VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vizualizace podnikových procesů

2006

Jan Dajč

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 16. května 2006 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Bakalářská práce se zabývá možností vizualizace podnikových procesů v 3D prostoru, v kterém bude výsledný graf automaticky uspořádán na základě fyzikálních sil. Práce se také snaží porovnat přehlednost grafů v 2D a 3D prostoru. K implementaci byl použit programovací jazyk Java resp. Java 3D.

Klíčová slova: Podnikový proces, Java, Java 3D, XML, Koordinační diagram

Abstract

Thesis presents posibility of business processes visualization in 3D area, in which the graph will be automatically organized in terms of physical forces. Thesis also tries to compare the transparency of the graphs in 2D and 3D area. The Java and Java3D technology was used for the implementation.

Keywords: Business proces, Java, Java 3D, XML, Coordinating diagram

Obsah

Seznam obrázků

1 Úvod

Podnikový proces se nejlépe pochopí z přehledného diagramu, který tento proces představuje. Tento diagram tvoří člověk, který prvky rozmístí a pospojuje. BPStudio je nástroj pro jejich modelování a usnadňuje tak práci při tvoření diagramů procesu. Vždy ale musí člověk tyto prvky rozmístit.

Cílem této práce by měla být aplikace schopná usnadnit lidem práci při vytváření modelů podnikových procesů. Měla by být schopná automaticky vytvořit diagram podnikového procesu jen na základě znalosti propojení jednotlivých objektů. Tento diagram by měl být dostatečně přehledný a také úhledný.

Proto nejprve analyzuji diagramy vytvořené lidmi, abych zjistil, jestli je nějaké pravidlo, které by se dalo použít při automatickém uspořádání prvků. Poté navrhnu aplikaci, která by zjištěné poznatky odzkoušela na již vytvořených modelech v programu BPStudio. Závěrem ji naimplementuji a otestuji, zda obstojí konkurenci lidmi vytvářených grafů.

2 Podnikový proces

Chceme-li vytvořit informační systém, který má být nasazen ve službách, bankovnictví, nebo ve výrobě, musíme nejdříve popsat procesy, které v danném odvětví probíhají a podle kterých určíme chování celého informačního systému. Tyto procesy, které popisujeme, nazýváme tzv. podnikovými procesy, neboli business processes. Abychom je mohli lépe prezentovat a snadněji jim porozumět, byla vyvinuta účinná metoda pro jejich modelování, kterou nazýváme BPM (Business Process Modeling).

2.1 Metoda BPM

BPM je formalizovaný nástroj vizuálního modelování, který umožňuje strukturální analýzu nutnou pro definování organzační struktury a analýzu prostřednictvím simulace nutnou ke kvantifikaci procesu (doba trvání, náklady, atd.). Modely lze poté využít i k počítačem podporovanému řízení a monitorování procesů. Model specifikuje podnikový proces ze tří relativně nezávislých pohledů, funkčního, objektového a koordinačního.

2.1.1 Funkční model

Funkční model specifikuje architekturu procesu, včetně určení jeho uživatelů a produktů a definuje vazby mezi podprocesy (spolupráci a obsažení). Úkolem modelu je zjistit, jaké funkce jsou požadované, jaké procesy je realizují a jaká je jejich struktura.

2.1.2 Objektový model

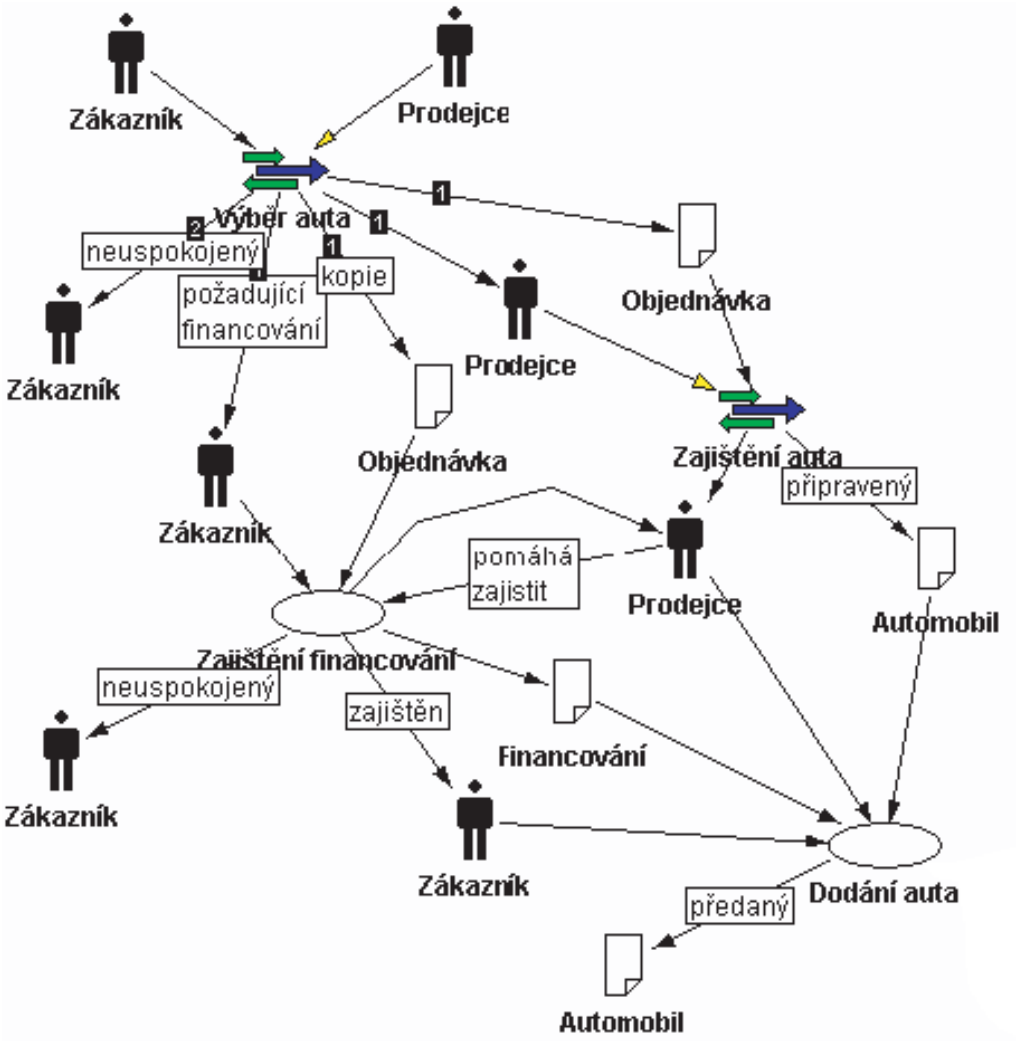
Znázornůje statickou strukturu obsahující všechny objekty, které v procesu vystupují. Objekty rozlišujeme na aktivní a pasivní podle jejich aktivní účasti. Aktivní objekty jsou realitázoři procesu a pasivní objekty jsou procesem manipulovány, vytvářeny či spotřebovávány. Model je sestavován pro každý proces definovaný ve funkčním modelu zvlášť.

2.1.3 Koordinační model

Vychází z předchozích dvou modelů a definuje, jak ve skutečnosti bude proces probíhat. K simulaci i řízení procesu používá koordinační model principů Petriho sítí. Základními prvky josu aktivní objekt, pasivní objekt a aktivita.

Obrázek 1: Prvky koordinačního modelu metody BPM

Aktivní objekt reprezentuje uživatele, osoby zodpovědné za proces či aktivitu. Pasivní objekt reprezentuje materiály, produkty, nebo data v podnikovém procesu a je charakterizován svými atributy. Příklad koordinačního modelu je na obrázku 2.



Obrázek 2: Ukázka koordinačního modelu metody BPM

2.2 Analýza systému na zobrazení

Koordinační model je pro popis procesu nejvýstižnější a proto se jím budu dále zabývat v rámci mé vizualizace podnikového procesu. Jak je vidno z obrázku 2, je graf celkem přehledný a uspořádaný. Do tohoto stádia přehlednosti se však dostal tak, že ho musel člověk, který jej vytvářel, takto uspořádat. Existují samozřejmě metody, které nám automaticky graf sestaví, ale je zde mnoho nepříjemností se kterými se musí daný algoritmus

vypořádat. Například si můžeme vzít křížení hran. Chceme li ve složitém grafu zajistit nezkřížení hran, nemusí to vždy jít a existují případy, ve kterých se jednoduše bez křížení neobejdeme. Další nástrahou pak mohou být překrývající se vrcholy, které se překrývají buď z části, nebo dokonce celé tak, že jeden vrchol může být skryt za druhým. Tyto situace se ale dají lehce algoritmem na uspořádání uhlídat, ale nezajistí nám pěkně vypadající a estetický graf.

Tyto poznatky platí pro grafy v 2D prostoru. Cílem této práce je přidat ještě jednu dimenzi navíc. Přidáním této dimenze vznikne zcela nový manipulační prostor, který nám umožní trochu jiný pohled na vytváření grafu. Jak už jsem dříve zmínil, tak pro 2D existují již postupy pro vytvoření grafu a ty budou samozřejmě fungovat i v 3D prostoru, ale je zde otázkou, podle čeho přidávat do grafu ještě jednu souřadnici. Vezměme si například jednoduchý algoritmus pro graf v rovině, který vrcholy grafu rozmístí po obvodu kružnice a pospojuje je tak, jak mu bylo zadáno. Takto vzniklý graf nebude moc pěkný a navíc, bude-li trochu složitější, budou se hrany křízit. Jestliže ale tento graf převedeme do 3D a úplně jednoduše dáme každému vrcholu jinou souřadnici z, která bude vyjadřovat např. výšku, tak nám vznikne, z určitého pohledu, v podstatě totožný graf. Kouzlo 3D prostoru je, že si tento graf můžeme otočit a tak se rázem zobrazí rozdíly výšky vrcholů a bude vidět, že se hrany nekříží. U tohoto jednoduchého algoritmu nebude výsledek oslnivý, chceme totiž, aby graf působil spořádaně podle daných pravidel a ne jako kružnice. Koordinační diagram na obrázku 2 by nejspíše nevypadal moc pěkně, kdybychom jeho vrcholy rozmístili po obvodu kružnice. Podíváme li se na tento diagram blíže, tak z něj vyčteme některé vlastnosti, které budeme moci použít u automatického tvoření grafu právě pro koordinační model.

2.2.1 Charakteristiky koordinačního diagramu

Na obrázku 2 vidíme, že vrcholy (aktivní objekty, pasivní objekty a aktivity) které mají mezi sebou hranu, jsou si nejblíže. Bude tedy třeba navrhnout sysém, který by tyto vrcholy udržoval co nejblíže, ale aby byly zároveň čitelné a nebyly tak blízko, aby se překrývaly. Ostatní vrcholy, které mezi sebou hranu nemají, jsou od sebe na první pohled vzdáleny libovolně. Podíváme li se na graf pořádně, zjistíme, že dva vrcholy, které mezi sebou hranu nemají jsou od sebe vzdáleny v závislosti na počtu vrcholů, přes které jsou tyto dva vrcholy na sebe napojeny. Bude-li například z vrcholu A do vrcholu B cesta rovna 3 bude jejich vzdálenost menší, než mezi vrcholy A a C, mezi kterými je cesta rovna 6. Samozřejmě toto pravidlo nemusí platit vždy. Dáme-li dvoum lidem stejný úkol, aby sestavili graf, u kterého budou pouze vědět, které vrcholy jsou mezi sebou spojeny, tak každý z nich sestaví graf tak, aby byl pro něj přehledný. Úkolem této práce je však tento postup automatizovat a vyzkoušet, jestli se alespoň přiblíží výsledkům, které by byly pro člověka přehledné, takže budu vycházet z poznatků o vzdálenostech, které jsem zmínil výše.

Vzhledem k vlastnostem koordinačního diagramu se tedy v této práci pokusím navrhnout systém vizualizace, který by se dal využít pro zobrazení pracovního procesu a který by byl stejně, nebo dokonce i lépe přehledný, přičemž jej vytvoří sám stroj a ne člověk. Podle dvou hlavních vlastností, které jsou zmíněny výše bych chtěl rozvinout

myšlenku, která se používá u 2D grafů. Je to tzv. ”pružinkový graf” který nahrazuje hrany pružinkami. Tato záměna zajistí, aby vrcholy spolu spojené zůstávaly mezi sebou v určitých vzdálenostech. Toto rešení se používá v 2D a bude tak dobré sledovat, jak bude vypadat po převodu do 3D. Druhá vlastnost diagramu je vzdálenost mezi nespojenými vrcholy. Toto se pokusím nasimulovat pomocí odpudivé elektrostatické síly, která bude působit mezi všemi vrcholy grafu.

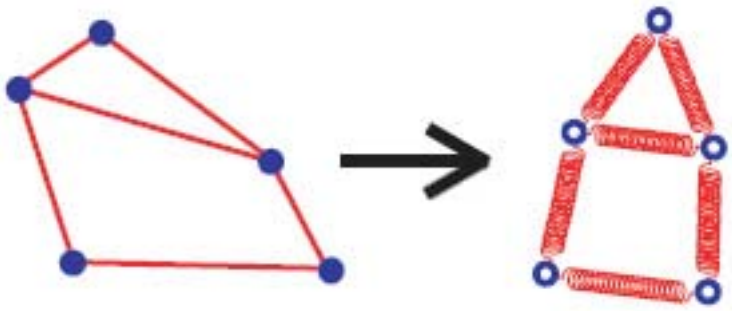
3 Návrh systému na vizualizaci procesů

3.1 Automatické rozložení prvků v grafu

Jak už bylo řečeno v předchozí kapitole, bude se graf uspořádávat automaticky, bez pomoci lidského zásahu. Umožní nám to tak např. generovat diagramy již existujicího podnikového procesu z komponent. Postup je většinou opačný, tedy že jsou dříve vytvořeny diagramy pro popis procesu a pak až se začína programovat, ale navržený systém by se měl dát použít i při opačném postupu, což je jeho výhoda. Cílem je hlavně dosáhnout takového výsledku, aby byl graf přehledný a byl podobný rozmístění prvků v grafu tak, jako by je rozmístil člověk. Proto jsem podle dvou hlavních vlastností koordinačního modelu zvolil tzv. "pružinkový graf".

3.2 Pružinkový graf

Tato metoda se používa ve 2D grafech na uspořádání prvků. Funguje na principu nahrazení hran grafu pružinami, které k sobě přitahují dva vrcholy, mezi nimiž je hrana. Tento algoritmus je celkem efektivní a většinou se podaří nepěkné grafy převést na grafy mnohem přehlednější a kompaktnější. Toho se dosáhne tak, že všechny pružiny mají tendenci se dostat do vyváženého stavu, přičemž změna jedné ovlivňuje ostatní. U tohoto algoritmu je pěkné sledovat krok po kroku, co se s celým grafem děje, protože je to zajímavá podívaná, jak pouhé zavedení jedné síly do grafu, dokáže s celým grafem hýbat a uspořádávat jej. Nicméně v 2D prostoru nám tento mechanismus nevyřeší křížení hran, jednoduše se může stát, že se hrany překříží, protože tento model samotný křížení nějak neřeší. V 3D se tento model pružinkového grafu využívá jen zřídka a proto se v této práci zaměřím na jeho převod do 3D. Tento převod sám o sobě není nějak složitý, použijí se principy nahrazení hran stejně tak, jako v 2D. Vzniká nám však větší prostor pro manipulaci s vrcholy. Proto bude třeba ještě zajistit, aby byly vůči sobě uspořádány i vrcholy, které mezi sebou hranu nemají.



Obrázek 3: Příklad nahrazení hran pružinkami

Při nahrazování hran pružinami, nám postačí jednoduchý fyzikální vzorec pro pružinový oscilátor, který má harmonické kmity a závisí pouze na tuhosti pružiny k a vzdále-

nosti vrcholů d. Vzorec přitažlivé síly F_p proto bude:

$$F_p = k * d \tag{1}$$

Tuhost pružiny k pak bude určovat, jak daleko od sebe vrcholy budou ve výsledném grafu.

3.2.1 Vylepšený pružinkový graf

Samotné nahrazení hran pružinami neřeší problém překřížení hran a vzdáleností vrcholů, mezi kterými hrana není. Proto budu muset zavést do celého systému ještě sílu, která bude působit mezi úplně všemi vrcholy, nehledě na to, zda-li mezi sebou hranu mají, či nikoliv. Na výběr bylo z více možností, at' už gravitační, či magnetická síla. Vybral jsem ale sílu elektrostatickou, konkrétně sílu, mezi dvěmi bodovými náboji. Budu ji používat v tom smyslu, že tyto bodové náboje budou představovat vrcholy grafu a nezáleží, jestli jsou spojeny hranou. Tuto elektrickou odpudivou sílu označím jako F_o, která se vypočítá takto:

$$F_o = k * \frac{Q_1 * Q_2}{r^2} \tag{2}$$

Kde k označuje konstantu úměrnosti, která závisí na prostředí, v kterém se náboje vyskytují, Q_1 a Q_2 jsou velikosti bodových nábojů a r je jejich vzdálenost. V praxi toto funguje tak, že jsou-li oba náboje nabity souhlasně, pak se se odpuzují, jsou li nabity opačně, pak se přitahují. Tohoto využijeme pro naše vrcholy tak, že všechny budou nabity souhlasně a tím pádem se budou všechny od sebe odpuzovat. Tím pádem, jestliže by v celém grafu nebyla jediná hrana, "rozletěly" by se vrcholy do všech stran. Budou-li v grafu hrany, které nahradíme pružinkami, začne na každý vrchol působit více sil, které se časem dostanou do rovnováhy a vytvoří tak námi požadovaný graf. Díky této vlastnosti se nám vyřeší hned několik problému. Nebude docházet k překrývání vrcholů.

V případě, že by ve grafu byly pouze 2 vrcholy a byla by mezi nimi hrana, pak by na každý vrchol působila výsledná síla F_v, která je dána rozdílem síly přitažlivé F_p a síly odpudivé F_o:

$$F_v = F_p - F_o \tag{3}$$

Ze vzorečků, které jsem použil je zřejmé, že bude záležet hlůavně na vzdálenosti mezi danými vrcholy. Pokud bude mezi vrcholy hrana a budou hodně vzdáleny, pak přitažlivá síla, kterou obstarává pružina, bude velká. Naopak odpudivá síla bude malá. Jestliže se ale vrcholy k sobě začnou přibližovat díky převaze přitažlivé síly, pak začne narůstat síla odpudivá a síla přitažlivá se bude zmenšovat. Z toho je jasné, že se tyto síly po určité chvíli vyrovnají, výsledná síla F_v bude rovna nule a graf bude stabilní.

3.2.2 Výsledné síly v pružinkovém grafu

Při větším počtu vrcholů než dva budou vzorce sil trochu pozměněny, protože vrcholy mohou mít hranu s více než jedním vrcholem a bude-li v grafu více vrcholů, tak na sebe

budou všechny působit onou odpudivou silou. Tím pádem bude vzorec pro přitažlivou (pružinkovou) sílu F_p roven:

$$F_p = \sum_{i=1}^{n}(k * d_i) \tag{4}$$

Kde n je počet vrcholů, s kterými má vrchol hranu, k je tuhost pružiny a d_i je vzdálenost mezi i-tým vrcholem a vrcholem, pro který danou přitažlivou sílu měříme. Obdobně bude vypadat vzorec pro výpočet odpudivé síly F_o:

$$F_o = \sum_{j=1}^{m}(k * \frac{Q * Q_j}{r_j^2}) \tag{5}$$

Kde m je počet vrcholů v grafu, k je konstanta úměrnosti, Q je náboj vrcholu, pro který sílu zjišťujeme, Q_j je náboj j-tého vrcholu a r_j je vzdálenost mezi j-tým vrcholem a vrcholem, pro který danou odpudivou sílu měříme. Vzorec pro výslednou sílu F_v proto bude vypadat:

$$F_v = \sum_{i=1}^{n}(k * d_i) - \sum_{j=1}^{m}(k * \frac{Q * Q_j}{r_j^2}) \tag{6}$$

Samozřejmě chceme li pracovat se silami ve 2D, nebo 3D prostoru musíme s nimi pracovat jako s vektory. Tudíž se při sčítání jednotlivých sil bude postupovat, jako při klasickém skládání vektorů.

3.3 Popis vstupních parametrů

Výsledná aplikace bude graf tvořit automaticky a je tedy rozumné navrhnout nějaký vhodný formát vstupních dat. Možností je celá řada, at' už si vezmeme obyčejné načtení parametrů z jednoduchého textového souboru se seznamem sousedů, nebo například matici sousednosti. K dispozici je ale aplikace BPStudio, která slouží k modelování podnikových procesů a obsahuje nástroj pro export vytvořených procesů do formátu XML. Formát XML je nejlepší volba, neboť se v dnešní době hojně využívá k reprezentaci dat a ke komunikaci mezi aplikacemi různého druhu. Při importu dat z XML souboru, který je vyexportován z BPStudia se budu zabývat jen objekty, které jsou pro tuto práci důležité. V exportovaném souboru totiž nalezneme všechny tři modely podnikových procesu a další různá nastavení, které nebudeme potřebovat. Struktura souboru je přesně stanovena v souboru bpmodel.dtd, který slouží k popisu výstupního souboru BPStudia. V souboru je slovník všech použitých objektů, které se vyskutují ve všech modelech. Příklad uloženého aktivního objektu:

```
<Active id="_2" class="bpm.method.Active">
  <Name>Grafik</Name>
  <Attributes>
```

```
    </Attributes>
    <Services>
    </Services>
</Active>
```

Výpis 1: Aktivní objekt v XML souboru

Pasivní objekt:

```
<Passive id="_4" class="bpm.method.Passive">
    <Name>Produkt</Name>
    <Attributes>
    </Attributes>
</Passive>
```

Výpis 2: Pasivní objekt v XML souboru

Aktivita:

```
<Activity id="_8" class="bpm.method.Activity">
    <Name>korektura</Name>
    <Scenarios>
        <Scenario>
            <Name>default</Name>
            <Time>00:00:00</Time>
            <Cost>0</Cost>
            <Spec/>
        </Scenario>
        <Scenario>
            <Name/>
            <Time>00:00:00</Time>
            <Cost/>
            <Spec/>
        </Scenario>
        <Scenario>
            <Name/>
            <Time>00:00:00</Time>
            <Cost/>
            <Spec/>
        </Scenario>
    </Scenarios>
</Activity>
```

Výpis 3: Aktivita v XML souboru

Podproces:

```
<Process id="_13" class="bpm.method.BusinessProcess" external="false">
    <Name>Prodej auta</Name>
    <Spec>Hlavní proces prodeje auta</Spec>
</Process>
```

Výpis 4: Podpůrný proces v XML souboru

Dále pak následují jednotlivé definice funkčního, objektového a koordinačního modelu. Princip sestavení všech modelů je shodný, takže ukáži, jak je graf v XML souboru

reprezentován. Při výpisu objektů, které se v modelech vyskytují jste si mohli povšimnout, že každý objekt má atribut id. Podle tohoto id se objekty do modelů přiřazují. V každém modelu se může jakýkoliv objekt libovolně opakovat, proto se musí určit nové identifikátory pro jednotlivé modely:

```
<Nodes>
   <Node id="_12" class="bpm.drawing.ActiveNode" element="_2" x="78" y="428"/>
   <Node id="_13" class="bpm.drawing.ActivityNode" element="_10" x="160" y="532"/>
   <Node id="_14" class="bpm.drawing.ActivityNode" element="_8" x="158" y="270"/>
</Nodes>
```

Výpis 5: Objekty v daném modelu v XML souboru

Jediné co zbývá je určit hrany mezi jednotlivými vrcholy:

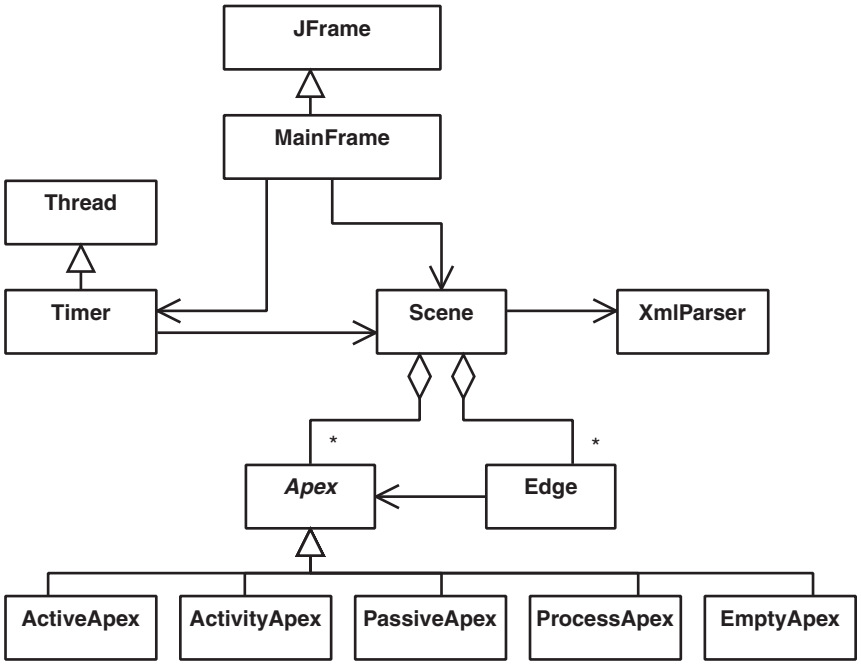
```
<Edge class="bpm.drawing.Responsibility" from="_12" to="_13">
   <Points>
      <Point x="100" y="458"/>
      <Point x="171" y="532"/>
   </Points>
   <Arrow>
      <Point x="167" y="523"/>
      <Point x="171" y="532"/>
      <Point x="162" y="527"/>
   </Arrow>
</Edge>
```

Výpis 6: Příklad hrany grafu v XML souboru

Lze vidět, že všechny parametru jednotlivých objektů potřebovat nebudeme, například pozice vrcholů a hran, ale tento seznam objektů nám trochu napovídá, jaké třídy budou nutné pro naši aplikaci.

3.4 Návrh základních tříd

Na obrázku 4 je zobrazen diagram základních tříd systému. Třída MainFrame je potomkem třídy JFrame a bude představovat hlavní okno aplikace, které bude uživateli zobrazovat celý graf a umožní nastavování různých vlastností základními ovládacími prvky, jako jsou například tlačítka. Dále pak bude odkazovat na třídu Scene a třídu Timer zděděnou ze třídy Thread. Ta se bude starat o časování a o průběh animace a díky tomu, že je potomkem třídy Thread, nám umožní manipulaci s grafem během transformace grafu, protože bude spuštěna jako samostatné vlákno. Bude volat metodu třídy Scene, která bude celý Graf transformovat. Třída Scene bude vlastně celý graf, bude jej tvořit a upravovat. Proto také bude muset shromažďovat prvky, které jsou v grafu obsaženy, tedy vrcholy hrany. Vrcholy budou potomky abstraktní třídy Apex a mohou být různého typu (ActiveApex, PassiveApex, ActivityApex, ProcessApex, EmptyApex). Vrcholy budou děděny z třídy Apex proto, protože v této třídě budou veškeré nástroje na práci s vrcholem a v jednotlivých potomcích bude jen specifikováno, jak budou ve grafu vypadat. Třída Edge bude velice jednoduchá, protože bude reprezentovat hranu a to znamená, že si bude udržovat referenci na dvě instance objektu typu Apex a bude také tvořit výslednou vizualizaci hrany. Zbývá třída XmlParser, která bude importovat XML soubor, který chceme



Obrázek 4: Třídní diagram základních prvků

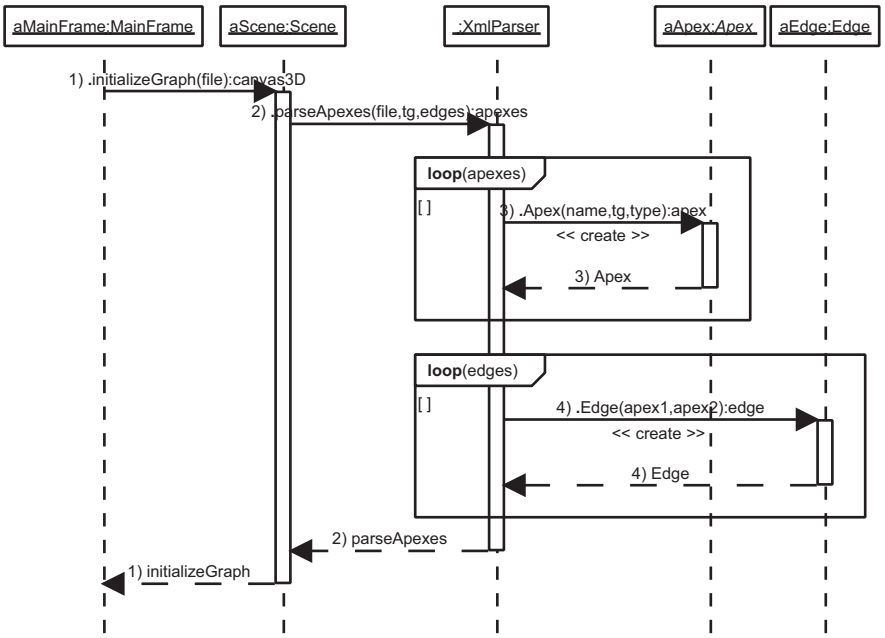
zobrazit jako graf procesu. Tato třída postupně rozebere vstupní soubor a to tak, že si zjistí všechny vrcholy a pak použije jen ty, které budou zobrazeny v koordinačním diagramu, pak už jen doplní a zinicializuje hrany a to vše předá zpět do scény, tedy do instance třídy Scene, která je přidá do scény a zobrazí.

3.5 Popis důležitých funkcí

3.5.1 Import XML souboru

Tato funkce je jednou z nejdůležitějších, nebot'bychom bez ní neměli co zobrazovat. Vše začíná ve třídě MainFrame, která je vpodstatě uživatelské okno, v kterém uživatel klikne někde na import souboru. Třída zareaguje otevřením dialogu pro nahrání souboru a poté co uživatel vybere soubor, volá metodu třídy Scene initializeGraph(file), kde file je název souboru. Třída Scene na toto bude reagovat vytvořením nového prostředí pro graf do kterého bude vkládat nově importované objekty. Ty získá spuštěním metody parseApexes(file) na třídě XmlParser. Ta nahraje vstupní soubor a pokusí se ho projít

a získat z něj potřebné informace pro sestavení vrcholů a hran, které budou do grafu patřit. Poté bude ve smyčkách vytvářet už konkrétní vrcholy a hrany, které potom v příslušných kolekcích vrátí zpět do třídy Scene. Ta přidá objekty do scény a celou scénu předá hlavnímu oknu, které je zobrazí. Celou tuto funkci zobrazuje sekvenční diagram na obrázku 5.



Obrázek 5: Sekvenční diagram importu souboru

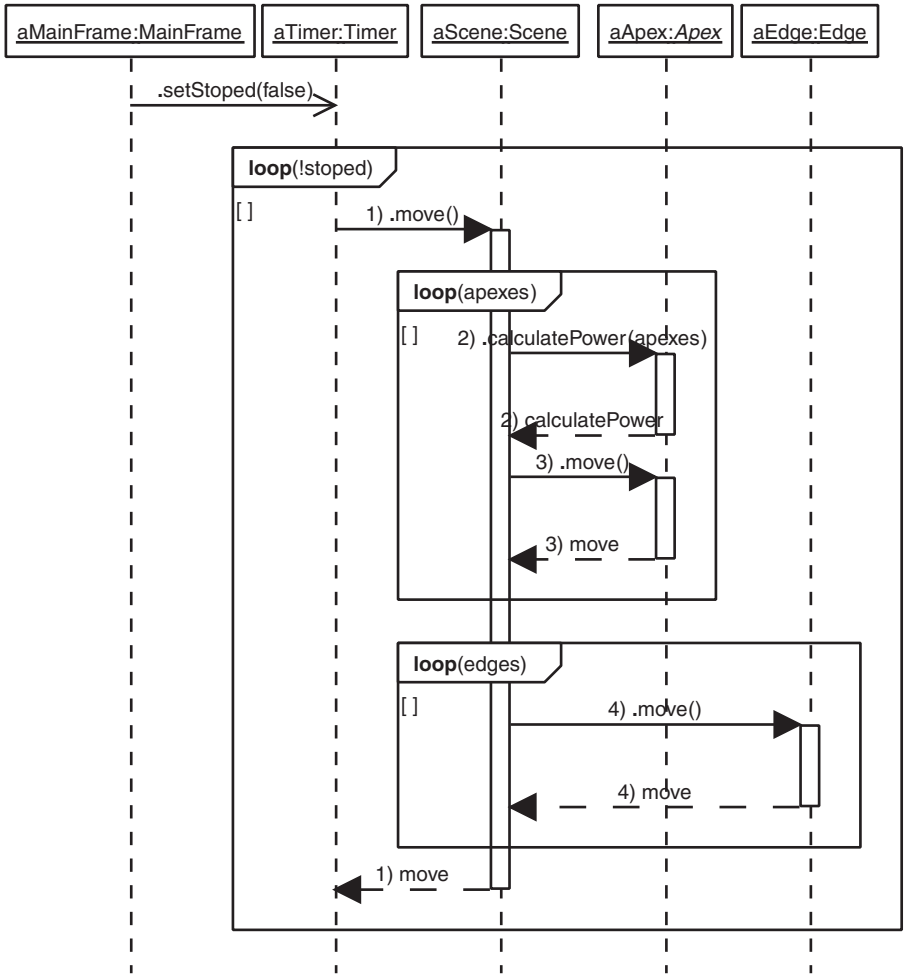
Import souboru:

- 1. Volání metody na nastavení nové scény
- 2. Požádání o analýzu vstupního souboru
- 3. Vytvoření všech vrcholů
- 4. Vytvoření všech hran

V sekvenčním diagramu na obrázku 5 je zobrazena abstraktní třída Apex. Ta se ale nevytváří, nebot'z abstraktních tříd nelze instance vytvářet. Je zde ale z důvodu názorné ukázky, že se vytváří instance podtříd této třídy podle typu, který je v importovaném souboru uveden. Apex je zde především z důvodu málo široké stránky pro rozsáhlejší sekvenční diagram.

3.5.2 Spuštění transformace grafu

Funkce transformace grafu je funkčním jádrem výše popsané metody pro úpravu grafu. Je to vlastně důvod, proč tato práce vznikla. Při průběhu této funkce se totiž graf bude automaticky uspořádávat a po určitém čase dosáhneme očekávaného výsledku, kdy bude graf ve výsledném stavu. Opět začneme v hlavním oknu, které představuje třída



Obrázek 6: Sekvenční diagram spuštění transformace

MainFrame. V tomto okně klikne uživatel na tlačítko Play a spustí tak transformaci.

Instance třídy MainFrame nastaví vlastnost časovače, aby začal v nekonečném cyklu volat metodu move() třídy Scene, která bude danou transformaci plnit. Projde všechny vrcholy a opět na každém z nich zavolá metodu move(), která se postará o vypočítání všech působicích sil na vrchol a posunutí tohoto vrcholu na své nové místo. Jakmile budou všechny vrcholy na nových pozicích, začne procházet všechny hrany a ty se automaticky překreslí na novou pozici, aby ukazovaly spravě.

Spuštění transformace:

- Nastavení vlastnosti časovače stoped na false a by se rozběhl a dostal do nekonečné smyčky
- 1. volání metody move() na instanci třídy Scene pro provedení transformace
- 2. vypočítání celkové síly, která na každý vrchol působí
- 3. posunutí každého vrcholu na svou pozici
- 3. přenastavení pozice každe z hran

4 Implementace systému na vizualizaci grafu

Rozhodl jsem se implementovat aplikaci v programovacím jazyku Java, resp. v její nádstavbě Java3D, takže jen něco málo o této technologii.

4.1 Java

Java je vvyšší objektově orientovaný programovací jazyk nezávislý na platformě. Vyvíjí ho především společnost SUN Microsystems a je zdarma dostupný pro různé operační systémy (Linux, Windows, Solaris). Tuto nezávislost na platformě zajišťuje způsob kompilace. Zdrojové kódy jsou překládány do tzv. byte-kódu, který je pro člověka nečitelný a při spouštění java programu se tento byte-kód převádí na strojový kód příslušného procesoru s ohledem na použitý operační systém. Tento převod provádí tzv. Java Virtual Machine (JVM), což je runtime, který musí být nainstalován tam, kde chceme program napsaný v javě spustit.

4.2 Java3D

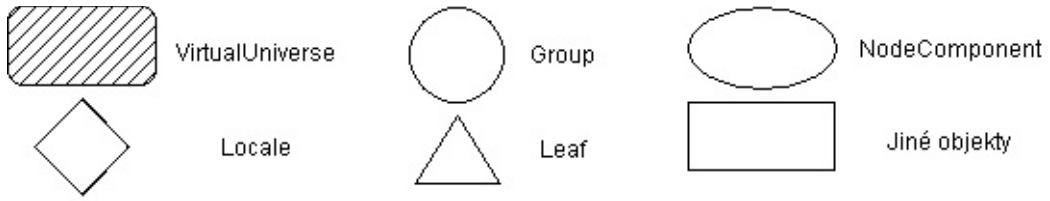
Programovací jazyk Java standardně ale nezahrnuje práci s 3D grafikou.Bylo ale vyvinuto API, které je distribuováno jako volitelný balíček. Jmenuje se Java 3D API a po nahrání balíčku do JVM nám umožňuje v Javě pracovat s 3D scénou, texturami, světly a různými objekty. Poskytuje elegantní mechanismy pro animování scén a pro definování chování objektů. Jako samotná Java je i Java3D platformě nezávislá, avšak pro různé platformy jsou k dispozici příslušné balíčky, nebot'firma SUN implementovala toto prostředí pouze pro Windows a Solaris, nicméně ostatní si pomohli sami a portovali API jak na Linux, tak HP-UX, nebo také Irix. K dispozici jsou dvě verze a to jak pro podporu OpenGL, tak DirectX.

4.2.1 Scéna v Java 3D

Základem scény je objekt VirtualUniverse, který buď přímo, nebo nepřímo odkazuje na všechny ostatní objekty, které celou scénu tvoří. Třída VirtualUniverse se však přímo nepoužívá, protože pro mnohé účely je dostačující její podtřída SimpleUniverse, která se používá místo ní a jejíž využití je mnohem snažší. Všechny objekty ve scéně vytvářejí stromovou strukturu, jejímž kořenem je právě VirtualUniverse, který odkazuje na objekt Locale, což je "počátek souřadné soustavy scény". Objekt Locale pak odkazuje na objekty BranchGroup, které jsou kořeny jednotlivých částí scény. Slovy se celá scéna jen těžko popisuje a tak byly určeny jednoduché značky pro zakreslení celé scény do grafu.

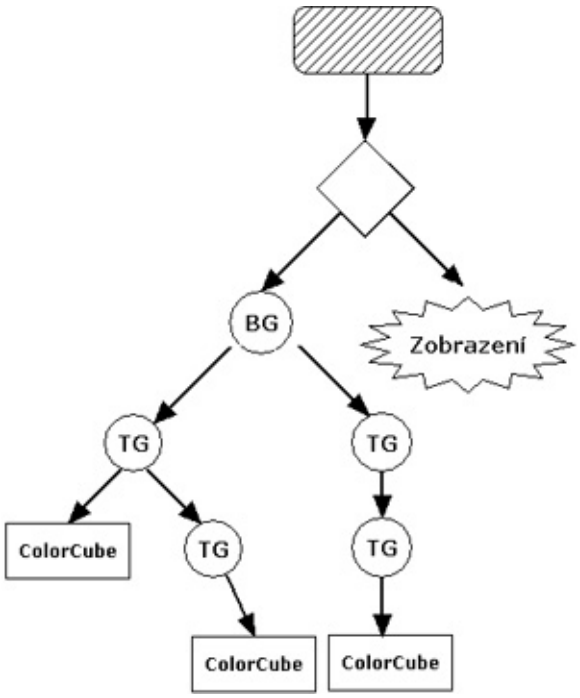
4.2.2 Graf scény v Java 3D

Jak jde vidět z obrázku 7, jsou symboly pro zakreslení grafu velice jednoduché. VirtualUniverse a Locale, jsem popisoval v sekci Scéna v Java 3D. Objekty Group jsou instance tříd odvozených od třídy Group. V grafu plní funkci uzlů, tzn. že mohou mít pouze jednoho rodiče a neomezeně mnoho potomků. Třída Leaf je nadtřídou pro třídy, jejichž



Obrázek 7: Prvky grafu scény v Javě 3D

instance jsou v daném stromu listy, což znamená, že nemají žádné potomky. NodeComponent je nadtřídou například třídy Appearance, která určuje některé vlastnosti objektů Leaf ve scéně. Tyto podtřídy nejsou formálně součástí stromu, neboť na ně odkazují objekty Leaf. Nejedná se proto formálně o vztah potomek-rodič. Obdelník znázorňující jiné objekty se používá z toho důvodu, že trojúhelník pro objekty typu Leaf se nedá dost dobře použít pro dlouhé názvy objektů a proto se listy většinou značí obdélníkem. Obrázek 8



Obrázek 8: Jednoduchý graf scény v Javě 3D

ukazuje příklad jednoduchého grafu scény, který popisuje 3 krychle ve scéně, z toho dvě jsou transformovány jedním společným objektem TransformGroup, přitom jedna je transformována ještě dalším objektem TransformGroup a třetí krychle je nezávisle na dvou předchozích krychlích transformována dvěma objekty typu TransformGroup. Z před-

chozí věty je patrné, že grafické znázornění je mnohem výstižnější a lepší na pochopení, než slovní vyjádření.

4.3 Popis balíků aplikace

Program je kvůli přehlednosti jednotlivých tříd hierarchicky rozdělen do několika balíku.

- **graph** - hlavní třídy aplikace (gui)
- **graph.scene** - třída scény, která tvoří graf
- **graph.objects** - třídy představující prvky ve scéně (vrcholy a hrany)
- **graph.objects.apexes** - třídy reprezentujicí vrcholy (aktivní, pasivní, atd.)
- **graph.utils** - třídy pro nastavení systému, import dat a časovače

4.4 Popis jednotlivých tříd a jejich vybraných metod

4.4.1 Balíček graph

4.4.1.1 Třída Main Třída Main je hlavní spouštěcí třída, obsahuje pouze metodu main(String[] args), která zajistí spuštení celé aplikace. Vytvoří instanci třídy MainFrame a nastaví ji viditelnou

4.4.1.2 Třída MainFrame Tato třída je hlavní GUI celé aplikace, je podtřídou třídy JFrame. Obsahuje ovládací prvky (menu, text inputy, talčítka) skrz které umožnuje nastavovat scénu. Většinu okna však tvoří prostor pro zobrazení plochy, ten zabrán instancí třídy Canvas3D umožnující vytvoření 3D scény pro graf. Implemetuje základní funkce pro obsluhu tlačítek a stisku klávesy enter v poli s nastavením.

Metoda loadFile() slouží k zobrazení dialogu pro otevření souboru, který je určen k importu.

Metoda initSettings() nastaví počáteční hodnoty pro ovládací prvky podle vrcholů ve scéně.

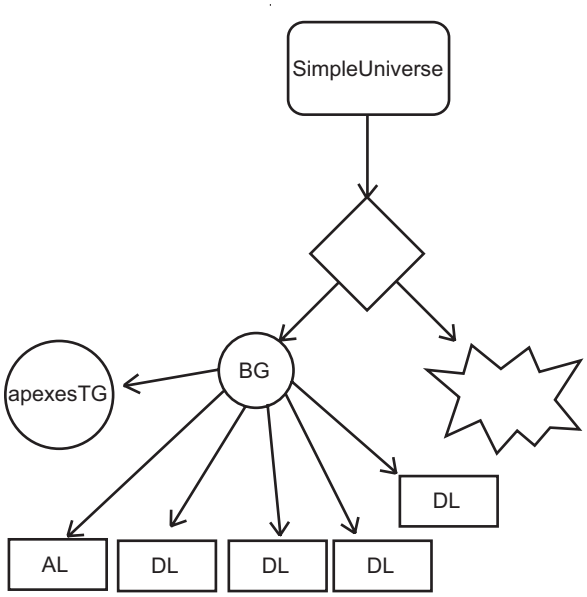
Metoda setMassAndStiffnes() nastavuje hodnoty vrcholů z ovládacích prvků.

Metoda animate() spouští, či zastavuje transformaci grafu.

Metoda captureJpeg() uloží aktuální snímek scény do jpg obrázku.

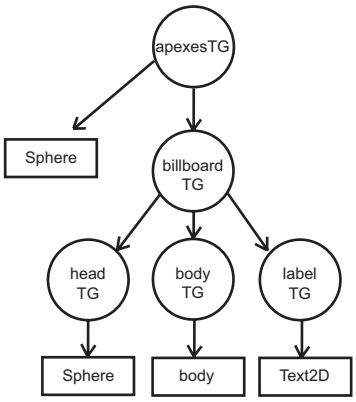
4.4.2 Balíček graph.scene

4.4.2.1 Třída Scene Tato třída je jediným zástupcem toho balíčku, ale je pro celou aplikaci stěžejní. Tvoří totiž celou scénu, do které se vkládají vrcholy, hrany, popisky a světla. Nejlépe je celá scéna popsatelná grafem scény, ten je nekompletní na obrázku 9, ale to jen z důvodu přehlednosti a místa na stránce. Z tohoto grafu můžeme vyčíst, že bude ve scéně 5 zdrojů světla a to čtyřikrát světlo směrové - DL (Directional Light) a jedno světlo okolní - AL (Ambient Light). Dále pak je ve scéně transformační skupina apexesTG.



Obrázek 9: Graf scény aplikace

Ta bude sloužit k připojování vrcholů do scény. Jestliže budeme s touto transformační skupinou rotovat, budou se nám otáčet všechny vrcholy a tím pádem bude zajištěna dobrá ovladatelnost celého systému. Příklad připojení aktivního objektu je na obrázku 10. Hlavní metodou třídy Scene je metoda initializeGraph(String file), která celou scénu



Obrázek 10: Graf scény aktivního objektu

vytvoří dokonce s importem souborů, protože uvnitř této metody je volání parseru xml. Vytvořenou scénu vrátí jako instanci třídy Canvas3D.

Metoda move() způsobuje přepočítání pozic všech vrcholů a hran. Tím pádem se celá scéna může měnit.

Metoda getApexes() vrací kolekci všech vrcholů. Tato metoda se používá chceme-li nastavit vlastnosti vrcholů.

4.4.3 Balíček graph.objects

4.4.3.1 Třída Apex Třída Apex je abstraktní a to z toho důvodu, aby bylo možné přidávat do aplikace nové typy vrcholů. Implementuje metody, které jsou pro všechny vrcholy společné a v potomcích je pak už jen třeba nastavit vzhled vrcholu. Třída implementuje mnoho metod typu get, set, které zde popisovat nebudu, jsou však popsány v JavaDocu.

Metoda setModel() je metodou abstraktní, tudíž ji musí implementovat každý potomek a tím nastaví svůj vzhled ve scéně.

Metoda changePower(Vector3D vector) slouží k změně výsledné síly, která na vrchol působí.

Metoda calculatePower(ArrayList apexes) vypočítá výslednou sílu, která na vrchol působí.

Metoda move() posune vrchol na novou pozici podle spočítané výsledné síly.

4.4.3.2 Třída Edge Tato třída reprezentuje hranu grafu mezi dvěma vrcholy. Udržuje si reference na dva vrcholy, aby se dalo určit, kde hranu zobrazit a také jaký směr má mít. Obsahuje jedinou důležitou metodu kromě metod get a set.

Metoda move() nastaví správnou pozici a směr hrany ve scéně

4.4.4 Balíček graph.objects.apexes

Tento balíček obsahuje všechny potomky třídy Apex. Jsou zde tedy pouze reprezentace vrcholů. Tito potomci implementují metodu setModel() a tím určí vzhled vrcholu ve scéně. Příklad jak vypadá graf scény aktivního objektu je na obrázku 10. Objekty mohou využít tříd Java3D k tomu, aby například importovaly konkrétní 3D objekt z externího souboru. K tomu slouží třída ObjectFile, která je součástí standardního balíku Java3D. Při použití této třídy, můžeme importovat například objekt z pokročilého modelovacího programu, jako je třeba 3DStudioMax, Maya, nebo Rhinoceros3D. Já jsem použil program Rhinoceros3D, v kterém jsem objekty, použité ve vrcholech grafu, vytvořil a vyexportoval do souboru .obj, které jsou uloženy v adresáři objects. Při použití této techniky lze ušetřit čas i námahu při tvoření objektů, které chceme v Java3D zobrazit. Na obrázku 10 je dobré si povšimnout složení aktivního objektu z několika součástí. Koule (Sphere), která je přímo navázána na transformační skupinu apexesTG bude tvořit bod, od kterého bude směřovat hrana. Ostatní koncové objekty jsou navázány přes transformační skupiny pro změnu jejich polohy na transformační skupinu billboardTG, která bude zajišťovat, aby byly tyto objekty vždy namířeny přímo na kameru a to z toho důvodu, aby například popisky byly vždy čitelné.

4.4.5 Balíček graph.utils

4.4.5.1 Třída SystemVariables Tato třída obsahuje poze konstanty pro nastavení různých vlastností systému, jako jsou typy objektů a barvy písem.

4.4.5.2 Třída Timer Třída Timer je potomkem třídy Thread. Vytváří se při spuštění aplikace a běží v samostatném vláknu, aby neblokovala aplikaci. Slouží k animaci transformace grafu. Běží v nekonečné smyčce a volá metodu move() třídy Scene, pokud je jeho vlastnost stoped rovna false. Implementuje proto také metody set a get pro nastavení této vlastnost. Implicitně je při startu systému nastavena na true, takže transformace neprobíhá.

4.4.5.3 Třída XmlParser Tato třída implementuje pouze jednu statickou metodu parseApexesAndEdges(String sourceFile, TransformGroup targetGroup, ArrayList edges), která se stará o import dat z XML souboru.

5 Uživatelská příručka

5.1 Instalace a spuštění

Jak již bylo zmíněno, celá aplikace je implementována Java technologií, takže pro její spuštění bude muset být nainstalována Java 2 SDK, standard Edition ve verzi minimálně 1.5.0 a Java3D ve verzi 1.4.0 a vyšší. Oba dva programy se dají volně stáhnout z internetu, ale k dispozici jsou i na přiloženém CD v adresáři install a ve verzi pro operační systém Microsoft Windows. Instalace samotné Javy probíhá pomocí instalátoru ale archív Java3D se musí rozbalit ručně nahrát do již nainstalovaného Java SDK a to do adresáře java_sdk/jre/. Pro běh Java3D je taky nutné mít nainstalovánu podporu OpenGL, nebo DirectX.

Aplikace samotná je na přiloženém CD v adresáři GraphVisualization v podobě Eclipse projektu. Instalovat se nemusí, stačí ji jen spustit a to buď ze zmiňovaného prostředí Eclipse, nebo z příkazové řádky příkazem java graph.Main z adresáře GraphVisualization. Po úspěšném spuštění by se mělo ukázat hlavní okno aplikace.

5.2 Ovládání

Jestliže proběhl start aplikace v pořádku a vidíte hlavní okno programu, můžete s ním začít pracovat. Ve vrchní části je umístěno menu, které slouží pro import souboru, export obrázku a k nápovědě. V levé části okna jsou prvky k nastavení vlastností vrcholů, spolu s tlačítky Nastavit a Play. Tlačítko Nastavit vezme hodnoty z výše umístěných textových políček a přiřadí je vrcholům, takže tímto nastavováním můžeme graf měnit. Tlačítko Play pak způsobí spuštění animace transformace grafu. Je li zmáčknuto, tak se jeho nápis změní na Stop a uživatel může animaci timto tlačítkem zastavit. Spodní řádek je stavový a zobrazuje uživateli informace o stavu aplikace. Ostatní část okna je vyplněna scénou pro visualizaci grafu, kterou můžeme ovládat pomocí myši. Jestliže na scénu klikneme a držíme levé tlačítko myši, bude scéna rotovat. Jestliže použijeme pravé tlačítko, bude se scéna přibližovat, či vzdalovat. Prostřední tlačítko nám umožní posun grafu ve směru os x a y.

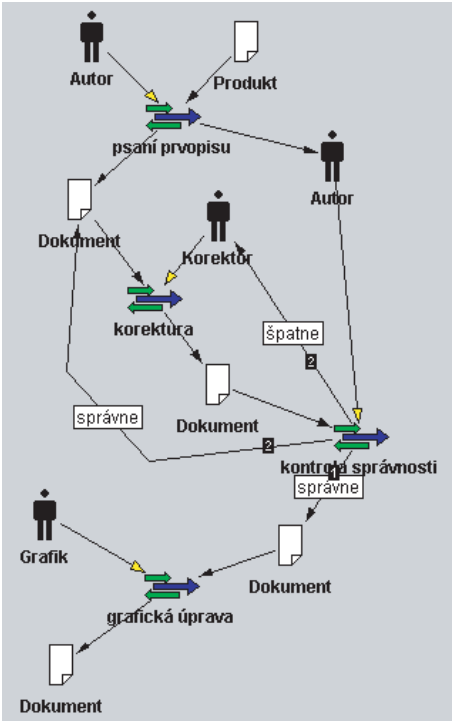
5.3 Demonstrační příklad

Na tomto příkladě ukáži, jak je snadné s aplikaci pracovat a vytvořit tak rychle a snadno graf podnikového procesu.

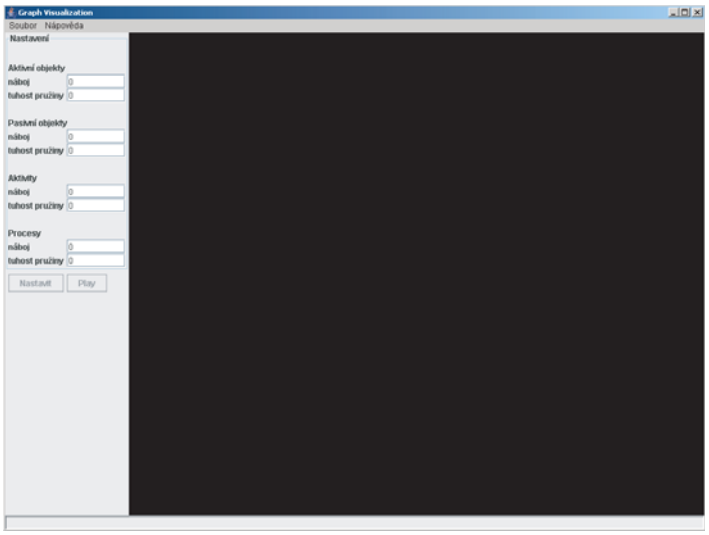
Nejprve vyexportuji z aplikace BPStudio nějaký podnikový proces, resp. koordinační diagram podnikového procesu, který budu chtít zobrazit v mé alikaci. Obrázek 11 znázorňuje proces v aplikaci BPStudio, který se budeme snažit zobrazit v našem programu.

Spustíme si aplikaci GraphVisualization. Po startu by měla vypadat jako na obrázku 12. Vidíme, že graf scény je prázdný a tlačítko Nastavit pro nastavení vlastností vrcholů a tlačítko Play nejdou zatím použít. Nemáme totiž co nastavovat a transformovat.

Proto klikneme na menu Soubor a tam na položku Import souboru. otevře se nám dialogové okno pro otevření souboru viz obrázek 13. Zvolím soubor manual.xml a stisknu tlačítko otevřít. Jestliže se soubor správně importuje, bude ve stavovém řádku

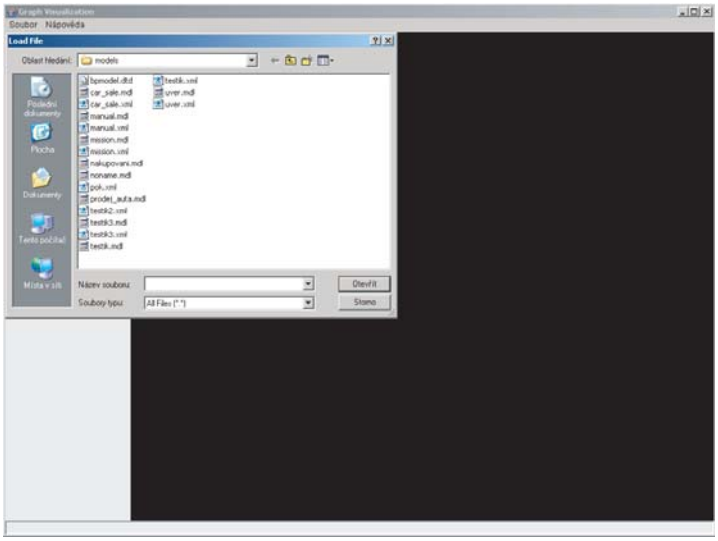


Obrázek 11: Diagram koordinačního modelu v aplikaci BPStudio



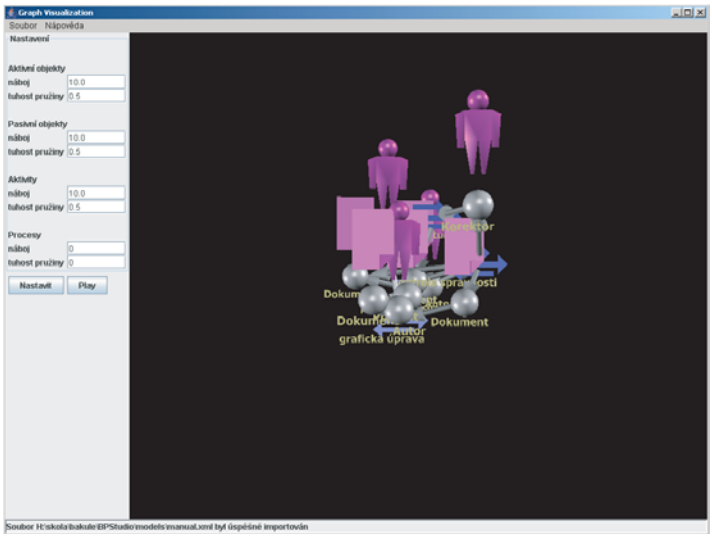
Obrázek 12: Aplikace GraphVisualization po spuštění

správa o úspěšném provedení operace. Jestliže se ale soubor importovat nepodaří, bude ve



Obrázek 13: Zvolení souboru pro import dat

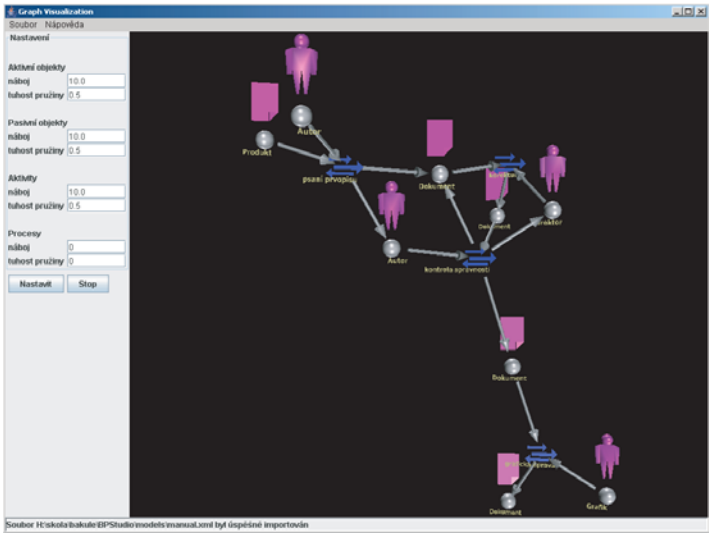
stavovém řádku vypsána chyba importu. Proběhlo-li vše vpořádku, uvidíme už konečně nějaké vrcholy a hrany, ale zatím jen ve shluku uprostřed, protože aplikace při importu umístí vrcholy náhodně kolem středu scény. Obrázek 14 ukazuje, jak by to mohlo vypadat.



Obrázek 14: Aplikace těsně po importu souborů

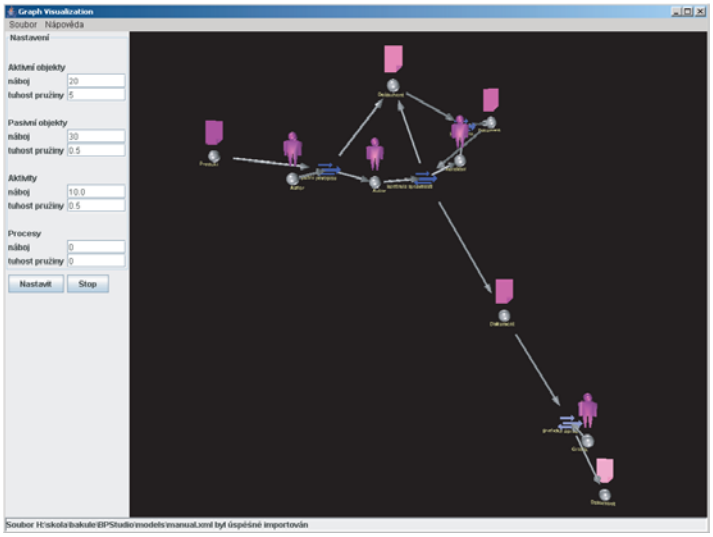
Nyní již můžeme s grafem manipulovat (rotovat, posouvat a zoomovat). To nám ale nebude k ničemu, když jsou vrcholy na sobě namačkány. Proto spustíme námi očekáva-

nou transformaci tlačítkem Play. Scéna se začne animovat a vidíme, jak na sebe vrcholy vzájemě působí. Po určité době se graf ustálí a nebude se dále měnit, výsledek transformace je na obrázku 15



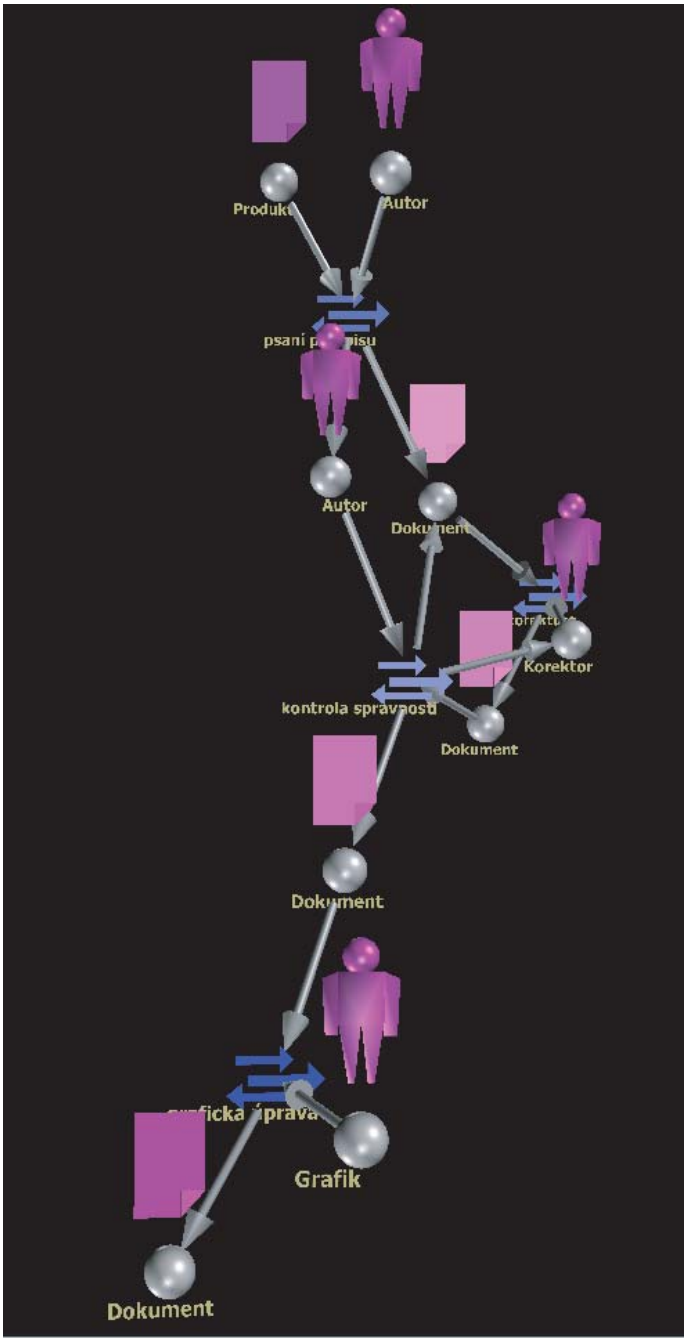
Obrázek 15: Graf po transformaci

Nyní si můžeme začít hrát s nastavováním vlastností vrcholů. Budou-li například vrcholy od sebe hodně vzdáleny můžeme buď snížit hodnotu jejich náboje, aby se méně odpuzovaly a ve výsledku byly blíže k sobě. Nastavování náboje ale ovlivňuje celý



Obrázek 16: Graf po přenastavení vlastností vlcholů

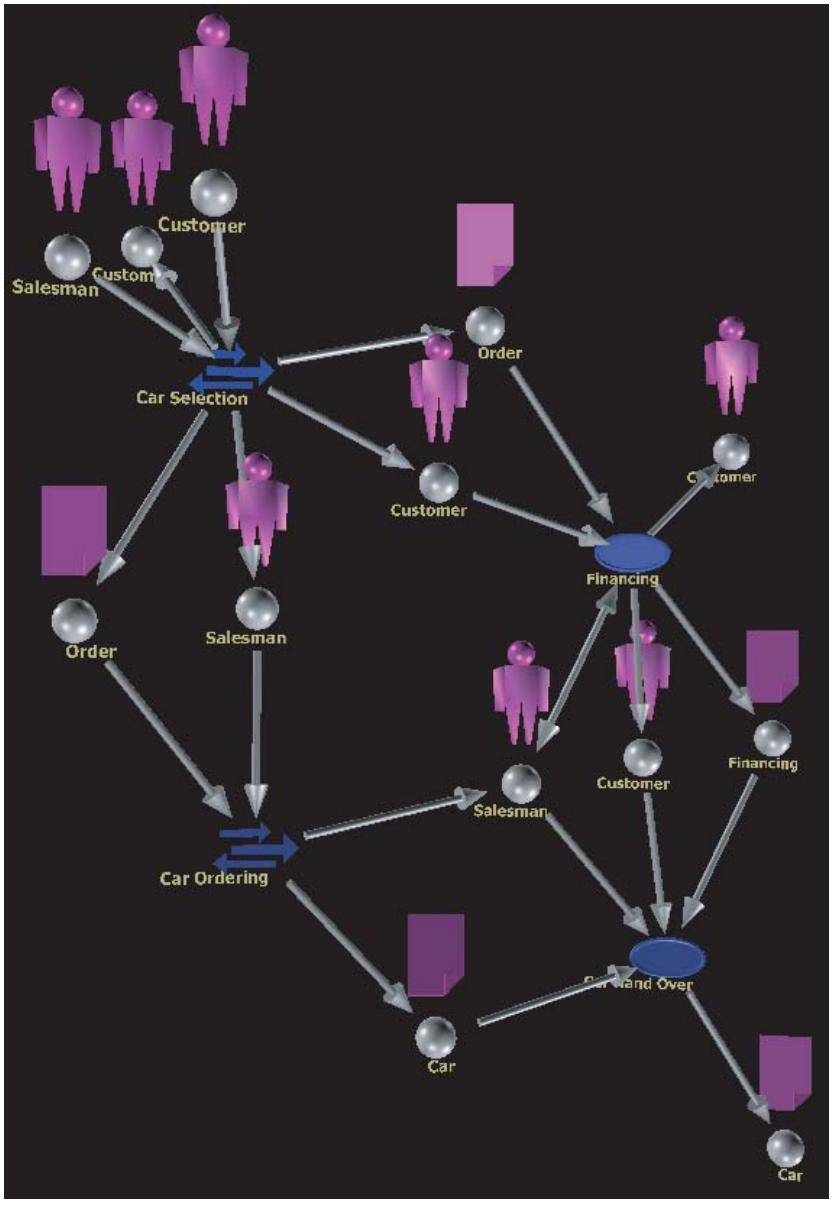
graf, neboť se mezi sebou odpuzují všechny vrcholy. Změníme-li tuhost pružiny, bude to ovlivňovat pouze vrcholy, které mezi sebou mají hranu. Stejný graf po úpravě několika parametrů může vypadat například jako na obrázku 16. Cílem ale je, aby byl graf co



Obrázek 17: Přehledný graf

nejpřehlednější. nastavíme proto ovládací prvky tak, aby jsme mohli graf natočit tak aby tento požadavek splňoval. Na obrázku 17 vidíme, že je tento graf přehledný a v přehlednosti velice konkuruje grafu, který vytvořil člověk viz. obrázek 11. Výsledný obrázek rafu si poté můžeme exportovat v podobě Jpeg obrázku. Kliknutím na menu Soubor a položku Uložit obrázek. Otevře se dialog pro uložení souboru, kde zadáme cestu, vyplníme název souboru s příponou .jpg a klikneme na tlačítko uložit.

Nakonec ještě jedna ukázko trochu složitějšího procesu prodeje auta na obrázku 18.



Obrázek 18: Ukázka vygenerovaného koordinačního diagramu prodeje auta

6 Závěr

Výsledkem této bakalářské práce je aplikace, která dokáže zobrazovat podnikové procesy v 3D scéně. Díky dobré analýze vzhledu koordinačního modelu se povedlo navrhnout a implementovat velice účinný systém na automatické rozmístění prvku ve scéně grafu. Výsledky jsou překvapivě dobré a dá se říci, že i srovnatelné s člověkem vytvořeným diagramem. Je jeasné že úplně bez lidské pomoci to nejde, je třeba si výsledný graf pootočit, aby byl přehlednější. Ale o tu právě jde.

V této aplikaci má člověk mnohem lepší možnosti pro manipulaci s diagramem, než v klasickém 2D zobrazení. Ještě lepší přehlednosti by se dosáhlo při použití efektivnější zobrazovací metody, jakou je například virtuální realita. Kdybychom například mohli jasně rozpoznat, který objekt je blíže a který dál, přehlednost grafu by se ještě zvýšila. Toto by se dalo implementovat i do tohoto programu, ale byl by zapotřebí harware, který by nám scénu zobrazoval a my bychom ji mohli vidět 3D, tak, jak to je například v kinech IMAX.

Výsledkem tedy je, že použitý pružinkový graf s elektrostatickou odpudivou silu funguje celkem dobře a dá se použít nejen k visualizaci podnikových procesů, ale k jakékoliv visualizaci grafu. Například by asi bylo vhodné využití v chemických programech pro zobrazení různych složitých molekul.

V případě pokračování na tomto projektu, bych do budoucna rád implementoval ještě některé věci, které by efektivitu algoritmu ještě zvýšily. Například uvedu alespoň funkce, jako je možnost uchycení některých bodů na pevnou pozici v grafu, nebo možnost posouvání jednotlivými vrcholy.

7 Literatura

[1] Java API: http://java.sun.com/j2se/1.5.0/docs/api/

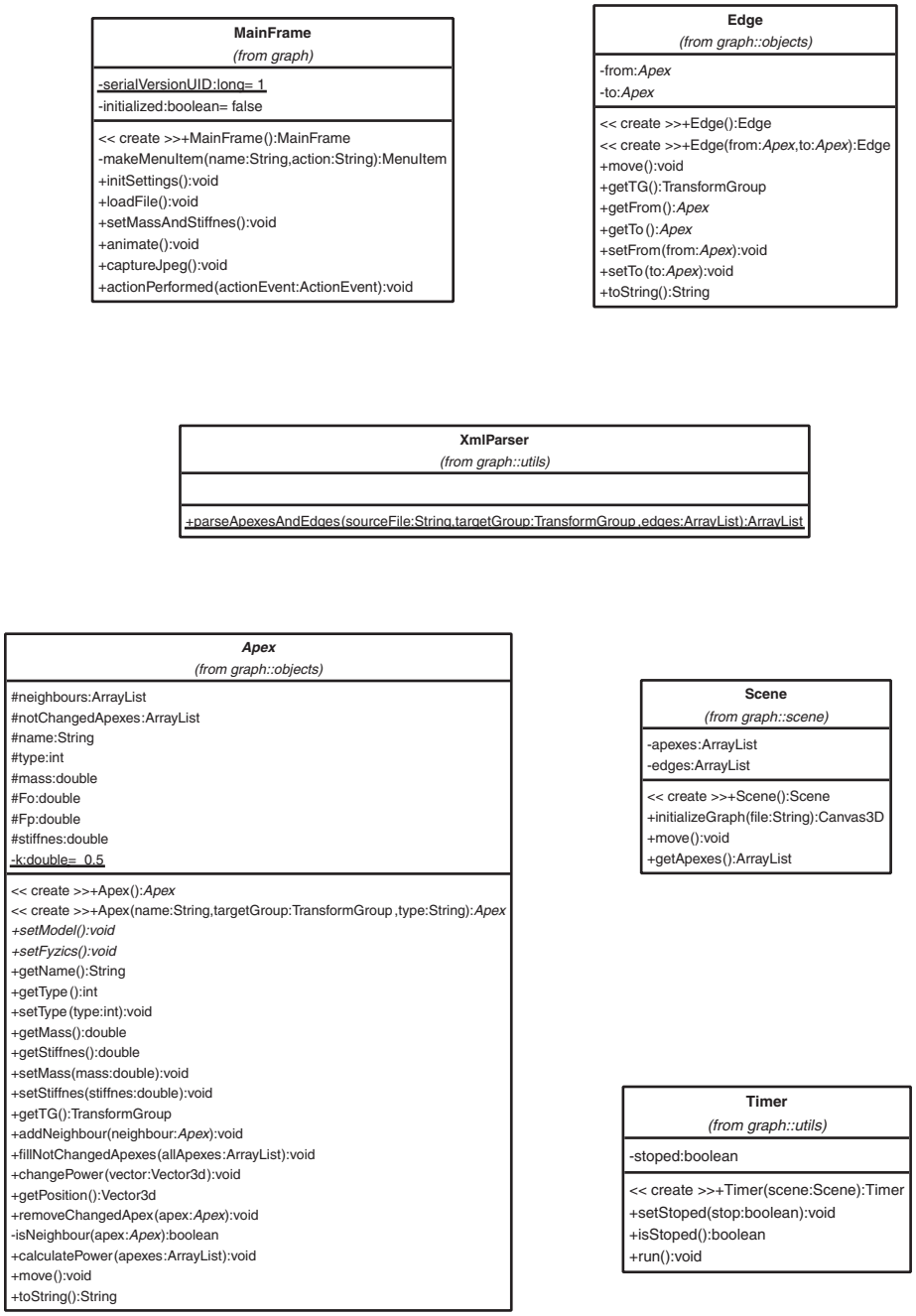
[2] Java 3D API: http://download.java.net/media/java3d/javadoc/1.4.0/index.html

[3] Java 3D API Tutorial: http://java.sun.com/developer/onlineTraining/java3d/

[4] Fedorčák Dušan: Distribuované workflow systémy

[5] Vondrák, Ivo: Skripta k předmětu Softwarové inženýrství

A Podrobný popis tříd



Obrázek 19: Podrobný popis tříd

B Obsah přiloženého CD

Seznam adresářů a popis jejich obsahu:

- **GraphVisualization** - aplikace GraphVisualization se zdrojovými kódy
- **text** - text bakalářské práce
- **text/src** - zdrojové kódy textu bakalářské práce ve formátu LaTeX
- **import** - xml soubory pro import do aplikace
- **install** - instalátor Java technologie a Java 3D