INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Platforma průmyslové spolupráce

CZ.1.07/2.4.00/17.0041

Název

CEP portál pro simulaci

Popis a využití

- komplexní zpracování událostí (CEP)
- aplikace pro spouštění CEP pravidel a sledování výstupů na předpřipraveném toku událostí
- výuka: pokročilá Java

Jazyk textu

- slovenský

Autor (autoři)

- Štefan Repček

Oficiální stránka projektu:

- http://lasaris.fi.muni.cz/pps

Dostupnost výukových materiálů a nástrojů online:

- http://lasaris.fi.muni.cz/pps/study-materials-and-tools

Obsah

Kapitola 1

Úvod

Počítačové systémy, zariadenia a aplikácie po celom svete zaznamenávajú každý deň obrovské množstvo informácii, ktoré môžu mať pre nás veľký potenciálny význam. Avšak je problém také množstvo rôznorodých informácií správne spracovať a vyhodnotiť. Na riešenie tohto problému vznikla nová technológia nazývaná Complex Event Processing (skrátene CEP). CEP nám poskytuje metódy a nástroje, pomocou ktorých môžeme efektívne zaznamenávať a analyzovať informácie, popisujúce aktivity, ktoré sa v systéme udiali. Tieto informácie v CEP označujeme pojmom udalosti. CEP nám umožňuje kontrolovať udalosťami riadené informačné systémy. V dnešnej dobe je skoro každý informačný systém riadený udalosťami [1, s. 1], preto táto technológia našla v praxi široké uplatnenie. Viac je technológia CEP popísaná v kapitole 2.

Cieľom tejto bakalárske práce bolo naimplementovať webovú aplikáciu v jazyku Java, ktorá umožní užívateľovi simulovať beh a spracovávanie reálnych udalostí s použitím technológie CEP na historických dátach. Táto aplikácia umožňuje užívateľovi spúšťať simulovaný prúd udalostí, nasadzovať CEP pravidlá a ukladať výsledky simulácie do databázy tak, aby bolo možné simuláciu kedykoľvek zopakovať. Tieto CEP pravidlá a sady udalostí sú nasadzované na CEP engine, ktorý beží na vzdialenom stroji. Engine následne udalosti a CEP pravidlá spracuje a výsledky simulácie zasiela späť užívateľovi. Zadanie požadovalo, aby komunikácia medzi enginom a webovou aplikáciu neprebiehala lokálne ale po sieti. Hlavným dôvodom tejto požiadavky je to, aby táto aplikácia v budúcnosti mohla fungovať ako distributívny systém. V praxi sa na spracovanie udalostí používa veľa druhov CEP enginov od rôznych firem. Pre túto aplikáciu bol vybraný nástroj Esper[1] od spoločnosti EsperTech. Nástroju Esper sa venuje celá kapitola 3.

Pretože aplikácia má slúžiť ako simulátor pre Complex Event Processing, tak bola nazvaná CEP portál pre simuláciu. Táto aplikácia má široké využitie. Je vhodná pre začiatočníkov, ktorí sa s jej pomocou môžu učiť, aké možnosti

1. `http://esper.codehaus.org/`

CEP ponúka. Taktiež je vhodná pre pokročilejších užívateľov, ktorí pomocou nej môžu testovať a analyzovať svoje CEP riešenia.

Ako už bolo vyššie spomenuté, základné princípy technológie CEP sú popísané v kapitole 2. Táto kapitola je vhodná pre tých, ktorí sa ešte s problematikou CEP nestretli. V kapitole 3 je stručne popísaný nástroj Esper a Esper EPL (Event pattern language). EPL je jazyk, prostredníctvom ktorého sa definujú CEP pravidlá v tomto nástroji. Je dôležité, aby sa užívateľ pred tým ako začne používať túto aplikáciu, oboznámil s týmto nástrojom a hlavne s jazykom EPL. Kapitola 4 sa venuje analýze požiadaviek, návrhu a implementácii aplikácie CEP portál pre simuláciu. Druhá časť tejto kapitoly sa zameriava na použité technológie a tiež na to ako boli implementované v tejto aplikácii. Kapitola 5 sa venuje tomu, ako má byť aplikácia nainštalovaná a používaná. Túto kapitolu môže užívateľ využiť na zoznámenie sa s aplikáciou.

Kapitola 2

Complex event Processing

V tejto kapitole sú stručne charakterizované základné princípy, na ktorých stojí Complex event processing (CEP). Táto kapitola je určená hlavne pre tých, ktorí sa s technológiou CEP ešte nestretli a je im táto oblasť neznáma. Podrobné informácie, komplexnejšia charakteristika technológie CEP a jej využitie v podnikových systémoch sa nachádza v knihe od Davida Luckhama: *The power of events: an introduction to complex event processing in distributed enterprise systems* [1], z ktorej čerpá aj táto bakalárska práca.

2.1 Úvod do Complex event processing

Čo je to vlastne CEP? Jedna z definícií CEP ho definuje takto: CEP je nová sieťová technológia, ktorá zbiera a analyzuje situačné informácie o udalostiach z distribuovaných systémov, databáz a aplikácií v reálnom čase, na základe ktorých sa systém rozhoduje ako reagovať. CEP môže poskytnúť systém, ktorý je schopný definovať, riadiť a predvídať udalosti, situácie, výnimočné podmienky, príležitosti a hrozby v komplexných, heterogénnych sieťach [2].

Zjednodušene sa dá povedať, že CEP slúži na spracovávanie veľkého množstva udalostí v reálnom čase. CEP umožňuje systému komplexne reagovať na množinu udalostí z najnižších vrstiev systému na tej najvyššej úrovni. To znamená, že CEP dokáže z jednoduchých udalostí na najnižšej vrstve (lower-level events) vytvoriť komplexné udalosti (complex events), ktorých význam môže byť závažný až kritický. Komplexná udalosť je vlastne agregáciou nižších udalostí (viď. 2.2). David Luckham vo svojej knihe [1, s. 4], ako príklad vzťahu medzi jednoduchou a komplexnou udalosťou, uvádza prípad preťaženého smerovača v Hongkongu. Táto jednoduchá udalosť v tomto príklade je súčasťou omnoho závažnejšej udalosti – nepodarila sa nám uskutočniť obchodná dohoda s naším partnerom v Tokiu. Z tohto príkladu vyplýva, že CEP umožňuje spracovávať a vyhodnocovať udalosti v širšom kontexte. Veľkým prínosom je, že CEP nám umožňuje pozerať sa na spôsob, ako je systém využívaný a nie na to, ako je postavený. („We need

to be able to view our enterprise systems in terms of how we used them – not in terms how we build them."[1, s. 4]).

Ďalším príkladom komplexnej udalosti a využitia CEP môže byť detekcia útoku *denial of service* (DOS)[1] [1, s. 23]. Jednoduchou udalosťou v tomto prípade bude webová požiadavka (web request) na server. Kým prichádzajú požiadavky v normálnom množstve, tak je všetko v poriadku a server požiadavky obsluhuje. Ako náhle prídu v krátkom časovom rozmedzí milióny požiadaviek, server detekuje možnosť DOS útoku a oznámi to administrátorovi. Ak prísun nových požiadaviek neprestáva, tak server môže začať blokovať tie IP adresy, z ktorých toto obrovské množstvo požiadaviek prichádza. V tomto prípade jedna komplexná udalosť (detekcia DOS útoku) vyvolá ďalšiu komplexnú udalosť (obrana proti DOS útoku).

Tento príklad tiež zdôrazňuje význam času v CEP. Množina jednoduchých udalostí, ktoré sa vyskytnú v určitom časovom intervale (v našom prípade množina požiadaviek na server), môže vytvoriť komplexnú udalosť. Pričom samotná jednoduchá udalosť nemusí mať veľký význam pre systém, avšak význam množiny týchto udalostí v krátkom čase môže byť pre systém kritický (zahltenie servera požiadavkami, nedostupnosť webovej služby).

2.2 Udalosť v CEP

Základným stavebným prvkom, na ktorom stojí CEP, je udalosť. Celý svet okolo nás je riadený udalosťami. Sme nimi obklopení. Pri každodennom používaní slovo udalosť označuje niečo, čo sa stalo. V CEP definujeme udalosť ako záznam aktivity v systéme. („An event is an object that is a record of an activity in a system."[1, str. 88]).David Luckham prisudzuje udalostiam tri hlavné aspekty [1, s. 88]:

- *Forma (Form)*: Udalosť v systéme reprezentujeme ako objekt, tento objekt môže obsahovať príslušne atribúty alebo dátové položky.
- *Význam (Significance)*: Udalosť indikuje, že v systéme prebehla aktivita. Hovoríme že, aktivita je významom udalosti. Udalosť obsahuje dáta, ktoré popisujú jej význam (aktivitu, ktorá sa odohrala v systéme).
- *Relativita (Relativity)*: Každá udalosť sa vzťahuje k ostatným udalostiam na základe ich času, kauzality a agregácie. Vzťah medzi udalosťou a ďalšími udalosťami sa nazýva relativita. Udalosť obsahuje dáta, ktoré tieto vzťahy špecifikujú.

1. DOS je typ útoku na internetové služby, pomocou ktorého chce útočník dosiahnuť nedostupnosť webovej služby pre ostatných užívateľov [4]

Udalosti sa v systémoch vyskytujú v rôznych formách. Napríklad ako objekty tried, XML dokumenty, alebo ako jednoduchý objekt typu `String`. Aby sa udalosť líšila od obyčajnej správy, nestačí aby mala iba svoju formu, musí obsahovať aj dáta, ktoré popisujú jej význam a relativitu.

Tri hlavné vzťahy medzi udalosťami už boli spomenuté vyššie. Sú to čas, kauzalita a agregácia. David Luckham ich charakterizuje takto [1, s. 94]:

- *Čas (Time)*: Je vzťah, ktorý udáva poradie udalostí, v akom sa odohrali. Čas závisí na type hodín, aké systém používa. Takýchto hodín môže mať systém viac.
- *Kauzalita (Causality)*: Ak aktivita, ktorá vytvorila udalosť A, musela prebehnúť, aby prebehla aktivita, ktorá vytvorila udalosť B, tak potom A spôsobilo. Kauzalita, v tomto prípade, je vzťah závislosti medzi aktivitami v systéme. Pretože udalosť popisuje aktivitu, tak môžeme definovať kauzalitu taktiež ako vzťah závislosti medzi udalosťami. Z vyššie uvedených definícií vyplýva, že dve udalosti sú navzájom nezávislé, keď žiadna z nich nespôsobila tú druhú.
- *Agregácia (Aggregation)*: Ak aktivita, ktorá vytvorila udalosť A, je tvorená aktivitami, ktoré vytvorili udalosti, B_1, B_2, B_3, . . . , tak udalosť A je agregáciou všetkých udalostí B. Zvyčajne udalosť A je vytvorená vtedy, keď už nastali všetky udalosti B_n. A je udalosť na vyššej úrovni, udalosti B_n sa nachádzajú na nižšej úrovni. A nazývame komplexnou udalosťou. Jej členovia B_n sú udalosti, ktoré ju spôsobili.

2.3 Complex event patterns

Aby sme boli schopní s komplexnými udalosťami pracovať, tak najprv potrebujeme definovať pravidlá, na základe ktorých budú udalosti spracovávané. K tomu sa používajú ich tzv. „vzory" (complex event patterns). Na definovanie týchto vzorov sa používa jazyk zvaný *event pattern language* (EPL). Vzor je šablóna, ktorá špecifikuje množinu udalostí, ktorú chceme nájsť. Vzor neopisuje len udalosti samotné ale tiež ich príčinné závislosti, časové závislosti, dáta, ktoré obsahujú, a kontext [1, s. 114]. Pri spracovávaní udalostí sa hľadá zhoda s nejakým definovaným vzorom, ak sa zhoda nájde, tak môžeme na ňu vhodne reagovať, napríklad vytvorením novej udalosti.

David Luckham špecifikuje niekoľko mechanizmov, ako zo vzormi pracovať. Prvým dôležitým mechanizmom sú špeciálne reaktívne pravidlá, ktoré pri zhode vyvolajú nejakú akciu. Tieto pravidlá pozostávajú z dvoch častí –

spínač (trigger) a akcia (action). Spínač je vzor udalosti, ktorý spúšťa akciu a akcia je udalosť, ktorá je vytvorená vtedy, keď sa spínač spustí. Luckham ich nazýva *pattern rules* [1, s. 119]. Príkladom reaktívneho pravidla môže byť reakcia na zmenu ceny zlata. Spínač takéhoto pravidla sa nachádza v ukážke 2.1.

```
ChangeGoldPriceEvent(price<1200) WHERE
timer:within(24 hours)
```

Ukážka 2.1: Vzor, ktorý reaguje na zmenu ceny zlata.

Vzor z ukážky 2.1 je zapísaný pomocou EPL jazyka nástroja Esper (pozri 3.3). Tento vzor reprezentuje spínač, ktorý sa spustí, ak cena zlata na burze spadne pod 1200 dolárov počas najbližších 24 hodín. Tento spínač po spustený vyvolá akciu `Predaj`. Celé reaktívne pravidlo môžeme potom popísať takto: Ak cena zlata klesne pod 1200 dolárov počas najbližších 24 hodín, tak predaj všetko zlato, čo vlastním. Vyhodnocovanie tohto pravidla je ukázané na príklade, ktorý sa nachádza v ukážke 2.2.

00:00 – ŠTART;
00:40 – `ChangeGoldPriceEvent {price = 1270}`, žiadna zhoda;
03:30 – `ChangeGoldPriceEvent {price = 1250}`, žiadna zhoda;
06:55 – `ChangeGoldPriceEvent {price = 1235}`, žiadna zhoda;
09:45 – `ChangeGoldPriceEvent {price = 1223}`, žiadna zhoda;
16:45 – `ChangeGoldPriceEvent {price = 1198}`, ZHODA;
16:45 – AKCIA `Predaj`;
16:45 – KONIEC;

Ukážka 2.2: Príklad vyhodnocovania reaktívneho pravidla.

Ďalším dôležitým mechanizmom sú *event pattern constraints*. Pojem *constraint* označuje špeciálnu podmienku, ktorá musí byť splnená aktivitami v systéme. Každý *constraint* vracia ako výsledok iba to, či daná aktivita splnila podmienku alebo nie. Pri nesplnení tejto podmienky hovoríme o narušení *constraint*. *Event pattern constraint* je špeciálny vzor označujúci také chovanie, ktoré v aktivite systému nikdy nesmie nastať [3, s. 4]. Takto môžeme zistiť nezvyčajné chovanie v systéme. *Constraints* môžu byť implementované pomocou *event pattern rules* [3, s. 5].

Kapitola 3

Engine Esper

Ako už bolo spomenuté v úvode, na spracovávanie komplexných udalostí a realizáciu CEP riešení sa v praxi používa *Complex event processing engine*. Tento nástroj realizuje výpočty nad udalosťami na základe vopred definovaných CEP pravidiel. Pre potreby tejto bakalárskej práce bol vybraný ako CEP engine nástroj Esper, ktorý bol vyvinutý spoločnosťou EsperTech v jazyku Java ako open-source software pod licenciou GNU GPL (GNU General Public License) [5].

V tejto kapitole je tento nástroj popísaný s dôrazom na informácie, ktoré sú podstatné vo vzťahu ku aplikácii CEP portál pre simuláciu. Ako informačný prameň bola použitá dokumentácia nástroja Esper, konkrétne pre verziu 4.7.0 [6].

3.1 Úvod

Engine Esper pracuje podobne ako databáza, ale namiesto toho, aby ukladal dáta a spúšťal dotazy (queries) na uložených dátach, Esper umožňuje ukladať dotazy a aplikovať ich na dáta (udalosti), ktoré cezeň prúdia. Tieto dáta v engine Esper označujeme ako prúd udalostí. Na rozdiel od databázy, Esper reaguje v reálnom čase (real-time), keď nastanú podmienky, ktoré vyhovujú uloženým dotazom. Obrazne môžeme povedať, že Esper pracuje na princípe prevrátenej databázy.

Esper poskytuje dve možnosti ako obsluhovať udalosti. Sú to vzory pre udalosti (event patterns) a dotazy pre prúd udalostí (event stream queries).

Na zapisovanie dotazov a vzorov sa využíva Esper EPL jazyk (event pattern language). EPL jazyk umožňuje prostredníctvom vzorov zadefinovať, ako má Esper reagovať na výskyt, alebo absenciu určitých udalostí. Vzory tiež umožňujú popísať vzťahy a časovú závislosť medzi udalosťami. Esper EPL bol navrhnutý tak, aby mal podobnú syntax s SQL jazykom, ale namiesto tabuliek používa pohľady (views).

Dotazy pre prúd udalostí poskytujú rôzne agregačné, spájacie a analytické funkcie, ktoré modifikujú prúd udalostí podľa požiadaviek na CEP

aplikáciu. Dotazy takisto umožňujú zadefinovať tzv. „dátové okná" (data windows) nad prúdom udalostí. Dátové okno je špeciálny pohľad na prúd udalostí, ktorý vyberie udalosť z prúdu udalostí na základe nejakého parametra, napr. času (time windows), alebo počtu udalostí (length window). Tieto dotazy rovnako využívajú EPL jazyk.

3.2 Udalosti v engine Esper

Udalosť v engine Esper je nemenný záznam, ktorý sa skladá z vlastností (properties). Vlastnosti sú dátové položky, ktoré obsahujú informácie o udalosti. Vlastnosť môže byť sama o sebe udalosťou, takáto vlastnosť sa nazýva fragment. Na to, aby Esper mohol s udalosťami pracovať, musí programátor najprv zadeklarovať formu, v akej ich bude engine spracovávať. V nástroji Esper reprezentujeme udalosti pomocou Java objektov, ktoré sú uvedené v tabuľke 3.1.

`java.lang.Object`	Ľubovoľný Java POJO (plain-old java object) s metódami *get*, ktoré dodržujú *JavaBean* konvencie.
`java.util.Map`	Udalosti typu `Map` implementujú rozhranie `java.util.Map`, kde každá položka `Map` je jedna hodnota vlastnosti (property value).
`Object[]` (pole objektov)	Udalosti typu `Object[]` sú polia objektov, kde každá položka poľa je jedna hodnota vlastnosti (property).
`org.w3c.dom.Node`	XML DOM (document object model)
`org.apache.axiom.om.OMDocument`	*Streaming API* pre XML (StAX)
Aplikačné triedy	Zásuvná (plug-in) reprezentácia udalostí cez rozširujúce API.

Tabuľka 3.1: Java objekty, pomocou ktorých sa v engine Esper reprezentujú udalosti [6, s. 3].

Všetky tieto reprezentácie udalostí podporujú vnorené a indexované vlastnosti (pozri 3.2.1). Vnáranie vlastností môže byť nekonečné. Všetky udalosti alebo ich ľubovoľné vlastnosti môžu byť použité pri vytvorení nových udalostí. To znamená, že Esper umožňuje prenášať informácie medzi udalosťami.

Aby bol Esper schopný rozpoznať prichádzajúce udalosti, je nutné ich zaregistrovať. Pri registrácii musíme zadať meno udalosti, zvoliť formu (jedna z reprezentácií uvedených vyššie) a špecifikovať aké vlastnosti obsahuje. CEP portál posiela enginu Esper udalosti vo forme XML.

3.2.1 Vlastnosti udalostí

Už vyššie bolo spomenuté, že vlastnosti udalostí obsahujú informácie, ktoré popisujú danú udalosť. Vlastnosti majú definovaný svoj typ, ktorý sa počas behu programu nemôže zmeniť. Vlastnosti môžeme rozdeliť na tieto štyri skupiny [6, s. 4]:

- *Jednoduché (Simple)*: Jednoduchá vlastnosť obsahuje jedinú hodnotu.
  Syntax v EPL: `name`
- *Indexované (indexed)*: Indexované vlastnosti obsahujú usporiadanú kolekciu objektov, ku ktorým môžeme pristupovať pomocou indexu. Objekty musia mať rovnaký typ.
  Syntax v EPL: `name[index]`
- *Mapované (mapped)*: Mapované vlastnosti obsahujú kolekciu objektov typu kľúč – hodnota.
  Syntax v EPL: `name['key']`
- *Vnorené (nested)*: Vnorené vlastnosti sú také vlastnosti, ktoré sa nachádzajú vo vnútri iných vlastností. Na reťazenie vlastností sa v EPL používa operátor bodka.
  Syntax v EPL: `name.nestedname`

Vlastnosti udalostí môžu byť tiež kombinované.
Napríklad – `company.department('financial').room[0]`.

3.3 EPL jazyk

V tejto časti je stručne popísaný jazyk EPL. EPL je veľmi rozsiahly jazyk, ktorý má veľkú vyjadrovaciu silu. Preto sú v tejto podkapitole charakterizované iba jeho základné nosné časti. Podrobný manuál k EPL sa nachádza v dokumentácii nástroja Esper v kapitolách 5 až 13 [6].

Event Processing Language (EPL) je veľmi podobný jazyku SQL, takisto obsahuje klauzuly ako SELECT, FROM, WHERE, GROUP BY, HAVING a ORDER BY, ktorých význam je rovnaký. Avšak EPL jazyk namiesto tabuliek pracuje s prúdmi udalostí (event streams), kde jedna udalosť nahrádza jeden

riadok tabuľky. Pretože udalosti sa tiež skladajú z údajov, môžeme v EPL efektívne používať agregačné, spájacie (join) a zoskupovacie (group) funkcie, ktoré sú odvodené z SQL.

Nie každá klauzula má ten istý význam ako v SQL, napríklad INSERT INTO v EPL slúži na presmerovanie udalostí do ďalších prúdov na následné spracovanie. Pomocou INSERT INTO môžeme taktiež vytvoriť nový prúd udalostí (viac pozri 3.4.1.1). UPDATE slúži na aktualizovanie vlastností daných udalostí.

EPL výrazy zapisujeme v engine Esper pomocou tzv. „*statements*". Môžeme ich rozdeliť na dve skupiny – *EPL statements* a *pattern statements*. *EPL statements* slúžia na zapisovanie EPL dotazov a *pattern statements* slúžia na zapisovanie vzorov.

3.3.1 EPL statements

Pomocou *EPL statements* sa dajú odvodiť a zoskupiť informácie z jedného, alebo viacerých prúdov udalostí a rovnako zlúčiť dva alebo viacero prúdov dokopy. *EPL statements* môžu obsahovať jeden alebo viacero pohľadov (views). Pohľady, podobne ako v SQL, definujú dáta, ktoré majú byť objektom dotazovania. Existuje viacero druhov pohľadov, ktoré sa používajú v *EPL statements*. Najčastejším druhom sú tzv. okná (windows), ktoré boli už spomenuté na začiatku tejto kapitoly. Druhou veľkou skupinou sú štatistické pohľady, ktoré vypočítajú štatistiky z vlastností udalosti. Taktiež je možné pohľady reťaziť. Príklad jednoduchého *EPL statement* je uvedený v ukážke 3.1.

```
SELECT avg(amount) FROM BankTransferEvent
.win:time(30 sec) WHERE bank='CSOB'
```

Ukážka 3.1: Jednoduchý *EPL statement*.

SELECT klauzula v EPL dotazoch špecifikuje vlastnosti alebo udalosti, ktoré chceme vybrať z prúdu udalostí. FROM klauzula špecifikuje meno prúdu udalostí, na ktorý aplikujeme daný SELECT. WHERE definuje kritérium, podľa ktorého sú udalosti vyhľadávané. Udalosť, odovzdaná klauzule SELECT, musí spĺňať toto kritérium. `Win:time` špecifikuje časové okno (time window). Uvedený *EPL Statement* vráti priemernú sumu z bankových transferov, ktoré sa udiali za posledných 30 sekúnd v ČSOB banke.

3.3.1.1 INSERT INTO klauzula

INSERT INTO klauzula je veľmi dôležitou častou EPL dotazov pri spracovávaní komplexných udalostí. Táto klauzula umožňuje z výsledku jedného *EPL statement* vytvoriť nový prúd udalostí. *EPL statement* s INSERT INTO klauzulou je definovaný v ukážke 3.2.

```
INSERT INTO CombinedEvent
SELECT A.customerId AS custId, A.timestamp -
B.timestamp AS latency
FROM EventA.win:time(30 min) A, EventB.win:time(30
min)
B WHERE A.txnId = B.txnId
```

Ukážka 3.2: *EPL statement* s INSERT INTO klauzulou [6, s. 125].

Tento statement vytvorí nový prúd udalostí `CombinedEvent`, ktorý obsahuje udalosti s vlastnosťami `custId` a `latency`. Kľúčové slovo AS sa používa na pomenovanie novej vlastnosti. Tento nový prúd môže byť použitý v ďalších *statements* ako objekt dotazovania.

INSERT INTO taktiež umožňuje spojiť viacero prúdov do jedného prúdu udalostí. Aby táto procedúra bola možná, musia *statements* byť zadefinované následným spôsobom. Prvý *EPL Statement* s touto klauzulou vytvorí nový prúd udalostí a špecifikuje typy udalostí, mená a typy vlastností, ktoré má obsahovať. Ďalšie *statements*, ktoré budú vkladať udalosti do tohto prúdu, musia danú špecifikáciu dodržiavať.

3.3.2 Pattern statements

Ďalšími dôležitými výrazmi EPL jazyka, ktoré stoja za zmienku, sú vzory udalostí. Ako už bolo spomenuté v druhej kapitole, tak vzor sa uplatní vtedy, keď nastane zhoda s podmienkami špecifikovanými v jeho definícii. Táto zhoda môže nastať v závislosti na prichádzajúcich udalostiach alebo uplynulom čase. Definície vzorov zapisujeme v engine Esper pomocou *pattern statements*.

Pattern statements sa skladajú zo vzorových atómov (pattern atoms) a zo vzorových operátov. Atóm je základný stavebný blok pattern statements. Slúži na špecifikovanie hľadaných udalostí. Takisto atóm môže definovať časový interval pri časovo-závislých udalostiach (time-based events). Operátory kombinujú atómy a riadia životný cyklus (lifecycle) celého vzoru.

Príklad jednoduchého *pattern statement* sa nachádza v ukážke 3.3.

```
every (e1=SensorEvent -> e2=SensorEvent(ID=e1.ID))
```

Ukážka 3.3: Jednoduchý *pattern statement*.

Vo vyššie uvedenom príklade operátor `every` definuje, že daný vzor sa má uplatniť pre každú udalosť `SensorEvent`, ktorá spĺňa ďalej uvedenú podmienku. V zátvorkách je uvedený vnorený výraz obsahujúci operátor „nasledovaný" (followed-by), značený symbolom `->`. Vnorený výraz môže sám o sebe tvoriť `pattern statement`. Operátor `->` indikuje, že udalosť naľavo má byť nasledovaná udalosťou napravo, aby sa daný vzor uplatnil. Taktiež je možné, aby sa namiesto udalostí napravo a naľavo vyskytovali vnorené výrazy. `SensorEvent(ID=e1.ID)` je atóm, ktorý špecifikuje udalosť `SensorEvent` s rovnakým ID, ako má udalosť `e1`. Tento konkrétny vzor sa uplatní pre každú udalosť `SensorEvent`, ktorá je nasledovaná ďalšou udalosťou `SensorEvent` s rovnakým ID.

Pattern Statements sa môžu vyskytovať aj v *EPL Statements*. Vzor sa môže v *EPL Statement* nachádzať za klauzulou FROM, a tiež môže byť použitý v kombinácii s klauzulami WHERE, GROUP BY, HAVING a INSERT INTO. Definícia vzoru musí byť uvedená v hranatých zátvorkách a pred ňou musí byť uvedené kľúčové slovo PATTERN. Použitie vzoru v *EPL Statement* môže vyzerať tak ako v ukážke 3.4.

```
SELECT a.custId, sum(a.price + b.price)
FROM PATTERN
[every a=ServiceOrder ->
b=ProductOrder(custId =a.custId)
where timer:within(1 min)].win:time(2 hour)
WHERE a.name in ('Repair', b.name)
GROUP BY a.custId
HAVING sum(a.price + b.price) > 100
```

Ukážka 3.4: Použitie vzoru v `EPL Statement` [6, str. 211].

3.4 Procesný model

Táto časť kapitoly sa stručne venuje procesnému modelu nástroja Esper. Ako už bolo spomenuté v predchádzajúcej podkapitole, chovanie Esperu voči udalostiam popisujú vzory a dotazy. Ako už bolo spomenuté vyššie, v engine Esper ich zapisujeme pomocou *statements*. Aby mohol Esper aplikácii poslať dáta naspäť, musí mať príslušný *statement* nastavený svojho poslucháča (listener) alebo odberateľa (subscriber).

Každý poslucháč musí implementovať rozhranie `UpdateListener`. Poslucháč musí obsahovať jedinú metódu `update`, ktorú si engine zavolá, keď má k dispozícii výsledok. Metóda update obsahuje dva parametre – `newEvents` a `oldEvents`. Oba parametre obsahujú pole objektov `EventBean`. Esper zapíše výsledok každého aplikovaného *statement* do inštancií objektov `EventBean` a predá ich metóde `update`. Jeden objekt `EventBean` reprezentuje jednu udalosť. `EventBean` následne umožňuje dostať sa k jednotlivým vlastnostiam udalostí, ktoré boli súčasťou definovaného pravidla.

Odberateľ sa od poslucháča líši vtom, že nepoužíva objekty typu `EventBean`. Výsledok príslušného *statement* je priamo doručený parametrom metódy `update`. Je nutné, aby sa počet a typy parametrov zhodovali s počtom a typmi stĺpcov daného výsledku. Trieda odberateľa nepotrebuje implementovať žiadne rozhranie. Výhodou je rýchlosť. CEP portál pre simuláciu odberateľov nepoužíva.

Udalosti sú do enginu Esper odovzdávané vo forme prúdu. V engine Esper sa prúd udalostí rozdeľuje na vstupný (insert stream) a výstupný (remove stream). *Insert stream* predstavuje novo prichádzajúce udalosti, ktoré vstupujú do dátového okna alebo agregácie [6, str. 40]. *Remove stream* predstavuje udalosti, ktoré odchádzajú z dátového okna alebo menia agregačné hodnoty [6, str. 40]. Štandardne sú poslucháčom a odberateľom odovzdávané iba udalosti zo vstupného prúdu. Avšak kľúčovými slovami `istream`, `rstream` a `irstream` v *EPL statements* je možné definovať, akého prúdu udalostí sa má daný EPL dotaz týkať, a aký prúd udalostí má byť doručený priradeným poslucháčom a odberateľom. Ako už bolo vyššie uvedené, metóda `update` má parametre `newEvents` a `oldEvents`. Pri použití kľúčového slova `istream` sú do parametra `newEvents` odovzdané udalosti zo vstupného prúdu. Rovnaké chovanie je aj pri neuvedení žiadneho kľúčového slovíčka. Pri použití `rstream` sú do parametra `newEvents` predané udalosti z výstupného prúdu. Parameter `oldEvents` má v oboch prípadoch hodnotu *null*. Tento parameter sa použije až pri použití kľúčového slova `irstream`. Vtedy poslucháč dostane na vstup

udalosti z oboch prúdov, do parametra `newEvents` sú predávané udalosti z vstupného prúdu a do parametra `oldEvents` sú odovzdávané udalosti z výstupného prúdu. Do parametrov metódy `update` je možné odovzdať aj viac udalostí naraz, napr. pri použití tzv. *„batch windows"*.

Nástroj Esper pracuje na základe takéhoto procesného modelu: CEP aplikácia pošle enginu pomocou metódy `sendEvent` rozhrania `EPRuntime` (viac pozri 3.5) určitý počet udalostí. Engine ich spracuje a v závislosti na aplikovaných *statements* odošle odpoveď CEP aplikácii, čo znamená, že si zavolá metódu `update` na každom poslucháčovi príslušných *statements*, ktoré vyhovujú danému prúdu udalostí.

3.5 Esper API

Samotný Esper je tvorený viacerými balíkmi a neposkytuje užívateľovi žiadne grafické rozhranie. Avšak poskytuje niekoľko Java rozhraní, prostredníctvom ktorých môže užívateľ k nemu (v jeho CEP aplikácii) pristupovať.

Najdôležitejším rozhraním je rozhranie `EPServiceProvider`, ktoré reprezentuje inštanciu enginu. Túto inštanciu môžeme dostať pomocou statických metód triedy `EPServiceProviderManager` `getDefaultProvider` a `getProvider`. `EPServiceProvider` nám poskytuje ďalšie dva dôležité rozhrania, sú to `EPAdministrator` a `EPRuntime`.

Prostredníctvom rozhrania `EPAdministrator` môžeme do enginu Esper pridávať *EPL* a *pattern statements*. Poskytuje tiež možnosť Esper konfigurovať prostredníctvom triedy `ConfigurationOperations` a priradiť ku pridaným *statements* poslucháčov. *EPL statement* pridávame pomocou metódy `createEPL` a *pattern statement* pomocou metódy `createPattern`. Obe metódy, ako návratovú hodnotu vracajú inštanciu triedy `EPStatement`, ku ktorej môže byť neskôr pripojený poslucháč alebo odberateľ. Konfigurovať Esper môžeme dvomi spôsobmi. Buď pomocou triedy `Configuration`, ktorú môžeme odovzdať metóde `getDefaultProvider` ako parameter, alebo pomocou triedy `ConfigurationOperations` počas behu aplikácie. Obe triedy nám ponúkajú niekoľko preťažených metód `addEventType` pre každú reprezentáciu udalosti (pozri 3.2), pomocou ktorých môžeme zaregistrovať udalosti, ktoré budeme enginu Esper posielať.

Rozhranie `EPRuntime` slúži najmä na posielanie udalostí do enginu Esper. Ako už bolo spomenuté v podkapitole 3.2, predtým ako začneme posielať udalosti do enginu Esper, musíme ich najprv zaregistrovať. Udalosti do Esperu posielame prostredníctvom metódy `sendEvent`. Je veľmi dôležité,

aby aplikácia nemenila stav inštancie udalostí, ktoré už boli odoslané do enginu. Typická aplikácia pre každú udalosť vytvorí novú inštanciu.

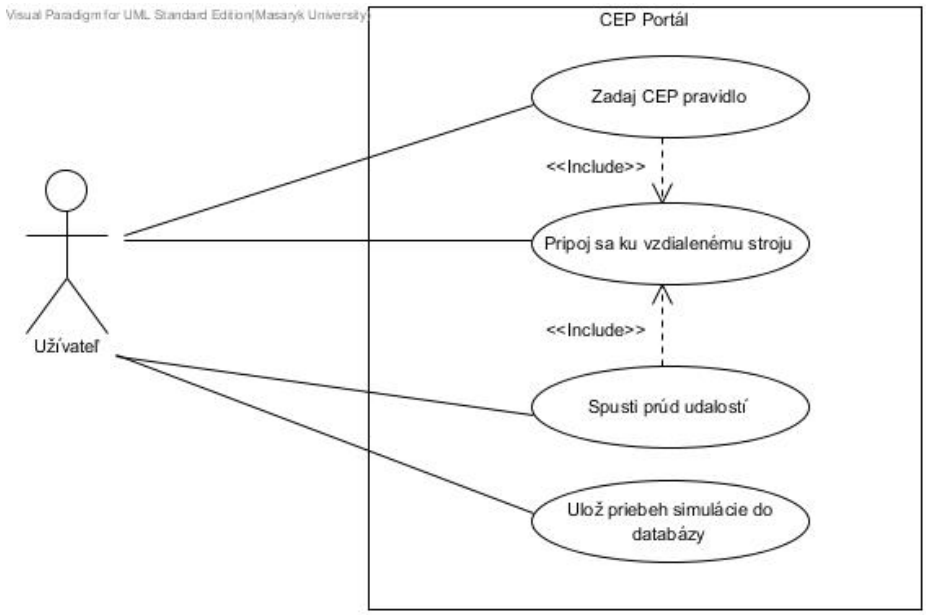
Kapitola 4

Analýza, návrh a implementácia

Aplikácia CEP portál pre simuláciu je naprogramovaná v programovacom jazyku Java. Hlavným dôvodom, prečo bol zvolený tento jazyk, je použitie nástroja Esper, ktorý vyžaduje, aby s ním CEP aplikácia komunikovala prostredníctvom Java objektov (viac pozri kapitola 3). Táto kapitola obsahuje popis, ako prebiehal návrh a implementácia tejto aplikácie. Taktiež sú tu popísané technológie, ktoré boli použité pri implementácii.

4.1 Špecifikácia požiadaviek

Prvou časťou vývoja softwaru býva zvyčajne špecifikácia požiadaviek. Inak to nebolo ani pri vývoji aplikácie CEP portál pre simuláciu. Zadanie tejto práce vyžadovalo vytvorenie webovej aplikácie, ktorá umožní užívateľovi simulovať beh reálnych udalostí na historických dátach a spúšťať na nich CEP pravidlá. CEP pravidlo je spoločné označenie pre vzory a dotazy (pozri 3.1 a 3.3). Hlavnou požiadavkou bolo použitie nástroja Esper ako CEP enginu. Navyše Esper musel bežať na vzdialenom stroji, čiže sa k nemu malo pristupovať cez sieť pomocou IP adresy. Ako už bolo spomenuté, aplikácia mala byť naprogramovaná v jazyku Java a na zostavenie aplikácie sa mal použiť nástroj Maven. Ďalšou požiadavkou bolo vytvorenie databázy, ktorá umožní uložiť výstup aplikácie a entity (pravidlá, udalosti, atď.) použité behom simulácie tak, aby danú simuláciu bolo možné ľahko zopakovať. Funkčné požiadavky na aplikáciu môžeme špecifikovať pomocou diagramu prípadov použitia, ktorý je na obrázku 4.1.



Obr. 4.1: Diagram prípadov použitia

Uvedené prípady použitia boli špecifikované nasledovne:

Pripoj sa ku vzdialenému stroju	
Popis	Užívateľ sa pripojí ku vzdialenému stroju, na ktorom beží engine Esper.
Aktér	Užívateľ
Predpoklad	Na vzdialenom stroji beží engine Esper.
Tok udalostí	1. Užívateľ zadá IP adresu do webového klienta. 2. Webový klient sa skúsi pripojiť ku vzdialenému stroju, kde beží engine Esper. 3. AK sa webový klient úspešne pripojí ku vzdialenému stroju. 3.1. Aplikácia presmeruje užívateľa na hlavnú stránku, pomocou ktorej môže ovládať celú aplikáciu. 4. INAK 4.1. Aplikácia oznámi užívateľovi, že sa jej nepodarilo pripojiť ku vzdialenému stroju.
Následné podmienky	–

Zadaj CEP pravidlo	
Popis	Užívateľ zadá do aplikácie CEP pravidlo zapísané jazykom EPL, ktoré aplikácia predá enginu Esper.
Aktér	Užívateľ
Predpoklad	Na vzdialenom stroji beží engine Esper a webový klient je pripojený ku vzdialenému stroju.
Tok udalostí	1. Užívateľ zadá CEP pravidlo do webového klienta. 2. Webový klient pošle CEP pravidlo enginu Esper. 3. Engine Esper skúsi pravidlo spracovať a uložiť. 4 AK bolo CEP pravidlo úspešne uložené v engine Esper. 4.1. Aplikácia informuje užívateľa, že jeho CEP pravidlo bolo úspešne uložené. 5. INAK 5.1. Aplikácia informuje užívateľa o chybe.
Následné podmienky	–

Spusti prúd udalostí	
Popis	Užívateľ spustí prúd udalostí, ktorý sa následne odošle do enginu Esper. Ten na základe uložených pravidiel (vzorov a dotazov) tento prúd udalostí vyhodnotí a informuje o výsledkoch užívateľa.
Aktér	Užívateľ
Predpoklad	Na vzdialenom stroji beží engine Esper a webový klient je pripojený ku vzdialenému stroju.
Tok udalostí	1. Užívateľ nahrá prúd udalostí do webového klienta. 2. Užívateľ spustí nahraný prúd udalostí. 3.Webový klient odošle udalosti tohto prúdu udalostí do enginu Esper. 4 Engine Esper udalosti spracuje na základe uložených CEP pravidiel. 5. Engine Esper informuje užívateľa o výsledkoch tohto spracovania.
Následné podmienky	–

Ulož priebeh simulácie do databázy	
Popis	Všetky použité entity, ako aj výstup z enginu Esper, si užívateľ môže uložiť do databázy. Užívateľ môže kedykoľvek záznamy v databáze zmeniť alebo zmazať.
Aktér	Užívateľ
Predpoklad	Webový klient je pripojený ku databáze.
Tok udalostí	1. Užívateľ nahrá entitu (CEP pravidlo, prúd udalostí, atď.) do webového klienta. 2. Užívateľ zadá, že chce uložiť entitu do databázy. 3. Aplikácia skúsi entitu uložiť do databázy. 4. AK sa uloženie podarilo. 4.1. Aplikácia aktualizuje zoznam uložených entít. 5. INAK 5.1. Aplikácia informuje užívateľa o chybe.
Následné podmienky	–

Táto aplikácia mala byť vhodná ako pre začiatočníkov, pre ktorých je CEP nový, tak aj pre skúsenejších užívateľov, ktorí sa už v problematike CEP vyznajú. Užívateľ by mal byť schopný prostredníctvom tejto aplikácie vytvárať rôzne experimenty, ktoré mu môžu pomôcť pri vývoji jeho CEP riešení. Výhodou CEP portálu malo byť to, že užívateľ nepotrebuje mať nainštalovaný Esper vo svojom počítači, pretože k aplikácii pristupuje pomocou webového prehliadača. Naviac engine Esper mal bežať na inom stroji ako webová aplikácia. Tým sa zvýši výpočtový výkon celej aplikácie.

4.2 Návrh

Pri návrhu aplikácie bolo dôležité vybrať vhodné technológie pre implementáciu a dôkladne sa zoznámiť s nástrojom Esper. Pretože Esper ponúka užívateľom širokú funkcionalitu, tak bolo dôležité určiť funkcie, ktoré mali byť v aplikácii obsiahnuté. Ďalej bolo potrebné definovať architektúru aplikácie a identifikovať entity, ktoré má obsahovať.

4.2.1 Entity v aplikácii CEP Portál pre simuláciu

Ako je spomenuté vyššie, dôležitou časťou návrhu bola identifikácia entít v aplikácii. Entity museli byť definované tak, aby s nimi mohol pracovať

ako Esper, tak aj užívateľ. Taktiež ich muselo byť možné uložiť do databázy. Entity v aplikácii sú reprezentované Java objektmi. Avšak nebolo nutné pre každú entitu vytvoriť vlastnú triedu. Niektoré entity sú reprezentované pomocou obyčajných objektov typu `String`.

Základnou entitou v aplikácii je udalosť. Avšak tie sú tiež súčasťou ďalšej entity – prúdu udalostí. Pretože Esper požaduje, predtým ako začne pracovať s udalosťami, zaregistrovanie ich typov, tak bolo nevyhnutné vytvoriť entitu reprezentujúcu typ udalosti. Typy udalostí a prúdy udalostí sú v aplikácii reprezentované pomocou objektov typu `EventType` a `EventStream`.

Pri návrhu tejto aplikácie bolo dôležité zvoliť vhodný spôsob, ako bude môcť užívateľ vkladať sady prúdov udalostí do aplikácie. Pretože engine Esper podporuje XML udalosti, tak je na vstupe prúd udalostí odovzdávaný prostredníctvom XML súboru. V aplikácii je reprezentovaný ako XML dokument. Pre jednoduchosť pri prenose, alebo ukladaní do databázy, je tento XML dokument v entite prúdu udalostí uložený pomocou objektu typu `String`. Typ udalosti takisto obsahuje XML súbor, konkrétne XML schému, ktorá popisuje ako má udalosť vyzerať. XML schéma je tiež pre jednoduchosť uložená v entite ako objekt typu `String`. Ďalej typ udalosti musí obsahovať okrem mena aj názov koreňového elementu. Podľa neho Esper rozoznáva o akú udalosť ide.

V tretej kapitole bolo spomenuté, že CEP pravidlá sú v engine Esper reprezentované pomocou *statements*. Táto entita je reprezentovaná v aplikácii prostredníctvom objektu typu `EPLStatement`. Na to aby Esper mohol užívateľovi zaslať odpoveď, bolo nutné vytvoriť entitu poslucháč. Poslucháč je súčasťou entity *statement* a je reprezentovaný pomocou objektu typu `String`. Poslednou entitou v aplikácii je výstup enginu Esper. Táto entita bola vytvorená kvôli tomu, aby bolo možné uložiť výstup do databázy. To poskytuje užívateľovi možnosť simuláciu ľahko zopakovať. Výstup je v aplikácii reprezentovaný objektom typu `CEPOutput`.

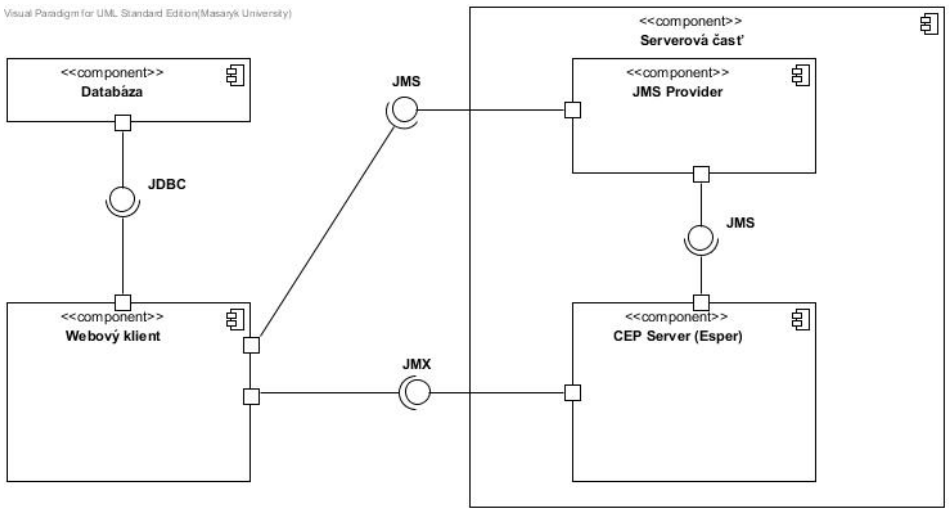
4.2.2 Architektúra aplikácie

Pretože CEP portál mal bežať na dvoch strojoch, tak výsledná aplikácia bola rozdelená na dve časti. Jedna časť slúži ako server, kde beží engine Esper. Druhá časť je webová aplikácia, ktorá slúži ako vstupný bod pre zadávanie CEP pravidiel a spúšťanie prúdov udalostí. Obe aplikácie komunikujú medzi sebou cez sieť a tvoria architektúru klient-server. Webová aplikácia poskytuje užívateľovi grafické rozhranie. Pri návrhu bolo dôležité zvoliť vhodný spôsob, ako budú obe strany spolu komunikovať. Možných riešení bolo viacej. Obe aplikácie mohli komunikovať medzi sebou napríklad

pomocou REST rozhrania alebo RMI (Remote Method Invocation). Nakoniec som zvolil kombináciu JMX (Java Management Extensions) a JMS (Java Message Service) (viac pozri 4.4.2 a 4.4.1).

Riešenie pomocou JMS zabezpečuje prenos udalostí z webovej aplikácie na server pri spustení prúdu udalostí a prenos výstupu z enginu Esper naspäť do webovej aplikácie. Komunikácia pomocou JMS nie je priama, ale sa realizuje prostredníctvom JMS providera. Udalosti sú zasielané najprv JMS providerovi a ten ich následne preposiela serveru. Opačná komunikácia prebieha rovnako. JMX sa v tejto aplikácii používa na konfiguráciu enginu Esper a zadávanie CEP pravidiel. Komunikácia prostredníctvom JMX prebieha priamo a je jednosmerná (iba smerom klient –> server). JMX riešenie pri sieťovej komunikácii využíva RMI, tým umožňuje volať metódy serverovej časti vzdialene.

Ďalším dôležitým komponentom tejto aplikácie je databáza. Tá umožňuje užívateľovi uchovávať dáta použité pri simulácii prostredníctvom webovej aplikácie. Databáza komunikuje s webovou aplikáciu pomocou JDBC (Java Database Connectivity) [8]. Aplikácia nie je závislá na konkrétnom type databázy, pretože používa ORM framework Java Persistance API (JPA) (viac pozri 4.4.3) . Jediná nevyhnutná podmienka je, aby bola použitá relačná databáza. Architektúra celej aplikácie je zachytená diagramom komponentov na obrázku 4.2.



Obr. 4.2: Architektúra aplikácie CEP portál pre simuláciu

4.3 Maven

Nezanedbateľnou súčasťou vývoja aplikácii je preklad a zostavenie zdrojových kódov. Tento proces označujeme anglickým slovom „build". K automatizácii tohto procesu sa používajú rôzne nástroje. Na zostavenie aplikácie CEP portál pre simuláciu bol použitý nástroj Maven.

Maven je obľúbený nástroj pre vývoj aplikácií v Java prostredí. Hlavným cieľom nástroja Maven je urýchliť vývoj a uľahčiť správu projektu. Aby bolo možné dosiahnuť tento cieľ, tvorcovia tohto nástroja definovali niekoľko oblastí, ktoré musí pokrývať. Sú to [9]:

- uľahčenie prekladu a zostavenia projektu
- poskytnutie jednotného systému na preklad a zostavenie projektu
- poskytnutie kvalitných informácií o projekte
- poskytnutie smerníc pre „best practices" vývoj
- umožnenie transparentného pridávania nových funkcií

Pre túto prácu bola podstatná najmä prvá oblasť. Veľkou výhodou pri zostavení aplikácie pomocou nástroja Maven je to, že potrebné knižnice sú stiahnutelné dynamicky z verejne dostupných úložísk, tzv. repozitárov, čiže nie je nutné ich mať uložené v projekte. Projekt musí obsahovať jedine zdrojové kódy a definície závislostí (dependecies) na požadované knižnice. Závislosti sú definované v špeciálnom XML dokumente nazývanom `pom.xml` (Project Object Model). Každý projekt zostavovaný nástrojom Maven musí obsahovať tento dokument [10, s. 5]. `Pom.xml` popisuje nielen závislosti, ale aj celkovú konfiguráciu a vlastnosti projektu (verzia, názov, popis projektu, atď.). Tento dokument sa nachádza v koreňovom adresári projektu. Maven taktiež umožňuje aplikáciu rozdeliť na viacero menších projektov tzv. modulov. Každý modul má potom svoj vlastný `pom.xml` a spolu tvoria stromovú štruktúru, navrchu ktorej je umiestnení rodičovský projekt [10, s. 28]. Ale táto aplikácia túto možnosť nevyužíva.

Aplikácia CEP portál pre simuláciu sa skladá z troch samostatných Maven projektov. Sú to:

- CEPPortal-server: Je serverová aplikácia, ktorá beží na vzdialenom stroji a je v nej integrovaný engine Esper.
- CEPPortal-web: Je webový klient, ktorý slúži ako vstupný bod, pre zadávanie CEP pravidiel, spúšťanie prúdov udalostí. Takisto umožňuje ukladať entity do databázy.

- CEPPortal-jmsconnector: Je externá knižnica, na ktorej sú závislé obidva vyššie uvedené projekty. Táto knižnica obsahuje funkcionalitu potrebnú pre pripojenie ku JMS providerovi a posielanie správ.

Väčšina funkcionality nástroja Maven je realizovaná pomocou pluginov. Zostavenie a preklad programu sa delí na niekoľko fáz. Každá fáza je realizovaná pomocou nejakého základného Maven pluginu (core plugins) [10, s. 27]. Napríklad, keď zavoláme príkaz `mvn compile`, tak Maven zavolá compiler plugin, ktorý skompiluje zdrojové kódy aplikácie. Pluginy sa konfigurujú v `pom.xml`, konkrétne v časti `<plugins>`. Niektoré pluginy môžu rozširovať funkcionalitu nástroja Maven. Ani implementácia aplikácie CEP portál pre simuláciu sa nezaobišla bez využitia funkcionality niektorých pluginov.

Pri zostavení projektu `CEPPortal-server` je použitý *shade plugin* [11]. Tento plugin slúži na zabalenie celej aplikácie do tzv. „uber-jar". Takýto špeciálny jar súbor obsahuje všetky potrebné artefakty (externé knižnice, zdrojové kódy, atď.) na spustenie aplikácie. V tejto práci bol využitý na zabalenie serverovej aplikácie do spustiteľného súboru jar (executable jar), ktorý je možné následne spustiť pomocou jednoduchého príkazu `java -jar`.

Ďalej projekt `CEPPortal-server` využíva *exec plugin* [12], ktorý umožňuje užívateľovi spustiť serverovú aplikáciu bez vytvorenia spustiteľného súboru jar. Jedine je nutné aplikáciu predtým zostaviť. Aplikácia sa následne spúšťa pomocou príkazu `mvn exec:java`.

4.4 Využité technológie a ich implementácia do aplikácie

Ako už bolo viackrát spomenuté, aplikácia je naimplementovaná v programovacom jazyku Java. Konkrétne je postavená na platforme Java EE 6 (Java Enterprise edition) a frameworku Spring. Niektoré použité technológie boli už spomenuté v kapitole 4.2. V tejto podkapitole je uvedená ich stručná charakteristika a dôvody, prečo boli použité. Taktiež je tu priblížená ich implementácia v rámci aplikácie CEP portál pre simuláciu.

4.4.1 Java Message Service

Java Message Service (JMS) je aplikačné rozhranie pre posielanie správ medzi dvoma, alebo viacerými klientmi. JMS API umožňuje aplikáciám vytvárať, posielať, prijímať a čítať správy [13]. Nato aby mohli dve strany spolu komunikovať prostredníctvom JMS, tak musí byť prítomný JMS provi-

der. JMS provider je systém, ktorý implementuje JMS rozhrania a riadi celú komunikáciu. V CEP portále je ako JMS provider prednastavený ActiveMQ.

JMS sa v aplikácii používa na prenos prúdu udalostí z webového klienta do enginu Esper, a tiež na prenos odpovedí z enginu späť do webovej aplikácie. Základnou jednotkou JMS je správa. Udalosti a správy majú podobné vlastnosti. Napríklad udalosť je, podobne ako správa po odoslaní, nemenný záznam. Z toho vyplýva, že udalosti je možné jednoducho reprezentovať pomocou správ. To bol hlavný dôvod, prečo na odosielanie udalostí bola vybraná technológia JMS.

Základom implementácie JMS v aplikácii je knižnica `CEPPortal-jms-connector`, ktorá obsahuje triedy `JMSContext` a `JMSContextFactory`. Pomocou statickej metódy `createContext` triedy `JMSContextFactory` sa vytvorí pripojenie k JMS providerovi a všetky potrebné informácie sa uložia do atribútov inštancie triedy `JMSContext`. JMS funkcionalita je potom v aplikácii realizovaná prostredníctvom tejto inštancie. Každá strana aplikácie obsahuje práve jednu.

V JMS je odosielanie správ realizované pomocou producentov (objekt typu `MessageProducer`) a prijímanie správ je realizované pomocou konzumentov (objekt typu `MessageConsumer`). Na prichádzajúce správy môžeme reagovať pomocou poslucháčov. Každý poslucháč musí implementovať rozhranie `MessageListener`, ktoré predpisuje metódu `onMessage`. Aby bolo možné poslucháča použiť, tak musí byť priradený nejakému konzumentovi [13].

Webový klient obsahuje producenta, ktorý vytvára správy reprezentujúce udalosti. Každá udalosť je reprezentovaná pomocou jednej textovej správy (inštancie triedy `TextMessage`). Ďalej ešte obsahuje konzumenta, ktorý prijíma odpovede z enginu Esper. Odpoveď je takisto reprezentovaná pomocou textovej správy. Konzument má nadstaveného poslucháča (inštancia triedy `ResponseMessageListener`), ktorý odpovede predáva do užívateľského rozhrania. Serverová časť aplikácie obsahuje producenta, ktorý zasiela odpovede webovému klientovi a konzumenta s poslucháčom (inštancia triedy `XMLEventMessageListener`), ktorý prijíma udalosti z webovej aplikácie a predáva ich inštancii enginu Esper.

4.4.2 Java Management Extensions

JMX technológia sa primárne používa na manažment a monitoring aplikácii, systémových nástrojov a zariadení. JMX poskytuje užívateľom rozhrania, pomocou ktorých môžu k týmto aplikáciám, alebo iným zdrojom pristupovať vzdialene. Aplikácie sú v JMS riadené pomocou tzv. `Mbeans`. Štandardný

`MBean` je definovaný pomocou Java rozhrania, ktoré sa musí volať „nejakýNázovMBean", a Java triedy, ktorá sa musí volať „nejakýNázov" a musí implementovať dané rozhranie [14]. Užívateľ potom aplikáciu môže ovládať prostredníctvom metód tohto rozhrania.

Ako už bolo spomenuté JMX sa v aplikácii využíva na zadávanie CEP pravidiel do enginu Esper a jeho konfiguráciu. Serverová aplikácia je takisto JMX server, ku ktorému sa webový klient pripojuje pomocou RMI (Remote Method Invocation). RMI je technológia, ktorá umožňuje vzdialené volanie metód. JMX technológia bola vybraná hlavne kvôli tomu, že v sebe zaobaľuje technológiu RMI. Použitie JMX výrazne zjednodušuje implementáciu, pretože programátor nemusí implementovať celú RMI architektúru. Tá je už implementovaná v JMX. JMX server, v tomto prípade serverová aplikácia, poskytuje klientovi RMI konektor, prostredníctvom ktorého sa pripojí k serveru [15]. JMX server s RMI konektorom je identifikovaný pomocou URL, ktorá je uložená v objekte typu `JMXServiceURL`. Táto URL je nastavená v rámci aplikácie a má takýto tvar:

```
service:jmx:rmi:///jndi/rmi:// 111.111.111.111:9999
/server
```

Kde 111.111.111.111 je IP adresa serveru a 9999 je port, na ktorom je spustený.

Serverová časť aplikácie obsahuje jeden `MBean`, reprezentovaný triedou `EsperServiceProvider` a rozhraním `EsperServiceProviderMbean`. Webový klient realizuje operácie prostredníctvom metód tohto rozhrania. Prostredníctvom tohto `MBean` sú realizované takmer všetky užívateľské operácie, ktoré sú vykonávané na engine Esper. Jedine zasielanie udalostí je realizované pomocou JMS. Webový klient inštanciu vzdialeného `MBean` získa až za behu, a to po pripojení sa k JMX serveru.

4.4.3 Databáza (Java Persistance API a Spring)

Ako už bolo popísané v predchádzajúcich častiach, entity v aplikácii bolo potrebné uložiť do databázy. Databázová vrstva je v aplikácii realizovaná pomocou JPA a Spring Frameworku. JPA je Java framework, ktorý umožňuje objektovo-relačné mapovanie (ORM)[1]. JPA zjednodušuje prístup k relačným databázam. Výhodou použitia JPA je to, že aplikácie nie je závislá na konkrétnom databázovom systéme. Preto si užívateľ môže vybrať, akú databázu použije.

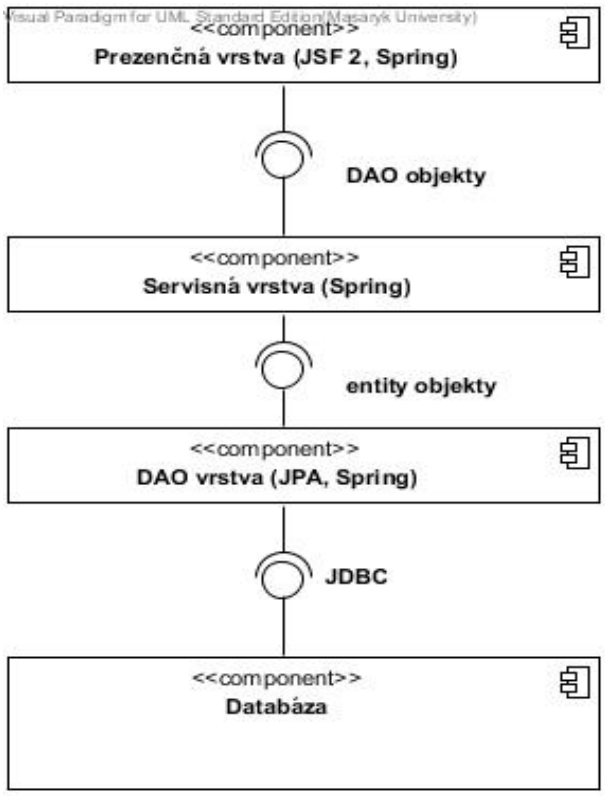
1. Objektovo-relačné mapovanie je programovacia technika, ktorá zaisťuje konverziu dát medzi relačnou databázou a objektovo orientovaným programovacím jazykom [16]

Ďalej aplikácia využíva Spring Framework, ktorý uľahčuje prácu z JPA a zjednodušuje implementáciu. Prostredníctvom Frameworku Spring je realizované vkladanie závislostí (dependency injection) v aplikácii. Vkladanie závislostí je proces poskytnutia externej závislosti nejakého softwarového komponentu, napr. triedy. V Jave to znamená, že namiesto toho aby bola závislosť inicializovaná programátorom v komponente, napríklad pomocou konštruktora, tak je do komponenta vložená kontajnerom zvonka [17, s. 37]. Stačí zadeklarovať anotáciou, do ktorého atribútu má byť závislosť vložená. Vo frameworku Spring realizujeme vkladanie závislostí pomocou anotácie `@autowired`.

Aplikácia sa pripája k databáze pomocou objektu JDBC `DataSource`. O vytvorenie daného `DataSource` sa stará framework Spring. Vlastnosti tohto objektu charakterizujú konkrétnu databázu, ku ktorej sa má aplikácia pripojiť. Tieto vlastnosti sú definované v `applicationContext.xml` (konfiguračný súbor frameworku Spring). Užívateľ prostredníctvom tohto XML súboru môže zadefinovať pripojenie k ľubovoľnému databázovému systému. Webový klient má v `applicationContext.xml` definovaný `DataSource` k HSQLDB databáze[2] a k PostgreSQL databáze[3]. Užívateľ si môže vybrať, ktorú z nich použije, prípadne môže definovať pripojenie k inej databáze. (viac pozri 5.1).

Perzistentná vrstva je v aplikácii reprezentovaná pomocou DAO (data access object) vrstvy. DAO vrstva realizuje databázové operácie a oddeľuje databázu od ostatných častí aplikácie. Nad ňou sa nachádza servisná vrstva, ktorá sa stará iba o výnimky z DAO vrstvy a o manažment transakcií. Obsluha výnimiek a manažment transakcií je realizovaný tiež pomocou frameworku Spring. Pretože na prezenčnej vrstve sú entity reprezentované pomocou DTO (Data transfer object) objektov. Tak na servisnej vrstve prebieha preklad entít na DTO a takisto preklad DTO objektov naspäť na entity. DTO objekty sú obyčajné POJO objekty, ktoré nesú informácie o entite, ale neobsahujú žiadnu biznis logiku. Výhodou DTO je, že prezentačná vrstva nepristupuje priamo k entitám, ktoré sú uložené v databáze, a preto nemôže zmeniť dáta [18]. To znižuje previazanosť medzi prezenčnou vrstvou a ostatkom aplikácie. Všetky triedy, ktoré zabezpečujú prenos dát z databázy na prezenčnú vrstvu, sa nachádzajú v balíku `cz.muni.fi.cepportal.database`. Architektúra webového klienta a prístup k databáze je zobrazený na obrázku 4.3.

2. `http://hsqldb.org/`
3. `http://www.postgresql.org/`



Obr. 4.3: Diagram komponentov znázorňujúci architektúru webového klienta a prístup k databáze

4.4.4 Prezenčná vrstva a užívateľské rozhranie (JSF 2, Spring)

Užívateľ vo webových aplikáciách komunikuje s aplikáciou prostredníctvom užívateľského rozhrania. Užívateľské rozhranie a prezentačná vrstva v tejto aplikácii je naimplementovaná pomocou kombinácie technológie JSF 2 a frameworku Spring.

JSF (JavaServer Faces) je štandardný komponentovo orientovaný webový framework používaný pri implementácii webových aplikácii v jazyku Java. JSF je založený na modele MVC (Model-View-Controller). Užívateľské rozhranie je v JSF 2 tvorené pomocou frameworku Facelets, ktorý nahradil staršiu technológiu JSP (Java Server Pages). Facelets je tvorený XHTML stránkami, ktoré obsahujú štítky (tags) z rôznych knižníc. XHTML stránky sú prepojené so špeciálnymi objektmi serverovej časti webovej aplikácie. Tieto

objekty sa nazývajú `ManagedBeans`. `ManagedBeans` sú obyčajné POJO objekty, prostredníctvom ktorých je webová aplikácia riadená a môžu byť deklarované pomocou XML súboru, alebo anotácií [19].

Webový klient aplikácie CEP portál pre simuláciu obsahuje päť `ManagedBeans`. Najdôležitejší z nich je `ClientManagedBean`. Prostredníctvom tejto triedy sa klient pripojí k vzdialenému stroju, na ktorom beží engine Esper. Táto trieda tiež obsahuje metódy, prostredníctvom ktorých sú odosielané všetky požiadavky zadané užívateľom na engine Esper. Prostredníctvom ostatných `ManagedBeans` sú realizované CRUD operácie s entitami. Každý z nich je pomenovaný podľa toho, s akou entitou pracuje. Sú to tieto štyri triedy:

- `EPLStatementManagedBean`
- `EventTypeManagedBean`
- `EventStreamManagedBean`
- `CEPOutputManagedBean`

`ClientManagedBean` je obyčajný `ManagedBean` deklarovaný pomocou klasickej JSF anotácie `@ManagedBean`. Ale ostatné `ManagedBeans` nie sú deklarované pomocou JSF anotácií, pretože sú implementované pomocou frameworku Spring (verzia 3). Aby bolo možné kombinovať framework Spring 3 a technológiu JSF 2, musia byť tieto triedy deklarované pomocou anotácie `@Named` a vo `faces-config.xml` musí byť deklarovaný `SpringBeanFacesELResolver` [20]. Spring sa taktiež, ako na servisnej vrstve, stará o vkladanie závislostí (inštancií servisných tried) v týchto triedach.

Kapitola 5

Inštalácia a používanie

V tejto kapitole je popísaná inštalácia a nasadenie aplikácie CEP Portál pre simuláciu. Ďalej sa v nej nachádza stručný popis toho, ako sa táto aplikácia používa, čo musia spĺňať vstupné dáta a aké možnosti užívateľovi aplikácia poskytuje. Táto kapitola môže poslúžiť ako stručný užívateľský manuál.

5.1 Inštalácia a spustenie aplikácie

Ako už bolo viackrát v tejto práci uvedené, aplikácia pozostáva z dvoch častí – webový klient a serverová aplikácia. Aby bolo možné celú aplikáciu používať, tak je nutné nainštalovať obe časti aplikácie. Predtým sa musia zostaviť projekty oboch častí aplikácie (`CEP Portal-server` a `CEP Portal-web`). Aby bolo možné projekty zostaviť, je nutné mať nainštalovaný prekladač jazyka Java a nástroj Maven.

Prvým krokom inštalácie je nasadenie serverovej aplikácie na server. Pretože časť komunikácie medzi klientom a serverom prebieha prostredníctvom JMS, tak pred spustením serverovej aplikácie je potrebné spustiť na serverovom stroji JMS provider. Serverová aplikácia a JMS provider musia bežať na jednom stroji pod rovnakou IP adresou. Toto obmedzenie sa nachádza v aplikácii kvôli tomu, aby užívateľ vo webovom klientovi nemusel nastavovať dve rôzne IP adresy, jednu pre JMS provider a jednu pre serverovú aplikáciu. Aplikácia CEP portál pre simuláciu má predvolený JMS provider Apache ActiveMQ. Avšak užívateľ môže použiť aj iný JMS provider. Konfigurácia nutná na pripojenie ku JMS provideru sa nachádza v property súbore `cepserver_config.properties`. Tento súbor je umiestený v oboch aplikáciách v zložke `/src/main/resources` a po zostavení aplikácie sa nachádza v adresári `/target/classes`. Súbor tiež obsahuje port na ktorom je spustený JMS provider a port na ktorom má byť spustený RMI register. ActiveMQ beží na porte 61616 a RMI register má prednastavený port 1199. JMS provider ActiveMQ sa jednoducho spúšťa z jeho inštalačného adresára pomocou súboru `activemq`, ktorý sa nachádza v podadresári `bin`.

Keď už JMS provider beží, tak je možné spustiť serverovú aplikáciu. To sa dá urobiť dvoma spôsobmi. Najjednoduchšie je použiť spustiteľný jar súbor, ktorý je vytvorený pri zostavení serverovej aplikácie v zložke `/target`. V tomto súbore sú zabalené všetky zdrojové súbory a externé knižnice potrebné na spustenie tejto aplikácie.

Ďalej je možné aplikáciu spustiť zo zložky projektu pomocou *exec pluginu*. Serverovú aplikáciu spúšťame vždy pomocou konzolového príkazu.

Ak použijeme prvý spôsob, tak príkaz vyzerá nasledovne:

```
java -jar -Djava.rmi.server.hostname=999.999.999.999
CEPPortal-cepserver-1.0.jar
```

Kde `999.999.999.999` je IP adresa stroja, na ktorom beží serverová aplikácia. Premenná `java.rmi.server.hostname` je JVM vlastnosť, ktorá reprezentuje IP adresu, pomocou ktorej môže klient volať metódy vzdialeného objektu (`MBean`). Štandardne je táto premenná nastavená na *localhost* [7]. Avšak pri tomto nastavení nie je možné, aby webový klient pristupoval k serveru zvonka. Preto ak obe časti aplikácie CEP portál pre simuláciu bežia na rovnakom stroji, tak konzolový príkaz nemusí obsahovať túto premennú.

Pomocou *exec pluginu* spúšťame serverovú aplikáciu takto:

```
mvn -Djava.rmi.server.hostname=147.251.48.1 exec:java
```

Po spustení aplikáciu môžeme jednoducho ukončiť pomocou konzolového príkazu `exit`. Je nutné podotknúť, že po ukončení serverovej aplikácie, sa všetky dáta uložené v engine Esper stratia.

Aby bolo možné spustiť webového klienta, tak sa musí predtým nasadiť na nejaký webový server. Webové aplikácie sa nasadzujú na server prostredníctvom súboru war. Tento súbor sa po zostavení projektu `CEP Portal-web` nachádza v zložke `/target`. Užívateľ môže nasadiť webovú aplikáciu na ľubovoľný webový server, ale predtým je dôležité webový server správne nakonfigurovať. Pretože táto aplikácia používa niektoré nové technológie, ktoré nemusia byť kompatibilné zo staršími verziami webových serverov (Tomcat 6 atď.), tak je doporučené nasadzovať aplikáciu na najnovšie verzie.

Aplikácia ako prednastavený webový server používa Apache Tomcat vo verzii 7.0.40. Webový klient je možné nasadiť na Apache Tomcat prostredníctvom Tomcat manažéra, ktorý sa po spustení serveru Tomcat pri

nezmenenej konfigurácii nachádza na adrese `http://localhost:8080/manager/html`.

Ešte pred nasadením webového klienta, je potrebné mať správne definované pripojenie k databáze. Ako už bolo spomenuté v predchádzajúcej kapitole, pripojenie k databáze sa definuje v súbore `applicationContext.xml`. Tento XML súbor sa nachádza v adresári `WEB-INF`. Pripojenie k databáze sa v ňom definuje pomocou elementu *bean* s ID `dataSource`. Tento element musí obsahovať URL, meno JDBC ovládača (driver), užívateľské meno a heslo, pomocou ktorých sa pripojí k databáze. Príklad definície pripojenia k databáze je uvedený v ukážke 5.1.

```
<bean id='dataSource' class='org.springframework.jdbc
.datasource.DriverManagerDataSource'>
<property name='url'
value='jdbc:postgresql://localhost:5432/cepportal'/>
<property name='password' value='admin'/>
<property name='driverClassName'
value='org.postgresql.Driver'/>
<property name='username' value='postgres'/>
</bean>
```

Ukážka 5.1: Definícia pripojenia k PostgreSQL databáze.

Tento príklad, ktorý definuje pripojenie k PostgreSQL databáze, bol zobraný z tejto aplikácie. Aby sa užívateľ mohol pripojiť k tejto databáze pod touto konfiguráciou, tak ju musí nasadiť na rovnaký stroj ako webovú aplikáciu. Ďalej musí byť spustená na porte s číslom 5432, musí sa volať cepportal. Prihlasovacie meno musí byť „postgres" a heslo musí byť „admin". Okrem pripojenia k PostgreSQL databáze je ešte v `ApplicationContext.xml` definované pripojenie k HSQLDB databáze. Výhodou tejto možnosti je, že užívateľ pri použití tejto databázy nemusí nič inštalovať a konfigurovať. Databáza sa vytvorí automaticky pri prvom spustení aplikácie na disku na adrese `/cepportal`. Užívateľ môže umiestnenie HSQLDB databázy na disku zmeniť. Užívateľ nemusí použiť jednu z vyššie uvedených možností. Môže definovať pripojenie k ľubovoľnej databáze. Avšak je dôležité, aby v `applicationContext.xml` bolo aktívne vždy iba jedno pripojenie (jeden element s ID `dataSource`). Ostatné musí užívateľ zakomentovať alebo zmazať.

5.2 Používanie aplikácie

Užívateľ pracuje s aplikáciou prostredníctvom webového klienta cez webový prehliadač. Po úspešnom spustení sa na adrese `http://WebClientIP:Port/cepportal-web/` zobrazí uvítacia stránka. V strede tejto stránky sa nachádza textové pole, kde užívateľ musí zadať IP adresu servera, na ktorom beží engine Esper. Po zadaní IP adresy sa užívateľ pomocou tlačidla `Start Application` pripojí k serveru. Ak sa klientovi podarí pripojiť k serveru, tak aplikácia presmeruje užívateľa na hlavnú stránku webového klienta.

5.2.1 Hlavná stránka aplikácie

Pomocou hlavnej stránky užívateľ môže ovládať celú aplikáciu CEP Portál pre simuláciu. Hlavná stránka je rozdelené na viacero častí, kde každá časť má inú funkciu. Na ľavej strane hlavnej stránky sa nachádzajú formuláre prostredníctvom, ktorých užívateľ zasiela požiadavky na engine Esper a ukladá entity do databázy. Vpravo hore sa nachádza okno, v ktorom je zobrazená databáza. Pod ním je umiestené okno, kde sa užívateľovi zobrazuje výstup z aplikácie. Rozloženie hlavnej stránky je zobrazené na obrázku 5.1.

Najvyššie je umiestený formulár na odosielanie *EPL statements*. Tento formulár obsahuje dve textového polia. Do prvého užívateľ musí zadať meno daného *EPL statement*. Tento údaj je povinný a pri ukladaní do databázy, musí byť meno unikátne. Do druhého poľa užívateľ zadá konkrétny EPL výraz. Po vyplnení týchto údajov, môže užívateľ odoslať *EPL statement* do enginu Esper pomocou tlačidla `Create EPL`, ktoré sa nachádza v pravom dolnom rohu formulára. Po odoslaní engine Esper užívateľovi oznámi, či sa mu podarilo daný *statement* vytvoriť, alebo došlo k nejakej chybe. V ľavom dolnom rohu sú umiestnené tlačidlá, ktoré umožňujú uložiť *EPL statement* do databázy, alebo ho zmeniť, ak je už uložený. Užívateľ môže daný *statement* odoslať tiež ako *pattern statement* (rozdiel medzi *EPL statement* a *pattern statement* pozri 3.4.1 a 3.4.3). Stačí, aby zaškrtol zaškrtávacie políčko (checkbox) `isPattern`, ktoré sa nachádza v pravom dolnom rohu formulára pri tlačidle `Create EPL`.

Hlavná stránka taktiež umožňuje užívateľovi pridať poslucháčov ku svojim *statements*. Užívateľ vytvorí poslucháča kliknutím na tlačidlo `Listener`, ktoré sa nachádza v spodnej časti formulára na odosielanie *EPL Statements*. Toto tlačidlo presmeruje užívateľa na ďalšiu stránku, na ktorej je zobrazený poslucháč príslušného *statement* (viac pozri 5.2.3).

Pod formulárom na odosielanie *EPL statements* sa nachádza formulár, ktorý slúži na odstránenie *statements* z enginu Esper. Užívateľ musí zadať

meno príslušného *statement*, ktorého chce odstrániť a pomocou tlačidla `Destroy EPL` odošle požiadavku na engine Esper. Ak daný *statement* existuje, tak ho engine Esper odstráni. Tento formulár taktiež obsahuje tlačidlo, pomocou ktorého je možné odstrániť všetky pravidlá nasadené v engine Esper.

Prostredníctvom ďalšieho formulára môže užívateľ zaregistrovať typy udalostí do enginu Esper a uložiť ich do databázy. Vyššie bolo spomenuté, že udalosti sú v aplikácii reprezentované pomocou XML, preto typ udalosti musí obsahovať meno udalosti, meno koreňového elementu udalosti a XML schému. Schéma definuje štruktúru udalosti a podľa mena koreňového elementu si engine Esper vie priradiť udalosť k zaregistrovanému typu udalosti. Meno udalosti a meno koreňového elementu užívateľ zadáva do dvoch textových polí tohto formulára. Meno udalosti je unikátna hodnota, ak už typ udalosti s rovnakým menom v engine Esper existuje, tak registrácia bude neúspešná. To isté platí aj pri ukladaní tejto entity do databázy.

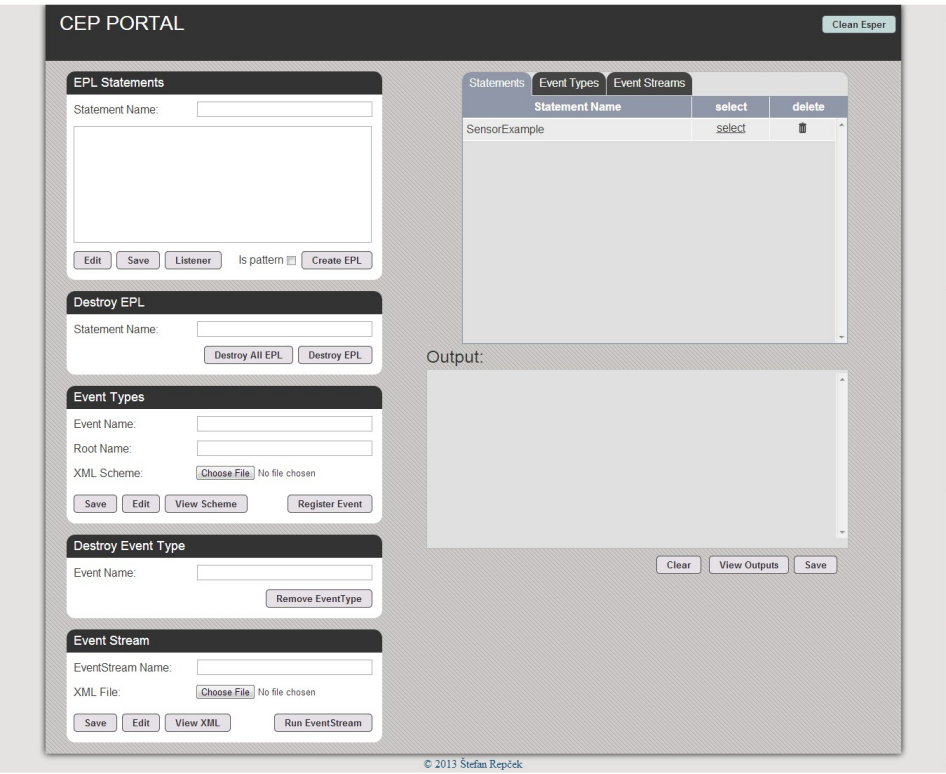
Pod nimi sa nachádza tlačidlo, pomocou ktorého je možné načítať XML schému z disku užívateľa. Načítanú schému si užívateľ môže pozrieť kliknutím na tlačidlo `View Scheme`. Všetky tieto hodnoty sú pre typ udalosti povinné. Ak užívateľ jednu z nich nevyplní, tak ho na to aplikácia upozorní. Po vyplnení týchto údajov, užívateľ môže typ udalosti zaregistrovať pomocou tlačidla, ktoré sa nachádza v pravom dolnom rohu formulára, alebo ho môže uložiť do databázy. Tlačidlá na uloženie, alebo zmenenie typu udalosti v databáze, sa podobne, ako v predchádzajúcom formulári nachádzajú v ľavom dolnom rohu.

Aplikácia taktiež umožňuje odobrať typ udalosti z enginu Esper. Túto funkciu poskytuje formulár, ktorý sa nachádza hneď pod formulárom na registráciu typov udalostí. Užívateľ by mal pristupovať k odoberaniu typov udalostí opatrne, pretože pri vykonávaní tejto požiadavky engine Esper zruší všetky aktívne *statements*, ktoré sa vzťahovali k danému typu udalosti.

Najnižšie sa nachádza formulár na spúšťanie prúdov udalostí. Rozloženie prvkov tohto formulára je rovnaké ako vo formulári na registráciu typov udalostí. Podobne ako v predchádzajúcich prípadoch, užívateľ má možnosť uložiť prúd udalostí do databázy. Užívateľ si musí pri ukladaní dávať pozor na to (ako v predchádzajúcich prípadoch), že meno prúdu udalostí je unikátne. Prúd udalostí vkladá užívateľ rovnako ako XML schému pomocou XML súboru (viac pozri 5.2.4). Po načítaní je možné zobraziť tento XML dokument, kliknutím na tlačidlo `View XML`. Prúd udalostí sa v tomto formulári spúšťa pomocou tlačidla `Run EventStream`.

Ako už bolo uvedené, vpravo hore sa nachádza okno, v ktorom sú zobrazené entity uložené v databáze. Toto okno obsahuje tri záložky –

`Statements`, `Event Types`, `Event Streams`. Každá záložka má priradenú jednu tabuľku z databázy. Užívateľ môže jednoducho prejsť z jednej tabuľky do druhej, kliknutím na jednu zo záložiek. Na konci každého riadku tabuľky sú tlačidlá `select` a `delete`, pomocou ktorých môže užívateľ vybrať alebo vymazať záznam z databázy. Po vybratí jedného záznamu z databázy sa entita, ktorú vybraný záznam obsahoval, zobrazí v jednom z vyššie uvedených formulárov, kde ju užívateľ môže upraviť alebo poslať do enginu Esper.



Obr. 5.1: Hlavná stránka webového klienta

5.2.2 Výstup aplikácie

Na to, aby užívateľ mal predstavu, čo sa deje v engine Esper, serverová aplikácia generuje výstup, ktorý následne odovzdáva webovému klientovi. Serverová aplikácia vždy po spracovaní požiadavky užívateľovi na výstup

zašle zrozumiteľnú odpoveď, či požiadavka bola spracovaná úspešne, alebo nastala nejaká chyba. Výstup z enginu Esper webový klient užívateľovi zobrazuje vo formulári `output`, ktorý sa nachádza na hlavnej stránke na pravej strane.

Výstup je kľúčový pri spracovávaní prúdu udalostí. Aby užívateľ mal informácie o tom ako bol jeho prúd udalostí na základe definovaných CEP pravidiel spracovaný, musí ku svojim *statements* priradiť poslucháčov. Engine Esper následne pri uplatnení nejakého pravidla, pošle prostredníctvom poslucháča užívateľovi odpoveď. To ako bude odpoveď vyzerať, necháva aplikácia na užívateľovi (viac pozri 5.2.3).

Aby bolo jednoduché zopakovať simuláciu, tak aplikácia umožňuje užívateľovi uložiť výstup do databázy. Po skončení simulácie môže výstup uložiť, pomocou tlačidla `Save`, ktoré sa nachádza pod formulárom `output`. Po stlačení tohto tlačidla, sa zobrazí okno, do ktorého užívateľ musí zadať názov ukladaného výstupu. Uložené výstupy si užívateľ môže pozrieť kliknutím na tlačidlo `View Outputs`. Aplikácia ho presmeruje na stránku `viewoutputs`, kde má možnosť výstupy vymazať, alebo zmeniť meno, pod ktorým sú uložené. Zoznam výstupov, uložených v databáze, sa na tejto stránke nachádza na pravej strane. Užívateľ môže vybrať požadovaný výstup, kliknutím na jeho meno v zozname. Následne sa obsah vybraného výstupu zobrazí vo vedľajšom okne.

5.2.3 Pridanie poslucháča

V predchádzajúcej časti bolo spomenuté, že poslucháči v aplikácii produkujú textový výstup, ktorý je počas spracovávania udalostí predávaný webovému klientovi. Aj keď nástroj Esper umožňuje, aby jeden *statement* mal viacero poslucháčov, tak v tejto aplikácii môže mať každý *statement* iba jedného poslucháča. Viac poslucháčov pre jeden *statement* by v tejto aplikácii nemalo význam, pretože poslucháči okrem generovania textového výstupu nevykonávajú žiadnu inú funkciu. Štruktúra poslucháča v engine Esper je popísaná v 3.4.

Poslucháč je v aplikácii tvorený dvojicami reťazcov. Prvý reťazec z tejto dvojice môže obsahovať ľubovoľný text, ktorý chce užívateľ vypísať pri vyhodnotení príslušného *statement*. Druhý reťazec obsahuje meno vlastnosti, alebo výraz vlastnosti (property expression)[1], pomocou ktorého môže užívateľ vytiahnuť hodnotu požadovanej vlastnosti z výsledku príslušného *statement* (inštancie `EventBean`). Túto hodnotu následne poslucháč predá na výstup. Spolu tieto dva reťazce vytvoria jeden riadok výstupu. Pričom

1. výraz vlastnosti je spoločný pojem pre indexované, mapované a vnorené vlasnosti [6, s. 4]

prvý z nich väčšinou popisuje hodnotu, ktorú užívateľ dostane pomocou druhého reťazca. Poslucháč môže obsahovať neobmedzený počet týchto dvojíc reťazcov. Takisto je možné, aby jeden z dvojice reťazcov obsahoval prázdny reťazec. V aplikácii prvý reťazec označujeme ako `Label` a druhý reťazec ako `Expression`. Každý `Label` a `Expression` je v poslucháčovi priradený buď k inštanciám objektov `EventBean`, ktoré sa nachádzajú v parametri `newEvents`, alebo k tým, ktoré sa nachádzajú v parametri `oldEvents`. Tak užívateľ môže reagovať na udalosti zo vstupného a aj výstupného prúdu (pozri 3.4).

Na pridanie poslucháča ku nejakému statement slúži v aplikácii stránka `listener`. Ako už bolo uvedené v 5.2.1, tak na túto stránku sa užívateľ dostane kliknutím na tlačidlo `Listener` vo formulári na odosielanie *EPL statements*. Na stránke `listener` užívateľ môže pridávať a upravovať tieto dvojice prostredníctvom textových polí. Na ľavej strane sa nachádza pole pre `Label` a na pravej strane pole pre `Expression`. Pretože poslucháč môže obsahovať ľubovoľný počet dvojíc, tak sú textové polia na stránke vytvárané dynamicky. Užívateľ môže pridať novú dvojicu pomocou tlačidla `Add Property`.

Aplikácia následne na stránke vytvorí dva nové polia pre novú dvojicu reťazcov. Dvojicu môže užívateľ odobrať pomocou tlačidla `Remove Property`. Na priradenie reťazca k jednému z parametrov poslucháča (`newEvents` alebo `oldEvents`) sa na stránke používa výberové menu, ktoré sa nachádza pod každým textovým poľom. Pretože výraz definovaný v reťazci `Expression` sa vždy aplikuje na nejakú inštanciu objektu `EventBean` a parametre poslucháča môžu obsahovať viac týchto inštancií, tak užívateľ môže definovať pomocou indexu, na ktorú inštanciu sa má daný výraz vzťahovať. Na to na stránke slúži textové pole `index`, ktoré sa nachádza pod každým textovým poľom definujúcim `Expression`. Štandardne sa výraz vlastnosti aplikuje na inštanciu s indexom 0.

5.2.4 Vkladanie prúdov udalostí do aplikácie

Viackrát bolo spomenuté, že sady prúdov udalostí sa do aplikácie vkladajú pomocou XML súborov. XML dokument, ktorý je obsiahnutý v tomto súbore, musí mať určitú formu, ktorá je v aplikácii definovaná pomocou jednoduchej XML schémy. Schéma predpisuje aby koreňový element tohto súboru mal meno „`events`". Jedna udalosť je tvorená jedným XML elementom, ktorý je podradený rodičovskému elementu `events`. Tento element je zároveň koreňovým elementom danej udalosti a môže obsahovať ďalších potomkov, ktoré popisujú vlastnosti udalosti. Koreňový element v tomto

XML dokumente môže obsahovať ľubovoľný počet udalostí s ľubovoľným názvom. Okrem udalostí môže užívateľ v tomto dokumente špecifikovať aj časový údaj, ktorý určuje časové rozmedzie medzi udalosťami. Tento časový údaj je v tomto dokumente definovaný pomocou jednoduchého elementu `time` v milisekundách. Tieto elementy užívateľ musí vložiť medzi jednotlivé udalosti.

Po spustení prúdu udalostí aplikácia najprv XML dokument rozloží na jednotlivé udalosti a postupne ich odošle do enginu Esper v takom poradí, ako boli definované v tomto dokumente. Pri tomto odosielaní aplikácia na základe časových údajov obsiahnutých v elementoch `time`, vytvára časové medzery medzi odoslaním jednotlivých udalostí do enginu Esper. Ak časový údaj medzi udalosťami nie je uvedený, tak aplikácia udalosť odošle okamžite. Užívateľ nesmie zabudnúť na to, že pred spustením prúdu udalostí sa musia zaregistrovať všetky typy udalostí, ktoré daný prúd obsahuje. XML dokument definujúci jednoduchý prúd udalostí môže vyzerať tak ako v ukážke 5.2.

```
<?xml version= '1.0' encoding='UTF-8' ?>
<events>
<Sensor>
<ID>urn:epc:1:4.16.26</ID>
<Observation Command='READ_PALLET_TAGS_ONLY'>
<ID>00000001</ID>
<Tag><ID>urn:epc:1:2.24.403</ID></Tag>
</Observation>
</Sensor>
<time>7000</time>
<Sensor>
<ID>urn:epc:1:4.16.36</ID>
<Observation Command='READ_PALLET_TAGS_ONLY'>
<ID>00000001</ID>
<Tag><ID>urn:epc:1:2.24.400</ID></Tag>
<Tag><ID>urn:epc:1:2.24.405</ID></Tag>
</Observation>
</Sensor>
</events>
```

Ukážka 5.2: XML dokument obsahujúci prúd udalostí.

Prúd udalostí z ukážky 5.2 obsahuje iba dve udalosti špecifikované ich koreňovým elementom `Sensor`. Element `time` v tomto dokumente špecifikuje, že medzi odoslaním týchto dvoch udalostí bude 7 sekundová medzera. Ostatné elementy, ktoré sa nachádzajú v elementoch `Sensor`, popisujú vlastnosti tejto udalosti.

Kapitola 6

Záver

V úvode bolo spomenuté, že cieľom tejto bakalárske práce bolo vyvinúť webovú aplikáciu, ktorá bude simulovať beh a spracovávanie reálnych udalostí prostredníctvom CEP. Aplikácia bola v priebehu práce úspešne naimplementovaná a otestovaná. Kľúčová požiadavka, aby sa ku enginu Esper pristupovalo vzdialene, bola splnená.

Aplikácia bola naimplementovaná vo vývojovom prostredí NetBeans[1]. UML diagramy, vypracované pri analýze a návrhu, boli vytvorené v programe VisualParadigm[2]. Táto aplikácia využíva množstvo rôznych technológií. Pri ich implementácii bol kladený dôraz na to, aby tieto techológie spolu správne spolupracovali. Toto bolo dôležité najmä pri integrácii frameworku Spring 3 a technológie JSF 2. Najnáročnejšou častou celého vývoja bol návrh a implementácia vzdialeného prístupu ku enginu Esper. Pôvodným zámerom bolo použiť technológiu RMI. Od tohto zámeru sa neskôr upustilo a nakoniec technológia RMI bola nahradená technológiou JMX, kvôli zjednodušeniu implementácie.

Výsledkom tejto práce je aplikácia, ktorá môže mať široké uplatnenie v akademickej, a tiež komerčnej sfére. Ako už bolo viackrát uvedené v tejto práci, aplikácia je vhodná pre užívateľov, ktorí majú záujem pochopiť problematiku CEP. Pre študentov môže slúžiť ako nástroj, ktorý im umožní reálne vyskúšať, prácu s technológiou CEP. V komerčnej sfére táto aplikácia môže slúžiť ako testovací nástroj pre CEP riešenia. Užívateľ si pomocou nej môže namodelovať určitú situáciu, ktorá môže v systéme nastať a následne otestovať svoje CEP riešenie. Veľkou výhodou tejto aplikácie je to, že umožňuje uložiť výsledok simulácie a všetky entity, ktoré boli pri nej použité. Tým umožní užívateľovi svoje CEP riešenia otestovať veľmi rýchlo viackrát na rôznych dátach, alebo s rôznymi zmenami v CEP pravidlách. Výsledkom toho je urýchlenie vývoja.

---

1. `https://netbeans.org/`
2. `http://www.visual-paradigm.com/`

Už pri zadaní tejto práce bolo plánované, že táto aplikácia prejde neskôr ďalším vývojom, ktorého výsledkom bude distributívny systém pre simuláciu CEP. Preto bola aplikácia naimplementovaná tak, že k enginu Esper sa pristupuje vzdialene. Aktuálna verzia tejto aplikácie je naimplementovaná tak, že ku vzdialenému stroju, na ktorom beží engine Esper, môže pristupovať iba jeden webový klient. Neskôr by malo byť možné, aby k serveru pristupovalo viac klientov naraz. Okrem tohto plánovaného rozvoja je možné aplikáciu rozšíriť aj iným smerom. Pri ďalšom vývoji by bolo vhodné naiplementovať podporu pre ďalšie formáty vstupu napr. pre CSV súbory, pridať do webovej aplikácie ďalšie možnosti konfigurácie enginu Esper alebo vylepšiť grafické rozhranie a dizajn aplikácie. Ďalší vývoj tejto aplikácie môže byť predmetom niektorej bakalárskej alebo diplomovej práce.

Literatúra

[1] LUCKHAM, David. *The power of events: an introduction to complex event processing in distributed enterprise systems.* Boston: Addison-Wesley, c2002. ISBN 0201727897.

[2] BASS, Tim. *What is Complex Event Processing?* [online]. 2007 [cit. 2013-04-18]. Dostupné z WWW: <http://www.thecepblog.com/2007/05/14/what-is-complex-event-processing-part-1/>

[3] LUCKHAM, David. *A Brief Overview of the Concepts of CEP* [online]. 2007 [cit. 2013-05-15]. Dostupné z WWW: <http://complexevents.com/wp-content/uploads/2008/07/overview-of-concepts-of-cep.pdf>

[4] MCDOWELL, Mindi. *Understanding Denial-of-Service Attacks* [online]. 2013 [cit. 2013-04-20]. Dostupné z WWW: <http://www.us-cert.gov/ncas/tips/ST04-015>

[5] EsperTech Inc. *Esper - License* [online]. 2013 [cit. 2013-04-22]. Dostupné z WWW: <http://esper.codehaus.org/about/license/license.html>

[6] EsperTech Inc. *Esper Refence* [online]. 2012 [cit. 2013-04-22]. Dostupné z WWW: <http://esper.codehaus.org/esper-4.7.0/doc/reference/en-US/pdf/esper_reference.pdf>

[7] Oracle. *Java SE Documentation* [online]. 2013 [cit. 2013-05-05]. Dostupné z WWW: <http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/javarmiproperties.html>

[8] Oracle. *JDBC Overview* [online]. 2010 [cit. 2013-05-06]. Dostupné z WWW: <http://www.oracle.com/technetwork/java/overview-141217.html>

[9] The Apache Software Foundation. *What is Maven* [online]. 2013 [cit. 2013-05-07]. Dostupné z WWW: <http://maven.apache.org/what-is-maven.html>

[10] SRIRANGAN. *Apache Maven 3 cookbook: over 50 recipes towards optimal Java software engineering with Maven 3.* Boston: Addison-Wesley, c2002. Birmingham: Packt Open Source, 2011. ISBN 9781849512459 (e-book).

[11] The Apache Software Foundation. *Maven Shade Plugin* [online]. 2012 [cit. 2013-05-07]. Dostupné z WWW: <http://maven.apache.org/plugins/maven-shade-plugin/>

[12] Codehaus. *Exec Maven Plugin* [online]. 2012 [cit. 2013-05-07]. Dostupné z WWW: <http://mojo.codehaus.org/exec-maven-plugin/>

[13] Oracle. *Overview of the JMS API* [online]. 2013 [cit. 2013-05-09]. Dostupné z WWW: <http://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html>

[14] Oracle. *Standard MBeans* [online]. 2013 [cit. 2013-05-09]. Dostupné z WWW: <http://docs.oracle.com/javase/tutorial/jmx/mbeans/standard.htmll>

[15] Oracle. *Lesson: Remote Management* [online]. 2013 [cit. 2013-05-07]. Dostupné z WWW: <http://docs.oracle.com/javase/tutorial/jmx/remote/index.html>

[16] LIPITSAINEN, Arvo. *ORM – Object Relational Mapping* [online]. [cit. 2013-05-12]. Dostupné z WWW: <http://www.dbtechnet.org/labs/dae_lab/Orm.pdf>

[17] Bien, Adam. *Real world Java EE patterns: rethinking best practices. S. l.* Press.adam-bien.com, 2009. ISBN 978-0-557-07832-5.

[18] FOWLER, Martin. *Data transfer object* [online]. [cit. 2013-05-13]. Dostupné z WWW: <http://thierryroussel.free.fr/java/books/martinfowler/www.martinfowler.com/isa/dataTransferObject.html>

[19] BURNS, Ed; SCHALK, Chris. *JavaServer Faces 2.0: The Complete Reference* [online]. The McGraw-Hill Companies, 2010. ISBN: 978-0-07-162510-4

[20] SMITH, Greg. *JSF 2 with Spring 3 and JSR-330* [online]. 2010 [cit. 2013-05-10]. Dostupné z WWW: <http://comdynamics.net/blog/89/jsf2-with-spring3-and-jsr-330/>

Obsah priloženého CD

Priložené CD obsahuje:

- inštalačný balíček, ktorý obsahuje:
  - JMS Provider ActiveMQ
  - predkonfigurovaný Tomcat 7
  - spustiteľný jar súbor serverovej aplikácie
  - war súbor webového klienta
  - README
- Maven projekty so zdrojovými kódmi aplikácie
- príklady simulácií
- túto prácu vo formáte PDF