Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Návrh a implementace datového modelu vizualizace přenosové soustavy

Plzeň 2018 Přemysl Kouba

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 28. června 2018

Přemysl Kouba

Abstract

The main goal of this bachelor thesis is creating the model of electric power transmission together with design and implementation of data exchange between application layers. This thesis is the second part of the ongoing project which was created in cooperation between Faculty of Applied Sciences and Department of Cybernetics. The thesis contains graph theory together with Java graph libraries description and comparison in the first part. Next chapter contains an analysis of data model and description of inter-component communication. Last part of this thesis include the description of the implemented data and improvements of the application data exchange.

Suggestions made in conclusion of this thesis can be used in future in subsequent bachelor or master thesis.

Abstrakt

Hlavním tématem této bakalářské práce je vytvoření modelu elektrické přenosové sítě spolu s návrhem a implementací výměny dat mezi vrstvami aplikace. Tato bakalářská práce navazuje na sérii předchozích, které vznikly v rámci spolupráce Katedry informatiky a výpocetní techniky a Katedry kybernetiky. V první části práce je popsána obecná teorie grafů spolu s popisem dostupných JAVA knihoven. V další kapitole je uveden detailní rozbor použitého datového i fyzického modelu přenosové sítě včetně popisu komunikace mezi jednotlivými komponentami. Následuje popis současného stavu architektury aplikace. V poslední části této bakalářské práce jsou pak navržena řešení datových modelů a vylepšení výměny dat.

Návrhy obsažené v závěru této práce mohou být využity v budoucnu v navazujících bakalářských nebo diplomových pracích.

Obsah

1 Úvod

Elektrická přenosová síť je definována jako soubor jednotlivých vzájemně propojených elektrických stanic, venkovních a kabelových vedení pro přenos a rozvod elektrické energie [22]. Uvnitř tohoto souboru zajišťuje přenos energie takzvaná přenosová síť vedení velmi vysokého napětí. Elektrické linky propojují jednotlivé zdroje a transformační stanice, aby bylo možno operativně řídit přenos energie v závislosti na okamžité spotřebě elektřiny v různých oblastech i v případě poruchy na některé části sítě.

Na přenosovou síť lze nahlížet také jako na graf, kde jednotlivé uzly reprezentují elektrická zařízení. Za hrany grafu pak lze považovat spojovací body. Elektrické přenosové sítě mohou nabývat rozměrů od několika zařízení po mezinárodní sítě. K lepší a rychlejší orientaci v datech o aktuálním stavu přenosové sítě slouží různé druhy vizualizací.

Spoluprací Katedry kybernetiky a Katedry informatiky a výpočetní techniky vznikl projekt, který má za cíl vytvořit webovou aplikaci pro vizualizaci přenosové sítě. Ta bude zobrazovat přenosovou soustavu na geografické mapě s možností snadné změny parametrů a následného výpočtu nového stavu sítě. Vzhledem k možným velkým rozměrům zobrazované sítě je v rámci projektu kladen důraz na efektivitu implementovaných výpočtů a použitého programovacího jazyka.

Tato práce je součástí rozsáhlejšího projektu. Zabývá se vytvořením komunikace datové vrstvy s webovým rozhraním na již vytvořeném prototypu serveru. Data musejí odpovídat danému formátu a splňovat podmínky daného datového modelu. Náplní práce je také reprezentace přenosové sítě jako grafu.

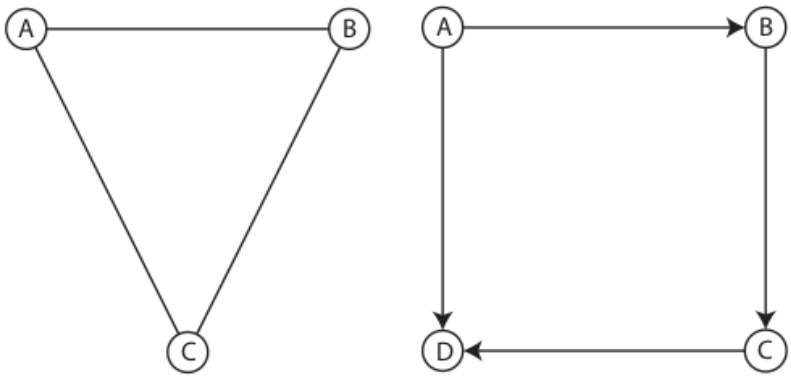
2 Technologie reprezentace grafu v jazyce Java

2.1 Teorie grafů

Necht' V je konečná množina. Dvojice $G = (V(G), H(G))$, kde $H(G) \subseteq V(G) \times \binom{V(G)}{2}$, se nazývá **smíšený graf**.

Je-li speciálně $H(G) \subseteq V(G) \times \binom{V(G)}{2}$,pak se G nazývá **neorientovaný graf**. Prvky množiny $V(G)$ budeme nazývat vrcholy(uzly), prvkům množiny $H(G)$ budeme říkat **hrany** grafu G. [7, str.1]

Graf je zobrazení množiny objektů a jejich spojení – skládá se z vrcholů (reprezentace objektů) a hran (jejich propojení). Graf je uspořádaná dvojice $G = (V(G), H(G))$, kde $V(G)$ je množina vrcholů a $H(G)$ množina hran. U orientovaných grafů záleží na směru propojení (tzv. orientované hrany). U neorientovaných naopak na směru nezáleží (viz 2.1). Hrany některých grafů mohou být ohodnoceny. [1]



Obrázek 2.1: Orientovaný (vlevo) a neorientovaný (vpravo) graf

V rámci vyvíjené aplikace definujeme množinu vrcholů $V(G)$ jako množinu doménových objektů. Množina uzlů $H(G)$ je poté množinou spojových bodů. Očekáváme, že vrchol V_1 bude propojen s vrcholem V_2 pomocí hrany H_1, kde typicky V_1 bude sběrnice a V_2 bude elektrické vedení viz obr. 3.2. Zároveň předpokládáme neorientovaný graf. Pro přesnost, množina spojových bodů je specifickou podmnožinou doménových objektů [kapitola 3.1].

2.2 Implementace grafů

V informatice lze grafy implementovat dvěma základními způsoby. Prvním způsobem je implementace grafu pomocí matice sousednosti a druhým způsobem je implementace pomocí spojového seznamu sousedních prvků.

2.2.1 Matice sousednosti

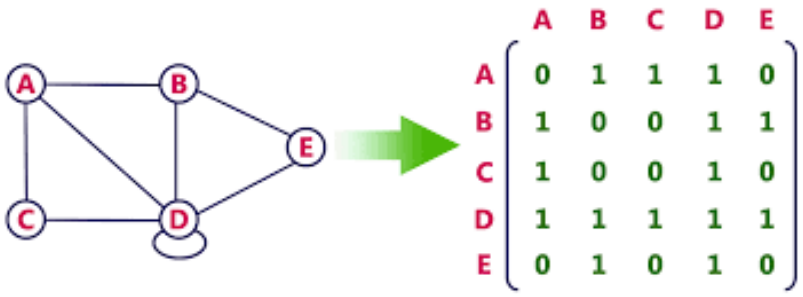
Nechť G je neorientovaný graf. Matice M(G) typu n/m definovaná předpisem

$$m_{i,j} = \begin{cases} 1, & \text{jestliže } v_1 \in h_j, \\ 0 & \text{jinak,} \end{cases}$$

se nazývá **vrcholově-hranová incidenční matice grafu** G. Převod grafu na matici sousednosti je ilustrován na obrázku (2.2). Další maticové reprezentace jsou pak vrcholově-hranová incidenční matice orientovaného grafu, redukovaná incidenční matice orientovaného grafu, matice vzdálenosti a Laplaceova matice [7].

Výhodou matice sousednosti je jednoduchost programování. Tato reprezentace je vhodná pro husté grafy (počet hran se blíží druhé mocnině počtu vrcholů). Přidání, odebrání hran a zjištění sousednosti jsou rychlé operace.

Nevýhodou tohoto přístupu je, že u řídkých grafů (počet hran je řádově menší než druhá mocnina počtu vrcholů) je matice stejně velká jako u hustých grafů. Při přidání nových vrcholů je pak potřeba rozšířit matici.



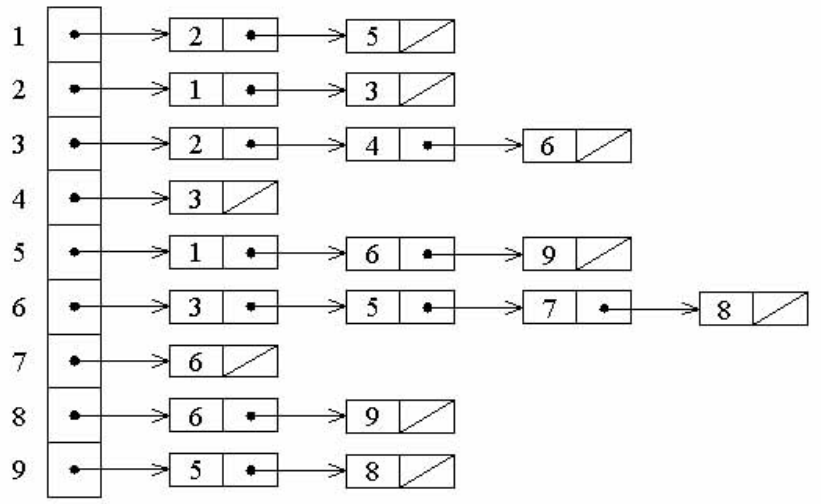
Obrázek 2.2: Reprezentace grafu maticí sousednosti

2.2.2 Spojový seznam

Každý vrchol grafu má seznam sousedů – tedy odkaz na všechny svoje sousedy. Tyto seznamy jsou obecně uloženy ve spojových seznamech v libovolném pořadí. K těmto seznamům je možné přidat také ohodnocení vzdálenosti (obr. 2.3).

Výhodou tohoto přístupu je jednoduché přidání dalších vrcholů a obecně menší paměťová náročnost (závislá zejména na počtu hran, nikoliv vrcholů). Dalším kladem je snadné zjištění stupně vrcholu.

Nevýhodou je pak zjišťování existence hrany - tedy hledání hledání vrcholu x_1 v seznamu sousedů vrcholu x_2. Další nevýhodou je vyšší paměťová náročnost u hustých grafů (může být dokonce vyšší, než u matice sousednosti).



Obrázek 2.3: Reprezentace grafu seznamem sousednosti

	Matice	Seznam
Sousednost vrcholů	$\theta(1)$	$\theta(E/V)$
Nalezení potomka	$\theta(V)$	$\theta(E/V)$
Nalezení předka	$\theta(V)$	$\theta(E+V)$
Zjištění stupně vrcholu	$\theta(V)$	$\theta(1)$
Paměťová náročnost	$\theta(V^2)$	$\theta(E+V)$

Tabulka 2.1: Složitost operací nad jednotlivými implementacemi, kde E je počet hran a V je počet vrcholů.[20]

2.3 Grafové algoritmy

Nad grafy se dá provádět velké množství operací – např. hledání nejkratší cesty, kostry grafů atd. V rámci této práce jsou rozebrány pouze ty operace, které budou následně implementovány.

2.3.1 Nalezení nejkratší cesty

- **Prohledávání do šířky**
  Algoritmus slouží k prohledání všech vrcholů grafu a nalezení nejkratší cesty. Využívá frontu.

- **Prohledávání do hloubky**
  Nalezne cestu mezi dvěma vrcholy za pomoci zásobníku. Nalezená cesta nemusí být vždy nejkratší.

- **Dijkstrův algoritmus**
  Vyhledávání nejkratší cesty z jednoho vrcholu do druhého. Vyžaduje nezáporné ohodnocení hran.

- **A* algoritmus**
  Hledání nejkratší cesty v nezáporně ohodnoceném grafu s přidáním heuristického prvku.

- **Floyd-Warshallův algoritmus**
  Algoritmus nalezne nejkratší vzdálenost mezi všemi vrcholy i se záporným ohodnocením hran. Je však náchylný na cykly v grafu.

- **Bellman-Fordův algoritmus**
  Podobně jako Floyd-Warshallův algoritmus hledá cesty s nejnižším ohodnocením. Tento algoritmus však hledá cestu pouze z daného vrcholu do okolních.

2.3.2 Nalezení kostry grafu

Kostra grafu je podgraf, který neobsahuje žádné cykly. To znamená, že libovolné dva vrcholy lze spojit jen jednou cestou. Cílem tohoto algoritmu je nalézt minimální kostru – nejmenší možný podgraf.

- **Kruskalův algoritmus**
  Při každém kroku vybere vždy nejméně ohodnocenou hranu, která netvoří s již vybranými smyčku.

- **Borůvkův algoritmus**
  Hledá kostru grafu s nezáporně a různě ohodnocenými hranami.

- **Jarníkův algoritmus**
  Hledá minimální kostru ohodnoceného grafu.

2.3.3 Hledání komponent

Úlohou je nalezení takového souvislého podgrafu, který není obsažen v žádném jiném větším souvislém podgrafu.

2.3.4 Nahrazení uzlů

Tato úloha je velmi specifická, neboť nemá algoritmický základ. Pro vyvíjenou aplikaci je však potřeba. Cílem je seskupit hrany a uzly tak, aby nedošlo ke ztrátě informací o přenosové síti a byl redukován jejich celkový počet. Výsledkem bude redukovaný graf, ve kterém se ale zachovají data a vlastnosti původního grafu.

2.4 Knihovny

Pro programovací jazyk Java existují knihovny určené k tvorbě grafů. Následující kapitola porovnává některé z nich v úlohách, které budou použity v aplikaci „Vizualizace přenosové sítě“.

2.4.1 Graph Stream

Webová adresa	http://graphstream-project.org/
Licence	CeCILL-C a LGPL v3.
Aktuální verze[1]	v1.3 - 2015
Autoři	S. Balev, J. Baudry, A. Dutot, Y. Pigné, G. Savin; University of Le Havre; LITIS computer science lab

GraphStream [3] je knihovna vyvíjená na University of Le Havre určená k modelování a analýze dynamických dat. Nabízí možnosti generování, importu, exportu, měření a vizualizace těchto dat. Poskytuje několik tříd grafů, které umožňují pracovat s různými typy orientovaných a neorientovaných grafů a multigrafů. GraphStream dovoluje v reálném čase pracovat s daty a změny okamžitě zobrazovat. Knihovna je rozdělena do tří balíků. Prvním je

[1] 12.04.2018

`gs-core`, který je kruciální, jelikož obsahuje implementaci grafů. Dalšími balíky jsou `gs-algo`, který obsahuje implementaci grafových algoritmů a balík `gs-ui` starající se o uživatelské rozhraní vizualizace grafu. V současné době vývoj již nepokračuje v oblasti jádra, avšak tvůrci pracují na multiplatformovosti a podpoře.

Práce s knihovnou

Graf je v GraphStream definován jako rozhraní *Graph*. Jednotlivé typy grafů jsou pak implementacemi tohoto rozhraní. Uzly a hrany implementují rozhraní *Node*, respektive *Edge*. Při vkládání nového uzlu nebo hrany musí být vždy definován řetězec *Id* sloužící k identifikaci prvku grafu. Prvky grafu jsou uloženy v mapě. Algoritmy jsou, podobně jako grafy, implementací rozhraní *Graph*.

Dokumentace

Na webových stránkách je k dispozici velká řada tutoriálů a rozsáhlá ***Javadoc*** dokumentace. V jednotlivých třídách je pak mnoho komentářů k použití knihovny.

Algoritmy

Algoritmy knihovny GraphStream se nacházejí balíku `gs-alg`. Většina grafů implementuje rozhraní *DynamicAlgorithm*. Při změně dat provede knihovna aktualizaci grafu. Podporované úlohy jsou uvedeny v tabulce 2.2.

algoritmus	**dostupnost**	**typ algoritmu**
Nalezení nejkratší cesty	ANO	Dijkstrův alg, A* algoritmus
Nalezení kostry grafu	ANO	Prim-Jarníkův algoritmus, Kruskalův algoritmus
Vyhledávání komponent	ANO	nespecifikováno

Tabulka 2.2: Tabulka algoritmů poskytovaných knihovnou GraphStream

2.4.2 JUNG

Webová adresa	http://jung.sourceforge.net/
Licence	Berkeley Software Distribution (BSD) license
Aktuální verze[2]	v2.0.1 - 24.1.2010
Autoři	Joshua O'Madadhain, Danyel Fisher a Scott White

Celý názvem „Java Universal Network/Graph Framework“ [18] je knihovna která vznikla mezi lety 2003 a 2010 na půdě Kalifornské univerzity. Knihovna nabízí flexibilní a robustní API pro modelování, analýzu a vizualizaci grafů. Architektura knihovny je navržená tak, aby podporovala velké množství reprezentací grafů a jejich vztahů jako jsou orientované a neorientované grafy, multigrafy a grafy s paralelními hranami. Knihovna nabízí mnoho algoritmů, mezi nimi i algoritmy pro data mining a následnou vizualizaci těchto dat.

Práce s knihovnou

Knihovna reprezentuje graf prostřednictvím rozhraní *Graph*. Pomocí tohoto rozhraní se přistupuje ke všem údajům jednotlivých uzlů a hran, které se nacházejí v daném grafu. Uzly i hrany mohou být reprezentovány libovolným objektem, je tedy nutné vlastnit odkaz na objekt třídy reprezentující uzel nebo hranu, abychom mohli zjistit, zda se v grafu nachází, či naopak. Cílené algoritmy jsou také implementací rozhraní. Při dodržení rozhraní je možné vytvářet vlastní algoritmy.

Dokumentace

Na webových stránkách [18] je k dispozici několik dokumentačních souborů ve formě prezentací a textu. Knihovna také disponuje vygenerovanou ***Javadoc*** dokumentací, která je velmi slušně zpracována. Bohužel mnoho odkazů z webových stránek je slepých, protože projekt je již delší dobu neudržován.

Algoritmy

Algoritmy knihovny jsou umístěny v balíku *edu.uci.ics.jung.algorithms*, kde jsou dále rozděleny do kategorií. Knihovna poskytuje rozhraní pro implementaci vlastních algoritmů. Jednotlivé algoritmy jsou vidět v tabulce 2.3.

algoritmus	**dostupnost**	**typ algoritmu**
Nalezení nejkratší cesty	ANO	Dijkstra, BFS
Nalezení kostry grafu	ANO	Prim-Jarníkův algoritmus
Vyhledávání komponent	ANO	nespecifikováno

Tabulka 2.3: Tabulka algoritmů poskytovaných knihovnou JUNG

[2] 12.04.2018

2.4.3 JGraphT

Webová adresa	http://jgrapht.org/
Licence	LGPL a EPL
Aktuální verze[3]	v1.10 - 15.11.2017
Autoři	Barak Naveh a přispěvatelé

JGraphT je grafová knihovna [10] vyvíjená od roku 2003, která je jednou z nejvyužívanějších. Knihovna se soustředí na práci s orientovanými a neorientovanými grafy a multigrafy. Pro každý typ grafu umožňuje i práci s podgrafy. Pro vizualizaci grafů je potřeba využít knihovnu JGraphX. Obě zmíněné knihovny spolu komunikují pomocí JGraphXAdaptéru a jsou neustále vyvíjeny.

Práce s knihovnou

S grafem knihovna pracuje prostřednictvím generického rozhraní:
Graph<V,E>
V je třída reprezentující uzly a E je třída reprezentující hrany. Samotné typy grafů jsou pak definovány různými třídami implementujícími toto rozhraní. Podobně jako při použití knihovny **JUNG** je nutné vlastnit odkaz na prvek, abychom se mohli dotázat, zda-li náleží grafu.

Dokumentace

Na webových stránkách knihovny nalezneme kromě základních informací jakýsi rozcestník na další zdroje. Jedním ze zdrojů je odkaz na Github, kde probíhá většina vývoje. Dále zde nalezneme velmi podrobnou **Javadoc** dokumentaci, Wiki stránky a odkaz na malou ukázku toho, jak se s knihovnou pracuje. Také zde najdeme odkaz na portál StackOverflow, kde je zodpovězeno mnoho otázek ohledně knihovny.

Algoritmy

Algoritmy se nachází v balíku *org.jgrapht.alg* a jsou rozděleny do mnoha kategorií. Jednotlivé třídy algoritmů neimplementují jednotné rozhraní, ale většinou jsou vytvářeny nad objektem typu *Graph*. Knihovna nabízí velké množství algoritmů. Algoritmy využívané naší aplikací jsou vidět v tabulce 2.4.

[3] 12.04.2018

algoritmus	dostupnost	typ algoritmu
Nalezení nejkratší cesty	ANO	Dijkstrův, Floyd-Marshallův, Bellman-Fordův, A* algoritmus
Nalezení kostry grafu	ANO	Prim-Jarníkův, Kruskalův a Borůvkům algoritmus
Vyhledávání komponent	ANO	Velké množství algoritmů

Tabulka 2.4: Tabulka algoritmů poskytovaných knihovnou JGraphT

2.4.4 Další knihovny

Apache Commons Graph

V současnosti nečinný projekt [2] malého rozsahu pod licencí Apache License Version 2.0, který implementuje algoritmy hledání nejkratší cesty (Dijkstrův algoritmus) a hledání nejmenší kostry (Borůvkův, Kruskalův a Prim-Jarníkův algoritmus). Knihovna je velmi špatně popsána a k dispozici jsou jen generické webové stránky.

yFiles

Stále vyvíjená komerční knihovna yFiles [19] společnosti yWorks podporuje širokou škálu algoritmů i vizualizačních prostředků. Má velmi podrobnou a kvalitní dokumentaci stejně jako širokou podporu.

BFO

Komerční knihovna [15] nabízí kromě tvorby grafů i jejich nasazení do webových aplikací, dále export grafů do grafickch formátů png, svg, Macromedia Flash, pdf nebo `java.awt.Image` a vizualizaci ve 2D i 3D.

2.5 Zvolená knihovna

Vybranou knihovnou pro projekt vizualizace přenosové sítě se stala knihovna JGraphT. Tato knihovna nabízí nejen velké množství algoritmů pro specifikované úkoly (tabulka 2.4), ale také širokou základnu uživatelů a podporovatelů. Lze předpokládat že se vývoj této knihovny v nejbližší době nezastaví. Další výhodou této knihovny je fakt, že oproti ostatním nabízí i specifické algoritmy pro hledání komponent grafu a nástroje pro práci s podgrafy. Na obrázku 2.4 je vidět srovnání knihoven podle možnosti řešení daných úloh.

Nalezení nejkratší cesty					
	Dijkstrův	**BFS**	**A***	**Floyd-Marshallův**	**Bellman-Fordův**
GraphStream	ANO	NE	ANO	NE	NE
JUNG	ANO	ANO	NE	NE	NE
JGraphtT	ANO	NE	ANO	ANO	ANO

Nalezení kostry grafu			
	Prim-Jarníkův	**Kruskalův**	**Borůvkův**
GraphStream	ANO	ANO	NE
JUNG	ANO	NE	NE
JGraphtT	ANO	ANO	ANO

Nalezení komponent grafu		
	Dostupnost	**Algoritmus:**
GraphStream	ANO	Nespecifikováno
JUNG	ANO	Nespecifikováno
JGraphT	ANO	Kosarajův, Gabowův a další

Obrázek 2.4: Srovnání grafových knihoven

3 Datový model

Každému objektu reálného světa lze přiřadit nějaký objekt informačního systému. Datový model má za úkol poskytovat informace o částech reálného světa, je tedy určitým druhem modelu reality. Před vybudováním databáze je tak nutné analyzovat realitu a vybrat objekty, o nichž chceme v databázi udržovat informace.

Tato práce je modelem reality elektrické přenosové sítě. K tomu, abychom mohli tuto síť modelovat, je potřeba analyzovat problematiku přenosových sítí. To v praxi znamená určit si, které prvky přenosové sítě jsou potřebné k tomu, aby výsledek složený z těchto entit odpovídal realitě. Tvorba modelu přenosové sítě není náplní této práce a byl již dán v zadání ERA modelem, který naleznete v příloze g. Tento model přenosové sítě vychází z normy [8].

Datový model jako celek lze rozdělit na tři nejdůležitější části :

- Doménové objekty
- Živá data
- Model (celek)

Mezi doménové objekty řadíme aktivní prvky elektrické sítě. Do živých dat patří vlastnosti těchto doménových objektů, které se mohou v čase měnit. Na model pohlížíme jako na grafovou reprezentaci celku.

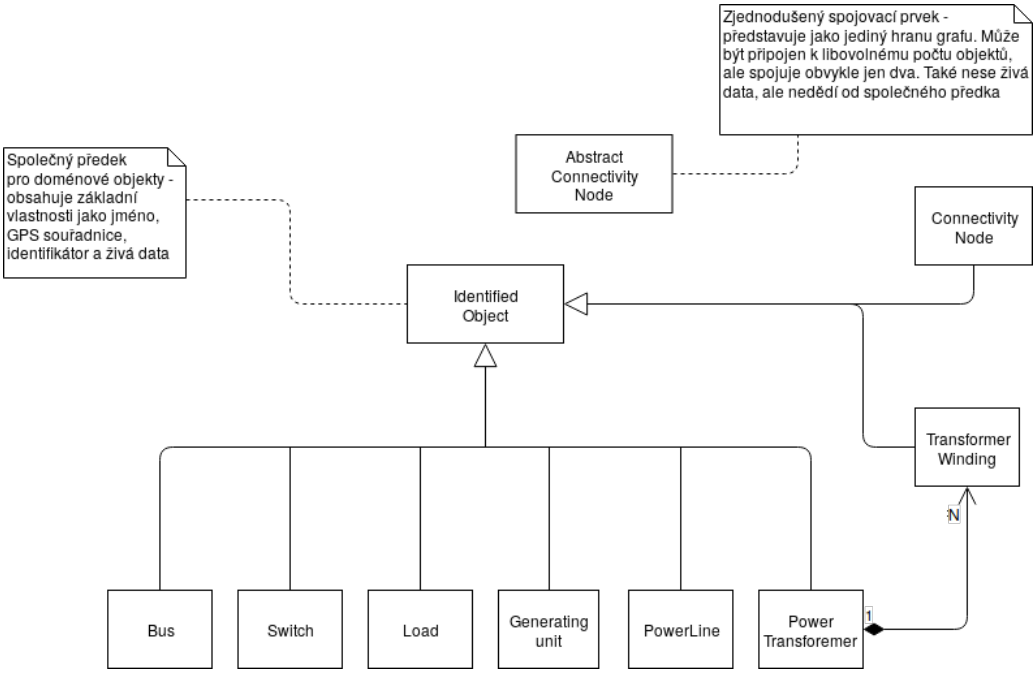
3.1 Doménové objekty

Mezi doménové objekty patří veškeré objekty a jejich atributy, které se v čase nemění. Všechny doménové objekty (kromě `AbstractConnectivityNode`) dědí od společného předka, kterým je třída `IdentifiedObject`. Doménové objekty se dále dají rozdělit na fyzické prvky elektrické sítě a jejich vlastnosti. Strukturu doménových objektů ilustruje velmi zjednodušený diagram na obrázku 3.1

Mezi fyzické prvky patří :

- `Bus` (sběrnice) – sběrnice slouží k rozvodu elektrické energie bez její transformace.
- `Load` (zátěž) – veškeré elektrické prvky, které spotřebovávají elektrický proud přenosové soustavy. Mezi zátěžové prvky můžeme zahrnout lampu pouličního osvícení i těžebnu kryptoměn.

- `GeneratingUnit` (generátor) – množina elektrických prvků generujících elektrický proud do přenosové sítě. Jako generátor je možné označit solární panel i jadernou elektrárnu.
- `PowerLine` (vedení elektrického proudu) - vedením elektrického proudu označujeme veškeré dráty, které propojují zbylá elektrická zařízení.
- `PowerTransformer` (transformátor) – zařízení, které umožňuje přenášet elektrickou energii z jednoho obvodu do jiného.
- `Switch` (spínač) – spínač je zařízení, které umožňuje odpojení prvků, které následují za spínačem.



Obrázek 3.1: Velmi zjednodušený diagram doménových objektů

Tyto prvky pak mají mnoho vlastností. Za zmínku stojí GPS souřadnice, které udávají pozici prvků. Dalo by se tak tedy tvrdit, že již není nic dalšího potřeba k tomu, aby se dal složit model elektrické sítě.

Z pohledu uživatele se elektrická přenosová síť skutečně skládá pouze z prvků sítě a elektrického vedení, kde prvky sítě jsou vrcholy a elektrické vedení jsou hrany. Takováto představa modelu by ale **neodpovídala** realitě. Samotné elektrické vedení je také součástí elektrické přenosové sítě a má mnoho důležitých vlastností. Zároveň se dá tvrdit, že elektrické vedení je druhem zátěže – při přenosu energie dochází ke ztrátám, které se projevují zahřátím vedení. Zavádí se proto takzvané spojovací prvky modelu.

3.1.1 Spojovací prvky modelu

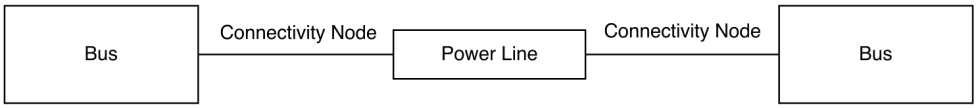
Spojovacím prvkem modelu jsou objekty typu `ConnectivityNode` (spojovací bod), který je **skutečnou hranou grafu**. Tento spojovací prvek není dobré zanedbat i z fyzikálního hlediska – při přenosu energie dochází v prvcích přenosové sítě ke ztrátám.

Například pokud chceme určit, jak velká ztráta nastala na elektrickém vedení, musíme si naměřit data na začátku a na konci elektrického vedení. Tato data se poté uloží příslušným spojovacím prvkům a rozdílem v těchto datech zjistíme, k jak velké ztrátě na vedení došlo.

Každý spojovací prvek nese identifikátory doménových objektů, které propojuje. Podle úrovně abstrakce rozlišujeme :

- `ConnectivityNode` – tento typ objektů dědí od společného předka `PowerSystemResource` a získává tak malou množinu vlastností doménových objektů. Mezi tyto vlastnosti patří například GPS souřadnice.
- `AbstractConnectivityNode` – je zjednodušením předchozího. Tyto abstraktní objekty již nepatří mezi doménové objekty, ale mohou nést živá data.

Na obrázku 3.2 je vidět skutečné propojení dvou sběrnic elekrickým vedením.



Obrázek 3.2: Propojení dvou sběrnic přenosové sítě a elektrického vedení

3.1.2 Transformátor

Dalším velmi zajímavým prvkem přenosové sítě je transformátor. Transformátor je elektrické zařízení, které umožňuje přenášet elektrickou energii z jednoho obvodu do jiného pomocí elektromagnetické indukce. Používá se většinou pro přeměnu střídavého napětí nebo pro galvanické oddělení obvodů. Transformátor jako zařízení se skládá z několik cívek. Pokud tedy chceme modelovat transformátor, je dobré si jej rozdělit na samotný transformátor a vinutí cívky. Skutečný transformátor má nejméně dvě cívky, ale také jich může být N. Pro více technických informací o transformátorech včetně detailního popisu viz [5].

3.2 Živá data

Ke každému doménovému objektu přísluší množina dat, která je označována jako živá data. Jedná se o fyzikální vlastnosti, které se mohou měnit v závislosti na čase. Tato data jsou sdružována do množin zvaných `workspace`. Příkladem těchto množin mohou být naměřená data, nebo data získaná z nějakého výpočtu. Tato data jsou také součástí datového modelu a jsou ukládána do databáze. Pro zjednodušení ukládání do databáze a lidskou čitelnost ukládáme tato data jako řetězce v JSON formátu. Nemusíme tím řešit rozklad mapy map na jednotlivé tabulky.

Ukázka toho, jak jsou živá data definována na serveru. Instance třídy `IdentifiedObject` mají atribut `liveData`, který obsahuje živá data. V následující ukázce je uvedena přesná definice živých dat.

```
HashMap<ValueType, liveDataSet> liveData
```

Živá data jsou ukládána v mapě, kde klíčem je instance třídy `ValueType` (workspace) a hodnotami pro každý workspace jsou naměřená živá data. Tato živá data jsou uložena taktéž v mapě, kde klíčem je instance třídy `Variable` (určení, o jakou hodnotu se jedná) a samotné hodnoty jsou uloženy jako prvky supertřídy `Object`. Znamená to tedy, že data mohou být jakéhokoliv typu a uživatel by měl vědět, jak s takovými daty nakládat.

Třída `ValueType` definuje, jaká data mohou být označena jako živá – **definice workspace** množin. Uvnitř této třídy je slabá hash mapa, kde každý workspace musí být definován jen jednou.

```
HashMap<Variable, ArrayList<Object>> liveDataSet
```

To samé platí i pro třídu `Variable`. Je to slabá hash mapa **definic hodnot**. Továrna poskytující instance zajistí, že bude existovat vždy jen jedna definice hodnot. To ve výsledku znamená úsporu paměti. Skutečné hodnoty pak mohou být uloženy i po více než jedné. Například výsledkem měření nemusí být jen jedna hodnota (nemusí být žádný výsledek), ale celá množina výsledků. Tyto výsledky pak mohou být zprůměrovány do jedné hodnoty, ale aplikace dovoluje uložit přímo naměřené hodnoty.

Velmi jednoduchá ukázka živých dat pro sběrnici :

```
"{
    "Loaded": {
        "Ground resistance [Ohm]": "[NaN]",
        "Ground reactance [Ohm]": "[NaN]"
    }
}"
```

Díky tomu, že jsou živá data ukládána jako čistý text, není nutné kontrolovat, jaké hodnoty se vkládají. Na rozdíl od doménových hodnot tedy lze vložit hodnoty jako NaN (není číslo) a inf (nekonečno), které jinak do MySQL databáze vložit nejdou.

3.3 Model

Pokud máme nadefinovány doménové objekty a živá data, můžeme složit model (graf). Modelem tedy chápeme množinu prvků sítě s jejich statickými vlastnostmi a dále množinu živých dat, které přísluší daným prvkům sítě a dynamicky se mění v závislosti na času, zátěži atp. Abychom mohli složit graf, je zapotřebí, aby mezi doménovými objekty byly spojovací body typu `ConnectivityNode`, nebo `AbstractConnectivityNode`. Spojovací prvky modelu jsou hranami grafu a zbylé doménové objekty jsou vrcholy tohoto grafu. Modely se dle složitosti dělí na lowLevel a highLevel. Tyto modely dále dělíme na plný a zjednodušený.

- `highLevel` – do tohoto modelu patří pouze sběrnice (Bus), elektrické vedení (PowerLine) a spojovací prvky (ConnectivityNode).
- `lowLevel` – prvky tohoto modelu jsou úplně všechny objekty modelu.
- zjednodušený model (`simple`) – v zjednodušeném modelu jsou pouze data potřebná k vykreslení daného modelu (grafu).
- plný model (`detail`) – součástí tohoto modelu jsou úplné informace o všech prvcích modelu včetně živých dat.

Detailnějšímu popisu API se věnují následující kapitoly [kapitola 4 – Stávající architektura] a [kapitola 5 – Vyvíjená architektura serveru].

HighLevel model

Příklad 3.1 ukazuje, jak vypadá struktura výsledku dotazu na highLevel. Každý prvek tohoto modelu má svoje připojené hrany `connectivities` (viz kapitola 3.1.1). Sběrnice (`Bus`) k tomu navíc mají popis, GPS souřadnice a úroveň napětí. Elektrické vedení (`PowerLine`) vlastní úroveň přenášeného napětí a popis.

Dotazem na zjednodušený model získáme statická data o modelu. V plném pohledu jsou kromě těchto statických dat i živá data.

```
{
  "buses": [{
      "connectivities": [{
          "idBaseObject": "string",
          "idConnectivity": "string",
          "idConnectivityNode": "string",
          "temporary": 0
        }],
      "descriptionVoltageLevel": "string",
      "gpsLatitude": 0,
      "gpsLongitude": 0,
      "idVoltageLevel": 0,
      "mRid": "string",
      "name": "string"
    }],
     "connectivityNodes": [{
      "connectivities": [{
          "idBaseObject": "string",
          "idConnectivity": "string",
          "idConnectivityNode": "string",
          "temporary": 0
        }],
      "gpsLatitude": 0,
      "gpsLongitude": 0,
      "mRid": "string",
      "name": "string"
    }],
  "powerLines": [{
      "connectivities": [{
          "idBaseObject": "string",
          "idConnectivity": "string",
          "idConnectivityNode": "string",
          "temporary": 0
        }],
      "descriptionVoltageLevel": "string",
      "gpsLatitude": 0,
      "gpsLongitude": 0,
      "idVoltageLevel": 0,
      "mRid": "string",
      "name": "string"
    }]
    }
```

Ukázka 3.1: HighLevel model – zjednodušený

LowLevel model

Příloha f zobrazuje strukturu výsledku dotazu pro lowLevel model. LowLevel model ukazuje všechny prvky modelu přenosové sítě. Na ukázce jsou vidět generátory (`GeneratingUnit`), zátěže (`Load`), přepínače (`Switch`) a elektrické vedení (`PowerLine`) a jejich vlastnosti. U tohoto detailnějšího pohledu opět platí to samé jako u highLevel modelu – dotaz na plný a zjednodušený model vrací vždy stejná data. Podrobnější data jsou vyžádána až při požadavku klienta.

Reprezentace modelu v paměti aplikace

Model je do paměti aplikace ukládán jako instance třídy Model:

```
public class Model {
@Id @GeneratedValue
private Long id;

private String name;

private String description;

UndirectedGraph<IdentifiedObject, DefaultEdge>
graph = new
   ListenableUndirectedGraph<>(DefaultEdge.class);
```

Ukázka 3.2: Definice třídy Model

Na ukázce 3.2 je vidět definice modelu, kde je vidět jak je graf ukládán do paměti. Využívá se prvků knihovny JGraphT – Rozhraní `UndirectedGraph` a implementace tohoto grafu `ListenableUndirectedGraph`. Třída `DefaultEdge` definuje hranu grafu. Tato třída obsahuje identifikátory vrcholů grafu.

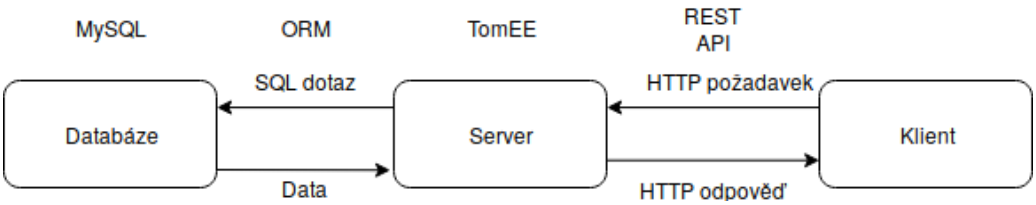
Algoritmus načítání modelu:

1. načtení spojovacích bodů z databáze.
2. načtení jednotlivých doménových objektů z databáze v závislosti na spojovacích bodech (spojovací body obsahují idnetifikátory doménových objektů) .
3. uložení doménových objektů a spojovacích bodů do grafu. Doménové objekty jsou vrcholy grafu a spojovací body jsou hranami tohoto grafu

Proces skládání grafu je vidět ve třídě `ModelCoposer` v metodě `compose`.

3.4 Výměna dat mezi serverem a databází

Na obrázku 3.3 je znázorněna výměna dat mezi serverem a databází z pohledu datového modelu. Samotnou výměnu dat mezi serverem a databází obstarává ORM knihovna OpenJPA komunikující jazykem SQL a komunikaci mezi serverem a klientem obstaráva rozhraní REST API, konkrétně knihovna JAX-RS. V následujících kapitolách se budu architekturou aplikace zabývat podrobněji.



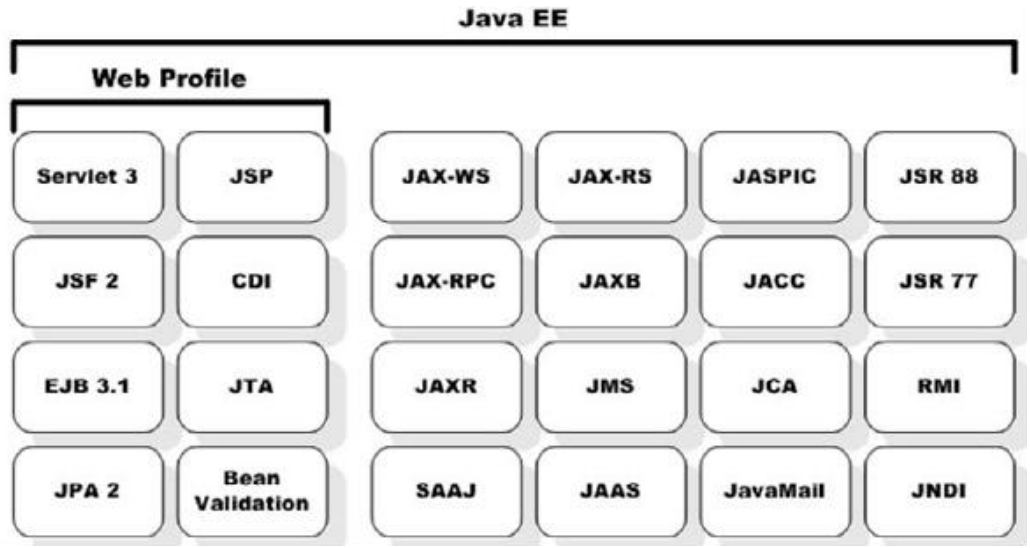
Obrázek 3.3: Výměna dat mezi serverem a databází

4 Stávající architektura serveru

4.1 Aplikační servery

Aplikačním serverem lze nazvat jakoukoliv knihovu, která splňuje specifikace **Java EE**[1] platformy zajišťující požadovanou funkcionalitu. Tyto knihovny implementují veškerá API definovaná ve specifikaci JavaEE. Kromě toho každý jednotlivý aplikační server poskytuje další funkce nad rámec specifikace.

Tato specifikace neimplementuje jednotlivé technologie, ale udává, jak mají fungovat. Do vývoje se zapojují nejen komerční subjekty, ale i open-source organizace a experti v oboru.



Obrázek 4.1: Technologie specifikace JavaEE

[1]Java Enterprise Edition. Jedná se o běžnou aplikaci, na kterou jsou kladeny určité nároky, co se týče spolehlivosti, dostupnosti, robustnosti a výkonnosti. Typická je také potřeba obsloužit současně velké množství požadavků a klientů. Specifikaci JavaEE navrhuje Java Community Process (www.jcp.org).

4.1.1 Komponenty

Na aplikační server se nasazují takzvané komponenty. Komponenty jsou základní jednotky, ze kterých je složena výsledná aplikce. Komponent existuje několik druhů, z nichž důležité pro server jsou pouze dva:

- Webové komponenty - servlety, JSP soubory a JSF soubory.
- Enterprise JavaBeans(EJB) komponenty - javovské třídy, které tvoří logiku aplikace a manipulují s daty (více informací o EJB viz [13]).

Současná aplikace vizualizace přenosové sítě využívá klientské části, která vznikla dříve. Nevyužívají se tedy webové komponenty, protože získaná data jsou posílána přes REST API klientovi a následně zpracována Javascriptovým frameworkem Angular. Oddělení těchto dvou částí je výhodné tím, že se zamezí závislosti serverové a klientské části.

4.1.2 Nejznámější aplikační servery

V této kapitole je porovnán výběr nejznámějších aplikačních serverů s ohledem na nároky aplikace vizualizace přenosové sítě. Na stránce 29 naleznete přehled 4.2 aplikačních serverů pro jazyk Java.

Apache Tomcat®

Apache Tomcat [17] je server vyvíjený pod svobodnou licencí Apache License 2.0, který implementuje pouze malé množství technologií specifikace JavaEE. Hodí se tak pro menší projekty. Server se dá rozšířit o potřebné knihovny, ale poté si musí tvůrce sám ohlídat jejich přidání, kompatibilitu, aktuálnost a stabilitu. S přidáváním knihoven roste jeho velikost. To, že tvůrce si sám může určit, jaké technologie chce využít, je pak velkou výhodou tohoto serveru. Nejnovější verzí je nyní Tomcat 9.0.8[2]. Očekává se, že budou i nadále vznikat nové verze tohoto serveru.

TomEE

TomEE [16] je server vyvíjený pod svobodnou licencí Apache License 2.0. Implementuje server Tomcat a k tomu několik dalších rozšiřujících technologií. Jmenovitě JSP, EJB, CDI, JPA, JTA, JSF, JMS, nástroje pro práci s databázemi a také frameworky pro komunikaci na webu.

[2] 3.května 2018

Nejnovější verzí je 7.0.4[3] postavená na Tomcat 8.5. Vzhledem k provázanosti obou zmíněných serverů se dá očekávat, že budou i nadále vznikat nové verze a bude zaručena podpora.

WildFly

Tento aplikační server [6] je open source komunitním projektem pod záštitou společnosti Red Hat. Vznikl otevřením původního projektu JBoss AS komunitě. Společnost dále nabízí JBoss EAP, který je určen pro komerční užití. Wildfly implementuje plnou specifikaci Javy EE ve verzi 7 a je specifický tím, že implementuje agresivní správu paměti – to má za následek nejenom menší paměťové nároky, ale také menší odezvu serveru snížením frekvence a rozsahu práce garbage kolektoru.

Vzhledem k tomu, že se jedná o komunitní projekt, je zde velká snaha používat nové a oblíbené technologie (např. Hibernate, Narayana, RESTEasy). Server je možné konfigurovat úpravou konfiguračního souboru, administrátorskou konzolí, nativní Java API, REST API založené na HTTP/JSON a JMX bránou.

Glassfish

Aplikační server Glassfish [11] od společnosti Oracle je open source projekt, který plně implementuje specifikaci Java EE 6. Velkou předností tohoto serveru je, že pracuje s frameworkem OSGi[4].

Další výhodou tohoto serveru je pak kompatibilita s vývojovými prostředími Eclipse a NetBeans. Server také nabízí grafické i konzolové uživatelské rozhraní a podporu technologií jako Maven, Ant, REST API a další.

Vzhledem k tomu, že poslední aktualizace tohoto serveru je ze srpna roku 2017 a také kvůli nedostupnosti webové stránky http://glassfish.org se dá předpokládat, že tento server již není dále vyvíjen a nemá plnou podporu.

Jetty

Eclipse Jetty [4] nabízí web server, javax.servlet kontejner a podporu technologií (a knihoven) jako HTTP/2, WebSocket, OSGi[4], JMX, JNDI, JAAS. Všechny tyto vyjmenované komponenty i celý server jsou open source a využitelné dokonce i pro komerční účely. Záštitu tomuto server poskytuje Eclipse

[3] 26. srpna 2017

[4] Technologie OSGi umožňuje zavádět části kódu za plného běhu serveru. To znamená, že při publikování nového kódu na server není nutný restart serveru. Více informací o OSGi framworku na webové adrese https://www.osgi.org/developer/what-is-osgi/.

Foundation od roku 2009, je tedy zajištěna dobrá podpora uvnitř vývojového prostředí Eclipse.

4.1.3 Zvolený aplikační server

Pro aplikaci „Vizualizace přenosové sítě“ je použit aplikační server TomEE, protože již implementuje většinu potřebných technologií a ty nemusí být distribuovány společně s aplikací. Další výhodou je fakt, že tento server je primárně určen pro JavaEE. Při rozhodování byl plusem i fakt, že TomEE přímo vychází ze serveru Tomcat, který byl použit v předchozích verzích vyvíjené aplikace.

Servery Glassfish a Jetty nejsou pro vyvíjenou aplikaci vhodné zejména kvůli velmi vysoké složitosti v porovnání se servery Tomcat, TomEE či Wildfly. Je to dáno zejména tím, že tyto servery využívají framework OSGi[2].

Product	Vendor	Edition	Last release	Java EE compatibility [4]	Servlet	JSP	HTTP/2	License
ColdFusion	Adobe Systems	2016.0.1	2016-05-01	7 partial platform	3.1	2.3	No	Proprietary, commercial
Enhydra	Lutris	5.1.9	2005-03-23	No			No	Free, GPL
Enterprise Server	Borland	6.7	2007-01	1.4	2.4	2.0	No	Proprietary, commercial
Geronimo	ASF	3.0.1	2013-05-28	6 full platform	3.0	2.2	No	Free, Apache
GlassFish	GlassFish Community	5.0.0	2017-09-21	8 full platform	4.0	2.3	No	Free, CDDL, GPL + classpath exception
iPlanet Web Server	Oracle Corporation	7.0.21	2015-04	Yes[5]	2.5	2.1	No	Proprietary, commercial
JBoss Enterprise Application Platform	Red Hat	7.0.0	2016-05	7 full platform	3.1	2.3	Yes	Free, LGPL
Jetty	Eclipse Foundation	9.3.3	2015-08-27	7 partial platform[6]	3.1	2.3	Yes	Free, Apache 2.0, EPL
JEUS	TmaxSoft	8	2013-08	7 full platform	3.0	2.2	No	Proprietary, commercial
JOnAS	OW2 Consortium (formerly ObjectWeb)	5.3	2013-10-04	6 Web Profile	3.0	2.2	No	Free, LGPL
JRun	Adobe Systems	4 updater 7	2007-11-06	1.3	3.1	2.3	No	Proprietary, commercial
Lucee (Formerly Railo)	Lucee Association Switzerland	5.2.5.25	2017-12-22	7 partial platform	3.1	2.3	No	Free, CDDL, GPL + classpath exception
NetWeaver Application Server	SAP AG	7.4	2013-01-11	5	2.5	2.1	No	Proprietary, commercial
Oracle Containers for J2EE	Oracle Corporation	10.1.3.5.0	2009-08	1.4	2.4	2.0	No	Proprietary, commercial
Orion Application Server	IronFlare	2.0.7	2006-03-09	1.3	2.3	1.2	No	Proprietary, commercial
Payara	Payara	4.1.2.181	2018-02-12	7 full platform	3.1	2.3	No	Free, CDDL, GPL + classpath exception
Resin Servlet Container (open source)	Caucho Technology	4.0.36	2013-04-25	6 Web Profile[7]	3.0	2.2	No	Free, GPL
Resin Professional Application Server	Caucho Technology	4.0.36	2013-04-25	6 Web Profile	3.0	2.2	No	Proprietary, commercial
Rupy	Rupy	1.2	2015-01-01	No			No	Free, LGPL
Tomcat	ASF	9.0.8	2018-05-03	8 partial platform	4.0	2.3	Yes	Free, Apache v2
TomEE	ASF	1.7.4	2016-03	6 Web Profile	3.0	2.2	No	Free, Apache
WebLogic Server	Oracle Corporation (formerly BEA Systems)	12.2.1.1	2016-06-21 [8]	7 full platform	3.1	2.3	No	Proprietary, commercial
WebObjects	Apple Inc.	5.4.3	2008-09-15	Partial[9]			No	Proprietary, commercial
IBM WebSphere Application Server	IBM	9.0	2016-06-24	6 & 7 full platform	3.1	2.3	No	Proprietary, commercial
WebSphere AS Community Edition	IBM	3.0.0.4	2013-06-21	6 full platform	3.0	2.2	No	Proprietary, commercial
WildFly (formerly JBoss AS)	Red Hat (formerly JBoss)	12.0.0.Final	2018-02-28	7 full platform & 8 preview mode	4.0	2.3	Yes	Free, LGPL

Obrázek 4.2: Tabulka aplikačních serverů k 11. červnu 2018

4.2 Architektura

Aplikace je rozdělena na dvě základní části – klientskou a serverovou. Klientská část je oddělena od té serverové. To má za následek fakt, že obě dvě části mohou být nezávisle na sobě měněny (pokud bude dodrženo stejné rozhraní pro vstup a výstup). Výměna dat mezi nimi probíhá pomocí REST API (viz obr. 3.3).

REST API je architektura rozhraní, navržená pro distribuované prostředí. Webové služby definují vzdálené procedury a protokol pro jejich volání. REST API určuje, jak se přistupuje k datům. V případě aplikace „Vizualizace přenosové sítě“ toto rozhraní určuje URL adresy, na kterých jsou definovány serverové služby.

Tato kapitola čerpá informace uvedené v bakalářské práci [21].

4.2.1 Uživatelské rozhraní

V projektu již existuje grafické uživatelské rozhraní, které je vytvořeno v programovacím jazyce JavaScript a využívá knihovny Angular. Aktuální verze umožňuje vizualizaci vybraného modelu na mapě generované pomocí knihovny OpenLayers. V rámci bakalářské práce [21] byla funkcionalita uživatelského rozhraní rozšířena o správu uživatelů. Je tedy možné přiřazovat modely jednotlivým uživatelům a uložit novější verzi těchto modelů. U vizualizace v budoucnu přibude možnost přidávat, odebírat, či dočasně vypnout jednotlivé prvky sítě.

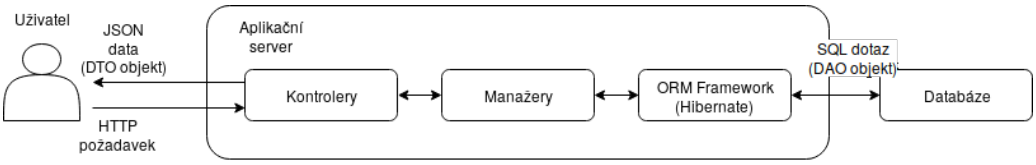
4.2.2 Server

Stávající verze serverové části aplikace je postavena na aplikačním serveru Tomcat a vytvořena v programovacím jazyce Java s využitím frameworku Spring. Data jsou uložena v SŘBD MySQL a výpočty jsou prováděny v knihovně Matlab. Aplikační server je rozdělen na několik vrstev (viz obr. 4.3)

- Kontrolery – Vrchní vrstva serveru, která naslouchá přístupům do aplikace na specifikovaných url adresách. Výstupem kontrolerů jsou data strukturovaná ve formátu JSON. Aby data mohla být strukturována do formátu JSON, je potřeba převést tato data na takzvané DTO objekty.

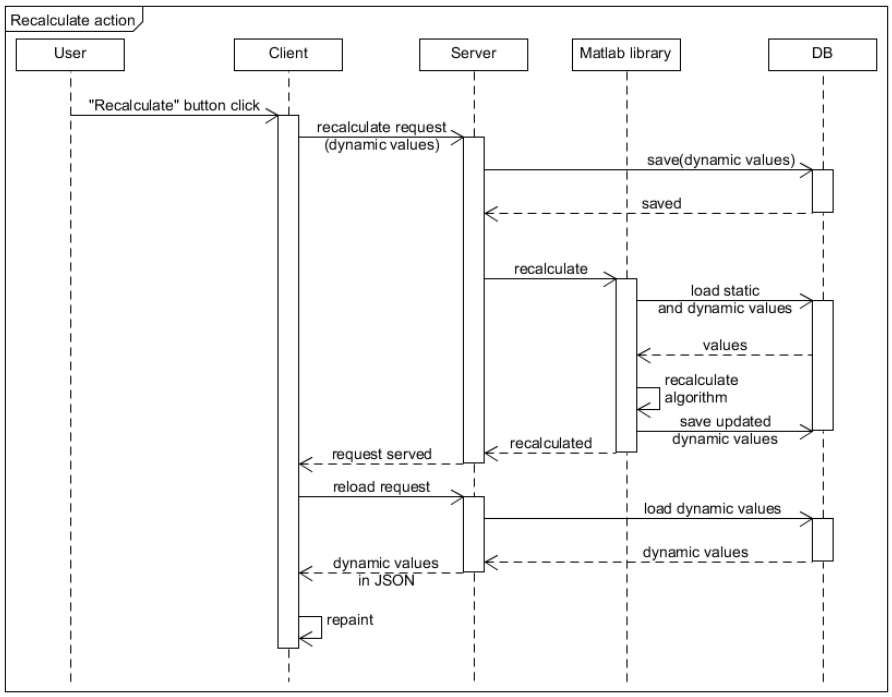
- Manažery – V manažerech probíhá zpracování získaných dat do formátu, se kterým se dá dále pracovat. To znamená, že zde probíhá konverze mezi DAO a DTO objekty.

- ORM framework (Hibernate) – Tato vrstva má za úkol správu databázové vrstvy. Pro komunikaci mezi manažery a databází jsou používány takzvané DAO objekty, které obsahují metody skládající konečné SQL dotazy do databáze. Hibernate je konkrétní knihovna, která implementuje specifikaci JPA.



Obrázek 4.3: Komunikace klient-server

Pokud dojde k upravení vlastností prvků sítě, nastává přepočítání modelu v knihovně Matlab. Celý tento proces přepočítávání je ilustrován na obrázku 4.4.



Obrázek 4.4: Přepočítání modelu

Po stisku tlačítka přepočítání jsou pozměněná data odeslána na server a uložena do databáze. Požadavek přepočítání je poté zaslán Matlab knihovně, která z aktuálních dat načtených z databáze vytvoří model a soustavu rovnic. Nad těmito rovnicemi provede výpočty. Následuje uložení nově získaných dat zpět do databáze a je předán požadavek serveru, aby si načetl nová data a vrátil je klientovi.

4.3 Popis API

API stávající architektury je velmi kvalitně dokumentováno v bakalářské práci Lukáše Černého [21]. Podrobnější informace o API jsou uvedené v přiloženém souboru s názvem *dokumentace_api.yaml*, který lze zobrazit ve Swagger nebo Apiary editoru. V souboru je detailně popsán každý dotaz na server včetně parametrů a jednotlivých atributů.

5 Navržené řešení

Cílem mojí práce bylo navrhnout a implementovat datový model přenosové soustavy vhodný pro Java server. Přibližná struktura datového modelu byla dána ERA diagramem **??**, avšak požadavky se během vývoje měnily do stávající podoby. Práce navazuje na prototyp serveru z první fáze vývoje. V první fázi vývoje server využíval několik knihoven, které zvětšovaly velikost aplikace. Nová verze serveru vychází z původní verze pouze ideově – je navržena od základu a využívá jiné knihovny a technologie.

Aplikace využívá nástroj **Maven**, který je určen ke správě a konfiguraci projektů napsaných převážně v programovacím jazyce Java. Konfigurace nástroje se nachází v souboru *pom.xml*. V tomto souboru jsou uvedeny veškeré informace potřebné k přeložení nástrojem Maven a spuštění na aplikačním serveru. Jedná se o název, verzi, profily, závislosti na přidaných knihovnách a jiné. Jsou definovány dva profily – **development**, který je určen pro interní vývoj aplikace a **production**, který je nasazován na reálný server.

5.1 Server a databáze

Aplikace využívá server TomEE – tento server obsahuje většinu potřebných knihoven. Výsledkem je menší aplikace, která využívá prostředky serveru. Aplikaci lze nasadit i na jiný server, je však nutné zajistit, aby daný server obsahoval všechny potřebné knihovny.

TomEE obsahuje technologie JPA pro komunikaci s databází – konkrétně se jedná o implementaci **OpenJPA**. Řízení této komunikace je plně pod správou serveru. K nastavení zdrojů této komunikace slouží soubor *tomee.xml* obsahující jednotlivé datové zdroje.

Nastavení technologie JPA se nachází v souboru *persistence.xml*. Soubor obsahuje nastavení perzistence dat pro jednotlivé datové zdroje. Jsou zde specifikovány perzistenční jednotky a jejich vlastnosti. Pro jednotlivé perzistenční jednotky jsou zde definovány i třídy aplikace, které budou ukládány do databáze.

V průběhu vývoje jsme v aplikaci přešli v rámci specifikace JPA z implementace **Hibernate** na implementaci **OpenJPA**. Jednotkové testy využívají právě Hibernate. Používání hibernate pro jednotkové testy umožňuje spouštění testů bez nasazení na server. Soubor *hibernate.cfg.xml* obsahuje konfiguraci této knihovny – nastavení konekce do databáze spolu s výpisem tříd, které budou ukládány do databáze.

Jako databázový systém využíváme v aplikaci systém MySQL. Požadavek na tento systém pochází od zadavatelů projektu. Aplikace využívá tři databáze:

- **users** – obsahuje informace o uživatelích.
- **models** – v této databázi jsou uloženy informace o modelech. Jedná se o topologii přenosové sítě, identifikátory uživatelů s právy a obecné informace jako identifikátor modelu, či název modelu.
- **domains** – obsahuje jednotlivé doménové objekty.

Účelem rozdělení na tři databáze je nezávislost dat. Doménové objekty mohou být prvky několika modelů zároveň. Oprávnění přístupu k modelům není kvůli rozdělení pevně svázáno s uživateli – je možné uživatele měnit, či je obnovit z uživatelské databáze při chybě.

Server TomEE neobsahuje potřebnou knihovnu pro komunikaci s databázovým systémem MySQL. Je tedy nutné tuto knihovnu ručně přidat do serveru. Jedná se o knihovnu *mysql-connector-java-5.1.6.jar* a je nutné ji přidat do adresáře:
`<cesta k serveru>/lib/mysql-connector-java-5.1.6.jar`

5.2 Popis API

Služby poskytované přes vyvíjené API vychází z původního návrhu a dále ho rozšiřují. Cesty byly oproti původnímu návrhu poněkud pozměněny, ale struktura dat zůstala zachována. Ve vyvíjené verzi přibyla správa doménových objektů a správa uživatelů. Práce s modelem je totožná s původním návrhem – jen pro modely byla přidána potřebná autorizace v hlavičkách požadavků. Aplikace využívá knihovnu Jonhnzon, která je implementací technologie JsonProcessing[1] (JSR-353). K této implementaci přidává rozšíření o ObjectMapper a nabízí podporu JAX-RS[2] na aplikačním serveru. Tato podpora vychází z použití technologie JSR 356 (JavaTM API for WebSocket[12]).

Knihovna Apache CXF zajišťuje implementaci výše zmíněné technologie JAX-RS. Obě tyto knihovny dohromady zajišťují spuštění obslužné metody na specifikované URL adrese.

[1] Tato technologie umožňuje převod objektového návrhu do JSON formátu a naopak. Více o technologii JSR-353 zde: https://jcp.org/en/jsr/detail?id=353

[2] „Java API for RESTful“ – webová služba, která umožňuje použití REST (REpresentational State Transfer)

Ve třídě CORSFilter jsou definovány povolené typy dotazů a metody přístupu. Na vyvíjenou aplikaci je možné posílat dotazy typu GET a POST. Metody přístupu jsou podle autorizace:

- neautorizovaný – neautorizovaný přístup je povolen pouze pro přihlášení do aplikace (autentizace).
- autorizovaný – veškeré ostatní požadavky vyžadují autorizaci.

Aplikace byla rozšířena o následující funkcionalitu:

Chybové stavy odpovědi na požadavek

Odpovědí na dotaz na adresu aplikace může být i chybový stav – může se stát, že uživatel nebyl autorizován, v databázi nejsou data apod. Proto byly definovány číselné odpovědi na tyto požadavky. Tyto odpovědi vychází z HTTP status kódů [9].

V naší aplikaci používáme tyto kódy:

- **200 OK** – Kód je vracen, pokud požadavek proběhne v pořádku
- **400 Bad Request** – Pokud uživatel zadá požadavek na cestu, která není definována, je mu navrácen tento kód
- **403 Forbidden** – Uživatele se nepodařilo autorizovat, nebo nemá přístup
- **405 Method Not Allowed** – Vrací se, pokud je podán špatný typ dotazu (GET, POST, DELETE atd.)
- **415 Unsupported Media Type** – Indikuje chybu ve formátu JSON v požadavku
- **500 Internal Server Error** – Obecná chyba aplikace

5.2.1 Správa doménových objektů

Do stávajícího API byla přidána možnost správy doménových objektů. Všechny tyto přístupy vyžadují autorizaci, a tedy roli administrátora.

Relativní adresy z následující ukázky jsou kořeny cest, na kterých jsou definovány funkce pro práci s doménovými objekty.

`.../pnp/api/bus/`
`.../pnp/api/connectivity_node/`
`.../pnp/api/generator/`
`.../pnp/api/load/`
`.../pnp/api/power_line/`
`.../pnp/api/switch/`
`.../pnp/api/transformer/`
`.../pnp/api/winding/`

Jedná se o tyto funkce:

- `all()` – Vratí informace o všech doménové objektech daného typu z databáze.
- delete(long) – Vymaže příslušný doménový prvek z databáze. Parametrem funkce je identifikátor doménového objektu.
- `info(long)` – Vrátí informace o příslušném doménovém prvku. Parametrem funkce je identifikátor doménového objektu.
- `insert(<objekt>DTO)` – Vloží doménový objekt do databáze. Parametrem funkce je DTO objekt.
- `update(<objekt>DTO)` – Aktualizuje doménový objekt v databázi. Parametrem funkce je DTO objekt.

DTO (Data Transfer Object) objektem z pohledu uživatele je daný doménový objekt v JSON formátu. Konverze DTO objektů na JSON formát a zpět probíhá v kontrolerech pomocí technologie **JsonProcessing** knihovny **Johnzon**.

Na ukázce 5.1 je vidět požadavek na informace o sběrnici. V URL adrese je vidět parametr busID, který je identifkátorem sběrnice. Dotaz vrací informace o sběrnici v JSON formátu. K zobrazení je nutná autorizace uživatele – je nutné udat autorizační token a Session ID v hlavičce. Také je nutné udat Content-Type. Pro zobrazení všech sběrnic, použijeme dotaz `GET: http://localhost:8080/pnp/api/bus/all` se stejnou hlavičkou.

```
GET: .../pnp/api/bus/info?busID=1494

Headers:
X-auth-Token: 1715ea37e19b41130cbdba2aafa5f4e0c41ee0fc......
SID: riigl7epflojjskfbvevn2ff4t
Content-Type: application/json

Response:
        [{
        "originalID": "BIE_EL_PRIPOJNICE.261573",
        "location": {
            "latitude": 49.56278666404082,
            "longitude": 13.26212843016418
        },
        "voltageLevel": {
            "voltage": 22000,
            "name": "22 kV",
            "id": 5
        },
        "json": "{"Loaded":
                {
                        "Ground resistance [Ohm]":"[NaN]",
                        "Ground reactance [Ohm]": "[NaN]"
                }",
        "busType": 1,
        "id": 1494
        }]
```

Ukázka 5.1: Informace o sběrnici

Ukázka 5.2 je příklad toho, jak se dá pomocí API vložit nová sběrnice do databáze. Parametrem funkce je DTO objekt v JSON formátu. Tento parametr se vkládá do těla požadavku. Opět je nutné zadat informace o přihlášení. Dotaz na funkci `update(<objekt>DTO)` je totožný s tím, který se používá při `insert(<objektDTO>)`.

```
POST: .../pnp/api/bus/insert

Headers:
X-Auth-Token: 1715ea37e19b41130cbdba2aafa5f4e0c41ee0fc......
SID: riigl7epflojjskfbvevn2ff4t
Content-Type: application/json

Body:
{
    "originalID": "create",
    "name": "bus z api, create",
    "aliasName": "bus",
    "description": "update",
    "normallyInService": true,
    "aggregate": true,
    "baseVoltage": 10,
    "voltageLevel": {
        "voltage": 10,
        "name": "napeti",
        "id": 5
    },
     "json": "{"Loaded":
                {"Ground resistance [Ohm]":"[NaN]",
                 "Ground reactance [Ohm]": "[NaN]"}",
    "busType": 1,
    "busStatus": 0,
    "id": 4
}

Response:
{
       200, OK
}
```

Ukázka 5.2: Vložení nové sběrnice

Dotaz 5.3 ukazuje, jak lze vymazat doménový objekt z databáze. Parametrem je identifikátor doménového objektu. Odpovědí serveru je hlášení o úspěšnoti požadavku.

```
GET: .../pnp/api/bus/delete?busID=2

Headers:
X-Auth-Token: 1715ea37e19b41130cbdba2aafa5f4e0c41ee0fc......
SID: riigl7epflojjskfbvevn2ff4t
Content-Type: application/json

Response:
{
        200, OK
}
```

Ukázka 5.3: Vymazání sběrnice ID=2

Následující ukázka 7.5 je zde uvedena pro ilustraci, jak se z doménových objektů skládají modely. Navrácenými hodnotami jsou všechny spojové body. Každý spojový bod se skládá z identifikátorů doménových objektů. Pokud tyto spojové body přiřadíme modelu, je možné nadefinovat model.

```
GET: .../pnp/api/connectivity_node/all

Headers:
X-Auth-Token: 1715ea37e19b41130cbdba2aafa5f4e0c41ee0fc......
SID: riigl7epflojjskfbvevn2ff4t
Content-Type: application/json

Response:
[{
        "src": 1498,
        "trg": 1494
    },
    {
        "src": 1501,
        "trg": 1495
    }
}]
```

Ukázka 5.4: Informace o spojových bodech

5.2.2 Správa uživatelů

V rámci bakalářské práce Lukáše Černého vznikla implementace správy uživatelů. Informace o uživatelích jsou ukládány do databází **users** a **models** současně. Tento návrh byl zvolen kvůli jednoduššímu načítání modelů, které náleží uživateli. K uživatelům lze přistupovat obdobně jako k doménovým objektům, k tomu slouží adresy:

```
GET: .../pnp/api/user/all
GET: .../pnp/api/user/current
GET: .../pnp/api/user/delete?userId
GET: .../pnp/api/user/info?userId
```

Parametr **userId** je identifikátor konkrétního uživatele. Autorizace je nutná.

```
POST: .../pnp/api/user/insert
POST: .../pnp/api/user/update
```

Funkce `insert` a `update` mají parametr **userDTO** – objekt uživatele v JSON formátu, který je posílán v hlavičce dotazu. Autorizace je nutná.

```
POST: .../pnp/api/auth/apitoken/login
POST: .../pnp/api/auth/apitoken/logout
```

K autentizaci (a následné autorizaci přístupu k datům) je používána funkce `login()`. Ta je definována jako bezparametrická, ale je nutné v hlavičce dotazu uvést **Base-auth** token. Tento token je zakódován v Base64 a má strukturu username:hash (heslo zakódované pomocí SHA256). Při úspěšném přihlášení je uživatel uložen mezi aktivní uživatele a je mu zaslán přihlašovací token (**X-Auth-Token**). Opětovnému přihlášení ze stejného sezení je zamezováno pomocí **SSID**, což je jednoznačný identifikátor sezení.

Funkce `logout()` potřebuje udat v hlavičce přihlašovací token(X-Auth-Token) a identifikátor sezení(SSID), aby mohl být uživatel odhlášen.

Autorizace je proces ověření oprávnění uživatele uvnitř aplikačního serveru. Pro přístup k datům aplikace je nutné mít definovanou roli. Role jsou definovány dvě - ADMIN a CUSTOMER. Uživatel s administrátorskou rolí má přístup ke všem funkcím. Uživatel se zákaznickou rolí má přístup jen ke specifikované množině modelů.

Poslední funkcí správy uživatelů je funkce na adrese:

```
POST .../pnp/api/user/changeModelPermission?modelID
```

Funkce `changeModelPermission` je na rozmezí mezi prací s uživateli a s modelem. Tato funkce změní model daný parametrem **modelID**. Cílem je změnit přístup uživatelů k modelu. Parametr **modelID** je indetifikátor modelu

a je uváděn v adrese, zatímco druhý parametr **GraphModelDTO** je daný model v JSON formátu. Autorizace je nutná.

Podrobnější informace o správě uživatelů jsou dostupné zde [21, Kapitola 4.2, str. 33-38]

5.3 Struktura aplikace

Tato kapitola stručně popisuje strukturu kódu aplikace

5.3.1 Struktura balíků

Aplikace je rozdělena do jednotlivých balíků. Tyto balíky vždy začínají názvem **cz.zcu.laps.pnp**. Jedná se o tyto balíky:

.converters

Obsahuje třídu **ModelConverter**, která konfiguruje knihovnu **ModelMapper**. Tato třída umožňuje konverzi z doménového objektu na DTO objekt (a naopak). Tento převod je prováděn přes reflexi. Uvnitř třídy jsou také výjimky – například číselníky jsou v doménových objektech reprezentovány enumerátory, zatímco v DTO objektech jsou reprezentovány číslovkou.

.core

Balík .core obsahuje základní konfiguraci celé aplikace. Třída **ApplicationConfig** je inicializační třída – načítá konfiguraci ze souboru config.properties, vytváří základního uživatele a nastavuje lokalizaci. Třída **JacksonConfig** nastavuje to, jak bude probíhat konverze doménových objektů do JSON formátu (přes konvertory). Tato konverze využívá knihovnu Jackson. Třída Messages načítá lokalizační řetězce podle nastavené lokalizace (locale).

.domain

V balíku domain jsou veškeré doménové objekty aplikace. Tyto doménové objekty, pokud mají být ukládány do databáze, obsahují anotace i knihovny *javax.persistence*. Tyto anotace určují, jak bude implementace technologie **JPA** (OpenJPA či Hibernate) ukládat doménové objekty do databáze. Také jsou zde takzvané „NamedQueries“, což jsou řetězce v Java Persistence query language, které jsou používány pro načítání dat z databáze. Pokud je pro danou implementaci JPA složité automaticky ukládat či načítat data do databáze, je možné použít tyto „NamedQueries“.

Do tohoto balíku jsem dále přidal další balíky tříd, které pracují s doménovými objekty:

- *controllers.api* – zde jsou definována rozhraní pro jednotlivé kontrolery. V těchto rozhraních jsou definovány cesty funkcí dostupné pomocí JAX-RS technologie.
- *controllers.core* – obsahuje implementace rozhraní kontrolerů.
- *managers.api* – balík managers.api obsahuje rozhraní manažerů.
- *managers.core* – v balíku jsou implementace jednotlivých manažerů.

.dto

Balík obsahuje třídy, které vycházejí z doménových objektů. Tyto třídy jsou konvertovány do formátu JSON. Je zde tedy možnost některé atributy skrýt, či naopak některé přidat.

.exceptions

Tento balík obsahuje vlastní aplikační výjimky. Vlastní vyjímky jsou využívány pro přehlednost kódu, ulehčení ladění aplikace a lepší komunikaci s uživatelem.

.idm

„Identify model“ – balík obsahuje logiku spojenou s autorizací a autentizací uživatele. Součástí jsou anotace autorizace, výjimky, interceptory, doménové objekty a nástroje pro přehlednost kódu.

.loader

V tomto balíku je logika načítání doménových objektů z textových souborů a jiných databází. Podporováno je načítání:

- textových .xml souborů
- reálných dat ze vzdálené databáze Oracle

Výsledný model přenosové sítě je z načítaných dat skládán uvnitř třídy **ModelComposer**.

.managers

Balík .managers obsahuje rozhraní a implementace manažerů uživatelů a modelů, které slouží k perzistenci objektů s databází.

.repositories

V tomto balíku jsou rozhraní a implementace tříd, které přímo komunikují s databází. Pro každou databázi (domains, users, models) jsou zde rozhraní přístupů do databáze a implementace jednotlivých tříd, které vyměňují data s databází. K tomu slouží balíky *javax.persistence.** které jsou součástí specifikace JavaEE. Samotný přenos dat pak je realizován pomocí technologie JPA.

.utils

Balík určený pro pomocné třídy a opakující se části kódu.

.ws

Třídy obsluhující webové služby práce s modely a uživateli. Tento balík obsahuje další podbalíky anotací, kontrolerů, pomocných enumů a filtrů serverových požadavků.

5.4 Perzistence dat

V předchozích kapitolách bylo rozebráno které technologie a knihovny jsou použity pro výměnu dat mezi databází a aplikací. Tato kapitola rozebírá výměnu dat podrobněji.

5.4.1 ORM

Aplikace využívá standart JPA, který umožňuje Objektově Relační Mapování (ORM). ORM umožňuje mapování Java objektů přímo do databáze, které je realizováno pomocí implmentací standartu JPA. V naší aplikace je využíván Hibernate (pro lokální testy) a OpenJPA. Knihovna OpenJPA je používána, protože ji obsahuje aplikační server TomEE. Nastavení knihovny je v souboru *tomee.xml*, který musí být přidán do nastavení aplikačního serveru. Hibernate je používán pro lokální testy – není nutné připojení na server. Konfigurace této knihovny je v souboru *hibernate.cfg.xml*.

Technologie JPA odstraňuje pro programátory nutnost správy databáze a umožňuje převod Java objektů (a jejich atributů) do databáze a to nejlépe

z primitivních datových typů na podobné typy databáze. Tento převod může být problematický, pokud není možné převést datové typy. Pro práci s ORM obsahuje JavaEE balík `javax.persistence.*`. V práci jsou používány převážně anotace z tohoto balíku. Technologie JPA podporuje také dědičnost objektů, kterou relační DB neumí. Volba strategie zpracování dědičnosti vyžaduje porozumění reprezentace dat v DB.

Existuje několik strategií jak pracovat s dědičností:

- **Mapped Superclass** – Třída označená jako mapovaná supertřída **není** ukládána do databáze. Atributy supertřídy jsou přidány entitám, které od dané supertřídy dědí, a jsou uloženy společně s entitami v jejich záznamu.
- **Table per Class** – Všechny třídy jsou v rámci této strategie označeny jako entity a jsou uloženy do databáze se svými atributy. Atributy jsou uloženy vždy k dané tabulce a jsou definovány vztahy mezi jednotlivými tabulkami. Zápis této strategie je jednoduchý, ale knihovna musí při požadavku na data řešit vztahy mezi tabulkami a vytvářet složité dotazy.
- **Single Table** – Třídy (entity) jsou namapovány do jedné velké tabulky. Tento přístup vykazuje nejlepší výkon při polymorfizmu. Při ukládání třídy jsou uloženy jen dané atributy a ostatní sloupce tabulky zůstavají null. Je také nutné pro každou entitu uvést podle čeho jí knihovna pozná (Diskriminátor). Může vést k neintegritě záznamů.
- **Joined** – Tato strategie sdružuje třidy do větších celků, které jsou poté uloženy do databáze. Nevznikají tedy tak velké tabulky jako při strategii „Table per Class“. Atributy všech sdružených tříd jsou uloženy v tabulce, ale díky menšímu počtu atributů je výsledná tabulka přehlednější než při strategii „Single Table“.

5.4.2 Zvolená strategie

V aplikaci je využívána kombinace dvou strategií. Vzhledem k tomu, že všechny doménové objekty dědí od třídy `IdentifiedObject`, je nasnadě, aby alespoň pro tuto třídu byla využita strategie mapované supertřídy. Všechny objekty, které dědí od třídy `IdentifiedObject` získávají její atributy.

Pro jednotlivé doménové objekty se používá „Joined“ strategie. Doménové objekty postupně dědí od předků atributy až ve výsledku jsou do databáze ukládány tyto třídy:

- `AbstractConnectivityNode (abstraktní spojový bod)`
- `Bus (sběrnice)`
- `ConnectivityNode (spojový bod)`
- `GeneratingUnit (generátor)`
- `Load (zátěž)`
- `PowerLine (elektrické vedení)`
- `PowerTransformer (tranformátor)`
- `Switch (přepínač)`
- `TariffLoad (tarif pro zátěž)`
- `TransformerWinding (cívka transformátoru)`
- `Variable (definice živých hodnot)`
- `VoltageLevel (úroveň napětí)`

Použitím „Joined“ strategie vznikají větší tabulky, které ale zůstávají stále přehledné (a to i samostatně v databázi).

5.4.3 Mapování transformátoru

Tato kapitola popisuje strukturu doménových objektů (a ORM anotací) s ukázkami kódu.

```
@MappedSuperclass
public abstract class IdentifiedObject implements
   LiveValueSettable{
        @Id
        @Column(name = "longID")
        protected Long ID;
```

Ukázka 5.5: Definice mapované supertřídy `IdentifiedObject`

Ukázka 5.5 ukazuje definici předka celého stromu doménových objektů. Tato třída má anotací `@MappedSuperclass` pro nastavení mapované supertřídy. Na ukázce je taky vidět definování primárního klíče (identifikátoru – longID) všech doménových objektů – k tomu slouží anotace `@Id`. Atributy, které mají být uloženy do databáze jsou anotovány `@Column(name = <jméno sloupce>)`.

```
@MappedSuperclass
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class PowerSystemResource extends
   IdentifiedObject {

        @Embedded
        protected GPSLocation location;
```

Ukázka 5.6: Definice třídy `PowerSystemResource`, která přidává entitám GPS souřadnice

Na ukázce 5.6 je vidět, že mapovaná supertřída nemusí být pouze jedna. Třída `PowerSystemResource` přidávám všech potomkům GPS souřadnice. Anotace `@Embedded` nastavuje atribut jako vnořený – že bude vložen do tabulek konečných tříd. `@Inheritance(strategy = InheritanceType.JOINED` definuje strategii práce s dědičností. Na ukázce je vidět použití strategie „Joined“ spolu s „MappedSuperclass“.

```
@MappedSuperclass
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Equipment extends
   PowerSystemResource {

        @Column
        protected Boolean normalIyInService;

        @Column
        protected Boolean aggregate;

        @JoinColumn
        private OperationalStatus operationalStatus;
```

Ukázka 5.7: Definice třídy `Equipment`, která udává zda-li je zařízení v provozu, nebo není

Příklad 5.7 ukazuje další anotaci, kterou je `@JoinColumn`. Tato anotace definuje vztah mezi třídou a atributem (další entita). Pokud je anotace bez parametrů, předpokládá se, že se odkazujeme na třídu s primárním klíčem a tento klíč bude použit jako hodnota ve sloupci tabulky.

```
@Inheritance(strategy = InheritanceType.JOINED)
@Entity
@NamedQuery(name="findAllPowerTransformer",
   query="SELECT transformer FROM PowerTransformer
   transformer")
public class PowerTransformer extends Equipment {

 @OneToMany(mappedBy = "powerTransformer",
    cascade=CascadeType.ALL, fetch=FetchType.EAGER)
```

Ukázka 5.8: Definice transformátoru, který může mít několik vinutí

Předposlední ukázkou 5.8 je třída definující transformátor. Na ukázce je vidět definice „NamedQuery“ – řetězec v jazyce Java Persistence query language. Takto lze definovat jaký má být dotaz do databáze pro práci s prvkem. Anotace `@OneToMany` definuje asociaci konkrétní entity s jinou entitou. Jedná se o vztah 1:N. Tento vztah je oboustranný, je nutné definovat parametr `mappedBy`, který určuje vlastníka vazby. Parametr `cascade` udává operace (merge, persist, refresh, remove) pro cílovou entitu a `fetch` definuje zda-li má být asociace řešena okamžitě (eager), nebo ne (lazy).

```
@Entity(name = "TransformerWinding")
@Inheritance(strategy = InheritanceType.JOINED)
@NamedQuery(name="findAllTransformerWinding",
   query="SELECT winding FROM TransformerWinding
   winding")
public class TransformerWinding extends
   ConductingEquipment {
@ManyToOne(fetch=FetchType.LAZY)
        protected PowerTransformer powerTransformer;
        @Transient
        private AbstractTapChanger
           abstractTapChanger;
```

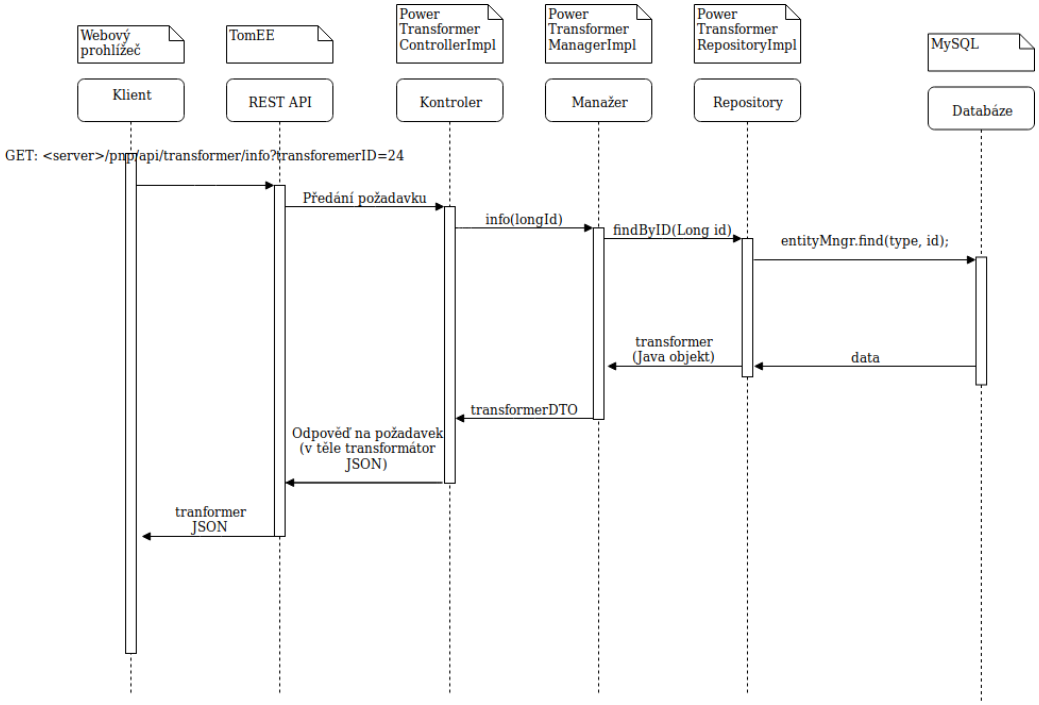
Ukázka 5.9: Definice vinutí transformátoru

Poslední příklad 5.9 ukazuje druhou stranu asociace transformátoru s vinutím transformátoru. Anotace `@ManyToOne` říká, že vztah mezi entitami je N:1. Platí, že transformátor má několik vinutí, ale vinutí patří vždy jen jednomu transformátoru. Díky nastavení ve třídě transformátoru není nutné definovat více parametrů. `@Transient` říká, že tento atribut nemá být ukládán do databáze pomocí ORM.

Příklady v této kapitole ukázaly jen několik anotací s minimem parametrů, které jsou použity v aplikaci. Všechny anotace s popisem parametrů jsou dostupné na [14]. Anotace v balíku *javax.persistence.** mají také kvalitně zpracovanou dokumentaci.

5.4.4 Požadavek transformátoru

Proces zpracování požadavku na transforámtor z klienta je vidět na diagramu 5.1. Na tomto diagramu je vidět výměna dat v rámci aplikace. Celý požadavek je vidět v příloze c (i s opovědí na požadavek).



Obrázek 5.1: Proces zpracování požadavku

6 Testování

Při vývoji aplikace vznikaly a budou i nadále vznikat jednotkové testy, které testují danou funkcionalitu. Jde vždy jen o testy základní funkcionality se základními daty a jedná se většinou pouze o takzvaný „Happy day“ scenář.

Otestován byl zatím správný převod doménových objektů na objekty v databázi. Ověřuje se i správnost uložených dat – tedy správný převod hodnot z aplikace do databáze a nazpátek.

V rámci bakalářské práce Lukáše Černého [21] vzniklo i testování přihlášení uživatele přes token, který se skládá z přihlašovacích údajů. Token je nutné dekódovat a získat potřebné údaje. Test popisující tuto funkčnost ověřuje, že nástroj funguje korektně pro různé chyby, které mohou při vytvoření tokenu vzniknout. Druhou nejdůležitější operací je změna uživatelských práv na model. Test obsahuje všechny možnosti, které mohou vzniknout. Například přidávání, odebírání, úprava práv.

V aplikaci také vznikly testy korektního ukládání živých (dynamických) dat do databáze a také testy načítání dat. Jedná se o načítání statických a dynamických dat z textových (XML) souborů a převod dat ze stávající databáze. Druhá zmíněná možnost však předpokládá přístup do databáze na vzdáleném serveru.

V průběhu vývoje jsme v aplikaci přešli v rámci specifikace JPA z implementace **Hibernate** na implementaci **OpenJPA**. Pro jednoduchost však jednotkové testy stále využívají Hibernate. Používání implementace Hibernate umožňuje lokální spuštění testu (bez nasazení na server). Pro správné fungování jednotkových testů je nutné využívat databázový systém MySQL. V nastavení projektu *pom.xml* je specifikováno používání knihovny *mysql-connector-java* ve verzi 5.1.6 pro komunikaci aplikace s databází **MySQL**. Novější verze konektoru byla testována, ale práce s ní byla složitější. Nastavení knihovny Hibernate je uvnitř souboru *hibernate.cfg.xml* a konfiguruje:

- Komunikaci s databází.
- Definování jména konkrétní databáze.
- Přihlášení k databázi.
- Nastavení knihovny Hibernate.
- Které objekty budou ukládány do databáze pomocí ORM.

Při testování mezi platformami jsme také zjistili, že každý operační systém a prostředí pracuje jinak s cestami uvnitř aplikace – stejná relativní adresa v kódu byla ve výsledku rozdílná v rámci aplikačního serveru na operačních systémech Windows a Linux. Počáteční adresa všech cest aplikačního serveru je uvedena ve třídě **ApplicationConfig.java**.

K zjištění konkrétní chyby vnitřních cest aplikačního serveru je dobré využít nástrojů pro vývojáře webového prohlížeče. Tento nástroj pro vývojáře ukazuje, které prvky jsou požadovány a na jakých cestách mají být dostupné.

V současné době je testována pouze výměna dat mezi aplikací a databází. Myslím si, že do budoucna by bylo určitě vhodné vytvořit komplexnější testy s použitím nástrojů pro testování webových aplikací, jako je například **Selenium** a pomocí tohoto nástroje poté testovat celý průchod aplikací – tedy od vstupu na URL adresách po skutečné uložení dat v databázi a naopak.

7 Závěr

Výsledkem této práce je aplikace vyvíjená podle specifikace Java EE. Aplikace zprostředkovává výměnu dat mezi klientem (který vznikl v rámci první části vývoje) a databázovým systémem MySQL. Aplikace rozšiřuje a mírně upravuje služby původního projektu a přidává možnosti správy uživatelů s možnou autorizací přístupu k datům aplikace.

V průběhu práce jsem se naučil mnoho o vývoji podnikových aplikací. Naučil jsem se, jak složité je přebírat projekt bez znalosti použitých technologií. Práce mi pomohla pochopit koncept ORM[1] a naučila mě pracovat s knihovnami, které jsou postaveny na tomto konceptu (Hibernate, OpenJPA). Během vývoje jsem pracoval na několika strojích (s rozdílným HW) a vyzkoušel přenositelnost aplikací psaných v jazyce Java mezi různými operačními systémy (Ubuntu, Fedora a Windows 7). Zpočátku jsem využíval vývojové prostředí IntelliJ IDEA, ale později jsem přešel na vývojové prostředí Eclipse, které považuji za více intuitivní.

Ve druhé fázi vývoje aplikace „Vizualizace přenosové“ sítě je prioritou efektivita serveru. Server je rozdělen do několika vrstev a vznikly dvě bakalářské práce. První bakalářská práce se zabývala vývojem prezentační vrstvy a správou uživatelů. Druhá bakalářská práce (tato) měla za úkol vytvořit prostředky pro komunikaci aplikace s databází a vytváření modelu přenosové sítě.

[1] Objektově relační mapování

a Požadavek elektrické vedení

```
GET: http://localhost:8080/pnp/api/power_line/info?powerLineID=2067

Headers:
X-auth-Token: 1715ea37e19b41130cbdba2aafa5f4e0c41ee0fc......
SID: riigl7epflojjskfbvevn2ff4t
Content-Type: application/json

Response:
[
    {
        "originalID": "BIE_EL_TU.2999379",
        "location": {
            "latitude": 49.56382480293602,
            "longitude": 13.261813712293788
        },
        "voltageLevel": {
            "voltage": 22000,
            "name": "22 kV",
            "id": 5
        },
        "resistance": 307.94157,
        "reactance": 198.4846152,
        "susceptance": 704.4601542,
        "length": 500.718,
        "powerLineType": {
            "type": "22V_50_AlFe6",
            "resistancePhase": 0.205,
            "reactancePhase": 0.13213333333333332,
            "diameterPhase": 50,
            "ratedCurrent": 531
        },
        "id": 2067
    },]
```

Ukázka 7.1: Požadavek na elektrické vedení

b Požadavek generátor

```
GET: http://localhost:8080/pnp/api/generator/info?generatorID=12345

Headers:
X-auth-Token: 1715ea37e19b41130cbdba2aafa5f4e0c41ee0fc......
SID: riigl7epflojjskfbvevn2ff4t
Content-Type: application/json

Response:
{
        "originalID": "123456",
        "name": "Generator",
        "aliasName": "generatingUnit",
        "description": "generator",
        "location": null,
        "normallyInService": true,
        "aggregate": true,
        "phaseCode": null,
        "baseVoltage": 10.0,
        "voltageLevel": {
                "voltage": 10,
                "name": "napeti",
                "id": 1
        },
        "nominalPower": 10.0,
        "nominalVoltage": 10.0,
        "nominalPowerFactor": 10.0,
        "controlMode": null,
        "pqdiagram": [
                [1.0, 2.0, 3.0]
        ],
        "id": 123456
}
```

Ukázka 7.2: Požadavek na generátor

c Požadavek transformátor

```
GET: http://localhost:8080/pnp/api/transformer/info?transformerID=24

Headers:
X-auth-Token: 1715ea37e19b41130cbdba2aafa5f4e0c41ee0fc......
SID: riigl7epflojjskfbvevn2ff4t
Content-Type: application/json

Response:
{
        "originalID": "-3099ad16:150b7a0f5a4:-7251",
        "name": "AELBS21_AELBS22_1",
        "aliasName": null,
        "description": null,
        "json": null,
        "location": {
                "latitude": "NaN",
                "longitude": "NaN"
        },
        "normalIyInService": null,
        "aggregate": null,
        "bmagSat": null,
        "magBaseU": null,
        "magSatFlux": null,
        "noLoadLoss": null,
        "noLoadCurrent": null,
        "loadLoss": null,
        "loadCurrent": null,
        "transformerWinding": {

                "originalID": "123456",
                "name": null,
                "aliasName": "Winding",
                "description": "Transformer winding",
                "json": null,
                "location": {
                        "latitude": 10.0,
                        "longitude": 10.0
                },
                "normalIyInService": true,
                "aggregate": true,
                "phaseCode": null,
                "baseVoltage": null,
                "voltageLevel": {
                        "voltage": 10,
                        "name": "napeti",
                        "id": 1
                },
                "resistance": null,
                "reactance": 21.0,
                "susceptance": 21.0,
                "conductance": 10.0,
                "ratedApparentPower": 21.0,
                "insultationVoltage": 15.0,
                "grounded": true,
                "resistanceGround": null,
                "reactanceGround": 21.0,
                "windingType": 3,
                "windingConnection": null,
                "id": 123456

        },
        "id": 24
}
```

Ukázka 7.3: Požadavek na transformátor (a jeho vinutí)

d Požadavek přepínač

```
GET: http://localhost:8080/pnp/api/switch/info?switchID=7

Headers:
X-auth-Token: 1715ea37e19b41130cbdba2aafa5f4e0c41ee0fc......
SID: riigl7epflojjskfbvevn2ff4t
Content-Type: application/json

Response:
{
        "originalID": "-3099ad16:150b7a0f5a4:-726c",
        "name": "ABURRE2_AELBS12_1:1:swt",
        "aliasName": null,
        "description": null,
        "json": null,
        "location": {
                "latitude": "NaN",
                "longitude": "NaN"
        },
        "normallyInService": null,
        "aggregate": null,
        "phaseCode": null,
        "baseVoltage": null,
        "voltageLevel": {
                "voltage": 0,
                "name": "0",
                "id": 0
        },
        "normalOpen": true,
        "retained": null,
        "id": 7
}
```

Ukázka 7.4: Požadavek na přepínač

e Požadavek zátěž

```
GET: http://localhost:8080/pnp/api/load/info?loadID=1

Headers:
X-auth-Token: 1715ea37e19b41130cbdba2aafa5f4e0c41ee0fc......
SID: riigl7epflojjskfbvevn2ff4t
Content-Type: application/json

Response:
{
        "originalID": "-3099ad16:150b7a0f5a4:-7272",
        "name": "ABABIC2:load",
        "aliasName": null,
        "description": null,
        "json": null,
        "location": {
                "latitude": "NaN",
                "longitude": "NaN"
        },
        "normallyInService": null,
        "aggregate": null,
        "phaseCode": null,
        "baseVoltage": null,
        "voltageLevel": {
                "voltage": 220000,
                "name": "220000",
                "id": 220000
        },
        "constantVariable": null,
        "pqdiagram": null,
        "id": 1
}
```

Ukázka 7.5: Požadavek na zátěž

f Požadavek lowLevel model


```
{ "generatingUnits": [{
      "connectivities": [{
          "idBaseObject": "string",
          "idConnectivity": "string",
          "idConnectivityNode": "string",
          "temporary": 0
        }],
      "descriptionVoltageLevel": "string",
      "gpsLatitude": 0,
      "gpsLongitude": 0,
      "idVoltageLevel": 0,
      "mRid": "string",
      "name": "string"
    }],
  "loads": [{
      "connectivities": [{
          "idBaseObject": "string",
          "idConnectivity": "string",
          "idConnectivityNode": "string",
          "temporary": 0
        }],
      "descriptionVoltageLevel": "string",
      "gpsLatitude": 0,
      "gpsLongitude": 0,
      "idVoltageLevel": 0,
      "mRid": "string",
      "name": "string"
    }],
  "powerTransformers": [{
      "connectivities": [{
          "idBaseObject": "string",
          "idConnectivity": "string",
          "idConnectivityNode": "string",
          "temporary": 0
        }],
      "descriptionVoltageLevel": "string",
      "gpsLatitude": 0,
      "gpsLongitude": 0,
      "idVoltageLevel": 0,
      "mRid": "string",
      "name": "string"
    }],
  "switches": [{
      "connectivities": [{
          "idBaseObject": "string",
          "idConnectivity": "string",
          "idConnectivityNode": "string",
          "temporary": 0
        }],
      "descriptionVoltageLevel": "string",
      "gpsLatitude": 0,
      "gpsLongitude": 0,
      "idVoltageLevel": 0,
      "mRid": "string",
      "name": "string"
 }]}
```


Ukázka 7.6: lowLevel model – zjednodušený

Literatura

[1] Bondy, J. A. – Murty, U. S. R. *Graph theory with applications (5th Ed.).* New York: Elsevier Science Publishing Co. Inc., 1982. ISBN 04-441-9451-7.

[2] Dixon-Peugh, D. et al. *Apache Commons Graph* [online]. 2011. [cit. 12.04.2018]. Dostupné z: `http://commons.apache.org/sandbox/commons-graph/`.

[3] Dutot, A. et al. *GraphStream* [online]. 2015. [cit. 12.04.2018]. Dostupné z: `http://graphstream-project.org/`.

[4] Eclipse, F. *Eclipse Jetty* [online]. 2018. [cit. 17.06.2018]. Dostupné z: `https://www.eclipse.org/jetty/`.

[5] Faktor, Z. *Transformátory a cívky, První vydání.* BEN, technická literatura, 1999. ISBN 80-86056-49-X.

[6] Hat, R. *WildFly* [online]. 2018. [cit. 15.06.2018]. Dostupné z: `http://wildfly.org/`.

[7] Holub, P. *Teorie grafů* [online]. ZČU, FAV, KMA, 2018. [cit. 09/04/2018]. KMA/DMA - Teorie grafů. Dostupné z: `https://courseware.zcu.cz/CoursewarePortlets2/DownloadDokumentu?id=94913`.

[8] IEC 61970-301:2016. TC 57 - Power systems management and associated information exchange. Standard, International Electrotechnical Commission, Geneva, CH, 2016.

[9] IETF. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content* [online]. 2014. [cit. 16.6.2018]. Dostupné z: `https://tools.ietf.org/html/rfc7231`.

[10] Naveh, B. – přispěvatelé. *JGraphT* [online]. 2017. [cit. 12.04.2018]. Dostupné z: `http://jgrapht.org/`.

[11] Oracle, C. *Glassfish* [online]. 2018. [cit. 17.06.2018]. Dostupné z: `https://javaee.github.io/glassfish/`.

[12] Oracle, C. *JSR 356, Java API for WebSocket* [online]. 2018. [cit. 15.06.2018]. Dostupné z: `https://www.jcp.org/en/jsr/detail?id=356`.

[13] Oracle, C. *JavaEE* [online]. 2018. [cit. 10.06.2018]. Dostupné z: `https://docs.oracle.com/javaee/7/tutorial/overview007.htm#BNACL`.

[14] Oracle, C. *TopLink JPA Annotation Reference* [online]. 2018. [cit. 15.06.2018]. Dostupné z: `http://www.oracle.com/technetwork/middleware/ias/toplink-jpa-annotations-096251.html`.

[15] Organization, B. F. *Big Faceless Graph Library* [online]. 2018. [cit. 12.04.2018]. Dostupné z: `http://bfo.com/`.

[16] foundation, A. *Apache TomEE* [online]. 2018. [cit. 10.06.2018]. Dostupné z: `http://bfo.com/`.

[17] foundation, A. *Apache Tomcat* [online]. 2018. [cit. 10.06.2018]. Dostupné z: `http://tomcat.apache.org/`.

[18] Team, T. J. F. D. [online]. 2010. [cit. 12.04.2018]. Dostupné z: `http://jung.sourceforge.net/`.

[19] yWorks, T. d. c. *yFiles* [online]. 2018. [cit. 12.04.2018]. Dostupné z: `https://www.yworks.com/`.

[20] Černý, J. *Základní grafové algoritmy* [online]. KAM, MFF UK, 2010. [cit. 09.04.2018]. Dostupné z: `http://kam.mff.cuni.cz/~kuba/ka/ka.pdf`.

[21] Černý, L. Návrh a implementace serveru pro podporu vizualizace přenosové soustavy. Master's thesis, FAV, ZČU, Plzeň, 2017.

[22] ČSN 33 0050-601. Mezinárodní elektrotechnický slovník. Kapitola 601: Výroba, přenos a rozvod elektrické energie. Všeobecně. Standard, Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, Biskupský dvůr 1148/5, 110 00 Praha 1, 1994.