ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

| | |
|---|---|
| **Název:** | Optimalizace importu dat o uživatelích a skupinách do systému Perun |
| **Student:** | Jan Zvěřina |
| **Vedoucí:** | RNDr. Michal Procházka, Ph.D. |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | Do konce letního semestru 2017/18 |

## Pokyny pro vypracování

Seznamte se se systémem Perun, který spravuje uživatele, skupiny a přístupy ke službám. Systém umožňuje importovat informace o uživatelích a skupinách z externích systémů. Současná implementace těchto importů je velice naivní. Proveďte analýzu požadavků na importy a navrhněte nový způsob, který bude brát v potaz plné, změnové a inkrementální synchronizace. Hlavním kritériem je rychlost a minimalizace zatížení na straně systému Perun, ale také na straně zdroje dat pro import. Pro demonstraci nově navrženého systému importů proveďte také ukázkovou implementaci.

## Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 1. listopadu 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

# Optimalizace importu dat o uživatelích a skupinách do systému Perun

***Jan Zvěřina***

Vedoucí práce: RNDr. Michal Procházka, Ph.D.

15. května 2017

# Poděkování

Chtěl bych poděkovat vedoucímu bakalářské práce panu RNDr. Michalu Procházkovi, Ph.D. a dále Mgr. Slávku Licehammerovi za všechny rady a konzultace během vytváření této bakalářské práce. Také bych chtěl poděkovat všem dalším členům týmu pracujícím na systému Perun za trpělivé zodpovídání dotazů.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2017 .....

### Odkaz na tuto práci

Zvěřina, Jan. *Optimalizace importu dat o uživatelích a skupinách do systému Perun*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

# Abstrakt

Bakalářská práce se zaměřuje na optimalizaci importu dat o uživatelích a skupinách do systému Perun, který spravuje uživatelské účty, skupiny a přístupy ke službám. Import uživatelů a členů skupin se provádí z externích zdrojů typu relační databáze, webová služba, soubor typu XML či CSV atd. Aktuální implementace těchto importů je v mnoha případech naivní a pomalá. Pokud aktualizujete byť jen jediný údaj o uživateli v externím zdroji, tak se přesto pošlou všichni uživatelé z dané skupiny při provádění synchronizace. Cílem práce je seznámit se se systémem Perun, analyzovat současný stav importu dat, navrhnout nové optimalizované řešení a vytvořit ukázkovou implementaci. Práce obsahuje analýzu, návrh, implementaci a test výkonu optimalizované synchronizace. Byla naprogramována v Javě a zefektivnila synchronizační proces skupin a uživatelů.

**Klíčová slova** Systém Perun, import a export dat, uživatelé, skupiny, synchronizace systémů, optimalizace

# Abstract

Bachelor thesis is focused on optimalisation of user and group data import into Perun system. Perun system manages user accounts, groups and access to services. Import of users and group members is done from external sources like relational database, web service, XML or CSV file etc. Current implementation of these imports is in many cases naive and slow. If you will update just one user in an external source, then all members of group will still be sent in group synchronization. The goal of the thesis is to learn how Perun works, to analyze the current state of data import, to design a new optimised solution of data import and to create an implementation. The thesis includes analysis, design, implementation and performance test of optimised synchronization. It has been programmed in Java and it optimised the synchronization process for groups and users.

**Keywords** Perun system, import and export of data, members, groups, synchronization of systems, optimalisation

# Obsah

# Seznam obrázků

# Úvod

Systém Perun slouží ke správě uživatelů, kteří jsou dále rozděleni do skupin a virtuálních organizací. Těmto skupinám se následně přiřazují přístupy k službám, které mohou používat. Můžete si pod tím představit například přidělení přístupu k datovému úložišti, jehož část můžou poté všichni členové skupiny používat.

V systému Perun se ale uživatelé nevytvářejí ručně. Importují se z různých externích zdrojů typu relační databáze, LDAP server, webová aplikace atd. V případě importu skupiny uživatelů se tomuto procesu říká synchronizace skupiny. Provádění synchronizace skupiny je ale v mnoha případech neefektivní. Například i když se žádná data o uživatelích na straně externího zdroje nezměnila, tak se i tak při další synchronizaci skupiny pošlou všechna data o uživatelích. Ta se následně musí zpracovat, což stojí systém Perun spoustu zbytečného času. Hlavním cílem této bakalářské práce je celý tento proces optimalizovat. Potřeba optimalizace synchronizace skupin vznikla kvůli vzrůstajícímu množství uživatelů a skupin, které aktuálně různé instance systému Perun spravují.

Toto téma jsem si vybral z důvodu práce na fungujícím systému, který se používá v reálném světě. Nejedná se tedy jen o školní projekt, který se jinak než jako bakalářská práce nikdy nevyužije.

## Cíl práce

Cílem práce je optimalizace importu dat o uživatelích ve skupinách do systému Perun. Tomuto procesu se říká synchronizace skupin. Musí se analyzovat externí zdroje, ze kterých se data o uživatelích importují. Tím se zjistí možnosti, které externí zdroje poskytují pro detekci změn v uživatelských datech. Na základě získaných informací se musí navrhnout řešení, které optimalizuje komunikaci s externími zdroji. Díky tomu by měl systém Perun v určitých případech podporovat získávání pouze modifikovaných uživatelů.

Nejprve je ale nutné seznámit se se systémem Perun, analyzovat jeho možnosti a pochopit mechanismy, které používá pro získávání dat z externích zdrojů. Dalším cílem této bakalářské práce je navrhnutí optimalizace zpracování získaných dat na straně systému Perun. Po návrhu optimalizovaného řešení je cíl toto řešení demonstrativně implementovat.

## Struktura práce

Text práce je členěn do čtyř kapitol. První kapitola se zabývá analýzou systému Perun a všemi typy používaných externích zdrojů. Řeší průběh synchronizace skupin a zkoumá možné způsoby optimalizace. Druhá kapitola se zabývá návrhem optimalizované komunikace s externími zdroji a efektivnějším zpracováním získaných dat v systému Perun. Třetí kapitola se zabývá implementací navržených řešení a popisuje problémy, které při implementaci nastaly. Závěrečná čtvrtá kapitola se zabývá testováním implementovaného řešení a vyhodnocením výsledků.

Kapitola 1

# Analytická část

## 1.1 Obecný úvod do problematiky

Hlavním tématem, kterým se zabývá tato bakalářské práce, je synchronizace dat. Synchronizace dat je *„proces, který zabezpečuje koordinaci dat mezi dvěma či více systémy.“* [3] Jinými slovy synchronizace umožňuje přenos změn v datech mezi systémy. Změny, které jsou provedeny v jednom systému, se propagují na další připojené systémy.

U procesu synchronizace vždy záleží na tom, která data mají být synchronizována a které systémy budou data vysílat či přijímat. Také záleží na tom, jak často se změny mají promítat. Data se například mohou synchronizovat napříč všemi propojenými systémy, kdy změnu v datech může provést jakýkoliv z nich a tato změna se propaguje na všechny ostatní. Nebo je možné zvolit primární zdroj dat, ze kterého se budou provádět propagace změn do ostatních podřízených systémů. Podobně to funguje v systému Perun, který zastává roli podřízeného systému a data o uživatelích a skupinách získává z primárních zdrojů dat, ale o tom později.

Při synchronizaci dat může nastat mnoho problémů. Jedním z nich jsou kolize, kdy se na více zařízeních upraví stejný soubor a ten se pak musí spojit do jednoho výsledku, aby obě strany byly spokojeny. Tento problém řeší například verzovací systém Git nebo SVN.

Dále je tu problém s množstvím posílaných dat. Při synchronizaci je nežádoucí posílat ta data, která cíli neposkytnou žádnou novou informaci. Ideálně by se měla posílat jen modifikovaná data. Myslí se tím například modifikované soubory či nejideálněji změněné části souborů nebo například ty záznamy z databázové tabulky, které byly upraveny, přidány či odstraněny.

U synchronizace systémů se také musí vyřešit otázka, jak často se mají data synchronizovat. V ideálním světě by se všechny změny promítali okamžitě, to ale bohužel není v mnoha případech možné. Velice záleží na možnostech zdroje dat, kdy spousta systémů není schopna poslat jen modifikované záznamy (např. o uživatelích ve skupině) a musíte tedy na cílové stanici zpra-


Obrázek 1.1: Logo systému Perun [1]

covat zbytečně velké množství dat (všechny uživatele ve skupině namísto pár modifikovaných), což zatěžuje jak síť, tak cílový systém. Proto je u takových zdrojů dat nutné nastavit rozumný interval pro získávání dat.

## 1.2 Představení systému Perun

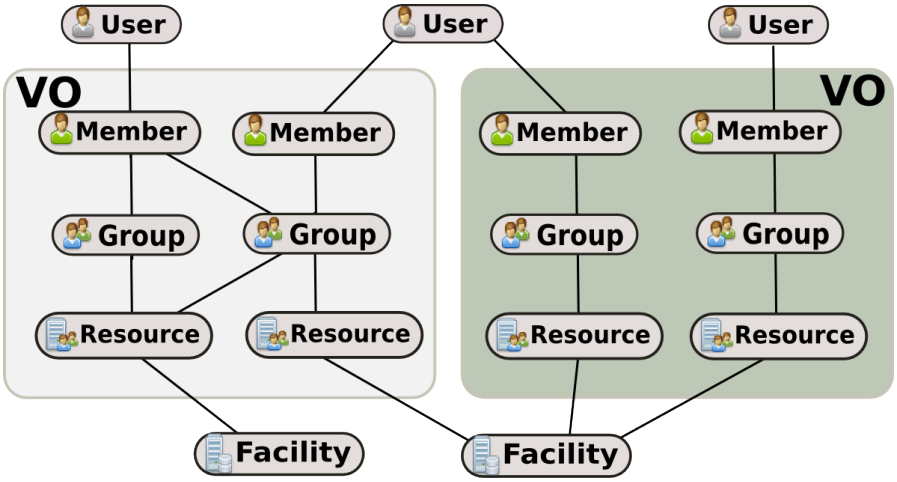
*„Systém Perun je vyvíjen ve spolupráci zájmového sdružení právnických osob CESNET a národního centra CERIT-SC ve spolupráci se studenty z Masarykovy univerzity v Brně.“* [4]

Bakalářská práce se zabývá úpravou systému Perun. Proto je dobré si ho představit a seznámit se s jeho principem a možnostmi. To je důležité pro pochopení souvislostí a smyslu této práce. *„Perun slouží pro správu uživatelů (users) a řízení přístupu ke službám (facilities). Tyto služby představují např. datová úložiště.“* [5]

Pro lepší představu si čtenář může představit, že v systému Perun existuje skupina uživatelů a této skupině se přidělí právo na využívání služeb určitého datového úložiště. Systém Perun zpropaguje tyto uživatele na datové úložiště, kde jim podle nastavení skupiny vytvoří účty, vyhradí místo na disku a povolí přístup (nejčastěji pomocí SSH klíče). Po úspěšném dokončení propagace mohou všichni členové skupiny využívat datové úložiště.

To bylo zjednodušené vysvětlení toho, jak systém Perun funguje. Systém Perun se dále skládá z virtuálních organizací (virtual organizations) a skupin (groups). Pod pojmem virtuální organizace si můžete představit například zaměstnance fakulty, lidi pracující na stejném projektu či jakoukoliv jinou větší skupinu lidí, kterou spojuje například budova, ve které pracují. Má definovaná pravidla, která uživatel musí splnit a souhlasit s nimi, aby mohl být do virtuální organizace přidán. Skupiny jsou součástí virtuálních organizací. Slouží k rozdělení členů virtuální organizace do skupin, které spojuje zaměření práce či například místnost výkonu práce, v případě fakulty např. katedra či oddělení v případě firmy. Do jedné skupiny se většinou přidělují lidé, kteří budou využívat stejný zdroj (resource) určité facility (služby). Z každého uživatele, jež dostane přístup do virtuální organizace, se stává člen (member) virtuální organizace.

Jak již bylo zmíněno, facility představuje zařízení poskytující určitou službu




Obrázek 1.2: Konceptuální schéma znázorňující propojenost entit v systému Perun [2]

(např. datové úložiště). Tím se myslí ale celá služba jako celek. Pro definování využitelné části tohoto celku je potřeba vytvořit resource (zdroj). Ten definuje, jaké všechny služby tohoto zařízení budou moci přiřazení členové skupiny využívat včetně různých omezení (např. vyčlenění pěti terabytů místa na datovém úložišti). Takto nastavený zdroj se poté přidělí určité virtuální organizaci, kde se následně přidělí skupinám. Ty ho následně mohou využívat. Každý zdroj může být přidělen vždy jen právě jedné virtuální organizaci, ale více skupinám uvnitř dané virtuální organizace.

Pro správu služeb, virtuálních organizací a skupin existují v systému Perun manažeři. Manažer služby (facility manager) vytváří zdroje a definuje služby na svém zařízení. Dále přiděluje virtuálním organizacím zdroje, přičemž je nepřiděluje přímo skupinám ve virtuální organizaci. Manažer virtuální organizace (VO manager) spravuje členy své virtuální organizace a rozděluje je do skupin. Dále těmto skupinám přiděluje zdroje, které dostala VO přidělené od manažera služby. Tito manažeři se vždy předem domluví na podmínkách využití přidělovaného zdroje. Manažer skupiny (group manager) spravuje členy skupiny a může editovat přijímací formulář, který musí uživatel vyplnit před vstupem do skupiny.

Další důležitou entitou v systému Perun jsou atributy. Atributy uchovávají informace vázané vždy na nějakou entitu či na dvojici entit. Jedná se například o emailovou adresu vázanou na uživatele. Dále se může atribut vázat například na vztah mezi členem virtuální organizace a skupinou (email člena v rámci skupiny). Všechny informace o entitách v systému Perun jsou uloženy v atributech. To je výhoda, protože pokud je potřeba uchovávat novou informaci o určité entitě (např. telefonní číslo uživatele), tak kvůli tomu není potřeba vytvářet nový sloupec tabulky v databázi, ale stačí vytvořit nový atri-

but. Jak jsou jednotlivé entity v systému Perun na sebe navázány znázorňuje obrázek 1.2.

V [6] jsou uvedeny následující typy atributů v systému Perun. Jsou to *defined*, *optional*, *core* a *virtual* atributy. Defined atributy mohou vytvořit pouze správci systému Perun. Hodnota takových atributů je kontrolována atributovými moduly, které se starají o kontrolu správnosti vkládaných hodnot a také o automatické vyplňování. Defined atribut tedy musí být buď prázdný, nebo musí splňovat určená pravidla. Optional atributy nejsou striktně kontrolovány pomocí atributových modulů a používají je správci VO a služeb pro uchovávání volitelných atributů. Core atributy umožňují přístup k základním údajům entit podle jména atributu. Virtual atributy získávají svou hodnotu z kombinace jiných atributů a nemají tedy vlastní hodnotu. Pro tyto atributy existuje modul pro virtuální atributy, který definuje pravidla pro získání jejich hodnoty.

Tato bakalářské práce se ale zabývá importem dat o uživatelích, kteří jsou rozděleni do skupin. Jak již bylo řečeno, Perun v sobě uchovává množinu informací (atributů) o skupinách a uživatelích. U skupin jsou v rámci této práce zajímavé hlavně atributy definující způsob komunikace s externím zdrojem, ze kterého se získávají aktuální data o uživatelích. Externí zdroj je popsán v části 1.5.

Synchronizace s externími zdroji nyní funguje ve většině případech naivním způsobem, který je neefektivní jak na datový přenos, tak na čas, za který se změny z externího zdroje promítnou do systému Perun. Některé externí zdroje již mají implementovaná určitá vylepšení, ale to je menšina. Cílem této bakalářské práce je synchronizační proces zefektivnit.

Sluší se dále zmínit, že systém Perun je naprogramován v jazyku Java (jeho jádro, další moduly připojené k jádru jsou napsané např. v jazyku Perl). Je postaven na třívrstvé architektuře. Prezentační vrstva slouží ke kontrole, zda uživatel systému Perun s určitou rolí má dovoleno používat volanou metodu. Business vrstva slouží k provádění algoritmů a práci s daty. Poslední datová vrstva slouží k práci s databází (ukládání a načítání dat). Každá entita v systému Perun má svého manažera pro práci s ní. Na každé vrstvě se tito manažeři skládají z rozhraní (interface) a samotné implementace metod.

## 1.3 Externí zdroj

Externí zdroj je entita v systému Perun popisující způsob, jakým se má Perun spojit se zdrojem dat o uživatelích ve skupině (např. url, hostname, login, heslo). Určuje, jaká data z něj potřebujeme o uživatelích získat a jak se mají získaná data přetransformovat na atributy v systému Perun. Externí zdroj se nejprve musí přiřadit virtuální organizaci, do které se importovaní uživatelé budou vkládat. Pod pojmem externí zdroj si můžete představit např. relační

databázi, LDAP server, soubor ve formátu XML či CSV nebo webovou aplikaci s určitým podporovaným rozhraním (Google groups, VOOT, Unity).

Z externího zdroje se mohou importovat jednotliví uživatelé nebo skupina uživatelů. Import jednotlivých uživatelů se provádí ručně, zatímco skupiny jsou synchronizovány zpravidla automaticky. Pro jednoznačnou identifikaci uživatele slouží jeho uživatelské jméno (login) a externí zdroj, ze kterého je importován. Tato dvojice je uložená do uživatelského externího zdroje (*user external source, zkráceně UES*), který je vázán k uživateli. To znamená, že jeden uživatel může být synchronizován i z více externích zdrojů a může k němu být vázáno více uživatelských externích zdrojů.

Všechny informace o konkrétních externích zdrojích jsou popsány v *XML souboru* uloženém na serveru se systémem Perun. U každého takového externího zdroje je popsáno, jakého typu je (LDAP, SQL, ...), adresa serveru, přístupové údaje, dotaz, kterým se získají data o uživatelích a také mapování jednotlivých informací o uživatelích z externího zdroje na atributy v systému Perun. Například LDAP atribut *sn* přiřazený atributu *lastName* v systému Perun. [7]

Pro ukázku je zde uvedena zjednodušená ukázka definice externího zdroje. Obsahuje jméno externího zdroje, typ (v tomto případě LDAP) a popis. Dále obsahuje atribut *url*, který definuje server, se kterým bude systém Perun komunikovat. Atribut *base* slouží pro definici adresáře s uživateli. Atributy *user* a *password* slouží k přihlášení k účtu určenému pro systém Perun. *Query* se používá pro získání větší množiny uživatelů ze skupiny včetně atributů. Atribut *loginQuery* získá všechny informace o jednom uživateli. Za otazník se dosadí vyhledávaný řetězec (např. login). Dále je zde uveden atribut *ldapMapping* obsahující již výše zmíněné mapování LDAP atributů na atributy v systému Perun.

Zdrojový kód 1.1: Ukázka XML definice externího zdroje LDAP

```
<extSource>
 <name>ldap-test-source</name>
  <type>cz.metacentrum.perun.core.impl.ExtSourceLdap</type>
  <description>Popis ext. zdroje</description>
  <attributes>
   <attribute name="url">ldaps://ldap.domain.com</attribute>
   <attribute name="base">ou=people,dc=domain,dc=com</attribute>
   <attribute name="query">(uid=?)</attribute>
   <attribute name="loginQuery">(uid=?)</attribute>
   <attribute name="referral">follow</attribute>
   <attribute name="user">perun_user_record</attribute>
   <attribute name="password">password</attribute>
   <attribute name="ldapMapping">
    firstName={givenName},
    lastName={sn},
    login={uid},
    urn:perun:member:attribute-def:def:mail={preferedMail}
```

```
    </attribute>
  </attributes>
</extSource>
```

Externí zdroje mají dva typy rozhraní: *ExtSourceSimpleApi* (jednoduché rozhraní) a *ExtSourceApi* (složitější rozhraní). Z externích zdrojů s jednoduchým rozhraním se data o uživatelích získavají pomocí dvou dotazů. Nejprve se ze zdroje dat získají všechna uživatelská jména všech členů skupiny. Následně se pro každé uživatelské jméno provede dotaz na zdroj dat, který získá všechny potřebné atributy o daném uživateli. V současnosti používá toto rozhraní pouze externí zdroj pro *SQL* (z historických důvodů, v systému Perun existuje i externí zdroj *SQLComplex* používající rozhraní složitější, ale o tom až dále).

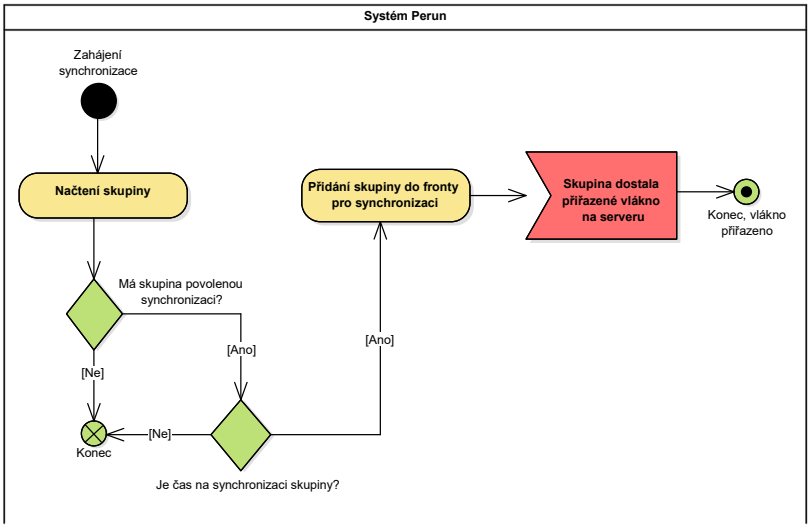
Oproti tomu externí zdroj se složitějším rozhraním je schopen získat data o všech uživatelích ve skupině včetně všech potřebných atributů jedním dotazem. Díky tomu je celý proces importu uživatelů do skupiny rychlejší. Toto rozhraní používají všechny ostatní externí zdroje jako XML, CSV, SQLComplex, Google, VOOT atd. Používá ho také LDAP, i když v principu pracuje jako zdroj s jednoduchým rozhraním. O tom ale až v detailním popisu jednotlivých externích zdrojů v kapitole 1.5.

## 1.4 Současné řešení importu dat o uživatelích

Každá skupina, která chce být synchronizována s Perunem, musí mít kromě přiřazení externího zdroje také vyplněný atribut *synchronization enabled* (povolení synchronizace) na hodnotu *true*. Atribut *synchronization interval* (synchronizační interval) je číslo (v minutách), které po vynásobení pěti udává velikost intervalu mezi jednotlivými synchronizacemi skupiny. Perun spouští synchronizační proces každých pět minut, proto je zadané číslo násobeno pěti.

Po spuštění synchronizačního procesu Perun pro každou evidovanou skupinu zkontroluje, zda má povolenou synchronizaci. Jestliže ano, tak zkontroluje, zda je na synchronizaci čas. Pokud ano, tak tuto skupinu Perun přidá do fronty skupin připravených k synchronizaci. Tuto frontu následně převezme třída starající se o přidělování procesorových vláken dostupných na serveru se systémem Perun. Jakmile se některé vlákno uvolní, přiřadí ho další skupině. Tento proces znázorňuje obrázek 1.3.

Synchronizace skupiny se spustí ve chvíli, kdy dostane přiděleno procesorové vlákno. Perun se spojí s externím zdrojem a získá z něj data o uživatelích. Proces získávání dat záleží na tom, jaké má externí zdroj rozhraní, viz sekce 1.3. Podrobně popsaný průběh synchronizace skupiny včetně zpracování dat je popsán v části 1.4.2. Implementační řešení synchronizace skupin včetně komunikace mezi třídami v systému Perun je popsáno v následující části 1.4.1.




Obrázek 1.3: Diagram aktivity znázorňující proces přiřazení vlákna skupině

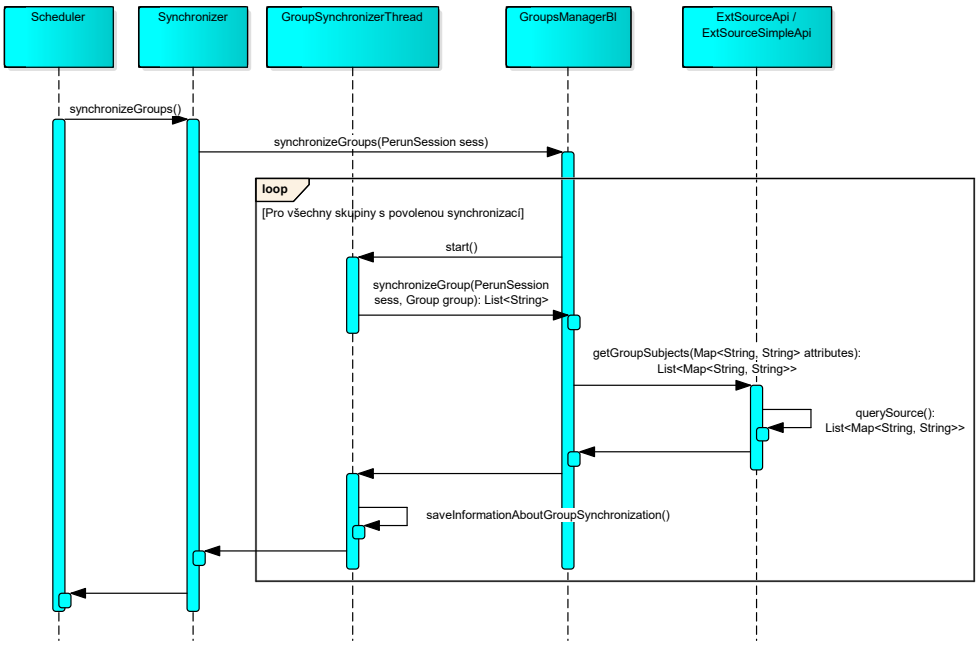
### 1.4.1 Implementační řešení procesu synchronizace

*Spring Scheduler* (plánovač automaticky spouštěných akcí) volá každých pět minut metodu *synchronizeGroups* třídy *Synchronizer*. Ta zavolá metodu *synchronizeGroups* manažera skupin na business vrstvě (*GroupsManagerBl*). Ta zkontroluje, jestli je opravdu čas na další synchronizační cyklus a pokud ano, tak započne synchronizovat všechny skupiny v systému Perun, které mají povolenou synchronizaci a u kterých je na ni čas.

V cyklu se postupně skupinám přidělují jednotlivá volná procesorová vlákna. Pokud je vlákno volné, tak se inicializuje třídou představující vlákno pro synchronizaci skupiny *GroupSynchronizerThread*. V této třídě se zavolá metoda manažera skupin *synchronizeGroup* pro synchronizaci jedné skupiny. Dále proběhne série operací podrobně popsaná v části 1.4.2. Nás v této chvíli zajímá, že se připraví atributy o skupině do mapy řetězců a tato mapa se v parametru předá metodě *getGroupSubjects(Map<String, String> attributes)* určitého externího zdroje.

Následná komunikace se zdrojem dat je závislá na typu externího zdroje. Nám bude pro příklad stačit, že se může zavolat metoda *querySource*, která nám získá data o členech skupiny. Získaná data se vrátí metodě *synchronizeGroup* manažera skupin, kde se zpracují a členové skupiny se aktualizují.

Následně se třídě pro práci s vláknem *GroupSynchronizerThread* vrátí seznam uživatelských jmen členů, kteří byli z nějakého důvodu přeskočeni a nemohli být synchronizováni. Tato třída tyto loginy zaznamená do logu spolu s dalšími informacemi o proběhlé synchronizaci (délka, výpis chybových hlášek atd.). Celý průběh je zobrazen v sekvenčním diagramu na obrázku 1.4.




Obrázek 1.4: Sekvenční diagram popisující průběh synchronizačního cyklu

### 1.4.2 Průběh synchronizace skupiny

Po započatí synchronizace skupiny si systém Perun načte do seznamu všechny aktuální členy skupiny spolu s jejich atributy. Také si z definice externího zdroje načte atribut *overwriteUserAttributes*, který obsahuje seznam uživatelských atributů k přepsání. Nepleťte si *člena (member)* a *uživatele (user)*. Uživatel může být členem ve více skupinách a virtuálních organizacích. Takový uživatel má jako člen skupin různé členské atributy a také má uživatelské atributy (např. jméno, příjmení, seznam ssh klíčů). Takové atributy mohou být díky atributu *overwriteUserAttributes* z určitého externího zdroje vždy přepsány na správnou hodnotu, pokud by je někdo v systému Perun ručně změnil.

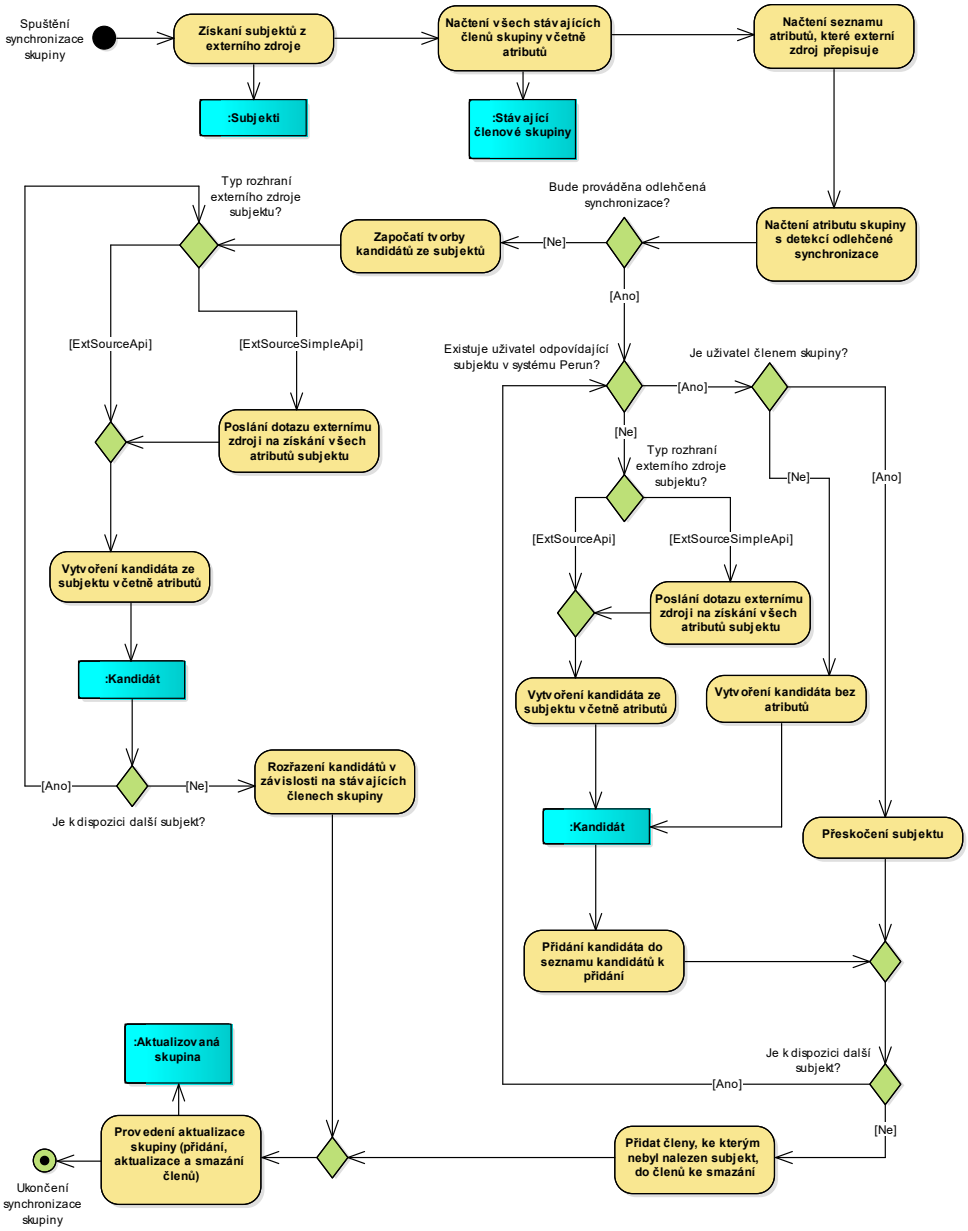
Dále metoda pro synchronizaci zjistí, zda se jedná o *odlehčenou synchronizaci (lightweight synchronization)*. Odlehčená synchronizace znamená, že se členové skupiny pouze přidávají a odebírají. Neprobíhá aktualizace atributů u stávajících členů. Díky tomu je takto prováděná synchronizace výrazně rychlejší než plná synchronizace, provádějící i aktualizaci stávajících členů. Tento typ synchronizace se používá u podskupin již existujících skupin, kdy potřebujeme členy pouze rozdělit a přidělit jim specifické služby.

Dalším krokem je získání dat o členech z externího zdroje skupiny. Všechny externí zdroje jsou naprogramovány tak, aby vygenerovaly ze získaných dat o uživatelích ve skupině seznam map, kde každá mapa představuje jednoho

člena skupiny a říká se jí *subjekt (subject)*. Mapa obsahující jeden *subjekt* se skládá z klíčů, které představují názvy atributů získaných o členovi, a jejich hodnot. Pokud externí zdroj používá jednoduché rozhraní, tak mapa obsahuje pouze atribut s uživatelským jménem. Pokud používá složité rozhraní, tak mapa obsahuje všechny potřebné atributy o uživateli.

U plné synchronizace se ze všech subjektů vytvoří kandidáti. *Kandidát* představuje uchazeče o členství ve skupině. Je to takový předstupeň člena obsahující atributy a uživatelské externí zdroje. *Uživatelský externí zdroj (user external source)* představuje objekt, který autentizuje uživatele externím zdrojem. Slouží k uložení externí identity uživatele. Pokud byl subjekt získán pomocí externího zdroje se složitějším rozhraním, tak obsahuje všechny potřebné atributy k vytvoření kandidáta. Pokud byl vytvořen externím zdrojem s jednoduchým rozhraním a subjekt obsahuje pouze uživatelské jméno uživatele v externím zdroji, tak se synchronizátor při vytváření kandidáta znovu spojí s externím zdrojem a získá z něj všechny potřebné atributy pomocí metody *getSubjectByLogin*. Následně se podle uživatelských externích zdrojů kandidátů snaží synchronizátor najít shodu s již existujícími členy skupiny. Pokud nalezne shodu, člen již existuje a je přidán do seznamu map členů k aktualizaci, kde klíč je kandidát a hodnota je člen skupiny s atributy (tzv. *richMember*). Pokud nenalezne shodu, tak se jedná o nového člena skupiny a kandidát je přidán do seznamu kandidátů k přidání. Všichni stávající členové, kteří nebyli spárováni s žádným kandidátem, byli v externím zdroji odstraněni ze skupiny a jsou přidáni do seznamu členů k odstranění ze skupiny v systému Perun. Následně se zavolají metody pro aktualizaci stávajících členů, přidání nových členů a odstranění již neexistujících členů.

Oproti plné synchronizaci je u odlehčené synchronizace změna v tom, že se nevytváří kandidáti ze všech subjektů. Nejprve se vytvoří mapa ze stávajících členů skupiny, kde klíč je identifikační číslo uživatele v systému Perun a hodnota je člen skupiny s atributy. Synchronizátor se každý subjekt na základě uživatelského jména a externího zdroje snaží spárovat s existujícím uživatelským externím zdrojem, pomocí něhož by našel uživatele v systému Perun. Pokud se mu to podaří a uživatele nalezne, zkontroluje, zda je uživatel s takovým identifikátorem v mapě stávajících členů skupiny. Pokud ano, tak tohoto uživatele přeskočí, protože ho nechce aktualizovat. Pokud ne, tak vytvoří kandidáta (jen s potřebnými informacemi bez atributů, co mohou být uloženy v subjektu) a přidá ho do seznamu kandidátů k přidání. Pokud vyhledávání uživatele skončí výjimkou a Perun takového uživatele vůbec nezná, tak je potřeba vytvořit úplně nového uživatele. Zavolá se tedy pro tento subjekt stejná metoda, která konvertovala subjekt na kandidáta u klasické synchronizace, aby získal v případě potřeby všechny atributy z externího zdroje. Následně je takový kandidát přidán do seznamu kandidátů k přidání. Všichni členové v mapě stávajících členů, které se nepodařilo spárovat s žádným subjektem, jsou přidáni do seznamu členů k odstranění. Dál už synchronizace probíhá stejně jako plná.




Obrázek 1.5: Diagram aktivity znázorňující proces aktualizace skupiny

Celý proces synchronizace skupiny a zpracování subjektů získaných z externího zdroje je znázorněn na obrázku 1.5. Proces získávání dat z různých externích zdrojů je popsán v části 1.5.

### 1.4.3 Nedostatky současného řešení

Problém současného řešení importu dat o uživatelích a skupinách do systému Perun je v tom, že se pokaždé posílá plný stav skupiny, byť se změnil jen jeden jediný záznam. Pokud to je např. skupina lidí z Masarykovi univerzity v Brně, která může mít tisíce uživatelů, tak databázový server této univerzity musí poslat zbytečně velké množství dat, což je náročné na datový přenos a čas. Systém Perun musí následně zpracovat veškerá přijatá data a vybrat z nich jen ta užitečná, což ho stojí výkon a hlavně čas. Kvůli tomu je pak vlákno zabráno delší dobu, i když by už mohlo zpracovávat další skupinu.

Nedostatek je také v délce intervalů pro synchronizaci. Například na instanci systému Perun, kterou spravuje CESNET, mají skupiny nastaven synchronizační interval na sedmdesát minut a některé i na šest hodin. Pokud by tedy přišel nový zaměstnanec do práce, zaměstnanec z personálního oddělení by ho zaregistroval a on by byl přiřazen do skupiny s šestihodinovým synchronizačním intervalem, tak by se skoro celý den nepřipojil na služby obsluhované Perunem. To se dá vyřešit pomocí funkce Peruna *force synchronization*, která synchronizaci vynutí ihned, ale to není úplně ideální způsob vyřešení tohoto problému. Takto dlouhé intervaly jsou na CESNETí instanci systému Perun nastaveny kvůli množství skupin, které obsluhuje. Rozhodně není žádoucí tyto intervaly odstranit úplně, protože je to u spousty skupin zbytečné. Tato bakalářská práce by ale měla usilovat o to, aby se intervaly u skupin, kde je to potřeba, mohli zkrátit.

Dalším nedostatkem je způsob, jakým se v současné implementaci synchronizace skupin určuje, zda již uživatel v systému Perun existuje a pokud ano, zda je členem skupiny. Tato bakalářská práce by měla najít způsob, jakým provádět vyhledávání existujících uživatelů efektivněji.

## 1.5 Analýza externích zdrojů

Tato část popisuje, jak systém Perun v současnosti komunikuje se všemi externími zdroji a jakým způsobem se následně zpracovávají získaná data, ze kterých se vytváří seznam subjektů.

### 1.5.1 SQL a SQLComplex

Externí zdroje *SQL* a *SQLComplex* slouží pro komunikaci s relační databází. Relační databáze představuje především tabulky, kde se každý řádek tabulky chápe jako jeden záznam. Pro komunikaci s relační databází se používá jazyk *SQL (Structured Query Language). „SQL je jazyk sloužící k ukládání dat,*

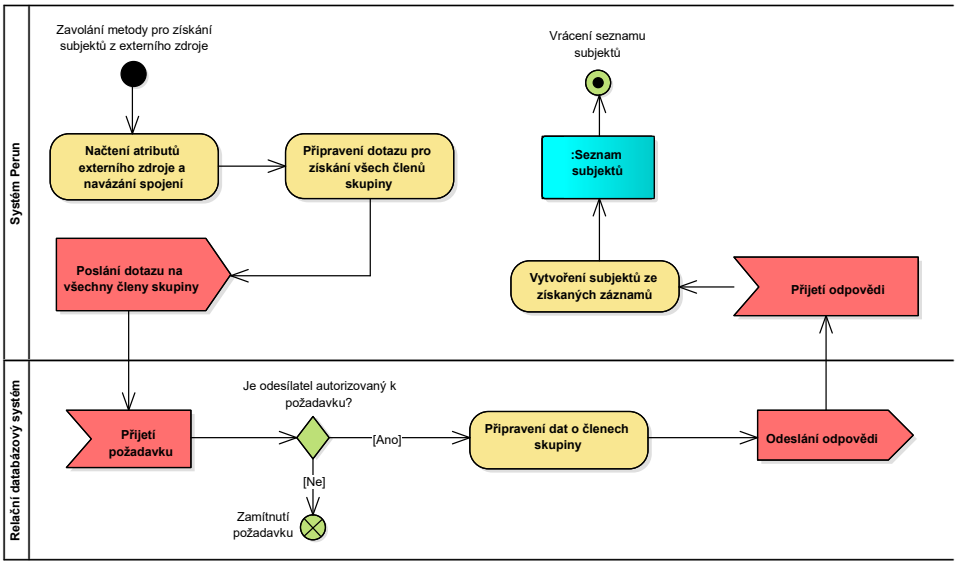
*manipulaci s daty a získávání dat z relačních databází.*“ [8]. Systém Perun potřebuje získat z takové databáze atributy všech uživatelů ve skupině. K tomu slouží příkaz *SELECT*, který dokáže spojit záznamy z více tabulek do jedné a podle výběrových podmínek vybere správné záznamy, které představují uživatele ve skupině. Spousty relačních databází má data o uživatelích uložená ve více tabulkách, kvůli čemuž jsou příkazy *SELECT* občas velmi složité. Příkaz *SELECT* podporuje přejmenovávání názvů sloupců tabulky, díky čemuž se mapování názvů atributů na ty používané v systému Perun může provést přímo v něm a není uvedeno v definici externího zdroje.

Externí zdroj *SQL* využívá jednoduché rozhraní. Možná se ptáte, proč systém Perun podporuje dva typy externích zdrojů pro SQL jazyk, když příkaz *SELECT* umožňuje získat všechna data o uživatelích jedním dotazem? Pravda je taková, že tato naivnější varianta externího zdroje, která nejprve získá uživatelská jména členů skupiny a následně se doptává na atributy uživatelů pro každý login zvlášť, existuje kvůli historickým důvodům. První verze systému Perun získávala data tímto naivním způsobem a ten se musel zachovat kvůli zpětné kompatibilitě.[1] *SQLComplex* používá rozhraní složitější. To znamená, že všechna data o uživatelích včetně jejich atributů získá jedním příkazem *SELECT*. Používání tohoto externího zdroje je tedy výrazně rychlejší, než používání jeho jednoduššího bratříčka.

Při započetí synchronizace s takovými externími zdroji se nejprve načtou atributy z jejich definice. Podle atributu *driver* zjistíme, jaký ovladač pro relační databáze (např. Oracle, PostgreSQL, MySQL) bude použit pro komunikaci s externím zdrojem. Atribut *url* specifikuje, na jaké adrese najdeme databázi, ke které budeme přistupovat. Poté navážeme s databází spojení, přičemž pošleme i hodnoty atributů *user* a *password* pro autentizaci. Aby systém Perun mohl přistupovat k databázi a potřebným tabulkám, je nejprve potřeba vytvořit mu v databázi účet s potřebnými právy a ten poskytnout správcům instance systému Perun, aby ho zadali do definice externího zdroje. Bohatě stačí právo k vykonávání příkazu *SELECT*, systém Perun data pouze čte, nezapisuje. Celý proces získání dat z externího zdroje typu SQL či SQLComplex je znázorněn na obrázku 1.6.

Relační databáze podporuje ukládání času poslední změny u všech záznamů. To může pomoc při vyhledávání modifikovaných členů skupiny. Dále podporuje výpočet kontrolního součtu. Ten by mohl být využit k zjištění, zda se množina vybraných záznamů od předchozí synchronizace skupiny změnila.

[1] Perun není vydáván jako většina programů ve verzích, kde by se mohlo určit, že od určité verze není externí zdroj podporovaný. Aktualizace Peruna se nahrávají do verzovacího systému GIT a testují se na vývojářské instanci. Po otestování se tato vylepšení postupně přidávají do větve (branch) systému GIT určené pro ostré nasazení.




Obrázek 1.6: Diagram aktivity znázorňující komunikaci s externím zdrojem SQL či SQLComplex

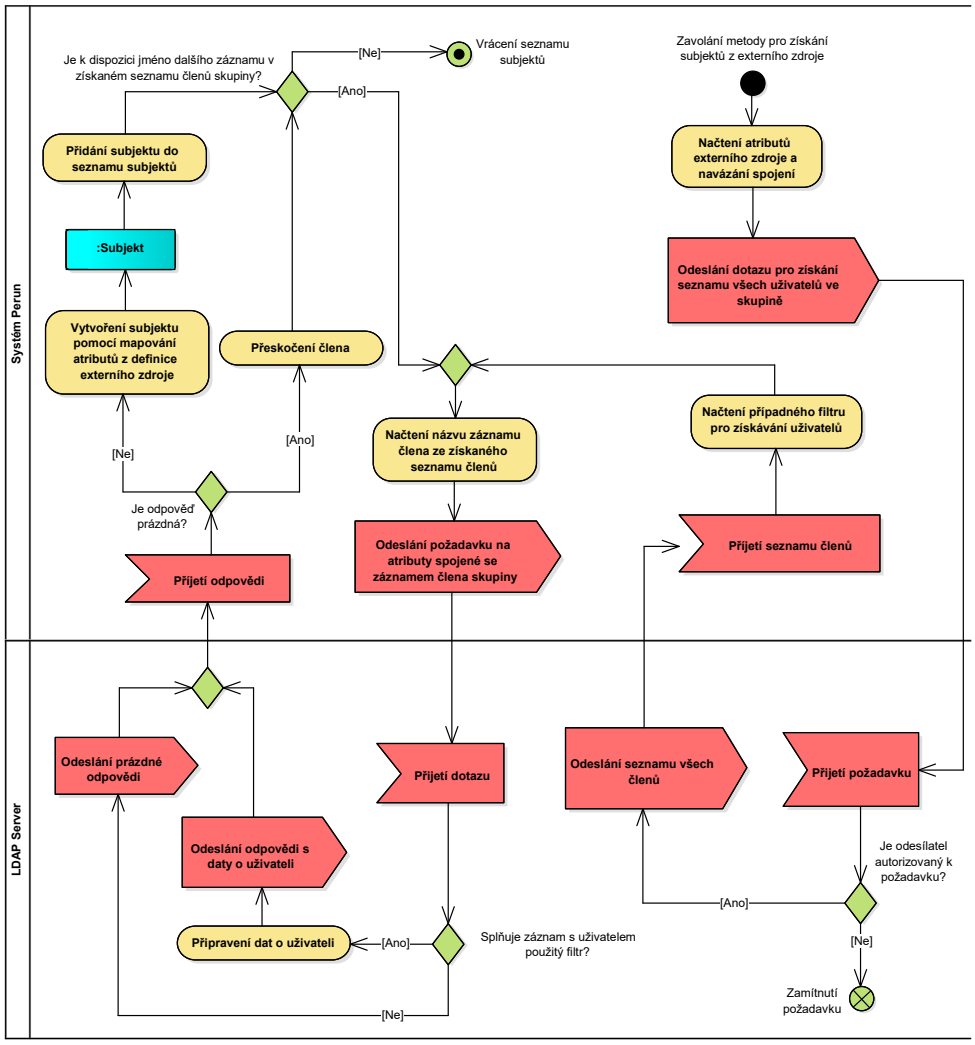
### 1.5.2 LDAP a EGISSO

LDAP (Lightweight Directory Access Protocol) je protokol pro komunikaci s adresářovou službou. Adresářová služba je aplikace shromažďující a organizující informace ve stromové struktuře. Je přizpůsobena hlavně pro čtení informací, zápisy se provádějí minimálně. [9]

Po započatí synchronizace se nejprve Perun spojí s externím zdrojem protokolem LDAP pomocí atributů: adresy *url* externího zdroje, dotazu pro získání uživatelských jmen uživatelů ve skupině *membersQuery* a přístupových údajů *user* a *password*. Dále načteme atribut *ldapMapping* obsahující mapování atributů adresářové služby na atributy v systému Perun.

Jako první se načte záznam se skupinou na externím zdroji. Ten mimo jiné obsahuje cestu ke všem záznamům o uživatelích ve skupině. Dále načteme atribut *filter* obsahující omezení pro výběr uživatelů ze skupiny. Pokud je tento atribut prázdný, použije se výchozí hodnota z atributu *filteredQuery* externího zdroje. Pokud chybí i ona, tak se nepoužije žádný filtr. Poté se v cyklu pro každé získané uživatelské jméno provede dotaz na externí zdroj, který získá všechny potřebné atributy o daném členovi. Ze získaných atributů se vytvoří subjekt a vloží se do seznamu. Tento seznam vrátíme synchronizátoru pro další zpracování. Celý proces získání dat z externího zdroje typu LDAP je znázorněn na obrázku 1.7.

Externí zdroj EGISSO se liší od LDAP tím, že je vytvořený na míru pro načítání uživatelů z LDAP serveru *EGI SSO (EGI Single Sign On)*. Je to




Obrázek 1.7: Diagram aktivity znázorňující komunikaci s externím zdrojem LDAP

z důvodu potřeby specifického načítání certifikátů vytvořeného na míru pro tento případ. Služba EGI SSO se využívá pro přihlašování jedním uživatelským jménem a heslem na všechny spravované služby. Dalším rozdílem oproti LDAP je ten, že uživatelé nejsou uvedeni ve skupině, ale jsou načítáni všichni členové tzv. organizační jednotky *(Organization Unit)*. Všechny informace o členech skupiny se tak získávají pouze jedním dotazem pro získání všech záznamů z dané organizační jednotky. [10]

Adresářová služba spravovaná protokolem LDAP umožňuje evidovat u záznamů čas poslední změny. Slouží k tomu například atribut *modifyTimestamp* používaný u *OpenLDAP* implementace protokolu LDAP nebo atribut *whenChanged* používaný u služeb používajících Active Directory. *Active Directory* je adresářová služba vyvinutá firmou Microsoft pro použití protokolu LDAP v prostředí Windows. [11] Díky času poslední změny lze zjistit, zda se množina uživatelů ve skupině změnila. Pokud by se nezměnila, tak by systém Perun mohl získat pouze ty záznamy uživatelů, které byly změněny od poslední synchronizace.

U externího zdroje EGISSO je problém s tím, že se získávají všichni členové z organizační jednotky a ti nejsou přiřazeni do žádné skupiny. Organizační jednotka ale nemění čas poslední změny v případě, kdy do ní někoho přidáte nebo z ní někoho odstraníte. Nemůže se tedy použít při návrhu řešení stejný přístup jako v případě externího zdroje LDAP, kde jsou uživatelé vždy přiřazeni do skupiny.

Z externího zdroje EGISSO by se mohly získat názvy všech záznamů s uživateli, ze kterých by se vypočítal kontrolní součet. Ten by se dále porovnal s tím z předchozí synchronizace. Zjistilo by se tak, jestli se množina uživatelů od předchozí synchronizace změnila a pokud ne, tak by se mohli získat pouze modifikovaní uživatelé upravení od poslední synchronizace. Problém s tímto řešením by byl ale ten, že LDAP samotný má problém s řazením. Odpověď s názvy všech záznamů s uživateli by se tedy musela zpracovat a seřadit v systému Perun. Až následně by bylo možné vypočítat kontrolní součet. To je příliš mnoho operací pro zjištění, zda se množina uživatelů v externím zdroji změnila a toto řešení se tudíž nevyplatí. Ve většině případů by se zjistilo, že se množina uživatelů změnila a musely by se z externího zdroje získat všechny atributy všech uživatelů v organizační jednotce, což by proces v konečném důsledku zpomalilo. Proto se komunikace s externím zdrojem EGISSO upravovat nebude.

### 1.5.3 CSV soubory

*CSV (Comma Separated Values)* formát je používaný k importu a exportu dat z různých aplikací. Obsahuje řádky, kde jsou jednotlivé pole od sebe odděleny čárkou. Každý řádek představuje jeden záznam. První řádek je hlavička, ve které jsou uvedeny názvy jednotlivých polí. [12]

V současnosti se import uživatelů do skupiny z CSV souborů provádí pouze ručně, kdy se CSV soubor nahraje na server se systémem Perun a spustí se synchronizace skupiny. Proto tento externí zdroj nemá metody pro navázání spojení, nebylo toho prozatím zapotřebí. Systémy pro export dat využívají běžněji formáty XML či JSON. Každý řádek v CSV souboru (kromě prvního) představuje jednoho uživatele. Nejprve se načte dotaz z atributu *query*, který bude sloužit pro kontrolu, zda jsou uživatelská data v pořádku a mají všechny potřebné atributy. Dále se načte cesta k souboru z atributu *file* a soubor s daty otevřeme.

Ze souboru se načte hlavička a dále se v cyklu načtou všechny řádky s členy skupiny. Každý načtený řádek se porovná s hlavičkou, zda má stejný počet polí. Dále se řádek porovná s dotazem z atributu *query* pro zjištění, zda je řádek ve správném formátu. Pokud ano, tak se zkonvertuje do subjektu a přidá se do seznamu subjektů. Po zpracování všech řádků se tento seznam vrátí synchronizátoru pro další zpracování.

Možné vylepšení importu dat z CSV je uložení kopie souboru z předchozí synchronizace a jeho porovnání s nově přijatým souborem. Pokud by nebyly nalezeny změny, synchronizace by byla ukončena. Při nalezení změn by se z uživatelů vytvořily subjekty a rozdělily se do tří skupin: přidaní, odebraní a modifikovaní. Následně by se tyto skupiny subjektů vrátily synchronizátoru pro další zpracování. Synchronizátor by následně tyto subjekty nemusel rozřazovat.

Toto řešení jsem zkušebně implementoval a otestoval. Zjistil jsem, že výkonnostní nárůst tohoto řešení je nevalný a navíc by se v systému Perun muselo zavést ukládání záložních souborů z minulých synchronizací. S novou logikou rozřazování subjektů, kterou představím v návrhové kapitole, by mělo být neupravené řešení bez záložních souborů rychlejší nebo minimálně stejně rychlé jako toto navrhované řešení, nicméně s tou výhodou, že se nemusí uchovávat záloha souborů a nemusí se měnit rozhraní pro získávání subjektů z externího zdroje kvůli zavedení tří skupin subjektů.

### 1.5.4 XML a ISXML

*XML* neboli *eXtensible Markup Language* je značkovací jazyk podobný HTML. Byl vytvořen za účelem skladování a přenosu dat. XML uchovává data v *elementech*. Ty se skládají z takzvaných *tagů*, které začínají znakem < a končí znakem >, a hodnot. Tyto elementy se do sebe můžou různě vnořovat a tvořit komplexnější objekty nazývané *uzly (nodes)*. [13]

Systém Perun používá externí zdroj XML u systémů, které jsou schopny vygenerovat výstup ve formátu XML. Soubor ve formátu XML je ale možno nahrát i lokálně a synchronizovat skupinu pomocí jej, stejně jako u CSV souborů. V definici externího zdroje je uvedena adresa, ze které systém Perun soubor získá, nebo cesta k lokální kopii souboru. Dále jsou v definici uvedeny potřebné údaje k získání přístupu k tomuto souboru. Soubor buď nemusí být

zabezpečen vůbec, nebo k němu má systém Perun přístup pomocí souboru s klíčem. Externí zdroj *ISXML* je nadstavba externího zdroje XML využívaný výhradně pro informační systém Masarykovy univerzity v Brně. Tam se pro přístup k danému souboru využívá přihlašovací jméno a heslo.

Po získání souboru probíhá jeho zpracování u obou externích zdrojů podobně, jen ISXML má v dotazu, kterým se budou vyhledávat záznamy, uvedeno navíc pracoviště a jméno skupiny. V souboru se vyhledávají správné uzly představující členy skupiny pomocí XPath výrazu definovaného v *Member's query* atributu skupiny. *XPath* slouží k procházení jednotlivých uzlů a elementů v XML souboru a vyhledání těch správných pomocí zmiňovaného výrazu. Ten může dokonce obsahovat i regulární výrazy pro složitější vyhledávání. [14]

Správné uzly s uživateli uložíme do seznamu. Tento seznam se následně zpracuje a jednotlivé uzly se překonvertují na mapy řetězců představující subjekty. Konverze probíhá pomocí mapování atributů z definice externího zdroje. Subjekty se následně vrátí synchronizační metodě pro další zpracování.

Podobně jako u CSV souborů jsem chtěl vylepšení importu dat z těchto externích zdrojů provést pomocí záložních souborů a porovnávání změn. Problém byl ale v tom, že XML soubory neobsahují vždy jen data o uživatelích a často obsahují plno pro nás zbytečných informací. Kvůli tomu musí být XPath výraz složitější a nalezení správných uzlů je časově náročnější. Do záložního souboru jsem sice nechtěl ukládat syrovou podobu přijatých dat, ale jen správné uzly získané pomocí XPath výrazu, nicméně i tak by zpracování přijatého souboru, načtení záložního souboru a jejich následné porovnání trvalo neúnosně dlouho. Stejně tak tyto XML soubory neobsahují žádnou informaci k detekci změny u jednotlivých záznamů. Proto se komunikace s těmito externími zdroji nebude upravovat.

### 1.5.5 Webové aplikace

Perun podporuje externí zdroje typu webových aplikací jako *VOOT*, *Unity*, *Perun*, *Google (Google Groups)* či informační systém Masarykovy univerzity v Brně *ISMU*. Obecně pro synchronizaci s těmito externími zdroji platí, že systém Perun naváže s webovou aplikací spojení a autorizuje se pomocí uživatelského jména a hesla. Následně se načte dotaz pro získání dat o uživatelích z atributu skupiny *Member's query* a pošle se pomocí *HTTP* dotazu *GET*[2] webové aplikaci. Ta nám poskytne data o uživatelích v určitém formátu.

Každá instance systému Perun má v sobě modul *RPC (Remote Procedure Call)*, který slouží ke vzdálenému volání metod. Díky tomu jsme schopni zavolat metodu pro získání všech členů určité skupiny z jiné instance systému

[2]GET metoda protokolu HTTP (HyperText Transfer Protocol) je používaná k získání dat uložených na webovém serveru dotazem na specifický zdroj (v našem případě na členy skupiny)[15]

Perun. RPC modul nám je schopný dodat data ve formátu, který požadujeme. V našem případě chceme formát JSON.

*JSON (JavaScript Object Notation)* je formát pro zapisování a posílání strukturovaných dat. Na rozdíl od jazyku XML je jednodušeji zpracovatelný a analyzovatelný strojem, nicméně je hůře čitelný pro člověka. Skládá se z *objektů*, které ve složených závorkách obsahují kolekci párů *název:hodnota*. Také se do sebe objekty mohou různě vnořovat.

Formát JSON je použit v případě externích zdrojů VOOT, Unity, Perun a Google. Formát podobný CSV je použit v případě ISMU (hodnoty jsou oddělené středníky namísto čárek). Takto získaná data se následně překonvertují pomocí mapování, které je definované v definici externího zdroje, do seznamu subjektů, který se následně vrátí synchronizační metodě pro další zpracování. Google Groups tato data sice posílá ve formátu JSON, ale metoda používaná v jazyku Java je umí překonvertovat do seznamu členů třídy *Member* (neplést s naším členem v systému Perun). Z nich pak získáme potřebná data a překonvertujeme je do subjektů.

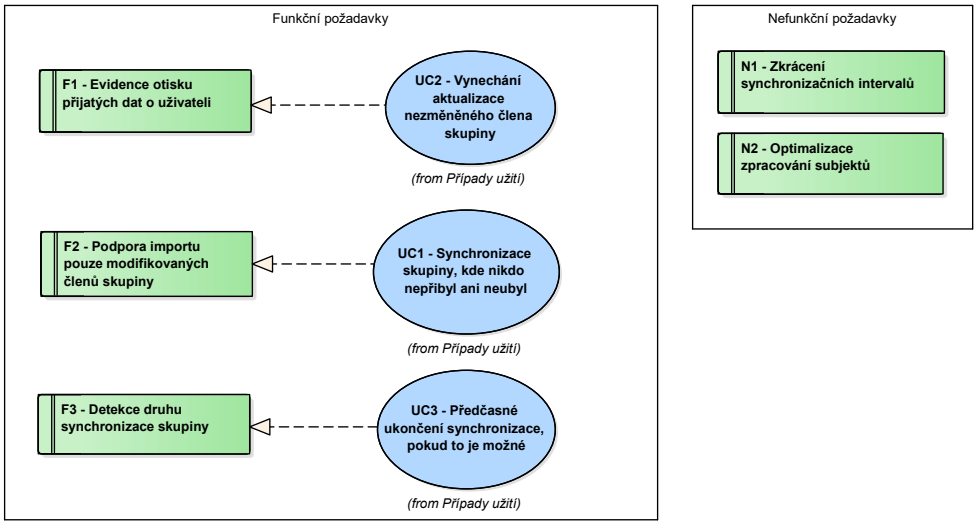
Žádná z těchto webových aplikací nepodporuje detekci změny záznamu. Jediná možnost, která přichází v úvahu, je použití *HTTP Caching*. Ten slouží k detekci, zda se odpověď serveru na dotaz systému Perun (požadavek na uživatele ve skupině) od předchozí komunikace změnila. Používá k tomu výpočet, který určí, jestli je odpověď „čerstvá“ a tím by se synchronizace mohla přeskočit, nebo jestli je „prošlá“ a musí se vygenerovat odpověď nová, kterou systém Perun musí následně zpracovat. Tento způsob se po domluvě s týmem pracujícím na systému Perun implementovat nebude, hlavně kvůli nejistým výsledkům detekce změny a problémům s implementací a výpočtem. [16]

## 1.6 Analýza požadavků

V analýze požadavků je popsáno, jaké funkční a nefunkční požadavky je potřeba splnit v rámci optimalizace importu dat do systému Perun. Mají vliv na podobu návrhu a výsledné implementace optimalizované synchronizace skupin systému Perun. Diagram funkčních a nefunkčních požadavků spolu s návazností funkčních požadavků na jednotlivé případy užití je zpracován v obrázku 1.8. Jednotlivé případy užití jsou popsány v části 1.6.3.

### 1.6.1 Funkční požadavky

Funkční požadavky představují funkcionality, které by měl optimalizovaný import členů skupin podporovat. Jedná se o požadavky, jejichž funkcionalita může být zkontrolována pomocí otestování jednotlivých případů užití, které jsou k nim navázány.




Obrázek 1.8: Diagram funkčních a nefunkčních požadavků s návazností na případy užití

**F1 - Evidence otisku přijatých dat o uživateli**
Hlavním požadavkem na optimalizaci importu dat je zrychlit proces zpracování dat na straně systému Perun. K tomu nám může pomoci výpočet *hashe (otisku)* ze všech dat uložených v subjektu. Ten by se při první synchronizaci člena skupiny uložil do jeho atributu (vztah člen-externí zdroj) a při dalším synchronizačním cyklu by se zkontrolovalo, zda je otisk stejný. Pokud by stejný byl, tak by se proces aktualizace tohoto člena vynechal.

**F2 - Podpora importu pouze modifikovaných členů skupiny**
Navrhnout systém importu uživatelů tak, aby se nemuseli pokaždé posílat všichni členové skupiny. Pokud externí zdroj podporuje detekci modifikovaných členů, tak by měl poslat v případe, kdy je množina uživatelů v externím zdroji stejná jako při předchozí synchronizaci, pouze modifikované členy skupiny.

**F3 - Detekce druhu synchronizace skupiny**
Navrhnout takový systém notifikací, aby externí zdroj poznal, zda synchronizuje skupinu s odlehčenou nebo plnou synchronizací. Dále navrhnout systém notifikací, který synchronizační metodu upozorní na to, jestli se z externího zdroje získali pouze modifikovaní členové skupiny, nebo jestli jsou v subjektech uloženi všichni členové skupiny.

### 1.6.2 Nefunkční požadavky

Díky nefunkčním požadavkům zjistíme, zda optimalizovaná podoba importu dat je opravdu efektivnější než řešení aktuálně používané. Jedná se zpravidla o požadavky směrované na výkon.

**N1 - Zkrácení synchronizačních intervalů**
Synchronizační intervaly jsou nyní u mnoha skupin nastaveny na větší hodnotu, než by bylo potřeba. Je to tak z toho důvodu, že mnoho instancí systému Perun eviduje velké množství uživatelů a skupin a velké skupiny, které mají tisíce uživatelů, brzdí synchronizaci skupin menších. Proto by optimalizace synchronizace skupin měla mít za následek snížení těchto intervalů.

**N2 - Optimalizace zpracování subjektů**
Jedná se o optimalizaci algoritmu pro rozřazování subjektů, kde nejdelší dobu trvá určení, zda daný subjekt již v systému Perun existuje jako uživatel a zároveň jestli je tento uživatel člen synchronizované skupiny. Navrhované řešení by mělo rozřazovat subjekty v závislosti na uživatelském jméně a externím zdroji, ze kterého je skupina synchronizovaná.

### 1.6.3 Případy užití (Use cases)

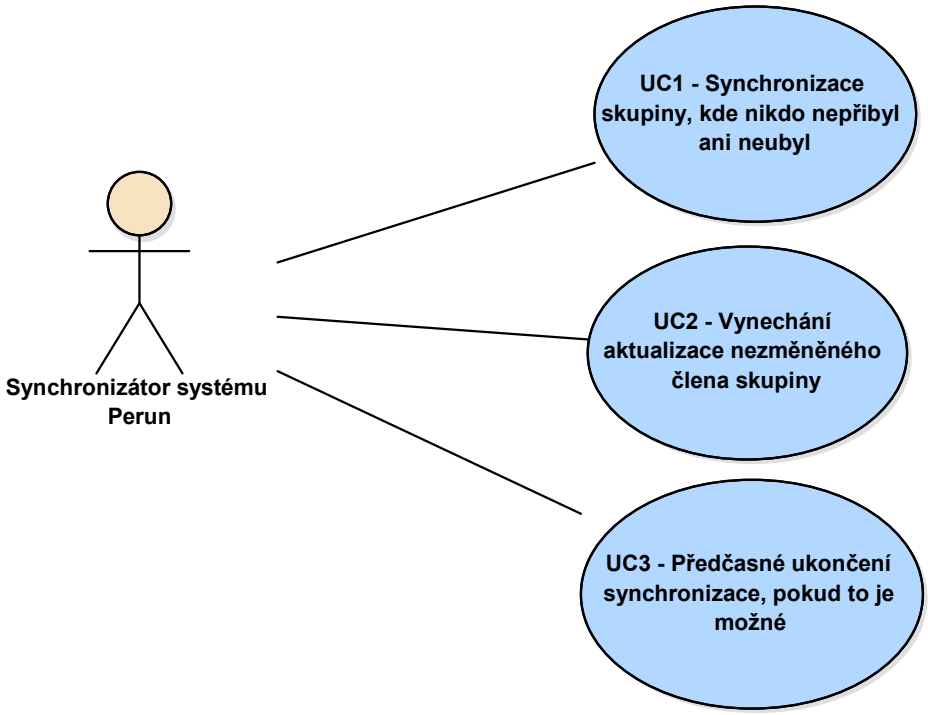
V této části jsou vypsány případy užití, jejichž funkčnost požadujeme od upraveného řešení synchronizací skupin v systému Perun. Díky případům užití navázaným na definované funkční požadavky (obrázek 1.8 se dá později určit, zda jsou funkční požadavky splněny, správně navrhnuty a implementovány.

**UC1 - Synchronizace skupiny, kde nikdo nepřibyl ani neubyl**
V tomto případu synchronizujeme skupinu z externího zdroje, jež musí umět detekovat, že se množina uživatelů ve skupině od minulé synchronizace nezměnila (externí zdroje LDAP, SQL, SQLComplex). V takovém případě pošleme externímu zdroji dotaz pro získání modifikovaných členů skupiny a ty zpracujeme. Tento případ užití se vztahuje k funkčnímu požadavku F2.

**UC2 - Vynechání aktualizace nezměněného člena skupiny**
Z externího zdroje získáme člena skupiny, kterého ve skupině již evidujeme. Tento člen již má v atributu uložený otisk získaných dat (subjektu) z předchozí synchronizace. Pokud tento otisk porovnáme s otiskem aktuálně přijatého subjektu a ten je shodný, tak se aktualizace tohoto člena přeskočí, protože se v něm nezměnil žádný atribut. Tento případ užití se vztahuje k funkčnímu požadavku F1.

**UC3 - Předčasné ukončení synchronizace, pokud to je možné**
Pokud se synchronizuje skupina, která má nastavenu odlehčenou synchronizaci (lightweight synchronization) a při komunikaci s externím

zdrojem se zjistí, že ve skupině nikdo nepřibyl ani neubyl, tak se komunikace předčasně ukončí. Tím se vynechá posílání dat o uživatelích z externího zdroje a vytváření subjektů. Dále se pošle upozornění synchronizační metodě, že může synchronizaci předčasně ukončit. Ta upozornění detekuje a synchronizaci skupiny ukončí. Tento případ užití se vztahuje k funkčnímu požadavku F3.




Obrázek 1.9: Diagram případů užití

#### 1.6.3.1 Podrobný popis případu užití UC1

1. Tento případ užití začíná zahájením synchronizace skupiny.

2. Po načtení všech potřebných atributů se zavolá metoda, která získá subjekty z externího zdroje. Tento externí zdroj musí podporovat detekci modifikovaných uživatelů. Jedná se o externí zdroj typu SQL, SQLComplex a LDAP.

3. Externímu zdroji se pošle dotaz, který zjistí, zda se množina uživatelů od předchozí synchronizace změnila.

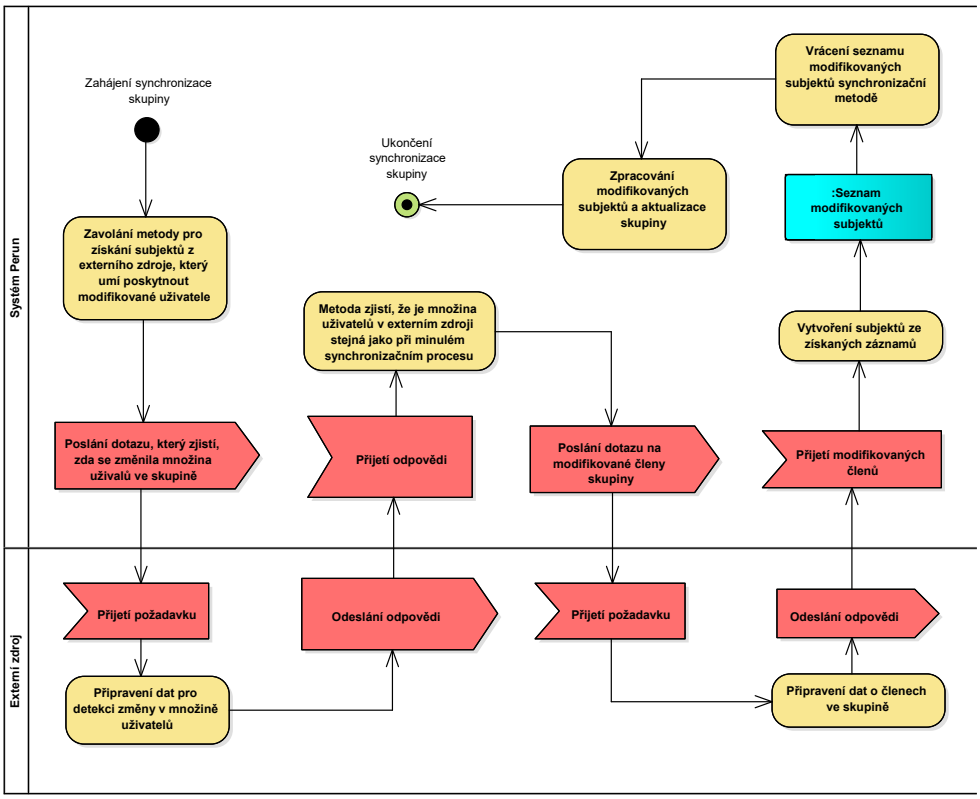
4. Externí zdroj požadavek přijme a vrátí nám data potřebná k rozhodnutí, jestli se množina uživatelů změnila. Může nám poskytnout například kontrolní součet z relační databáze nebo datum poslední změny záznamu skupiny v případě LDAP serveru.
5. Tato data systém Perun přijme a na základě porovnání hodnot s těma z předchozí synchronizace rozhodne, zda se množina uživatelů změnila.
6. V tomto případě se množina uživatelů nezměnila a můžeme tedy externímu zdroji poslat dotaz, který nám získá jen modifikované uživatele.
7. Externí zdroj nám je na základě dotazu specifického pro skupinu poskytne.
8. Systém Perun tyto modifikované členy skupiny přijme a vytvoří z nich subjekty.
9. Subjekty metoda komunikující s externím zdrojem vrátí metodě pro synchronizaci skupiny. Zároveň vrátí notifikaci, podle které synchronizační metoda bude vědět, že se jedná pouze o modifikované subjekty.
10. Synchronizační metoda tyto subjekty zpracuje a aktualizuje stávající členy skupiny. Zavolání metod pro přidávání a odstraňování členů vynechá. Metodu pro přidání člena by zavolala jen v případě, kdy by byl modifikovaný člen skupiny během synchronizace z nějakého důvodu odstraněn ze systému Perun.

#### 1.6.3.2 Podrobný popis případu užití UC2

1. Případ užití začíná úspěšným získáním subjektů z externího zdroje. Započne jejich rozřazení podle toho, zda se uživatelé reprezentovaní těmito subjekty mají do skupiny přidat, zda se mají jejich členské atributy aktualizovat nebo zda se mají ze skupiny odstranit.
2. Začneme subjekty v cyklu zpracovávat. Vybereme prvního.
3. U tohoto subjektu zjistíme, že představuje uživatele, který již v systému Perun existuje a zároveň je členem právě synchronizované skupiny.
4. V takovém případě porovnáme hodnotu otisku subjektu z minulé synchronizace s tím ze současné. Pokud by toto byla první synchronizace skupiny nově implementovaným způsobem, tak otisk z předchozí synchronizace neexistuje a uživatel tedy musí být aktualizován.
5. Zjistíme, že se tyto otisky shodují. To znamená, že u tohoto člena neproběhly na straně externího zdroje žádné změny.
6. Díky tomu můžeme aktualizaci tohoto člena vynechat.
7. Dále pokračujeme ve zpracovávání všech dalších subjektů.

#### 1.6.3.3 Podrobný popis případu užití UC3

1. Případ užití začíná zjištěním, že skupina má nastavenu odlehčenou synchronizaci (*lightweight synchronization*).
2. Dále se načtou potřebné atributy skupiny a zavolá se metoda pro získání subjektů z externího zdroje. Této metodě se předá notifikace, že se jedná o odlehčenou synchronizaci.
3. Metoda pro získání subjektů zjistí, že se množina uživatelů v externím zdroji od předchozí synchronizace nezměnila.
4. Díky tomuto zjištění a tomu, že má skupina nastavenu odlehčenou synchronizaci, nemusí dále od externího zdroje získávat množinu subjektů.
5. Vrátí tedy synchronizační metodě prázdný seznam subjektů a notifikaci toho, že nemusí provádět synchronizaci skupiny, protože v ní nikdo nepřibyl ani neubyl.
6. Synchronizační metoda přijme notifikaci, že nemusí dále v synchronizaci skupiny pokračovat, a proces ukončí.




Obrázek 1.10: Diagram aktivity případu užití UC1

Kapitola 2

# Návrhová část

V této kapitole je uveden návrh řešení požadavků, které je potřeba vyřešit pro optimalizaci synchronizace skupin. Sluší se zmínit, že před vytvořením návrhu jsem si musel vyzkoušet komunikaci s různými externími zdroji (především s LDAP), abych zjistil, co vše je a co není možné použít.

## 2.1 Návrh způsobu zavedení otisku přijatých dat u členů skupin

Tato část pokrývá funkční požadavek F1. *Hash (otisk)* dat přijatých o uživateli bude použit k zjištění, zda se od předchozí synchronizace přijatá data změnila a zda se má spustit metoda pro aktualizaci daného člena skupiny. Hash se vypočte ze všech dat, která jsou uložena v subjektu[3]. Porovnání hashe s tím uloženým z předchozí synchronizace bude probíhat jen v případě, kdy je subjekt představující uživatele již členem synchronizované skupiny. V případě rovnosti těchto hashů se aktualizace tohoto člena přeskočí. Jedná-li se o nového člena skupiny, pak mu bude atribut uchovávající hash zaveden při vytváření.

Jelikož se zavádí porovnávání hashů do již používaného systému, tak se při prvních synchronizacích existujících skupin bude stávat, že členové nebudou mít vedený atribut pro uchovávání hashe. V takovém případě se provede aktualizace člena skupiny spolu se zavedením tohoto atributu. Nový atribut uchovávající hash ze subjektu bude mít název *hashCode* a bude se vztahovat k dvojici entit *member-group (člen-skupina)*.

Metoda pro rozřazování subjektů bude muset poznat, zda jsou přijatá data o subjektu získána z externího zdroje s jednoduchým či složitějším rozhraním. V případě jednoduchého rozhraní bude subjekt obsahovat jen login a tudíž se z něj nemůže vypočítat hash. V takovém případě se zavolá metoda pro

[3]Pro zopakování, subjekt představuje všechna data přijatá o členovi skupiny ve formě mapy řetězců (*Map<String,String> subject*) skládající se z názvů atributů a jejich hodnot.

vytvoření kandidáta ze subjektu, která z externího zdroje získá všechna data o členovi skupiny a vytvoří z nich subjekt. Pomocí něj se následně vytvoří kandidát. Do této metody se přidá výpočet hashe ze subjektu, který se následně přidá jako další atribut do kandidáta. Po dokončení vytváření kandidáta se z něho získá hash, který se porovná s tím z minulé synchronizace. U externího zdroje se složitějším rozhraním je subjekt kompletní již před tvorbou kandidáta. Tudíž se může vytváření kandidáta přeskočit a výpočet hashe ze subjektu a následné porovnání proběhne rovnou.

## 2.2 Návrh nového způsobu kategorizace členů

Tato část řeší nefunkční požadavek N2. Subjekty získané z externího zdroje se nyní rozřazují v metodě *categorizeMembersForSynchronization* tak, že se z nich nejprve vytvoří kandidáti a pak se porovnávají *uživatelské externí zdroje* členů skupiny s těmi v kandidátech. Shoda znamená, že kandidát je již členem skupiny a přidá se do seznamu členů k aktualizaci. V opačném případě se kandidát přidá do seznamu kandidátů k přidání. Stávající členové, u kterých se nenašla shoda s žádným kandidátem, se přidají do seznamu členů k odstranění.

Navrhované řešení by mělo rozřazovat subjekty podle loginu, který uživatel má v externím zdroji, ze kterého je právě synchronizován. Nejprve se tedy musí vytvořit mapa (*Map<String,RichMember> loginsOfMembers*) stávajících členů skupiny, kde klíč bude login člena skupiny v právě synchronizovaném externím zdroji (získaném z *uživatelského externího zdroje (dále UES)* a hodnota bude člen skupiny s atributy.

Díky tomu nyní při procházení subjektů můžeme rychle zjistit, zda daný subjekt představuje některého člena skupiny dotazem na vytvořenou mapu pomocí loginu v konstantním čase. V případě shody víme, že subjekt je členem skupiny a můžeme porovnat hash subjektu tak, jak je popsán v části 2.1.

Pokud v mapě nenalezneme žádného člena pod daným loginem, tak vytvoříme ze subjektu kandidáta. Následně zkusíme pomocí všech UES kandidáta získat případného existujícího uživatele v systému Perun. Pokud ho získáme, tak takový uživatel již v systému Perun existuje. Následně znovu zkontrolujeme, jestli je takový uživatel již členem této skupiny. Může se totiž stát, že byl přidán do skupiny ručně a tudíž ještě nemá přiřazený UES z právě synchronizovaného externího zdroje. V případě, že je členem skupiny, tak bude přidán do seznamu členů k aktualizaci, jinak bude přidán do kandidátů k přidání.

Problém v systému Perun je ten, že pro jednoho člověka může v určitých případech, pokud byl synchronizován z více externích zdrojů, existovat více uživatelských účtů. V takovém případě by se tyto uživatelské účty měly sloučit do jednoho. Tuto práci vykonává ručně administrátor systému Perun. Může se totiž například stát, že existuje člověk s dvěma uživatelskými účty v systému Perun. První uživatelský účet je členem synchronizované skupiny, ale druhý uživatelský účet má k sobě přiřazený UES ze synchronizovaného externího

zdroje. V takovém případě si bude synchronizátor myslet, že tento uživatel v systému Perun existuje, ale není členem skupiny. Tudíž ho přidá do kandidátů k přidání. Při vytváření člena skupiny se k němu ale nepřiřadí UES, protože již v systému Perun existuje. A tudíž i při dalším synchronizačním cyklu by si Perun myslel, že tento uživatel není členem skupiny.

Pokud tedy tento problém nastane a při synchronizaci se zjistí, že v systému Perun uživatel existuje vícekrát, tak se tato událost musí zaznamenat pomocí vyhození výjimky, aby to administrátor zaznamenal a uživatelské účty ručně sloučil. Výjimka bude obsahovat oba uživatele, aby administrátor věděl, které uživatelské účty je nutno sloučit.

## 2.3 Návrh systému notifikací pro synchronizaci skupin

Tato část pokrývá funkční požadavek F3. V současné podobě není třída starající se o komunikaci s externím zdrojem schopna vědět, zda synchronizuje skupinu s plnou nebo odlehčenou synchronizací. Stejně tak metoda *synchronizeGroup* v manažerovi skupin na business vrstvě (*GroupsManagerBl*) neví, zda získala z externího zdroje všechny členy skupiny, nebo jen ty modifikované. Doposud to nebylo potřeba. Nový návrh komunikace s určitými externími zdroji, které podporují získávání modifikovaných uživatelů, zapříčiňuje potřebu předávat si status.

Proto jsem se rozhodl změnit metodu *getGroupSubjects* tak, aby nyní přijímala jako parametr status, který bude obsahovat řetězec s informací o tom, zda se jedná o odlehčenou či plnou synchronizaci. Zároveň návratová hodnota bude místo seznamu subjektů (*List<Map<String, String» subjects*) řetězec informující o tom, zda se z externího zdroje získali jen modifikovaní členové skupiny, či všichni. Seznam subjektů se namísto návratové hodnoty bude předávat jako parametr. V metodě *synchronizeGroup* se tento seznam inicializuje, předá se prázdný jako parametr metody *getGroupSubjects* a v ní se následně vyplní subjekty.

Řetězce představující statusy budou definovány v manažerovi skupin jako konstanty. Metoda *synchronizeGroup* bude předávat status buď typu *full* (plná synchronizace) či *lightweight* (odlehčená synchronizace). Metoda *getGroupSubjects* bude jako návratovou hodnotu vracet buď *full* (seznam subjektů obsahuje všechny členy skupiny) či *modified* (seznam členů obsahuje pouze modifikované členy skupiny).

## 2.4 Optimalizace externích zdrojů SQL a SQLComplex

Tato část pokrývá funkční požadavek F2. V této části si popíšeme navržené změny v importu dat z externích zdrojů *SQL* a *SQLComplex*. Tyto externí zdroje jsou schopny vypočítat, zda někdo v množině uživatelů přibyl nebo ubyl podle primárních klíčů záznamů. Říká se tomu *výpočet kontrolního součtu*. Zároveň jsou schopny zaslat nám jen ta data, která se od poslední synchronizace změnila. Každý záznam totiž může obsahovat čas poslední změny. Tento čas porovnáme s *časem začátku poslední synchronizace* (ne koncem poslední synchronizace kvůli tomu, pokud by byly během předchozí synchronizace záznamy na externím zdroji pozměněny). Čas začátku poslední synchronizace bude vytvořen jako nový atribut skupiny.

### 2.4.1 Popis optimalizované podoby importu dat

1. Synchronizační metoda zavolá metodu pro získání subjektů z externího zdroje.

2. Načtou se potřebné atributy externího zdroje pro navázání spojení a spojení se naváže. Dále se načtou atributy skupiny obsahující potřebné dotazy pro získání dat.

3. Systém Perun pošle externímu zdroji požadavek na výpočet kontrolního součtu skupiny (dotaz uložený v atributu skupiny *groupChangeDetectionQuery*.

4. Externí zdroj požadavek přijme a zkontroluje, zda je stroj se systémem Perun autorizován. Pokud ano, tak pomocí přijatého dotazu vypočte kontrolní součet dané skupiny. Ten odešle systému Perun zpět.

5. Systém Perun přijme kontrolní součet a porovná ho s tím, který má uložený v atributu skupiny *groupChangeDetectionValue* z předchozí synchronizace.

6. Pokud je kontrolní součet shodný, tak je množina uživatelů stejná. Nastavíme tedy status externího zdroje na *modified*, který znamená, že se z externího zdroje budou získávat pouze modifikovaní uživatelé. Pokud má skupina nastavenu odlehčenou synchronizaci, tak se synchronizační metodě vrátí prázdný seznam subjektů a zmiňovaný status externího zdroje. Pokud skupina nemá nastavenu odlehčenou synchronizaci, tak se externímu zdroji pošle dotaz pro získání modifikovaných členů skupiny (atribut skupiny *groupModifiedMembersQuery*) od startu poslední úspěšné synchronizace (atribut skupiny *startOfLastSuccessSynchronizationTimestamp*).

7. Pokud kontrolní součet shodný není, tak se množina uživatelů v externím zdroji změnila. Nastavíme tedy status externího zdroje na *full*, který detekuje, že seznam subjektů bude obsahovat všechny členy skupiny. Následně se pošle externímu zdroji dotaz pro získání všech uživatelů ve skupině. Dál synchronizace probíhá stejně.

8. Relační databázový server přijme požadavek a zkontroluje, zda je k němu systém Perun autorizovaný. Pokud ano, tak odešle požadovanou množinu dat o uživatelích systému Perun.

9. Systém Perun tato data přijme a metoda pro získání členů skupiny z externího zdroje vyplní parametrem předaný seznam subjektů. Tento seznam subjektů se spolu se statusem externího zdroje vrátí synchronizační metodě pro další zpracování.

## 2.5 Optimalizace externího zdroje LDAP

Tato část pokrývá funkční požadavek F2. V této části si popíšeme navržené změny v importu dat z externího zdroje *LDAP*. LDAP podporuje u záznamů atribut obsahující čas poslední změny. Tento čas využijeme dvěma způsoby. U záznamu reprezentujícího skupinu uživatelů tím ověříme, zda se množina uživatelů ve skupině od minulé synchronizace změnila. U záznamů představujících uživatele se tento atribut využije k získání jen těch záznamů, které byly modifikované od startu poslední úspěšné synchronizace skupiny.

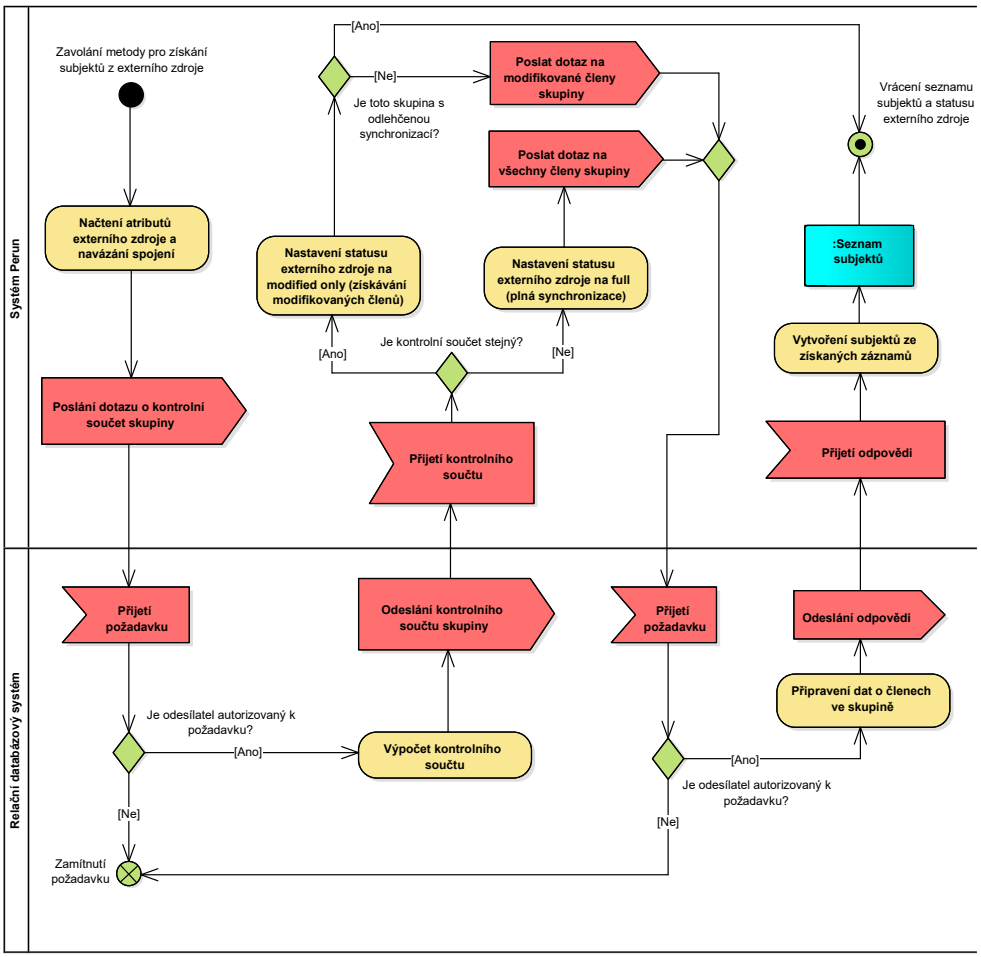
### 2.5.1 Popis optimalizované podoby importu dat

1. Synchronizační metoda *synchronizeGroup* v *GroupsManagerBl* zavolá metodu pro získání subjektů z externího zdroje *getGroupSubjects*.

2. Načtou se potřebné atributy externího zdroje pro navázání spojení a spojení se naváže. Dále se načtou atributy skupiny obsahující potřebné dotazy pro získání dat.

3. Zkontroluje se, zda definice externího zdroje obsahuje atribut *timestampAttribute* obsahující informaci o tom, v jakém atributu na LDAP serveru jsou ukládány časy posledních úprav záznamů. Pokud je tento atribut definován, tak systém Perun pošle externímu zdroji dotaz pro získání času poslední úpravy záznamu se skupinou a také požadavek pro získání názvů záznamů všech uživatelů ve skupině.

4. Externí zdroj požadavek přijme a zkontroluje, zda je systém Perun autorizován pro takové požadavky. Pokud ne, požadavek zamítne. Pokud ano, odešle zpět odpověď s časem poslední změny skupiny a s názvy záznamů uživatelů ve skupině.




Obrázek 2.1: Diagram aktivity znázorňující upravenou komunikaci s externím zdrojem SQL a SQLComplex

5. Systém Perun přijme data a zkontroluje, zda se čas poslední úpravy skupiny shoduje s časem, který si eviduje z předchozích synchronizací jako atribut skupiny *groupChangeDetectionValue*. Pokud ano, znamená to, že se množina uživatelů ve skupině na externím zdroji nezměnila.

6. Nastaví se tedy status externího zdroje na *modified*, který slouží k zaznamenání, že se budou z externího zdroje přijímat pouze modifikovaní členové skupiny.

7. Dále se zkontroluje, zda má skupina nastavenu odlehčenou synchronizaci. Pokud ano, tak se synchronizační metodě vrátí prázdný seznam subjektů spolu se zmiňovaným statusem externího zdroje, aby mohla synchronizační metoda synchronizaci předčasně ukončit.

8. Pokud skupina nemá nastavenu odlehčenou synchronizaci, tak se do filtru pro získávání atributů jednotlivých členů skupiny přidá časový filtr, který povolí získání atributů jen těch členů, kteří byli od začátku poslední úspěšné synchronizace (atribut *startOfLastSuccessSynchronizationTimestamp*) změněni. Formát data je uveden v definici externího zdroje v atributu *modifyTimestampFormat*.

9. Pokud nebyl čas poslední úpravy záznamu skupiny shodný s tím z předchozí synchronizace, tak se nastaví status externího zdroje na *full* k zaznamenání, že seznam subjektů bude obsahovat všechny členy skupiny. Dále se pokračuje stejně.

10. Systém Perun začne v cyklu procházet všechny názvy záznamů členů skupiny, které získal ze záznamu se skupinou. Tyto názvy záznamů jen definují umístění tohoto uživatele (záznamu) na LDAP serveru.

11. Systém Perun pošle externímu zdroji požadavek na získání atributů jednoho uživatele.

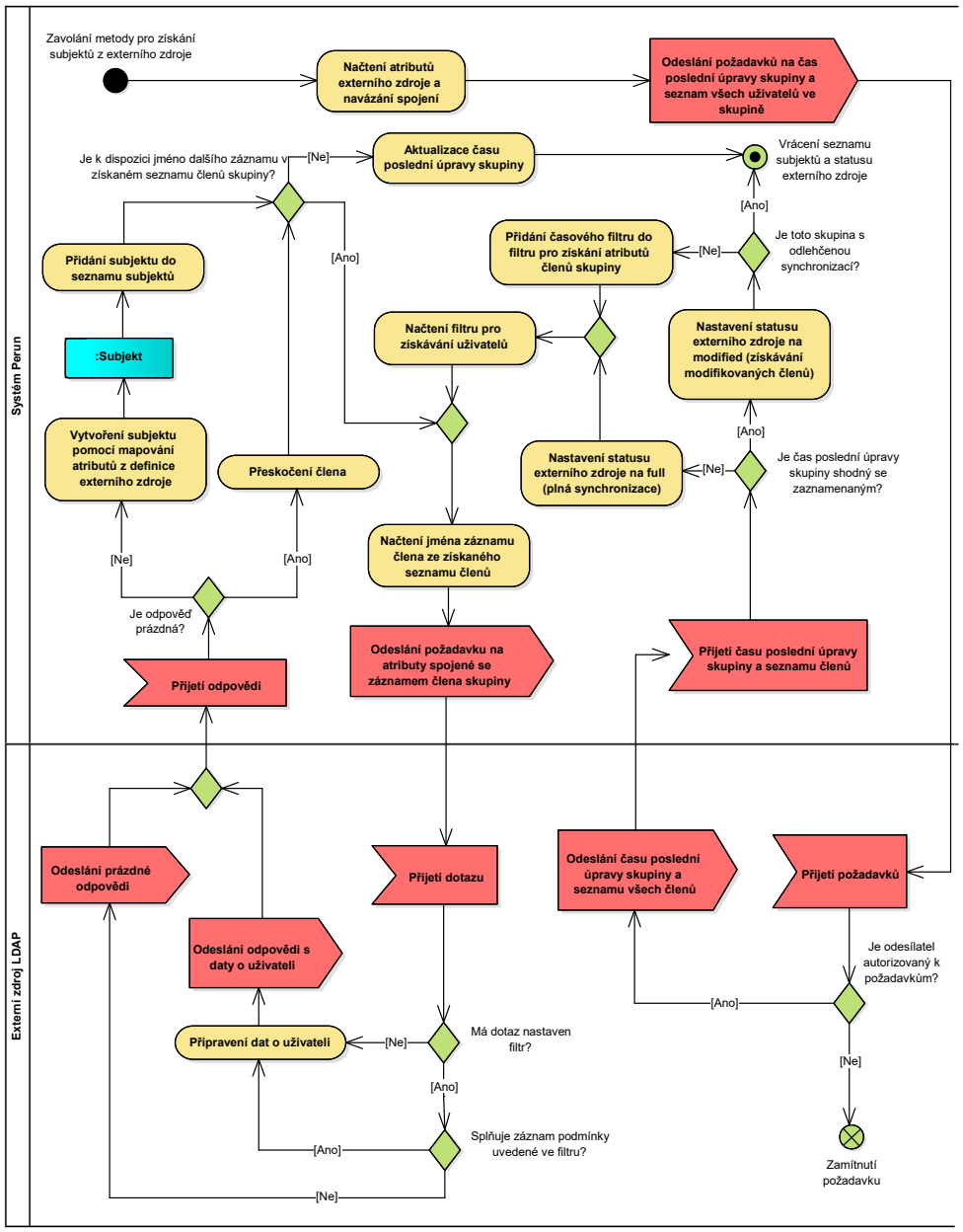
12. LDAP server přijme požadavek. Zkontroluje, zda dotaz obsahuje časový filtr. Pokud ano, tak zkontroluje, zda byl záznam modifikován od času uvedeného ve filtru.

13. Pokud záznam upravován nebyl, tak pošle prázdnou odpověď. Pokud záznam upravován byl, tak připraví atributy uživatele a odešle je systému Perun.

14. Systém Perun přijme odpověď od LDAP serveru. Zkontroluje, zda je odpověď prázdná. Pokud je odpověď prázdná, tak vytváření subjektu z tohoto člena přeskočí. Pokud prázdná není, tak pomocí mapování atributů uvedeného v definici externího zdroje vytvoří subjekt. Ten přidá do seznamu subjektů.

15. Dále se zkontroluje, jestli seznam názvů uživatelů ve skupině obsahuje další záznam. Pokud ano, tak se stejný postup opakuje pro dalšího člena skupiny. Pokud ne, tak se aktualizuje atribut s časem poslední úpravy skupiny.

16. Nyní se už jen synchronizační metodě vrátí seznam subjektů a status externího zdroje. Subjekty se dále zpracují a aktualizuje se skupina.




Obrázek 2.2: Diagram aktivity znázorňující upravenou komunikaci s externím zdrojem LDAP

Kapitola 3

# Implementační část

V této kapitole je popsán způsob, jakým bylo navrhované řešení implementováno včetně ukázek zajímavých částí kódu. Dále je popsáno prostředí, ve kterém byla implementace prováděna, a také to, co vše bylo zapotřebí do systému Perun zavést, aby nové řešení synchronizace skupin fungovalo. Celá práce byla naprogramována v jazyku Java.

## 3.1 Příprava instance systému Perun

Vývoj optimalizovaného řešení synchronizace skupin probíhal na mojí vlastní instanci systému Perun, která byla zkopírována z vývojářské instance. Běží na stejném serveru jako vývojářská instance z důvodu, aby měla přístup k reálným externím zdrojům a mohlo být otestováno, zda bylo řešení naprogramováno správně. Také bylo nutné vytvořit si kopii databáze systému Perun a připojit ji k mé instanci, aby nebyly zaváděny případné chyby v datech do práce ostatních.

Po zprovoznění vlastní instance bylo nutné do systému Perun zavést atributy, které bude nově synchronizace skupin používat. Zatímco *core* a *defined* atributy se uvádějí přímo v kódu v rámci manažera atributů (*Attributes manager*) a do systému se zavádějí při sestavení, tak ostatní atributy (*virtuální (virt)* a *volitelné (opt)*) se musí do každé instance systému Perun zadat ručně. Prozatím není v systému Perun žádná funkce, která by na instanci nahrála automaticky úplně všechny atributy. Dělá se to pomocí vyhrazené čisté instance systému Perun, která obsahuje všechny potřebné atributy a při novém nasazení systému se kopíruje.

Přímo do kódu byly zavedeny defined atributy *startOfLastSuccessSynchronizationTimestamp* a *hashCode*. První z nich slouží k uložení času poslední úspěšné synchronizace skupiny a váže se ke skupině. Druhý slouží k uložení kontrolního součtu vypočítaného ze subjektu. Váže se ke vztahu člen-skupina. Ručně byly do mé testovací instance vloženy volitelné atributy skupin *groupChangeDetectionQuery* k uložení dotazu pro detekování, zda se množina uži-

vatelů na externím zdroji změnila, *groupChangeDetectionValue* pro uložení výsledku z předchozího zmíněného dotazu a *groupModifiedMembersQuery* pro dotaz k získání modifikovaných členů skupiny.

## 3.2 Implementace kategorizace členů

Přijaté subjekty z externího zdroje se kategorizují do tří skupin: kandidátů k přidání, členů k aktualizaci a členů k odstranění. Tento postup se provádí ve dvou metodách: *categorizeMembersForLightweightSynchronization* pro odlehčenou synchronizaci a *categorizeMembersForSynchronization* pro plnou a změnovou synchronizaci. Rozdíl je v tom, že v odlehčené synchronizaci se kandidát vytváří bez všech poskytnutých atributů, protože u něj záleží jen na členství ve skupině. Zároveň se v odlehčené synchronizaci přeskakují již existující uživatelé a neaktualizují se.

Rozřazování subjektů má podle návrhu hledat existující členy skupiny pomocí uživatelského jména získaného ze synchronizovaného externího zdroje. Uživatelské jméno a příslušný externí zdroj jsou k uživateli v systému Perun vázány pomocí uživatelského externího zdroje. Největší problém při implementaci tohoto řešení byl s tím, kdy uživatel v systému Perun existuje vícekrát. Jak bylo popsáno v návrhu řešení v části 2.2, uživatelský externí zdroj může v určitých případech být vázaný na jiného uživatele, který není členem skupiny. V takovém případě pak metoda nenajde existujícího člena skupiny s tímto uživatelským externím zdrojem a vytvoří vždy z tohoto subjektu kandidáta k přidání. Následně při přidávání kandidáta pomocí přídavných uživatelských externích zdrojů zjistí, že tento kandidát již je členem skupiny a pokusí se ho aktualizovat. Ovšem přeskočí přidání hlavního uživatelského externího zdroje z toho důvodu, že už v systému Perun existuje (vázán k jinému uživateli, ale reálně k stejnému člověku). Tento proces se následně děje stále dokola při každé synchronizaci a zpomaluje tak celý průběh.

Tento problém byl vyřešen metodou *checkDuplicationOfUser* v ukázce kódu 3.1, která pomocí všech uživatelských externích zdrojů uložených v kandidátovi dokáže detekovat, zda v systému Perun existuje více uživatelů reálně představujících jednoho člověka. Metoda v cyklu prochází všechny uživatelské externí zdroje kandidáta a zkouší podle nich hledat uživatele v systému Perun. Tohoto uživatele si uloží a porovná ho s uživatelem získaným z předchozího uživatelského externího zdroje. Pokud se uživatelé rovnají, tak je vše v pořádku. Pokud jsou to dva různí uživatelé, tak metoda vyhodí výjimku *UserDuplicateException*. Ta se pro administrátora systému Perun zapíše do logu. Ten by měl následně tyto uživatele ručně sloučit. Slučování takových uživatelů je poměrně problematická věc, protože se musí řešit i se správci externích zdrojů a služeb. Každá taková kopie uživatele může být propojená se službami například pomocí jiného ssh klíče, což by po sloučení mohlo vést k problémům s přístupem ke službám. Proto bohužel není možné uživatele

slučovat automaticky.

Zdrojový kód 3.1: Metoda pro detekci duplikace uživatelů

```
private void checkDuplicationOfUser(PerunSession sess, Candidate candidate)
    throws InternalErrorException, UserDuplicateException {
  User user = null;
  // Check if user does not exist in Perun multiple times
  if (candidate.getUserExtSources() != null) {
    for (UserExtSource ues: candidate.getUserExtSources()) {
      try {
        // Check if the extSource exists
        getPerunBl().getExtSourcesManagerBl().checkExtSourceExists(sess,
            ues.getExtSource());
        // Try to find the user by userExtSource
        User tmp = getPerunBl().getUsersManagerBl().getUserByUserExtSource(sess,
            ues);
        if (user != null && !tmp.equals(user)) {
          throw new UserDuplicateException(user, tmp);
        }
        user = tmp;
      } catch (UserNotExistsException | ExtSourceNotExistsException IGNORE) {
      // Ignore, we are only checking if the user exists
      }
    }
  }
}
```

V ukázce kódu 3.2 je uveden algoritmus pro zjištění, zda se člen skupiny změnil od předchozí synchronizace. Pokud se zjistí, že subjekt je členem skupiny, tak se zkontroluje, zda se atributy člena nemají získat z jiného externího zdroje a zda se synchronizuje externí zdroj s jednoduchým rozhraním. Pokud ne, tak v sobě subjekt uchovává všechny potřebné informace o členovi a může se z něj vypočítat hash, který se porovná s tím uloženým v systému Perun u člena skupiny.

Pokud v sobě subjekt neobsahuje všechny potřebné atributy, tak se zavolá metoda pro vytvoření kandidáta *convertSubjectToCandidate*. Ta se postará o získání všech atributů z externího zdroje a získá z něj subjekt. Z něj následně vypočítá hash a uloží ho jako atribut do kandidáta. Ten se následně porovná a zjistí se, zda se má člen skupiny aktualizovat nebo ne.

Posledním krokem je odstranění tohoto člena z mapy loginů a členů skupiny. Po zpracování všech subjektů se tito zbývající členové v mapě přidají do seznamu těch, kteří budou odstraněni ze skupiny. Největším problémem při implementaci porovnávání hash kódů bylo určení, kdy je možné hash vypočítat předem, aby bylo možné vyvarovat se vytváření kandidáta a urychlit tak celý proces.

Zdrojový kód 3.2: Algoritmus detekce změny člena skupiny

```
if (loginSource.equals(memberSource) && memberSource instanceof ExtSourceApi) {
  // Count hash code of subject
  String hashCode = Integer.toString(subject.hashCode());
  // Compare hash code of subject and member
  if (!hashCode.equals(hashCodeMember.getValue())) {
    // If they are different, create candidate and put him to membersToUpdate
    candidate = convertSubjectToCandidate(sess, subject, memberSource,
        loginSource, skippedMembers);
    membersToUpdate.put(candidate, richMember);
  }
} else {
  // If we don't have subject with attributes, get him and convert him to
      candidate
```

```
  candidate = convertSubjectToCandidate(sess, subject, memberSource, loginSource,
      skippedMembers);
  // Get hashCode of subject with attributes from candidate
  String hashCodeSubject =
      candidate.getAttributes().get(MembersManager.MEMBERGROUPHASHCODE_ATTRNAME);
  // Compare hash code of subject and member
  if (hashCodeSubject != null &&
      !hashCodeSubject.equals(hashCodeMember.getValue())) {
    // If they are different, put candidate to membersToUpdate
      membersToUpdate.put(candidate, richMember);
  }
}
// Remove this login from map
loginsOfMembers.remove(login);
```

## 3.3 Implementace optimalizace externího zdroje LDAP

Pro implementaci a testování komunikace s externím zdrojem LDAP byl použit LDAP server sdružení CESNET. Při implementaci jsem se setkal s problémem, kdy mi LDAP nevracel tzv. *operational attribute (operační atribut)* s časem poslední změny skupiny (ve volně šířitelné implementaci LDAP protokolu *OpenLDAP* to je *modifyTimestamp*). Bylo to tím, že k získání operačních atributů musíte být k LDAP serveru přihlášeni účtem a navíc tento účet musí mít povoleno tyto atributy číst. To jsem nevěděl, protože jiné atributy mi byly poskytnuty i bez povolení. LDAP server musí mít vedení tohoto atributu povoleno, aby bylo možné optimalizované řešení komunikace použít.

Pro použití vylepšené verze komunikace s externím zdrojem LDAP se musí v definici externího zdroje uvést nový atribut *timestampAttribute*, ve kterém je uveden výše zmíněný atribut LDAP protokolu uchovávající v sobě čas poslední změny záznamů. Může to být např. atribut *modifyTimestamp* či *whenChanged* v závisloti na použité implementaci LDAP protokolu. Dále se musí přidat atribut *timestampFormat* pro určení formátu data a času, podle kterého se budou získávat modifikovaní uživatelé (např. yyyyMMddHHmmss'Z'). Pokud bude definován timestampAttribute, ale ne timestampFormat, tak synchronizace skupin s tímto externím zdrojem skončí výjimkou. Pokud nebude definován ani timestampAttribute, tak se provede získávání subjektů starým způsobem.

V ukázce zdrojového kódu 3.3 je vyřešeno nastavení časového filtru pro získávání atributů členů skupiny. Pro spuštění tohoto kódu je ale potřeba, aby měl externí zdroj definovány výše zmíněné atributy a čas změny záznamu se skupinou na LDAP serveru se rovnal tomu uloženému v systému Perun v atributu *groupChangeDetectionValue*.

Dále se z atributů skupiny získá řetězec s časem začátku poslední úspěšné synchronizace skupiny. Pokud je získán, tak se z řetězce vytvoří objekt uchovávající datum (*Date*) podle formátu, ve kterém systém Perun uchovává datum a čas. Následně načteme z definice externího zdroje formát, ve kterém LDAP server uchovává datum a čas. Do tohoto formátu se převede čas začátku poslední úspěšné synchronizace skupiny a vytvoří se řetězec s filtrem. Následně se

časový filtr přidá k případným ostatním filtrům, které má skupina či externí zdroj definovány. S tím byl největší problém, protože pokud filtr již nějaké podmínky obsahuje, tak je potřeba časový filtr uzavřít do závorek a spojit ho s ostatními filtry pomocí logického *and*. Posledním krokem je nastavení statusu, který se bude vracet metodě *synchronizeGroup*, na hodnotu detekující, že seznam subjektů bude obsahovat pouze modifikované členy skupiny.

Zdrojový kód 3.3: Přidání filtru s časem poslední úpravy členů skupiny

```
String startOfLastSuccessSync =
    groupAttributesMap.get(GroupsManager.GROUPSTARTOFLASTSUCCESSSYNC_ATTRNAME);
if (startOfLastSuccessSync != null) {
  Date startDate = BeansUtils.getDateFormatter().parse(startOfLastSuccessSync);
  // Create string with filter to detect modified records
  SimpleDateFormat sdf = new SimpleDateFormat(modifyTimestampFormat);
  String dateForFilter = sdf.format(startDate);
  String additionToFilter = "(" + modifyTimestampName + ">=" + dateForFilter +
      ")";
  if (filter == null) {
    filter = additionToFilter;
  } else {
    String tmp = "(&" + additionToFilter + filter + ")";
    filter = tmp;
  }
  typeOfSynchronization = GroupsManager.GROUP_SYNC_STATUS_MODIFIED;
}
```

## 3.4 Implementace optimalizace externích zdrojů SQL a SQLComplex

Optimalizace metody pro získávání členů z externích zdrojů SQL a SQLComplex probíhala naprosto totožně a kód je tudíž identický. V obou případech se získávají data pomocí příkazu *SELECT*, jen v případě SQL se získají pouze loginy členů skupiny a v případě SQLComplex se získají navíc i všechny atributy členů.

Největší problém byl s výpočtem kontrolního součtu členů skupiny pro zjištění, zda se jejich množina od předchozí synchronizace změnila. Vypočtená hodnota musí být řetězec nebo číslo, které lze případně převést na řetězec a uložit do atributu skupiny v systému Perun. Tento výpočet neprobíhá na straně systému Perun, ale na straně externího zdroje. Pro otestování byl kontrolní součet vypočítán v relační databázi používající *PostgreSQL*.

V případě PostgreSQL byla použita metoda *md5* pro výpočet hashe z identifikátoru člena. V testovaném prostředí představoval identifikátor člena číslo. Musel být tedy metodou *to_char* převeden na řetězec. Následně byla použita metoda md5 pro vytvoření hashe. Tím byl vytvořen pro každý identifikátor člena hash. Tyto řetězce bylo nutné spojit do jednoho zavoláním metody *string_agg*. Tím byl získán jeden dlouhý řetězec, který ale může být velice dlouhý. Proto je na něj znovu zavolána metoda md5, aby byl zkrácen. Výsledný hash je poslán systému Perun. Celý příkaz SELECT je uveden níže.

```
SELECT MD5(STRING_AGG(MD5(TO_CHAR(users.id,
    '999999999')), '-')) AS "crc" FROM users;
```

K výpočtu kontrolního součtu lze použít mnoho způsobů. Tento způsob slouží jen jako příklad. Jedinou podmínkou pro výpočet kontrolního součtu je, aby systém Perun získal jeden záznam s číslem či řetězcem a aby se tento sloupec nazýval *crc*. Kontrolní součet musí být pro stejnou množinu uživatelů vždy stejný.

Pro použití optimalizované verze komunikace musí mít skupina vyplněný atribut *groupChangeDetectionQuery* pro detekci změny množiny uživatelů v externím zdroji. Dále je nutný atribut skupiny *groupModifiedMembersQuery* pro dotaz k získání modifikovaných uživatelů a atribut *timestampFormat* uvedený v definici externího zdroje s formátem, v jakém ukládá externí zdroj datum a čas. Pokud bude mít skupina vyplněný první zmiňovaný atribut a další dva zmíněné budou chybět, tak synchronizace skupiny skončí vyhozením výjimky.

Případ, kdy systém Perun chce získat z externího zdroje pouze modifikované uživatele, je velice podobný tomu popsanému u externího zdroje LDAP. Liší se v tom, že místo přidání časového filtru do dotazu na uživatele se v takovém případě nepoužije dotaz pro získání všech uživatelů, ale jen těch modifikovaných uvedený v atributu *groupModifiedMembersQuery*. Tento příkaz *SELECT* musí mít uvedenou podmínku pro získání uživatelů, jejichž záznam se změnil od určité doby $x$. Místo $x$ bude ale použit otazník, který se v dotazu v průběhu synchronizace nahradí časem začátku poslední úspěšné synchronizace skupiny. V následující ukázce se spojují dvě tabulky a je tedy nutné porovnávat čas změny spojených záznamů z obou tabulek.

```
SELECT users.id AS "login" FROM users LEFT JOIN
   user_attr_values uav ON uav.user_id=users.id
   WHERE uav.attr_id=1361 AND
   GREATEST(uav.modified_at, users.modified_at)>'?';
```

Kapitola 4

# Vyhodnocení

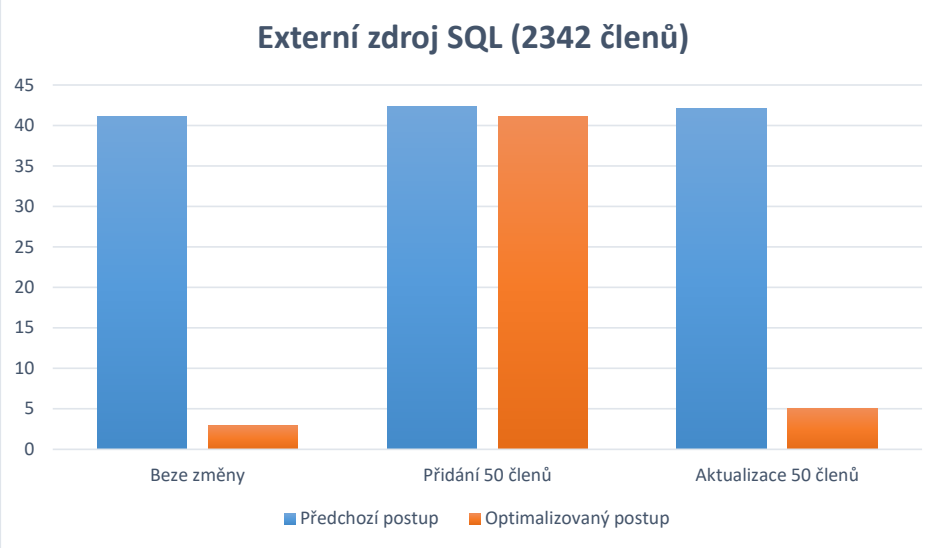
V této kapitole je popsáno testování rychlosti optimalizovaného řešení. V části 4.1 je popsáno, jakým způsobem probíhalo testování. V grafech jsou znázorněny rychlosti synchronizace skupin pro různé externí zdroje a různé případy. V části 4.2 je vyhodnocen přínos nově naimplementovaného řešení pro výkon systému Perun.

## 4.1 Testování

Testování probíhalo na externích zdrojích typu *SQL*, *SQLComplex*, *LDAP*, *EGISSO* a *XML*. Tyto externí zdroje byly vybrány z důvodu otestování vlivu optimalizované komunikace u externích zdrojů SQL, SQLComplex a LDAP a zároveň otestování vlivu nového rozřazování subjektů a porovnávání otisků subjektů na neupravené externí zdroje, jež zastupují EGISSO a XML. Modrý sloupec v grafech představuje neupravenou verzi synchronizace skupin. Oranžový sloupec představuje nový optimalizovaný způsob. Délka trvání synchronizace skupiny je měřena v sekundách.

Pro testování externích zdrojů SQL a SQLComplex byla zvolena komunikace s databází testovací instance systému Perun, která běží na stejném serveru jako má vývojářská instance. V tomto případě je tedy odstraněn vliv rychlosti přenosu dat po internetu.

Optimalizované řešení komunikace s externím zdrojem SQL je výrazně rychlejší v případech, kdy se od předchozí synchronizace nezměnila množina uživatelů (obrázek 4.1). Nejpomalejší částí synchronizačního procesu je v tomto případě navazování komunikace s relační databází pro každého člena zvlášť. Čím více členů skupiny musí být z externího zdroje získáno, tím menší je rozdíl mezi předchozím a optimalizovaným řešením. Významně tomu nepomáhá ani vylepšené rozřazování subjektů a kontrola hashů členů skupiny. V tomto případě totiž není možné vypočítat ze subjektu hash ihned, protože neobsahuje všechny potřebné atributy o členovi skupiny. Ty se získají až po vytvoření kandidáta, jehož tvorbu nelze přeskočit.

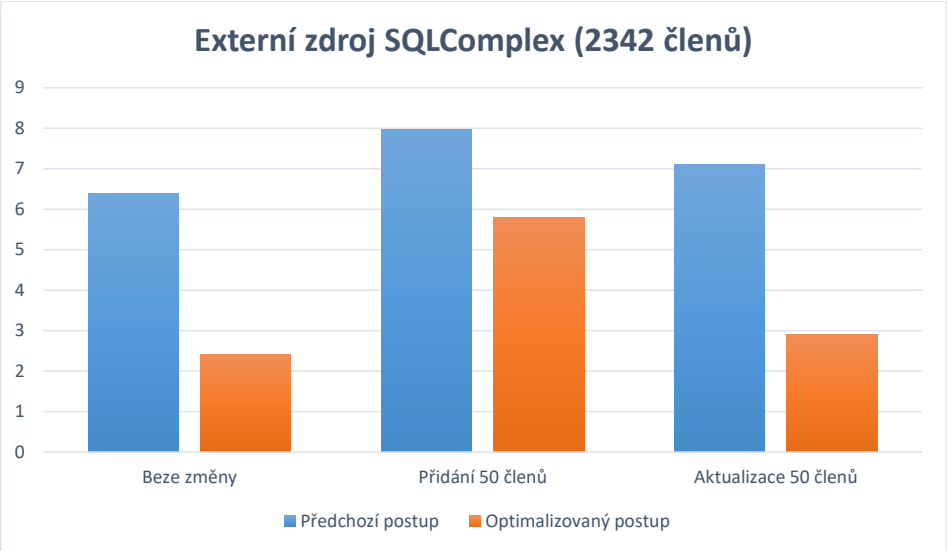



Obrázek 4.1: Graf rychlosti synchronizace skupiny z externího zdroje SQL

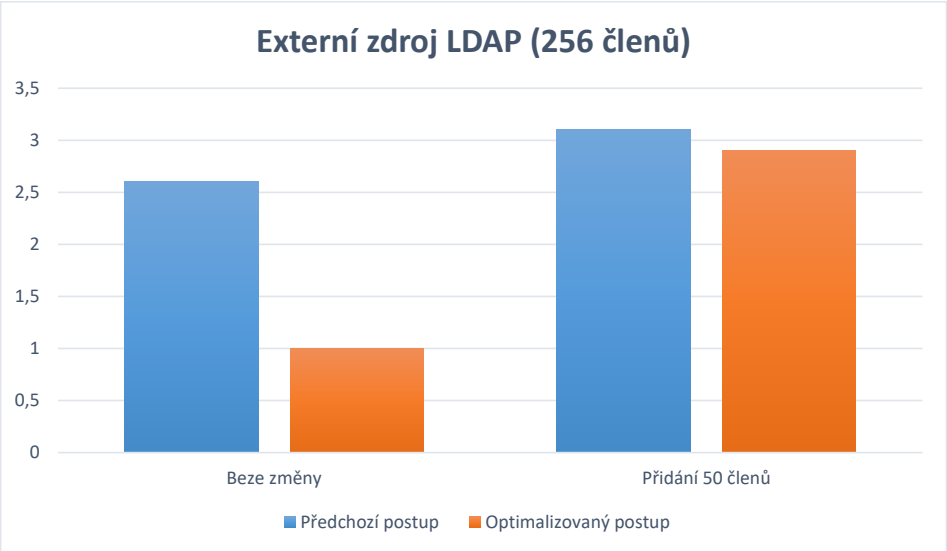
V případě externího zdroje SQLComplex (obrázek 4.2) je optimalizovaná synchronizace modifikovaných členů skupiny víc jak dvakrát rychlejší oproti původní. Opět záleží na množství členů, které je potřeba z externího zdroje získat. Synchronizace skupiny je rychlejší o několik sekund i v případě, kdy se množina uživatelů změní a je tak nutno získat všechny členy skupiny. Lze z toho vypozorovat, že optimalizace zpracování dat v systému Perun pomáhá. Na druhou stranu je přidávání členů o trochu pomalejší z důvodu přidané detekce duplikovaných uživatelů.

V případě externího zdroje LDAP (obrázek 4.3) byl použit pro testování LDAP server sdružení CESNET. Problém v tomto případě byl, že není možno nijak ovlivnit modifikování členů na LDAP serveru a tudíž se nemohla otestovat rychlost v případě získávání jen modifikovaných členů skupiny. LDAP je ale často používaný jen ke čtení dat, která se modifikují velmi zřídka, tudíž je zde zajímavý údaj, kdy se na externím zdroji nic nezměnilo. V takovém případě je rychlost víc jak dvojnásobná. V případě změny množiny uživatelů na externím zdroji je rychlost synchronizace podobná, s narůstajícím počtem členů by ale jistě pomohla optimalizace rozřazování členů a porovnávání otisku k zjištění, zda byl člen od předchozí synchronizace modifikován.

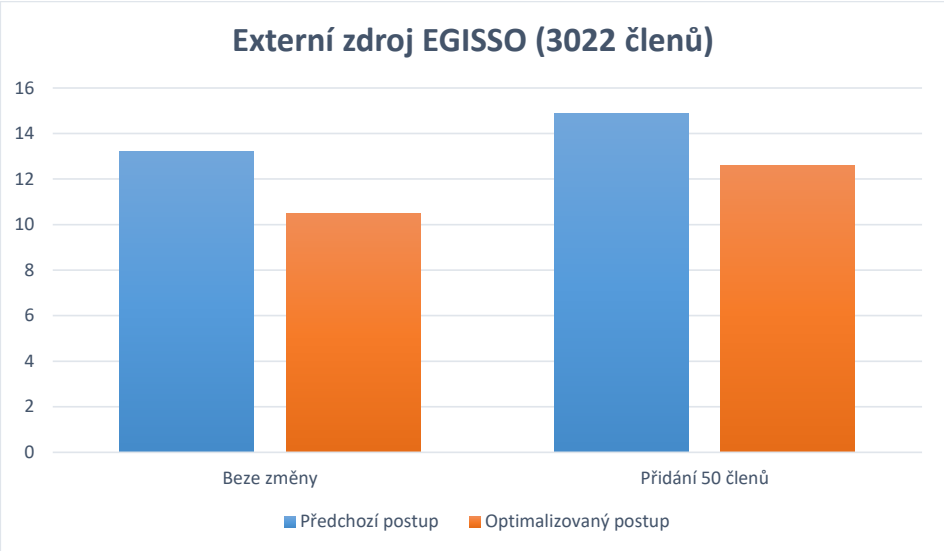
Externí zdroj EGISSO (obrázek 4.4) je používán výhradně pro komunikaci s jedním externím zdrojem EGI infrastruktury. V tomto případě lze detekovat přínos optimalizovaného rozřazování a porovnávání otisků subjektů. Díky tomu je synchronizace o pár sekund rychlejší, i když se musí z externího zdroje získat vždy všichni členové skupiny.




Obrázek 4.2: Graf rychlosti synchronizace skupiny z externího zdroje SQL-Complex




Obrázek 4.3: Graf rychlosti synchronizace skupiny z externího zdroje LDAP




Obrázek 4.4: Graf rychlosti synchronizace skupiny z externího zdroje EGISSO

Pro externí zdroj typu XML byla vybrána *VOMS (Virtual Organization Membership Service)* služba, která data poskytuje v XML formátu. Synchronizovaná skupina má pět tisíc členů, tudíž na ní lze zjistit vliv optimalizovaného zpracování dat v systému Perun. V grafu 4.5 lze vidět, že je rychlost optimalizovaného řešení vždy o několik sekund rychlejší.

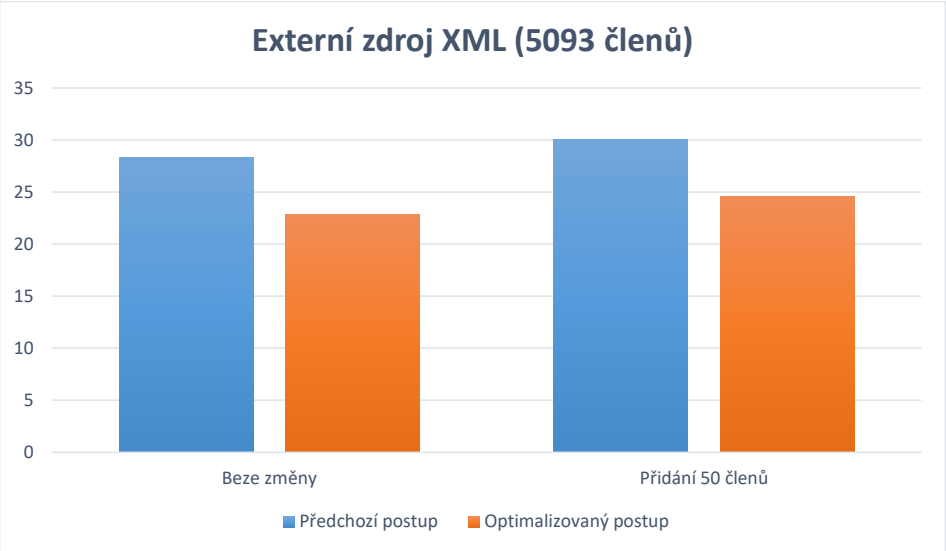
## 4.2 Vyhodnocení výsledků

Z grafů lze vyčíst, že v případě optimalizovaných externích zdrojů typu SQL, SQLComplex a LDAP rychlost minimálně dvojnásobně vzrostla, pokud se synchronizuje stejná množina uživatelů. V takovém případě záleží na množství členů, kteří byli od poslední úspěšné synchronizace na externím zdroji modifikováni.

Synchronizace stejné množiny uživatelů externími zdroji SQL a SQLComplex ukazuje až téměř sedminásobný rozdíl v rychlosti. Proto velice doporučuji převedení veškerých skupin používajících zastaralý externí zdroj SQL na externí zdroj typu SQLComplex.

Změna v rozřazování získaných subjektů a porovnávání otisků členů skupin pro zjištění, zda byli od předchozí synchronizace modifikováni, v některých případech zrychlila proces synchronizace skupiny o několik sekund. Na druhou stranu nově zavedená kontrola duplikace uživatelů mírně zpomaluje proces přidávání nových členů skupiny.

Celkově jsem s rychlostí optimalizovaného řešení spokojený. Přestože jsou




Obrázek 4.5: Graf rychlosti synchronizace skupiny z externího zdroje XML

optimalizovány jen externí zdroje typu SQL, SQLComplex a LDAP, tak to nevadí, protože to jsou zároveň externí zdroje nejpoužívanější a synchronizuje se z nich nejvíce uživatelů.

# Závěr

Cílem práce bylo seznámit se se systémem Perun, analyzovat jej a navrhnout optimalizované řešení získávání dat o uživatelích ve skupinách z externích zdrojů. Dalším požadavkem bylo optimalizovat zpracování získaných dat v systému Perun. Celé navržené řešení bylo nutné implementovat.

Vytvořené optimalizované řešení synchronizace skupin bylo naprogramováno v Javě a zlepšilo komunikaci s externím zdrojem typu relační databáze a LDAP server tak, aby z něj bylo možné získávat jen modifikované uživatele v případě, kdy se množina uživatelů ve skupině nezměnila. Z dalších externích zdrojů není možné získávat efektivně pouze modifikované uživatele z důvodů popsaných v analýze externích zdrojů.

Zpracování získaných dat v systému Perun bylo optimalizováno pomocí nového rychlejšího rozřazování získaných členů skupiny do tří kategorií: noví členové k přidání, modifikovaní členové k aktualizaci a členové k odstranění. Toto rozřazování nyní hledá existující členy v konstantním čase pomocí uživatelského jména v externím zdroji. Dalším implementovaným zlepšením je ukládání hashe přijatých dat o uživateli v atributu vázaném ke vztahu člen-skupina, díky kterému můžeme zjistit, zda se přijatá data o uživateli od předchozí synchronizace změnila a jestli je nutné provádět aktualizaci člena. Další implementovanou novinkou je detekce duplicitních uživatelských účtů, které by se měly spojit do jednoho. V takovém případě systém Perun vyhodí výjimku s hlášením pro administrátora.

Optimalizované řešení synchronizace skupin je v případě získávání pouze modifikovaných členů z relačních databází či LDAP serveru více jak dvakrát rychlejší. Vylepšené zpracování uživatelských dat v systému Perun zrychlilo synchronizaci řádově o sekundy u skupin s tisíci uživateli (např. zrychlení o šest sekund u skupiny s pěti tisíci členy importovanými z externího zdroje typu XML soubor). Celkově jsem s nárůstem výkonu spokojen.

Pro další zlepšení výkonu synchronizace dat bych doporučil převést všechny existující skupiny používající zastaralý externí zdroj typu SQL na externí zdroj typu SQLComplex, který získá všechna data o uživatelích jedním dotazem

a nedoptává se relační databáze pro každého člena zvlášť tak, jak to děla externí zdroj SQL. To by přineslo výrazné zrychlení synchronizačních cyklů.

# Literatura


[1] CESNET, z. s. p. o: Logo systému Perun [online]. 2016, [cit. 2017-05-01]. Dostupné z: https://perun.cesnet.cz/web/img/logo/perun-logo-transparent.png

[2] CESNET, z. s. p. o: Konceptuální schéma [online]. Březen 2017, [cit. 2017-05-01]. Dostupné z: https://wiki.metacentrum.cz/w/images/f/fe/Concept_scheme.png

[3] soLNet, s.: *WebIS – manuál [online].* 2001-2015, [cit. 2016-12-09]. Dostupné z: http://www.solnet.cz/files/manual/webis/2008_2/sync.html

[4] CESNET, z. s. p. o.: Perun, Identity and Access Management System, About us [online]. Leden 2016, [cit. 2016-12-05]. Dostupné z: https://perun.cesnet.cz/web/about.shtml

[5] CESNET, z. s. p. o.: Wiki MetaCentrum, Conceptual scheme and definition of terms [online]. Srpen 2015, [cit. 2017-02-21]. Dostupné z: https://wiki.metacentrum.cz/wiki/Conceptual_scheme_and_definition_of_terms

[6] LICEHAMMER, S.: *Správa atributů systému Perun [online].* Diplomová práce, Masarykova univerzita, Fakulta informatiky, Brno, 2012 [cit. 2017-05-11]. Dostupné z: http://is.muni.cz/th/255920/fi_m/

[7] ŠŤAVA, M.: Homeproj: Redmine for CESNET, ExtSources a práce s nimi - Perun [online]. Říjen 2016, [cit. 2017-02-27]. Dostupné z: https://homeproj.cesnet.cz/projects/perun/wiki/ExtSources_a_prace_s_nimi

[8] Refsnes Data: *w3schools.com - SQL Tutorial [online].* 1999-2017, [cit. 2017-03-29]. Dostupné z: https://www.w3schools.com/sql/

[9] BOUŠKA, P.: Samuraj-cz, Adresářové služby a LDAP. *Samuraj-cz [online]*, Září 2007, [cit. 2017-02-10]. Dostupné z: http://www.samuraj-cz.com/clanek/adresarove-sluzby-a-ldap/

[10] EGI: Intranet - EGIWiki [online]. Listopad 2016, [cit. 2017-03-29]. Dostupné z: https://wiki.egi.eu/wiki/Intranet

[11] Microsoft: *When-Changed attribute (Windows) [online]*. 2017, [cit. 2017-03-29]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms680921(v=vs.85).aspx

[12] Edoceo: Comma Separated Values (CSV) Standard File Format [online]. 1999-2016, [cit. 2017-03-29]. Dostupné z: http://edoceo.com/utilitas/csv-file-format

[13] W3C: *Extensible Markup Language (XML) 1.0 (Fifth Edition) [online]*. Listopad 2008, [cit. 2017-03-30]. Dostupné z: https://www.w3.org/TR/REC-xml/

[14] Refsnes Data: *XPath Tutorial [online]*. 1999-2017, [cit. 2017-03-30]. Dostupné z: https://www.w3schools.com/xml/xpath_intro.asp

[15] Refsnes Data: *HTTP Methods GET vs POST [online]*. 1999-2017, [cit. 2017-04-01]. Dostupné z: https://www.w3schools.com/tags/ref_httpmethods.asp

[16] CONNOLLY, D.: *HTTP/1.1: Caching in HTTP [online]*. Listopad 2004, [cit. 2017-03-31]. Dostupné z: https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html

Příloha A

# Seznam použitých zkratek

**CERIT-SC** CERIT Scientific Cloud

**CESNET** Czech Education and Scientific Network

**CSV** Comma-separated values

**EGI SSO** EGI Single Sign On

**LDAP** Lightweight Directory Access Protocol

**SQL** Structured Query Language

**SSH** Secure Shell

**SVN** Subversion

**UES** User External Source

**URL** Uniform Resource Locator

**VO** Virtual Organization

**VOMS** Virtual Organization Membership Service

**XML** eXtensible Markup Language

Příloha **B**

# Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `perun` ..... zdrojové kódy systému Perun
  - `changes.patch` ..... výpis změn v kódu systému Perun
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `diagrams` ..... diagramy použité v práci
    - `pictures` ..... obrázky použité v práci
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF