každé oprávnění namodelováno jako binární relace mezi subjektem a objektem, kde oprávnění specifikuje povolené operace nad objektem. Z toho vyplývá, že subjektu bude povoleno provedení operace nad objektem právě tehdy, když existuje relace mezi tímto subjektem a objektem, která povoluje provedení dané operace. Díky oddělení oprávnění od subjektů a objektů lze snadno modelovat dynamické změny u oprávnění. Vzhledem k binární relaci mezi objekty a subjekty se *RelBAC* označuje jako *entitně-relační* (anglicky *entity-relationship*) model. Díky tomu tento model umožňuje přesně specifikovat kardinalitu vztahů mezi subjekty a objekty. Subjekty a objekty mohou být členěny do složitých struktur (typicky hierarchická struktura), kde tyto struktury nemusí být stabilní, jelikož *RelBAC* umožňuje dynamické provádění změn v relacích. Na obrázku 2.10 lze spatřit znázornění entitně-relačního diagramu pro RelBAC model.



Obrázek 2.10: Entitně-relační diagram RelBAC modelu. Předloha [13].

Z obrázku je patrné, že entitně-relační diagram RelBAC modelu se skládá z následujících komponent:

- *Subjekt* - aktivní entita jako je člověk, proces nebo zařízení,
- *Objekt* - pasivní entita obsahující data,
- *Oprávnění* - popis typu interakce, kterou subjekt může mít s objektem,
- *Skupina* - jakákoliv množina skupin nebo subjektů,
- *Kategorie* - jakákoliv množina kategorií nebo objektů.

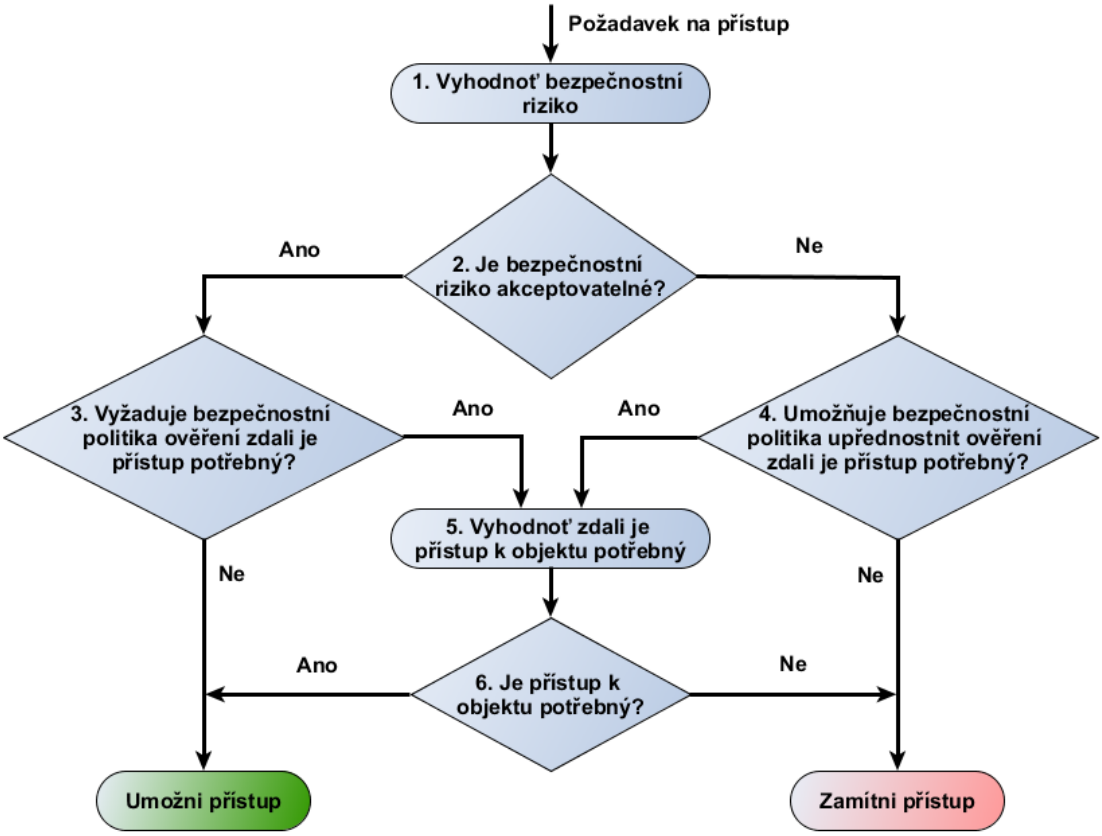
Každá z komponent kromě komponenty *oprávnění* může být vnitřně uspořádaná do hierarchické struktury. Pro zdárné řízení přístupu se předpokládá existence pravidel pro řízení přístupu, kde pravidlem pro řízení přístupu se rozumí oprávnění, které má přiřazenou určitou množinu subjektů a objektů. Existují dva různé typy pravidel pro řízení přístupu. Prvním je takzvané *user-centric* pravidlo, které umožňuje určité podmnožině všech subjektů vykonat operaci nad určitou množinou objektů. Druhým pravidlem je takzvané *object-centric* pravidlo, které specifikuje určitou podmnožinu všech objektů, nad kterými může určitá podmnožina subjektů provádět operace.

Mezi hlavní výhody *RelBAC* modelu patří fakt, že oprávnění jsou oddělená od subjektů a objektů, kde v důsledku toho mohou být relace mezi zmíněnými objekty dynamicky měněny. Mezi další výhody patří možnost přesné specifikace kardinality vazeb.

2.3.5 RAdAC

RAdAC je zkratka pro *risk-adaptive access control*, česky doslova *řízení přístupu přispůsobující se riziku*. Jedná se o poměrně nový model, který rozhodování o povolení nebo zamítnutí přístupu k objektu provádí na základě informací o možných rizicích spojených s tímto přístupem a informací o tom, zdali je tento přístup pro daný subjekt potřebný. Přístupem se zde rozumí provedení nějaké operace nad objektem. Informace o rizicích se týkají především typu přístupu a prostředí, z kterého je požadováno provedení operace. Například provedení operace z počítače, který uživatel používá již

několik hodin znamená menší riziko než provedení stejné operace z neznámého počítače. Ověření, zdali je přístup pro daný subjekt potřebný provádí odpovědná osoba v organizaci, kde v důsledku této kontroly má subjekt přístup jen a pouze k takovým objektům, které skutečně potřebuje. Na obrázku 2.11 lze spatřit kompletní diagram pro rozhodování o povolení nebo zamítnutí přístupu k objektu.



Obrázek 2.11: Rozhodovací diagram modelu RAdAC. Předloha [14].

Z rozhodovacího diagramu modelu *RAdAC* je jasně patrné, že nejdříve dochází k určení a vyhodnocení bezpečnostního rizika, které souvisí vždy s aktuálním požadavkem na přístup k objektu. Riziko se určuje především v oblasti lidských zdrojů, používané techniky a prostředí, z kterého pochází požadavek. Pokud je bezpečnostní riziko akceptovatelné, dojde k určení, zdali je podle bezpečnostní politiky nutné ověřit potřebnost přístupu subjektu k danému objektu. V rámci tohoto ověření může být zkoumáno několik různých faktorů, které jsou vždy určeny bezpečnostní politikou. Příkladem takových faktorů může být členství subjektu ve skupině nebo lokace subjektu. Pokud je výsledek ověření pozitivní, dojde k povolení přístupu, v opačném případě dojde k zamítnutí přístupu. Přístup může být povolen i tehdy, pokud bezpečnostní riziko není akceptovatelné ale bezpečnostní politika umožňuje upřednostnit ověření, zdali subjekt daný přístup potřebuje.

Hlavní výhoda tohoto modelu spočívá ve velké flexibilitě, jelikož proces řízení přístupu je vždy ovlivňován bezpečnostním rizikem, které odráží aktuální podmínky.

Kapitola 3

Analýza

Tato kapitola se zabývá analýzou modelových organizací definovaných v podkapitole 3.1. Analýza probíhala pomocí řízených rozhovorů s kontaktními osobami v daných organizacích. Za účelem získání dostatečného povědomí o fungování cílených organizací a potvrzení informací získaných prostřednictvím řízených rozhovorů bylo dále provedeno dotazníkové šetření na dalším vzorku organizací, jejichž zástupci byli ochotni odpovědět na zaslaný dotazník. Získané informace byly použity k definování vhodného *RBAC* modelu (viz podkapitola 4.1).

Dále se tato kapitola zabývá analýzou požadavků na systém ze strany zadavatele. Stejně jako v předchozím případě probíhala analýza požadavků formou řízených rozhovorů. Získané informace byly použity k definování funkčních a nefunkčních požadavků na systém.

V neposlední řadě byla v této kapitole provedena analýza existujících řešení.

3.1 Modelové organizace

V souladu se zadáním práce byly určeny dvě modelové organizace, jejichž fungování bylo analyzováno. Kontaktní osoby si nepřáli uvedení konkrétních názvů organizací v práci. Autor práce tomuto přání vyhověl, nicméně si dovoluje deklarovat, že uvedené informace jsou skutečné a pravdivé.

První z analyzovaných organizací je organizace, která vznikla počátkem roku 2008 a její specialisté poskytují komplexní služby v oblasti vývoje software a informačních systémů na trzích střední a západní Evropy. Z hlediska velikosti se jedná o malou organizaci, která zaměstnává do 25 zaměstnanců, nicméně rozvoj a růst je velmi dynamický. Hlavní sídlo organizace je v Praze, pobočka se nachází v Českých Budějovicích.

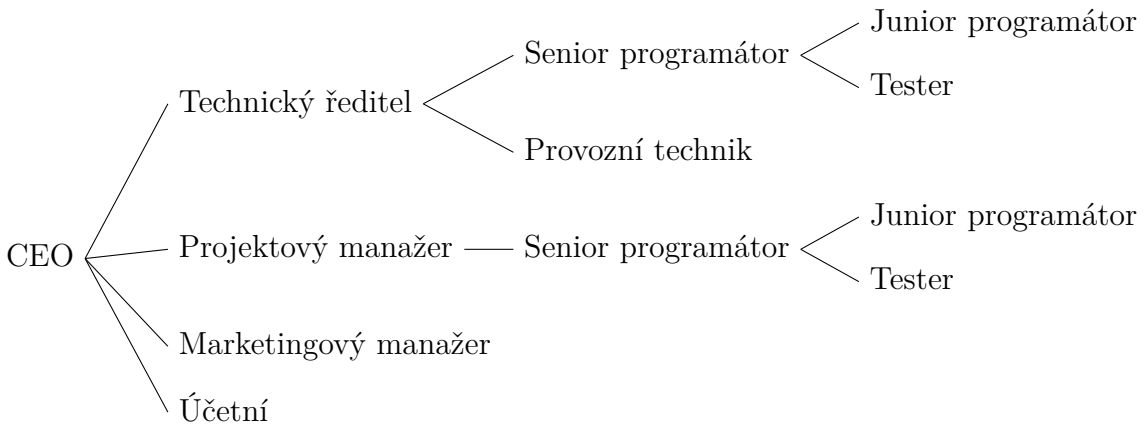
Druhou z organizací je oficiální vydavatel mezinárodních identifikačních průkazů studentů, mládeže do 26 let a pedagogické veřejnosti pro Českou republiku. Z hlediska velikosti se jedná o menší organizaci, která zaměstnává do 30 zaměstnanců. Sídlo organizace je v Praze.

3.1.1 Výsledky řízených rozhovorů

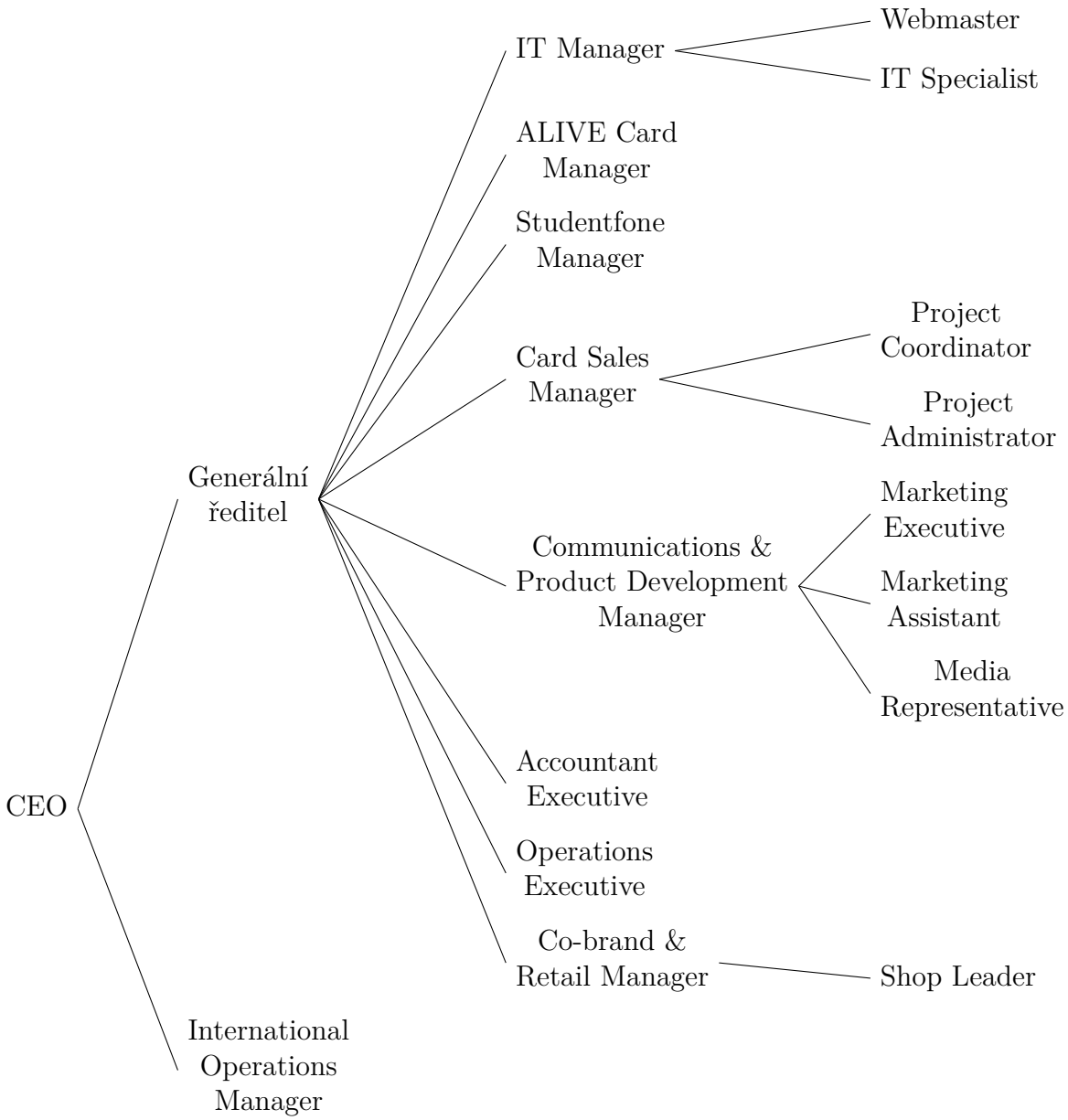
Řízené rozhovory probíhaly podle agendy definované autorem práce. Agenda obsahovala klíčové body, které se autor práce snažil prodiskutovat s kontaktními osobami. Veškeré zajímavé informace si autor práce vždy poznamenal do svých interních poznámek, které byly následně využity pro vyvození zjištění po ukončení rozhovoru. Mezi

zásadnější zjištění patří:

- U obou organizací se překrývají oprávnění mezi různými pozicemi,
- V první organizaci pracuje na alespoň jedné pozici více zaměstnanců,
- Pro první organizaci platí, že se liší oprávnění zaměstnanců, kteří pracují na stejné pozici (hlavní příčinou je rozdílné projektové zaměření),
- V druhé organizaci pracuje na každé pozici právě jeden zaměstnanec,
- Pro obě organizace platí, že oprávnění uživatelů jsou měněna spíše méně často,
- Úložištěm uživatelů u první organizace je *OpenLDAP*,
- První organizace používá HR systém *FlexiBee*,
- Pro první organizaci má zcela zásadní význam aplikace *Redmine*:
  - Autentizace uživatelů probíhá prostřednictvím LDAP,
  - Autorizace uživatelů probíhá na základě přiřazených skupin,
  - Skupiny jsou přiřazovány napřímo uživatelům podle potřeby,
  - Neexistuje přehledná evidence přiřazených skupin,
  - Skupiny zpravidla nejsou uživatelům odebírány (i pokud již nepotřebují přístup k nějakému zdroji).
- Uložištěm uživatelů u druhé organizace je *Active Directory* a *SQL tabulka*,
- Druhá organizace používá větší množství aplikací s autentizací/autorizací uživatelů (je jich více než 15),
- Druhá organizace často využívá aplikaci *BellaDati*:
  - Autentizace uživatelů probíhá vůči *Active Directory*,
  - Autorizace uživatelů probíhá na základě přiřazených skupin,
  - Skupiny jsou přiřazovány napřímo uživatelům podle potřeby.
- U obou organizací se vyskytují vztahy nadřízený/podřízený,
- První organizace má následující organizační strukturu:



- Druhá organizace má následující organizační strukturu:



3.2 Výsledky dotazníkového šetření

Dotazníkové šetření proběhlo za pomoci formuláře cloudové služby *Google Drive* od společnosti Google. Autor práce oslovil prostřednictvím elektronické pošty organizace, o kterých má alespoň částečné povědomí a které splňují požadavky na cílové organizace. Přesto, že samotný dotazník byl deklarován jako anonymní, a tedy společnosti se nemusely obávat případného vyzrazení informací o jejich fungování, byla ochotna odpovědět jen menšina dotázaných. Autor práce se taktéž domnívá, že nezájem o vyplnění dotazníku plynul z obecného nezájmu o dané téma. Nicméně i tak autor práce považuje dotazníkové šetření za úspěšně provedené a získané informace potvrdily předpoklady získané během řízených rozhovorů se zástupci modelových organizací.

Samotný dotazník se skládal z několika otázek týkajících se využívaných aplikací, struktury organizace a správy oprávnění. Graficky zpracované výsledky je možné spatřit

v příloze A. Z výsledků je jasně patrné, že se podařilo oslovit několik převážně menších organizací, které mají vždy do 50 zaměstnanců. Mezi nejzajímavější zjištění patří fakt, že všichni respondenti se shodují, že se v jejich organizaci překrývají oprávnění mezi různými pozicemi. Tento fakt koreluje s dalším zjištěním, kdy organizační struktura je vždy víceúrovňová, a tedy existují přímí nadřízení a podřízení. Dá se tedy předpokládat, že překryv oprávnění je způsobem vztahem mezi nadřízeným a podřízeným. Drtivá většina respondentů deklaruje, že na některé pozici pracuje více zaměstnanců, kde jejich oprávnění se zpravidla částečně liší. Samotná oprávnění jsou uživatelům měněna, nicméně spíše s menší intenzitou. Řízení přístupu uživatelů ke zdrojům je nejčastěji prováděno na základě přiřazených skupin. V oblasti používaných úložišť uživatelů mají jasnou majoritu LDAP implementace.

3.3 Analýza existujících řešení

3.3.1 OpenIDM

OpenIDM je open-source IDMS napsaný v jazyce Java, kde zdrojový kód je dostupný pod licencí CDDL[1]. Systém je spravován komunitou s názvem *ForgeRock* a v době vzniku této práce byla poslední stabilní verze 2.1.0.

OpenIDM umožňuje synchronizaci objektů (tzv. *managed objects*) mezi datovým úložištěm a cílovými systémy (dále jen provisioning), základní uživatelský self-service, synchronizaci hesel uživatelských účtů, vytváření uživatelských účtů, auditní logy a definici workflow. Data jsou uchovávána v některé z podporovaných databází. Pro produkční prostředí jsou podporovány databáze *MySQL*, *MS SQL Server* a *Oracle* a dále lze neoficiálně použít další databáze připojené prostřednictvím *JDBC*[2]. Pro vývojové účely je využívána *NoSQL* databáze *OrientDB*. Pro manipulaci s objekty může být kromě databázové úrovně využito poskytované REST API[3]. *OpenIDM* umožňuje specifikovat business pravidla, která mají být uplatňována během provisioningu. Pro implementaci pravidel je využíván objektově orientovaný skriptovací jazyk Javascript. Provisioning může být vyvolán nějakou událostí (např. vytvoření objektu) nebo může být pravidelně spouštěn pomocí cron démonu. Schematické znázornění jednotlivých komponent je možné spatřit na obrázku 3.1.

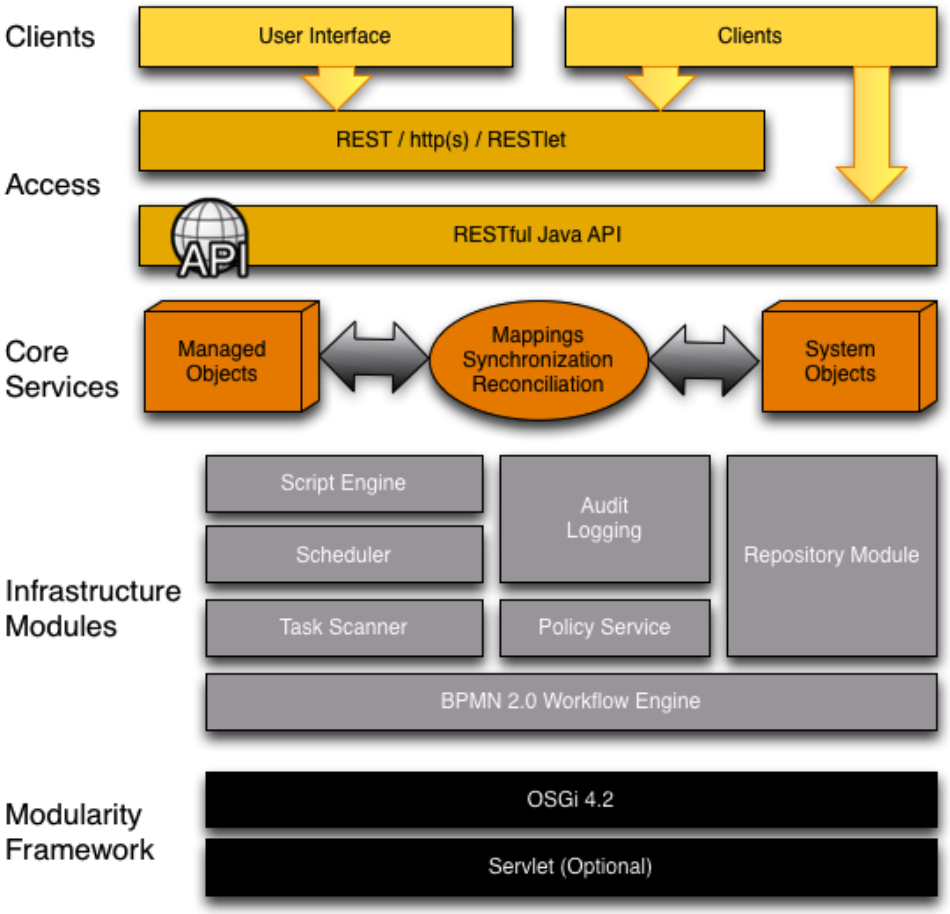
Pro integraci se systémy se využívají *OpenICF* konektory. Tyto konektory byly odvozeny od konektorů společnosti Sun (Sun ICF) a jsou spravovány *ForgeRock* komunitou a organizací *Evolveum*. Prostřednictvím těchto konektorů je možné provést integraci například s *LDAP* implementací, *Google Apps* nebo *Oracle ERP*. Konfigurace konektorů se provádí prostřednictvím již zmíněných textových konfiguračních souborů.

Veškerá konfigurace je prováděna prostřednictvím textových konfiguračních souborů ve formátu JSON. Pomocí těchto souborů se například definuje mapování objektů na databázové tabulky, nastavení parametrů jednotlivých konektorů nebo popis atributů zpracovávaných během provisioningu.

[1] Open source iniciative. Common Development and Distribution License (CDDL-1.0) [online]. [cit. 2014-07-12]. Dostupné z: http://opensource.org/licenses/CDDL-1.0.

[2] Součást programovacího jazyka Java, která definuje jednotné rozhraní pro přístup k různým relačním databázím.

[3] Rozhraní používané pro snadný přístup ke zdrojům (resources) pomocí základních metod protokolu HTTP. Každý zdroj má vlastní identifikátor (URI) a může mít různé reprezentace (XML, HTML, JSON...).



Obrázek 3.1: Schéma komponent OpenIDM systému. Zdroj [15].

OpenIDM dále poskytuje omezené grafické uživatelské rozhraní. Prostřednictvím tohoto rozhraní je možné provádět správu uživatelů a řídit členství ve skupinách. Dále je možné řídit životní cyklus jednotlivých instancí workflow. Přístup do rozhraní je zabezpečen pomocí formulářové autentizace.

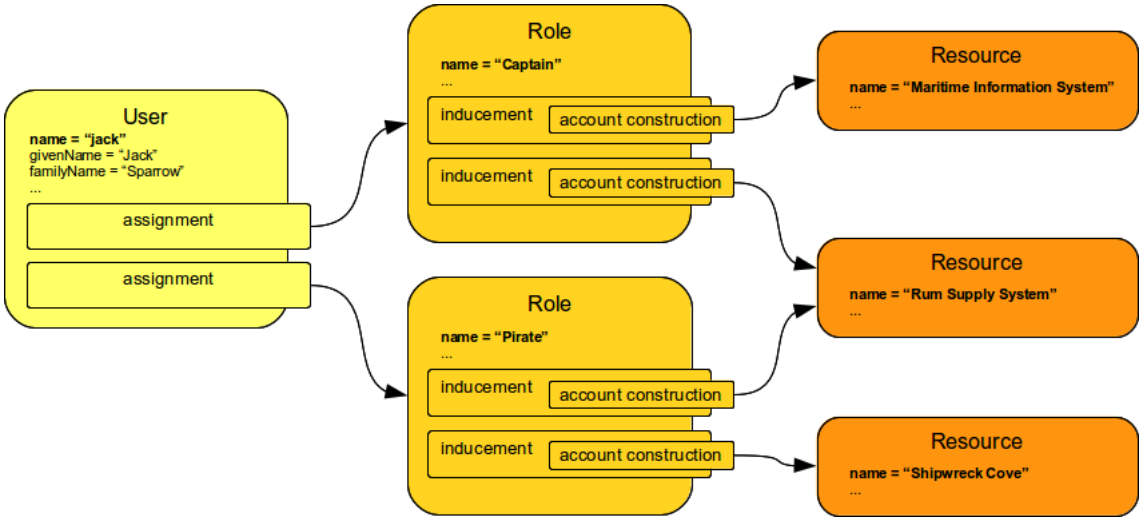
3.3.2 MidPoint

MidPoint je open-source IDMS napsaný v jazyce Java, kde zdrojový kód je dostupný pod licencí Apache License, Version 2.0[4]. Systém je spravován organizací s názvem *Evolveum* a v době vzniku této práce byla poslední stabilní verze 3.0.

Mezi hlavní funkcionality systému *midPoint* patří provisioning engine, synchronizace uživatelských identit, workflow engine, auditní logy, reporty a podpora modelu *RBAC*. Data jsou uchovávána v některé z podporovaných databází (*H2*, *MySQL*, *MS SQL Server*, *PostgreSQL* a *Oracle*). Stejně jako u *OpenIDM* lze od verze 3.0 využít pro manipulaci s objekty REST API. *MidPoint* taktéž umožňuje specifikovat business pravidla, která mají být uplatňována během provisioningu, kde pro implementaci pravidel je využíván jazyk Groovy. Dále je implementována podpora vytváření základních reportů (např. report účtů zobrazující přiřazené role).

[4]THE APACHE SOFTWARE FOUNDATION. Apache License, Version 2.0 [online]. 2004 [cit. 2014-07-13]. Dostupné z: http://www.apache.org/licenses/LICENSE-2.0.html

MidPoint má implicitní podporu modelu *RBAC*. Klasický model, kdy uživatel získá veškerá oprávnění z přiřazené role je mírně rozšířen tím, že v rolích je možné specifikovat atributy, které následně určují, kdy a jak bude role používána. Vlastnosti role se tedy můžou měnit v závislosti na uživateli, kterému je přiřazena. Toto rozšíření bylo zavedeno především kvůli velkým organizacím, v kterých je nutné spravovat tisíce rolí, kde správa takového počtu rolí je složitá. Zavedením zmíněného rozšíření je možné snížit počet rolí a část logiky umístit přímo na role. Schematické znázornění *RBAC* modelu u systému *midPoint* je vyobrazeno na obrázku 3.2.



Obrázek 3.2: Schéma RBAC modelu u midPoint systému. Zdroj [5].

Pro integraci se systémy se využívají výše zmiňované *OpenICF* konektory. V důsledku toho je možné provést integraci se stejnými systémy jako u systému *OpenIDM*.

MidPoint taktéž poskytuje grafické uživatelské rozhraní. Prostřednictvím tohoto rozhraní je možné provádět správu uživatelů, rolí a cílových systému. Dále je možné řídit životní cyklus jednotlivých instancí workflow a vytvářet reporty. Přístup do rozhraní je zabezpečen pomocí formulářové autentizace.

3.4 Logický rámec projektu

Metoda logického rámce (anglicky *logical framework approach*) je používána jako přehledný nástroj pro navržení a následné uspořádání základních charakteristik projektu pomocí přesně definované struktury. Tato struktura se nazývá matice logického rámce (anglicky *logical framework matrix*). Definice říká, že metoda logického rámce je nástrojem pro plánování projektů s využitím stanovených cílů, kde tato metoda může být použita pro analýzu, sledování a vyhodnocení projektů[16].

[5]SEMANCIK, Radovan. Advanced RBAC [online]. 2014 [cit. 2014-07-13]. Dostupné z: https://wiki.evolveum.com/display/midPoint/Advanced+RBAC

Popis projektu	Objektivně ověřitelné ukazatele	Prostředky ověření	Vnější předpoklady
Cíl projektu *Zefektivnit a zpřehlednit správu identit u malých a středních organizací.*	*Organizace využívající systém ušetří zdroje, které souvisí s IDM procesy.*	*Série řízených rozhovorů s kontaktními osobami z organizací, které využívají systém.*	
Účel projektu *Využití prostředků IT k zautomatizování IDM procesů v organizaci a dále k přesnému zobrazení veškerých dostupných informací, které souvisejí s těmito procesy.*	*IDM procesy v organizaci jsou plně či alespoň částečně zautomatizovány. Uživatelé systému mají jasný přehled o informacích souvisejících s IDM procesy.*	*Rešerše časové náročnosti IDM procesů, které byly zautomatizovány. Data od uživatelů systému získané pomocí dotazníku.*	*Provedení testů náročnosti IDM procesů v organizacích. Ochota uživatelů zodpovědět otázky v dotazníku.*
Výstupy *Uživatelská dokumentace, instalační dokumentace, dokumentace kódu aplikace. Otestovaný aplikační backend spolu s grafickým uživatelským rozhraním aplikace.*	*Existuje kompletní dokumentace. Vytvořená prototypová aplikace odpovídá návrhu, formálním a neformálním požadavkům.*	*Repozitář projektu. Analýza naplnění definovaných požadavků na systém.*	*Implementace probíhá podle stanoveného plánu. Při implementaci je dodržován vytvořený návrh implementace.*
Činnosti • *Analýza modelových organizací,* • *Analýza požadavků zadavatele,* • *Určení funkčních a nefunkčních požadavků,* • *Návrh řešení,* • *Implementace první fáze řešení,* • *Průběžné testování a tvorba dokumentace,* • *Vedení iterativního vývoje až do ukončení vývoje*	**Prostředky pro realizaci** • *Návrhové diagramy,* • *Nezbytné technologie,* • *Součinnost zadavatele,* • *Prostředí pro vývoj, testování a produkční nasazení.*	**Časové údaje k realizaci** • *Kompletní analýza a návrh řešení - 03/2014,* • *Implementace prototypu - 11/2014,* • *Testovací provoz - 11/2014 ,* • *Zahájení dalších iterací vývoje - 12/2014.*	*Dodržení stanovených termínů realizace jednotlivých činností. Komunikace se zadavatelem bude probíhat bez větších obtíží. Dostatečná odbornost implementátora ve zvolených technologiích. Použité technologie jsou vhodně zvolené.*

3.5 Požadavky na systém

V této podkapitole jsou popsány veškeré požadavky na systém, které musí být reflektovány při vývoji aplikace. Tyto požadavky byly vyprodukovány na základě informací získaných během řízených rozhovorů se zadavatelem. Během zmíněných rozhovorů byly využity informace získané během analýzy existujících řešení (viz 3.3).

3.5.1 Funkční požadavky

Funkční požadavky definují funkcionality systému, které bude systém obsahovat po jeho implementaci. Základní funkční požadavky jsou:

- Systém je možné používat na platformách, které jsou v dnešní době nejrozšířenější (Windows, Linux, Mac OS),
- Autentizace v aplikace se provádí pomocí uživatelského jména a hesla,
- Uživatel může žádat o přiřazení a odebrání rolí a oprávnění,
- Uživatel může měnit hesla u svých účtů,
- Systém obsahuje tři uživatelské role (administrátor, manažer, uživatel),
- Uživatel může provádět právě takové operace, které jsou odvozeny z jemu přiřazené role,
- Systém přehledně zobrazuje veškeré uchovávané objekty,
- Systém umožňuje integraci se zdrojovým systémem,
- Systém umožňuje obousměrnou integraci s cílovými systémy,
- Systém umožňuje import uživatelských identit z datového souboru,
- Systém využívá schvalovací workflow pro asociování datových objektů,
- Systém umožňuje evidenci *orphan*[6] účtů,
- Systém vytváří auditní logy[7],
- Systém využívá centrální datové úložiště pro uchovávání datových objektů,
- Systém může využívat více přihlášených uživatelů v jeden okamžik.

[6] Jedná se o účty, které jsou v IDMS vytvářeny pomocí provisioningu a nemají vazbu na příslušné uživatelské identity.

[7] Jedná se o logy, které jsou vytvářeny při jakékoliv manipulaci s datovými objekty. Na jejich základě lze dohledat historické hodnoty atributů a je zajištěna nepopiratelnost.

3.5.2 Nefunkční požadavky

Nefunkční požadavky definují vlastnosti prostředí, ve kterém bude systém nasazen a technologie, které budou využity během implementace. Základní nefunkční požadavky jsou:

- Serverová část aplikace je implementována pomocí platformy *Java EE*[8], kde pro provoz této části je nezbytné, aby bylo na serveru nainstalováno běhové prostředí *Java 7*.
- Uživatelské rozhraní má formu webové aplikace. V důsledku toho nemusí uživatelé instalovat žádný software ale postačí jim některý z běžně používaných internetových prohlížečů[9],
- Data jsou uchovávána v relační databázi (díky pozitivní zkušenosti autora práce bude použita open-source relační databáze *PostgreSQL*[10]),
- Systém je škálovatelný,
- Systém splňuje základní požadavky na spolehlivost, dostupnost a odezvu,
- Systém splňuje základní požadavky na bezpečnost,
- Zdrojový kód je srozumitelný a dobře dokumentovaný,
- Systém umožňuje snadnou rozšiřitelnost a modifikovatelnost.

3.5.3 Uživatelské role

Aplikaci budou využívat uživatelé, kteří budou mít přiřazenou jednu z následujících rolí:

- *Administrátor* - uživatelé této skupiny mají neomezený přístup ke všem datovým objektům v aplikaci, kde tyto objekty mohou libovolně editovat. Administrátoři mohou vytvářet libovolné datové objekty včetně uživatelských identit. Dále mají neomezený přístup k auditním logům a neomezený přístup ke schvalovacím workflow.
- *Manažer* - uživatelé této skupiny mají neomezený přístup k datovým objektům, ve kterých plní roli nadřízeného nebo garanta. Tyto objekty mohou libovolně editovat. Manažeři mohou vytvářet libovolné datové objekty včetně uživatelských identit pro své podřízené. Dále mají přístup ke svým auditním logům a schvalovacím workflow a dále auditním logům a schvalovacím workflow svých podřízených.
- *Uživatel* - uživatelé této skupiny mají omezený přístup k datovým objektům, kde editovat mohou pouze své účty a objekty, u kterých jsou vedeni jako garanti. Tito uživatelé nemohou vytvářet žádné datové objekty. K ostatním datovým objektům

[8] Java EE at a Glance. [online]. [cit. 2014-07-11]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/overview/index.html

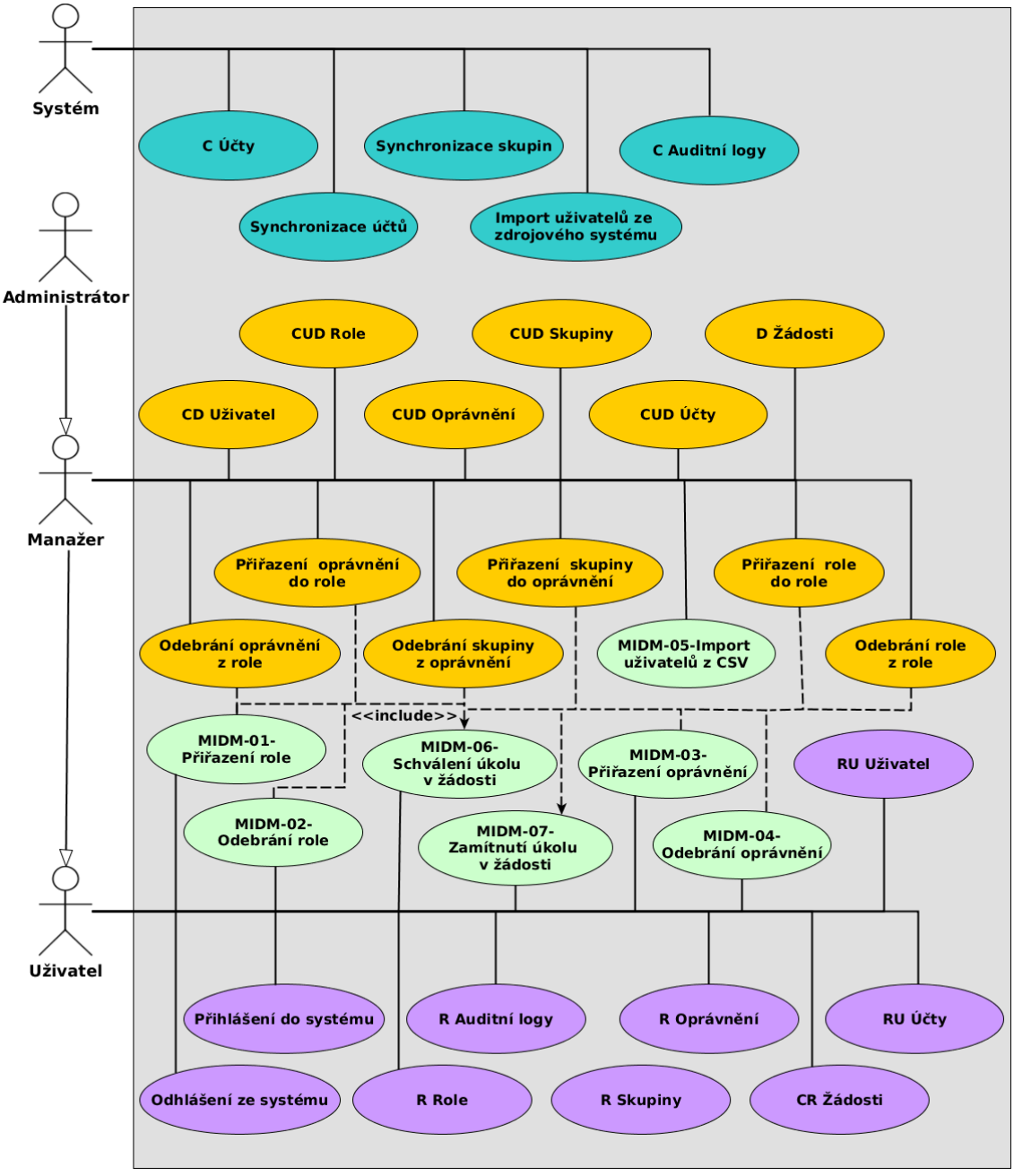
[9] Web Browser Market Share In June 2014. [online]. [cit. 2014-07-11]. Dostupné z: http://www.w3counter.com/globalstats.php?year=2014&month=6

[10] PostgreSQL: The world's most advanced open source database. [online]. [cit. 2014-07-11]. Dostupné z: http://www.postgresql.org/.

mohou přistupovat v *read-only* režimu. Dále mohou vytvářet žádosti a zobrazovat auditní logy, které zobrazují jejich provedené akce.

3.5.4 Případy užití

Na základě výše uvedených funkčních a nefunkčních požadavků byly identifikovány případy užití (anglicky *Use Case*), které bude vytvořený systém podporovat. Identifikované případy užití jsou namodelovány pomocí diagramu případů užití na obrázku 3.3. Případy užití, které se týkají objektů RBAC modelu byly do diagramu doplněny po definování vhodného modelu v podkapitole 4.1. Pro zeleně označené případy užití byly definovány scénáře případů užití, které lze nalézt v příloze D.



Obrázek 3.3: Zjednodušený Use Case diagram.

Kapitola 4

Návrh

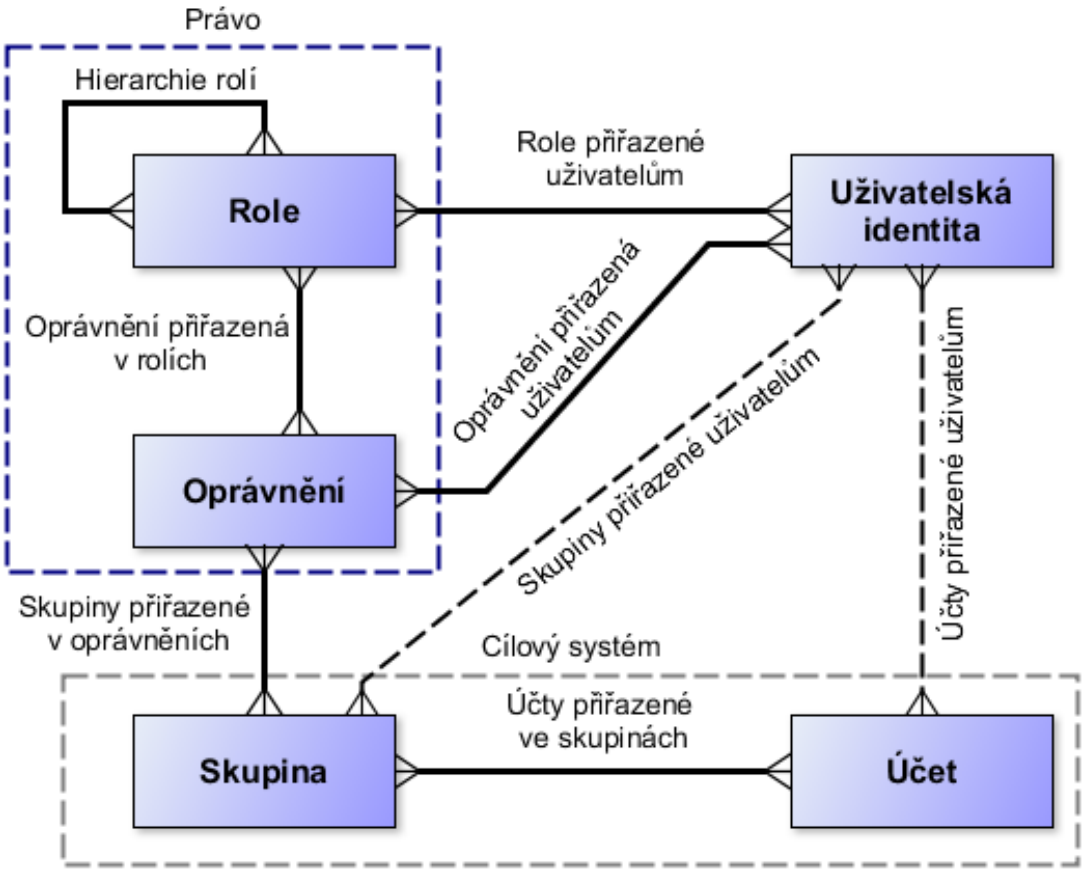
4.1 RBAC model

Na základě analýzy modelových organizací bylo provedeno zjednodušení RBAC modelu (viz 2.2.6) tak, aby byl vhodný pro malé a střední firmy. Ve výsledném návrhu modelu se objevují následující objekty:

- *Uživatelská identita* - jednoznačně identifikovatelný subjekt, tj. osoba načtená ze zdrojového systému,
- *Role* - funkce v organizaci zastupující množinu oprávnění,
- *Oprávnění* - popis interakce, kterou bude moci držitel vykonávat,
- *Skupina* - představuje množinu uživatelů v cílovém systému,
- *Účet* - uživatelský účet v cílovém systému.

Grafické znázornění modelu lze spatřit na obrázku 4.1. Zjednodušený model zajišťuje dostatečnou flexibilitu v oblasti přiřazování oprávnění, jelikož umožňuje individuální přiřazení oprávnění uživatelům a zároveň umožňuje přiřazení rolí, v rámci kterých jsou oprávnění shlukována. To odpovídá výsledkům analýzy, kdy v organizacích pracuje více osob na stejných pozicích, u kterých se předpokládá částečný překryv oprávnění. Proto lze vytvořit takové role, které budou zastupovat společnou množinu oprávnění pro stejně zaměřené osoby. Nepřekrývající se oprávnění lze následně řešit pomocí individuálního přiřazování. Oprávnění spolu s rolemi se označují jako práva. Mezi rolemi lze vytvářet hierarchickou strukturu, která modeluje vztahy nadřazených a podřazených rolí. Nadřazené role vždy přebírají oprávnění podřazených rolí. Vzhledem ke zjištění používání skupin v rámci procesu autorizace uživatelů jsou do modelu taktéž zahrnuty a shlukovány do oprávnění. Do skupin jsou následně shlukovány jednotlivé uživatelské účty. Samotné uživatelské účty a skupiny se uchovávají v cílových systémech, vůči kterým následně probíhá proces autentizace, resp. autorizace uživatelů.

Ostatní prvky byly z modelu vypuštěny kvůli jejich bezvýznamnosti pro malé a střední firmy. Například sady byly vypuštěny na základě zjištění, že oprávnění zaměstnanců na stejných pozicích se liší. Neboli v organizacích není tolik zaměstnanců s naprosto stejnými oprávněními, aby zavedení sad dávalo smysl, respektive členění oprávnění do rolí je dostačující.



Obrázek 4.1: Zjednodušený RBAC model vhodný pro malé a střední firmy.

4.2 Použité technologie

V této podkapitole jsou v krátkosti popsány softwarové technologie a nástroje, které byly použity pro návrh a implementaci MIDM. Veškeré technologie a nástroje byly vybrány na základě pozitivní zkušenosti autora práce, který s nimi přišel do styku při řešení školních a firemních projektů.

4.2.1 Java EE

Java Platform, Enterprise Edition (zkráceně Java EE) je standardizovaná platforma určená pro vývoj a provoz informačních systémů. Základem je platforma Java SE (*Java Platform, Standard Edition*), nad kterou je definována množina součástí tvořící Java EE. Samotná platforma Java EE je definována obecnou specifikací udržovanou pomocí JCP[1]. Jednotlivé součásti této platformy jsou poté definovány pomocí vlastních specifikací. S využitím API specifikovaných součástí jsou následně vyvíjeny Java EE aplikace.

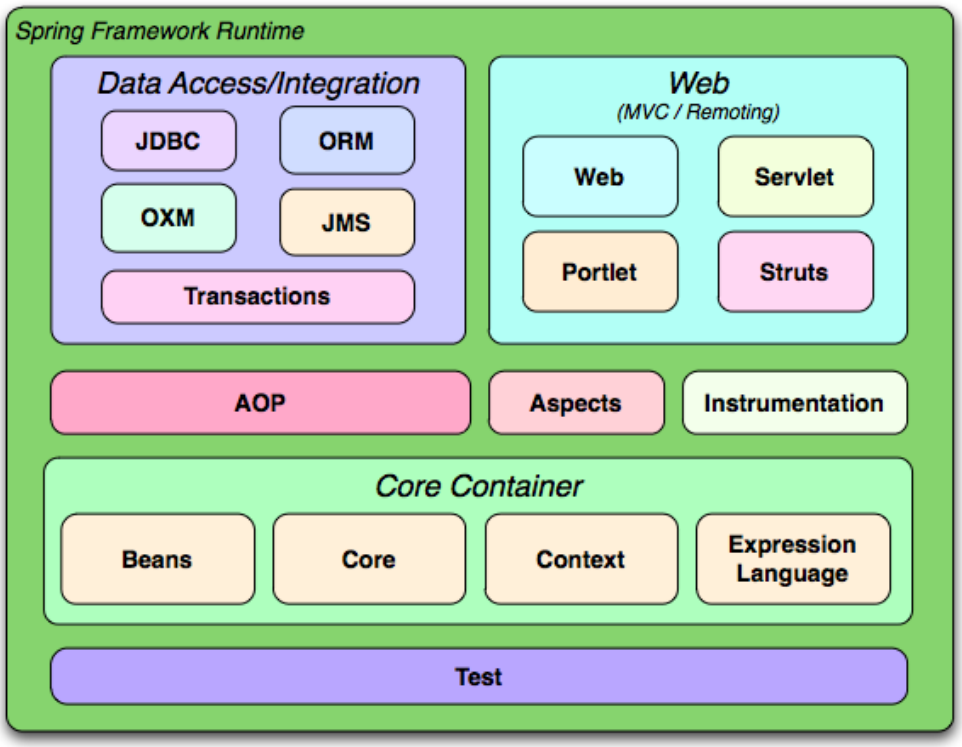
Běhovým prostředím pro Java EE aplikace je takzvaný aplikační server, který implementuje veškerá rozhraní obsažená ve specifikaci Java EE. Mezi nejznámější aplikační

[1]Zkratka pro *Java Community Process*, jedná se o mechanismus používaný k vývoji součástí platformy Java umožňující zapojení veřejnosti.

servery patří *JBoss* vyvíjený organizací Red Hat nebo *GlassFish* vyvíjený organizací Oracle, kde tento aplikační server se považuje za referenční implementaci. Dále existují částečné implementace zmíněné specifikace, jako je servlet kontejner Apache Tomcat (4.2.10).

4.2.2 Spring

Spring je velmi populární open-source (Apache licence, Version 2.0[2]) framework pro vývoj systémů založených na platformě Java. Spring se stará o kompletní infrastrukturu kolem aplikace a umožňuje vývojáři se soustředit jen a pouze na implementování logiky aplikace[17]. O neustálý vývoj se stará rozsáhlá komunita uživatelů, kde při vývoji se klade důraz na zpracování dokumentace, která je u Springu opravdu na špičkové úrovni. Spring se obecně považuje za alternativu k těžkopádným *Enterprise Java Beans* (EJB)[3]. Mezi hlavní přednosti patří modulární architektura, která umožňuje použít vždy jen a pouze tu část, která je potřebná. Seskupení jednotlivých modulů do logických celků je možné spatřit na obrázku 4.2.



Obrázek 4.2: Přehled modulů Spring frameworku. Zdroj [17].

Nejvyužívanějším logickým celkem je bezpochyby *Core Cointainer*. Jádro Spring frameworku využívá návrhový vzor *Inversion of Control* a je označováno jako IoC kontejner. Tento kontejner poskytuje jednotný přístup k vytváření a konfiguraci jednotlivých komponent systému, kde závislosti mezi komponentami jsou řešeny pomocí

[2]THE APACHE SOFTWARE FOUNDATION. Apache License, Version 2.0 [online]. 2004 [cit. 2014-07-15]. Dostupné z: http://www.apache.org/licenses/LICENSE-2.0.html

[3]Jedná se o jednu ze specifikací Java EE popisující standardizovaný způsob implementace business logiky na straně serveru.

vsazování závislostí (anglicky *Dependency Injection*). Pomocí zmíněného přístupu dochází k přesunutí zodpovědnosti za vytváření objektů[4] a dosazení nezbytných závislostí z aplikace na samotný framework. Samotné vytváření objektů je prováděno na základě konfiguračních metadat, které mohou mít formu XML souborů a nebo Java anotací.

Dalším velmi důležitým modulem je AOP (*Aspect Oriented Programming*), který umožňuje využívat technik aspektově orientovaného programování. S využitím těchto technik je možné vyseparovat opakující se fragmenty kódu do takzvaných aspektů. Následně je pomocí tzv. přípojných bodů definováno kde se mají aspekty aplikovat. Typicky se AOP používá pro řízení transakcí při vykonávání metod na servisní vrstvě v aplikaci. Před vykonáváním takových metod se pomocí AOP vytvoří nová transakce a po dokončení dojde k jejímu *commitu* nebo *rollbacku*.

Pro implementace webových aplikací je zásadní logický celek s názvem *Web*. Základním modulem tohoto celku je implementace architektonického vzoru *Model-view-controller* (MVC). Základním prvkem této implementace je *DispatcherServlet*, který je implementací návrhové vzoru *Front Controller*[5]. Tento servlet je zodpovědný za delegaci řízení zpracování HTTP požadavků na implementace rozhraní definovaných Spring frameworkem.

Spring dále poskytuje několik dalších podprojektů, které řeší konkrétní problémy při vývoji Java EE aplikací. Příkladem může být projekt *Spring Security*, který se používá pro zabezpečení aplikací a řízení přístupu k jednotlivým částem zabezpečených aplikací.

4.2.3 JavaServer Pages

JavaServer Pages (JSP)[6] je prezentační technologie, která umožňuje jednoduchým způsobem vytvářet dynamicky generované webové stránky. Z hlediska architektury lze na JSP nahlížet jako na abstrakci Java servletů, jelikož ve skutečnosti jsou jednotlivé JSP stránky za běhu transformovány na Java servlety. O transformaci se stará engine aplikačního serveru nebo servlet kontejneru, kde u Tomcatu (viz 4.2.10) se tento engine nazývá *Jasper*.

Samotný obsah webových stránek je vytvářen pomocí tagů. Mezi základní používané tagy patří tagy jazyka HTML, dále mohou být použity rozšiřující kolekce tagů. Příkladem takové kolekce může být knihovna JSTL[7], jejíž tagy poskytují efektivní způsob vložení logiky do JSP (např. iterace) bez nutnosti explicitního používání kódu jazyka Java. Pro přístup k datům uchovávaných v Java beans komponentách je používán jazyk *Unified Expression Language* (EL), který je součástí specifikace JSP verze 2.0.

4.2.4 Twitter Bootstrap

Twitter Bootstrap je volně dostupný front-endový framework poskytující množinu CSS stylů a Javascriptových komponent pro implementaci příjemného responzivního grafic-

[4]Objekty vytvářené v kontextu IoC kontejneru se nazývají *Beany* (anglicky *Beans*).

[5]ORACLE. Core J2EE Patterns - Front Controller [online]. 2002 [cit. 2014-07-15]. Dostupné z: http://www.oracle.com/technetwork/java/frontcontroller-135648.html

[6]JAVA COMMUNITY PROCESS. JSR 152: JavaServer Pages 2.0 [online]. 2006 [cit. 2014-07-14]. Dostupné z: https://jcp.org/en/jsr/detail?id=152.

[7]JAVA COMMUNITY PROCESS. JSR 52: A Standard Tag Library for JavaServer Pages [online]. 2002 [cit. 2014-07-14]. Dostupné z: https://jcp.org/en/jsr/detail?id=52.

kého uživatelského rozhraní. Nezbytným rysem je kompatibilita se všemi nejpoužívanějšími internetovými prohlížeči.

4.2.5 Font Awesome

Font Awesome poskytuje množinu vektorových ikon, které lze využít ve front-endu aplikací. Parametry ikon, jako je velikost, barva nebo rotace, mohou být triviálním způsobem upravovány pomocí kaskádových stylů. Verze *4.2.0* obsahuje přesně 479 různých ikon znázorňujících objekty z reálného světa či různorodé operace. Font Awesome je vhodným doplňkem frameworku Twitter Bootstrap a výrazným způsobem zvyšuje uživatelskou přívětivost front-endu webových aplikací.

4.2.6 Hibernate

Hibernate je framework zajišťující objektově-relační mapování (ORM) a je jednou z implementací specifikace JPA[8]. Mapováním se rozumí převod doménového modelu Java aplikace na entity v relační databázi a zpět. K popsání způsobu mapování jednotlivých doménových objektů na tabulky v relační databázi lze využít dva odlišné přístupy. Prvním přístupem je vytvoření konfiguračních souborů ve formátu XML. Druhý přístup zahrnuje speciální anotace, které jsou používány přímo v jednotlivých doménových objektech. Hibernate tedy vytváří vrstvu mezi aplikací a databází, kde tato vrstva je následně v aplikaci využita pro ukládání dat. To je výhodné, jelikož je tím odstraněna závislost Java aplikace na konkrétním databázovém systému a přechod na jiný systém spočívá v jednoduchém zásahu do konfigurace Hibernate.

4.2.7 QueryDSL

QueryDSL je framework umožňující vytváření typově bezpečných SQL dotazů. Pro použití frameworku je nejdříve nutné vygenerovat pomocné třídy z doménových objektů (vygenerované třídy mají prefix *Q*). Na základě vygenerovaných tříd jsou poté vytvářeny vlastní dotazy. Výhodou je především spravovatelnost, jelikož změny v modelových objektech způsobí kompilační chyby v souvisejících dotazech, které je následně nutné odstranit. Další výhodou oproti konstruování SQL dotazů pomocí řetězců je zabezpečené vkládání parametrů do dotazů. QueryDSL umožňuje integraci s Hibernate.

4.2.8 PostgreSQL

PostgreSQL je open-source objektově-relační databázový systém (ORDBMS). Na vývoji se podílí komunita vývojářů a firem. Mezi uživateli se jedná o velmi oblíbený databázový systém především kvůli jeho bezpečnosti a rychlosti. PostgreSQL je primárně vyvíjen pro unixové systémy, nicméně operační systém Windows je taktéž podporován. Kromě množiny standardních funkcí obsahuje PostgreSQL mnoho dalších užitečných funkcí (např. jazyk *PL/pgSQL*, pomocí kterého lze psát sofistikované procedury spouštěné na SQL serveru).

[8]JAVA COMMUNITY PROCESS. JSR 317: Java Persistence 2.0 [online]. 2009 [cit. 2014-07-14]. Dostupné z: https://jcp.org/en/jsr/detail?id=317

4.2.9 HSQLDB

HSQLDB (Hyper SQL Database) je relační databázový systém napsaný v jazyce Java. Hlavní předností je kompaktnost, rychlost, podpora více vláken, podpora transakcí a možnost persistovat data v tabulkách uložených buď v operační paměti nebo na disku. V Java aplikacích je tento databázový systém často využíván k persistování dat v operační paměti, kde tato data jsou následně zpracovávána v rámci testování pomocí frameworku JUnit.

4.2.10 Apache Tomcat

Apache Tomcat je velmi oblíbený open-source servlet kontejner implementující specifikace *Java Servlet* a *JavaServer Pages*. Tomcat je založen na jazyce Java z čehož vyplývá jeho platformní nezávislost. Mezi další výhody patří především jednoduchost a relativní nenáročnost.

4.2.11 Apache Maven

Apache Maven je softwarový nástroj používaný pro správu a řízení procesu sestavování (anglicky *building*) Java aplikací. Základním prvkem je soubor *pom.xml*, který reflektuje takzvaný *Project Object Model* a popisuje projekt jako objekt. Tento soubor má XML strukturu a jsou v něm mimo jiné definovány závislosti projektu na externích knihovnách. Definované knihovny jsou následně staženy z veřejných repozitářů do lokálního repozitáře a použity při sestavování aplikace. Tato funkcionalita patří mezi nejzásadnější, jelikož odpadá nutnost ručního stahování a importování nezbytných knihoven do projektu.

Dále jsou v *pom.xml* definovány zásuvné moduly (pluginy), které jsou následně volány příkazem ve formátu *mvn {název}:{funkce}*. Názvem se rozumí název pluginu a funkcí se rozumí specifická funkce pro daný plugin. Příkladem pluginu může být *maven-war-plugin*, který se stará o sestavení aplikace do WAR[9] archivu.

4.2.12 Activiti

Activiti je open-source workflow a BPMN platforma, která je napsána v Javě a šířena pod Apache licencí. Mezi přednosti Activiti patří rychlý BPMN 2 process engine, integrace se Spring frameworkem a dobře zdokumentované API. Activiti poskytuje několik zajímavých nástrojů, které usnadňují práci při vytváření složitých procesů a také při monitoringu běhu procesů. Kompozice těchto nástrojů spolu s BPMN 2 process enginem tvoří komponenty celé Activiti platformy. Mezi hlavní komponenty především patří:

- *Activiti Engine* - engine pro exekuci BPMN procesů,
- *Activiti Modeler* - sofistikovaná webová aplikace pro modelování BPMN procesů,
- *Activiti Designer* - plugin do vývojového prostředí Eclipse umožňující grafické modelování a testování BPMN procesů,

[9]Zkratka pro *Web Application Archive*, jedná se o archiv obsahující soubory webové aplikace, kde tento archiv je následně použit pro nasazení aplikace do aplikačního serveru či servlet kontejneru.

- *Activiti Explorer* - webová aplikace umožňující správu životního cyklu všech instancí jednotlivých BPMN procesů.

Základem celého API je třída *ProcessEngine*. Z této třídy lze získat instance servisních tříd, které obsahují veškerou logiku pro manipulaci s BPMN procesy. Mezi nejdůležitější servisní třídy patří:

- *RepositoryService* - třída umožňující správu definic BPMN procesů,
- *RuntimeService* - třída umožňující správu instancí BPMN procesů (typicky se využívá k vytváření nových instancí BPMN procesů),
- *TaskService* - třída umožňující správu úkolů, které jsou obsaženy v instancích BPMN procesů (typicky se využívá k přiřazování úkolů uživatelům nebo dokončení daných úkolů),
- *HistoryService* - třída umožňující provádění komplexních dotazů nad historickými daty shromážděnými pomocí Activiti enginu.

Pro uchovávání veškerých dat je možné použít jednu z podporovaných databází. Mezi podporované databáze mimo jiné patří PostgreSQL. Jak již bylo zmíněno, mezi hlavní přednosti Activiti patří podpora integrace se Spring frameworkem. Díky této podpoře lze veškeré potřebné třídy vytvořit jako standardní Spring beany a poté je používat napříč aplikací.

K definici BPMN procesů se používá jazyk XML. V definicích je možné použít elementy definované pomocí BPMN 2.0 a dále rozšiřující elementy specifikované pomocí namespace Activiti BPMN. Při integraci se Spring frameworkem je možné v definicích využít zaregistrované beany jako hodnoty atributů.

4.3 OpenIDM

Na základě analýzy dostupných komunitních projektů (viz 3.3) bylo v souladu se zadáním rozhodnuto, že pro samotnou synchronizaci účtů a skupin s cílovými systémy bude využito open-source řešení *OpenIDM*. Vzhledem k nutnosti zachování nezávislosti MIDM nebude komunikace probíhat prostřednictvím poskytovaného REST API, ale MIDM bude veškerá data persistovat v databázi, ke které bude taktéž přistupovat *OpenIDM*. Na základě konfigurace budou následně specifikovány konkrétní databázové tabulky, které mají být synchronizovány. Sychronizace bude automatizovaně spouštěna v pravidelných intervalech prostřednictvím softwarového démonu cron. Veškeré nepotřebné funkcionality (GUI, workflow engine...) poskytované *OpenIDM* budou odstraněny nebo deaktivovány.

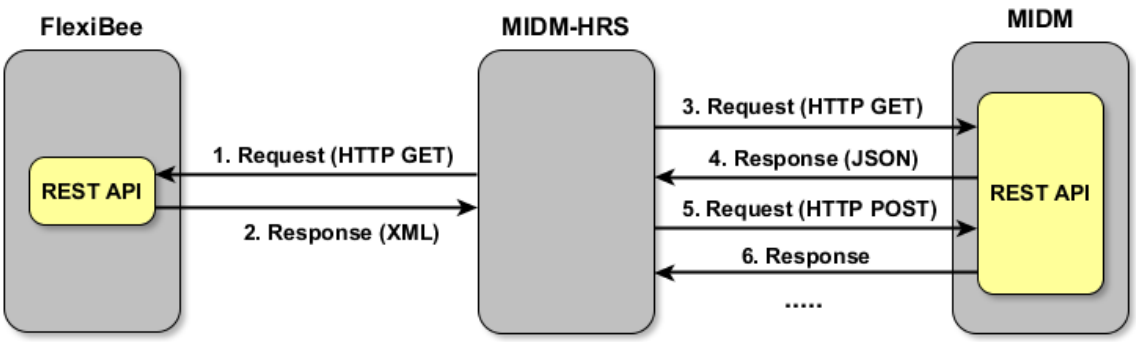
4.4 Zdrojové systémy

Jedním z definovaných funkčních požadavků bylo, že výsledné řešení musí umožnit integraci se zdrojovým systémem tak, aby mohl probíhat import uživatelských identit. V první fázi se očekává integrace s jedním systémem, nicméně v dalších fázích projektu bude docházet k rozšiřování podpory o další systémy. Vzhledem k zachování nezávislosti MIDM bylo rozhodnuto, že veškerá logika týkající se integrace se zdrojovými systémy

bude koncentrována do podprojektu s názvem *MIDM-HRS*. Architektura této aplikace je popsána v podkapitole 4.6. Komunikace bude probíhat pomocí REST webových služeb, kdy MIDM bude poskytovat patřičné REST rozhraní pro import uživatelů.

Na základě analýzy modelových organizací (viz 3.1) bylo zjištěno, že jedna z organizací využívá ekonomický systém *FlexiBee*[10]. Tento systém podporuje celou řadu evidencí, mezi které patří i evidence zaměstnanců. Vzhledem k tomu, že organizace *FlexiBee Systems* sama nazývá tento systém jako *ekonomický systém pro malé a střední firmy* bylo rozhodnuto, že v první fázi bude podporován právě tento zdrojový systém. Velikou výhodou je existence REST webových služeb, pomocí kterých lze exportovat záznamy z jednotlivých evidencí v několika podporovaných formátech (XML, JSON...).

Schéma komunikace během importu uživatelských identit je možné spatřit na obrázku 4.3. Nejprve je pomocí HTTP GET dotazu získána množina zaměstnanců evidovaných v *FlexiBee* (označeno body 1 a 2). Následně proběhne iterace přes získanou množinu, kdy nejdříve je pomocí HTTP GET dotazu ověřeno, zdali daná uživatelská identita již existuje či nikoliv (označeno body 3 a 4). Pokud daná uživatelská identita neexistuje, dojde k jejímu naimportování (označeno body 5 a 6).



Obrázek 4.3: Schéma komunikace během importu uživatelských identit.

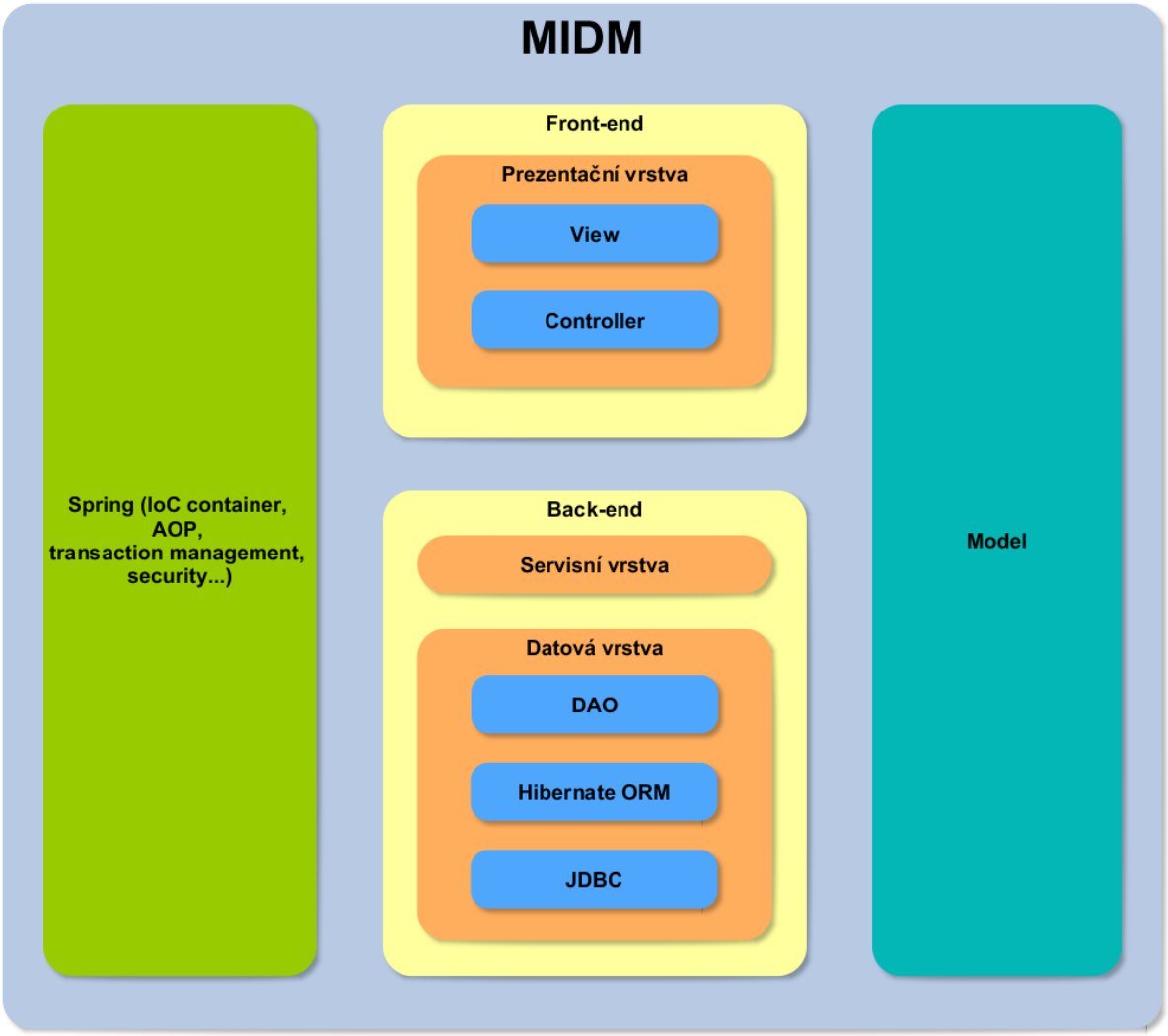
4.5 Architektura aplikace MIDM

Architektura aplikace reflektuje všechny výše zmíněné technologie. Jedná se o klasickou klient-server aplikaci. Aplikace se dělí na front-endovou část a back-endovou část. Tyto části jsou dále děleny do dalších logických celků, přičemž celková architektura aplikace je rozdělena do tří vrstvev (prezentační, servisní a datová). V prezentační vrstvě aplikace je obsažen architektonický vzor *model-view-controller* (MVC).

4.5.1 Vrstvový diagram aplikace

Vrstvový diagram je používán jako nejabstraktnější znázornění architektury aplikace. Takový diagram lze spatřit na obrázku 4.4.

[10]FlexiBee: Internetové ekonomické systémy [online]. [cit. 2014-07-17]. Dostupné z: http://www.flexibee.eu/.



Obrázek 4.4: Vrstvový diagram aplikace.

4.5.1.1 Prezentační vrstva

Jak již bylo řečeno, prezentační vrstva aplikace je tvořena pomocí architektonického vzoru *model-view-controller*, který odděluje uživatelské rozhraní aplikace, řídicí logiku a datový model do tří nezávislých komponent.

Komponenta *view* je založená na technologii JSP (viz 4.2.3). Aplikace bude obsahovat webové stránky, prostřednictvím kterých bude možné provádět CRUD operace s modelovými objekty a řídit kompletní životní cyklus jednotlivých schvalovacích workflow. Tyto stránky budou vytvářeny s využitím předdefinovaných tagů a zároveň tagů vlastních, které budou zapouzdřovat opakující se logiku.

Controllery jsou komponenty, které se starají o zpracování příchozích požadavků (anglicky *request*), připravení modelu s daty a předání modelu komponentě *view* k vykreslení. Základním prvkem u Springu je *DispatcherServlet*, který je implementací návrhové vzoru *Front Controller*. *DispatcherServlet* vždy přijme HTTP požadavek a na základě URL deleguje zpracování požadavku na příslušný controller. Controller na základě volání nižších vrstev připraví model s daty a vrátí logické jméno *view*, které se má zobrazit. Každý controller by měl zpracovávat požadavky týkající se společného

logického celku (např. pro každý modelový objekt by měl být samostatný controller).

4.5.1.2 Servisní vrstva

Servisní vrstva tvoří prostřední vrstvu mezi vrstvou prezentační a datovou. Jen a pouze v této vrstvě je zapouzdřena veškerá business logika. Z hlediska prezentační vrstvy určuje tato vrstva množinu dostupných funkčností. Za účelem manipulace s daty probíhá komunikace mezi servisní vrstvou a datovou vrstvou. Právě na této vrstvě probíhá řízení transakcí (viz 4.5.1.6) a zabezpečení servisních metod (viz 4.5.1.7).

4.5.1.3 Datová vrstva

Datová vrstva komunikuje s databází a zajišťuje ukládání dat a jejich následné vybavení. Skutečný přístup k databázovému serveru probíhá prostřednictvím JDBC a příslušného ovladače. S JDBC API následně komunikuje Hibernate, který je implementací specifikace JPA a zajišťuje objektově-relační mapování. Nad Hibernate je poté definována množina DAO[11] objektů. Každý objekt ze zmíněné množiny využívá Hibernate a poskytuje množinu konkrétních operací pro práci s daty. Je patrné, že díky tomuto přístupu je aplikace odstíněna od specifik relačních databází a veškerá manipulace s daty probíhá v duchu objektově orientovaného programování.

4.5.1.4 Model

Modelem jsou označovány objekty, které zapouzdřují persistentní data. Jsou používány ve všech vrstvách aplikace. Objekty neobsahují žádnou business logiku. Třídy, z kterých byly tyto objekty instanciovány, obsahují jen a pouze instanční proměnné a příslušné *settery* a *gettery*. Ve skutečnosti se tedy jedná o instance, které byly vyvozené z tříd reprezentujících některou z databázových tabulek.

4.5.1.5 Spring IoC kontejner

Spring IoC kontejner se stará o vytvoření a řízení životního cyklu všech nezbytných komponent systému. Dále řídí vazby zmíněných komponent pomocí vsazování závislostí (anglicky *Dependency Injection*). Z hlediska architektury tedy pracuje nad všemi vrstvami.

4.5.1.6 Řízení transakcí

Jednou z důležitých součástí Spring frameworku je komplexní podpora transakcí umožňující jejich deklarativní řízení. Toho je využíváno na servisní vrstvě, kde jsou metody označovány anotací *@Transactional*. Tato anotace způsobí před samotném vykonáním kódu dané metody vytvoření nové transakce. Při vyhození jakékoliv výjimky během vykonávání kódu metody nastane *rollback* této transakce (tzn. případná modifikovaná data zůstanou konzistentní). V opačném případě nastane *commit* transakce. Při použití anotace v deklaraci rozhraní bude tento mechanismus aplikován na každou metodu jejíž signatura je deklarována v tomto rozhraní. Pro samotné řízení transakcí se využívá technik AOP.

[11] ORACLE. Core J2EE Patterns - Data Access Object [online]. 2002 [cit. 2014-07-17]. Dostupné z: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

4.5.1.7 Zabezpečení aplikace

Pro kompletní zabezpečení aplikace je využit framework Spring Security. Uživatel se do aplikace přihlašuje pomocí uživatelského jména a hesla. Každý uživatel má přiřazenou interní roli, která je následně autorizována na několika místech v aplikaci. Zabezpečení se týká front-endové části i části back-endové. Ve front-endové části je prováděna autorizace jednotlivých HTTP požadavků a dále jsou v JSP stránkách používány speciální security tagy pro provádění autorizace role přihlášeného uživatele. V back-endové části jsou používány speciální anotace pro zajištění provádění autorizace role přihlášeného uživatele před vykonáním oanotovaného kódu.

4.5.2 Uživatelské role

V podkapitole 3.5.3 proběhla analýza uživatelských rolí systému. Pro každou ze zmíněných rolí je vytvořena interní role, která je následně přiřazována uživatelům. Každý uživatel může mít v jeden okamžik přiřazenu právě jednu interní roli. Na základě interních rolí se v aplikaci provádí autorizace operací. Mezi interní role patří:

- *ROLE_ADMIN* - interní role pro administrátora,
- *ROLE_MANAGER* - interní role pro manažera,
- *ROLE_USER* - interní role pro uživatele,
- *ROLE_SYSTEM* - interní role pro systémového uživatele.

4.6 Architektura aplikace MIDM-HRS

Jedná se o běžnou *stand-alone* Java aplikaci, která je následně interpretována prostřednictvím JVM. Aplikace využívá Spring IoC kontejner pro vytvoření všech nezbytných komponent a vyřešení jejich závislostí. Pro pravidelné spouštění úloh je využívána komponenta *Task Scheduling*[12] ze Spring frameworku. Díky této komponentě je možné využívat anotaci *@Scheduled*, kde parametrem této anotace může být *cron expression*[13]. Oanotovaná metoda je následně pravidelně spouštěna na základě specifikovaného výrazu.

Veškerá konfigurace aplikace je umístěna v externím souboru. Prostřednictvím tohoto souboru je možné řídit vykonávání jednotlivých úloh (povolovat či zakazovat), specifikovat kdy se mají vykonávat a specifikovat nezbytné parametry pro komunikaci. Aplikace nevyužívá žádné úložiště. Komunikace se systémy (zdrojové systémy, MIDM) probíhá prostřednictvím REST webových služeb.

4.7 Diagram nasazení

Diagram nasazení (anglicky *Deployment Diagram*) ilustruje rozložení jednotlivých softwarových komponent na hardwarových zdrojích a dále komunikaci mezi těmito kompo-

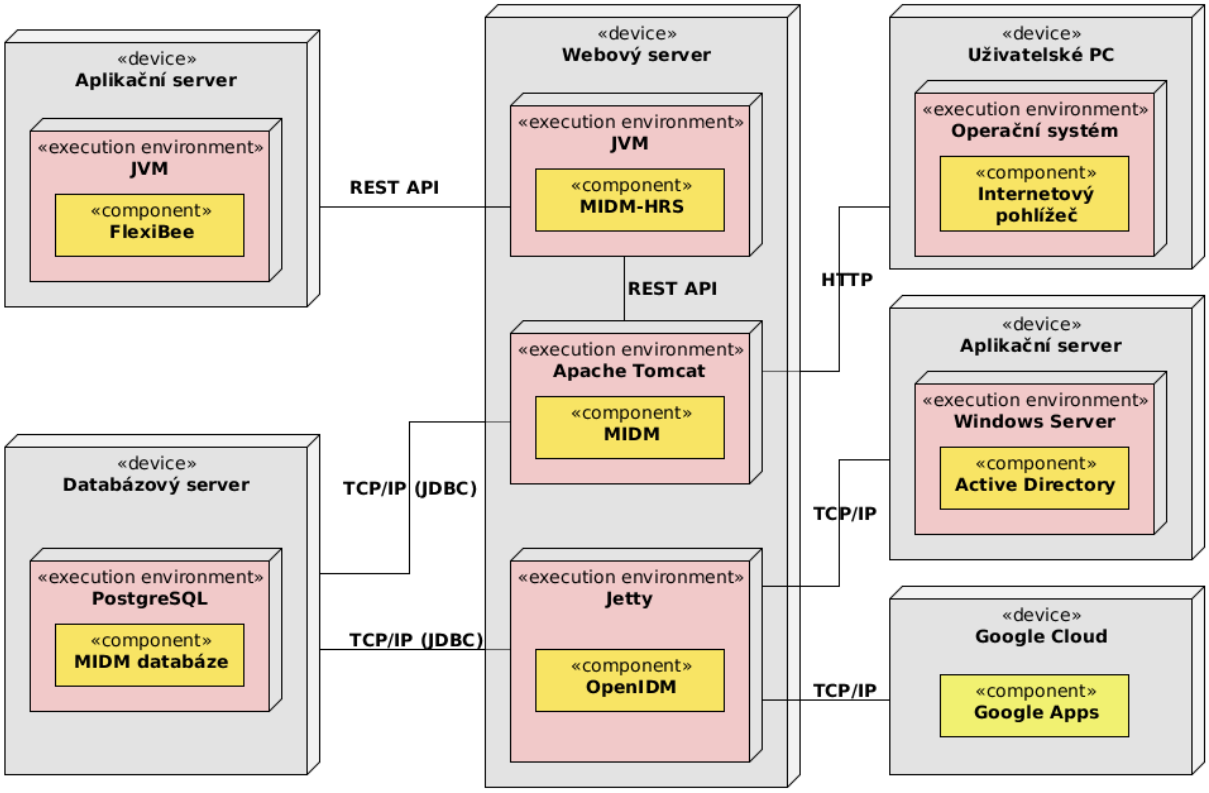
[12]SPRINGSOURCE. Task Execution and Scheduling [online]. [cit. 2014-07-18]. Dostupné z: http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/scheduling.html

[13]Jedná se o řetězec obsahující několik hodnot, kde každá hodnota hraje určitou roli při specifikaci času. Příkladem může být výraz *0 * * * * MON-FRI*, který znamená každou minutu ve všední dny.

nentami. Tento diagram se tedy používá pro specifikování fyzické architektury systému. V diagramu jsou používány tyto základní objekty:

- *Uzel* (*Device Node*) - představuje fyzické zařízení, na kterém budou spouštěny softwarové komponenty,
- *Prostředí* (*Execution Environment Node*) - představuje softwarové prostřední poskytující služby pro hostování softwarových komponent,
- *Komponenta* (*Component*) - představuje softwarovou komponentu běžící v daném prostředí.

Diagram nasazení pro tento projekt je možné spatřit na obrázku 4.5.



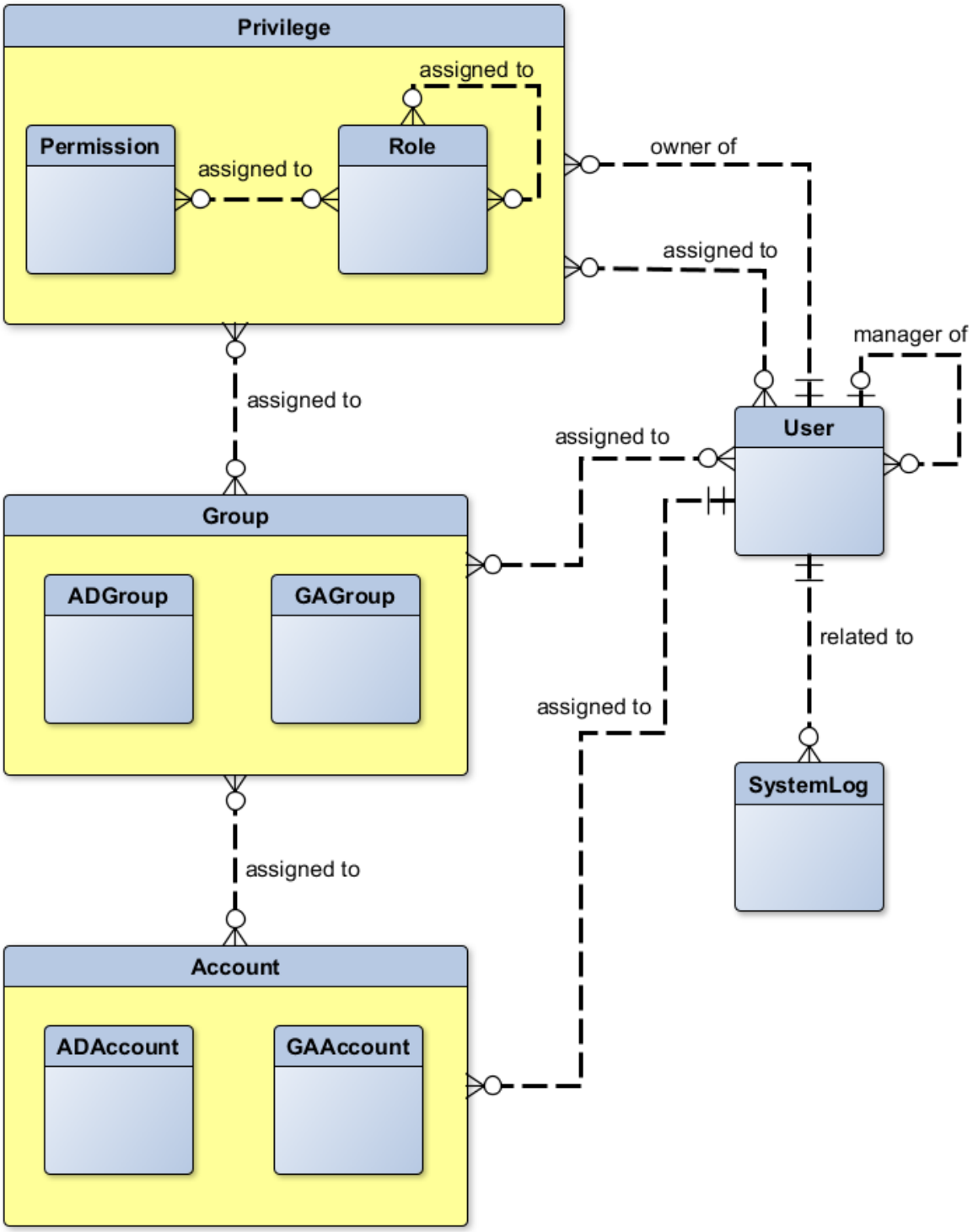
Obrázek 4.5: Diagram nasazení.

4.8 Entitně relační diagram

Entitně relační diagram (anglicky Entity–Relationship Diagram) slouží ke grafickému znázornění modelované reality pomocí entit, atributů a vztahů mezi entitami. Entitou se rozumí množina objektů z reálného světa, které mají stejné vlastnosti. Atribut popisuje jednu vlastnost entity, která může nabývat některé z hodnot definovaných pomocí tzv. domény atributu. Pro vytvoření diagramu lze použít řadu různých notací, kde každá notace definuje způsob znázornění jednotlivých komponent diagramu. Autor práce se rozhodl použít notaci *Information Engineering*[14].

[14]MARTIN, James. Information engineering: Planning and Analysis. Englewood Cliffs, N.J: Prentice-Hall, 1990. ISBN 01-346-4885-4.

Entitně relační diagram pro MIDM je možné spatřit na obrázku 4.6. Z důvodu zachování přehlednosti nejsou v tomto diagramu znázorněny atributy jednotlivých entit. Tyto atributy jsou detailně popsány v podkapitole 4.8.2. V podkapitole 4.8.3 jsou detailně popsány vztahy mezi entitami.



Obrázek 4.6: Entitně relační diagram MIDM.

4.8.1 Popis diagramu

Základní entitou je uživatel (*User*), který může mít vazbu na další uživatele, čímž je možné evidovat vztah mezi nadřízeným a podřízeným. Uživatel může mít přiřazená práva (*Privilege*). Právo je nadtypem pro roli (*Role*) a oprávnění (*Permission*). Oprávnění mohou být přiřazena v rolích. Mezi rolemi může být vytvářena hierarchická struktura. Do práv mohou být přiřazeny skupiny (*Group*). V současné době jsou podporovány dva cílové systémy (viz 1.1), proto je skupina nadtypem pro *ADGroup* (*Active Directory*) a *GAGroup* (*Google Apps*). Do skupin mohou být přiřazeny uživatelské účty (*Account*). Stejně jako u skupin i zde je účet nadtypem pro *ADAccount* (*Active Directory*) a *GAAccount* (*Google Apps*). Prostřednictvím přiřazování práv vznikají vazby mezi skupinami, účty a uživateli. Veškeré změny provedené uživatelem jsou zaznamenány v systémovém logu (*SystemLog*). Záznam v systémovém logu se vždy váže k právě jednomu uživateli, který vytvoření tohoto záznamu zapříčinil.

4.8.2 Popis entit

V této podkapitole jsou detailně popsány atributy znázorněných entit.

4.8.2.1 User

Entita uživatelské identity obsahující následující atributy:

- Křestní jméno,
- Příjmení,
- E-mail,
- Telefonní číslo,
- Osobní číslo,
- Nadřízený,
- Uživatelské jméno,
- Heslo,
- Interní uživatelská role,
- Příznak, zda byla identita smazána.

4.8.2.2 Privilege

Entita pro právo, která je nadtypem pro roli (*Role*) a oprávnění (*Permission*). Jednotlivé atributy práva jsou zděděny rolemi i oprávněními. Mezi tyto atributy patří:

- Název,
- Popis,
- Cílový systém,

- Garant,
- Klasifikace,
- Příznak, zda lze právo přiřadit uživatelům,
- Příznak, zda bylo právo smazáno.

4.8.2.3 Group

Entita pro skupinu, která je nadtypem pro skupiny specifické pro jednotlivé cílové systémy. Mezi atributy skupiny patří:

- Název,
- Cílový systém,
- Příznak popisující realizaci v cílovém systému.

Skupina specifická pro cílový systém *Active Directory* obsahuje mimo základních atributů tyto atributy:

- Distinguish Name (DN),
- AD kontejner,
- Popis,
- Typ skupiny,
- Email skupiny,
- Účty přiřazené ve skupině.

Skupina specifická pro cílový systém *Google Apps* obsahuje mimo základních atributů tyto atributy:

- Doménové jméno,
- Oprávnění,
- Popis,
- Účty přiřazené ve skupině.

4.8.2.4 Account

Entita pro účet, která je nadtypem pro účty specifické pro jednotlivé cílové systémy. Mezi atributy účtu patří:

- Uživatelské jméno,
- Cílový systém,
- Příznak popisující realizaci v cílovém systému.

Účet specifický pro cílový systém *Active Directory* obsahuje mimo základních atributů tyto atributy:

- Distinguish Name (DN),
- Popis,
- Osobní číslo uživatele,
- Křestní jméno uživatele,
- Příjmení uživatele,
- Heslo,
- Skupiny, v kterých je účet přiřazen,
- Příznak, zda je účet aktivní,
- Příznak, zda je heslo expirované,
- Příznak, zda heslo neexpiruje,
- Vlastník účtu.

Účet specifický pro cílový systém *Google Apps* obsahuje mimo základních atributů tyto atributy:

- Křestní jméno uživatele,
- Příjmení uživatele,
- Heslo,
- Příznak, zda je účet aktivní,
- Vlastník účtu.

4.8.2.5 SystemLog

Entita systémového logu obsahující následující atributy:

- Objekt, ke kterému se vztahuje log,
- Uživatel, který zapříčinil vytvoření logu,
- Datum vzniku,
- Popis změny dat,
- Provedená operace.

4.8.3 Popis vztahů mezi entitami

- *User* → *manager of* → *User* - uživatel může být nadřízeným jednoho nebo více uživatelů. Uživatel může mít právě jednoho nadřízeného.
- *User* → *owner of* → *Privilege* - uživatel může být garantem jednoho nebo více práv. Právo má právě jednoho garanta.
- *Privilege* → *assigned to* → *User* - právo může být přiřazeno jednomu nebo více uživatelům. Uživatel může mít přiřazeno jedno nebo více práv.
- *Permission* → *assigned to* → *Role* - oprávnění může být přiřazeno v jedné nebo ve více rolích. Role může mít přiřazené jedno nebo více oprávnění.
- *Role* → *assigned to* → *Role* - role může mít přiřazenu jednu nebo více nadřazených rolí. Role může být přiřazena v jedné nebo ve více podřazených rolích.
- *Group* → *assigned to* → *Privilege* - skupina může být přiřazena do jednoho nebo do více práv. V právu může být přiřazena jedna nebo více skupin.
- *Group* → *assigned to* → *User* - skupina může být přiřazena jednomu nebo více uživatelům. Uživatel může mít přiřazenu jednu nebo více skupin.
- *Account* → *assigned to* → *Group* - účet může být přiřazen v jedné nebo ve více skupinách. Ve skupině může být přiřazen jeden nebo více účtů.
- *Account* → *assigned to* → *User* - účet je přiřazen právě jednomu uživateli. Uživatel může mít přiřazen jeden nebo více účtů.
- *SystemLog* → *related to* → *User* - systémový log je vztažený k právě jednomu uživateli. K uživateli může být navázán jeden nebo více systémových logů.

Kapitola 5

Implementace

V této kapitole je popsána implementace systému pro správu identit u malých a středních firem (MIDM). Implementace reflektuje návrh provedený v předchozí kapitole. V samostatné podkapitole je popsána implementace aplikace MIDM-HRS (viz 4.6).

5.1 Konvence pro vývoj

V rámci zachování čitelnosti a srozumitelnosti kódu i pro osoby, které se přímo nepodílí na implementaci, a tedy nemají žádné konkrétní informace o dané implementaci, je nutné dodržovat definované konvence. Mezi definované konvence především patří:

- Dodržovat maximální počet znaků na řádku (rozumná hranice se pohybuje kolem 100 znaků),
- Dodržovat správné odsazení u vnořeného kódu,
- Vytvářet nezbytné komentáře u metod a tříd,
- Vytvářet *inline* komentáře u složitých implementací (implementace skládající se z několika logických celků),
- Vytvářet identifikátory metod a proměnných v duchu *self-documenting*[1],
- Psát veškerý obsah v anglickém jazyce.

Na obrázku 5.1 je možné spatřit ukázku kódu, která dodržuje definované konvence. Jedná se o rozhraní *RequestService* obsahující deklaraci dvou abstraktních metod. Obě metody jsou zdokumentovány a to včetně popisu formálních parametrů, kde je určeno, zdali je přípustná hodnota *null*. Pokud bude skutečná hodnota parametru rovna *null* i přesto, že podle komentáře není tato hodnota přípustná, dojde ke vzniku *NullPointerException*[2].

Dalším principem aplikovaným při vývoji je tzv. programování proti rozhraním (anglicky *program to an interface*). V praxi to znamená, že pro systémové komponenty existuje rozhraní (např. *RequestService*) a minimálně jedna implementace tohoto

[1] Jedná se o zásadu, na jejímž základě by mělo být z názvu patrné, jaká je funkce dané metody nebo proměnné.

[2] ORACLE CORPORATION. Class NullPointerException [online]. [cit. 2014-07-22]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/lang/NullPointerException.html

rozhraní (např. *RequestServiceImpl*). Následně se na všech místech v aplikaci, kde je potřebné využívat danou komponentu, pracuje s definovaným rozhraním. Tento přístup poskytuje velkou flexibilitu, jelikož jsou odstraněny závislosti na konkrétních implementacích a výměna implementace zpravidla spočívá pouze ve změně konfigurace kontextu aplikace.

```
16 /**
17  * Service for logic related with request workflows.
18  *
19  * @author Karel Maxa
20  */
21 @Transactional
22 public interface RequestService {
23
24     /**
25      * Create assignment request workflows.
26      * @param subjectIds Collection of subject identifiers. Never null.
27      * @param objectIds Collection of object identifiers. Never null.
28      * @param workflowDefinition Workflow definition name. Never null.
29      */
30     @Secured({ROLE_ADMIN, ROLE_USER, ROLE_MANAGER})
31     void createAssignmentRequests(Set<String> subjectIds, Set<String> objectIds,
32             String workflowDefinition);
33
34     /**
35      * Get manager identifier of the user.
36      * @param userId User to extract manager identifier from. Never null.
37      * @return Identifier of manager or <code>null</code> if no manager is defined. Can be null.
38      */
39     @Secured({ROLE_ADMIN, ROLE_USER, ROLE_MANAGER})
40     String getUserManagerId(String userId);
```

Obrázek 5.1: Ukázka kódu z rozhraní *RequestService*.

5.2 Verze závislostí

V této podkapitole jsou specifikovány verze jednotlivých softwarových technologií a nástrojů, které byly použity při vývoji. Verze použitých softwarových technologií jsou následující:

- *Java Platform* - verze 1.7.0_45,
- *Spring framework* - verze 3.2.4.RELEASE,
- *Spring security* - verze 3.2.4.RELEASE,
- *JUnit* - verze 4.11,
- *Twitter Bootstrap* - verze 3.1.1,
- *Font Awesome* - verze 4.0.3,
- *Hibernate* - verze 4.2.0.Final,
- *QueryDSL* - verze 3.3.2,
- *PostgreSQL* - verze 9.3,
- *HSQLDB* - verze 2.3.2,

- *Activiti* - verze 5.15.1.

Verze použitých nástrojů jsou následující:

- *Apache Tomcat* - verze 7.0.42,
- *Apache Maven* - verze 3.0.4,
- *OpenIDM* - verze 3.1.0-SNAPSHOT.

5.3 Vývojové prostředí

Pro provedení implementace zamýšlené aplikace bylo nezbytné zvolit vhodné vývojové prostředí. Podle názoru autora práce připadají v úvahu dvě open-source varianty, které jsou určeny pro vývoj Java aplikací. Konkrétně se jedná o vývojové prostředí *NetBeans*[3] od organizace Oracle Corporation a *Eclipse*[4] od organizace Eclipse Foundation. Vzhledem k tomu, že autor práce má několikaleté pozitivní praktické zkušenosti s programováním ve vývojovém prostředí *Eclipse*, padla volba právě na toto prostředí. Použita byla verze Kepler 4.3.2.

5.4 Správa kódu

Vzhledem k rozsáhlosti projektu je nezbytné provádět verzování čili uchovávat historii veškerých provedených změn v projektu tak, aby bylo možné se kdykoliv vrátit k některé z předchozích verzí. Přestože má autor práce praktické zkušenosti s verzovacím systémem *Apache Subversion*[5], pro verzování této práce byl využit distribuovaný systém *Git*[6]. Poskytovatelem webového repozitáře je webová služba *Bitbucket*[7], která nabízí bezplatný hosting pro projekty.

5.5 Adresářová struktura aplikace

Adresářová struktura aplikace je rozdělena do dvou hlavních částí. První částí je webový obsah, který je umístěn v adresáři */src/main/webapp/* s následující strukturou:

- **/resources/bootstrap/** - zdrojové soubory frameworku Twitter Bootstrap,
- **/resources/css/** - soubory s kaskádovými styly pro MIDM,
- **/resources/font-awesome/** - zdrojové soubory pro toolkit *Font Awesome*[8],
- **/resources/images/** - obrázky zobrazované v MIDM GUI,

[3]ORACLE CORPORATION. Netbeans [online]. 2013 [cit. 2014-07-21]. Dostupné z: https://netbeans.org/

[4]ECLIPSE FOUNDATION. Eclipse [online]. 2014 [cit. 2014-07-21]. Dostupné z: http://eclipse.org/

[5]THE APACHE SOFTWARE FOUNDATION. Apache Subversion [online]. 2011 [cit. 2014-07-24]. Dostupné z: https://subversion.apache.org/

[6]Git [online]. 2014 [cit. 2014-07-24]. Dostupné z: http://git-scm.com/

[7]ATLASSIAN. Bitbucket [online]. 2014 [cit. 2014-07-24]. Dostupné z: https://bitbucket.org/

[8]GANDY, Dave. Font Awesome: The iconic font and CSS toolkit [online]. [2014] [cit. 2014-07-25]. Dostupné z: http://fortawesome.github.io/Font-Awesome/

- **/resources/jquery/** - zdrojové soubory frameworku jQuery,
- **/resources/js/** - javascriptové soubory pro MIDM,
- **/resources/tablesorter/** - zdrojové soubory pro plugin *tablesorter*[9],
- **/WEB-INF/jsp/** - JSP soubory pro MIDM,
- **/WEB-INF/messages/** - lokalizační soubory pro MIDM,
- **/WEB-INF/tags/** - soubory s JSP tagy pro MIDM.

Druhou částí jsou třídy jazyka Java, které jsou umístěné v adresáři */src/main/java/*. Tyto třídy jsou shlukovány do tzv. balíčků (anglicky *packages*), kde každý balíček obsahuje třídy patřící do stejné kategorie nebo poskytující podobnou funkcionalitu. MIDM obsahuje následující balíčky:

- **cz.maxa.midm.config** - třídy obsahující konfiguraci celé aplikace,
- **cz.maxa.midm.core.audit** - třídy pro extrakci auditních logů,
- **cz.maxa.midm.core.audit.data** - hodnotové objekty pro auditní logy,
- **cz.maxa.midm.core.audit.listener** - listenery pro databázové události (insert, update, delete),
- **cz.maxa.midm.core.auth** - třídy sloužící pro autentizaci systémových uživatelů,
- **cz.maxa.midm.core.dao** - rozhraní DAO objektů pro jednotlivé doménové objekty,
- **cz.maxa.midm.core.dao.impl** - třídy implementující rozhraní DAO objektů,
- **cz.maxa.midm.core.model** - třídy reprezentující doménové objekty,
- **cz.maxa.midm.core.model.list** - třídy reprezentující výčtové typy používané v modelových objektech,
- **cz.maxa.midm.core.reqwf** - třídy reprezentující objekty pro schvalovací workflow,
- **cz.maxa.midm.core.reqwf.listener** - třídy reprezentující listenery používané v definicích schvalovacích workflow,
- **cz.maxa.midm.core.service** - rozhraní servisní vrstvy,
- **cz.maxa.midm.core.service.impl** - třídy implementující rozhraní servisní vrstvy,
- **cz.maxa.midm.core.service.search** - třídy zapouzdřující parametry pro vyhledávání modelových objektů,

[9]BACH, Christian. Tablesorter [online]. [2012] [cit. 2014-07-25]. Dostupné z: http://mottie.github.io/tablesorter/

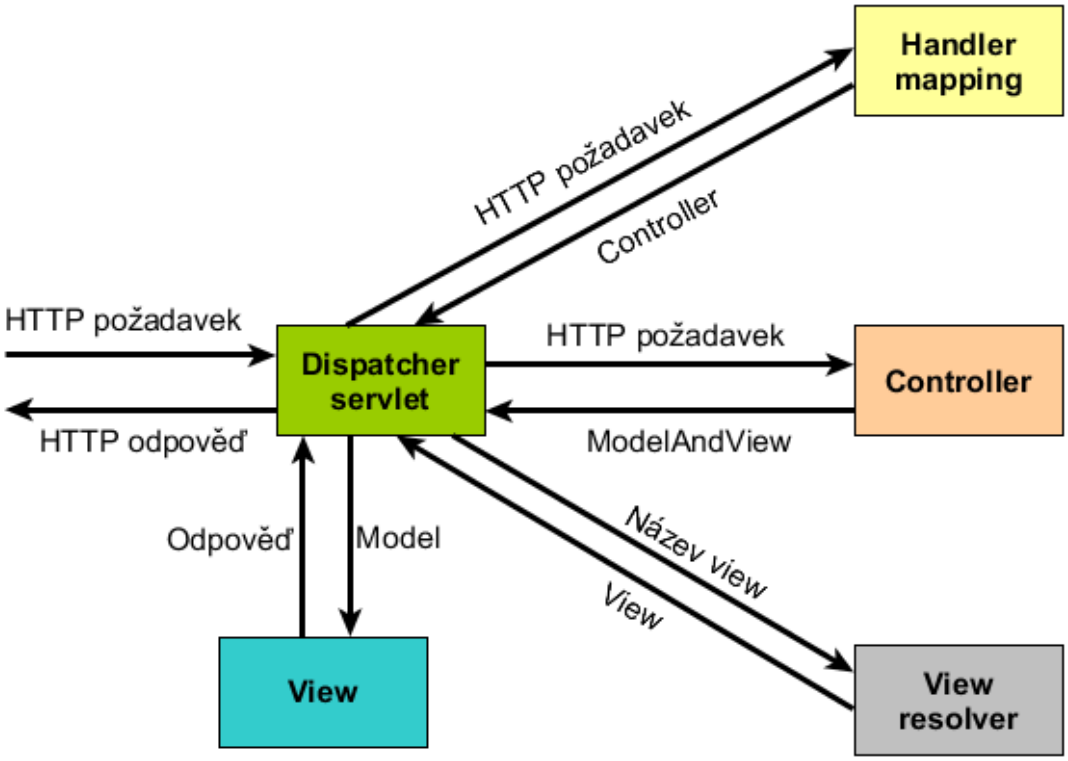
- **cz.maxa.midm.core.service.support** - pomocné třídy servisní vrstvy,
- **cz.maxa.midm.core.service.update** - update objekty pro doménové objekty,
- **cz.maxa.midm.core.util** - třídy zapouzdřující pomocné metody používané napříč aplikací,
- **cz.maxa.midm.webapp.api.rest** - třídy obsahující logiku pro REST API,
- **cz.maxa.midm.webapp.api.controller** - třídy reprezentující controllery prezentační vrstvy,
- **cz.maxa.midm.webapp.api.form** - třídy reprezentující formulářové objekty,
- **cz.maxa.midm.webapp.api.mapper** - třídy zajišťující mapování mezi formulářovými, update a doménovými objekty,
- **cz.maxa.midm.webapp.api.message** - třídy zapouzdřující logiku pro překladové hlášky.

5.6 Front-end

V této podkapitole je popsána konkrétní implementace prezentační vrstvy a grafického uživatelského rozhraní.

5.6.1 Prezentační vrstva

Prezentační vrstva aplikace je tvořena pomocí architektonického vzoru *model-view-controller* s využitím modulu Spring Web MVC. Kompletní proces zpracování HTTP požadavku je možné spatřit na obrázku 5.2. Jak již bylo řečeno, základním prvkem Spring Web MVC je třída *DispatcherServlet*, která je implementací návrhového vzoru *Front Controller*. Skrze tuto třídu procházejí veškeré HTTP požadavky a odpovědi. Při přijetí nového HTTP požadavku musí *DispatcherServlet* rozhodnout, na který controller daný požadavek deleguje. Toto rozhodování provádí některá z implementací rozhraní *HandlerMapping* na základě URL požadavku. Tato aplikace využívá třídu *SimpleUrlHandlerMapping*, která provádí rozhodování na základě anotací *@RequestMapping* (využití této anotace v controlleru je možné spatřit na obrázku 5.3) a zároveň dovoluje definovat explicitní mapování při vytváření instance této třídy (toho se využívá pro mapování webových "resources"). Po nalezení správného controlleru zajistí *DispatcherServlet* delegaci požadavku. Controller převezme žádost a zajistí exekuci správné metody (na základě URL a typu HTTP požadavku). V rámci této metody dochází ke zpracování žádosti, což zpravidla rezultuje v komunikaci s nižšími vrstvami aplikace a následném uložení dat do modelu. Jakmile je model připraven, dochází ke vrácení jména view, které se má zobrazit, zpět do *DispatcherServletu*. Následně *DispatcherServlet* přeposílá název view do některé z implementací rozhraní *ViewResolver*, která vrátí instanci daného view (JSP stránku). Tato aplikace využívá třídu *InternalResourceViewResolver*, která mapuje názvy views na příslušné URL, podle kterých jsou views nalezeny. Jakmile je vráceno konkrétní view, *DispatcherServlet* do něj vloží model a tím vznikne HTTP odpověď, která je vrácena právě tomu, kdo odeslal původní HTTP požadavek.



Obrázek 5.2: Zpracování HTTP požadavku u Spring MVC.

Na obrázku 5.3 je možné spatřit ukázku implementace controlleru *PermissionsController*. Tato ukázka obsahuje jednu metodu, která je používána pro renderování detailu oprávnění. Metoda je namapována na URL *http://midm-base-url/permissions/{permissionId}* a očekává HTTP GET požadavek. Pomocí identifikátoru *permissionId* je určeno konkrétní oprávnění, pro který se má zobrazit detail. Při exekuci metody je nejdříve nalezeno oprávnění s daným identifikátorem a následně přidáno na model. Metoda vrací název view (*/permissions/detail*). *ViewResolver* na základě tohoto názvu vrátí view *detail.jsp* ze složky *permissions*. Prostřednictvím tohoto view je zobrazen detail oprávnění, které je umístěno na modelu.

Class diagram controllerů lze nalézt v příloze E.1.

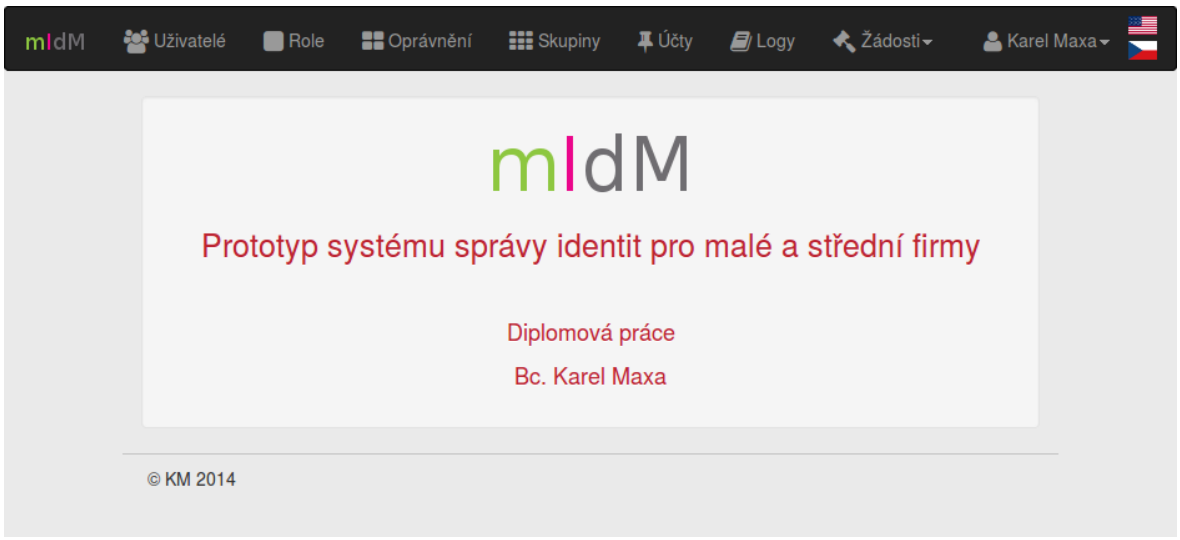
```
45 @Controller
46 @RequestMapping(value="/permissions")
47 public class PermissionsController {
48
49     @Autowired
50     private ManagementService managementService;
51
52     @RequestMapping(value="/{permissionId}", method=RequestMethod.GET)
53     public String renderDetail(@PathVariable String permissionId, Model model) {
54         Permission permission = managementService.getPermissionById(permissionId);
55         AssertUtils.assertExists("permission", permissionId, permission);
56         model.addAttribute("permission", permission);
57         return "/permissions/detail";
58     }
```

Obrázek 5.3: Ukázka implementace controlleru *PermissionsController*.

5.6.2 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní (zkráceně GUI) bylo implementováno s využitím front-endového frameworku Twitter Bootstrap. Důraz byl kladen především na jednoduchost a intuitivnost. Díky využití *grid system*[10] je celý design webového rozhraní responzivní, což znamená, že velikost jednotlivých prvků obsahu webové stránky je optimalizována pro různé druhy zařízení. Prostřednictvím GUI je možné provádět kompletní správu uživatelských identit a objektů RBAC modelu. Dále je možné vytvářet žádosti a spravovat jejich kompletní workflow. Detailní popis GUI je uveden v uživatelské příručce (viz příloha B).

Na obrázku 5.4 je znázorněna úvodní stránka aplikace MIDM, která je zobrazena po přihlášení do aplikace.



Obrázek 5.4: Úvodní stránka aplikace MIDM.

5.7 Back-end

V této podkapitole je popsána konkrétní implementace servisní a datové vrstvy.

5.7.1 Servisní vrstva

Jak již bylo řečeno v návrhové části práce, servisní třídy zapouzdřují veškerou business logiku aplikace. Tyto třídy obsahují odkazy na instance DAO objektů, jelikož implementace business logiky zpravidla provádí manipulaci s persistentními daty. Pro přenos dat doménových objektů mezi prezentační a servisní vrstvou jsou využívány tzv. update objekty. Tyto objekty neobsahují žádnou logiku a jedná se o běžné POJO objekty. V servisní vrstvě jsou namapovány na doménové objekty, které jsou předány datové vrstvě ke zpracování.

V rámci zachování konzistence dat je nutné, aby veškeré metody manipulující s daty byly prováděny v databázové transakci. O řízení transakcí se u Springu stará

[10] CSS Bootstrap: Grid system. [online]. 2011 [cit. 2014-10-18]. Dostupné z: http://getbootstrap.com/css/#grid

některá z implementací rozhraní *PlatformTransactionManager*. V této aplikaci byla využita instance třídy *HibernateTransactionManager*, která reflektuje použitý databázový systém. Pro povolení řízení transakcí je v databázovém konfiguračním souboru použita anotace *@EnableTransactionManagement*. Poté lze v aplikaci využívat anotaci *@Transactional* pro označení metod, které mají probíhat v transakcích. V této aplikaci je anotace *@Transactional* umístěna nad definici rozhraní servisní třídy, čímž je zajištěno, že veškeré definované metody v daném rozhraní budou prováděny ve vlastní databázové transakci.

Ukázku kódu rozhraní *ManagementService* je možné spatřit na obrázku 5.5. Class diagram servisní vrstvy lze nalézt v příloze E.2.

```
39 @Transactional
40 public interface ManagementService extends UserDetailsService {
41
42     /**
43      * Get all {@link User} entities from the repository.
44      * @param usc Search criteria for restrict returned results. Can be null.
45      * @return {@link List} of {@link User}s or empty list. Never null.
46      */
47     @Secured({ROLE_ADMIN, ROLE_USER, ROLE_SYSTEM, ROLE_MANAGER})
48     List<User> getUsers(UserSearchCriteria usc);
49
50     /**
51      * Create new {@link User} entity.
52      * @param user User to create. Never null.
53      */
54     @Secured({ROLE_ADMIN, ROLE_MANAGER})
55     void createUser(UserUpdate user);
```

Obrázek 5.5: Ukázku kódu rozhraní *ManagementService*.

5.7.2 Datová vrstva

V rámci podkapitoly 4.8 došlo ke znázornění modelované reality pomocí entit, atributů a vztahů. Na základě tohoto obecného modelu byly specifikovány doménové objekty[11], které jsou v aplikaci používány pro uchovávání dat. S využitím objektově-relačního mapování (anglicky *Object-Relational Mapping*) je ze specifikovaných doménových objektů vytvořen fyzický model dat. To znamená, že doménové objekty jsou převedeny na databázové tabulky, kde atributy objektů tvoří sloupce databázových tabulek a na základě vztahů mezi objekty jsou vytvářeny cizí klíče v tabulkách.

5.7.2.1 Doménové objekty

Doménové objekty jsou umístěny v balíčku *cz.maxa.midm.core.model*. V rámci každého doménového objektu jsou používány JPA anotace pro specifikování způsobu mapování na databázovou tabulku. Příklad implementace doménového objektu *Role* je možné spatřit na obrázku 5.6. Z obrázku je patrné, že třída *Role* je namapována na tabulku *midm_role* pomocí anotace *@Table*. Dále je specifikován jeden sloupec tabulky pomocí anotace *@Column*. Kompletní class diagram modelových objektů lze nalézt v příloze E.4.

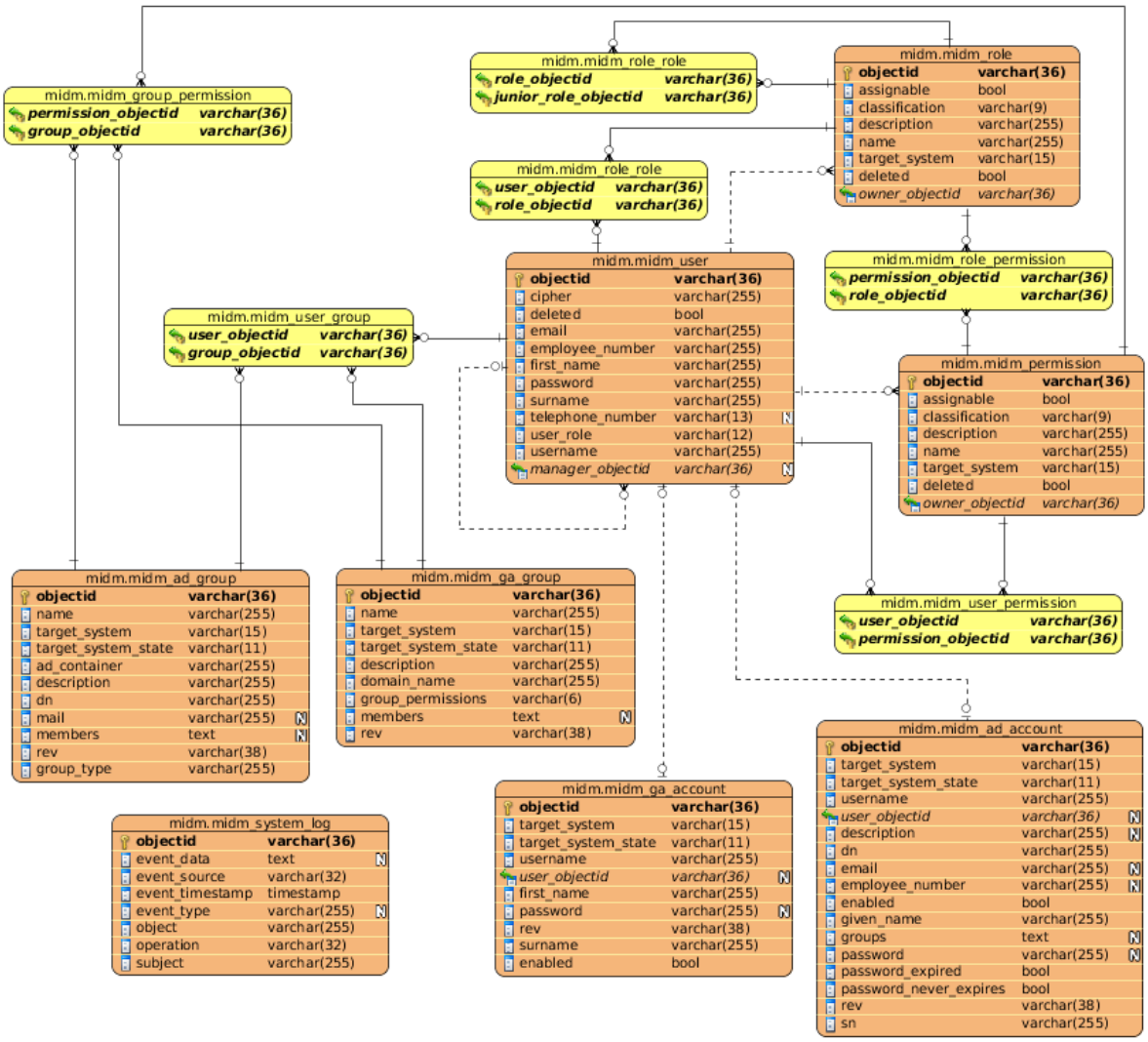
[11] Též označované jako modelové objekty.

```java
@Entity
@Table(name="midm_role")
public class Role extends Privilege {

    private boolean deleted;
    private Set<Permission> permissions;
    private Set<User> users;
    private Set<Role> seniorRoles;
    private Set<Role> juniorRoles;

    /**
     * <code>true</code> if the role is deleted.
     */
    @Column(name="deleted", nullable=false)
    public boolean isDeleted() {
        return deleted;
    }
    public void setDeleted(boolean deleted) {
        this.deleted = deleted;
    }
```

Obrázek 5.6: Ukázka kódu doménového objektu *Role*.

5.7.2.2 Fyzický model dat

Fyzický model dat byl vytvořen z definovaných doménových objektů pomocí Hibernate frameworku. V databázové konfiguraci aplikace je vytvářena beana pro třídu *LocalSessionFactoryBean*. Jedním z parametrů této třídy je pole řetězců specifikující balíčky s doménovými objekty. Veškeré objekty ze specifikovaných balíčků, které obsahují patřičné JPA anotace, budou zařazeny do objektově-relačního mapování. Fyzický model dat je možné spatřit na obrázku 5.7.



Obrázek 5.7: Fyzický model dat aplikace MIDM.

5.7.2.3 DAO objekty

DAO objekty jsou používány pro přímou manipulaci s persistentními daty. Základní třídou aplikace je abstraktní třída *AbstractHibernateDaoImpl*, která slouží k získání databázové *session*. Z této třídy dědí generická abstraktní třída *AbstractEntityDaoImpl*, která definuje metody pro modifikaci záznamů. Veškeré DAO objekty jsou poté potomkem třídy *AbstractEntityDaoImpl*. Pro konstrukci databázových dotazů je používán framework *QueryDSL*. Příklad implementace DAO objektu je možné spatřit na obrázku 5.8. V ukázce je znázorněna metoda, která je používána pro vyhledání logu na základě identifikátoru události. Je patrné, že syntaxe *QueryDSL* je velmi dobře čitelná a jednoduchá pro použití. Kompletní class diagram DAO objektů lze nalézt v příloze E.3.

```java
@Repository
public class SystemLogRecordDaoImpl extends AbstractEntityDaoImpl<SystemLogRecord>
        implements SystemLogRecordDao {

    public SystemLogRecord findById(String eventId) {
        SystemLogRecord systemLogRecord = createDslQuery().
                from(QSystemLogRecord.systemLogRecord).
                where(QSystemLogRecord.systemLogRecord.eventId.eq(eventId)).
                singleResult(QSystemLogRecord.systemLogRecord);
        return systemLogRecord;
    }
```

Obrázek 5.8: Ukázka kódu DAO objektu *SystemLogRecordDaoImpl*.

5.8 RESTová webová služba

Implementované RESTové webové služby slouží pro import uživatelských identit do systému. V současné době jsou podporovány dvě služby. První služba slouží k vrácení uživatelské identity na základě osobního čísla, druhá k vytvoření nové uživatelské identity.

Webové služby byly vytvořeny za pomoci využití prostředků Spring MVC. Strukturovaná data jsou předávána pomocí datového formátu JSON. Pro mapování mezi Java objekty a strukturovanými daty ve formátu JSON je používána knihovna *Jackson*[12].

Na obrázku 5.9 je možné spatřit konkrétní implementaci. Pomocí anotací *@RequestMapping* jsou mapovány HTTP požadavky na konkrétní controllery a metody. Stavové HTTP kódy jsou vraceny za pomoci volání výjimek, které jsou na konkrétní stavové kódy namapovány (např. *UserImportedException* je namapována na stavový kód *200 OK*). Přenášená data jsou zapouzdřena do takzvaných datových transfer objektů (DTO, anglicky *Data Transfer Object*), které obsahují jen a pouze atributy doménových objektů. Zabezpečení webových služeb je popsáno v podkapitole 5.9.

Získání informací o uživateli lze provést pomocí odeslání HTTP GET požadavku na adresu *http://midm-base-url/rest/users/123*. Obdržená odpověď ve formátu JSON může vypadat následovně:

```
{
   "firstname": "Karel",
   "surname": "Maxa",
   "email": "karel.maxa@maxa.cz",
   "employeeNumber": "123",
   "telephoneNumber": "+420123456789"
}
```

Pro vytvoření nové uživatelské identity je nutné odeslat HTTP POST požadavek na adresu *http://midm-base-url/rest/users/*. V těle požadavku jsou specifikovány atributy uživatelské identity ve formátu JSON.

[12]FASTERXML. Jackson: High-performance JSON processor [online]. [2004] [cit. 2014-07-25]. Dostupné z: http://jackson.codehaus.org/.

```java
@Controller
@RequestMapping("/rest/users")
public class UserResource extends AbstractBaseResource {

    private final Logger logger = LoggerFactory.getLogger(getClass());

    @Autowired
    private ManagementService managementService;

    @RequestMapping(value="/{employeeNumber}", method = RequestMethod.GET)
    @ResponseBody
    public UserTo getUserByEmployeeNumber(@PathVariable("employeeNumber") String employeeNumber) {
        UserSearchCriteria usc = new UserSearchCriteria();
        usc.setEmployeeNumber(employeeNumber);
        List<User> users = managementService.getUsers(usc);
        if (users.isEmpty()) {
            throw new EntityNotFoundException("User with employee number=" + employeeNumber + " does not exists.");
        }
        if (users.size() > 1) {
            throw new BadRequestException("More than one user exists with employee number=" + employeeNumber + ".");
        }
        return UsersMapper.mapToUserTo(users.iterator().next());
    }

    @RequestMapping(method=RequestMethod.POST)
    public void importUser(@RequestBody UserTo userTo) {
        // Do the validation of given user transfer object
        validateUserTo(userTo);
        // Check if exists user with given identifier
        UserSearchCriteria usc = new UserSearchCriteria();
        usc.setEmployeeNumber(userTo.getEmployeeNumber());
        if (!managementService.getUsers(usc).isEmpty()) {
            throw new BadRequestException("User with employee number " + userTo.
                    getEmployeeNumber() + " already exists.");
        }
        logger.info("Importing 'User' with params " + ReflectionToStringBuilder.toString(userTo));
        try {
            // Do the user import
            managementService.createImportedUser(UsersMapper.mapToUserUpdate(userTo));
        } catch (Exception e) {
            logger.warn("User import failed.", e);
            throw new UserImportFailedException("User import failed.");
        } finally {
            logger.warn("'User' was successfully imported.");
        }
        // User was successfully imported
        throw new UserImportedException();
    }
```

Obrázek 5.9: Implementace RESTových webových služeb.

5.9 Zabezpečení aplikace

Jak již bylo řečeno v návrhové části práce, pro zabezpečení celé aplikace byl využit framework Spring Security. Zabezpečení se týká jak front-endové části, tak části back-endové. Uživatel se do aplikace přihlašuje pomocí formulářové autentizace. Ve front-endové části jsou v jednotlivých JSP využívány security tagy, které kontrolují roli přihlášeného uživatele. To znamená, že určité části webových stránek se zobrazují podmíněně. Zároveň se role přihlášeného uživatele autorizuje při HTTP žádostech, jelikož každá URL má v konfiguraci zabezpečení specifikované uživatelské role, kterým je přístup povolen. V back-endové části jsou využívány *@Secured* anotace, které jsou umisťovány nad deklarace servisních metod (viz ukázka kódu rozhraní *ManagementService* 5.5). V anotacích jsou specifikovány uživatelské role, které mohou vykonávat danou metodu. Pro kontrolu uživatelských rolí před vykonáním zabezpečených metod jsou využívány techniky AOP. Pokud uživatelská role není vyhovující, dochází vyhození výjimky *AccessDeniedException*.

Zabezpečeny jsou taktéž REST webové služby, kde je zavedena HTTP basic autentizace[13]. To znamená, že je v rámci každého HTTP požadavku přenášeno uživatelské

[13]NETWORK WORKING GROUP. HTTP Authentication: Basic and Digest Access Authentication [online]. 1999 [cit. 2014-07-18]. Dostupné z: http://tools.ietf.org/html/rfc2617.

jméno a heslo zakódované pomocí *Base64*[14]. Tyto údaje se následně ověřují na straně serveru, kde je definovaný systémový uživatel. Pod účtem systémového uživatele jsou následně prováděny operace vyvolané prostřednictvím zmíněných webových služeb.

5.10 Implementace MIDM-HRS

Jedná se o běžnou *stand-alone* Java aplikaci, kde pro řízení životního cyklu objektů byl použit Spring IoC kontejner. Pro inicializování aplikačního kontextu je používána třída *AnnotationConfigApplicationContext*, které jsou předány konfigurační Java třídy.

Pro zasílání HTTP požadavků je používána třída *CloseableHttpClient*. Vzhledem k tomu, že *FlexiBee* i MIDM vyžadují HTTP Basic autentizaci, je nezbytné přidat do každého HTTP požadavku příslušnou hlavičku. Hlavička je vytvořena pomocí následujícího kódu:

```
new BasicScheme().authenticate(new UsernamePasswordCredentials(
        username, password), request, null);
```

Předchozí kód obsahuje následující argumenty:

- *username* - uživatelské jméno,
- *password* - heslo,
- *request* - instance HTTP požadavku (*HttpGet* nebo *HttpPost*).

Vzhledem k tomu, že *FlexiBee* REST API defaultně využívá HTTPs spolu s využitím *self-signed* certifikátu, bylo nezbytné *CloseableHttpClient* nakonfigurovat tak, aby bylo při komunikaci důvěřováno *self-signed* certifikátům pro danou doménu. K tomuto účelu byl použit následující kód:

```
SSLContextBuilder builder = new SSLContextBuilder();
builder.loadTrustMaterial(null, new TrustSelfSignedStrategy());
SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(
        builder.build(), SSLConnectionSocketFactory.
        ALLOW_ALL_HOSTNAME_VERIFIER);
```

V rámci předchozího kódu dochází k vytvoření instance *SSLConnectionSocketFactory* s takovými parametry, které zajistí důvěřování *self-signed* certifikátům pro všechny domény. Tato instance je poté předána samotné instanci HTTP klienta.

Samotný import uživatelských identit probíhá v pravidelných intervalech na základě *cron* výrazu, který je definován v externím *properties* souboru. Během importu je vždy pomocí HTTP GET získána definice osoby ze zdrojového systému *FlexiBee* ve formátu XML. Získané XML je zpracováno pomocí *DOM*[15] parseru a poté je pomocí HTTP GET ověřeno, zdali daná osoba v MIDM již existuje či nikoliv. Pokud neexistuje, je pro danou osobu vytvořen *transfer object*, který je následně pomocí HTTP POST předán RESTové webové službě MIDM k dalšímu zpracování.

[14] JOSEFSSON, Simon. Base-N Encodings: RFC 4648 [online]. 2006 [cit. 2014-07-27]. Dostupné z: http://tools.ietf.org/html/rfc4648.

[15] Document Object Model. W3C. [online]. 2009 [cit. 2014-08-16]. Dostupné z: http://www.w3.org/DOM/.

5.11 Cílové systémy

5.11.1 Active Directory

Prvním z využívaných cílových systémů je *Active Directory*. Při implementaci MIDM byl použit operační systém Microsoft Windows Server 2008 R2 Standard. V systému byly aktivovány dvě role. Konkrétně se jedná o službu *Active Directory Domain Services* a server DNS[16]. V rámci *Active Directory* byla vytvořena právě jedna doména. Název domény je následně nastaven v konfiguraci MIDM a synchronizace skupin a účtů probíhá vůči této doméně.

Vzhledem k tomu, že *OpenICF* konektor pro *Active Directory* je napsán v jazyce C# a pro komunikaci používá ADSI[17], je nezbytné aby byl nainstalován OpenICF .NET Connector Server, který slouží jako zprostředkovatel komunikace mezi konektorem napsaným v jazyce C# a *OpenIDM* napsaným v jazyce Java.

5.11.2 Google Apps

Druhým z využívaných cílových systémů je cloudová služba *Google Apps*, kde tato služba existuje v několika verzích. Při implementaci MIDM byla využita verze *Google Apps pro firmy*. Při zakládání služby je nutné specifikovat doménové jméno organizace, pro kterou je služba zřizována a vytvořit administrátorský účet. Doménové jméno a administrátorský účet je následně využíván *OpenICF Google Apps* konektorem pro synchronizaci účtů a skupin.

5.12 OpenIDM

Jak již bylo řečeno v návrhové části práce, pro synchronizaci skupin a účtů s cílovými systémy je využíván open-source systém *OpenIDM*. Konfigurace je prováděna pomocí následujících JSON souborů:

- *Managed.json* - definice synchronizovaných objektů (skupiny a účty pro každý cílový systém),
- *Repo.jdbc.json* - definice databázového připojení a specifikace mapování objektů na databázové tabulky,
- *Sync.json* - specifikace mapování atributů mezi zdrojovými a cílovými objekty,
- *Provisioner.openicf-ad.json* - definice připojení k cílovému systému *Active Directory* a specifikace atributů skupin a účtů,
- *Provisioner.openicf-ga.json* - definice připojení k cílovému systému *Google Apps* a specifikace atributů skupin a účtů,
- *Schedule-xxxReconcile.json* - definice cron výrazů pro provádění pravidelných synchronizací jednotlivých objektů (skupiny a účty pro každý cílový systém).

[16]NETWORK WORKING GROUP. DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. [online]. 1987 [cit. 2014-08-16]. Dostupné z: https://www.ietf.org/rfc/rfc1035.txt.

[17]ADSI je zkratka pro *Active Directory Service Interfaces*. Jedná se o sadu COM rozhraní používaných pro přístup k funkcím adresářových služeb.

OpenIDM přistupuje přímo k databázi *PostgreSQL*. V této databázi jsou uchovávána nezbytná data pro korektní běh *OpenIDM* a zároveň jsou zde uchovávány záznamy s definicemi skupin a účtů, které jsou synchronizovány s cílovými systémy. Pro samotnou komunikaci je používán *PostgreSQL JDBC driver* a konfigurace databázového připojení v souboru *repo.jdbc.json* je následující:

```
"connection" : {
    "dbType" : "POSTGRESQL",
    "driverClass" : "org.postgresql.Driver",
    "jdbcUrl" : "jdbc:postgresql://localhost:5432/midm",
    "username" : "username",
    "password" : "password",
    "defaultCatalog" : "midm",
    "maxBatchSize" : 100,
    "maxTxRetry" : 5
}
```

Ve stejném souboru zároveň dochází ke specifikování mapování mezi objekty a databázovými tabulkami. Například mapování skupiny pro cílový systém *Active Directory* je zajištěno následujícím kódem:

```
"managed/AdGroup" : {
    "table" : "midm_ad_group",
    "objectToColumn" : {
        "_id" : "objectid",
        "_rev" : "rev",
        "description" : "description",
        "member" : "members",
        "groupType" : "group_type",
        "mail" : "mail",
        "ad_container" : "ad_container",
        "__NAME__" : "dn",
        "cn" : "name",
        "targetSystemState" : "target_system_state"
    }
}
```

V souboru *sync.json* je definováno mapování atributů mezi zdrojovými a cílovými objekty. Zároveň jsou zde definovány takzvané *script hooks*, neboli skripty napsané v jazyce Javascript, které mají být vykonány při určitých událostech týkajících se životního cyklu objektů (například událost vytvoření objektu nebo změny objektu). Příklad mapování účtu systému *Active Directory* na objekt, který je následně uložen do příslušné databázové tabulky je následující:

```
{
    "name" : "systemADAccounts_managedADAccounts",
    "source" : "system/ad/account",
    "target" : "managed/AdAccount",
    "properties" : [
        {
```

```
            "source" : "enabled",
            "target" : "enabled"
        },
        {
            "source" : "dn",
            "target" : "dn"
        },
        ...
        {
            "source" : "groups",
            "target" : "groups"
        }
    ],
    "onCreate" : {
        "type" : "text/javascript",
        "source" : "target.targetSystemState='CREATED';"
    },
    "onUpdate" : {
        "type" : "text/javascript",
        "source" : "target.targetSystemState='CREATED';"
    }
}
```

Z předchozí ukázky kódu jsou jasně patrné zmíněné *script hooks*, kde v tomto případě je při události vytvoření nebo modifikování účtu nastaven atribut určující stav v cílovém systému na hodnotu *vytvořeno*.

Synchronizace probíhá v pravidelných intervalech, čehož je dosahováno díky cronu. Na následující ukázce je možné spatřit konfiguraci pravidelného spouštění jednosměrné synchronizace účtů systému *Active Directory* a databáze. Synchronizace je spouštěna každých 10 minut v přesně definovaných časech.

```
{
    "enabled" : true,
    "type" : "cron",
    "schedule" : "0 0,10,20,30,40,50 * * * ?",
    "concurrentExecution" : false,
    "invokeService" : "sync",
    "invokeContext" : {
        "action" : "reconcile",
        "mapping" : "systemADAccounts_managedADAccounts"
    }
}
```

Samotná synchronizace je konfigurována jako *rekonciliace*[18]. Při provádění *rekonciliace* dochází k důkladné analýze zdrojových a cílových systémů za účelem odhalení nesrovnalostí mezi spárovanými objekty, které je následně nezbytné synchronizovat.

[18]OpenIDM Integrator's Guide: Configuring Synchronization. FORGEROCK AS. [online]. 2014 [cit. 2014-10-26]. Dostupné z: http://docs.forgerock.org/en/openidm/3.0.0/integrators-guide/index/chap-synchronization.html#sync-types.

Tento proces je tedy poměrně náročný a může být náročnější, než přístup, kdy dochází jen a pouze k synchronizaci provedených změn, které jsou ukládány do logu (tento proces se nazývá *LiveSync*). Na druhou stranu je *rekonciliace* robustní a může odhalit chybové stavy, které by v rámci *liveSync* odhaleny nebyly.

Synchronizace je nakonfigurována jako obousměrná. Nejdříve je zpracován směr, kdy se synchronizují objekty z interní databáze do cílového systému. Až poté je zpracován směr opačný. To v podstatě znamená, že stav v cílovém systému bude vždy odpovídat stavu IDMS, kdy případné změny synchronizovaných objektů v cílovém systému budou srovnány podle IDMS úložiště.

Alternativním přístupem by bylo povolení automatické synchronizace, která proběhne vždy při změně nějaké hodnoty objektu v cílovém systému nebo úložišti IDMS. Při takovém přístupu nelze spoléhat na to, že skutečný stav bude odpovídat stavu v IDMS, jelikož změny provedené v cílovém systému budou zapracovány do úložiště IDMS.

5.13 Přiřazování vlastníků orphan účtům

V rámci procesu *rekonciliace* dochází k přidělování vlastníků účtům v cílových systémech na základě logických podmínek jako je korelace uživatelského jména nebo emailu. Vzhledem k tomu, že se jedná o poměrně náročný proces, byla tato funkcionalita zapouzdřena v rámci aplikace MIDM do vlastního jobu s názvem *AccountOwnerUpdateJob*. Tento job je spouštěn v pravidelných intervalech a není tedy přímou součástí *OpenIDM* a související synchronizace.

V samotném jobu jsou nejdříve nalezeny veškeré účty bez vlastníka (tzv. *orphan* účty). Následně probíhá iterace přes získané účty, kde pro každý účet je provedeno hledání vlastníka. Proces hledání je rozdílný pro každý cílový systém. Hledání probíhá na základě následujících atributů a jejich kombinací:

- Cílový systém *Active Directory*:
  1. Uživatelské jméno,
  2. Email,
  3. Kombinace jméno a příjmení.
- Cílový systém *Google Apps*:
  1. Uživatelské jméno,
  2. Kombinace jméno a příjmení.

Vlastník je účtu přiřazen právě tehdy, když je na základě daných atributů nalezena právě jedna uživatelská identita.

5.14 Schvalovací workflow

Jedním z požadavků na aplikaci byla podpora pro budoucí zavedení schvalovacích workflow (viz cíle práce 1.1). Autor práce se rozhodl schvalovací workflow implementovat

již v první fázi projektu, jelikož se podle jeho názoru jedná o zcela zásadní funkcionalitu. Pro implementaci byla využita open-source workflow a BPMN platforma Activiti, která byla popsána v návrhové části práce.

Konfigurace Activiti je velice jednoduchá, jelikož framework obsahuje přímou podporu pro Spring framework. Konfigurace je prováděna s využitím třídy *SpringProcessEngineConfiguration*, které je předáno připojení k databázi a pole obsahující cesty k definicím jednotlivých workflow. Dále jsou vytvořeny Spring beany pro nezbytné servisní třídy.

Jednotlivá schvalovací workflow se skládají z uživatelských schvalovacích aktivit, které následují sekvenčně za sebou. Jakmile dojde ke schválení všech dílčích aktivit, celá žádost je označena jako schválená. Při zamítnutí první aktivity dojde k zamítnutí celé žádosti. Na událost vytvoření schvalovací aktivity jsou vždy připojeny dva listenery. První s názvem *ApproverNotDefinedListener* zajistí automatické schválení aktivity za předpokladu, že schvalovatel není definován. Druhý s názvem *RequestInitiatorApproverListener* zajistí automatické schválení aktivity za předpokladu, že schvalovatel je stejný jako iniciátor žádosti.

Aplikace v současné době obsahuje tyto schvalovací workflow:

- Přiřazení role uživateli
  - Schválení nadřízeným uživatele,
  - Schválení garantem role.
- Odebrání role uživateli
  - Schválení nadřízeným uživatele,
  - Schválení garantem role.
- Přiřazení oprávnění uživateli
  - Schválení nadřízeným uživatele,
  - Schválení garantem oprávnění.
- Odebrání oprávnění uživateli
  - Schválení nadřízeným uživatele,
  - Schválení garantem oprávnění.
- Přiřazení role do role
  - Schválení garantem první role,
  - Schválení garantem druhé role.
- Odebrání role z role
  - Schválení garantem první role,
  - Schválení garantem druhé role.
- Přiřazení oprávnění do role
  - Schválení garantem role,

  - Schválení garantem oprávnění.
- Odebrání oprávnění z role
  - Schválení garantem role,
  - Schválení garantem oprávnění.
- Přiřazení skupiny do oprávnění
  - Schválení garantem oprávnění
- Odebrání skupiny z oprávnění
  - Schválení garantem oprávnění

5.15 Vztahy mezi objekty RBAC modelu

Vztahy mezi doménovými objekty RBAC modelu vznikají či zanikají právě tehdy, když dojde ke schválení souvisejících žádostí. V rámci každé žádosti je vždy určen *subject*, kterému bude přiřazen nebo odebrán *object*. Přiřazení či odebrání vždy probíhá v rámci celé hierarchie RBAC modelu. To znamená, že pokud například uživateli přiřadíme roli, budou mu zároveň přiřazena oprávnění, která obsahuje daná role. Dále budou uživateli přiřazeny skupiny, které jsou přiřazené v daných oprávnění a pokud v cílovém systému přiřazované role neexistuje pro daného uživatele uživatelský účet, bude vytvořen. Zmíněnému účtu budou přiřazeny skupiny, které jsou přiřazeny do oprávnění dané role. Ve stejném duchu probíhá proces opačný a tedy například odebrání role uživateli. Pokud jsou uživatelskému účtu odebrány veškeré skupiny, dojde k jeho zneaktivnění. Při opětovném přiřazení nejméně jedné skupiny je uživatelský účet aktivován.

Vytváření vazeb mezi rolemi taktéž prochází schvalovacím procesem. Každá role má vždy přiřazená veškerá oprávnění podřazených rolí. Z každé role je možné odebrat jen a pouze přímo přiřazená oprávnění, oprávnění pocházející z podřazených rolí odebírat nelze.

Modifikaci vztahů je možné provádět vždy jen a pouze mezi objekty, které sdílí totožný cílový systém. To znamená, že například není možné přiřadit oprávnění cílového systému *Active Directory* do role cílového systému *Google Apps*.

5.16 Auditní logy

Auditní logy jsou vytvářeny pomocí Hibernate listenerů, které jsou zaregistrovány na události vznikající při manipulaci s doménovými objekty. Konkrétně se jedná o operace *Insert*, *Update* a *Delete*. Listener vždy implementuje příslušné rozhraní (například *PostInsertEventListener* pro operaci *Insert*), což znamená, že tělo implementované metody bude vykonáno po provedení dané operace. V rámci implementace listenerů dochází k extrahování provedených změn z objektu provedené operace. Extrahované změny jsou serializovány do formátu JSON a následně spolu s hodnotami dalších atributů uloženy do databáze. Dalšími atributy se rozumí především identifikátor uživatele, který změnu provedl, dále časová značka, identifikátor auditního logu a typ provedené operace. Při zobrazování auditních logů v aplikaci probíhá u změn proces opačný a tedy deserializace změn v JSON formátu na kolekci modifikací. Každý objekt modifikace obsahuje název atributu, jehož hodnota byla změněna a dále původní a novou hodnotu.

5.17 Lokalizace aplikace

Veškeré zobrazované hlášky v grafickém rozhraní aplikace jsou specifikovány v samostatných *properties* souborech. Každý záznam v daném souboru je tvořen klíčem a samotnou hláškou. Aplikace v současné době podporuje dvě jazykové mutace (čeština a angličtina), což rezultuje v existenci dvou *properties* souborů s překladovými hláškami. Tak aby mohly být definované hlášky zobrazovány, je nutné vytvořit spring beanu pro jejich *resolving*. K tomuto účelu je vytvářena instance třídy *ReloadableResourceBundleMessageSource*, kde hlavním parametrem je cesta k souborům s překladovými hláškami. Soubor se správnou jazykovou mutací je vybrán na základě aktivního *locale*. Defaultně je aktivní české *locale*. Aktivní *locale* je možné měnit za běhu aplikace pomocí ikon v menu aplikace. Ve skutečnosti je po kliknutí na ikonu odeslán HTTP POST požadavek na příslušnou URL, kde v těle požadavku je parametr určující *locale*, které má být nastaveno jako aktivní. Pro určení aktivního *locale* je nezbytné vytvoření instance třídy *SessionLocaleResolver*, kde je zároveň definováno defaultní české *locale*.

5.18 Úprava OpenIDM komponenty

Při implementaci synchronizace mezi OpenIDM a databází *PostgreSQL* vyvstal problém, který spočíval v tom, že k dnešnímu dni 7. 9. 2014 OpenIDM podporuje ukládání hodnot jen a pouze do sloupců řetězcového datového typu [19]. Tato situace představovala problém kvůli tomu, že projekt využívá několik sloupců typu *boolean*. Prvním z možných řešení bylo změnit datové typy zmíněných sloupců na řetězcové a upravit související kód přímo v aplikaci MIDM. Druhé z možných řešení bylo založeno na předpokladu, že zdrojový kód OpenIDM je dostupný a mohla by být upravena příslušná logika, která zajišťuje ukládání hodnot do databázových sloupců a aplikace MIDM by mohla zůstat nezměněna. Vzhledem k tomu, že se autor práce snaží dodržovat *best practices*, které první varianta porušuje, rozhodl se tento problém vyřešit pomocí složitější druhé varianty.

Po delší analýze kódu bylo zjištěno, že třída odpovědná za vytváření a exekuci SQL příkazů je třída *MappedTableHandler* nacházející se v podprojektu *openidm-repo-jdbc*. V rámci metody *populatePrepStatementColumns(PreparedStatement prepStatement, JsonValue objVal, List<JsonPointer> tokenPointers)* jsou předávány hodnoty jednotlivých sloupců do SQL příkazu, uchovávaném v instanci třídy *PreparedStatement*. Níže je možné spatřit původní ukázku kódu, který se stará o předání hodnot do SQL příkazu:

```
int colPos = 1;
for (JsonPointer propPointer : tokenPointers) {
    // TODO: support explicit column types/conversion specified in
    // column mapping
    // This is currently limited to STRING handling
    JsonValue rawValue = objVal.get(propPointer);
    if (null == rawValue) {
        prepStatement.setString(colPos, null);
    } else if (rawValue.isString() || rawValue.isNull()) {
```

[19] PostgreSQL Documentation: Character Types. POSTGRESQL. [online]. [cit. 2014-09-07]. Dostupné z: http://www.postgresql.org/docs/9.3/static/datatype-character.html

```
            prepStatement.setString(colPos, rawValue.asString());
        } else {
            prepStatement.setString(colPos, mapper.writeValueAsString(
                    rawValue.getObject()));
        }
        colPos++;
    }
```

Z výše uvedené ukázky je jasně patrné, že ačkoliv je hodnota jakéhokoliv datového typu, je vždy přetypována na řetězec. Z uvedeného komentáře lze vyčíst, že se jedná pouze o dočasný stav a je zde předpoklad, že kód bude v budoucnosti přepsán, kdy bude zavedena podpora pro explicitní specifikování datového typu v konfiguračních souborech. Výše uvedený kód byl autorem práce upraven tak, aby podporoval ukládání hodnot do databázových sloupců, které mají i jiný než řetězcový datový typ. Upravený kód má následující podobu:

```
int colPos = 1;
for (JsonPointer propPointer : tokenPointers) {
    JsonValue rawValue = objVal.get(propPointer);
    if (rawValue == null) {
        prepStatement.setString(colPos, null);
    } else if (rawValue.isString() || rawValue.isNull()) {
        prepStatement.setString(colPos, rawValue.asString());
    } else if (rawValue.isBoolean()) {
        prepStatement.setBoolean(colPos, rawValue.asBoolean());
    } else if (rawValue.isNumber()) {
        prepStatement.setInt(colPos, rawValue.asNumber().intValue());
    } else {
        prepStatement.setString(colPos, mapper.writeValueAsString(
                rawValue.getObject()));
    }
    colPos++;
}
```

Je patrné, že byla přidána podpora pro hodnoty typu *boolean* a čísla. Vzhledem k tomu, že třída *JsonValue* obsahuje logiku pro zjištění datového typu dané hodnoty a pro vrácení hodnoty s daným datovým typem, bylo zavedení podpory velice jednoduché. Je potřeba říci, že autor práce se pozastavuje nad tím, že podpora pro jiné než řetězcové datové typy nebyla implementována *out-of-the-box*, vzhledem k tomu, že zavedení podpory není nikterak složité.

Po provedení úpravy byl podprojekt *openidm-repo-jdbc* zkompilován pomocí nástroje *Maven*, čímž vznikl soubor *openidm-repo-jdbc-3.1.0-SNAPSHOT.jar*. Tento soubor byl následně nakopírován do složky *%OpenIDM_HOME%/bundle/*, kde nahradil původní soubor.

Kapitola 6

Testování

Testování je velmi důležitou součástí procesu vývoje software. Testování lze rozdělit do několika úrovní z hlediska rozsáhlosti testované části. Testovací úrovně použité v rámci této práce lze rozdělit do jednotkových (anglicky *unit*) testů a systémových testů.

6.1 Unit testy

V rámci jednotkových testů jsou testovány funkčnosti nejmenších částí (jednotek) aplikace. Zpravidla se jedná o třídy a jejich metody. Tyto testy jsou implementovány programátorem během vývoje funkcionalit aplikace a jsou plně automatizované. Jednotkové testy jsou mimo jiné spouštěny během sestavování aplikace pomocí nástroje Apache Maven. Pro implementaci jednotkových testů byl použit framework *JUnit 4*.

V rámci zachování konzistence produkčních dat se zpravidla testy neprovádějí proti produkční databázi. Stejně tak se testy neprovádějí proti vývojářské databázi v důsledku zachování opakovatelnosti. Nejčastěji se používá některá z *in-memory* databází, která je vždy vytvořena a inicializována před spuštěním samotných testů. V rámci této práce byla použita *in-memory* databáze HSQLDB.

Vzhledem k tomu, že při vytváření databázového připojení pro aplikaci je používáno velmi obecné rozhraní *javax.sql.DataSource*, výměna mezi produkční databází PostgreSQL a testovací databází HSQLDB je naprosto triviální a skládá se jen a pouze ze změny hodnot atributů při vytváření databázového připojení. Za tímto účelem jsou využívány profily Spring frameworku, které umožňují podmínit konfiguraci aplikace pomocí různých profilů (např. produkční profil, vývojový profil nebo testovací profil). Ve skutečnosti to znamená, že pokud je aktivní produkční profil, dochází k vytvoření datového připojení pro databázi PostgreSQL. Ve chvíli kdy je aktivní testovací profil, dochází k vytvoření databázového připojení pro databázi HSQLDB.

Dle konvencí je pro každou testovanou třídu vytvořena stejnojmenná třída se sufixem *Test*. Například pro třídu *RoleDao* je vytvořena testovací třída *RoleDaoTest*, kde každá testovaná metoda obsahuje prefix *test*. Příklad kódu třídy *RoleDaoTest* je možné spatřit na obrázku 6.1.

Na tomto obrázku je možné spatřit dvě testovací metody. V rámci první metody je testováno získání role podle definovaného identifikátoru. V rámci druhé metody je testováno získání všech rolí bez jakýchkoliv kritérií. Zmíněné metody spoléhají na testovací data, která jsou do databáze insertována během inicializace testovací databáze. Dále je možné spatřit, že testovací třída *RoleDaoTest* dědí ze třídy *AbstractSpringContextTest*. V rámci této třídy dochází k inicializaci aplikace a nastavení příslušného testovacího

```java
@SecurityContextTest
public class RoleDaoTest extends AbstractSpringContextTest {

    @Autowired
    private RoleDao roleDao;

    @Test
    public void testFindById() {
        String identifier = "ff80818147d8f1b20147d8f28d940000";
        Role role = roleDao.findById(identifier);
        Assert.assertNotNull(role);
        Assert.assertEquals(identifier, role.getId());
    }

    @Test
    public void testFindRoles() {
        List<Role> roles = roleDao.findRoles(null);
        Assert.assertNotNull(roles);
        Assert.assertEquals(1, roles.size());
    }
```

Obrázek 6.1: Ukázka kódu z testovací třídy *RoleDaoTest*.

profilu. Vzhledem k tomu, že v aplikaci dochází při volání metod ke kontrole role přihlášeného uživatele, je nezbytné, aby i při provádění testů byl přihlášený systémový uživatel obsahující příslušnou roli. To je zajišťováno pomocí anotace *@SecurityContextTest*, v rámci které dochází k přihlášení systémového uživatele před provedením testů a k jeho odhlášení po provedení testů.

6.2 Systémové testy

V rámci systémových testů dochází k testování větších funkčních celků. Testování by mělo probíhat na základě testovacích scénářů, kdy jsou testovány případy užití, které mohou v praxi nastat. Tyto testy jsou prováděny odpovědnou osobou (testerem) a jsou používány v pozdější fázi implementace. Každý testovací scénář musí obsahovat tyto náležitosti:

- Popis případu užití,
- Popis jednotlivých kroků pro provedení testu,
- Popis stavu systému před provedením testu,
- Popis stavu systému po provedení testu.

S přihlédnutím k současné fázi projektu nebyly testovací scénáře dosud vytvořeny a systémové testy byly prováděny vždy jen a pouze vývojářem systému. To znamená, že vždy po implementaci nějakého logického celku bylo ověřeno, že se aplikace chová přesně tak, jak bylo zamýšleno.

Kapitola 7

Provoz řešení v cloudu

Tato kapitola pojednává o možnosti nasazení a využívání vytvořeného řešení v cloudu, tedy bez nutnosti využívat infrastrukturu organizace.

Pro úspěšné nasazení vytvořeného řešení je nezbytné nainstalovat veškeré prerekvizity uvedené v příloze B.1. Tento krok nepředstavuje žádný problém, jelikož požadované závislosti jsou volně dostupné pro různé platformy. Mnohem komplikovanějším úkolem může být zajištění konektivity pro integraci s cílovými a zdrojovými systémy. Zajištění konektivity pro integraci s cílovým systémem *Google Apps* problémem není, jelikož samotný cílový systém je taktéž cloudovou službou s veřejně dostupným rozhraním. U zajištění konektivity pro integraci s cílovým systémem *Active Directory* nastává problém právě tehdy, když je samotný cílový systém provozován ve vnitřní síti organizace s blokací veškerého vnějšího síťového provozu prostřednictvím firewallu. Prvním z možných řešení je vytvoření prostupu skrze firewall, což se v konzervativní organizaci nemusí setkat s pochopením. Alternativním přístupem je vyvolávat komunikaci z vnitřní sítě organizace. Tento krok by vyžadoval naprogramování aplikace, která by byla nainstalována na server k *Active Directory* a sloužila by jako zprostředkovatel komunikace mezi *Active Directory* a řešením nasazeným v cloudu. V podstatě stejná situace platí i pro zdrojové systémy.

V dnešní době se v kontextu cloudových služeb poměrně často mluví o takzvané multitenantnosti (z anglického *multitenancy*). Jedná se o softwarovou architekturu, kdy jednu instanci aplikace využívá více organizací neboli nájemců (jeden nájemce je anglicky označován jako *tenant*). Formální definice říká, že multitenantnost je přístup ke sdílení aplikace mezi více tenantů tak, že každému tenantovi je poskytována vyhrazená část aplikace, která je oddělená od ostatních s přihlédnutím k výkonu a bezpečnosti [18]. Zmíněné organizace nemusí mít mezi sebou žádný vztah. Zajištění izolace jednotlivých tenantů je prováděno pomocí striktního oddělení jejich dat. Způsobů, jak oddělit data je vícero. Jedním ze způsobů je rozdělení dat mezi různé databázové stroje, kdy pro každého tenanta bude existovat právě jeden databázový stroj. Jiným přístupem je použití jednoho databázového stroje a oddělení databázových schémat, kde bude opět existovat právě jedno schéma pro každého tenanta. Hlavní výhodou multitenantnosti je snížení nákladů pro samotné tenanty, výhodou pro implementátora je nutnost spravovat jen a pouze jednu instanci aplikace. Řešení, které bylo vytvořeno v této práci, defaultně nepodporuje multitenantnost. Je to z toho důvodu, že zadavatel nevznesl požadavek na implementaci této architektury. Případné zavedení podpory by muselo být řešeno v následujících fázích projektu jako změnový požadavek.

Kapitola 8

Návrhy pro budoucí rozvoj aplikace

V této kapitole budou v krátkosti popsány návrhy funkcionalit, které by mohly být implementovány v následujících fázích projektu.

První funkcionalitou je zavedení *delegací*. Delegace jsou používány pro převádění povinností mezi uživateli. Delegovaný tedy přebírá veškeré povinnosti uživatele, který na něj své povinnosti delegoval, což ve výsledku znamená, že za něj například schvaluje vytvořené žádosti. Delegace jsou vždy vytvářeny na definované období.

Druhou funkcionalitou je zavedení *eskalací*. Eskalace jsou využívány v rámci schvalovacích workflow, kde zajišťují nahrazení schvalovatelů na základě dalších kritérií. Příkladem může být využití eskalací na nadřízené právě tehdy, když daný úkol bude bez odezvy po dobu jednoho týdne. To znamená, že pokud schvalovatel daného úkolu neprovede schválení nebo zamítnutí do jednoho týdne od přiřazení, bude tento schvalovatel nahrazen jeho nadřízeným, který se následně bude účastnit schvalovacího procesu.

Třetí funkcionalitou je zavedení podpory pro generování rozličných *reportů*. Reporty mohou obsahovat různorodé informace a shrnutí. Příkladem může být report účtů, kde jsou zobrazeny veškeré vytvořené účty a hodnoty atributů těchto účtů. Dalším příkladem by mohl být tzv. *non-compliance* report. Tento report obsahuje záznamy o nesrovnalostech skupin a účtů mezi IDMS a připojenými cílovými systémy. Reporty mohou mít různé formáty jako je PDF nebo HTML.

Čtvrtou funkcionalitou je zavedení podpory pro emailové *notifikace*. Emailové účty jsou v současné době evidovány pro každého uživatele aplikace. Notifikace mohou být odesílány v různých situacích, jako je například přiřazení úkolu ve schvalovacím workflow.

Samostatnou kapitolou je zavedení nových zdrojových a cílových systémů podle dalších požadavků zadavatele projektu, případně budoucích zákazníků.

Kapitola 9

Závěr

Cílem této práce bylo analyzovat, navrhnout a implementovat systém pro správu identit, který bude jednoduše uplatnitelný u malých a středních firem. Důvodem pro implementaci takového řešení byla potřeba zařazení řešení do portfolia organizace, v které je autor práce zaměstnán.

V teoretické části práce byl vhodnou kompozicí dostupné literatury vytvořen detailní, přesto kompaktní, popis RBAC i non-RBAC přístupů v modelování zabezpečení. V rámci analytické části práce byla prostřednictvím dotazníkového šetření provedena analýza vybraného vzorku cílených organizací. Dále byl na základě řízených rozhovorů se zadavatelem vytvořen popis funkčních a nefunkčních požadavků na systém. Na základě analýzy organizací byl v návrhové části vytvořen RBAC model, který zcela vyhovuje potřebám cílených organizací. Dále byl proveden kompletní návrh systému tak, aby reflektoval zmíněné funkční a nefunkční požadavky a byly implementovány požadované funkčnosti. Následně byla provedena samotná implementace systému, který byl postaven na platformě Java s využitím populárního frameworku Spring.

Úspěšnost celého projektu bude vyhodnocena až po nasazení do ostrého provozu u některého ze zákazníků. V době psaní této práce byla ukončena první fáze projektu, kde výsledkem je plně funkční aplikace obsahující veškeré navrhnuté funkcionality a schopná produkčního nasazení. Skutečné nasazení prozatím proběhlo jen a pouze v rámci testovacího prostředí zadavatele, kde hlavním účelem bylo odstranění nesrovnalostí.

Autor práce předpokládá, že rozvoj aplikace bude dále probíhat především na základě požadavků od budoucích zákazníků. Přesto byly v předchozí kapitole popsány návrhy pro budoucí rozvoj aplikace, které mají podle autora práce značnou přidanou hodnotu a měly by být implementovány v následujících fázích projektu.

Na základě všeho výše uvedeného si autor práce dovoluje považovat veškeré cíle práce za úspěšně splněné. Mimoto byly některé úkony (např. implementace schvalovacích workflow či implementace importu uživatelských identit z konkrétního HR systému) provedeny nad rámec samotného zadání.

Literatura

[1] Lewis, Jamie. *Enterprise Identity Management - It's About the Business*. [online]. 2003 [cit. 2014-06-21]. Dostupné z: https://www.researchgate.net/publication/220174164_Enterprise_Identity_Management_-_It%27s_About_the_Business

[2] Official journal of the european union. *COMMISSION RECOMMENDATION of 6 May 2003 concerning the definition of micro, small and medium-sized enterprises*. [online]. 2003 [cit. 2014-06-24]. Dostupné z: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2003:124:0036:0041:en:PDF

[3] Jøsang, Audun a Simon Pope. *User Centric Identity Management*. [online]. 2005 [cit. 2014-06-27]. Dostupné z: http://persons.unik.no/josang/papers/JP2005-AusCERT.pdf

[4] Itu-t. *Y.2720: NGN identity management framework*. [online]. 2009 [cit. 2014-06-27]. Dostupné z: http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.2720-200901-I!!PDF-E&type=items

[5] Sandhu Ravi, Coyne Edward, Feinstein Hal, Youman Charles. *Role-Based Access Control Models*. [online]. 1996 [cit. 2014-06-27]. Dostupné z: http://modelibra.googlecode.com/svn/trunk/Modelibra/doc/security/sandhu96.pdf

[6] Ferraiolo, David a Richard Kuhn. *Role-Based Access Controls*. [online]. 1992 [cit. 2014-06-27]. Dostupné z: http://arxiv.org/pdf/0903.2171.pdf

[7] Saltzer, Jerome a Michael Schroeder. *The Protection of Information in Computer Systems*. [online]. 1975 [cit. 2014-06-27]. Dostupné z: http://ix.cs.uoregon.edu/~butler/teaching/10F/cis607/papers/saltzer1975.pdf

[8] Kugblenu, Francis a Memon Asim. *Separation of Duty in Role Based Access Control System*. [online]. 2007 [cit. 2014-06-28]. Dostupné z: http://www.bth.se/fou/cuppsats.nsf/all/52d12689b4758c84c12572a600386f1d/$file/mcs-2006-16.pdf

[9] Zhang, Dana, Kotagiri Ramamohanarao a Tim Ebringer. *Role Engineering using Graph Optimisation*. [online]. 2007 [cit. 2014-06-28]. Dostupné z: http://www.researchgate.net/publication/221366849_Role_engineering_using_graph_optimisation/file/72e7e5245bca0a1815.pdf

[10] Apte, S. S . a S. A. Ubale. *Comparison of ACL Based Security*. [online]. 2014 [cit. 2014-07-04]. Dostupné z: http://ijshre.com/wp-content/uploads/2014/06/IJSHRE-2644.pdf

[11] NIST. *A survey of access control models*. [online]. 2009 [cit. 2014-07-04]. Dostupné z: http://csrc.nist.gov/news_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf

[12] HU, Vincent C., David FERRAIOLO a Rick KUHN. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. [online]. 2014 [cit. 2014-07-04]. Dostupné z: http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf

[13] GIUNCHIGLIA, Fausto, Rui ZHANG a Bruno CRISPO. *RelBAC: Relation Based Access Control*. [online]. 2008 [cit. 2014-07-05]. Dostupné z: http://eprints.biblio.unitn.it/1460/1/040.pdf

[14] MCGRAW, Robert W. *Risk-Adaptable Access Control (RAdAC)*. [online]. 2009 [cit. 2014-07-06]. Dostupné z: http://csrc.nist.gov/news_events/privilege-management-workshop/radac-Paper0001.pdf

[15] ASKÅSEN, Anders, Paul BRYAN, Mark CRAIG, Andi EGLOFF, Laszlo HORDOS, Matthias TRISTL a Lana FROST. *OpenIDM 3.1.0 Integrator's Guide*. [online]. 2014 [cit. 2014-07-12]. Dostupné z: http://openidm.forgerock.org/doc/integrators-guide/

[16] SWEDISH INTERNATIONAL DEVELOPMENT COOPERATION AGENCY. *The Logical Framework Approach*. [online]. 2004 [cit. 2014-07-12]. Dostupné z: http://www.mfem.gov.ck/docs/AMD/Development%20Resources/Extra%20Reading/SIDA_LFA.pdf

[17] SPRING SOURCE. *Spring Framework - Reference Documentation*. [online]. [cit. 2014-07-15]. Dostupné z: http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/

[18] KREBS, Rouven, Christof MOMM a Samuel KOUNEV. *Architectural Concerns in Multi-Tenant SaaS Applications*. [online]. 2012 [cit. 2014-10-15]. Dostupné z: https://sdqweb.ipd.kit.edu/publications/pdfs/KrMoKo2012-closer-multitenant-sass.pdf

Seznam obrázků

Seznam použitých zkratek

ABAC Attribute Based Access Control

ACL Access Control List

AD Active Directory

AOP Aspect Oriented Programming

API Application Programming Interface

BPMN Business Process Model and Notation

CRUD Create, Read, Update, Delete

CSS Cascading Style Sheets

CSV Comma-Separated Values

DAC Discretionary Access Control

DAO Data Access Object

DN Distinguished Name

EJB Enterprise Java Beans

EL Unified Expression Language

ERP Enterprise Resource Planning

GA Google Apps

GUI Graphical User Interface

HR Human Resources

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IDM Identity Management

IDMS Identity Management System

IoC Inversion of Control

IT Information Technology

JCP Java Community Process

JDBC Java Database Connectivity

JPA Java Persistence API

JSON JavaScript Object Notation

JSP JavaServer Pages

JSTL JSP Standard Tag Library

JVM Java Virtual Machine

LDAP Lightweight Directory Access Protocol

MAC Mandatory Access Control

MVC Model-View-Controller

ORM Object-Relational Mapping

PDF Portable Document Format

POJO Plain Old Java Object

RAdAC Risk-Adaptive Access Control

RBAC Role-Based Access Control

RelBAC Relation-Based Access Control

REST Representational State Transfer

SMB Small and Medium Business

SQL Structured Query Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

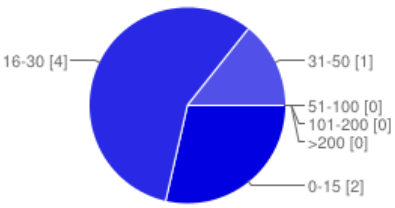
WAR Web Application Archive

XML Extensible Markup Language

Příloha A

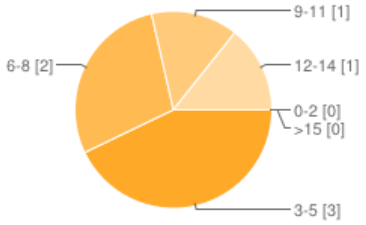
Výsledky dotazníkového šetření

Kolik má vaše organizace zaměstnanců, kteří mají přístup do nějaké aplikace?



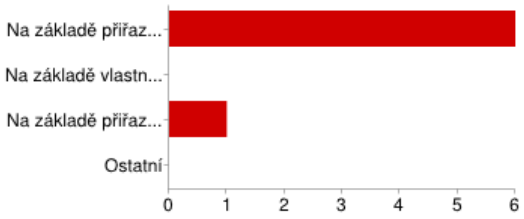
0-15	**2**	29 %
16-30	**4**	57 %
31-50	**1**	14 %
51-100	**0**	0 %
101-200	**0**	0 %
>200	**0**	0 %

Kolik v organizaci využíváte aplikací s autentizací/autorizací uživatelů?



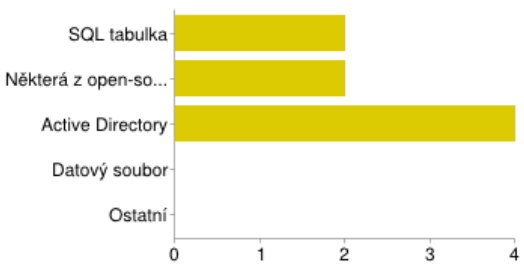
0-2	**0**	0 %
3-5	**3**	43 %
6-8	**2**	29 %
9-11	**1**	14 %
12-14	**1**	14 %
>15	**0**	0 %

Jakým způsobem je řešeno řízení přístupu uživatelů ke zdrojům?



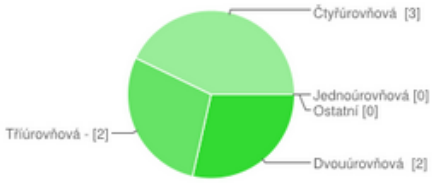
Na základě přiřazených skupin	**6**	86 %
Na základě vlastnictví objektu	**0**	0 %
Na základě přiřazených atributů	**1**	14 %
Ostatní	**0**	0 %

Vyjmenujte používaná uložiště uživatelů?



SQL tabulka	**2**	29 %
Některá z open-source implementací LDAP	**2**	29 %
Active Directory	**4**	57 %
Datový soubor	**0**	0 %
Ostatní	**0**	0 %

Jak vypadá struktura organizace?

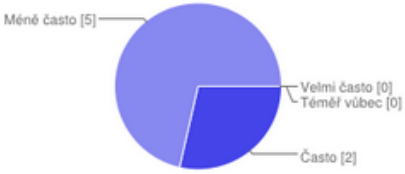


Jednoúrovňová - neexistuje vztah 'nadřízený/podřízený'	**0**	0 %
Dvouúrovňová - počet úrovní mezi kořenem a listem stromu je maximálně dva	**2**	29 %
Tříúrovňová - počet úrovní mezi kořenem a listem stromu je maximálně tři	**2**	29 %
Čtyřúrovňová - počet úrovní mezi kořenem a listem stromu je maximálně čtyři	**3**	43 %
Ostatní	**0**	0 %

Překrývají se oprávnění mezi různými pozicemi?

Ano	**7**	100 %
Ne	**0**	0 %

Jak často jsou zaměstnancům přiřazována/odebírána oprávnění?

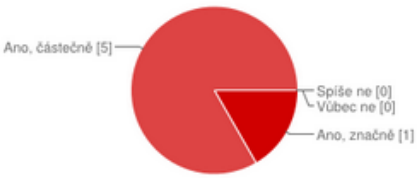


Velmi často	**0**	0 %
Často	**2**	29 %
Méně často	**5**	71 %
Téměř vůbec	**0**	0 %

Pracuje na některé pozici více zaměstnanců?

Ano	**6**	86 %
Ne	**1**	14 %

Liší se oprávnění zaměstnanců pracujících na stejné pozici?



Ano, značně	**1**	14 %
Ano, částečně	**5**	71 %
Spíše ne	**0**	0 %
Vůbec ne	**0**	0 %

Příloha B

Uživatelská příručka

V této příloze lze nalézt uživatelskou příručku aplikace MIDM obsahující základní popis informací nezbytných pro provoz aplikace a dále základní popis uživatelského rozhraní aplikace a uživatelských funkcionalit.

B.1 Popis instalace

Pro úspěšné spuštění aplikace jsou nezbytné následující prerekvizity:

- Java JRE 7 nebo vyšší,
- Apache Tomcat 7 nebo vyšší,
- PostgreSQL 9.3 nebo vyšší.

Pro úspěšné napojení aplikace na cílové systémy jsou nezbytné následující prerekvizity:

- Microsoft Windows Server 2008 R2 Standard s aktivní rolí *Active Directory Domain Services* a *Server DNS* a nainstalovaným OpenICF .NET konektorem[1],
- Doména a vytvořený Google Apps administrátorský účet pro danou doménu.

Postup pro úspěšné spuštění aplikace je následující:

1. Vytvořit databázi s libovolným názvem a vlastníkem,
2. Aplikovat SQL skripty pro inicializaci databáze ze složky */WEB-INF/classes/sql/*, která se nachází v archivu *midm.war*,
3. Upravit hodnoty atributů databázového připojení v souboru */WEB-INF/classes/application.properties*, který se nachází v archivu *midm.war*,
4. Rozbalit adresář *midm_openidm.zip* do libovolné složky,
5. Upravit hodnotu atributu *openidm.security.keystore* v souboru */WEB-INF/classes/application.properties*, který se nachází v archivu *midm.war*,

[1] Viz http://openidm.forgerock.org/doc/integrators-guide/index/chap-resource-conf.html#install-.net-connector.

6. Upravit hodnoty atributů databázového připojení v souboru *repo.jdbc.json*, který se nachází ve složce *%midm_openidm_HOME%/midm/conf/*,

7. Upravit nastavení OpenICF konektorů:

   (a) Upravit hodnoty atributů pro připojení k OpenICF .NET konektoru v souboru *provisioner.openicf.connectorinfoprovider.json*, který se nachází ve složce *%midm_openidm_HOME%/midm/conf/*,

   (b) Upravit hodnoty atributů Active Directory OpenICF konektoru v souboru *provisioner.openicf-ad.json*, který se nachází ve složce *%midm_openidm_HOME%/midm/conf/*,

   (c) Upravit hodnoty atributů Google Apps OpenICF konektoru v souboru *provisioner.openicf-ga.json*, který se nachází ve složce *%midm_openidm_HOME%/midm/conf/*.

8. Spustit *OpenIDM* s konfiguračními soubory ze složky *midm*:

   (a) Na operačním systému Linux provést příkaz *sudo ./startup.sh -p midm* v rootu složky *%midm_openidm_HOME%*,

   (b) Na operačním systému Windows provést příkaz *startup.bat -p midm* v rootu složky *%midm_openidm_HOME%*.

9. Nakonfigurovat a spustit aplikaci pro import uživatelských identit *MIDM-HRS*:

   (a) Upravit hodnoty atributů pro připojení ke zdrojovému systému v souboru *application.properties*, který se nachází v archivu *midm-hrs.jar*,

   (b) Spustit aplikaci pomocí příkazu *java -jar midm-hrs.jar*.

10. Nakopírovat soubor *midm.war* do složky *%CATALINA_HOME%/webapps* a spustit Apache Tomcat.

Veškeré archivy a podklady jsou dostupné na CD, které je přiloženo k této práci. Upravením hodnot atributů se rozumí nahrazení *placeholderů* skutečnými hodnotami. Pro samotné spuštění aplikace MIDM bez jakékoliv synchronizace s cílovými systémy je možné kroky 4-8 vynechat. Pro spuštění aplikace MIDM bez importu uživatelských identit ze zdrojového systému je možné krok 9 vynechat.

B.2 Popis aplikace

Tato podkapitola obsahuje základní popis uživatelského rozhraní a uživatelských funkcionalit aplikace MIDM. Pro doplnění textového popisu je možné ve složce *guide_images*, která se nachází na přiloženém CD, nalézt obrázkové podklady.

B.2.1 Přihlášení do aplikace

Pro provedení jakéhokoliv úkonu v aplikaci je nutné se přihlásit. Přihlášení probíhá pomocí příslušné stránky a zadání uživatelského jména a hesla. Tato stránka je zobrazena vždy při přístupu nepřihlášeného uživatele na jakoukoli stránku aplikace. Uživatel, který by měl sloužit jen a pouze pro účely prvního přihlášení a vytvoření dalších uživatelských identit má přihlašovací údaje nastavené na *admin* : *admin*.

B.2.2 Úvodní obrazovka aplikace

Po přihlášení do aplikace je zobrazena úvodní obrazovka aplikace. Na této obrazovce jsou mimo jiné uvedeny základní informace o projektu. V horní části je zobrazena lišta tvořící menu aplikace. Menu aplikace obsahuje následující položky:

- Uživatelé,
- Role,
- Oprávnění,
- Skupiny,
- Účty,
- Logy,
- Žádosti.

Prostřednictvím zmíněných položek v menu je možné se prokliknout na příslušný seznam objektů. V pravé horní části je uvedeno jméno přihlášeného uživatele, kde po kliknutí na toto jméno je rozevřeno menu s položkami pro odhlášení a přechod na detail uživatele. Vedle jména přihlášeného uživatele jsou umístěny vlaječky pro změnu jazykové mutace aplikace.

B.2.3 Uživatelé

Po kliknutí na stejnojmennou položku v menu je uživatel přesměrován na stránku se seznamem uživatelů. Na této stránce je možné provést několik úkonů:

- Zobrazit detail vybraného uživatele,
- Vytvořit nového uživatele,
- Importovat uživatele ze souboru CSV.

Na detail konkrétního uživatele je možné se prokliknout pomocí souvisejícího identifikátoru v seznamu všech uživatelů. Na stránce s detailem uživatele lze spatřit veškeré informace o daném uživateli. Dále je možné spatřit přiřazené role, oprávnění, skupiny a účty a je možné provést následující úkony:

- Upravit atributy daného uživatele,
- Smazat daného uživatele,
- Vytvořit žádost na přiřazení rolí nebo oprávnění,
- Vytvořit žádost na odebrání rolí nebo oprávnění.

B.2.4 Role

Po kliknutí na stejnojmennou položku v menu je uživatel přesměrován na stránku se seznamem rolí. Na této stránce je možné provést několik úkonů:

- Zobrazit detail vybrané role,
- Vytvořit novou roli pro konkrétní cílový systém.

Na detail konkrétní role je možné se prokliknout pomocí souvisejícího identifikátoru v seznamu všech rolí. Na stránce s detailem role lze spatřit veškeré informace o dané roli. Dále je možné spatřit přiřazené nadřazené a podřazené role, přiřazené oprávnění a seznam uživatelů, kteří mají přiřazenou danou roli. Na detailu je taktéž možné provést následující úkony:

- Upravit atributy dané role,
- Smazat danou roli,
- Vytvořit žádost na přiřazení role do nadřazené role, přiřazení podřazené role, přiřazení oprávnění nebo přiřazení role uživatelům,
- Vytvořit žádost na odebrání role z nadřazené role, odebrání podřazené role, odebrání oprávnění nebo odebrání role uživatelům.

B.2.5 Oprávnění

Po kliknutí na stejnojmennou položku v menu je uživatel přesměrován na stránku se seznamem oprávnění. Na této stránce je možné provést několik úkonů:

- Zobrazit detail vybraného oprávnění,
- Vytvořit nové oprávnění pro konkrétní cílový systém.

Na detail konkrétního oprávnění je možné se prokliknout pomocí souvisejícího identifikátoru v seznamu všech oprávnění. Na stránce s detailem oprávnění lze spatřit veškeré informace o daném oprávnění. Dále je možné spatřit přiřazené skupiny, seznam rolí, v kterých je dané oprávnění přiřazené a seznam uživatelů, kteří mají přiřazené dané oprávnění. Na detailu je taktéž možné provést následující úkony:

- Upravit atributy daného oprávnění,
- Smazat dané oprávnění,
- Vytvořit žádost na přiřazení skupin, přiřazení oprávnění uživatelům a přiřazení oprávnění do rolí,
- Vytvořit žádost na odebrání skupin, odebrání oprávnění uživatelům a odebrání oprávnění z rolí.

B.2.6 Skupiny

Po kliknutí na stejnojmennou položku v menu je uživatel přesměrován na stránku se seznamem skupin. Na této stránce je možné provést několik úkonů:

- Zobrazit detail vybrané skupiny,
- Vytvořit novou skupinu pro konkrétní cílový systém.

Na detail konkrétní skupiny je možné se prokliknout pomocí souvisejícího identifikátoru v seznamu všech skupin. Na stránce s detailem skupiny lze spatřit veškeré informace o dané skupině. Dále je možné spatřit přiřazené uživatelské účty a seznam oprávnění, v kterých je daná skupina přiřazená. Na detailu je taktéž možné provést následující úkony:

- Upravit atributy dané skupiny,
- Smazat danou skupinu (pouze pokud není přiřazená do oprávnění a tedy není přiřazená žádným uživatelům),
- Vytvořit žádost na přiřazení do oprávnění,
- Vytvořit žádost na odebrání z oprávnění.

B.2.7 Účty

Po kliknutí na stejnojmennou položku v menu je uživatel přesměrován na stránku se seznamem účtů. Prostřednictvím této stránky je možné zobrazit detail vybraného účtu.

Na detail konkrétního účtu je možné se prokliknout pomocí souvisejícího identifikátoru v seznamu všech účtů. Na stránce s detailem účtu lze spatřit veškeré informace o daném účtu a uživateli, pro kterého byl daný účet vytvořen. Dále je možné spatřit seznam skupin, v kterých je daný účet přiřazen. Administrátor nebo manažer může z detailu účtu provést úpravu atributů daného účtu. Uživatel může z detailu účtu provést změnu hesla svého účtu.

B.2.8 Logy

Po kliknutí na stejnojmennou položku v menu je uživatel přesměrován na stránku se seznamem logů. V tabulce jsou pro každý záznam zobrazeny následující atributy:

- Identifikátor logu,
- Uživatel, který vznik logu vyvolal,
- Objekt, kterého se daný log týká,
- Datum vytvoření logu,
- Typ provedené operace.

Po kliknutí na typ operace je zobrazeno modální okno s detaily o daném logu. Především se jedná o zobrazení změněných hodnot daného objektu. Logy není možné nijak modifikovat.

B.2.9 Žádosti

Po kliknutí na stejnojmennou položku v menu je rozevřeno další menu s následujícími položkami:

- Všechny žádosti,
- Úkoly.

Po kliknutí na položku *Všechny žádosti* je uživatel přesměrován na stránku se seznamem žádostí. Prostřednictvím této stránky je možné zobrazit detail vybrané žádosti.

Na detail konkrétní žádosti je možné se prokliknout pomocí souvisejícího identifikátoru v seznamu všech žádostí. Na stránce s detailem žádosti lze spatřit následující informace:

- Identifikátor žádosti,
- Typ žádosti,
- Iniciátor žádosti,
- Výsledek žádosti,
- Datum vzniku žádosti,
- Datum dokončení žádosti,
- Subjekt žádosti (objekt, kterému je jiný objekt přiřazován nebo odebírán),
- Objekt žádosti (objekt, který je přiřazován nebo odebírán),
- Schvalovatelé žádosti.

Na detailu každé žádosti je vždy zobrazena sekce obsahující informace o schvalovatelích žádosti. V rámci této sekce je mimo jiné zobrazen název daného úkolu (např. *Schválení garantem oprávnění*), jméno schvalovatele, výsledek či důvod schválení nebo zamítnutí. Uživatel, který je určen jako schvalovatel může z detailu pomocí tlačítka *Schválit* nebo *Zamítnout* schválit či zamítnout daný úkol v žádosti. Po kliknutí na zmíněné tlačítko je vždy zobrazen modální dialog pro zadání důvodu.

Neukončené žádosti lze mazat pomocí příslušného tlačítka na detailu žádosti. Neukončenou žádostí se rozumí žádost, která ještě nebyla zcela schválena.

Po kliknutí na položku *Úkoly* je uživatel přesměrován na stránku se seznamem úkolů. Jedná se o všechny žádosti, které čekají na schválení daným uživatelem. Na detail konkrétní žádosti je možné se prokliknout pomocí souvisejícího identifikátoru v seznamu úkolů. Na detailu je poté možné danou žádost schválit či zamítnout.

Příloha C

Obsah přiloženého CD

Na přiloženém CD lze nalézt adresáře v následující struktuře:

- /
  - **archives** - adresář obsahující archivy související s projektem
  - **images** - adresář obsahující složky s obrázky
    - **guide** - adresář obsahující obrázky pro doplnění textové dokumentace
    - **class_diagram** - adresář obsahující obrázky class diagramů
  - **videos** - adresář obsahující videa, která znázorňují běžné případy užití aplikace
  - **thesis** - adresář obsahující tento dokument ve formátu PDF
    - **source** - adresář obsahující zdrojový kód tohoto dokumentu ve formátu LaTeX
  - **src** - adresář obsahující zdrojový kód vytvořeného řešení

Příloha D

Scénáře případů užití

Název případu užití	Přiřazení rolí
Identifikátor případu užití	MIDM-01
Účel případu užití	Přiřazení rolí uživateli systému.
Primární aktéři • Uživatel • Manažer • Administrátor	
Pomocní aktéři • Systém	
Vstupní podmínky Uživatel, manažer nebo administrátor (dále osoba) je přihlášena do systému a byla přesměrována na příslušnou webovou stránku.	
Základní scénář 1. Osoba vybere uživatele systému, kterému chce přiřadit role. Po kliknutí na tlačítko *přiřadit role* přejde na stránku příslušné žádosti. 2. Systém zobrazí role, které mohou být přiřazeny danému uživateli. 3. Osoba zvolí role, které chce danému uživateli přiřadit a po kliknutí na tlačítko *odeslat* dojde k odeslání žádosti. 4. Systém na základě odeslaných dat vytvoří žádost(i). 5. Schvalovatelé (osoby, které musí odeslanou žádost schválit) schválí žádost pomocí UC **MIDM-06-Schválení žádosti**. 6. UC končí.	

Alternativní scénáře

4.a - Žádost nebylo možné odeslat

1. Osobě se zobrazí chybová hláška. Akce odeslání může být opakována.

5.a - Žádost nebyla schválena (viz **MIDM-07-Zamítnutí žádosti**)

1. Osobě se zobrazí hláška s informací o neschválení žádosti. Žádost může být znovu vytvořena a odeslána.

Název případu užití	Odebrání rolí
Identifikátor případu užití	MIDM-02
Účel případu užití	Odebrání rolí uživateli systému.

Primární aktéři

- Uživatel
- Manažer
- Administrátor

Pomocní aktéři

- Systém

Vstupní podmínky

Uživatel, manažer nebo administrátor (dále osoba) je přihlášena do systému a byla přesměrována na příslušnou webovou stránku.

Základní scénář

1. Osoba vybere uživatele systému, kterému chce odebrat role. Po kliknutí na tlačítko *odebrat role* přejde na stránku příslušné žádosti.
2. Systém zobrazí role, které mohou být odebrány danému uživateli.
3. Osoba zvolí role, které chce danému uživateli odebrat a po kliknutí na tlačítko *odeslat* dojde k odeslání žádosti.
4. Systém na základě odeslaných dat vytvoří žádost(i).
5. Schvalovatelé (osoby, které musí odeslanou žádost schválit) schválí žádost pomocí UC **MIDM-06-Schválení žádosti**.
6. UC končí.

Alternativní scénáře

4.a - Žádost nebylo možné odeslat

1. Osobě se zobrazí chybová hláška. Akce odeslání může být opakována.

5.a - Žádost nebyla schválena (viz **MIDM-07-Zamítnutí žádosti**)

1. Osobě se zobrazí hláška s informací o neschválení žádosti. Žádost může být znovu vytvořena a odeslána.

Název případu užití	Přiřazení oprávnění
Identifikátor případu užití	MIDM-03
Účel případu užití	Přiřazení oprávnění uživateli systému.

Primární aktéři
- Uživatel
- Manažer
- Administrátor

Pomocní aktéři
- Systém

Vstupní podmínky
Uživatel, manažer nebo administrátor (dále osoba) je přihlášena do systému a byla přesměrována na příslušnou webovou stránku.

Základní scénář
1. Osoba vybere uživatele systému, kterému chce přiřadit oprávnění. Po kliknutí na tlačítko *přiřadit oprávnění* přejde na stránku příslušné žádosti.
2. Systém zobrazí oprávnění, které mohou být přiřazeny danému uživateli.
3. Osoba zvolí oprávnění, které chce danému uživateli přiřadit a po kliknutí na tlačítko *odeslat* dojde k odeslání žádosti.
4. Systém na základě odeslaných dat vytvoří žádost(i).
5. Schvalovatelé (osoby, které musí odeslanou žádost schválit) schválí žádost pomocí UC **MIDM-06-Schválení žádosti**.
6. UC končí.

Alternativní scénáře
4.a - Žádost nebylo možné odeslat
1. Osobě se zobrazí chybová hláška. Akce odeslání může být opakována.

5.a - Žádost nebyla schválena (viz **MIDM-07-Zamítnutí žádosti**)
1. Osobě se zobrazí hláška s informací o neschválení žádosti. Žádost může být znovu vytvořena a odeslána.

Název případu užití	Odebrání oprávnění
Identifikátor případu užití	MIDM-04
Účel případu užití	Odebrání oprávnění uživateli systému.

Primární aktéři
- Uživatel
- Manažer
- Administrátor

Pomocní aktéři
- Systém

Vstupní podmínky
Uživatel, manažer nebo administrátor (dále osoba) je přihlášena do systému a byla přesměrována na příslušnou webovou stránku.

Základní scénář
1. Osoba vybere uživatele systému, kterému chce odebrat oprávnění. Po kliknutí na tlačítko *odebrat oprávnění* přejde na stránku příslušné žádosti.
2. Systém zobrazí oprávnění, které mohou být odebrány danému uživateli.
3. Osoba zvolí oprávnění, které chce danému uživateli odebrat a po kliknutí na tlačítko *odeslat* dojde k odeslání žádosti.
4. Systém na základě odeslaných dat vytvoří žádost(i).
5. Schvalovatelé (osoby, které musí odeslanou žádost schválit) schválí žádost pomocí UC **MIDM-06-Schválení žádosti**.
6. UC končí.

Alternativní scénáře
4.a - Žádost nebylo možné odeslat
1. Osobě se zobrazí chybová hláška. Akce odeslání může být opakována.

5.a - Žádost nebyla schválena (viz **MIDM-07-Zamítnutí žádosti**)
1. Osobě se zobrazí hláška s informací o neschválení žádosti. Žádost může být znovu vytvořena a odeslána.

Název případu užití	Import uživatelů z CSV
Identifikátor případu užití	MIDM-05
Účel případu užití	Importování uživatelských identit do systému za pomoci CSV souboru.

Primární aktéři
- Manažer
- Administrátor

Pomocní aktéři
- Systém

Vstupní podmínky
Manažer nebo administrátor (dále osoba) je přihlášena do systému a byla přesměrována na příslušnou webovou stránku.

Základní scénář
1. Osoba klikne na tlačítko *importovat z CSV*.
2. Systém zobrazí dialog pro vybrání souboru, z kterého se mají naimportovat uživatelské identity.
3. Osoba vybere soubor a kliknutím na tlačítko *naimportovat* zahájí import uživatelských identit.
4. Systém na základě zvoleného souboru provede import uživatelských identit.
5. UC končí.

Alternativní scénáře
4.a - Při importu nastane chyba
1. Osobě se zobrazí chybová hláška. Import může být proveden znovu.

Název případu užití	Schválení úkolu v žádosti
Identifikátor případu užití	MIDM-06
Účel případu užití	Schválení úkolu v žádosti pro vytvoření nebo odebrání asociace mezi objekty.

Primární aktéři
- Manažer
- Administrátor
- Uživatel

Pomocní aktéři
- Systém

Vstupní podmínky
Manažer, administrátor nebo uživatel (dále osoba) je přihlášena do systému a je určena jako schvalovatel úkolu v žádosti. Osoba byla přesměrována na příslušnou webovou stránku.

Základní scénář
1. Osoba klikne na tlačítko *schválit*.
2. Systém zobrazí dialog pro zadání důvodu.
3. Osoba vyplní důvod a klikne na tlačítko *uložit*.
4. Systém provede schválení úkolu a uloží vyplněný důvod schválení.
5. UC končí.

Alternativní scénáře
4.a - Při exekuci schválení dojde k chybě
1. Osobě se zobrazí chybová hláška. Schválení může být provedeno znovu.

Název případu užití	Zamítnutí úkolu v žádosti
Identifikátor případu užití	MIDM-07
Účel případu užití	Zamítnutí úkolu v žádosti pro vytvoření nebo odebrání asociace mezi objekty.

Primární aktéři
- Manažer
- Administrátor
- Uživatel

Pomocní aktéři
- Systém

Vstupní podmínky
Manažer, administrátor nebo uživatel (dále osoba) je přihlášena do systému a je určena jako schvalovatel úkolu v žádosti. Osoba byla přesměrována na příslušnou webovou stránku.

Základní scénář
1. Osoba klikne na tlačítko *zamítnout*.
2. Systém zobrazí dialog pro zadání důvodu.
3. Osoba vyplní důvod a klikne na tlačítko *uložit*.
4. Systém provede zamítnutí úkolu a uloží vyplněný důvod zamítnutí.
5. UC končí.

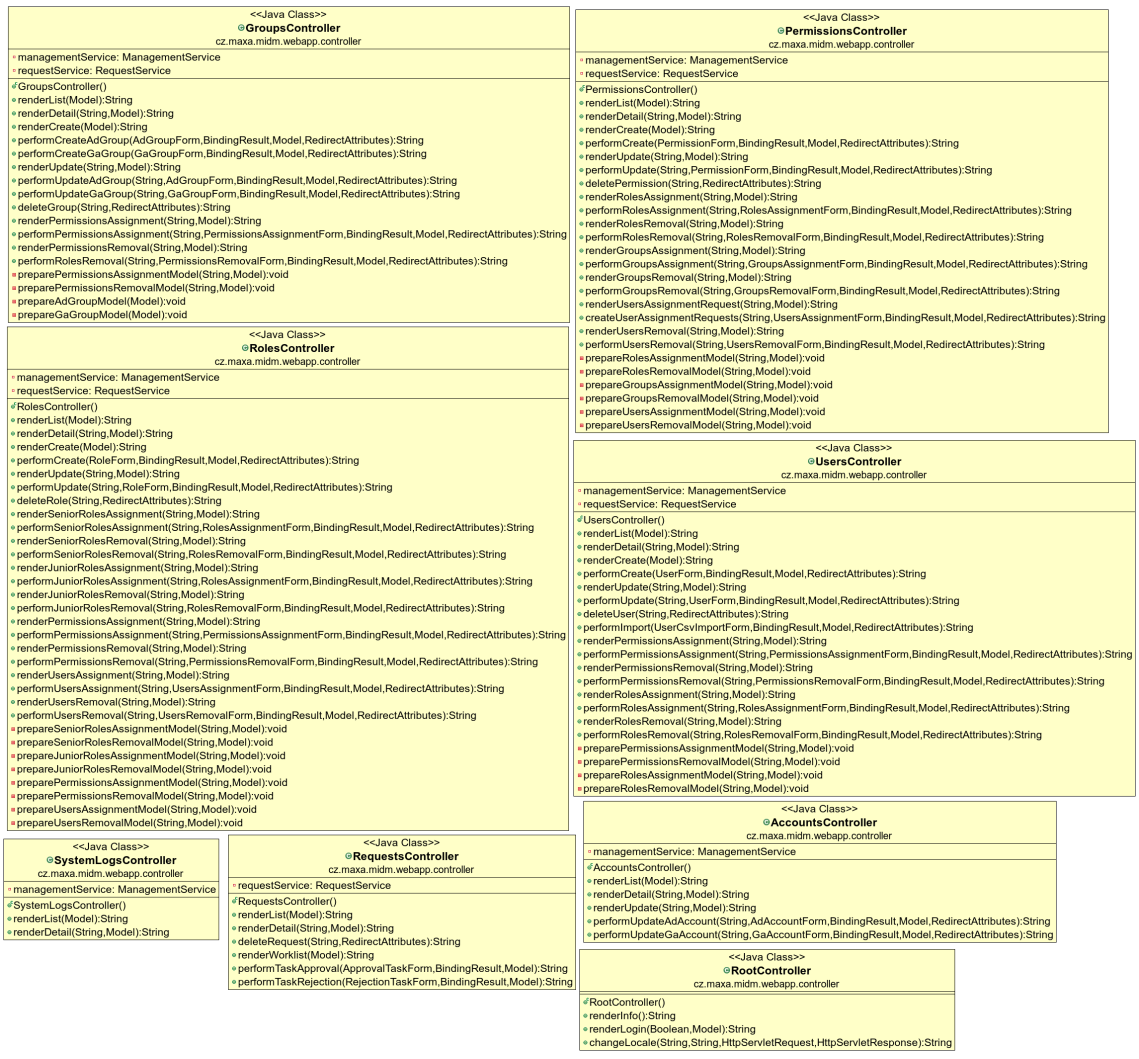
Alternativní scénáře
4.a - Při exekuci zamítnutí dojde k chybě
1. Osobě se zobrazí chybová hláška. Zamítnutí může být provedeno znovu.

Příloha E

Class diagramy

V této příloze lze nalézt class diagramy nejdůležitějších částí aplikace. Obrázky ve větším rozlišení jsou umístěny na přiloženém CD ve složce */images/class_diagram*.

E.1 Class diagram controllerů

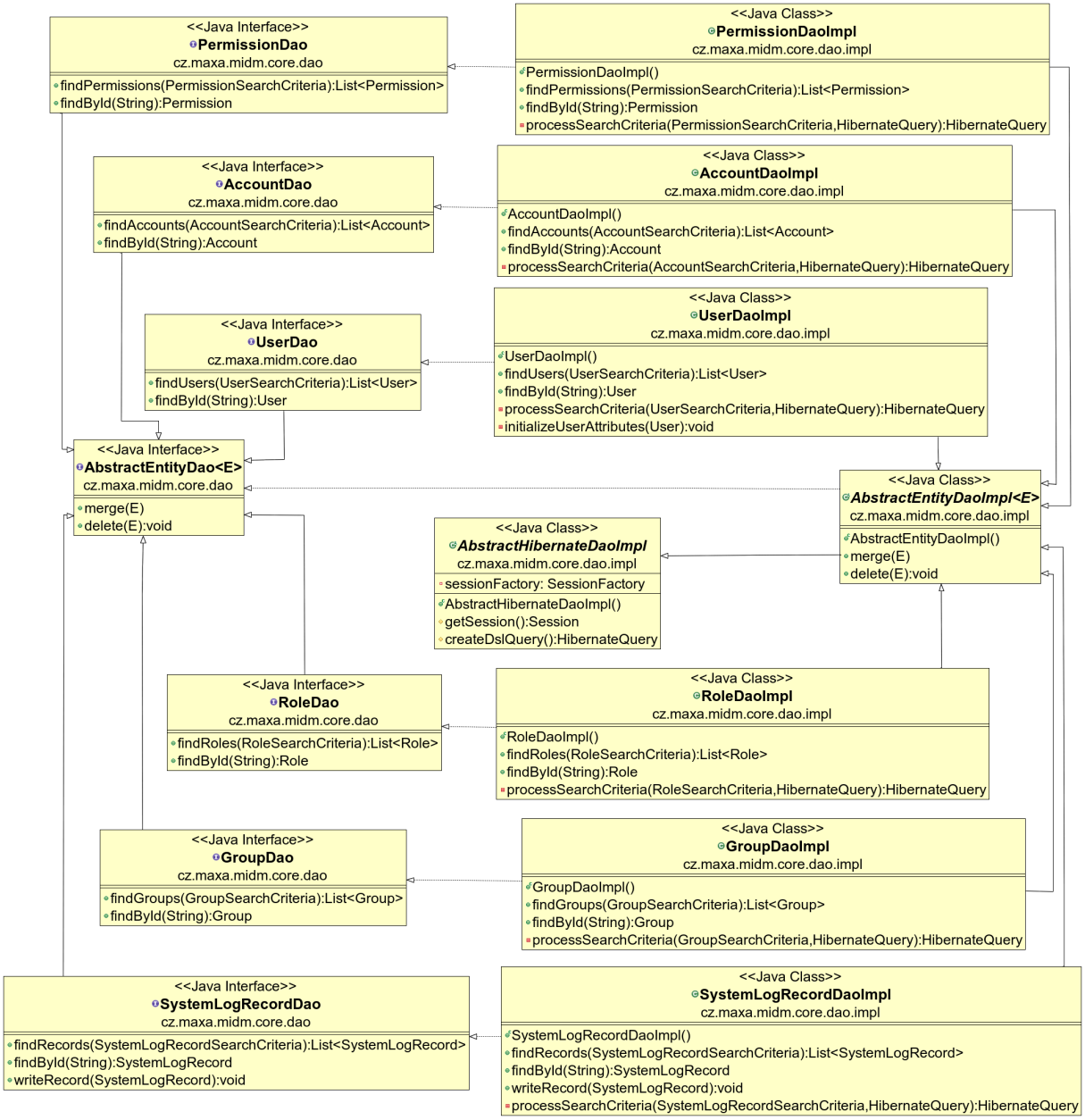


Obrázek E.1: Class diagram controllerů.

E.2 Class diagram servisní vrstvy

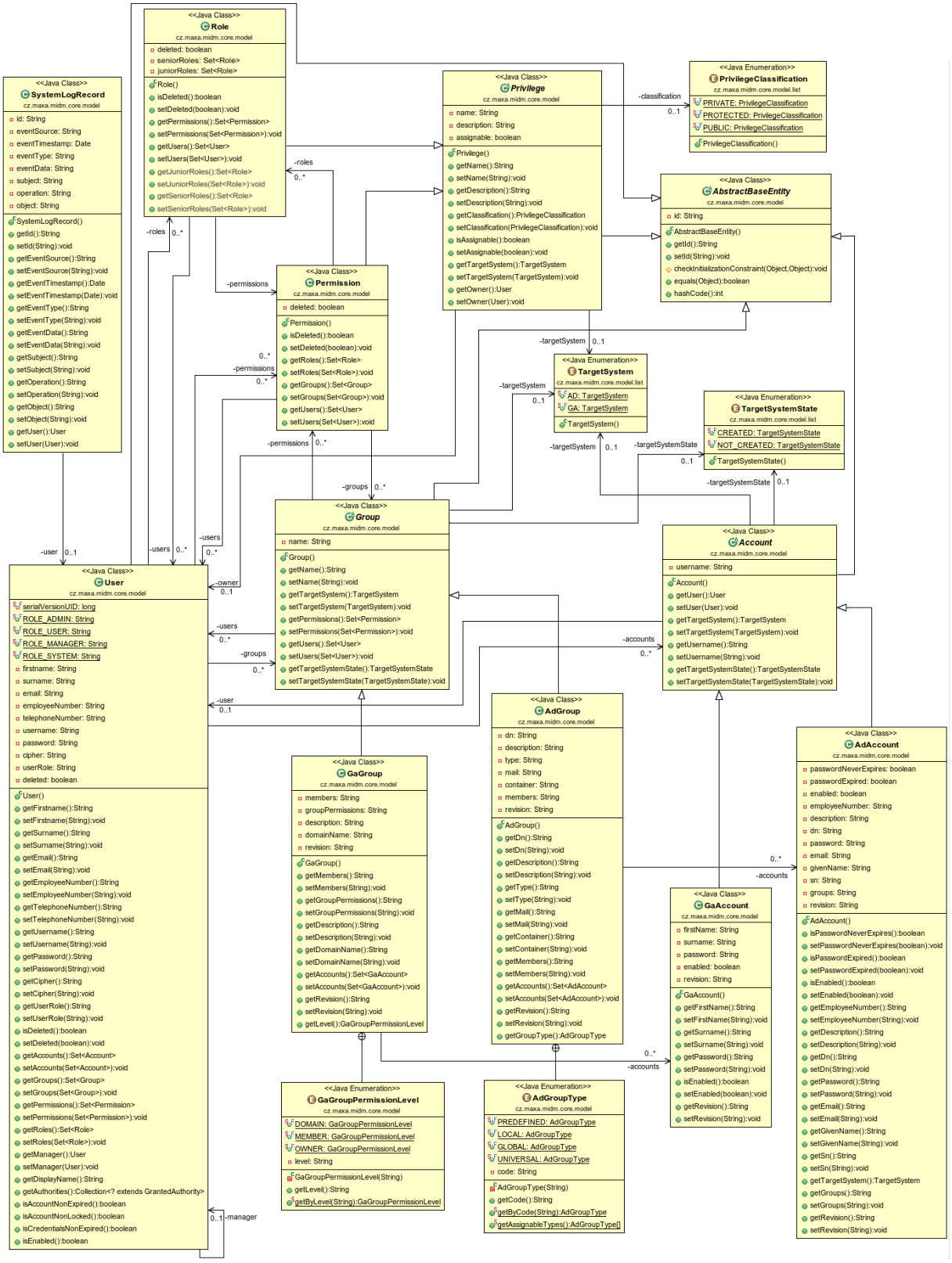
Obrázek E.2: Class diagram servisní vrstvy.

E.3 Class diagram DAO objektů



Obrázek E.3: Class diagram DAO objektů.

E.4 Class diagram modelových objektů



Obrázek E.4: Class diagram modelových objektů.