Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Optimalizace tvorby rolí pomocí RBAC modelu

Diplomová práce

Bc. Martin Klíma

Školitelka: Ing. Marta Vohnoutová

České Budějovice 2017

Bibliografické údaje

Martin Klíma, 2017: Optimalizace tvorby rolí pomocí RBAC modelu. [Role optimization using RBAC model. Mgr. Thesis, in Czech.] – 87 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

The aim of the thesis is to develop algorithm which will be able to optimize roles using RBAC model. The intent of the theoretical part is to analyze RBAC model and present current options which are available for role optimization. The practical part deals with development of algorithm which allows to optimize roles based on defined criteria from user. This algorithm is implemented in programming language Java and builds on Role Process Optimization Model (ROPM). In the last part is showed on example set of data how this algorithm works, step by step, with explanation of each step. Result of this algorithm is new RBAC model defined by user criteria. In this thesis are also listed different approach in role optimization, possible future development and concept of mapping RBAC model to mathematical and data-mining techniques.

NÁZEV:

Optimalizace tvorby rolí pomocí RBAC modelu

ABSTRAKT:

Cílem diplomové práce je vytvoření algoritmu pro optimalizaci tvorby rolí pomocí RBAC modelu. Teoretická část diplomové práce je zaměřena na analýzu RBAC modelu a možností, které existují pro vytváření rolí a jejich optimalizaci. Praktická část se zabývá vývojem vlastního algoritmu, který umožňuje optimalizaci stávajícího či nově vznikajícího RBAC modelu dle zadaných parametrů. Tento algoritmus je implementován v programovacím jazyku Java a staví na Role Optimalizačním Procesním Modelu (ROPM). V poslední části práce je ukázán průchod algoritmu pro ukázková data společně s jednotlivými operacemi a jeho výstupem, který generuje výsledný RBAC model podle zadaných parametrů. V práci jsou taktéž uvedeny možnosti rozšíření algoritmu, společně s jinými přístupy k optimalizaci tvorby rolí a konceptem mapování RBAC modelu na matematické a programové řešení.

KLÍČOVÁ SLOVA:

optimalizace tvorby rolí, RBAC model, role mining, data mining, identity management, java programování, algoritmus pro optimalizaci rolí

TITLE:

Role optimization using RBAC model.

SUMMARY:

The aim of the thesis is to develop algorithm which will be able to optimize roles using RBAC model. The intent of the theoretical part is to analyze RBAC model and present current options which are available for role optimization. The practical part deals with development of algorithm which allows to optimize roles based on defined criteria from user. This algorithm is implemented in programming language Java and builds on Role Process Optimization Model (ROPM). In the last part is showed on example set of data how this algorithm works, step by step, with explanation of each step. Result of this algorithm is new RBAC model defined by user criteria. In this thesis are also listed different approach in role optimization, possible future development and concept of mapping RBAC model to mathematical and data-mining techniques.

KEYWORDS:

role optimization, RBAC model, role mining, data mining, identity management, java programming, role optimization algorithm,

Prohlašuji, že v souladu s § 47b zákona . 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 . 111/1998 Sb. zveřejněny posudky školitele a oponent práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s data-bází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

Prohlašuji, že jsem diplomovou práci „Optimalizace tvorby rolí pomocí RBAC modelu“ vypracoval pod vedením Ing. Marty Vohnoutové samostatně na základě vlastních zjištění a za použití pramenů uvedených v seznamu citované literatury

České Budějovice, 9. dubna 2017

.....

podpis

Poděkování

Rád bych na tomto místě poděkoval vedoucí práce Ing. Martě Vohnoutové za cenné rady, připomínky a za čas strávený konzultacemi. Dále děkuji vedoucímu UAI panu RNDr. Liboru Dostálkovi za pomoc při hledání vhodného tématu a následně kolegovi Ing. Tomáši Vrábelovi za jeho cenné rady a pomoc při analýze a získávání dat. Poděkování patří také mé rodině a přátelům, kteří mě po celou dobu studia podporovali.

Obsah

Seznam obrázků

Seznam tabulek

Slovníček pojmů

IDM / IM Identity Management

RBAC Role Based Access Control

NIST National Institute of Standards and Technology

AM Access Management

DAC Discrete Access Control

MAC Mandatory Access Control

ACL Access Control List

ANSI American National Standards Institute

ROPM Role Optimalization Process Model

JVM Java Virtual Machine

SAP ERP systém

MSSQL Microsoft SQL Server

1. Úvod

V dnešní době velké množství organizací využívá informační systémy jako nepostradatelnou součást svého podnikání. Díky tomu se organizace začínají potýkat s významným růstem počtu uživatelů a informačních zdrojů, které musí spravovat a kontrolovat. Společně s tímto nárůstem se vynořuje potřeba zabezpečit přístup k informacím pouze pro ty, kteří ho skutečně potřebují a mají k němu potřebné povolení od vedení organizace.

Proto, aby si organizace udržela přehled o přístupu svých uživatelů k jednotlivým zdrojům a zabezpečila či minimalizovala riziko neoprávněného přístupu, implementuje některý druh systému pro řízení identit uživatelů, tedy Identity Managementu (IM).

V praxi se dnes nejvíce setkáme s modelem, který je založen na vytváření a přidělování oprávnění, nepřímo přes skupiny oprávnění, pojmenované jako role. Tento model, založený na principu přidělování rolí, se označuje jako „Role Based Access Control“ neboli RBAC.

Samotná definice rolí v RBAC modelu není triviální. Ať už se jedná o implementaci na zelené louce, tedy zcela od začátku, anebo do již fungujícího řešení. V obou případech je nutné správně definovat a nalézt role, které bude následně daný RBAC model využívat. V případě, kdy hledáme roli, která nejvíce vyhovuje naší situaci, mluvíme o optimální nebo ideální roli, kterou následně definujeme v RBAC modelu.

V případě, že budou role špatně nadefinovány, může se stát, že nebudou dávat smysl a budou podstupovat uživatelům více práv, než kolik by měly mít, případně nebudou efektivně plnit úlohu zjednodušení systému a budou zbytečně zatěžovat systém přidělováním oprávněním mimo tyto definované role. Tomuto chceme zabránit takzvanou optimalizací RBAC modelu, případně optimalizací procesu tvorby rolí v tomto modelu.

2. Cíle práce

Hlavní cíle práce jsou:

- Vytvořit algoritmus na optimalizaci tvorby rolí pomocí RBAC modelu.
- Algoritmus musí být parametrizovatelný prostřednictvím konfiguračního souboru nebo údajů zadávaných na vstupu.
- Algoritmus musí umět načíst data ze standartní struktury, zpracovat je s ohledem na vstupní parametry a opět vrátit ve formě standartní struktury.
- Součástí řešení bude ověření funkcí algoritmu zpracováním konkrétních dat z existující organizace.

Dílčí cíle této práce:

- Představit teoretické poznatky ohledně možnosti optimalizace rolí v RBAC modelu, dostupných nástrojů a přístupů, které mohou pomoci při vytváření a optimalizaci rolí.
- Algoritmus bude popsán ve formě administrátorské a uživatelské dokumentace.
- V práci bude ukázán postup algoritmu na vstupních datech a porovnána výsledná data se vstupními.

3. Obsah práce

V první části je představena teoretická část, která popisuje druhy přístupu k řízení přístupu uživatelů, druhy bezpečnostních modelů, předcházejících RBAC modelu a následně je taktéž popsána standardizace RBAC modelu, kterou definovala organizace NIST.

V další části práce je popsán význam tvorby a optimalizace rolí v RBAC modelu společně s přístupy, kterými lze role vytvářet. Následně je představen optimalizační proces, kterým lze přistupovat k optimalizaci rolí strukturovaně a dělí jej na několik samostatných částí. Tento proces definuje pouze teoretický způsob, jakým lze přistupovat k optimalizaci. Aby se tento proces mohl implementovat je třeba převést jednotlivé jeho kroky do programovacího nebo matematického vyjádření. O převedení RBAC modelu do matematického zápisu pojednává Kapitola 8.

V praktické části je pomocí výše zmíněných poznatků navrhnut a zkonstruován algoritmus, který definuje jednotlivé kroky z optimalizačního procesu a spojuje je s mapováním jednotlivých kroků na matematické či programovací řešení. Tento algoritmus umožňuje definovat parametry, které ovlivňují výsledný model.

Samotný algoritmus je popsán jak teoreticky, tak po stránce funkční a následně je taktéž u některých důležitých částí popsána implementace, která je prováděna v programovacím jazyku Java.

V závěrečné části je ukázán průchod algoritmu na ukázkových datech a vysvětleny jednotlivé operace, které algoritmus nad daty provádí. Taktéž je popsán způsob, jakým byl algoritmus testován, společně s výsledky a možnými rozšířenými, které mohou být do algoritmu zapracovány v budoucím výzkumu.

4. Řízení přístupu

Řízení přístupu k datům neboli „Access management (AM)“ je způsob, jakým určujeme, kdo má k daným datům přístup a kdo nikoliv. [1]

Při správě dat pouze pro jednoho uživatele je problematika řízení přístupu triviální. Pokud existuje pouze jeden uživatel, je daná situace lehká. Tento uživatel má přístup ke všem dokumentům, složkám, souborům, zařízením, ke kterým se může připojit.

Daný uživatel může přistupovat ke všemu a není třeba ho nikterak omezovat. Problém ovšem nastává, pokud cílové systémy využívá více uživatelů a je nutno rozhodnout, kterému uživateli patří daný informační zdroj, tedy složka, soubor či dokument.

Pokud sdílí jeden informační prostor více uživatelů, nelze je již nechat, aby měli přístup všichni ke všemu, je nutno rozlišovat a nějakým způsobem kontrolovat, kam který uživatel má přístup a kde mu bude přístup odepřen. Touto obsáhlou problematikou se právě zabývá obor řízení přístup k datům.

Obor řízení přístupu k datům představuje dnes velice citlivou záležitost. Je to z důvodu větší koncentrace dat a výraznému tlaku na zabezpečení a ochranu dat, které jsou dnes korporace a firmy vystaveny.

Organizace a firmy, které pracují s citlivými daty potřebují mít absolutní kontrolu nad tím, který uživatel může dané informace či data číst, zapisovat do nich nebo je měnit. Společně s tím je nutno tyto informace o přístupu někde shromažďovat a auditovat. Je nutné dodržovat základní bezpečnostní pravidlo tzv. „neodmítnutelnost odpovědnosti“, organizace musí být vždy schopna prokázat, kdo přistupoval k datům nebo je modifikoval. Každý uživatel musí používat výhradně svůj vlastní uživatelský účet, který je svázán s jeho identitou.

Daný problém se pak stává o to více složitým, pokud pracujeme s více informačními zdroji, které jsou rozdistribuovány po celé organizaci. Tyto informační zdroje nemusí být pouze na lokálním počítači může se jednat o databáze, tiskárny, síťové prvky, mobilní

zařízení a další. Pro každý tento informační zdroj je nutné mít informace, který uživatel k němu má oprávnění a taktéž tyto informace ověřit.

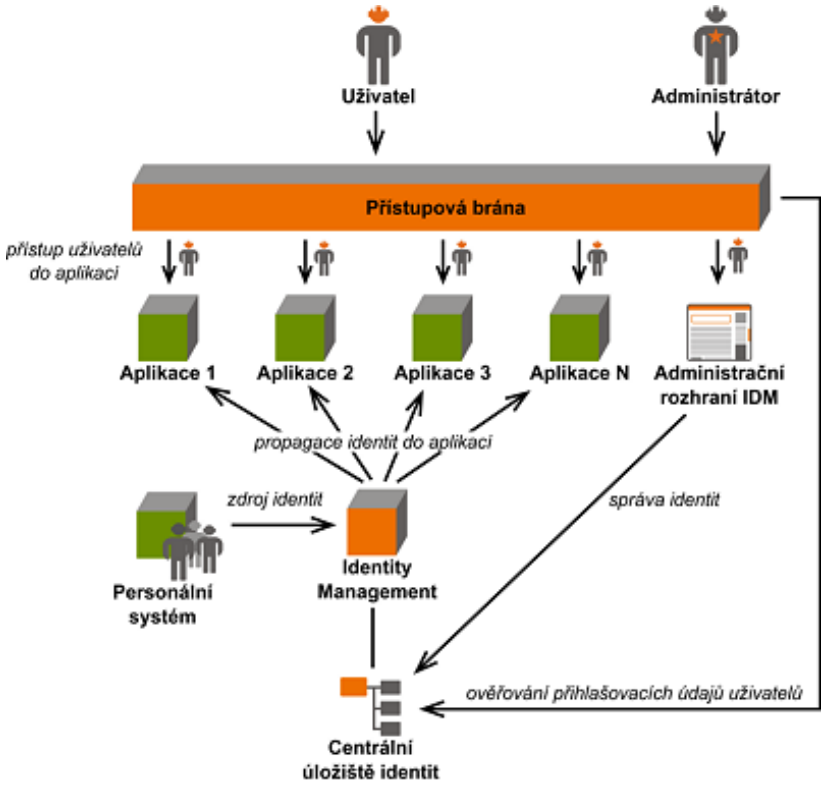
4.1. Kontrola přístupu

V oblasti řízení přístupu k datům je otázka kontroly přístupu nejzákladnější. Musíme mít vždy přehled, kam který uživatel přistupuje a musíme rozhodnout, zda k tomuto přístupu má oprávnění. [2]

Pokud uživatel potřebuje přistupovat k tiskárně, je nutné nejprve ověřit, že k ní máme přístup. Toho lze docílit zavedením uživatele a jeho práv do systému tiskárny, pokud disponuje libovolným systémem pro řízení přístupu. Mnohdy se ovšem jedná o situaci, kdy není možné evidovat takové množství uživatelů či daný informační zdroj nemá v sobě implementován systém řízení přístupu k datům.

V takovém případě je nutno implementovat systém, který se bude starat výhradě o řízení přístupu uživatelů k datům a bude to centrální bod, ze kterého se budou řídit uživatelské účty a jejich oprávnění. Tento systém taktéž musí implementovat způsob, jakým budou dané informace o uživatelích a jejich právech zpropagovány do cílových systémů.

Pokud nelze jednoduchým způsobem daný systém napojit na centrální bod, který v sobě obsahuje informace o uživatelích a jejich oprávněních, využívá se mezivrstvy, které se cílové systémy dotazují, zda daný uživatel má k danému úkonu právo, ta následně tento dotaz vyhodnotí a odpoví jim.



Obrázek 1–Kontrola přístupu. Zdroj: [2]

Pokud již v dané organizaci máme implementovaný některý z IDM řešení (Identity Management), musíme určit, jaký model řízení přístupu k uživatelům a jejich oprávnění zvolíme.

Historicky se využívalo několik různých druhů přístupů. Každý z nich má svoje výhody i nevýhody. V následující kapitole představíme dva nejznámější, které se do dnes používají například v operačním systému nebo v přístupu k utajovaným datům.

Jedná se o Diskrétní kontrolu přístupu (DAC) a Povinnou (Mandatorní) kontrolu přístupu (MAC).

4.2. Druhy přístupů k řízení dat

V předchozí kapitole bylo teoreticky popsáno, jakým způsobem můžeme kontrolovat, kam který uživatel v daném systému přistupuje. Tento teoretický princip je ale potřeba převést do praxe, respektive implementovat ho v daném informačním systému.

4.2.1. Diskrétní přístup (DAC)

Diskrétní přístup k řízení přístupu (Discretionary Access Control) byl vyvinut v roce 1974 a publikován autorem B. W. Lampsonem, který tento model popsal ve své práci ACM SIGOPS Operating System Review.

Jedná se o velice známý koncept přístupu, se kterým se můžeme například setkat v operačním systému Windows nebo Linuxu. Tento přístupový model je založen na identifikaci uživatele, který hodlá přistupovat ke zdroji a následně na základě pravidel je zjištěno, zda k danému zdroji má přístup. [3]

V tomto modelu je uživatel držitel zdroje a může následně umožnit ostatním uživatelům delegovat přístup ke zdroji, případně je přímo učinit vlastníky zdroje. Tímto lze docílit, že jakýkoliv uživatel je následně schopen přidávat oprávnění ke zdroji a taktéž delegovat tato oprávnění na další uživatele bez jakékoliv kontroly.

Samotná implementace tohoto přístupu není náročná. Informační zdroj sám určuje, který uživatel má kam přístup a tyto informace mohou být evidovány přímo na samotném zdroji. Pro určení konkrétního uživatele se pak stačí pouze podívat, zda je on sám nebo jeho skupina má daný přístup povolen. Existuje mnoho druhů implementací. Velmi často se můžeme setkat s takzvanými „Access control listy“, které uchovávají informace o tom, kdo má jaké oprávnění přiděleno. Tento princip lze velmi dobře vidět na Tabulka 1 - Ukázka oprávnění v DAC modelu

Tabulka 1 - Ukázka oprávnění v DAC modelu

	Soubor Z	**Složka X**	**Složka Y**
Uživatel 1	pouze čtení	-	-
Uživatel 2	čtení a zápis	-	-
Uživatel 3	vlastník	čtení	zápis

Uživatel 4	čtení	zápis a mazání	-

Z tohoto systému přidělování práv ale také plynou některá závažná omezení:

a) Uživatel může neomezeně manipulovat se zdroji, pokud je jejich vlastník.

   Uživatel, který se zdrojem může manipulovat, může taktéž určovat kdo k němu má přístup. Může přidat ostatní uživatele, odebrat jim práva, či změnit skupiny, které k tomuto zdroji mají přístup. To vede k tomu, že i uživatel, který dostal ke zdroji přístup omylem, může odebrat ostatní uživatele nebo přidělit přístup uživatelům, kterým on sám chce. Není zde žádná centrální autorita, která by měla na toto přidělení přístup anebo ho mohla ověřit.

b) Programy mají stejná práva jako uživatel, který je spouští.

   Při spuštění přebírají programy či aplikace oprávnění, která má uživatel, který je spustil. Tímto se může velmi snadno stát, že jakýkoliv program, spuštěný pod správcovským účtem, může zasahovat do libovolných dat, protože on sám bude mít oprávnění shodná se správcem. Toto je velmi nebezpečné v případě, kdy je například účet infiltrován virem nebo malwarem.

c) Administrátor nemá kontrolu nad delegováním informací.

   Samotná práva může přidělovat kdokoliv, kdo je nastaven jako vlastník souboru. Pokud tedy uživatel přiřadí jako vlastníka další osobu, nemá už žádnou další kontrolu, co daný uživatel s tímto souborem koná a v ojedinělém případě může i teoreticky ztratit k tomu souboru přístup nebo mu bude přístup omezen uživatelem, kterému sám přístup poskytl.

d) Chybí tu centralizovaná správa.
   Při správě většího množství uživatelů je potřeba centralizovat správu uživatelů, přidělování jejich oprávnění a taktéž odebírání či manipulaci s již vytvořenými

relacemi. V tomto případě žádná centralizovaná správa neexistuje a veškerá oprávnění jsou přidělována ad-hoc na základě uživatelské vůle.

Přes tyto problémy se DAC velmi dobře využívá tam, kde existuje potřeba pro jednoduchý způsob řízení přístupu, který není náročný na implementaci a taktéž na modelování. Opakem decentralizované správy je Mandatorní přístup - tedy MAC.

4.2.2. Mandatorní přístup (MAC)

Tento přístup řízení vychází z opačných předpokladů než diskrétní přístup. Veškerá oprávnění jsou tu řízena centrálně a uživatel nemá žádnou pravomoc delegovat nebo jinak přidělit oprávnění cizímu uživateli.

Samotné přidělování práv se následně řídí bezpečnostními politikami, které nastavení administrátor v systému globálně. Jednotlivý uživatel je následně zatříděn do kategorie nebo stupně pověření, ve kterém se může pohybovat a každý zdroj musí ověřit, že tento stupeň pověření, má právo k němu přistupovat. [3]

Tento způsob řízení přístupu k datům se velmi dobře uplatní například v bezpečnostních složkách, či nemocnicích, kde vyžadujeme určitý stupeň důvěry v toho, kdo přistupuje k citlivým dokumentům, které by mohly být zneužity při neoprávněném přístupu.

Na druhou stranu, pokud bychom chtěli tento systém řízení aplikovat na každodenní operace, vystavovali bychom se téměř nesplnitelnému úkolu kategorizovat všechny zdroje a následně rozhodovat, zda uživatel má daný stupeň prověření, aby k tomuto zdroji mohl přistoupit. Už jen samotné plánování by bylo velice nákladné a znemožňovalo by uživatelům kontinuální pracovní nasazení, pokud by se tento model skutečně implementoval jako základní bezpečnostní model. Proto se tento model používá pouze pro skutečně bezpečnostně citlivé dokumenty a samotné nástroje, jako jsou třeba pracovní stanice, jsou z tohoto systému vyjmuty.

Oba dříve zmíněné přístupy mají určité vady a nedostatky. V prvním případě chybí centralizovaná správa, která by umožňovala řídit přidělování oprávnění uživatelům a zajišťovala nějakou formu auditu. V druhém případě je řízení nastaveno centrálně, ale je

velmi omezující a neumožňuje delegování oprávnění nebo flexibilní přidělování či odebírání přístupu.

Abychom mohli přehledně efektivně řídit uživatele, jejich oprávnění a přístupy, je potřeba tyto přístupy zkombinovat. K tomu došla i vědecká komunita a výsledkem této snahy o kombinaci byl model, který místo přímého přidělování práv využívá mezi vrstvu, která oprávnění seskupuje do logických celků, tedy rolí. Výsledný model dostal název „Role Based Access Control“ zkráceně tedy RBAC model.

5. RBAC Model

RBAC model se vyvinul z předcházejících dvou modelů přístupů k řízení přístupu k datům. Důvodů bylo několik. V první řadě se s velkým rozvojem IT infrastruktury vyvinula potřeba pro centralizovanou správu zdrojů a taktéž přístupu k nim, což žádný z dříve zmíněných modelů nesplňoval.

Velmi značně se taktéž zvýšil počet uživatelů a zdrojů, které bylo potřeba v daném modelu spravovat. Náklady na tyto změny se stávaly velké a upravovat databáze, které čítaly tisícovky záznamů při každé změně oprávnění nebo odebrání přístupu uživatele, se stávalo neudržitelné. [4]

Z těchto důvodů, bylo třeba přijít s novým přístupem řízení, který by dané nedostatky řešil a následně by umožňoval jednodušší centralizovanou správu uživatelů a jejich oprávnění. Tento nový přístup měl zahrnovat možnost flexibilně měnit oprávnění, centrálně auditovat jednotlivé přístupy uživatelů, přidělování jejich práv a taktéž odebírání.

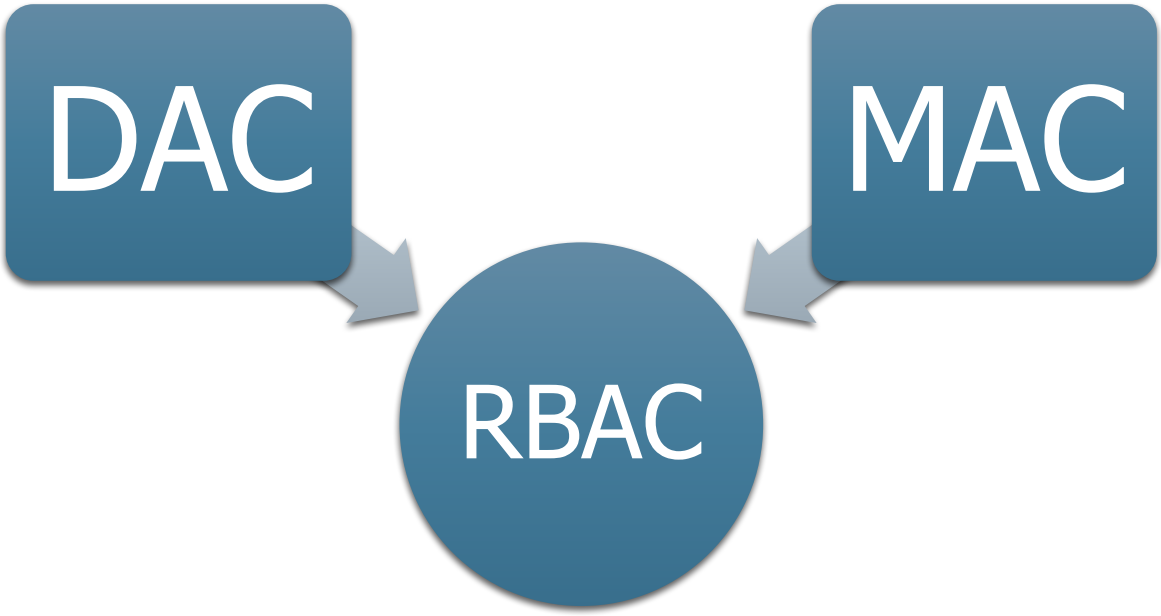
Všechny tyto problémy řešil nový model, který kromě přímého přidělování oprávnění uživatelům taktéž umožnil seskupovat oprávnění do menších celků, které většinou mají logický či technický význam. [3]

Díky přidání těchto skupin, nazývaných role, vznikl nový model řízení přístupu, který se pojmenoval „Role Based Accesss Control“, tedy RBAC model. Tento nový RBAC model těží z mnoha výhod, které předchozí modely MAC a DAC neumožňovaly.

V první řadě se tu jedná o větší stabilitu celého modelu. Uživatelé a jejich oprávnění se mění na téměř denní bázi. Jsou přijímáni další zaměstnanci či externí dodavatelé, kteří potřebují přístup ke zdroji a k systému.

V dřívějších modelech bylo nutné, aby uživatel po zavedení do systému taktéž obdržel všechna oprávnění a bylo to nutné tuto změnu zanést do každého systému, ve kterém měl uživatel mít cílový účet. Díky implementaci RBAC modelu, který přímo počítá s centralizovanou správou, lze všechny tyto změny řídit z jednoho místa.

Přidělování oprávnění přes role je taktéž mnohem výhodnější než přidělování oprávnění přímo uživatelům. Role v RBAC modelu, správně definované, jsou velmi stabilní a mnohem častěji se mění uživatelé než role, což vede k minimalizaci složitosti při zadávání či odebrání uživatele nebo jeho práv.



5.1. NIST RBAC

Od svého vzniku prošel RBAC model velkým vývojem a jeho adaptace se postupně dostala do většiny firem. Mnoho velkých společnostní jako je Siemens, IBM nebo Sysbase začalo vyvíjet vlastní produkty, které implementovaly určitou část RBAC modelu a taktéž ho velmi často rozšiřovaly o vlastní doplňky, které byly závislé na konkrétním modelu užití.

V roce 2000 byl vydán článek „NIST RBAC model“ [5], ve kterém byl představen generizovaný model RBAC modelu, očištěný od implementačních detailů a zaměřující se pouze na samotný teoretický základ.

Následně čtyři roky po vydání tohoto článku byl přijat ANSI/INCITS standard, který definoval RBAC model ve svém současném stavu. Společně s tím organizace „National

Institute of Standards and Technology“, zkráceně NIST, poskytuje rozsáhlou dokumentaci a podporu k standardizovanému modelu RBAC modelu na svých stránkách: http://csrc.nist.gov/groups/SNS/rbac/index.html

Ve zbytku této práce se budu odkazovat na tento standard, jelikož implementace u jednotlivých řešení se mohou lišit ale teoretické prvky, které jsou v tomto standartu obsaženy, mohou být aplikovány univerzálně. Tedy propojení uživatelů, jejich oprávnění a rolí. Krom toho, tento standard definuje několik druhů a omezení RBAC modelu, o kterých bude více informací v následující kapitole.

5.2. Struktura RBAC modelu

Existuje mnoho výkladů a terminologií při popisu RBAC modelu. V této kapitole budou definovány ty nejdůležitější, se kterými budu následně pracovat v dalších částech.

Základní RBAC model se skládá z uživatelů, rolí a oprávnění. Model dále definuje vztahy, které mezi sebou jednotlivé entity mají: UA *(uživatel x role),* PA *(role x oprávnění),* UPA *(uživatel x oprávnění)*, Tyto vztahy definují vazby mezi uživateli, rolemi a oprávněními.

Role

Role je množina oprávnění, která sdílejí nějaký logický kontext. V RBAC modelu se role většinou odvozují od funkce v dané organizaci (asistent), hierarchické pozici (manager) nebo typu vztahu k firmě (externista).

Role mohou mít mezi sebou vztahy. Role, která je podřízená roli nadřazené, říkáme *„juniorní“*. Roli, která je nadřazená jiné roli, označujeme jako *„seniorní“*.

Oprávnění

Oprávnění je povolení k vykonávání akce na určitém objektu. Mezi základní oprávnění, která se přidělují, patří například: oprávnění čtení, zápisu nebo vykonávání *(execute)*.

Objekt

Objektem se rozumí informační zdroj, který buď přijímá nebo zveřejňuje informace. Samotným objektem může být soubor, složka, řádek v tabulce anebo přímo tiskárna, počítač nebo síťový disk.

Uživatel

Uživatel je člověk, který pracuje s daným informačním systémem. Pod pojmem uživatele lze občas taktéž zařadit informační systém či službu nebo program, v závislosti na kontextu.

Identita

Uživatel může mít mnoho identit. Tentýž uživatel může vůči systému vystupovat v několika identitách. Například jako manažer, schvalovatel nebo běžný uživatel systému. Tyto identity následně slouží pro určení práv při konkrétní operaci, kterou si uživatel přeje autorizovat.

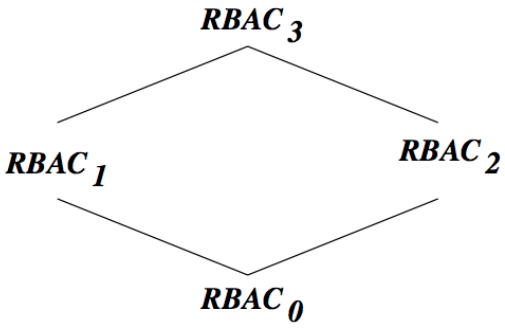
5.3. Druhy RBAC modelu

Organizace NIST v roce 2004 standardizovala RBAC model s několika jeho stupni. Jednotlivé stupně implementují různé druhy RBAC modelu a přidávají logická nebo funkční omezení, která mají zvýšit bezpečnost a oddělit práva rolí.

Dle NIST standardu existují tyto druhy RBAC modelu

a) RBAC 0 – CORE RBAC / FLAT RBAC
b) RBAC 1 – HIEARCHICAL RBAC
c) RBAC 2 – CONSTRAINED RBAC
d) RBAC 3 – SYMETRIC RBAC

Samotné modely nejsou hierarchicky kompatibilní. RBAC 0 je obsažen ve všech dalších stupních ale například RBAC 2 nedědí funkcionalitu z RBAC 1 nýbrž přímo z RBAC 0. Nejvyšší model RBAC 3 je spojení modelu RBAC 1 a RBAC 2.



Obrázek 2 - Struktura RBAC modelu podle úrovní Zdroj: [4]

Všechny tyto modely jsou pouze teoretické a vždy záleží na samotné implementaci, která daná pravidla implementuje podle vlastního uvážení a funkcionality, kterou zrovna potřebuje.

5.3.1. RBAC 0 – CORE / FLAT RBAC

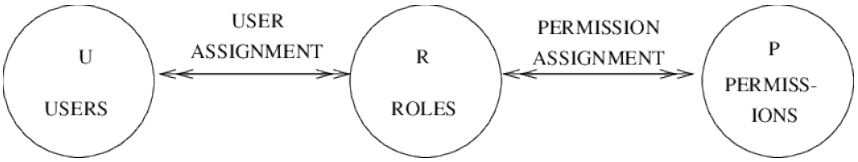
Základní formou definovanou v NIST standardu je RBAC 0 – takzvaný základní či plochý RBAC. Tento model v sobě implementuje základní funkcionalitu RBAC modelu, jak jej známe: uživatele, oprávnění, role a vztahy mezi nimi. Uživatelé následně obdrží přístupy k objektům pouze pomocí rolí, které jsou jim přiděleny.

V tomto modelu jsou definovány tyto entity:

- Uživatelé (U), Oprávnění (P) , Role (R)

Tyto entity mají mezi sebou vzájemné vztahy:

- Role mohou mít přiděleno více oprávnění a jedno oprávnění může být přiděleno více rolím. Vazba M:N. (PA).
- Uživatelé mohou mít přiděleno více rolí a jedna role může být přidělena více uživatelům. Vazba M:N. (UA).

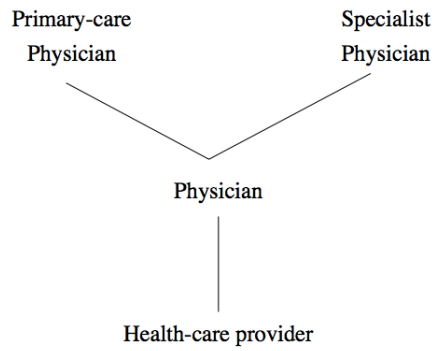


Obrázek 3 - Flat RBAC Zdroj: [1]

5.3.2. RBAC 1 – HIEARCHICAL RBAC

Druhý stupeň RBAC vychází z předchozího modelu RBAC 0 a definuje oproti předchozímu hierarchickou strukturu jednotlivých rolí. Tato vylepšení modelu znamená, že jednotlivé role mohou od sebe dědit oprávnění či s přiřazením role nadřazené uživatel obdrží taktéž všechny role podřazené.

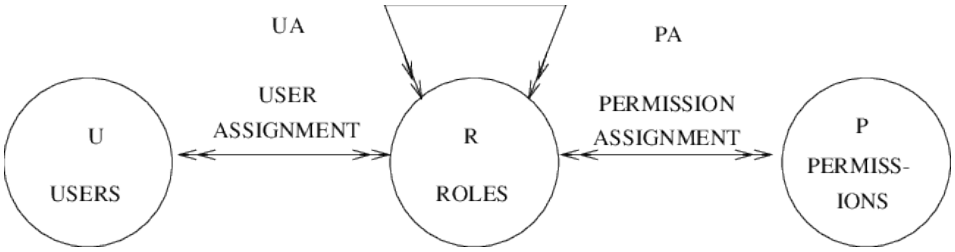
Zavedení hierarchie rolí umožňuje flexibilně definovat organizační strukturu, kde například existuje základní role „zaměstnanec“ ze které následně dědí ostatní role a přidávají k ní nová oprávnění – tyto nadřazené role se nazývají „senioní“ a podřazené role „junioní“.



Obrázek 4 - Ukázka hierarchie rolí Zdroj: [4]

Tento RBAC model definuje novou vazbu *RH (Role Hierarchy)*, která umožňuje rolím vytvářet vztahy mezi sebou:

$$RH \subseteq R \times R$$



Obrázek 5 - RBAC 1 - Hierarchický RBAC Zdroj: [1]

5.3.3. RBAC 2 – CONSTRAINED RBAC

Při vytváření rolí v RBAC modelu se vyskytují případy, kdy nelze přidělit dvě role jednomu uživateli. Ať už se jedná o role navzájem neslučitelné – jako je například „končící zaměstnanec“ a „nastupující zaměstnanec“ nebo o role, které by neměla vykonávat jedna jediná osoba ať už z důvodu logického kontextu nebo bezpečnostní politiky organizace.

Například role „žadatele“ a „schvalovatele“ by měla být oddělená. Nemá logicky smysl, aby o schválení projektu žádal uživatel sám sebe.

Z těchto důvodů RBAC model zavádí omezení, kdy se jednotlivé role navzájem vylučují. Jedná se o takzvaný koncept oddělení povinností – „Separation of Duty (SOD)“. Samotná implementace tohoto omezení závisí na daném prostředí a standard NIST nedefinuje způsob její implementace.

5.3.4. RBAC 3 – SYMETRIC RBAC

Předchozí modely RBAC 1 a RBAC 2 vycházely ze základního modelu RBAC 0. Tento model RBAC 3 spojuje jak hierarchickou strukturu, tak oddělení rolí pomocí principu „Separation of Duty“.

Toto spojení je velmi výhodné, protože umožňuje například limitovat počet rolí, které mohou být „juniorní“ či „seniorní“ pro danou roli. Případně lze například zakázat, aby některá role dědila z role jiné.

Tabulka níže zobrazuje srovnání jednotlivých modelů a jejich základní funkcionalitu.

Tabulka 2 - Přehled druhů RBAC modelu

RBAC 0	CORE / FLAT	• Definuje Uživatele, Role, Oprávnění. • Uživatelé mohou mít jednu či více rolí. • Role může mít jednoho či více uživatelů. • Oprávnění může být přiřazeno do jedné či více rolí. • Uživatelé obdrží oprávnění přes přidělené role.
RBAC 1	HIEARCHICAL	• Definuje hierarchii rolí • Role mohou navzájem dědit svoje oprávnění, případně uživatel pomocí nadřazené role obdrží i všechny role podřazené.
RBAC 2	CONSTRAINED	• Definuje oddělení rolí pomocí principu rozdělení povinností – „Separation of Duty“. • Jeden uživatel nemůže mít dvě navzájem neslučitelné role.
RBAC 3	SYMETRIC	• Slučuje v sobě jak hierarchické členění rolí, tak oddělení povinností a vzájemné vylučování rolí.

6. Vytváření a optimalizace tvorby rolí

Pokud se organizace rozhodne implementovat RBAC model do svého Identity Management systému, musí k tomu mít informační systém, který RBAC model podporuje, ale stejně tak musí sama nadefinovat, jak daný RBAC model bude vypadat.

Implementovaný RBAC model by měl reflektovat strukturu organizace. Jeho účelem není měnit strukturu organizace, nýbrž co nejvíce se jí přiblížit a umožnit kontrolovat přístupy a přidělování oprávnění, které na základě procesů v organizace jednotliví uživatelé vyžadují.

Definice rolí a jejich údržba není triviální. V případě velkého množství rolí musí administrátoři systému udržovat přehled, které role jsou aktuální, které mají být zrušeny, případně které jsou vhodné na změnu. Krom toho musí do systému zavádět nové uživatele, kteří potřebují oprávnění a musí rozhodnout, zda jim tato oprávnění přidělit mimo role, případně zda vytvořit roli novou.

Takové rozhodnutí je důležité, jak z pohledu bezpečnosti organizace – je nutné přesně definovat kam má daný uživatel přístup a která oprávnění už pro něj nejsou potřeba, tak z pohledu uživatelského komfortu. Pokud bude uživateli odebrán přístup ke zdroji či aplikaci, kterou potřebuje k práci, znemožní to výkon jeho činnosti a takovýto přehmat může mít velké důsledky na ekonomiku a operativní schopnosti celé organizace.

Udržování RBAC modelu a jeho správa je kontinuální proces, který v organizaci musí někdo řídit a spravovat. Velkou výzvou v tomto případě je potřeba udržování přehledu, kdo má kam přidělený přístup. Tyto přístupy garantují určité pozice ve firmě, které spravují určité oblasti a rozhodují, které role mohou být přiděleny, případně které již nejsou potřeba. Kupříkladu ve firmě může existovat garant pro přístup k SAP systému, garant práv pro přístup k MSSQL databázi či garant přístupu k interní zákaznické databázi.

Daný uživatel musí mít pouze ty práva, která potřebuje k výkonu práce – princip „*Least privilege*“. Pokud má přidělena práva, která v minulosti potřeboval, ale nyní už tuto pozici opustil, je potřeba mu tyto oprávnění odebrat a zajistit, aby měl přístup pouze tam, kam má mít.

Mezi další pohledy, které bychom měli při tvorbě rolí reflektovat, patří:

- bezpečnost
- efektivita (zavádění uživatele, změny)
- udržovatelnost
- škálovatelnost
- slučování (organizací, systémů)
- administrace (lidský operátor, administrátor systému)

6.1. Přístupy k vytváření rolí

Pro vytváření rolí existují v současné době dva hlavní přístupy. Prvním z nich je přístup „top-down**“**a druhý „bottom-up“.

První metoda „top-down“ je volně přeložitelná jako „od shora dolů“ a využívá se v případě, kdy je pro organizaci možné provést analýzu, s cílem zmapovat jednotlivé procesy, informační zdroje, uživatele a oprávnění, které potřebují k danému přístupu. [5]

Tato metoda má velmi dobré výsledky, pokud máme přístup a přehled nad celkovým obrázkem organizace, procesy jsou precizně definovány a každý uživatel má přesně vymezené pole působností.

Po této analýze se tyto procesy znovu revidují a dle nich se modelují vazby a entity v RBAC modelu. Jedná se o velmi náročný proces, který může trvat velmi dlouho, vyžaduje velké úsilí a může se stát, že i po náročné analýze nebude zcela kompletní. Z těchto výše zmíněných důvodů, se tento přístup nehodí pro organizace, kde procesy nejsou přesně definovány, je jich velké množství a nelze přesně dohledat či vydefinovat, které konkrétní oprávnění má daný uživatel mít.

V případě, kdy nastane tato situace a nelze použít analytický přístup „top-down“ je nutné využít přístup opačný, který nepracuje s analýzou procesů, ale již stávajících vazeb a oprávnění, která jsou v dané organizaci definovány.

Tento opačný přístup se nazývá „bottom-up“ (tedy od „spodu-nahoru“).

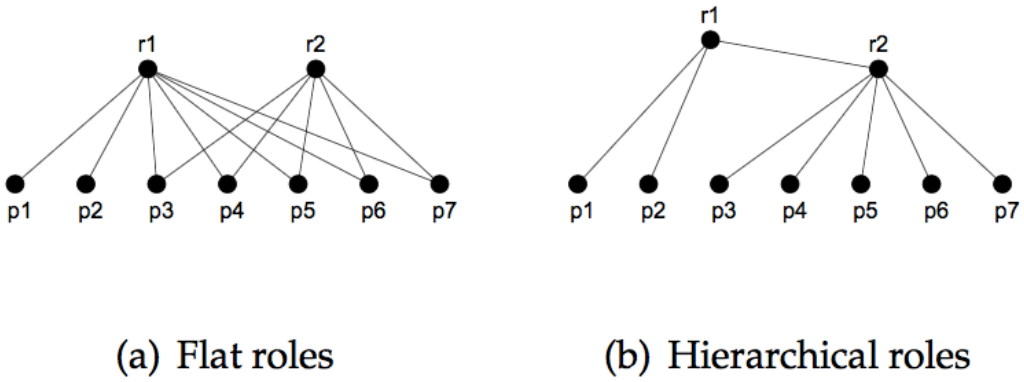
V tomto přístupu pracujeme s daty, která máme již v existujícím informačním systému a pomocí jejich analýzy se snažíme vytvořit možné role, které mohou mít konkrétní uživatelé. Aplikujeme na data statistické metody, případně se snažíme odhalit shluky, které mohou být kandidáty na jednotlivé role. U tohoto přístupu je důležité, že k datům většinou chybí kontext. Pracujeme čistě s exportem nebo extrakcí dat tak, jak jsou uloženy v systému či databázi.

Tento přístup nám může poskytnout velmi cenné vodítko při následné tvorbě rolí. Přesto je však nutné, aby samotné role vytvořil a definoval administrátor či analytik, který má informace o potřebném kontextu celé aplikace.

6.2. Grafová optimalizace

Jednou z metod pro vytváření rolí je grafová optimalizace. Grafová optimalizace se zaměřuje na vyjádření RBAC modelu jako grafu. V tomto grafu prezentují jednotlivá propojení přiřazení uživatelů do rolí, rolí do oprávnění a taktéž tu může být znázorněn vztah mezi hierarchii rolí. [6]

Toto zobrazení lze vidět na obrázku níže:



Obrázek 6 - Reprezentace rolí pomocí grafového vyjádření Zdroj: [6]

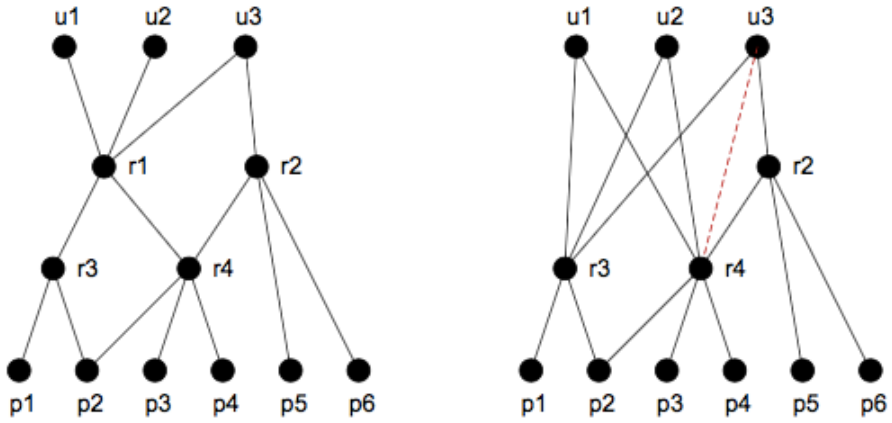
Díky tomuto vyjádření můžeme na RBAC model uplatnit různé grafové algoritmy a následně podle stanovených kritérií spočítat buď nákladnost jednotlivých cest anebo cenu celého grafu. Pokud nás zajímá pouze nejmenší možný počet propojení v celém grafu a nejmenší počet rolí, můžeme graf optimalizovat tak, aby součet jeho rolí a přiřazení rolí x uživatel byl nejmenší.

Cenu takového grafu pak lze vypočítat například takto:

$$cost(G) = c_1|V_r| + c_2|E|$$

Kde c_1 je konstanta, která udává váhu pro roli v grafu a c_2 je konstanta, která udává váhu za uživatele, oprávnění, či role x uživatel přiřazení. Proměnná V_r je počet uzlů, které reprezentují rolí a proměnná E reprezentuje počet hran, které znázorňují přiřazení mezi uživateli a rolemi. V tomto modelu není zahrnuto přiřazení přímého přidělování oprávnění a uživatelů.

Názorná ukázka ceny grafu lze vidět na obrázku níže:



(a) Graph cost = 18. (b) Graph cost = 17.

Obrázek 7 - Ukázka grafové optimalizace Zdroj: [6]

7. Procesní model pro optimalizaci rolí

Samotné přístupy, ať už analytické nebo statistické, v praxi nedokáží většinou konkrétně odpovědět na otázku, které role je potřeba vytvořit, případně které je potřeba upravit a změnit. Tyto přístupy nám mohou posloužit velmi dobře, pokud je začleníme do nějakého rámce a jistým způsobem je společně nakombinujeme.

Touto problematikou se zabývá článek „ROLE MODEL OPTIMIZATION FOR SECURE ROLE-BASED IDENTITY MANAGEMENT“ [7]. V tomto článku autoři analyzují komplexní téma optimalizace tvorby rolí a představují návrh standardizovaného procesního modelu, který slouží k optimalizaci rolí v RBAC modelu.

Jejich motivací je vytvořit rámec pro organizace, které mají velké množství uživatelů a mění, či přidávají role na denní bázi. Pro tyto organizace je velmi důležité, mít definovaný postup jakým způsobem tyto role vytvářet a jak postupovat. Svou definicí Procesního modelu pro optimalizaci rolí „ROPM“ adresují tyto požadavky a umožňují implementovat tento proces flexibilně pro vlastní potřeby organizace.

Samotný ROPM definuje tyto fáze procesního cyklu:

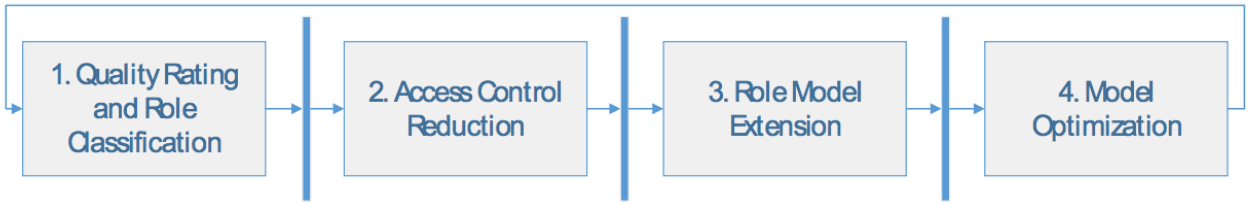


Figure 2: The Role Optimization Process Model

Obrázek 8 - ROPM jednotlivé fáze Zdroj: [7]

Na obrázku výše můžete vidět jednotlivé fáze. Tyto fáze lze charakterizovat takto:

1. Hodnocení a klasifikace

2. Redukce přístupu
3. Definice nových vztahů
4. Optimalizace modelu.

Velkou výhodou tohoto procesního modelu oproti ostatním je jeho možnost přizpůsobení. Každá fáze má několik vnitřních sub-procesů, které jdou uzpůsobit konkrétní situace organizace a jejím potřebám. Autoři v článku teoreticky definují sub-procesy u každé fáze a taktéž nástroje, kterým lze tyto procesy podpořit.

7.1. Fáze procesu

Hodnocení a klasifikace

První fáze má za cíl zhodnotit a klasifikovat jednotlivé role, které následně budou v dalším modelu podrobeny analýze, zda se mají zachovat či mají být změněny.

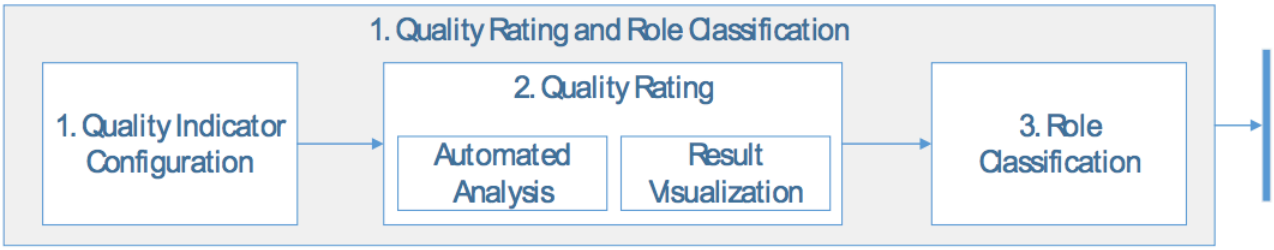


Figure 3: Quality Rating and Role Classification

Obrázek 9 - Hodnocení a klasifikace Zdroj: [7]

Tento proces umožňuje v prvním sub-procesu definici parametrů. Tuto definice většinou provádí administrátor a může definovat druhy, jakými lze role klasifikovat a pomocí business kontextu roztřídit. Následně pomocí statistických nástrojů lze provést analýzu a pomocí vizuálního nástroje nebo jiného vytvořit ke každé roli hodnocení. V samotném závěru je opět potřeba vzít ohodnocené role a nějakým způsobem je přiřadit do skupin. Tyto skupiny se následně mohou zachovat, případně být znovu vymodelovány.

Redukce přístupu

Druhá fáze navazují na první, která měla připravit role jejich klasifikací a jednotlivě je ohodnotit. Následně se v této fázi snažíme identifikovat nepoužívané role a duplicitní vztahy, které mohou být zrušeny.

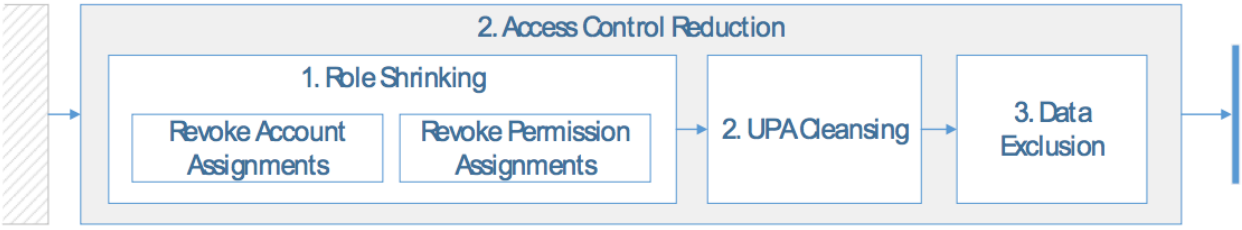


Figure 5: Access Control Reduction

Obrázek 10 - Redukce přístupu Zdroj: [7]

Tento přístup je velmi užitečný, protože zbavuje systém nepotřebných vazeb. Autoři ve svém článku uvádějí například přístup pomocí analýzy historických dat a logů, které mohou poskytnout vodítko, které role nebo oprávnění nebyla nikdy použita nebo se používají velice málo.

Pro tuto operaci taktéž využívají svůj nástroj, který dokáže určitým způsobem vizualizovat role a zobrazovat přesahy, které následně mohou opraveny. V článku je taktéž zmíněné, že tyto jednotlivé operace je potřeba vždy konzultovat s administrátory a následně pro každé rozhodnutí shromáždit dostatečné podklady.

Definice nových vztahů a optimalizace

Po očištění stávajících vazeb je potřeba znovu ověřit, že všechny role odpovídají a případně zda mohou být rozšíření nebo spojeny. Pro tento účel slouží třetí fáze, která se stará o možnou definici nových rolí ze stávajícího modelu.

Autoři ve svém článku zmiňují, že proto fázi lze využít vizualizační techniky, které mohou ukázat jednotlivé role, které se mohou spojit nebo vytvořit role nové. Ve svém článku taktéž popisují nástroj, takzvaný Access Grid, který se velmi podobná Excelu a umožňuje zobrazit jednotlivé přiřazení role a uživatele a následně ho barevně odlišit.

8. Vyhledání nejmenší množiny rolí

V předchozích kapitolách byl popsán základ teoretického RBAC modelu tak, jak jej formuluje organizace NIST. Tento model popisuje RBAC z jeho konceptuální roviny a popisuje ho formou jednotlivých entit a vztahů mezi nimi.

Na tento přístup lze navázat díky výzkumu publikovaného pod názvem „The Role Mining Problem: Finding a Minimal Descriptive Set of Roles“ [8]. Tento výzkum se zabývá způsobem, jak definovat minimální množinu rolí, která plně popisuje aktuální RBAC model. Tato množina minimálních rolí může sloužit při optimalizaci jako minimální možný základ, který musí být definován, aby zůstaly vztahy v RBAC modelu bez ztráty.

Autoři této práce pracují s tím, že při vytváření rolí jsme velmi omezeni, pokud nemáme kontextové informace k daným datům a nemůžeme určit, které přiřazení mají jaký logický kontext. Přesto ve většině případů skutečně nejsou k dispozici analýzy nebo interní informace, které by nám mohly sloužit jako vodítko. Právě proto je nutné nalézt kandidátní role, které jsme schopni nalézt pouze ze samotných dat, která už v systému existují a mohou nám poskytnout hrubou představu, jaké role ve výsledném modelu můžeme používat. Tímto způsobem můžeme bez jakýchkoliv interních informací popsat aktuální stav v dané organizaci a z tohoto stavu dále vycházet.

Aby tohoto mohli docílit, pohlížejí na RBAC model jako na matematickou reprezentaci množin a ve své práci dokazují, že na tento problém lze použít algoritmy, které již byly definovány a používají se kupříkladu v dolování dat nebo v databázích – *„database tiling“* a *„discrete basis problem“*.

Jejich argumentace vychází z toho, že úloha nalezení minimálního počtu rolí, které popisují daný RBAC model, je shodná s úlohou, kdy v databázovém systému rozkládáme tabulky do menších celků, například pomocí normalizace a jde nám o co největší zachování dat a vazeb.

8.1. Definice problému

Ve své práci definují RBAC pomocí těchto vazeb, které jsou více zaměřené na reálné použití a implementaci RBAC modelu v praxi. Tuto definici budu používat i v popisování funkce algoritmu, v následujících kapitolách.

Definice rolí, uživatelů, operací a objektů:

$$U(uživatelé), ROLES(role), OPS(operace), a\ OBJ\ (objekty)$$

Definice oprávnění:

$$PRMS\ (oprávnění) \subseteq \{(op, obj)|op \in OPS\ obj \in OBJ\}$$

Definice vztahu mezi uživatelem a rolemi:

$$UA \subseteq U \times ROLES$$

Definice vztahu mezi rolemi a oprávněními:

$$PA \subseteq ROLES \times PRMS$$

Definice vztahů mezi uživatelem a oprávněními:

$$UPA \subseteq U \times PRMS$$

Samotný problém hledání minimálního počtu rolí, definují takto:

> *„Definition 5 (Role Mining Problem (RMP)). Given a set of users U, a set of permissions PRMS, and a user- permission assignment UPA, find a set of roles, ROLES, a user-to-role assignment UA, and a role-to-permission as- signment PA 0-consistent with UPA and minimizing the number of roles, k.“* [8]

Tato definice říká, že za dané množiny uživatelů U, množiny *oprávnění (PRMS)* a matice přiřazení *Uživatel x Oprávnění(UPA),* chceme najít množinu *rolí (ROLES)*a přiřazení *Uživatel x Role (UA),* a *Role x Oprávnění (PA)*, které přesně a kompletně popisují matici *Uživatel x Oprávnění (UPA)*a zároveň minimalizují počet rolí, *k.*

V jednoduché řeči hledám nejmenší počet rolí, které dokáží popsat námi definované vztahy Uživatelé x Oprávnění.

8.2. Navrhované varianty

Tuto definici autoři ve své práci dále rozvádějí a definují různé varianty tohoto problému.

První variantou je *„δ-approx Role Mining Problem“*. Tedy jak už je z názvu patrné, jedná se zde o určitou aproximaci, tedy z nepřesnění daného výsledku, kdy jsme ochotni obětovat některé vztahy, například 98 % pokrytí za cenu toho, že dosáhneme lepšího výsledku při vyhledávání rolí.

Definice:

> *„Definition 6 (δ-approx RMP). Given a set of users U, a set of permissions PRMS, a user-permission assign- ment UPA, and a threshold δ, find a set of roles, ROLES, a user-to-role assignment UA, and a role-to-permission as- signment PA, δ-consistent with UPA and minimizing the number of roles, k.“* [8]

Druhou variantou je *„Minimal-Noise Role Mining Problem“*. V tomto případě máme definován počet rolí, které chceme najít a samotné hledání se soustřeďuje na nalezení daného počtu rolí.

V této práci se budu zabývat pouze variantou, kde existuje absolutní (100%) kompatibilita. Je to z důvodu, že chceme zachovat veškeré vztahy a přiřazení, které jsme dostali na vstupu. Jak už bylo zmíněné výše, představovalo by velké bezpečnostní riziko přidat či odebrat uživateli oprávnění při optimalizaci.

	r_1	r_2	r_3
u_1	0	0	1
u_2	1	0	1
u_3	0	1	1
u_4	1	0	0

(a) User-role assignment

	p_1	p_2	p_3	p_4	p_5
r_1	1	1	1	0	0
r_2	1	1	0	1	0
r_3	0	1	0	0	1

(b) Role-permission assignment

Table 2: Basic Role Mining Problem

Obrázek 11 - Basic Role Mining Problem [8]

8.3. Binární matice a RBAC model

Jak už bylo popsáno v předchozích kapitolách, RBAC model se skládá z rolí, uživatelů a oprávnění. Tyto entity mají mezi sebou vztahy, UA *(uživatel x role),*PA *(role x oprávnění),*UPA *(uživatel x oprávnění)*, které definují vazby mezi uživateli, rolemi a oprávněními. Samotná role je taktéž pouze skupina oprávnění, která jsou logicky spojena do ucelené entity.

Pro skutečnou implementaci RBAC modelu se velmi často využívají relační databáze, jelikož vztahy mezi uživateli, rolemi a oprávněními se dají velice dobře vyjádřit v relačním světě pomocí tabulkového zápisu.

Tento zápis ve své publikaci hojně využívají autoři článku a představují zobrazení jednotlivých relací, pomocí takzvané *„binárního matice“ (Binary matrix).* [8]

Pokud tedy potřebujeme vyjádřit tyto tři vztahy, lze je vyjádřit pomocí binárního maticového zápisu. Tento zápis lze definovat jako matici $m \times n$,kde m je počet řádků, v našem případě uživatelů a n počet sloupců, které udávají počet oprávnění, která mohou

být přidělena. Samotná hodnota definovaná v řádku $\{ij\}$ určuje, zda je toto oprávnění přiděleno uživateli.

Samotné přiřazení oprávnění uživateli lze vyjádřit hodnotou 0 či 1 v daném řádku a sloupci – toto vyjádření lze vidět na Obrázek 11, případně pomocí logické definice *true* nebo *false,* tedy vyplněné hodnoty nebo prázdné – toto zobrazení je vidět na Obrázek 12. Toto zobrazení slouží pro jednodušší abstrakci a vizualizaci.

Díky tomuto vyjádření lze konstatovat, že na jednotlivé relace vyjádřené v binární matici lze aplikovat matematické metody. Mezi tyto metody můžeme zařadit shlukovou analýzu, vyhledávání vyplněných částí, K-Means a mnoho dalších. Vždy je ovšem nutno brát v potaz jednotlivá kritéria, která platí pro RBAC model, jako například vícenásobné přiřazení role uživateli, vnoření role a další, které kupříkladu u shlukové analýzy velmi snižují efektivitu.

	P1	P2	P3	P4	P5	P6
U1	x	x	x	x	x	
U2	x	x	x	x	x	
U3	x	x				
U4		x	x		x	x
U5		x	x		x	x
U6		x	x	x	x	x
U7	x		x	x		

Obrázek 12 - Vyjádření RBAC modelu v maticovém zobrazení

Sami autoři ve své práci přímo zmiňují, že problém s hledáním minimálního počtu rolí je přímo totožný s problémem oblasti dekompozice relačních databází. Konkrétně s algoritmem na dělení databází na nejmenší části – anglicky „*minimum tailing*“.

Tento algoritmus má jistá omezení a v praxi velmi závisí na tom, jakým způsobem data uspořádáme. Ve svém výzkumu následně představují teoretický zápis algoritmu, který může nalézt z aktuálních oprávnění, která jsou přiřazena uživateli, minimální množinu, která odpovídá jejich požadavkům, aby zpětná rekonstrukce odpovídala původnímu vstupu. Je nutno dodat, že tento algoritmus je pouze teoretický, nejedná se o jeho

implementaci na reálná data. Tato implementace by se patrně potýkala s mnoha různými problémy, které by bylo nutno řešit.

Přesto tato práce velmi zevrubně mapuje a následně demonstruje, že RBAC model může být velmi dobře znázorněn pomocí matematického zápisu, což otvírá velmi mnoho možností pro automatizaci jednotlivých částí, které jsou potřeba při optimalizaci rolí a oprávnění přiřazených uživateli.

Kromě jiného taktéž zdůrazňují nutnost při jakékoliv strojové analýze a zpracování dat z RBAC modelu brát v potaz, že se vždy jedná pouze o holá data, která nemají veškerý kontext.

Z tohoto důvodu je nutno veškeré výsledné změny konzultovat s administrátorem, který má informace a poznatky k tomu, aby dokázal rozhodnout, které role jsou schopny v daném prostředí být definovány a které jsou správné. Samotný automatizovaný algoritmus má sloužit pouze jako nástroj, který umožňuje získat kandidátní role, které následně daný uživatel bude schopen dále definovat.

9. Algoritmus pro optimalizaci tvorby rolí v RBAC modelu

V předcházejících kapitolách byl představen teoretický základ pro mapování RBAC modelu na matematické operace a strojové zpracování – v kapitole 8. Vyhledání nejmenší množiny rolí. Společně s tím byl představen popis Procesní model pro optimalizaci rolí, který se zabývá procesním postupem při optimalizaci tvorby rolí.

Kombinací těchto dvou teoretických poznatků a jejich implementací na reálná data bylo možné vytvořit algoritmus, který v sobě implementuje oba přístupy a z reálných dat je schopen vytvořit optimalizovanou množinu rolí, která odpovídá vstupním parametrům a udržuje si 100 % kompatibilitu se vstupní množinou.

Je třeba mít na paměti, že algoritmus vychází z přístupu dříve popsaného jako „bottom-up“. Tento přístup mu umožňuje zpracovávat data bez jakékoliv znalosti dřívějšího kontextu a vazeb, které jsou definovány mimo samotný IDM systém. Tato skutečnost ovšem znamená, že výstup algoritmu musí být vždy konzultován s administrátorem nebo pracovníkem, který má k rolím tak potřebný kontext.

Následující kapitoly se budou zabývat definicí jeho jednotlivých součástí společně s popisem jeho implementačních funkcí a teoretického postupu.

9.1. Struktura algoritmu

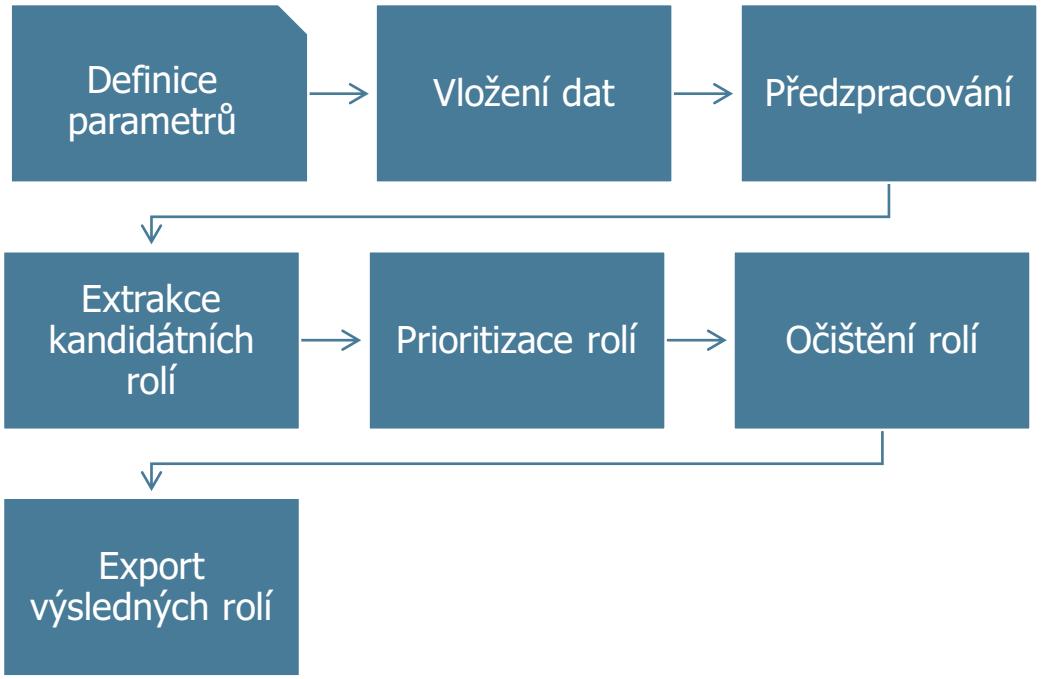
Definovaná struktura algoritmu reflektuje jednotlivé fáze dříve zmíněné v optimalizačním procesu. Přesto z praktických důvodů jsou některé fáze částečně odebrány případně rozšířeny, aby odpovídaly konkrétnímu druhu implementace na testovací data.

Strukturu algoritmu lze rozdělit do několika kroků:

- Definice parametrů
- Vložení dat

- Předzpracování
- Extrakce kandidátních rolí
- Prioritizace rolí
- Očištění rolí
- Export výsledných rolí

V následujících kapitolách budou jednotlivé kroky rozebrány a bude popsán jejich účel společně s implementací, kterou tento algoritmus poskytuje.



Obrázek 13 - Diagram kroků algoritmu

9.2. Definice parametrů

Administrátor při činnosti vytváření nebo optimalizaci tvorby rolí musí mít kritéria, která jsou vhodná pro danou situaci, ve které aplikuje RBAC model.

Tyto zadané parametry určují fungování daného algoritmu a ovlivňují výsledný výstup včetně nalezených rolí. Řídí se dle nich extrakce rolí, jejich prioritizace a jejich očištění.

Maximální velikost role

Tento parametr určuje, jaká maximální možná velikost role bude ve výsledném RBAC modelu. Velikost role je určena jako počet oprávnění, která má tato role přiřazena. Algoritmus se při svém průchodu bude řídit tímto parametrem a bude ho používat jako horní hranici, kam až vyhledávat možné kombinace, které by následně mohly být označeny jako kandidátní role.

U tohoto parametru je nutné zohlednit, že jeho hodnota ovlivňuje náročnost algoritmus. Čím větší je velikost maximální role, tím více kombinací a kandidátních rolí musí algoritmus projít a očistit. Díky tomu se jeho doba trvání může radikálně prodloužit.

V config souboru se definuje tento parametr pomocí klíče:

„max_role_size = 12“

Minimální velikost role

Tento parametr určuje minimální velikost role. Je to hranice, jaké nejmenší role se budou vytvářet ve výsledném RBAC modelu. Minimální velikost role je použita k předzpracování dat. Algoritmus nehledá role menší, než je daný parametr, a proto se role menší, než je tento parametr negenerují a ani nezařazují k porovnání s ostatními, a nemohou proto být ve výstupu tohoto algoritmu.

Při definici tohoto parametru je nutno brát v potaz, že hodnota nastavená na velmi nízkou hodnotu může znamenat i velmi malé role, které následně budou fragmentovat RBAC model. Hodnota nastavená na moc vysoko může znamenat, že se některé role, které by mohly existovat v RBAC modelu, nepodaří nalézt a nebudou zařazeny do výstupu.

V config souboru se definuje tento parametr pomocí klíče:

„min_role_size = 6“

Optimální velikost role

Předchozí dva parametry – „Minimální velikost role“ a „Maximální velikost role“ určují hranice, kde všude má algoritmus hledat výsledné role.

Z důvodu lepších výsledků bylo potřeba definovat i velikost, která se pro danou situaci jeví jako optimální a kolem které by výsledné role měly oscilovat.

Jak už z jeho názvu vyplývá, tento parametr určuje, jaká velikost role je optimální – v daném kontextu. Při průchodu algoritmu se tato hodnota využívá pro porovnání velikostí rolí a pomocí absolutní hodnoty rozdílu se prioritizují role, které mají blíže k tomuto parametru pro optimální velikost role.

Samotný parametr je možno odvodit například z průměrné velikosti role nebo mediánu, který vypovídá kolik je „normální“ velikost role v RBAC modelu.

V config souboru se definuje tento parametr pomocí klíče:

„optimal_role_size = 8“

Minimální počet uživatelů pro roli

Pomocí předchozích parametrů se dalo definovat rozmezí, ve kterém se mají pohybovat rozměry výsledné role. Kromě této velikosti algoritmus taktéž umožňuje definovat parametr, který nepracuje s velikostí role, nýbrž počtem uživatelů, kteří mají tuto roli přiřazenou.

Tímto parametrem lze zamezit tomu, aby měla daná role počet uživatelů menší, než je minimální velikost definovaná administrátorem. Při moc nízké hodnotě nebudou nalezené role optimální a může se stát, že budou existovat role pouze pro jednoho či dva uživatele, což není správně.

V config souboru se definuje tento parametr pomocí klíče:

„min_users_for_role = 5“

9.3. Definice, vložení a tvorba dat

Při tvorbě algoritmu bylo bráno v potaz, že existují možnosti, kdy již pracujeme s existujícím RBAC model, ve kterém role jsou definovány, ale z nějakého důvodu je nepovažujeme za optimální, případně s čistou implementací RBAC modelu, který implementujeme do nového systému nebo nějakou kombinací těchto dvou variant.

Algoritmus umožňuje využívat data z již existujícího RBAC modelu, stejně tak jako z jednoduché implementace, například DAC modelu nebo nějaké jeho alternativy.

Pro svůj vstup předpokládá algoritmus vložení dat, která definují pouze vztah uživatel a jeho oprávnění. Tyto data musíme mít definovány, pokud chceme určit, zda daný uživatel má přístup k danému zdroji a lze je nějakým způsobem extrahovat z daného systému - ať už využívá RBAC, MAC, DAC případně jiný systém pro řízení přístupu.

9.3.1. Tvorba dat z existujícího RBAC modelu

V případě existujícího RBAC modelu potřebujeme získat data, která máme uložena v některém reálném systému LDAP nebo relační databázi. Implementace RBAC modelu se může v praxi lišit, může mít jiné atributy nebo upravený model, který více vyhovuje dané situaci, ve které se pohybujeme.

Pro lepší představu budeme vycházet z toho, že definovaný RBAC model pracuje s relační databází a má definovány tyto tabulky:

- users, permissions, roles

Jednotlivé tabulky obsahují záznamy o uživatelích – tabulka „users“, následně o oprávněních, která je možno přidělit – tabulka „permissions“ a informace o rolích, které existují – tabulka „roles“.

Vztahy mezi jednotlivými entitami – uživatelé x oprávnění, uživatelé x role a role x oprávnění jsou definovány v těchto tabulkách:

- user_permission, user_role, role_permission

První tabulka „user_permission“ v sobě udržuje vztah definovaný v teoretickém modelu RBAC jako:

$$UPA \subseteq U \times PRMS$$

Tato tabulka slouží k udržování přehledu o uživatelích a jejich přidělených oprávnění, které jsou přiděleny mimo role. O této tabulce můžeme taktéž mluvit jako o tabulce „Uživatel x Oprávnění“.

Druhá tabulka „user_role“ je v teoretickém modelu RBAC definována takto:

$$UA \subseteq U \times ROLES$$

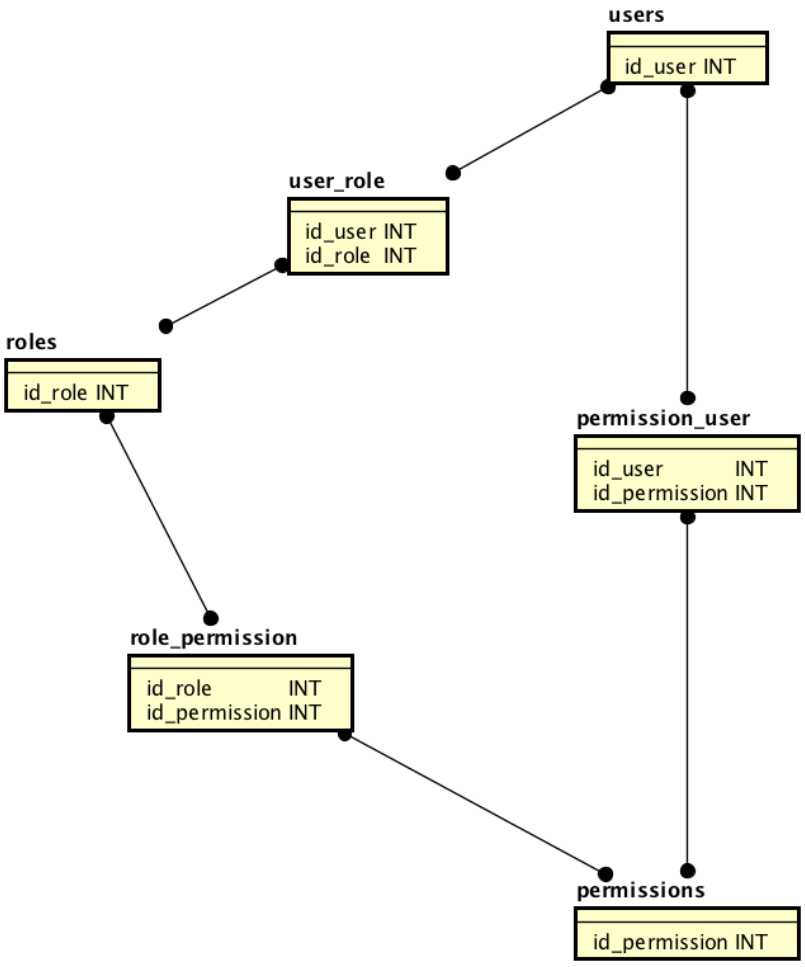
Tato tabulka slouží k definici přiřazení role k uživateli. Uživatel může mít více rolí, stejně jako jedna role může být přiřazena k více uživatelům. Tato tabulka se taktéž dá pojmenovat jako „Uživatel x Role“.

Třetí tabulka „role_permission“ je v teoretickém modelu RBAC definována takto:

$$PA \subseteq ROLES \times PRMS$$

V této tabulce se definují vztahy mezi rolemi a oprávněními, která jsou v rolích přiřazena. Stejně jako v předchozích případech je zde vztah M : N, tedy že role může mít přiřazeno více oprávnění a jedno oprávnění lze přiřadit do více rolí.

Celkový obrázek ukázkové teoretické a zjednodušené implementace RBAC modelu v relační databázi zobrazuje Obrázek 14.



Obrázek 14 - Class diagram ukázkové implementace RBAC modelu v relační databázi

Pro svůj vstup předpokládá algoritmus vložení dat ve formátu „Uživatel x Oprávnění“. Pokud již systém, ze kterého budeme používat data, implementuje RBAC model, musíme nejprve rozložit role na unikátní relace „Uživatel x Oprávnění“ a ty následně spojit s právy, která jsou přidělována uživatelům přímo.

Dle výše zmíněného teoretického případu implementace RBAC modelu v relační databázi postupujeme takto, abychom získali data, která následně potřebujeme pro vstup do algoritmu.

1. Spojíme tabulku „user_role“ s tabulkou „role_permission“.
2. Následně tuto výslednou tabulku spojíme s tabulkou „user_permission“.

V teoretickém zápisu množin tento úkon definujeme takto:

$$(UA \cup PA) \cup UPA$$

Při praktickém případě by se jednalo o zápis v SQL pomocí tohoto výrazu:

$$SELECT\ t1.id\_user, t2.id\_permission\ FROM\ user\_role\ as\ t1$$

$$JOIN\ permission\_role\ as\ t2\ ON\ t1.id\_role\ =\ t2.id\_role$$

$$UNION$$

$$SELECT\ id\_user, id\_permission\ FROM\ user\_permission;$$

V případě, že RBAC model není ještě implementován a chystáme se vytvářet role nově, musíme ze současného systému vyextrahovat uživatele a jejich oprávnění. Samotný způsob extrakce závisí na daném systému a jeho způsobu ukládání oprávnění.

9.3.2. Struktura vstupních dat

Aktuální implementace algoritmu v sobě zahrnuje možnost importu dat pomocí standardizovaného způsobu.

Tento způsob zahrnuje předání csv souboru, který obsahuje data pro zpracování. Algoritmus tyto data očekává na vstupu a pokud mu nebudou předána, tak algoritmus nemůže dále pokračovat.

Vstupní data jsou zapsána jako páry (uživatel : oprávnění) a oddělená pomocí středníku. Oba parametry, tedy uživatel i oprávnění, jsou identifikovány unikátně jako jejich ID či jiný identifikátor, který by měl být unikátní pro danou množinu dat.

Formát vstupních dat vypadá takto:

id_user; id_permission;

Ukázka vstupních dat lze vidět na *Tabulka 3 - Ukázka formátu vstupních dat*.

U vstupních dat se předpokládá přeskočení prvního řádku – z důvodu definice sloupců. Stejně tak je nutné data před prvním vložením očistit. Algoritmus není koncipován na

data, která jsou nekonzistentní nebo neúplná. Při nevalidních datech algoritmus tyto záznamy přeskočí a nebude je zpracovávat.

Tabulka 3 - Ukázka formátu vstupních dat

id_user	id_permission
18	4172
18	4162
18	5647
18	6931
18	7036
18	2491
18	5785
18	2982
18	2759
18	2294

9.3.3. Načítání vstupních dat

Samotná třída Source.java slouží k tomu, aby transformovala vstupní data a přiřadila je do logického formátu, který se dále bude v aplikaci používat. Krom toho, instance této třídy vstupuje jako parametr ostatním, které očekávají, že ze zdroje budou moci číst data a ty následně transformovat.

Při načtení dat se volá tato metoda:

$$getDataSet(csvFileName:\ String) : ArrayList < User >$$

Tato metoda obsluhuje načtení dat z csv souboru a vrací list uživatelů. Při načítání dat se ověřuje, že definovaný soubor existuje, pokud nebude definován či nebude zapisovatelný, aplikace vrátí výjimku.

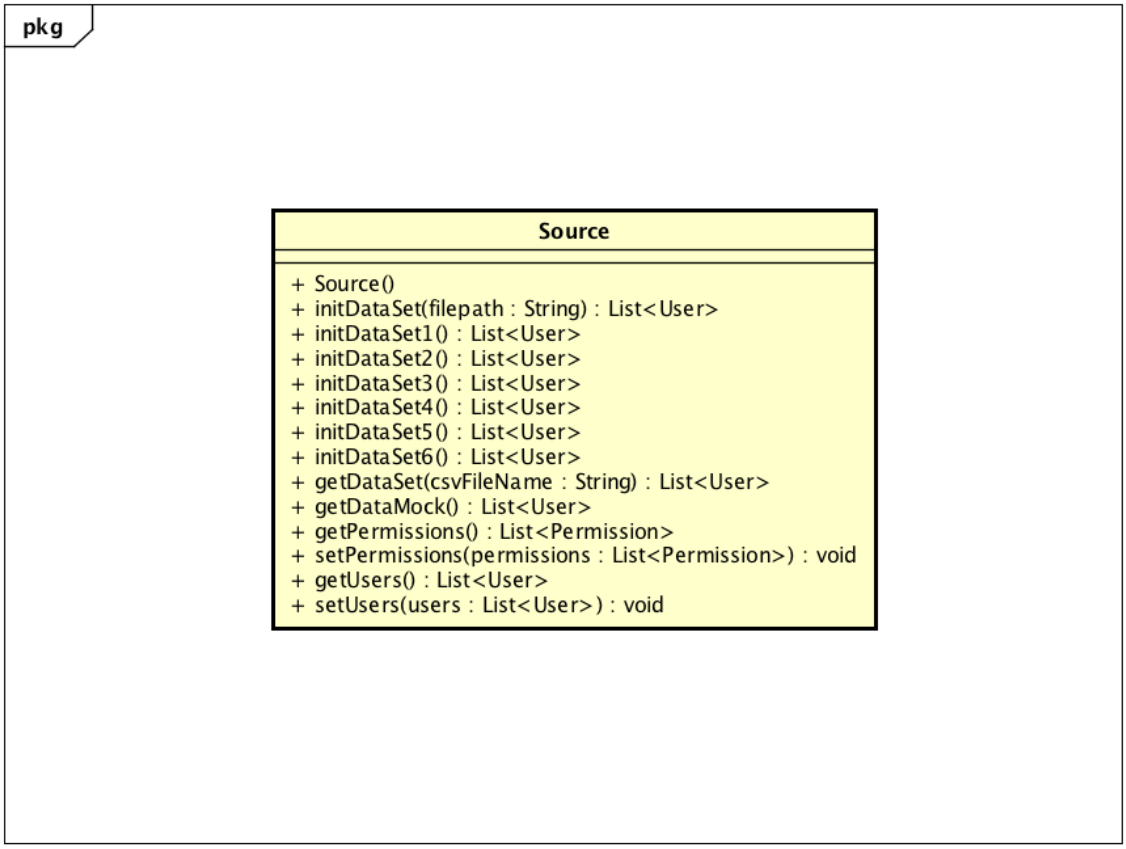
Struktura třídy Source.java je dobře patrná z Obrázek 15. Z této třídy lze získat data pomocí metody

$$getPermissions(\ \ ) : ArrayList < Permission >$$

Tato metoda vrátí list všech oprávnění. Všechna tato oprávnění mají definovány základní atributy společně s listem uživatelů, který tyto oprávnění vlastní.

Druhá metoda vrací pole uživatelů, které se vyskytují ve vstupním data-setu. Každý uživatel má taktéž definován list, ve kterém jsou uvedena oprávnění, které má tento uživatel přiřazena.

$$getUsers(\ \ ) : ArrayList < User >$$



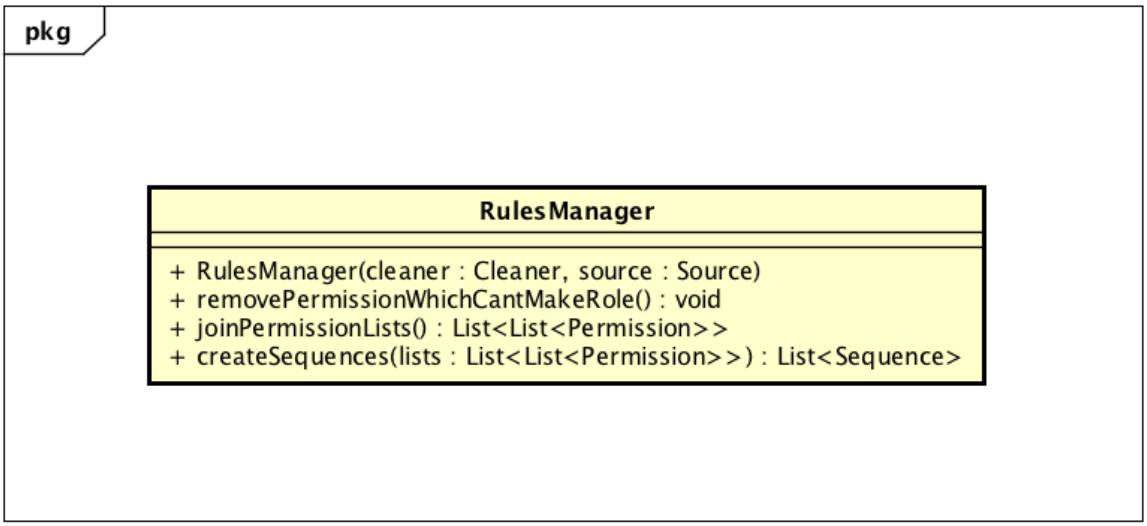
Obrázek 15- Class diagram pro třídu Source

9.4. Předzpracování dat

Po zadání dat do programu a definici parametrů přichází na řadu předzpracování dat. Samotné předzpracování dat je důležité k tomu, aby se data očistila od těch, která nemají při dalším kroku smysl a pouze by zatěžovala daný výpočet.

Tuto část, ve které jsou data předzpracována, lze velmi jednoduše rozšířit. Přímo v programu lze definovat další metody, které budou upravovat data a případně je i čistit nebo mazat. Tyto metody, nebo lépe řečeno pravidla, jsou definovány v třídě, která se jmenuje RulesManager.java

Class diagram této třídy lze vidět na obrázku níže:



Obrázek 16 - Class diagram třídy RulesManager.java

V současné době je v algoritmu implementováno několik způsobů předzpracování dat.

9.4.1. Odebrání oprávnění

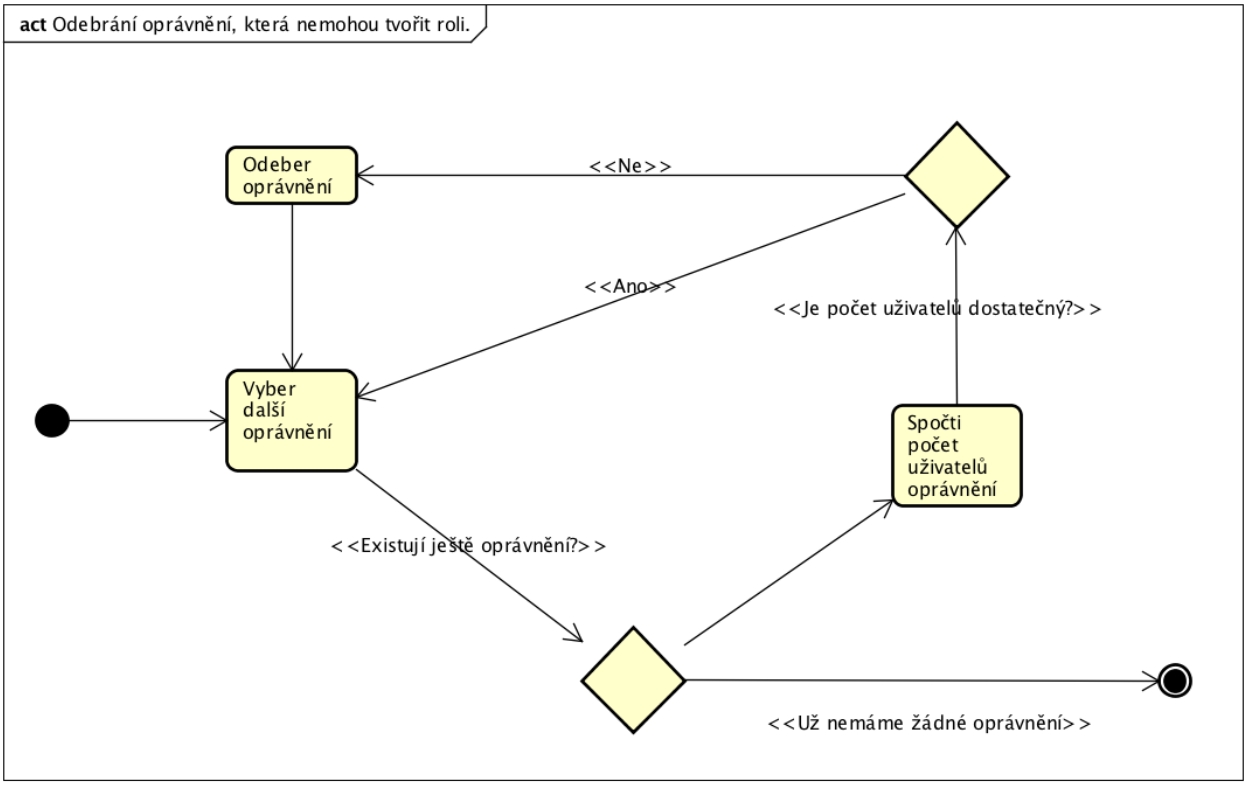
V první řadě je to odebrání oprávnění, která budou následně použita pro generování kandidátních rolí. Kandidátní role se rozumí možná výsledná role. Díky tomu můžeme odebrat oprávnění taková, která nemají dostatečný počet přiřazených uživatelů, aby mohla tvořit roli. Tato funkcionalita je definována v metodě

```
void removePermissionWhichCantMakeRole();
```

Tato metoda vychází z předpokladu, že oprávnění, která nejsou přiřazena dostatečnému počtu uživatelů, nemohou tvořit role. Proto je zbytečné, aby se s těmito oprávněními dále pracovalo.

Veškerá oprávnění, jejichž počet uživatelů je menší než parametr **„Minimální počet uživatelů“,** budou po tomto kroku odebrána a přidána do množiny zbytkových oprávnění. Tyto zbytková oprávnění jsou přímo přiřazená uživateli.

Samotný proces odstranění oprávnění dobře reprezentuje aktivity diagram na Obrázek 17.



Obrázek 17 - Odebrání oprávnění ze seznamu, která nemohou tvořit role.

9.4.2. Sloučení oprávnění

Druhá metoda, která předzpracovává data, se zaměřuje na sloučení posloupnosti oprávnění, která jsou stejná. Můžeme vycházet z předpokladu, že mnoho uživatelů bude

mít stejné definice oprávnění. Pokud tito uživatelé mají stejně definovaná oprávnění, není třeba je procházet pro všechny, ale stačí pouze jeden zástupce.

Tato funkcionalita je definována v metodě:

```
List<List<Permission>>joinPermissionLists()
```

Vrací list unikátních posloupností, která budou nadále zpracována v dalších částech programu. Jak je vidět na obrázku, v tomto případě lze sloučit posloupnosti oprávnění uživatelů U1 a U2, jelikož jsou tyto posloupnosti identické:

	P1	P2	P3	P4	P5	P6
U1	x	x	x	x	x	
U2	x	x	x	x	x	
U3	x	x				
U4		x	x		x	x

Obrázek 18 - zobrazení sloučení oprávnění

Společně s tím jsou taktéž odstraněny posloupnosti oprávnění, která jsou menší, než minimální počet oprávnění pro roli.

9.5. Extrakce kandidátních rolí

Po předchozích krocích máme posloupnosti oprávnění, ze kterých budeme vytvářet kandidátní role.

Pod pojmem „kandidátní role“ v tomto případě rozumíme všechny role, které odpovídají parametrům. Tyto role se mohou navzájem překrývat, případně býti duplicitní, ale v tomto kroku je potřeba vygenerovat všechny možné, protože následně budeme tyto role třídit a čistit.

Díky parametrizaci máme hranice, kde všude musíme dané role hledat a kam až v daném algoritmu máme zajít. Samotný seznam oprávnění je už očištěn od těch, ze kterých nemohou vzniknout oprávnění a díky předzpracování dat máme pouze data, která pro nás mají smysl.

Nyní přichází krok, ve kterém musíme najít všechny role, které vyhovují našim pravidlům. Toho se v daném algoritmu docílí pomocí nalezení kombinací.

9.5.1. Mapování kombinací na kandidátní role

Z principu definice role v RBAC modelu se role definuje jako množina oprávnění. Z tohoto důvodu lze tvrdit, že dvě role jsou shodné, pokud mají shodná oprávnění a to bez ohledu na pořadí, ve kterém jsou tato oprávnění definována.

Pokud má daná role oprávnění: *{P1, P2, P3}* ,tak je shodná s rolí, která vlastní oprávnění {*P2, P1, P3*}. Při hledání kandidátních rolí nám na pořadí nezáleží a hledáme všechny možnosti uspořádaných n-tic v daném rozmezí. Z tohoto předpokladu lze vyvodit, že hledáme tedy kombinace.

Kombinace lze vyjádřit jako podmnožinu o velikosti k z dané množiny M.

$$K \subseteq M$$

Samotná operace vyhledání všech kandidátních rolí se zaměřuje na hledání kombinací z dané sekvence, kterou jsme získali v předzpracování. Tyto kombinace pro nás představují všechny možné role, které můžeme najít a potřebujeme proto všechny, které odpovídají našemu kritériu.

Počet možností, které lze dosáhnout lze matematicky vyjádřit takto:

$$C(k,n) = \frac{n!}{(n-k)! * k!}$$

Kde *k* je velikost kombinace, kterou hledáme a *n* je velikost prvků ze kterých dané kombinace skládáme.

Z tohoto zápisu lze říci, že v případě, kdy hledáme role o velikosti k = 5, ze sekvence oprávnění velikosti n = 22, dosadíme do daného vzorce a po výpočtu máme 26 334 možných kombinací.

$$C\ (5,22) = \frac{22!}{(22-5)! * 5!} = 26\ 334$$

Z důvodu, že hledáme veškeré kombinace v daném intervalu <5, 12> se jedná o složitost vyjádřenou součtem náročnosti jednotlivých kombinací.

V tomto případě je velmi důležité brát tuto složitost na zřetel. Například při vytváření kombinací C(5, 70) se počet možných kombinací pohybuje kolem čísla 12.103.014 možných kombinací. Tento počet je pouze pro jedinou variantu kombinací. Samotný součet možných kombinací bude ještě mnohem větší.

Velmi dobře by se v této fázi mohly uplatnit paralelizační algoritmy, které by tuto úlohu extrakce mohly převzít a vrátit pouze výsledný data set, na kterém by se dále pracovalo.

9.6. Prioritizace rolí

V předchozích částech bylo demonstrováno, jakým způsobem lze vygenerovat kandidátní role, které v sobě obsahují všechny možné a myslitelné role, které vyhovují definovaným kritériím.

Počet těchto rolí je ještě velmi značný, a proto je nutné nějakým způsobem tyto role setřídit a dle definovaných parametrů určit, které si přejeme mít zachovány a ostatní, které se dle těchto rolí budou měnit. Toto uspořádání rolí bude mít následně velký vliv na vytváření výsledných rolí a utváření hierarchií, které budou postupně reflektovat pozice jednotlivých rolí v seznamu.

Pro prioritizace rolí a jejich uspořádání byly definovány tyto možnosti:

- Dle počtu uživatelů
- Dle velikosti role

Nyní si jednotlivé možnosti rozebereme společně s jejich implementací. V budoucnu je zde velký prostor pro to definovat nové metody, případně hodnocení rolí, které mohou lépe vystihnout danou situaci, kdy potřebujeme určité role upřednostnit.

9.6.1. Dle počtu uživatelů

Tato metoda prioritizace role vychází z faktu, že ze vstupního data-setu nejsme sice schopni vyčíst logické vazby, ale jsme schopni spočítat, kolik uživatelů má tuto roli přiřazenou. Respektive, kolik uživatelů je přiřazeno dané roli.

Tento součet jsme schopni dohledat již ve fázi extrakce rolí, kde se každá kandidátní role ověřuje, zda již není definována a případně jí je přiřazen uživatel, který drží stejná oprávnění.

Samotný počet uživatelů role je důležitý ukazatel kvality této role. Ze statistického hlediska jsme schopni říct, že role s největším počtem uživatelů je role základní, tedy role, kterou mohou mít přiřazenu všichni uživatelé a jedná se o základní práva, která jsou potřeba pro uživatelskou práci.

Proto je více než důležité využít tento atribut pro setřídění rolí a na vrchol listu přesunout role, které jsou definovány pro maximální počet uživatelů.

9.6.2. Dle velikosti role

V předchozím případě při setřídění pomocí počtu uživatelů přidělených roli může nastat situace, kdy existuje stejný počet uživatelů pro dvě role. V tomto případě je nutné rozhodnout, která z těchto rolí je důležitější.

Pro tento případ je vhodné využít velikost role jako ukazatel. V případě, kdy se jedná o shodný počet uživatelů, lze spočítat rozdíl mezi velikostí role a optimální definovanou velikostí v algoritmu. Z tohoto rozdílu vezmeme absolutní hodnotu a porovnáme tyto dvě hodnoty, která z nich je menší, ta se blíží více optimální velikosti role.

9.7. Očištění a hierarchie rolí

V této části již máme kandidátní role společně s jejich setříděným seznamem dle definovaných kritérií. Jak definuje NIST RBAC, role mohou mít hierarchické uspořádání. Což znamená, že role může obsahovat několik vnořených rolí. Bez definice této

hierarchie jsme velmi omezeni na pouhé shodnosti a překrytí množin, které můžeme v daném algoritmu odhalit.

Z tohoto důvodu je nutné porovnat proti sobě jednotlivé kandidátní role a vyřešit možné nastalé situace. Jsou možné tyto nastalé situace:

- $R1 \cup R2 = 0$ – role nemají žádné společné elementy
- $R1 \cup R2 \; ! = 0$ – role mají mezi sebou průnik
- $R1 = R2$– role jsou identické

Jestliže se role překrývají, tedy mají mezi sebou průnik, musíme zjistit jaký vztah tyto role mezi sebou mají. Může se jednat o role juniorní, seniorní anebo pouze o role, které mají shodné určité části oprávnění nikoliv však všechny.

Nastalé situace jsou definovány níže:

- $R1 \in R2$- role R1 obsahuje všechna oprávnění z R2, R1 je seniorní role roli R2
- $R2 \in R1$ – role R2 obsahuje všechna oprávnění z R1, R2 je seniorní role roli R1

Všechny tyto situace je nutno v algoritmu řešit. Samotný algoritmus vychází z principu postupného dělení rolí na menší pod části. Při každém porovnání role R1 a R2 se musí rozhodnout, zda se daná role bude měnit či nikoliv.

V současnosti algoritmus umožňuje tyto možnosti změny role:

1. ponechání současné role
2. rozdělení porovnávané role R2
3. odstranění role R2

Při každé změně, kdy se odstraní anebo rozdělí role, je nutno znovu setřídit a prioritizovat role. Protože při dělení nebo odstranění role se změnil počet uživatelů stejně jako další možná kritéria, která mohou mít vliv na pořadí rolí.

Samotný algoritmus lze pak popsat v pseudokódu takto:

❖ *Pro všechny kandidátní role:*

- *Vyber dvě role pro porovnání – R1 a R2.*
- *Jsou tyto role shodné?*
  - *Vyber další roli R2 pro porovnání.*
- *Vlastní role R1 všechna oprávnění R2?*
  - *Lze tyto role rozdělit?*
    - *Pokud ano, rozděl tyto role na dvě jiné role.*
- *Vlastní role R2 všechna oprávnění R1?*
  - *Lze tyto role rozdělit?*
    - *Pokud ano, rozděl tyto role na dvě jiné role.*
  - *Pokud ne, sluč tyto role a to následovně:*
    - *Přiřaď uživatele R2 k R1.*
    - *Oprávnění, která jsou rozdílná* $(R2 - R1)$ *ulož do zbytkových oprávnění.*
    - *Odeber R2.*
- *Vlastní role R1 pouze některá z oprávnění R2?*
  - *Vlastní role R1 všechny uživatele R2?*
    - *Odeber roli R2.*

Implementaci algoritmu si lze prohlédnout v přiloženém CD.

9.8. Ukázkové zpracování dat

Pro názornou ukázku práce algoritmu předpokládejme tyto vstupní data:

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
U1	x	x			x		x	x	x			x
U2	x	x			x		x	x	x			x
U3	x	x		x			x	x	x	x		
U4	x	x	x	x		x		x	x			
U5	x	x	x	x				x	x		x	
U6	x	x	x	x				x	x		x	
U7	x	x			x		x	x	x			x
U8	x	x			x		x	x	x			x
U9	x	x	x	x		x		x	x			
U10	x	x	x	x		x		x	x			
U11	x	x		x			x	x	x	x		
U12		x							x			

Obrázek 19 - Vstupní data algoritmu

Na obrázku výše můžeme vidět tabulku, kde jsou znázorněna vstupní data. V levém sloupci jsou uživatelé a v horním řádku jednotlivá oprávnění.

Tento vzorek obsahuje 12 uživatelů – U1, U2, … U12 a 12 různých oprávnění – P1, P2, ... P12.

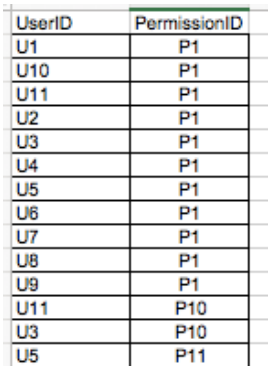
Z této tabulky můžeme vyčíst jednotlivé přiřazení oprávnění pro uživatele. Pokud má v daném řádku a sloupci značku *„x“*, má toto oprávnění přiřazeno.

Například:

- U1 má tyto oprávnění {P1, P2, P5, P7, P8, P9, P12}
- U3 má tyto oprávnění {P1, P2, P4, P7, P8, P9, P10}
- U10 má tyto oprávnění {P1, P2, P3, P4, P6, P8, P9}

Tímto postupem jsme schopni vyčíst pro každého uživatele jeho oprávnění, stejně tak pro každé oprávnění uživatele, kteří ho mají přiděleno.

Pro vstup do algoritmu je nutno tuto tabulku přepsat do vstupního formátu dat CSV. Vstupní soubor pro tuto ukázkovou tabulku obsahuje 80 záznamů. Ukázka vstupu lze vidět na obrázku níže.



UserID	PermissionID
U1	P1
U10	P1
U11	P1
U2	P1
U3	P1
U4	P1
U5	P1
U6	P1
U7	P1
U8	P1
U9	P1
U11	P10
U3	P10
U5	P11

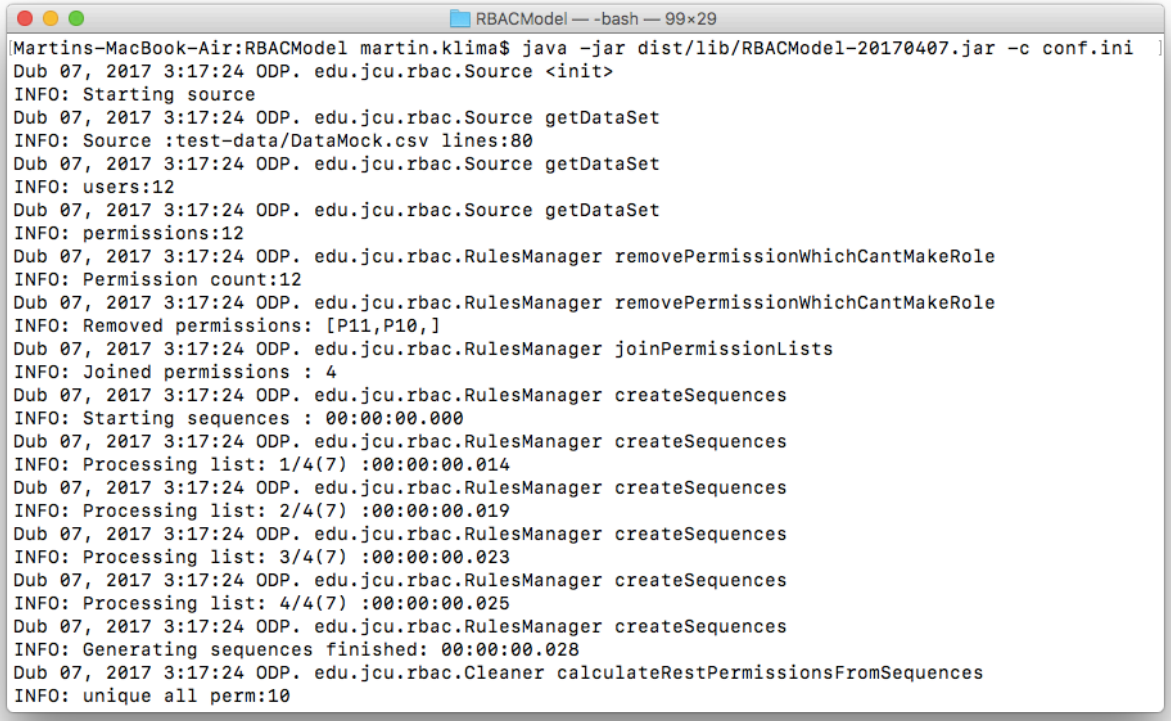
Obrázek 20 - Vstup algoritmu pro testovací data

Pro první průchod algoritmu byly zvoleny tyto parametry:

min_role_size	*3*
max_role_size	*10*
optimal_role_size	*5*
min_users_for_role	*3*

Tyto parametry definují, že vytvořená role by měla být tvořena minimálně třemi oprávněními, maximálně však deseti a optimální velikost pro nás je pět oprávnění.

Stejně tak nalezená role musí mít alespoň tři uživatele. Ukázku spuštění programu s takto zadanými parametry a vstupními daty lze vidět na obrázku níže.



Obrázek 21 - Ukázka spuštění programu pro testovací data

V prvním kroku algoritmus úspěšně načetl vstupní data.

Ve druhém kroku začal předpracováním dat. Algoritmus zjistil, že podle zadaných parametrů nelze vytvořit role z oprávnění P10 a P11. Tyto oprávnění jsou přiřazena pouze dvěma uživatelům a neodpovídá to našemu požadavku, aby výsledná role měla minimálně tři přiřazené uživatele. Algoritmus tyto oprávnění nebude dále využívat pro generování kandidátních rolí, ale uloží je jako oprávnění přiřazená mimo role.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
U1	x	x			x		x	x	x			x
U2	x	x			x		x	x	x			x
U3	x	x		x			x	x	x	x		
U4	x	x	x	x		x		x	x			
U5	x	x	x	x				x	x		x	
U6	x	x	x	x				x	x		x	
U7	x	x			x		x	x	x			x
U8	x	x			x		x	x	x			x
U9	x	x	x	x		x		x	x			
U10	x	x	x	x		x		x	x			
U11	x	x		x			x	x	x	x		
U12		x							x			

Obrázek 22 - Algoritmus identifikovat oprávnění, která se mají odebrat

Následně algoritmus bude spojovat posloupnosti oprávnění, aby nalezl unikátní posloupnosti. Z našeho vzorku můžeme nalézt čtyři unikátní posloupnosti oprávnění. Tyto unikátní posloupnosti jsou:

- {P1, P2, P3, P4, P8, P9}
- {P1, P2, P5, P7, P8, P9, P12}
- {P1, P2, P3, P4, P6, P8, P9}
- {P1, P2, P4, P7, P8, P9}

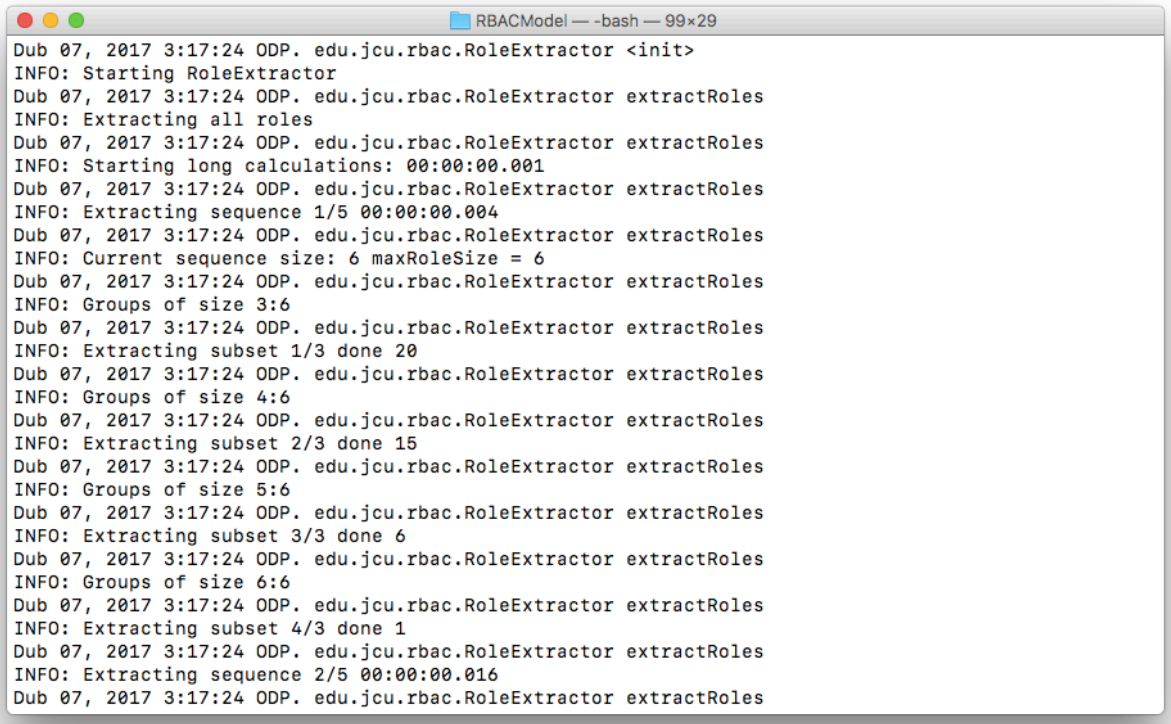
Pouze pro tyto unikátní posloupnosti má smysl hledat kandidátní role. Upravená tabulka níže ukazuje, kteří uživatelé patří do těchto unikátních posloupností.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P12
U5	x	x	x	x				x	x	
U6	x	x	x	x				x	x	
U2	x	x			x		x	x	x	x
U1	x	x			x		x	x	x	x
U7	x	x			x		x	x	x	x
U8	x	x			x		x	x	x	x
U9	x	x	x	x		x		x	x	
U4	x	x	x	x		x		x	x	
U10	x	x	x	x		x		x	x	
U3	x	x		x			x	x	x	
U11	x	x		x			x	x	x	
U12		x							x	

Obrázek 23 - Rozčleněná tabulka do unikátních posloupností

Uživatel U12 není v žádné posloupnosti, protože jeho oprávnění jsou pouze dvě, tedy méně, než kolik je limit pro vytvoření role. Z tohoto důvodu bude uživatel U12 odebrán a následně mu budou oprávnění přiřazena pomocí zbytkových oprávnění.

Po očištění oprávnění přichází fáze, kdy generujeme z výše popsaných unikátních posloupností kandidátní role.



Obrázek 24 - Extrakce kandidátních rolí

V současném případě bylo vygenerováno 192 kandidátních rolí, které budou následně očištěny, prioritizovány a hierarchicky rozděleny. Po tomto očištění a rozdělení podle hierarchie algoritmus vrátí výsledné role.

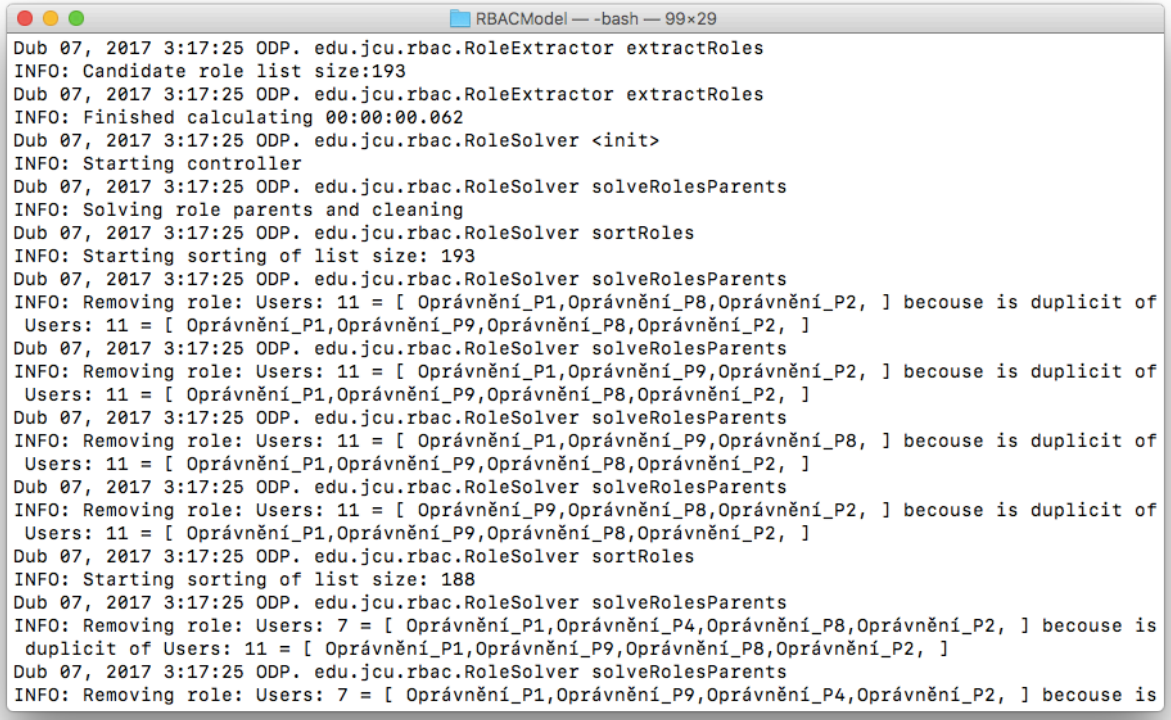
Algoritmus při zadaných parametrech nalezl tři výsledné role.

- Role 1 = {P1, P2, P8, P9}
- Role 2 = {P3, P4, P6}
- Role 3 = {P5, P7, P12}

Následně byla přiřazena tato oprávnění uživatelům mimo role.

- Oprávnění P11 = Uživatel U5, Uživatel U6,
- Oprávnění P10 = Uživatel U3, Uživatel U11,
- Oprávnění P9 = Uživatel U12,
- Oprávnění P2 = Uživatel U12,

- Oprávnění P4 = Uživatel U3, Uživatel U5, Uživatel U11, Uživatel U6,
- Oprávnění P7 = Uživatel U3, Uživatel U11,
- Oprávnění P3 = Uživatel U5, Uživatel U6,



Obrázek 25 - Třídění a řešení hierarchie rolí

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
U1	x	x			x		x	x	x			x
U2	x	x			x		x	x	x			x
U3	x	x						x	x			
U4	x	x	x	x		x		x	x			
U5	x	x						x	x			
U6	x	x						x	x			
U7	x	x			x		x	x	x			x
U8	x	x			x		x	x	x			x
U9	x	x	x	x		x		x	x			
U10	x	x	x	x		x		x	x			
U11	x	x						x	x			
U12												

Obrázek 26 - Výsledek algoritmu při definovaných parametrech (černě označené buňky jsou oprávnění přiřazená mimo role)

Z daného výsledku je vidět, že existuje role, které mají pouze dva uživatelé – U11 a U3, stejně tak U5 a U6. Tyto role je možno najít, pokud upravíme parametry algoritmu.

Snížíme tedy parametr minimální počtu uživatelů na nižší hodnotu.

min_users_for_role	*2*

V této konfiguraci výsledek algoritmu vypadá takto:

- Role 1 = {P1, P2, P8, P9}
- Role 2 = {P5, P7, P12}
- Role 3 = {P3, P4, P6}
- Role 4 = {P3, P11, P4}
- Role 5 = {P7, P10, P4}

Následně byla přiřazena tato oprávnění uživatelům mimo role.

- Oprávnění P9 : Uživatel U12,
- Oprávnění P2 : Uživatel U12

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
U1	x	x			x		x	x	x			x
U2	x	x			x		x	x	x			x
U3	x	x		x			x	x	x	x		
U4	x	x	x	x		x		x	x			
U5	x	x	x	x				x	x		x	
U6	x	x	x	x				x	x		x	
U7	x	x			x		x	x	x			x
U8	x	x			x		x	x	x			x
U9	x	x	x	x		x		x	x			
U10	x	x	x	x		x		x	x			
U11	x	x		x			x	x	x	x		
U12												

Obrázek 27 - Výsledek algoritmu po změně parametrů

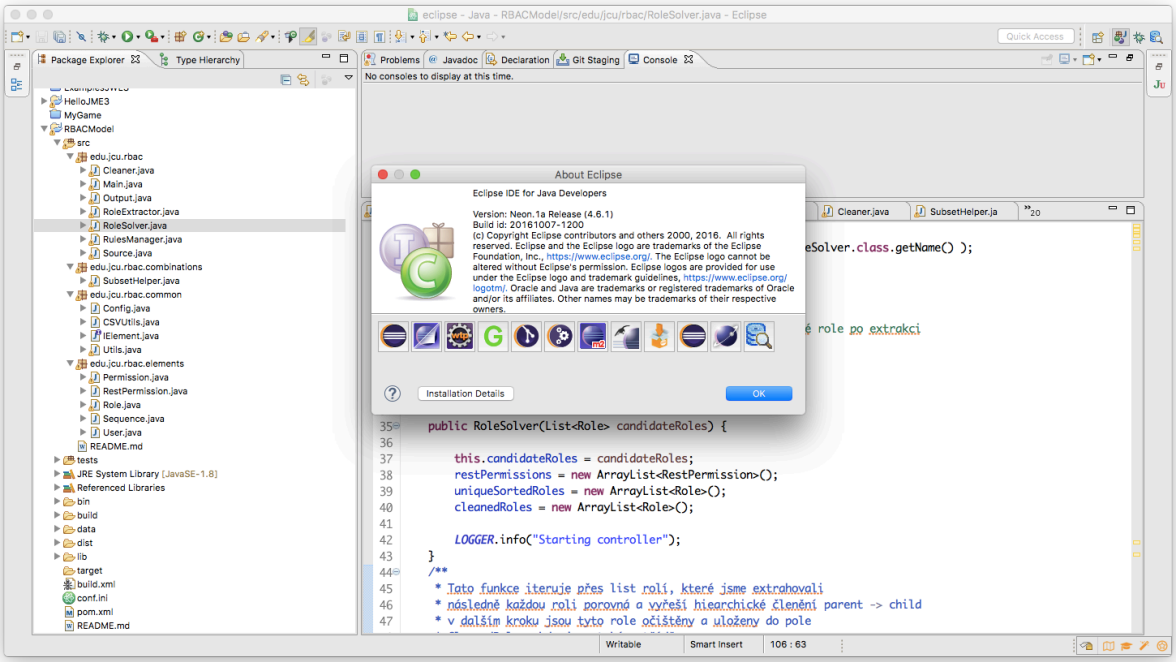
10.Detaily implementace

V této části budou popsány informace ohledně implementace algoritmu.

Algoritmus je implementován v Javě. Tento programovací jazyk byl vybrán z několika důvodů:

1. Jedná se o staticky typovaný jazyk, který se kompiluje, díky tomu lze již při kompilaci odhalit chyby, které by byly jinak objeveny až za běhu.
2. Java ke svému běhu používá mezivrstvu – Java Virtual Machine (JVM). Díky tomu lze tento algoritmus spouštět na jakémkoliv systému, který je s JVM kompatibilní.
3. Jazyk Java je velmi stabilní, precizně zdokumentovaná, její zdrojový kód je otevřený a taktéž obsahuje množství již odzkoušených a ověřených knihoven, které lze pro vývoj využívat.

Pro tuto konkrétní implementaci byla použita Java verze 1.8.Vývojovým nástrojem pro vytváření a testování algoritmu byl Eclipse Neon.



Obrázek 28 - Vývojové prostředí Eclipse Neon

10.1. Kompilace a knihovny

Algoritmus pro svůj běh potřebuje mít instalovanou Javu verze 1.8. a vyšší.

Krom toho využívá taktéž tyto knihovny, které musejí být při kompilaci vloženy k projektu.

- ini4j-0.5.4.jar
- commons-lang3-3.4.jar
- commons-cli-1.3.1.jar

Tyto knihovny většinou slouží pro parsování argumentů z příkazové řádky nebo čtení ini, či csv souboru. Lze je najít ve složce „libs“ na přiloženém CD.

Algoritmus lze kompilovat pomocí nástroje Apache Ant. Součástí algoritmu je soubor *build.xml*, který definuje způsob kompilace. Pro správnou kompilaci je nutné mít instalován Apache Ant a přidán ho do globálního PATH pro spuštění z příkazové řádky.

Apache Ant lze stáhnout zde: http://ant.apache.org/

Verze používána ke kompilaci projektu*: Apache Ant(TM) version 1.10.1*

Při kompilaci se vytvoří výsledný jar soubor ve složce „dist/lib/“. Tento soubor má název např. „RBACModel-20170330.jar“, hodnota za pomlčkou se mění dle časového razítka při kompilaci.

Obrázek 29 - Ukázka kompilace projektu

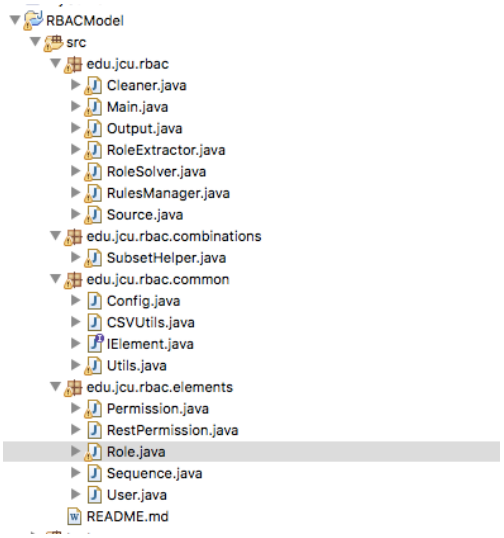
10.2. Struktura projektu

Projekt je rozčleněn do několika balíčků:

- **edu.jcu.rbac**
  - *V tomto balíčku jsou umístěny základní třídy, které implementují jednotlivé kroky RBAC algoritmu.*
- **edu.jcu.rbac.combinations**
  - *V tomto balíčku jsou umístěny třídy pro výpočet kombinací, které se generují při extrakci kandidátních rolí.*
- **edu.jcu.rbac.common**
  - *V tomto balíčku jsou umístěny třídy, které jsou společné pro celý projekt.*
- **edu.jcu.rbac.elements**

- *V tomto balíčku jsou třídy, které tvoří jednotlivé elementy (Role, Oprávnění, Uživatel…) a využívají se ve fázích procesu.*

Struktura projektu lze velmi dobře vidět na Obrázek 30 - Struktura projektu.



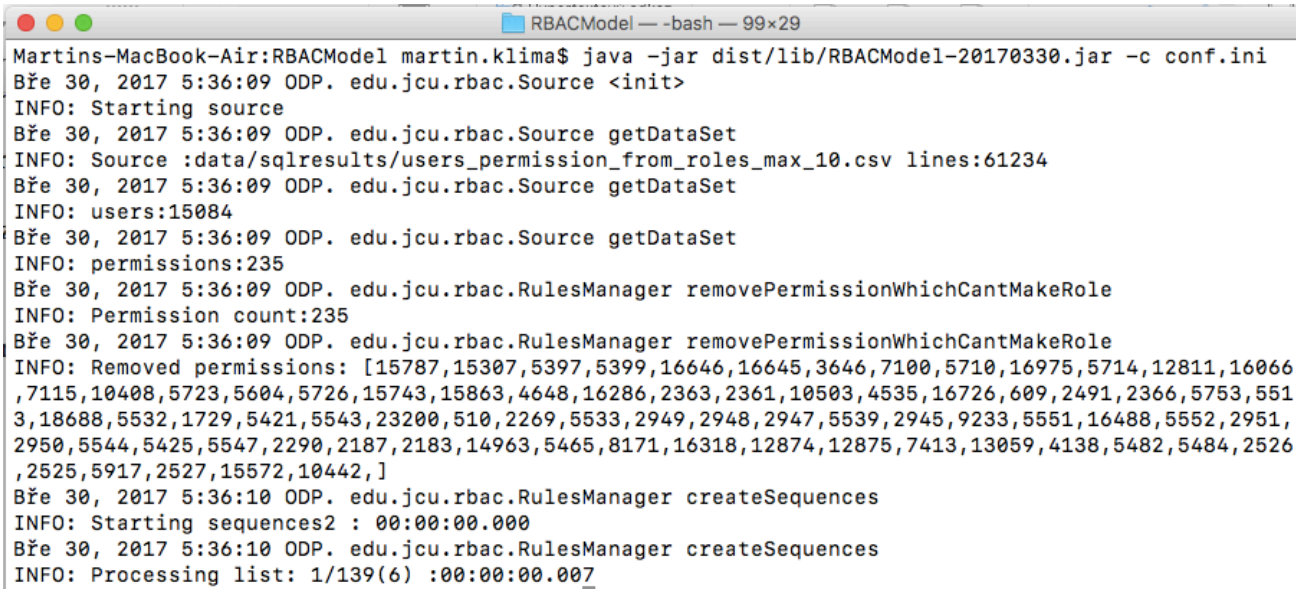
Obrázek 30 - Struktura projektu

Projekt je taktéž dostupný online na sdíleném úložišti GitHub. Lze ho nalézt pod adresou: https://github.com/Pooky/rbac-algorithm.V případě dalších změn nebo vylepšení budou všechny nejaktuálnější změny k nalezení na této stránce.

Spuštění algoritmu lze provést pomocí následujícího příkazu:

```
java -jar dist/lib/RBACModel-20170330.jar -c conf.ini
```

Tento příkaz volá aktuální verzi Javy, které následně předává cestu ke spustitelnému .jar souboru společně s parametrem, kde je umístěn konfigurační soubor.



Obrázek 31 - Ukázka spuštění programu.

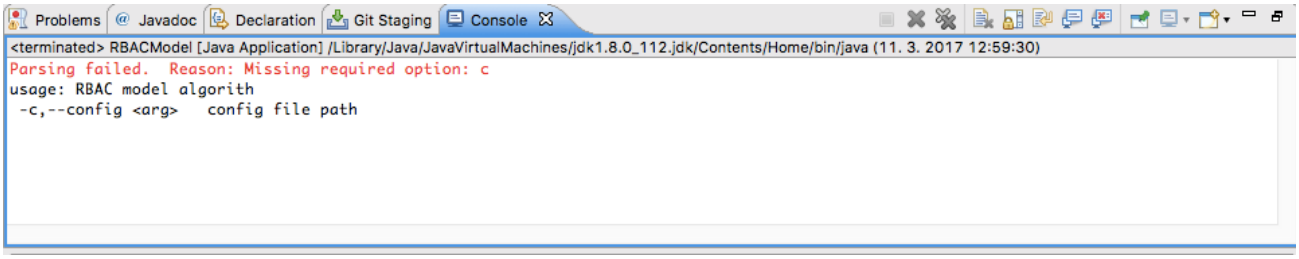
10.2.1. Konfigurace algoritmu

Parametry programu je nutno definovat na vstupu pomocí .ini souboru, ve kterém budou všechny potřebné informace. Tento formát je velmi jednoduchý, jedná se o zápis pomocí párů, ve kterém je na prvním místě klíč a následně jeho hodnota. Program očekává tento soubor při svém vstupu. Je nutné zadat jeho cestu jako parametr při spouštění.

Ukázková syntaxe parametru je následující:

```
-c, --config config.ini
```

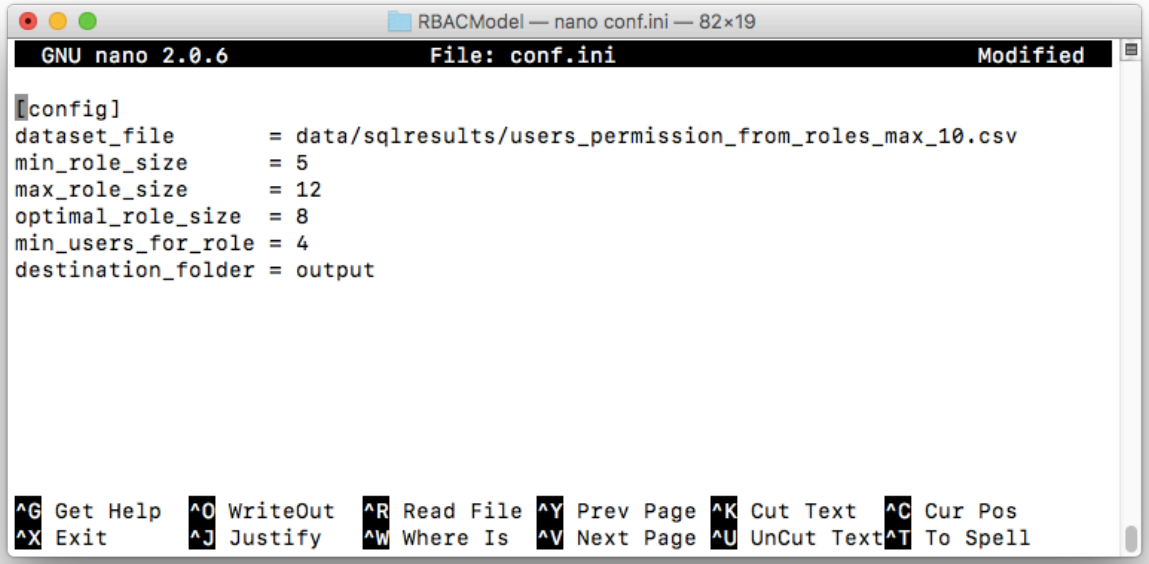
Pokud tento parametr chybí, program vrátí chybu, společně s popisem, jak má být tento argument zadán.



Obrázek 32 - programu nebyl zadán konfigurační soubor

V současné době je možno definovat tyto parametry:

Název atributu	Typ atributu	Popis
dataset_file	String	Cesta k CSV souboru pro import dat.
min_role_size	Integer	Minimální velikost role
max_role_size	Integer	Maximální velikost role
optimal_role_size	Integer	Optimální velikost role
min_users_for_role	Integer	Minimální počet uživatelů pro roli
destination_folder	String	Složka pro uložení výsledného modelu.



Obrázek 33 - ukázka konfiguračního souboru

10.2.2. Implementace Uživatele

Stejně jako v teoretické definici RBAC modelu je potřeba v implementaci reprezentovat uživatele. Uživatele reprezentujeme v jeho základní podobě, kterou používáme při načítání dat. Nejsou tu žádná omezení, dovolujeme uživateli mít více stejných oprávnění, stejně jako neunikátní identifikátor. Tyto nekonzistence by se měly řešit dříve, než bude uživatel vytvořen.

Tato entita je prezentována třídou *User.java*

Plný název cesty: */RBACModel/src/edu/jcu/rbac/elements/User.java*

Balíček třídy: *edu.jcu.rbac.elements*;

Třída User.java má tyto atributy:

- $\boldsymbol{Id : String}$ – identifikátor uživatele, tento atribut je povinný a měl by být unikátní.
- $\boldsymbol{Name : String}$ – název uživatele, tento atribut je dobrovolný a slouží k lepší identifikaci uživatele.
- $\boldsymbol{Permissions : List} < Permission >$- seznam oprávnění, které má uživatel definováno.

Všechny definované metody je možno najít v implementaci či na class diagramu na Obrázek 34. Pro nás jsou důležité tyto metody:

Přidání oprávnění:

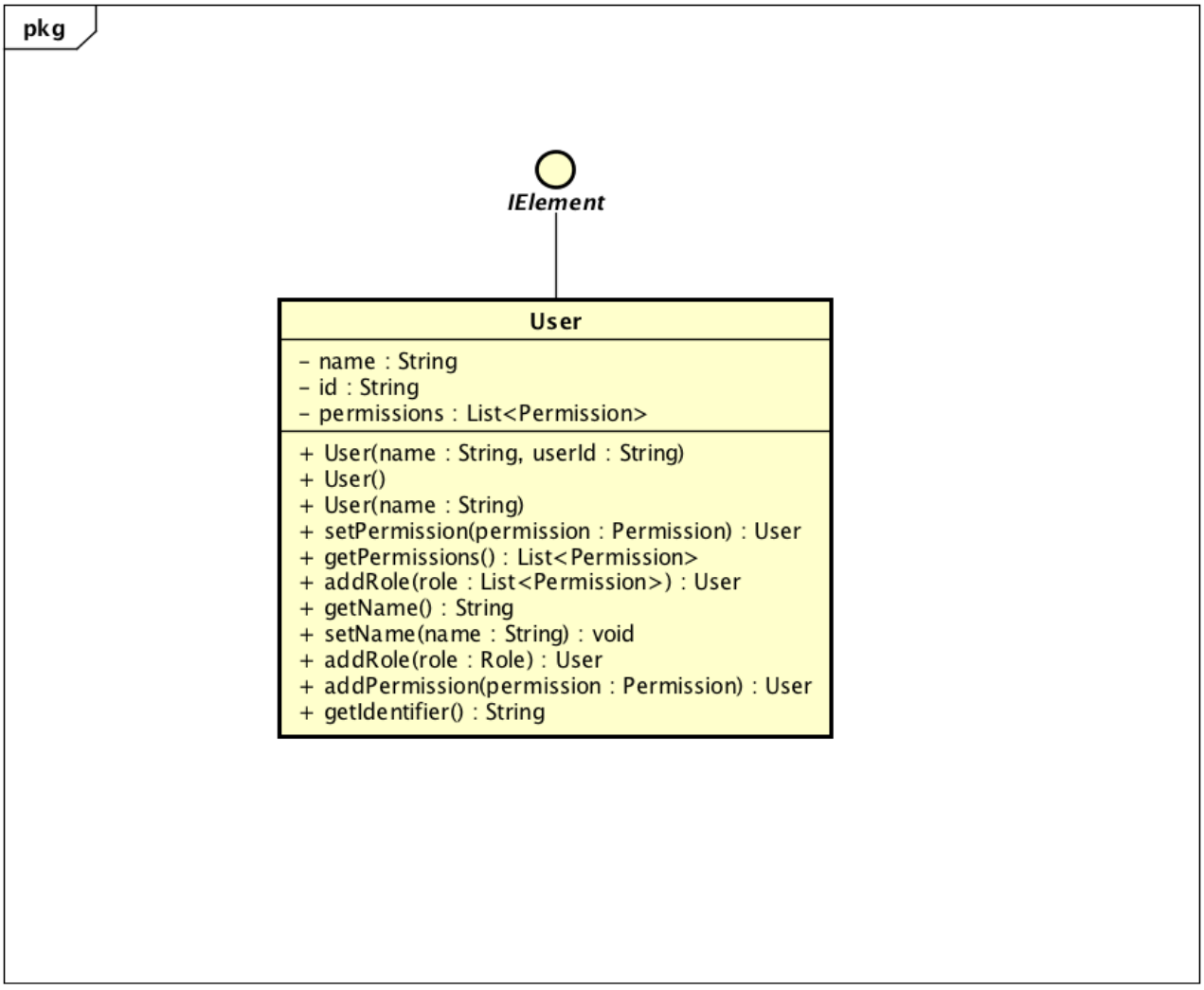
```
public User addPermission(Permission permission) {
      this.permissions.add(permission);
      return this;
}
```

V této metodě je na vstupu definováno oprávnění, které je následně přidáno uživateli do jeho listu s oprávněními.

Přidání role:

```
public User addRole(Role role) {
      this.permissions.addAll(role.getPermissions());
      returnthis;
}
```

V této metodě je přidána role uživateli. Jako parametr tato metoda přijímá danou roli, kterou chceme uživateli přiřadit, následně jsou z role zavolána oprávnění a tato oprávnění jsou přiřazena uživateli.



Obrázek 34 - Class diagram třídy User.java

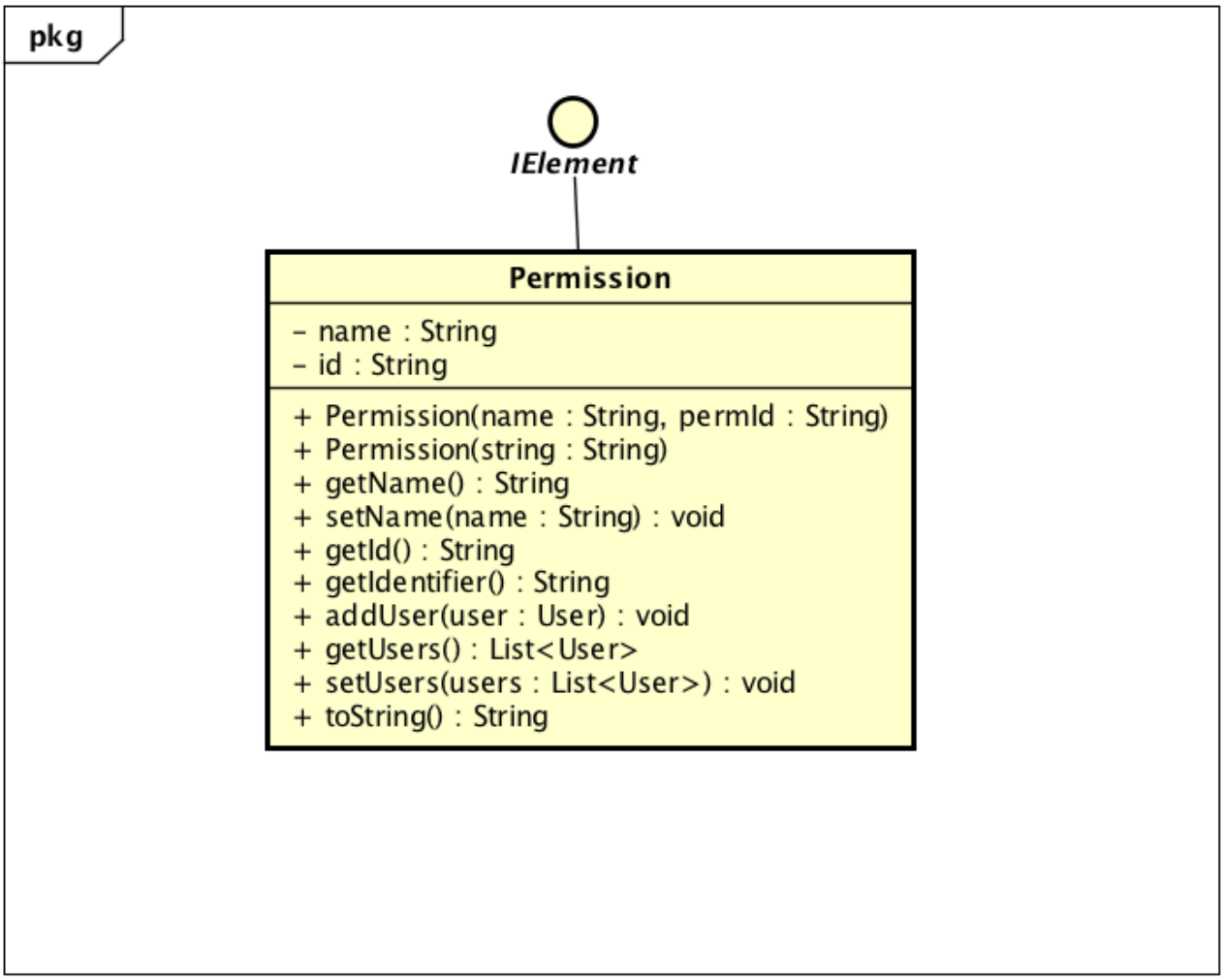
10.2.3. Implementace Oprávnění

Teoretické oprávnění z RBAC modelu implementujeme v algoritmu pomocí jednoduché třídy Permission.java. Tato třída slouží pro uchovávání informací o oprávnění, jeho ID a vazbách na uživatele, kteří toto oprávnění vlastnili z importovaného data setu.

Tato entita je prezentována třídou *Permission.java*

Plný název cesty: */RBACModel/src/edu/jcu/rbac/elements/Permission.java*

Balíček třídy: *edu.jcu.rbac.elements*;



Obrázek 35 - Implementace Permission.java

10.2.4. Implementace Role

V RBAC modelu velmi často mluvíme o rolích a jak je optimalizovat. V samotné implementaci slouží k vyjádření role tato třída. Při importu dat není tato třída vůbec využita, jelikož vstup algoritmu nepočítá s již definovanými rolemi. Tato třída je využívána až v kroku, kdy se z předpřipravených dat generují kandidátní role.

Kromě samotné abstrakce třídy z RBAC modelu má tento objekt ještě velmi důležitou logiku při porovnávání svých vztahů s jinými rolemi – zda jsou tyto role identické, nadřazené, podřazené, případně zda existuje jejich průnik.

Tato entita je prezentována třídou *Role.java*

Plný název cesty: */RBACModel/src/edu/jcu/rbac/elements/Role.java*

Balíček třídy: *edu.jcu.rbac.elements*;

Shodnost rolí definujeme jako stejný obsah listu jejich oprávnění. O porovnání těchto dvou objektů se stará tato přetěžovaná metoda „equals“, která porovnává dvě role:

```
public boolean equals(Object obj){

            Role role = (Role) obj;

      if(role.getPermissions().containsAll(this.getPermissions()
      )  &&
      this.getPermissions().containsAll(role.getPermissions())){
                        return true;
                  }
            return false;

      }
```

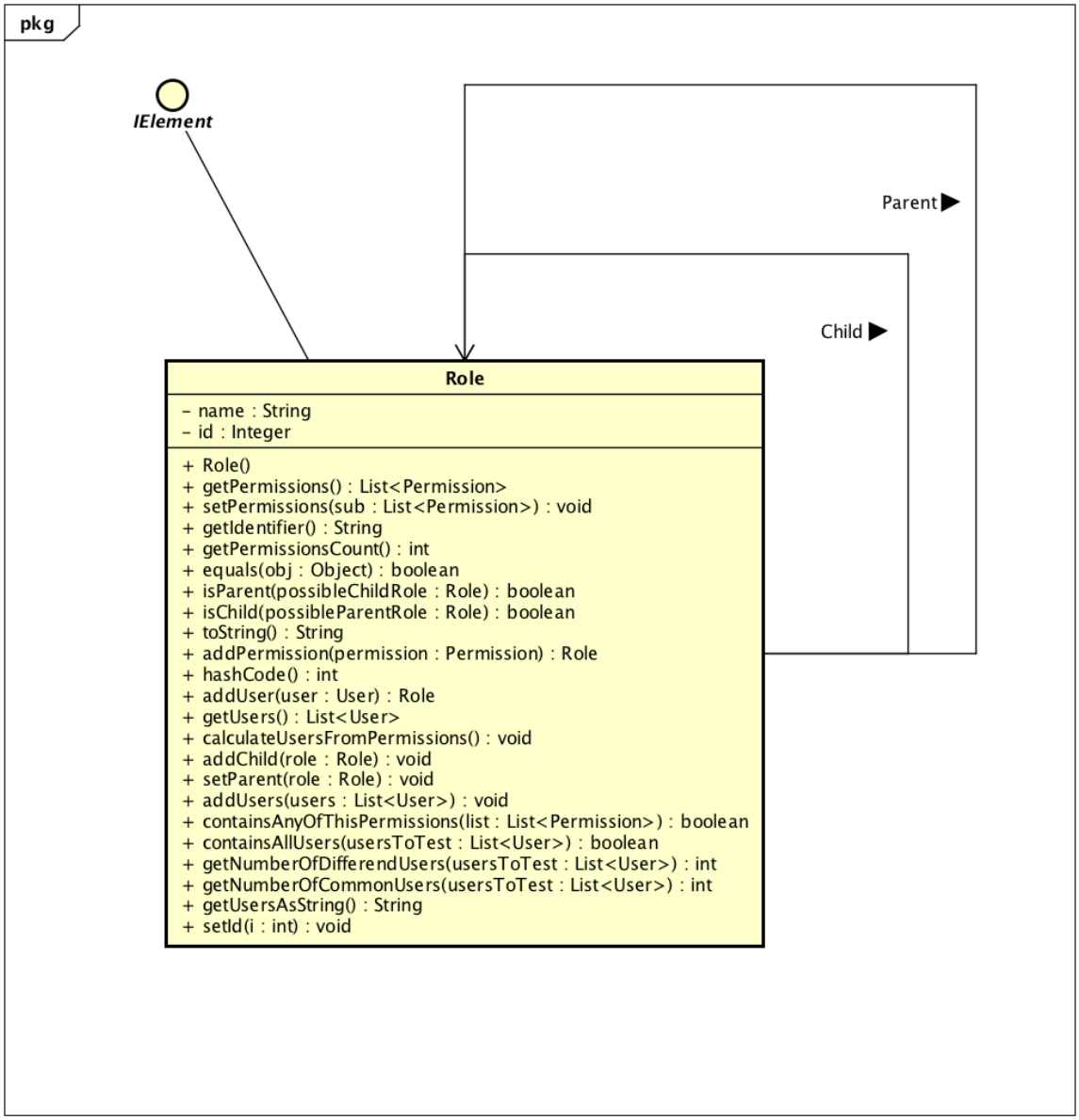
Mnohdy v implementaci pracujeme s hierarchickým uspořádáním rolí. Z tohoto důvodu je potřeba porovnání, zda je aktuální role seniorní nebo juniorní porovnávané roli. K tomu slouží metoda „isParent“, případně „isChild“, která slouží pro opačný případ. Samotné určení, zda se jedná o juniorní nebo seniorní roli závisí na oprávnění, která musí být obsažena všechna v nadřazené roli.

```
public boolean isParent(Role possibleChildRole){

      if(this.equals(possibleChildRole) ||
            this == possibleChildRole){
            return false;
      }

      if(getPermissions().containsAll(
            possibleChildRole.getPermissions())){
            return true;
      }else{
            return false;
      }

}
```



Obrázek 36 - Implementace Role.java

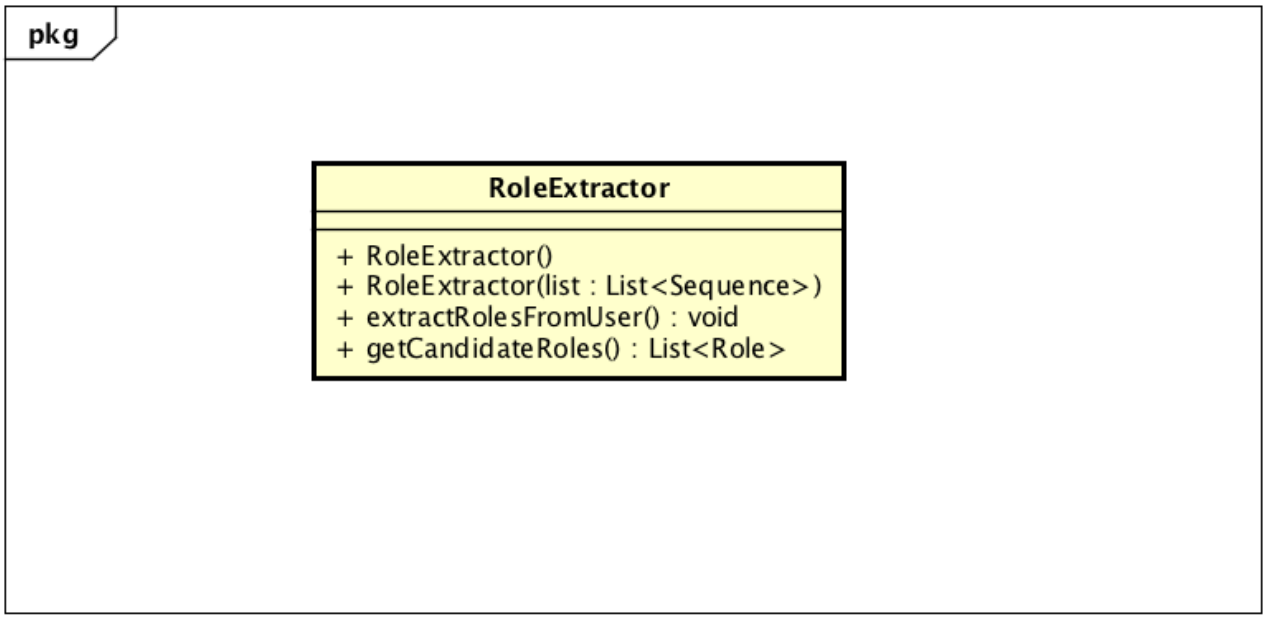
10.2.5. Implementace Extrakce rolí

Pro úlohu extrakce všech kandidátních rolí je vytvořena třída RoleExtractor.java, která zajišťuje extrakci všech kandidátních rolí. Tato třída se stará o tyto činnosti:

- Vygenerování sekvencí dle zadaných parametrů
- Ze sekvence vygenerování kandidátních rolí
- Vybrání unikátních kandidátních rolí a uložení do seznamu
- Vrácení unikátních kandidátních rolí

Plný název cesty: */RBACModel/src/edu/jcu/rbac/RoleExtractor.java*

Balíček třídy: *edu.jcu.rbac*;



Obrázek 37 - Class Diagram třídy RoleExtractor.java

K samotnému vygenerování jednotlivých kombinací slouží třída SubsetHelper.java, která se stará o vygenerování všech možných párů dle zadaných parametrů.

10.3. Výstup a Testování algoritmu

Algoritmus vrací optimalizovanou množinu rolí pro RBAC model společně se zbytkovými oprávněními, která jsou přidělena uživateli napřímo.

Výstup algoritmu je ve formátu .csv. Je téměř totožný se vstupními daty, která jsou popsána v kapitole: 9.3.2 Struktura vstupních dat. Ukládá se do složky definované uživatel v configu pod parametrem: „destination_folder“. Obsahuje tyto tři soubory:

- permission_role.csv – v tomto souboru je výsledný vztah Rolí x Oprávněním.
- user_role.csv – v tomto souboru je výsledný vztah mezi Uživatel x Role.
- user_permission.csv – v tomto souboru je výsledný vztah mezi Uživatelem x Oprávněním.

Pro účely testování byla algoritmu poskytnuta testovací data. Tyto data bylo možné ověřit vůči kontextu a obsahovala hierarchické struktury a překrývající se role. Pro tyto data byl algoritmus schopen zrekonstruovat původní rozložení rolí při správně zadaných parametrech.

Následně byl algoritmu poskytnut anonymní vzorek dat z velké firmy, která má implementovaný RBAC model. Tento vzorek obsahoval 61, 235 záznamů Uživatel x Oprávnění. Algoritmus byl ověřen pomocí porovnání výsledného data setu vůči původnímu vzorku.

Algoritmus dokázal správně vyextrahovat výsledný data-set, který byl při zpětném rozložení totožný se vstupem. Byla tedy splněna podmínka 100 % kompatibility.

Následně algoritmus vygeneroval role, které oscilovaly kolem optimální velikosti a splňovaly zadané parametry. Tedy maximální a minimální velikost role společně s minimálním počtem uživatelů.

Samotné testování při vstupu a výstupu bylo prováděno v relační databázi MySQL.

11. Možnosti rozšíření

Výzkum v oblasti optimalizace rolí v RBAC modelu je velice široká disciplína. I přes snahu autora bylo možno zaměřit se pouze na velmi malou část, která byla v této práci obsažena a následně prakticky demonstrována.

Při tvorbě algoritmu a jeho testování se nalezlo velmi mnoho cest, kterým by bylo možno zvýšit efektivitu případně funkcionalitu celého řešení.

Mezi tyto oblasti patří:

1. Paralelizace algoritmu při extrakci rolí – *tvoření kombinací ve fázi extrakci kandidátních rolí je velmi náročné na výpočetní výkon, a přitom je tato úloha je matematicky dobře popsatelná a taktéž lze rozdělit na několik dílčích kroků, které by mohly vykonávat oddělené procesy paralelně.*

2. Definice statických rolí společně s jejich hodnocením – *tato funkcionalita by umožňovala uživateli definovat role, které on sám považuje za neměnné a následně by tyto role byly použity pro rozdělení rolí ostatních. Lze taktéž například definovat jejich bodové ohodnocení, které by pomáhalo ve fázi prioritizace rolí rozhodnout, na kterou pozici tyto role umístit.*

3. Předzpracování dat pomocí statistických metod – *mnoho statistických metod umožňuje projít daný vzorek dat a například navrhnout parametry, které se jeví jako nejvíce používané v daném vzorku. Tento krok by bylo vhodné začlenit jako pomůcku pro administrátory, kteří by měli alespoň částečné vodítko, kde by se tyto parametry měli pohybovat.*

12. Alternativní algoritmy a nástroje

Otázkou optimalizace tvorby rolí se ve svých výzkumech nebo pracích zabývá mnoho výzkumníků, v této části bych rád uvedl několik z nich, které mohou poskytnout srovnání nebo jiný pohled přístupu k dané problematice.

ORCA Miner [9]

Tento nástroj se zaměřuje na vyhledávání rolí pomocí shlukové analýzy. Algoritmus použitý v tomto nástroji hledá shluky (clustery), které mají největší možný počet společných uživatelů s daným oprávněním.

Velkým nedostatkem tohoto algoritmu je nezohlednění stavu, kdy dvě role mohou mít stejné oprávnění. Algoritmus tento stav nepodporuje, jelikož při hledání clusterů neumožňuje žádné překrývání, což má za následek fakt, že role nemohou mít shodná oprávnění.

Subset Miner [10]

Autoři teoretické definice hledání nejmenšího možného mima rolí v RBAC modelu vyvinuli svůj vlastní algoritmus, který by měl sloužit pro efektivní vyhledání rolí z přiřazení mezi uživatel x oprávněním.

Přístup, který autoři prezentují se velmi shoduje s prvním krokem ve výše popsaném algoritmu pro optimalizaci tvorby rolí – v kapitole 9.4.2 Sloučení oprávnění. Jejich nástroj postupuje obdobným přístupem, kdy slučuje jednotlivá oprávnění od uživatelů, čímž dle jejich popisuje generují možné kandidátní role.

Nevýhodou tohoto přístupu je nezohlednění možnosti, že uživatel má přiřazeno více než jednu roli. Výhodou algoritmu je jeho složitost, která je lineární. Stačí pouze procházet uživatele a následně ukládat unikátní sekvence oprávnění, která ještě nejsou zaznamenána.

RMiner [11]

Nástroj RMiner jako takový nepředstavuje nový algoritmus nebo metodu, jak vytvářet či optimalizovat role v RBAC modelu. Místo tohoto se zaměřuje na možnost spojit jednotlivé algoritmy dříve již známé a vytvořit platformu, která by umožňovala tyto nástroje použít.

Výsledkem jejich bádání je nástroj, který umožňuje pro data využít těchto 7 algoritmů:

- ORCA
- HierarchicalMiner
- CompleteMiner
- FastMiner
- StateMiner
- GraphOptimization
- MinimalPerturbation

Tento nástroj vychází z dataminingového nástroje WEKA. Pro svůj vstup využívá formátu TARF a ARFF, což jsou formáty používané pro datamining a vazbu dat.

13.Závěr

Cílem této práce bylo představit RBAC model a prezentovat algoritmus, který umožňuje optimalizovat tvorbu rolí v tomto modelu.

V první části byla představena teoretická definice různých druhů přístupů, popsán standard NIST RBAC modelu, který definuje jeho součásti a jednotlivé vztahy a taktéž představena problematika tvorby rolí obecně. Následně byl představen proces, který se optimalizací rolí zabývá a definuje klíčové komponenty pro optimalizaci tvorby rolí. Krom tohoto modelu bylo vysvětleno, jak lze RBAC model mapovat na binární matrix – tedy na maticové řešení, které lze řešit matematicky či programově.

V druhé půli této práce je představen algoritmus, který se prakticky zabývá implementováním jednotlivých prvků, které jsou zmíněny v předchozích kapitolách. Tento algoritmus je rozdělen na 8 kroků a popsán podle jednotlivých funkčních bloků, společně s popisem a znázornění postupu při každé operaci. Algoritmus umožňuje parametrizaci a přijímá na vstup definici uživatele a jeho oprávnění.

Kromě popisu teoretického je tento algoritmus taktéž implementován v Javě a jednotlivé implementační detaily jsou uvedeny v jednotlivých kapitolách společně s ukázkou kódu. Celkový funkční kód je připojen v příloze na CD.

Algoritmus byl testován na testovacích a následně anonymních datech. Tento anonymizovaný vzorek měl velikost 64.000 záznamů. Samotné testování potvrdilo, že vstupní data jsou totožná s výstupními, tedy že nedochází k omylnému porušení vazeb nebo jinému znečištění původního vstupu. Tento fakt je při reálném využití klíčový. Dále se potvrdilo, že výstup algoritmu reflektuje zadané hranice, ve kterých se mají hledat role a taktéž výstupní role oscilují kolem optimální velikosti zadané uživatelem.

Samotný algoritmus má velmi flexibilní strukturu a může být v budoucnu vylepšen případně upraven pro konkrétní potřeby.

14. Bibliografie

[1] R. Sandhu a Samarati, „Access control: principle and practice,“ *IEEE Comunications Magazine,* 1994.

[2] M. Šlancar, „Bezpečnost zaměstnanecké identity,“ [Online]. Available: http://computerworld.cz/securityworld/bezpecnost-zamestnanecke-identity-52280.

[3] B. Jayant, S. Ubale, S. Apte a D. Modani, „Analysis of DAC MAC RBAC Access Control based Models for Security,“ *International Journal of Computer Applications (0975 – 8887),* 2015.

[4] R. Sandhu, H. Feinstein a C. Youman, „Role-Based Access Control Models,“ *IEEE Computer,* 1996.

[5] Sandhu, Ferraiolo a Kuhn, „The NIST model for role-based access control: towards a unified standard,“ *ACM workshop on Role-based access control.*

[6] D. Zhang, K. Ramamohanarao, R. Zhang a S. Versteeg, „Efficient Graph Based Approach to Large Scale Role Engineering,“ *TRANSACTIONS ON DATA PRIVACY,* 2014.

[7] L. Fuchs, M. Kunz a G. Pernul, „ROLE MODEL OPTIMIZATION FOR SECURE ROLE-BASED IDENTITY MANAGEMENT,“ 2014.

[8] J. Vaidya, V. Alturi a Q. Gui, „The Role Mining Problem: Finding a Minimal Descriptive Set or Roles“.

[9] J. Schlegelmilch a U. Steffens, „Role Mining with ORCA,“ *SACMAT’05,* 2005.

[10] J. Vaidya, V. Atluri a J. Warner, „RoleMiner: Mining Roles using Subset Enumeration∗,“ *CCS'06,* 2006.

[11] R. Li, H. Li, W. Wang, X. Ma a X. Gu, „RMiner: A Tool Set for Role Mining,“ 2013.

[12] A. C. O'Connor a R. J. Loomis, „2010 Economic Analysis of Role-Based Access Control,“ *National Institute of Standards and Technology,* December 2010.

[13] Science Applications International Corporation (SAIC), „RBAC Role Engineering Process,“ 2004.

[14] M. Klíma, „RBAC Algorithm GitHub,“ [Online]. Available: https://github.com/Pooky/rbac-algorithm.