VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GRAFICKÝ EDITOR PRO KONFIGURACI STRUKTURY ÚLOH V RÁMCI JAVA EE BATCHING API

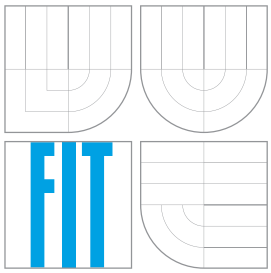
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ HANUS

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GRAFICKÝ EDITOR PRO KONFIGURACI STRUKTURY ÚLOH V RÁMCI JAVA EE BATCHING API

GRAPHICAL EDITOR FOR JOB STRUCTURE CONFIGURATION IN BATCHING API

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ HANUS
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZBYNĚK KŘIVKA, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá popisem implementace grafického editoru pro konfiguraci struktury úloh v rámci Java EE Batching API v podobě zásuvného modulu pro vývojové prostředí IntelliJ IDEA. Zaměřuje se hlavně na možnost obousměrné propagace změn mezi textovou a grafickou reprezentaci jazyka JSL. Přibližuje problematiku dávkového zpracování. Zároveň je zde představena i architektura prostředí IDEA. Stručně popisuje návrh zásuvného modulu a postupy využité při jeho implementaci. Použitelnost nástroje je demonstrována pomocí pěti příkladů dávkových úloh. V závěru jsou zmíněny některé vylepšení, na které by bylo možné se v budoucnu zaměřit. Výsledný nástroj je volně dostupný jako otevřený software.

Abstract

This bachelor's thesis describes implementation of a graphical editor for a job structure configuration in Java EE Batching API in the form of a plugin for integrated development enviroment IntelliJ IDEA. It focuses mainly to allow a bi-directional propagation of changes between both graphical and text representation of JSL language. This document introduces the characteristic of batch processing and also the architecture of IntelliJ IDEA. It also briefly describes the design of the plugin and some of the procedures used in its implementation. The usability of this tool is demonstrated by five examples of batch jobs. In the end, there are mentioned some possibilities for the future improvements. The tool is available under Open Source license.

Klíčová slova

IntelliJ IDEA, zásuvný modul, dávkové zpracování, Java EE 7, grafický editor, obousměrná propagace změn, jazyk JSL

Keywords

IntelliJ IDEA, plugin, batch processing, Java EE 7, graphical editor, bi-directional propagation of changes, JSL language

Citace

Tomáš Hanus: Grafický editor pro konfiguraci struktury úloh
v rámci Java EE Batching API, bakalářská práce, Brno, FIT VUT v Brně, 2015

Grafický editor pro konfiguraci struktury úloh v rámci Java EE Batching API

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Hanus
18. května 2015

Poděkování

Chcel by som sa poďakovať svojmu vedúcemu práce pánovi Ing. Zbyňkovi Křivkovi Ph.D. za jeho odborný prístup, ochotu a cenné rady pri písaní práce, rovnako ako pri tvorbe samotného zásuvného modulu. Ďalej by som rád poďakoval pánovi Ing. Jiřímu Pechancovi z firmy Red Hat za jeho ochotu pri komunikácií a rovnako cenné rady.

Obsah

Kapitola 1

Úvod

Dávkové spracovanie je vykonávanie viacerých úloh, ktoré sú vykonávané sériovo, paralelne, poprípade kombináciou oboch princípov. Dávkové úlohy sú také typy úloh, ktoré nevyžadujú užívateľskú interakciu, teda žiadny zásah užívateľa, nanajvýš počiatočné spustenie. Je pre ne typická dlhšia doba behu, ako aj väčší objem spracovaných dát. Z dôvodu väčšieho zaťaženia systému sa spravidla vykonávajú len pár krát za deň alebo ako reakcia na určitú udalosť.

1.1 Motivácia

Táto práca sa zaoberá popisom problematiky, návrhu a implementácie nástroja, ktorý slúži ako grafický editor pre konfiguráciu štruktúry úloh v rámci Java EE Batching API dostupného od špecifikácie Java 7. Nástroj je určený pre vývojárov a je implementovaný formou zásuvného modulu pre vývojové prostredie (IDE[1]) IntelliJ IDEA.

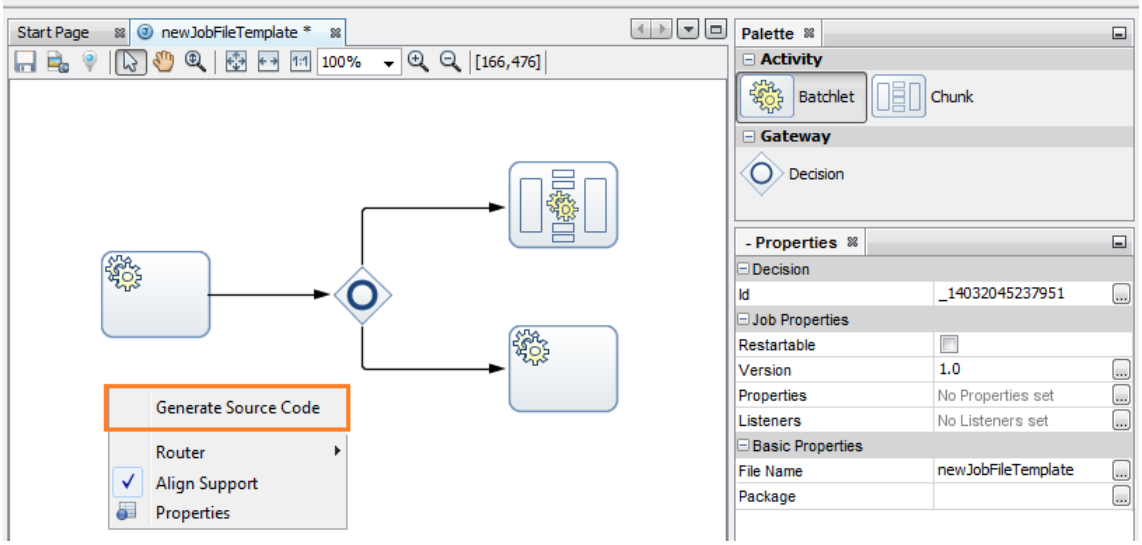
Jednou z požiadaviek na tento nástroj je možnosť graficky navrhovať a upravovať štruktúru workflow pre dávkové úlohy a najmä obojsmerná tranformácia zmien mezdi ich textovou a grafickou reprezentáciou.

Cieľom práce bolo vytvorenie nástroja, ktorý by čo najviac programátorovi zjednodušoval prácu spojenú s dávkovými procesmi. Nevyslovenou požiadavkou na grafický editor je určite možnosť prácou v ňom zastrešiť všetky úpravy, ktoré sú prevádzané nad dávkovým súborom. Najlepším spôsobom by bola možnosť plnohodnotného nahradenia textového editoru grafickým bez toho, aby bol programátor vyslovene nútený vykonávať niektoré zmeny v textovom režime.

1.2 Súčasný stav

V čase písania tejto práce a po overení aj v čase jej odovzdávania neboli objavené žiadne podobné nástroje pre IDE IntelliJ IDEA. Avšak pre IDE Netbeans existuje podobný nástroj. Volá sa jBatch Suite.

[1] IDE - Integrated Development Environment - integrované vývojové prostredie (spojenie editora, prekladača a iných ladiacich prostriedkov)



Obrázek 1.1: jBatch Suite

Tento plugin disponuje vyššie spomínanou možnosťou generovania zdrojového kódu podobne ako plugin popisovaný týmto dokumentom. Naopak ešte v čase odovzdávania práce nepodporoval funkcionalitu obojsmernej synchronizácie zmien medzi grafickou a textovou (XML) reprezentáciou JDF súboru, ktorá je silnou vlastnosťou pluginu, ktorý je cieľom mojej práce, a tak isto v čase písania tejto práce plne nepodporoval všetky vlastnosti dané špecifikáciou jazyka JSL[2].

Samotné IDE IntelliJ IDEA však poskytuje pokročilú asistenciu v písaní kódu (JSL) pre popis dávkových procesov, tak isto doplňovanie kódu a navigáiciu v ňom.

1.3 Štruktúra práce

V druhej kapitole sú vysvetlené základné pojmy a problematika súvisiaca s Batching API. Podrobne popisujem jednotlivé elementy (Steps, Flows, Splits, Decisions), ktorých skladaním vznikajú komplexnejšie dávkové úlohy. V nadväznosti na tieto elementy bude vysvetlený aj jazyk, ktorým sa tieto dávkové úlohy popisujú. Ide o jazyk JSL založený na značkovacom jazyku XML.

V tretej kapitole popisujem architektúru prostredia IntelliJ IDEA. Detailnejšie je popísaný spôsob prístupu k vývoju zásuvných modulov pre toto prostredie, jeho virtuálny súborový systém a vstavané komponenty zaujímavé pre tento zásuvný modul.

Štvrtá kapitola je venovaná návrhu celého pluginu. Podrobnejšie vysvetľuje spôsob návrhu medzivrstvy medzi textovou a grafickou reprezentáciou dávkových úloh.

V piatej kapitole je priblížená samotná implementácia zásuvného modulu a na záver nasledujú dve kapitoly venované testovaniu a záverečnému zhrnutiu celej práce.

[2]JSL - Job Specification Language, jazyk ktorý pomocou XML súboru popisuje presný význam a poradie jednotlivých dielčích úloh

Kapitola 2

Batching API

Batching API sú pre platformu definované ako JSR-352 a sú dostupné od príchodu Java EE 7.

Dávkové spracovanie je vykonávanie série určitých úloh vo vopred stanovenom poradí na počítačovom systéme. Základnou črtou je vykonávanie úloh neinteraktívnou formou, kedy nevyžadujú žiadnu, alebo len minimálnu spätnú väzbu od používateľa. Často sú spojené so spracovaním veľkého množstva dát, ktoré trvá dlhšiu dobu. S narastajúcim množstvom spracovaných dát sa priamo úmerne zvyšuje výpočetná náročnosť, a teda aj vyťaženie zariadenia, na ktorom tieto úlohy bežia. Príkladom môže byť spracovanie veľkého množstva dát v podobe logovacích súborov, rôznych záznamov z databáz alebo spracovanie obrazu. V reálnom svete sa takéto úlohy môžu vyskytovať v podobe koncodňových sumarizačných výpočtov alebo ako koncomesačné generovanie výpisov z bankových účtov.

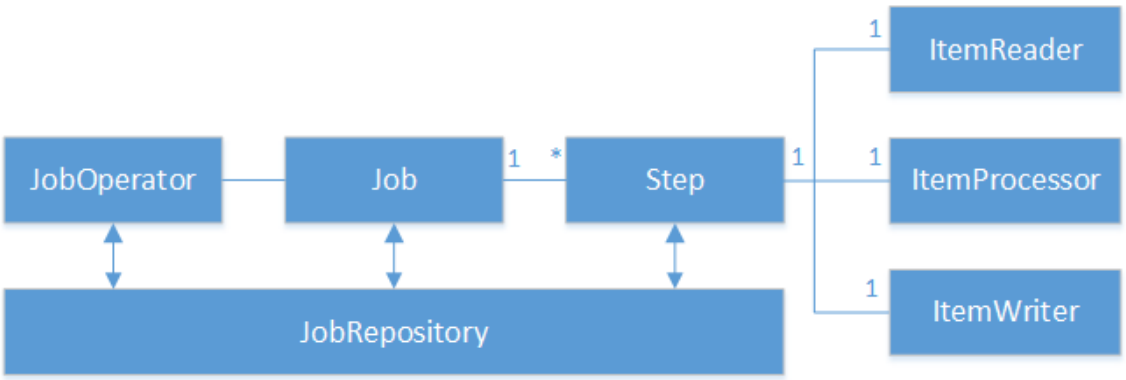
Existuje druhý spôsob využitia dávkového spracovania. Narozdiel od vyššie popísaného spôsobu, kedy je spracovanie vyvolané v istú časovú dobu, ako napríklad na konci dňa, keď zariadenia nie sú až tak využívané a je priestor pre náročnejšie výpočty, druhý spôsob predstavuje vyvolávanie spracovania na základe nejakej udalosti. Typicky sa jedná o úlohy, ktoré sa periodicky opakujú, nepotrebujú žiadnu interakciu s používateľom a len ukladajú určité dáta. Ako príklad si môžeme predstaviť nejakú fakturačnú aplikáciu počítajúcu prevolané minúty.

2.1 Architektúra Java Batch

Na obrázku č. 2.1 je pohľad na zjednodušenú architektúru Java Batch, ktorá je používaná už niekoľko rokov. Popisuje aj niektoré úrovne, ktoré nie sú priamo spojené so samotným pluginom. Na druhej strane je vhodné mať predstavu o tom, ako sú samotné dávkové procesy spúšťané na vyššej úrovni.

- *Job* je entita najvyššej úrovne, ktorá zabaľuje celý dávkový proces (viď kapitola 2.3.1). Obsahuje nula až n prvkov typu `step`.
- *Step* je základný stavebný článok každej dávkovej aplikácie. Nie je závislý na ostatných krokoch a jednotlivé kroky sú vykonávané sekvenčne. Záleží na programátorovi danej aplikácie, ako komplexne a v akom rozsahu bude implementovaný daný krok (viď. kapitola 2.3.2).
- *JobOperator* sprístupňuje rozhranie pre riadenie všetkých častí dávkového spracovania.

- *JobRepository* uchováva informácie o všetkých aktuálne bežiacich prácach a zároveň uchováva aj ich históriu. Každý krok, ktorý je `chunk-oriented` musí obsahovať trojicu prvkov ItemReader, ItemProcessor a ItemWriter. Kroky, ktoré sú tzv. `task-oriented` neobsahujú spomínanú trojicu prvkov.
- *ItemReader* čítač, ktorý zabezpečuje vstupné zdroje pre daný krok. Zo vstupu číta položku po položke. V prípade vyprázdnenia zdroja dát indikuje koniec čítania.
- *ItemProcessor* získava dáta od čítača a vykonáva samotnú logiku spracovania, a teda transformáciu vstupných dát na výstupné.
- *ItemWriter* zapisuje dáta na výstup.



Obrázek 2.1: Koncept architektúry Java Batch

2.2 Jazyk JSL

Job Specification Language (JSL) je jazyk popisujúci samotnú prácu, jednotlivé kroky, z ktorých sa skladá, vlastností špecifických pre typ daného kroku a poradie ich exekúcie. Je založený na značkovacom jazyku XML.

Z jazyka vyplývajú určité obmedzenia z hľadiska povolenej kombinácie a následnosti jednotlivých elementov, ktoré budú presnejšie popísané v ďalších podkapitolách venujúcich sa konkrétnym elementom.

Na obrázku 2.2 je ukážka jednoduchého súboru napísaného v jazyku JSL[1]. Ako môžeme vidieť, obsahuje XML hlavičku s potrebnými údajmi o verzii, kódovaní a samotný popis práce uzavretý do značiek (tagov) koreňového elementu `Job`.

[1] schéma popisujúca jazyk `JSL` je dostupná na nasledujúcej adrese: http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<job id="simpleJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
    <properties>
        <property name="input_file" value="intput.txt"/>
        <property name="output_file" value="output.txt"/>
    </properties>

    <step id="mychunk" next="myTask">
        <chunk>
            <reader ref="MyReader"> </reader>
            <processor ref="MyProcessor"> </processor>
            <writer ref="MyWriter"> </writer>

        </chunk>
    </step>
    <step id="myTask">
        <batchlet ref="MyBatchlet"> </batchlet>
        <end on="COMPLETED"/>
    </step>

</job>
```

Obrázek 2.2: Ukážka jazyka JSL

2.3 Elementy

V JDF súbore môžeme teda okrem hlavičky nájsť aj jednotlivé elementy dávkového spracovania, ktoré budú ďalej v tejto sekcii priblížené. Konkrétne sú to tieto elementy:

- Job element
- Step element
- Flow element
- Split element
- Decision element

2.3.1 Job Element

Job Element je prvok, ktorý sa vyskytuje na najvyššej úrovni. Obaľuje celý dávkový proces, ktorý typicky pozostáva zo sekvencie jednotlivých krokov, ktoré na seba logicky nadväzujú. V podstate je to akýsi kontajner, ktorý dovoľuje nastaviť globálne vlastnosti pre všetky kroky.

```
<job id="{name}" restartable="{true|false}">
      .
      .
      .
</job>
```

Na predchádzajúcej ukážke je možné vidieť všetky povolené atribúty prvku Job. Konkrétne je to povinný atribút `id`, ktorý špecifikuje presný názov daného prvku Job a voliteľný atribút `restartable`.

Môže v sebe obsahovať jeden `properties` prvok obsahujúci viacero `property` prvkov, za ním jeden `listeners` prvok, ktorý môže obsahovať opäť viacero `listener` prvkov a nula alebo viac elementov typu `Step`, `Flow`, `Decision` alebo `Split`.

Prvok poslucháč (`listener`) má za úlohu zachytávať požadované informácie o vykonávaní procesu. Životný cyklus poslucháča je totožný so životným cyklom celej práce. Na úrovni prvku Job môže byť tak isto definovaných viacero vlastností (`properties`), pomocou ktorých je možné definovať vlastnosti ktorémukoľvek vnorenému prvku.

```
<properties>
       <property name="propertyName1" value="propertyValue1"/>
       <property name="propertyName2" value="propertyValue2"/>
</properties>
```

2.3.2 Step Element

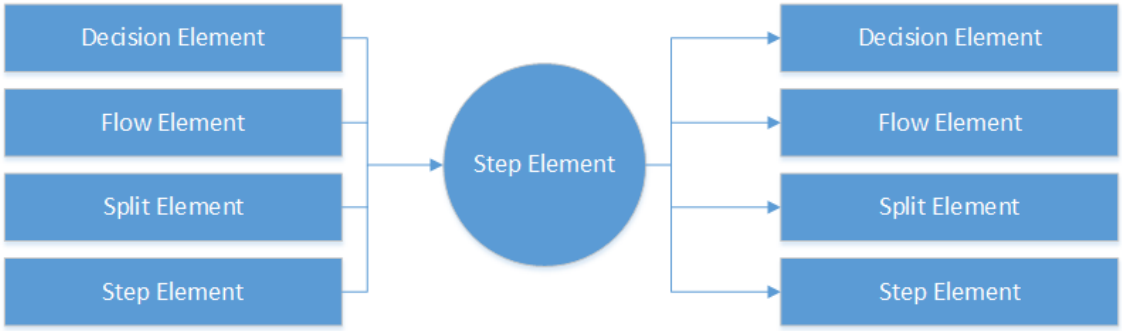
Krok (`step`) je základný stavebný článok pri popise dávkového spracovania úloh jazykom JSL. Obvykle reprezentuje nezávislú, dielčiu časť celej práce a obsahuje, alebo je schopný získať všetky potrebné informácie, aby vykonal činnosť, ktorá mu náleží a po jej ukončení bolo možné pokračovať v poradí ďalším prvkom definovaným v súbore `JDF`. Spravidla sa vyskytuje ako potomok koreňového `Job` elementu, ale môže byť definovaný aj ako vnorený prvok prvku `Flow`.

Nasledujúca ukážka prezentuje všetky atribúty pre objekt typu krok, a tak isto približuje hodnoty, ktoré sú pre ne prístupné. Zo všetkých je povinný len atribút `id` predstavujúci identifikátor daného kroku. Atribút `start-limit` udáva počet možných spustení alebo reštartovaní prvku. Jeho štandardná hodnota je 0, ktorá symbolizuje to, že neexistuje žiadny limit. Ďalším v poradí je `allow-start-if-complete`. Dovoľuje nastaviť pravidlo o povolení znovu-spustenia prvku pri reštarte práce, aj keď už predtým dokončil svoju činnosť. Implicitne je hodnota nastavená na `false`. Napokon je to atribút `next`, spoločný pre všetky ďalšie elementy. Nastavuje nasledujúci prvok, ktorý sa vykoná po aktuálnom prvku.

```
<step id="{nazov}" start-limit="{číslo}"
      allow-start-if-complete="{true|false}"
      next="{ flow-id|step-id|split-id|decision-id}">
      .
      .
      .
</step>
```

Na nasledujúcom obrázku č. **??** sú zobrazení možní predchodcovia a následníci pre daný Step Element.

Kroky sa na základe charakteru ich spracovania rozdeľujú na dve skupiny. Buď obsahujú objekt typu chunk, alebo objekt typu task. Podľa toho rozoznávame tzv. “chunk-oriented“ spracovanie a “task-oriented“ spracovanie. Oba tieto princípy budú priblížené na nasledujúcich riadkoch.



Obrázek 2.3: predchodcovia a následníci prvku Step

Chunk-oriented spracovanie

Chunk-oriented spracovanie je primárnym vzorom, ktorý sa používa v dávkovom spracovaní podľa špecifikácie JSR-352. Jedným zo základných črtov je výskyt trojice entít reader-processor-writer. Dalo by sa povedať, že ide o spracovanie tranzakčného charakteru orientované na jednotlivé položky. Tieto položky sú prečítané čítačom, spracované procesorom do podoby "chunk" a následne zapísané na cieľové miesto.

Nasledujúca ukážka obsahuje všetky dostupné atribúty spolu s načrtnutým spôsobom použitia. Všetky sú nepovinné. Atribút `checpoint-policy` definuje spôsob vytvárania kontrolných bodov. Zatiaľ čo pri implicitnej hodnote `item` je kontrolný bod vytvorený po spracovaní presného počtu položiek definovaného atribútom `item-count`, pri hodnote `custom` sú vytvárané podľa implementovaného algoritmu. Nasledujúci `time-limit` určuje počet sekúnd, po ktorých sa vytvorí kontrolný bod. Môže byť kombinovaný s atribútom `item-count` pri politike `item`, ale pri politike `custom` je ignorovaný. Počet preskočiteľných výnimiek (`skippable-exceptions`), ktoré budú pri vykonávaní ignorované udáva `skip-limit` a `retry-limit` stanoví počet pokusov, kedy sa krok pokúsi zopakovať svoju činnosť po zachytení `retryable-exception`.

```
<chunk checkpoint-policy="{item|custom}"
       item-count{hodnota}"
       time-limit="{hodnota}"
       skip-limit="{hodnota}"
       retry-limit="{hodnota}">
       .
       .
       .
</chunk>
```

Každý "chunk" môže byť nastavený tak, aby sa periodicky vytváral kontrolný bod (checkpoint[2]), ktorý v určitých prípadoch umožní reštart kroku, od posledného konzistentného bodu.

Takýto typ kroku musí vždy obsahovať elementy `ItemReader` a `ItemWriter`. Voliteľný je element `ItemProcessor`. V prípade definovania hodnoty politiky kontrolných bodov na hodnotu "custom" je potrebné dodať `checkpoint-algorithm`, ktorý špecifukuje logiku výskytov kontrolných bodov. Jazyk JSL dovoľuje kroku typu `chunk` nastaviť nasledujúce

[2]checkpoint - kontrolný bod, stanovište; v prípade neúspechu je možné pokračovať vo vykonávanej činnosti od tohoto bodu namiesto vykonávania činnosti od začiatku

vlastnosti: `skippable`, `retryable` alebo `no-rollback-exception-classes`. Ako príklad je uvedená skupina výnimiek `skippable-exception-classes`:

```
<skippable-exception-classes>
       <include class="java.lang.Exception"/>
       <exclude class="java.io.FileNotFoundException"/>
</skippable-exception-classes>
```

kde môže mať definované triedy výnimiek, ktoré majú byť v tomto prípade pri vykonávaní práce preskočené (include class), poprípade skupiny výnimiek, ktoré nemajú byť preskočené (exclude class). Pre podrobnejšie informácie viď špecifikácia jazyka Java.

Task-oriented spracovanie

V tomto prípade ide o alternatívu k chunk-oriented spracovaniu. Obe sa však vzájomne vylučujú. Z označenia krokov typu task-oriented vyplýva, že sú zamerané na úlohu ako celok, narozdiel od predchádzajúceho typu spracovania zameraného na jednotlivé položky (čítanie, spracovanie, zápis). Úloha je spustená vždy len raz, vykoná sa ako celok a na záver vráti programátorom zvolenú návratovú hodnotu.

Samotný task-oriented krok v jazyku JSL reprezentuje jeden `batchlet` element. Môže obsahovať len jeden atribút `ref`, ktorý je zároveň povinný a obsahuje referenciu triedy, ktorá implementuje tento prvok. Z dôvodu možnosti nastavovania rôznych vlastností môže `batchlet` element obsahovať `properties` elementy.

```
<batchlet ref="myBatchlet"/>
```

Každý `step` element môže obsahovať buď trojicu objektov `reader-processor-writer`, alebo objekt `batchlet`, v závislosti na tom či je krok `chunk-oriented`, alebo `task-oriented`. Ďalej bez ohľadu na jeho orientáciu môže krok obsahovať:

- jeden `listeners` element, ktorý môže obsahovať jeden alebo viac vnorených `listener` prvkov
- jeden `properties` element, ktorý môže obsahovať viacero vnorených property elementov
- jeden `partition` element
- viacero `fail`/`stop`/`end` elementov pre nastavenie návratovej hodnoty na `FAILED`/`STOPPED`/`COMPLETED`
- jeden `next` atribút, prípadne jeden alebo viacero `next` elementov (nie je povolené ich kombinovať)

Partitioning

Vykonávanie kroku môže byť rozdelené do viacerých oddielov (partitions). Takýto rozdelený krok beží ako viacero inštancií toho istého kroku. Pre každý oddiel je zvlášť vyhradené vlákno, v ktorom spracúva dáta, ktoré musia byť pre každý oddiel rozdielne. To dovoľuje rozdeliť dáta, a teda aj výpočet do viacerých vlákien a spracovávať ich simultánne.

Spolu s týmto rozdelením kroku je spojená skupina prvkov, ktoré je možné rozdeleniu definovať. Sú to: `mapper`, `plan`, `collector`, `analyzer` a `reducer`. Pre každý z nich je

v prípade ich využitia nutné uviesť referenciu na implementujúcu triedu a ďalej je možné nastaviť im ich vlastné `properties`. Pre podrobnejší popis významu a obmedzení spojených s jednotlivými prvkami viď [5].

2.3.3 Flow Element

`Flow` prvok predstavuje tok (sekvenciu) vnorených `Step` elementov. Každý krok vnútri toku musí obsahovať buď atribút `next`, alebo element `next`. Výnimkou je posledný krok v toku. Z kroku vnútri toku môže existovať len prechod na krok vnútri toku, nie mimo neho a to isté platí aj pre kroky, ktoré sú mimo tohoto toku.

```
flow id="{nazov}"next="{ flow-id|step-id|split-id|decision-id}">
      .
      .
      .
</flow>
```

Jediný povinný atribút je `id`. Štandardne prvok `Flow` predstavuje posledný element v celom procese. Ak je však definovaný atribút `next`, môžu za ním nasledovať prvky typu `Flow`, `Step`, `Decision` a `Split` rovnako ako na obrázku č. **??** pri kroku. Štvorica prvkov, ktoré mu môžu nasledovať tak isto reprezentuje prvky, ktoré mu môžu byť vnorené.

2.3.4 Split Element

Jednoducho povedané, `Split` predstavuje skupinu tokov(prvkov `Flow`), ktoré sa vykonávajú simultánne. Split ukončí svoju činnosť, a teda môže pokračovať vo vykonávaní v poradí ďalšieho prvku, až keď svoju činnosť ukončí každý tok. Množina povolených atribútov je rovnaká ako pri prvku `Flow`, a tak isto skupina povolených predchodcov a následníkov je rovnaká ako pre prvky `Flow` a `Step`. Vnorené toky nemôžu byť spojené hranou so žiadnym prvkom mimo `Split` prvku, ktorému sú vnorené.

2.3.5 Decision Element

Doteraz boli spomenuté prvky, pri ktorých sme pred spustením dávkového spracovania vedeli presne určiť nasledujúci element. Aby mal programátor možnosť meniť priebeh spracovania počas behu programu na základe nejakého výpočtu, existuje rozhodovací `Decision` element. Ten si uchováva informáciu o vykonávaní predchádzajúceho kroku. Na základe znalosti tejto informácie a logike implementovanej programátorom dokáže určiť nasledujúci krok.

Vyžaduje dva povinné atribúty `id` (názov) a `ref` (odkaz na implementujúcu triedu). Môže obsahovať nasledujúce voliteľné elementy:

- jeden alebo viac `fail`/`stop`/`end` elementov, ktoré nastavujú status dávky na `FAILED`/`STOPPED`/`COMPLETED`
- jeden alebo viac `next` elementov
- jeden `properties` element obsahujúci jeden alebo viac vnorených `property` prvkov

Pre Decision prvky nie sú následníci určení pomocou `next` atribútu, ale pomocou `next` elementu. Ako je možné vidieť na nasledujúcej ukážke kódu, takýto `next` element naviac oproti `next` atribútu obsahuje povinný atribút `on`, ktorý určuje na základe akej hodnoty bude zvolený práve daný nasledujúci prvok určený hodnotou atribútu `to` (rovnako to funguje aj pri `next` elementoch pri prvku `Step` a `Flow`).

```
<next on="{navratova hodnota}" to="{element id}"/>
```

Predchodcovia a následníci sú opäť rovnakí ako pre `Step` prvok (viď obrázok č. **??**).

Kapitola 3

Architektúra vývojového prostredia IntelliJ IDEA

IntelliJ IDEA je grafické vývojové prostredie pre programovací jazyk Java. Verejnosti je dostupné od januára v roku 2001. Vývoj tohoto IDE má pod záštitou spoločnosť JetBrains. IntelliJ IDEA je medzi vývojármi dobre známe a v poslednom čase naberá na svojej popularite. Ako z názvu vyplýva, ide naozaj o jedno z najinovatívnejších a najinteligentnejších vývojových prostredí pre jazyk Java dostupných v dnešnej dobe. Vyniká najmä integráciou moderných osvedčených postupov, akými je napríklad inteligentný asistent pre písanie a editáciu zdrojového kódu alebo refaktorizácia kódu. Na IntelliJ IDEA je napríklad postavené aj známe Android Studio pre tvorbu Android aplikácií, ale aj mnoho ďalších: AppCode, PhpStorm, PyCharm, RubyMine, WebStorm a najnovšie aj CLion pre jazyk C a C++. Ďalej disponuje širokou podporou rôznych serverov pracujúcich na J2EE[1] technológií. Medzi podporované servery patria napríklad JBoss, Geronimo, GlassFish, TomCat ale aj ďalšie. Pre koncového používateľa je IDE dostupné v dvoch verziách: Ultimate verzia a komunitná verzia. Rozdiel je v tom, že Ultimate verzia má plnú funkcionalitu a je platená, zatiaľ čo komunitná verzia je dostupná pod licenciou Apache 2 license, a teda je možnosť ju voľne stiahnuť. Neposkytuje plnú funckionalitu čo sa týka podporovaných J2EE serverov alebo rôznych programovacích jazykov. Na druhej strane poskytuje plnú funkcionalitu potrebnú pre Plugin Development, teda vývoj zásuvných modulov. Z toho dôvodu som využíval pre vývoj pluginu práve komunitnú verziu softvéru, ktorá mi úplne postačovala. IntelliJ IDEA je samozrejme dostupná pre platformu Windows, Linux ale aj Mac OS X, a teda aj výsledný plugin nie je obmedzený na jednu platformu, ale môže byť využívaný vývojármi na ktorejkoľvek z troch uvedených platforiem. V čase písania práce a vývoja pluginu bola aktuálna verzia 14.1.2. Prostredie využívalo nainštalovaný Java SE Development Kit vo verzií 1.8.0._25, ktorý je voľne dostupný pod licenciou GPL.

3.1 Vytvorenie nového zásuvného modulu

V tejto časti budú popísané kroky potrebné k vytvoreniu nového projektu, konkrétne pre tvorbu nového pluginu.

Pred samotným vytváraním nového projektu je najprv potrebné nakonfigurovať SDK. V hlavnom menu užívateľ zvolí kartu `File` a následne `Project Structure`. V zobrazenom dialógovom okne pridá nové SDK a nastaví mu cestu do adresára, kde je samotné IDE

[1] J2EE - Java 2 Platform Enterprise Edition - definuje štandard pre vývoj firemných aplikácií

nainštalované a priradí mu Java SDK predinštalované v počítači. Ďalší krok tvorby nového pluginu sa nijako neodlišuje od tvorby štandardnej java aplikácie, keďže každá verzia IDE IntelliJ IDEA podporuje tvorbu pluginov. Najprv si používateľ zvolí možnosť vytvorenia nového projektu. Následne v zobrazenom dialógovom okne zvolí možnosť "IntelliJ Platform Plugin" a ako SDK nastaví SDK nakonfigurované v minulom kroku. Na záver si môže zvoliť medzi dvomi formátmi ukladania projektu a to ukladanie zamerané na adresár, alebo ukladanie zamerané na súbor (pre bližšie informácie viď [7]).

3.1.1 Konfiguračný súbor

Jedným zo základných súborov celého projektu je konfiguračný súbor `plugin.xml`.

```
<IDEA-plugin url="http://www.jetbrains.com/IDEA">
        <name>PluginName</name>
        <id>UniqueIdentifier</id>
        <description>PluginDescription</description>
        <change-notes>Initial release of the plugin.</change-notes>
        <version>1.0</version>
        <application-components>
                <component/>
        </application-components>
        <project-components>
                <component/>
        </project-components>
        <module-components>
                <component/>
        </module-components>
        <actions>
                <action/>
        </actions>
        <extensionPoints>
                <extensionPoint/>
        </extensionPoints>
        <extensions>
                <testExtensionPoint/>
        </extensions>
</IDEA-plugin>
```

V predchádzajúcej ukážke je možné vidieť zjednodušenú verziu konfiguračného súboru, ktorý je akousi riadiacou jednotkou celého projektu. Obsahuje v sebe základné informácie o projekte ako napr. názov projektu, identifikátor, popis projektu, verziu, domovskú stránku a mnohé iné. Na druhej strane však obsahuje aj zložitejšie konštrukcie, v ktorých sú zapísané základné komponenty, rôzne rozšírenia, prípadne akcie s ktorými projekt pracuje.

Ďalej v tejto kapitole budú bližšie popísané niektoré základné prvky z konfiguračného súboru ako aj ďalšie esenciálne prvky využívané pre vývoj pluginov pre prostredie IDEA.

3.2 Komponenty zásuvného modulu

Komponenty sú základným stavebným prvkom a nástrojom, ktorý umožňuje integráciu novovzniknutých pluginov do samotného IDE. Existujú tri úrovne komponentov: komponenty aplikačnej úrovne, projektovej úrovne a komponenty modulovej úrovne.

Pre každý komponent by malo byť v konfiguračnom súbore definované rozhranie a tak isto implementujúca trieda. Implementujúca trieda sa bude používať na inštanciáciu a rozhranie bude slúžiť na to, aby bolo možné k danému komponentu pristúpiť z ostatných komponentov. Každý z komponentov obsahuje vlastné unikátne meno, aby ho bolo možné naprieč ostatnými komponentmi presne identifikovať.

- *aplikačné komponenty* sú vytvorené a inicializované pri štarte IntelliJ IDEA
- *projektové komponenty* sú vytvorené pre každý projekt v IntelliJ IDEA
- *modulové komponenty* sú vytvorené pre každý modul v každom projekte aktuálne načítanom v IntelliJ IDEA

Perzistentný stav komponentov

Vlastnosť ktorá dovoľuje uchovať komponentom svoj stav aj napriek vypnutiu/zapnutiu IntelliJ IDEA.

3.3 Rozšírenia

Koncept rozšírení prináša možnosť interakcie programátorom implementovaného pluginu s ostatnými zásuvnými modulmi. Nemenej významná je možnosť interakcie so samotným jadrom prostredia IntelliJ IDEA.

Existujú dva spôsoby použitia rozšírení. Prvý spôsob predstavuje použitie samotných rozšírení ako takých. Ak chce programátor rozšíriť funkcionalitu iných pluginov alebo jadra IDE IDEA, musí toto rozšírenie definovať v konfiguračnom súbore projektu, ako bolo načrtnuté v ukážke súboru v sekcii 3.1.1.

Druhá možnosť programátora je použitie takzvaných bodov rozšírenia. Tieto body samé o sebe nijako nerozširujú funkcionalitu, či už ostatných zásuvných modulov alebo samotného prostredia. Naopak dávajú ostatným pluginom a vývojárom možnosť, aby mohli nejakým spôsobom rozšíriť svoju vlastnú funkcionalitu. Vďaka tomu sa predchádza zbytočnej duplicite kódu pri vytváraní podobných zásuvných modulov, kedy ich netreba vytvárať od začiatku, ale stačí len rozšíriť existujúce riešenia.

```
<extensionPoints>
       <extensionPoint name="bodRozsirenia1" beanClass="MyBeanClass1">
       <extensionPoint name="bodRozsirenia2 interface="MyInterface">
</extensionPoints>
```

Každý bod rozšírenia musí definovať buď triedu, alebo rozhranie, ktorým je dovolené k nemu pristupovať, a tým dá ostatným pluginom informáciu o tom, akým spôsobom je potrebné daný plugin rozširovať a pristupovať k nemu.

3.4 Systém akcií

Aby mohol programátor vytvoriť vlastné položky v paneloch nástrojov, hlavnom menu programu alebo dokonca vytvoriť vlastný panel nástrojov zabudovaný do IDE, musí využiť systém akcií sprostredkovaný prostredím IDEA.

Každá akcia je implementovaná triedou odvodenou od triedy `AnAction`. Vždy, keď používateľ klikne na položku v menu alebo na paneli nástrojov, zavolá sa jej metóda `actionPerformed`. Akcia má svoj jedinečný identifikátor a môže byť priradená do skupiny akcií, ktorá má tak isto svoj identifikátor. Každá skupina akcií môže obsahovať podskupiny akcií, čím vzniká hierarchia. Takéto skupiny môžu vytvárať celé menu alebo panel nástrojov a ich podskupiny môžu byť akési pod-menu.

Jedna akcia môže byť zahrnutá vo viacerých skupinách, a teda môže byť vyvolaná viacerými tlačidlami v rámci užívateľského rozhrania. Pre každé z tých tlačidiel môže mať definovanú inú ikonu. Aby bol zaručený aktuálny stav akcií, IDEA periodicky volá metódu `AnAction.update()`.

Existujú dve možnosti registrovania akcií. Prvá možnosť je registrovanie pomocou konfiguračného súboru (viď sekcia 3.1.1). Druhá možnosť je programové registrovanie akcií zo zdrojového kódu pomocou triedy `ActionManager` (pre bližšie informácie viď [2]).

3.5 Súborový systém

Súčasťou IntelliJ IDEA je virtuálny súborový systém. Keďže väčšina aktivít, ktoré sa vykonávajú za použitia tohoto IDE je spojená práve s úpravami, vytváraním alebo mazaním súborov, úlohou tohoto virtuálneho súborového systému (VFS) je vytvorenie abstrakcie nad prácou s týmito súbormi. Toto sú hlavné účely VFS:

- sprístupňuje univerzálne API[2] pre prácu so súbormi, pričom nezáleží na ich type, veľkosti či umiestnení;
- zisťuje zmeny súborov na disku a v prípade, že nastala zmena, sprístupní obe verzie súborov (pred zmenou aj po zmene).

Snímka súborov

Aby bolo možné v prípade zmeny zabezpečiť starú aj novú verziu súboru, VFS vytvára snímku určitého obsahu pevného disku. Nachádzajú sa v nej však len tie súbory, ku ktorým bol už aspoň raz vyžiadaný prístup cez VFS API. Táto snímka sa vytvára na aplikačnej úrovni. To znamená, že ak je jeden súbor v rámci jedného behu aplikácie referencovaný z viacerých miest, tak v snímke súborov sa vyskytuje len jeden krát.

Obsah, ktorý je zobrazovaný v rámci grafického rozhrania je čerpaný zo snímky súborov. Táto snímka ja asynchrónne aktualizovaná zo súborov na disku. Aktualizácia môže byť vyvolaná vnútorne samotným IDE alebo vyžiadaná programovo zo zásuvného modulu. To znamená, že zobrazovaný obsah nemusí byť stále aktuálny a zobrazované súbory už na disku nemusia existovať, avšak v aktuálnej snímke sa stále nachádzajú. Zmeny v snímke sa prejavia po ďalšej aktualizácii. Narozdiel od čítania súborov, ktoré prebieha z aktuálnej snímky, zápis súborov je synchrónna akcia a obsah sa zapisuje priamo na disk.

[2] Application programming interface (rozhranie pre programovanie aplikácií) - zbierka funkcií, procedúr, tried alebo protokolov nejakej knižnice, ktoré môžu byť využívané programátorom.

Virtuálny súbor

Každý súbor vo VFS je reprezentovaný ako virtuálny súbor nezáležiac na jeho type. Virtuálny súbor je reprezentácia súboru s najnižšou úrovňou abstrakcie v rámci VFS. Nezahŕňa v sebe informácie o koncoch riadkov alebo o kódovaní. Jeho obsah je čisto binárneho charakteru a čítanie/zápis je možné len pomocou prúdu bytov. Medzi bežné operácie nad súborom patrí získanie dát, presun, premenovanie a mazanie súboru. Zmeny sú vykonávané na aplikačnej úrovni.

Dokument

Dokument je vyššia úroveň abstrakcie nad virtuálnym súborom. Jeho obsah by sa dal charakterizovať ako sekvencia Unicode[3] znakov, ktorá zodpovedá binárnemu obsahu virtuálneho súboru. Zároveň obsahuje aj informácie o koncoch riadkov a za znak konca riadku sa považuje znak `'\n'`. Dovolené operácie sú akékoľvek operácie, ktoré menia obsah súboru na úrovni obyčajného textu a zmeny sú opäť vykonávané na aplikačnej úrovni.

PSI Súbor

Narozdiel od predchádzajúcich dvoch prípadov, zmeny nad PSI súbormi sú vykonávané na projektovej úrovni. Obsah PSI súboru predstavuje koreň hierarchickej štruktúry prvkov, ktoré sú typické pre daný programovací jazyk (napr. `XmlFile`, `PsiJavaFile` atď.).

PSI Element

Ako bolo spomenuté v predhádzajúcich riadkoch, PSI Súbor je koreň hierarchickej štruktúry prvkov. Tie prvky sú konkrétne PSI elementy, ktoré môžu obsahovať ďalšie PSI elementy ako svoje potomky, a tým vzniká usporiadanie do stromovej štruktúry. V rámci jedného PSI súboru môže byť definované ľubovoľné množstvo PSI stromov. Každý z týchto stromov môže byť pre iný jazyk. PSI stromy nachádzajú využitie napríklad pri analýze kódu.

3.6 Distribúcia pluginu

Zverejnenie pluginu je skutočne jednoduchý krok. Programátor potrebuje len aktívny Jetbrains účet a spustené IDE IntelliJ IDEA s otvoreným projektom spolu s vyvíjaným zásuvným modulom. Projekt sa musí zostaviť a následne z "Build" menu zvoliť možnosť "Prepare Plugin Module For Deployment". IDEA následne informuje programátora o vytvorení .zip súboru. Tento súbor obsahuje .jar súbor s pluginom a v prípade používania ďalších knižníc obsahuje aj tieto knižnice. Posledný krok je nahranie pluginu do online repozitára[4].

[3] Unicode - medzinárodný štandard, ktorý definuje kódovanie pre väčšinu znakov a rôznych ďalších symbolov

[4] repozitár dostupný na adrese: https://plugins.jetbrains.com/plugin/addPlugin

Kapitola 4

Návrh

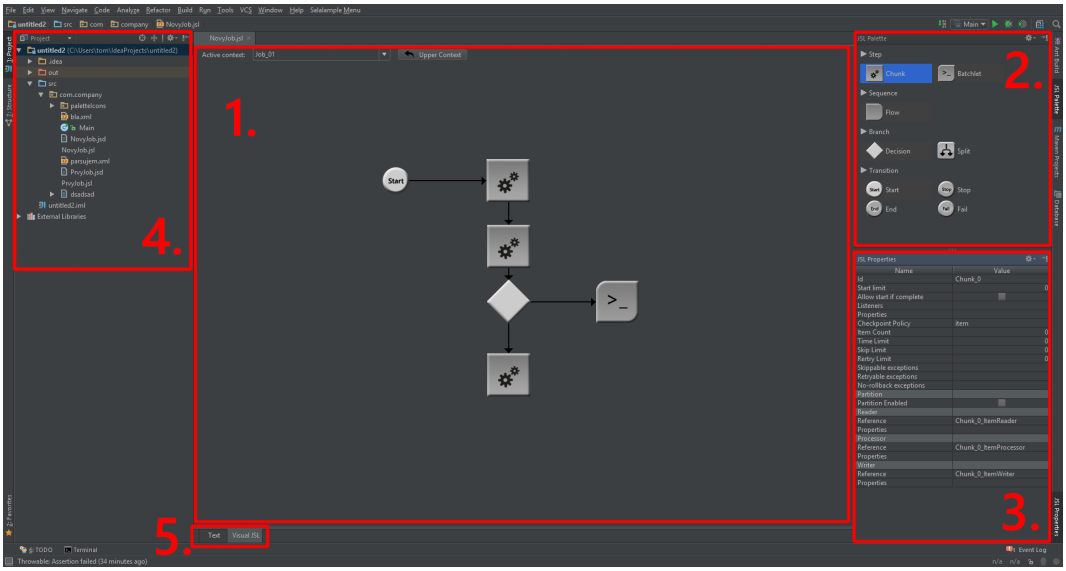
Najdôležitejším krokom tvorby celého pluginu, a tak isto krokom, ktorý zabral najviac času bol návrh. Samotné IDE IntelliJ IDEA bolo pre mňa pred začiatkom prác na tomto projekte jedna veľká neznáma. Z toho dôvodu som musel vynaložiť mimoriadne úsilie na poriadne preštudovanie dostupných materiálov a dokumentácií, aby som bol schopný správne zvoliť, ktoré zo vstavaných nástrojov tohoto prostredia bude vhodné použiť. To všetko, či už z dôvodu uľahčenia práce, ale aj preto, aby výsledný plugin naozaj vyzeral ako plugin pre IntelliJ IDEA, a nie ako zásuvný modul, ktorého určenie je len veľmi ťažko identifikovať.

V úvode tejto kapitoly sú popísané základné grafické prvky pluginu a ich rozmiestnenie v rámci grafického rozhrania prostredia IDEA. V ďalšej sekcií je vysvetlený spôsob ukladania potrebných dát do súborov a v závere tejto kapitoly budú priblížené navrhnuté spôsoby manipulácie s jednotlivými prvkami.

4.1 Popis rozloženia prvkov

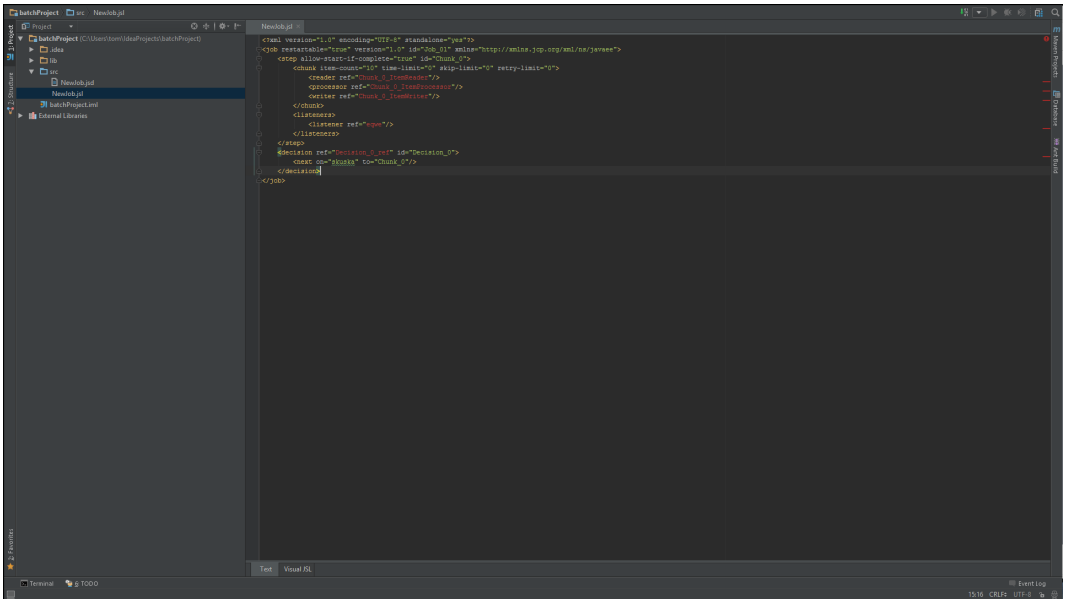
Podľa obrázku č. 4.1 sa dá povedať, že dominantným prvkom grafického režimu je samotné plátno (vyznačený blok č. 1) pre zobrazenie diagramu. Ide o najdôležitejší prvok celého pluginu, keďže na ňom budú prebiehať všetky úpravy grafickej reprezentácie daného .jsl súboru. Z toho dôvodu zaberá takmer celú šírku okna. Na ľavej strane sa môžu nachádzať štandardné panely nástrojov samotného prostredia ako napríklad panel so štruktúrou celého projektu (vyznačený blok č. 5). Na pravej strane obrazovky sa nachádzajú dva panely, ktoré sú súčasťou tohoto pluginu. V prípade, že užívateľ potrebuje maximalizovať šírku plátna na celú šírku okna, môže tieto panely oboch stranách obrazovky skryť.

Spomínané dva panely na pravej strane sú "`JSL Palette`" panel (blok č. 2) a "`JSL Properties`" panel (blok č. 3). Umiestnenie na pravej strane samozrejme nie je jedinou možnosťou. Užívateľ si tieto panely môže umiestniť na ľubovoľné miesto (vľavo, dole), ale z dôvodu prednastavenej pozície panelu so štruktúrou projektu na ľavej strane je pozícia týchto dvoch panelov implicitne prednastavená na pravú stranu.



Obrázek 4.1: Návrh rozloženia prvkov v grafickom režime

Keďže jednou z požiadaviek bola možnosť obojsmernej transformácie zmien medzi textovou a grafickou reprezentáciou jazyka JSL, popri grafickom editore je potrebný aj textový editor (viď obrázok 4.2). Ako správny textový editor prostredia IDEA by mal disponovať vymoženosťami ako inteligentné doplňanie kódu, priebežná kontrola kódu a zvýrazňovanie chýb, ale aj refaktorovanie kódu.



Obrázek 4.2: Návrh rozloženia prvkov v textovom režime

Ako posledný, ale nemenej dôležitý prvok, ktorý je potrebné spomenúť, je dvojica kariet v spodnej strane obrazovky grafického a textového režimu (viď obrázok č. 4.1 (blok č. 5) a obrázok č. 4.2). Tieto karty slúžia na jednoduché prepínanie medzi obomi editormi.

Paleta

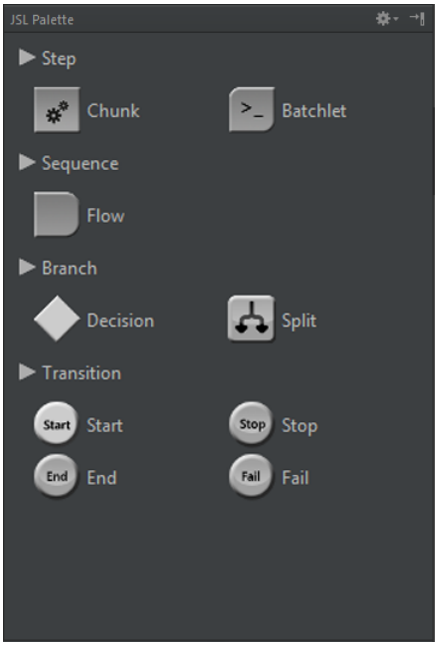
Paleta nástrojov (viď obr. 4.3) je veľmi jednoduchá. Obsahuje v sebe zoznam grafických elementov, ktoré sú z dôvodu rôznych významov a zvýšeniu intuitívnosti výberu zoradené do kategórií: `Step`, `Sequence`, `Branch` a `Transition`. Používateľ si môže z týchto elementov vybrať prvok a pomocou jednoduchého DnD[1] gesta ho umiestniť na požadované miesto na plátne.

Paleta obsahuje zmenšené verzie obrázkov, ktoré reprezentujú dané prvky na plátne. Jednotlivé elementy sú odlíšené tvarmi, aby ich bolo možné jednoducho a rýchlo identifikovať. Naopak elementy, ktoré predstavujú koncové prechody a element `Start` majú veľmi podobný vzhľad, ktorý sa na druhej strane viditeľne líši od ostatných základných elementov.

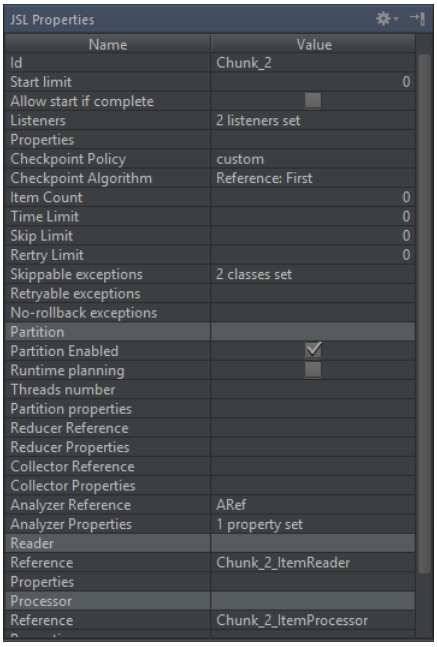
Panel vlastností

Panel vlastností (viď obr. 4.4) je narozdiel od palety nástrojov podstatne komplikovanejší. Obsahuje v sebe zoznam všetkých vlastností, ktoré je možné nastaviť pre aktuálne označený prvok na plátne. Takmer všetky prvky sa vo vlastnostiach principiálne odlišujú, s výnimkou podobných prvkov ako `End`, `Fail` a `Stop`, pre ktoré vyzerá panel vlastností rovnako.

Okrem elementov (vrátane koreňového Job Elementu), ktorým sa dajú nastavovať vlastnosti, je možné nastaviť vlastnosť aj prechodom. Konkrétne ide o prechody, ktorých cieľom sú prvky obsahujúce atribút `on` (Stop, Fail, End) alebo prechody pri ktorých v zdrojovom elemente vzniká vnorený `next` element, ktorý tak isto obsahuje atribút `on`. Tento atribút určuje hodnotu, po ktorej bude vykonaný práve nasledujúci krok, a teda viac intuitívnym prístupom je nastavovať tento atribút hrane a nie samotnému elementu.



Obrázek 4.3: Návrh palety nástrojov



Obrázek 4.4: Návrh panelu vlastnosťí

Panel vlastností zobrazuje rôzne typy vlastností. Môžu sa vyskytovať jednoduché dátové typy ako `boolean`, `String` a `int`. Zároveň sú zastúpené aj zložitejšie dátové typy,

[1]Drag and Drop (v preklade ťahaj a pusti) znamená spôsob výberu objektu, ťahaním a následným pustením objektu na miesto kam má byť premiestnený, za účelom vykonania nejakej akcie.

ktoré reprezentujú zložitejšie vlastnosti prvkov (napr. `Listeners`, `Properties`, `Checkpoint Algorithm` atď.), a preto vyžadovali aj zložitejšiu implementáciu editorov týchto vlastností (viac v kapitole Implementácia).

4.2 Previazanie špecifikácie a grafickej reprezentácie

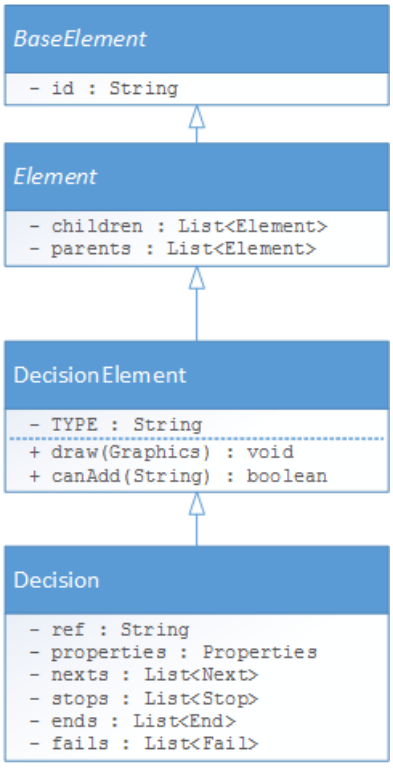
Prvky z palety sa dajú na základe ich reprezentácie v definičnom `.jsl` súbore rozdeliť do troch skupín. Do prvej skupiny patria prvky, ktoré sú samostatne stojace, teda môžu byť priamym potomkom koreňového `Job` elementu. Medzi ne patria `Chunk, Batchlet, Flow, Decision a Split`. Do druhej skupiny patria prvky, ktoré v `.jsl` môžu existovať len ako vnorený element `Decision` alebo `Step` elementu. V tretej skupine je len prvok `Start`, ktorý v definičnom súbore nemá priame zastúpenie. Jeho význam bude popísaný ďalej v tejto kapitole.

Pre všetky tri spomenuté skupiny platí podobný princíp prepojenia `JSL` špecifikácie a ich grafickej reprezentácie. Ako príklad je na obr. 4.2 uvedený zjednodušený diagram tried prvku `Decision`. Na samom vrchu je abstraktná trieda `BaseElement`, ktorá v sebe obsahuje jediný spoločný atribút pre všetky elementy. Je to atribút `id` a je súčasťou špecifikácie prvku.

Abstraktná trieda `Element` je tak isto spoločná pre všetky prvky. Obsahuje v sebe základné informácie ako pozícia na plátne, predchodcov a následníkov daného prvku z pohľadu grafickej reprezentácie, a teda aj určuje orientáciu hrán spojených s ním. Dá sa povedať, že väčšina manipulácií s prvkami, presúvanie, označovanie a pridávanie/odoberanie nasledovníkov je spojená s touto triedou.

Ďalšia trieda je `DecisionElement`, ktorá rozširuje triedu `Element`. Obsahuje špecifické informácie pre `Decision` element. Uchováva v sebe informáciu o type. Ďalej sa stará o vykreslenie odpovedajúcej grafickej reprezentácie prvku na plátno na pozíciu, ktorá je uchovaná v triede `Element`.

Posledná trieda z diagramu už nie je nijako spojená s vykresľovaním ani umiestňovaním prvku. Ide o triedu ktorá v sebe obsahuje celú špecifikáciu `Decision` elementu s výnimkou atribútu `id`, ktorý je zahrnutý v triede `BaseElement`, keďže je pre všetky prvky spoločný.



Obrázek 4.5: UML Diagram pre Decision Element

4.3 Súbory .jsl a .jsd

Pri tvorbe diagramu vznikajú informácie, ktoré je potrebné nejakým spôsobom uložiť. Sú to dáta, ktoré predstavujú samotný popis štruktúry práce v jazyku `JSL` a ďalej sú to doplňujúce definície tejto práce, pozícií prvkov atď. Tieto dáta by mohli byť uložené v jednom

súbore, ale to by prinieslo zložitejšiu úpravu textovej reprezentácie popisu práce z dôvodu veľkého množstva redundantných informácií. V textovom režime programátora nezaujíma umiestnenie prvkov na plátne rovnako ako ďalšie špecifikácie spojené s grafickým editorom.

Ukladanie dát je rozdelené do dvoch súborov s príponami `.jsl` a `.jsd`. V prvom prípade ide výhradne o popis štruktúry práce v jazyku `JSL` s malou výnimkou pri koncových prechodových elementoch, kedy je potrebné ukladať prvok `id`, ktorý sa v originálnej špecifikácií pre tieto prvky nenachádza, ale je potrebný pre plnú funkčnosť pluginu. V druhom prípade ide o súbor `.jsd` obsahujúci spomínané umiestnenie prvkov na plátne, niektoré prídavné elementy ako napr. `Start` element, ktorý sa nevyskytuje ako samostatný element v popisovanom jazyku a z toho dôvodu je potrebné ho uložiť ako prídavný element.

Obidva editory tohoto pluginu sú editormi súboru `.jsl`, a preto zmeny v nich vykonané sa priamo zapisujú do týchto súborov. Súbor `.jsd` je využívaný len grafickým editorom.

V prípade, že pre daný element z definičného súboru neexistuje ekvivalentný záznam o jeho pozícií v súbore so špecifikáciou, bude tento prvok umiestnený na základnú pozíciu, programátor si ho presunie na požadované miesto a pri ďalšom uložení programu sa už vytvorí záznam s jeho špecifikáciou. Rovnakým princípom je zvládnutá aj situácia, keď chýba celý špecifikačný súbor `.jsd`. V takom prípade sa plugin postará o rozmiestnenie prvkov tesne jeden vedľa druhého a je na programátorovi, aby si tieto prvky rozmiestnil podľa seba. Následne sa pri ďalšom uložení opäť vytvorí súbor so špecifikáciou daných elementov.

4.4 Manipulácia s prvkami

4.4.1 Návrhársky kontext

Zo špecifikácie vyplýva, že prvky typu `Flow` môžu obsahovať vnorené elementy. Návrhom odpovede na otázku, akým spôsobom na plátne zobrazovať prvky viacerých úrovní je vytvorenie návrhárskych kontextov. V jednom momente môžu byť zobrazené len prvky rovnakého kontextu. Ak sa v štruktúre prvkov nenachádza žiadny `Flow` element s vnorenými prvkami, tak jediným kontextom je kontext koreňového `Job` elementu. Ďalšie kontexty vznikajú len v prípade vloženia vnoreného prvku ľubovoľnému `Flow` elementu.

Zmena kontextu môže byť vykonaná viacerými spôsobmi. Prvý spôsob je dvojklik na ľubovoľný `flow` element, kedy sa aktuálny kontext zmení na kontext zvoleného `Flow` prvku. Možnosť návratu do kontextu o úroveň vyššie prinášajú ovládacie prvky na malom paneli nad plátnom (viď 4.1). Nachádza sa tam ovládací prvok `ComboBox` obsahujúci list všetkých dostupných kontextov a tlačidlo `Upper Context`, ktoré zmení aktuálny kontext na kontext, ktorý je o jednu úroveň vyššie (v prípade kontextu `Job` už neexistuje rodičovský kontext).

4.4.2 Vytváranie/Mazanie/Úprava elementov

Vytváranie

Pri vytváraní elementov by sa patrilo spomenúť možné spôsoby ich vytvárania. Zdrojom týchto elementov ja vždy paleta a pomocou jednoduchého DnD gesta je možné tieto prvky umiestniť na plátno. Plátno inteligentne reaguje a napomáha používateľovi počas ťahania prvku ponad plátno. Zvýrazňuje prvky, s ktorými je možné novovytvorený prvok nejakým spôsobom asociovať. Buď je to pridanie nového prvku ako následníka, alebo ako koncového prechodového bodu, alebo ako vnorený prvok prvku `Split`. Okrem priamej asociácie novo-

vytvoreného prvku je možné vytvoriť prvok bez asociácie, to znamená vytvorenie nového prvku, ku ktorému nebude viesť žiadna hrana.

Mazanie a úprava

Vzniknuté asociácie medzi prvkami je možné ľubovoľne meniť, prípadne mazať. Ak dôjde k zmene pôvodného návrhu štruktúry a programátor bude chcieť zmeniť predchodcu daného prvku, môže jednoducho odstrániť hranu medzi pôvodným predchodcom a prvkom a vytvoriť novú hranu s požadovaným predchodcom. Nastaviť predchodcov už vytvoreným prvkom je možné aj pomocou DnD gesta rovnako ako pri vytváraní nových prvkov, kedy plátno tak isto zvýrazňuje možných predchodcov.

Funkcia mazania elementov, mazania hrán alebo vytvárania nových hrán je dostupná po kliknutí pravým tlačidlom myši na danú hranu/element.

Kapitola 5

Implementácia

Táto kapitola bude venovaná bližšiemu popisu implementácie súčastí popísaných v kapitole č. 4. Bližšie spomenuté budú aj niektoré z komponentov prostredia IDEA z kapitoly č. 3 spolu s konkrétnym spôsobom ich využitia v zásuvnom module. Zároveň budú popísané aj ďalšie dôležité, zatiaľ nespomenuté časti týkajúce sa implementácie.

Zásuvný modul je implementovaný v jazyku Java, nakoľko celé vývojové prostredie je napísané práve v tomto jazyku. Vďaka tomu je zaručená multiplatformnosť celého IDE, ako aj implementovaného pluginu. Samotný vývoj vyžaduje nainštalované a nakonfigurované IntelliJ IDEA SDK pre daný projekt. Aby bolo možné preložiť a spustiť aktuálny projekt, je potrebné povoliť plugin `DevKit`, ktorý je súčasťou IDE pri platenej verzii Ultimate ako aj pri voľnej komunitnej verzii. Na mapovanie tried jednotlivých vlastností elementov na ich XML reprezentáciu je využitá `JAXB`[1] architektúra, ktorá je súčasťou Java SE platformy.

Implementácia zásuvného modulu je logicky rozdelená do troch balíkov:

- balík `designer`
- balík `specification`
- balík `codeGeneration`.

Poradie vymenovania týchto balíkov zodpovedá ich dôležitosti a tak isto ich rozsahu z pohľadu veľkosti a množstva tried, ktoré obsahujú. Balík `designer` je najdôležitejším z celého pluginu. Obsahuje v sebe implementáciu takmer celej funkcionality. Konkrétne ide o realizáciu základných grafických stavebných prvkov zásuvného modulu: grafický dizajnér (plátno), panel palety prvkov, z ktorých je možné zostaviť diagram a panel pre nastavenie vlastností daným elementom. Spomenuté tri časti sa nachádzajú v pod-balíku `ui`, čo je skratka pre "user-interface", a teda prvky reprezentujúce grafické rozhranie. Ďalej sa v tomto balíku nachádzajú pod-balíky: `actions` s akciami zaregistrovanými v konfiguračnom súbore projektu a `resources`, ktorý obsahuje zdroje potrebné k behu pluginu.

V balíku `specification` sa nachádzajú triedy zabezpečujúce špecifikáciu pre jednotlivé elementy pomocou použitej architektúry `JAXB` vysvetlenej ďalej v tejto kapitole.

Posledný balík `codeGeneration`, ako z názvu vyplýva, v sebe zahŕňa funkcionalitu zodpovednú za generovanie súborov. Konkrétne ide o dvojicu `.jsl` a `.jsd` súborov potrebných pre vnútornú funkcionalitu pluginu a možnosti textovej úpravy diagramu. Zároveň generuje aj výsledný `.xml` súbor, s ktorým potom pracuje server a získava z neho inštrukcie v poradí definovaným programátorom, ktoré sú potrebné pre samotné dávkové spracovanie.

[1] JAXB - Java Architecture for XML Binding

5.1 Grafický dizajnér

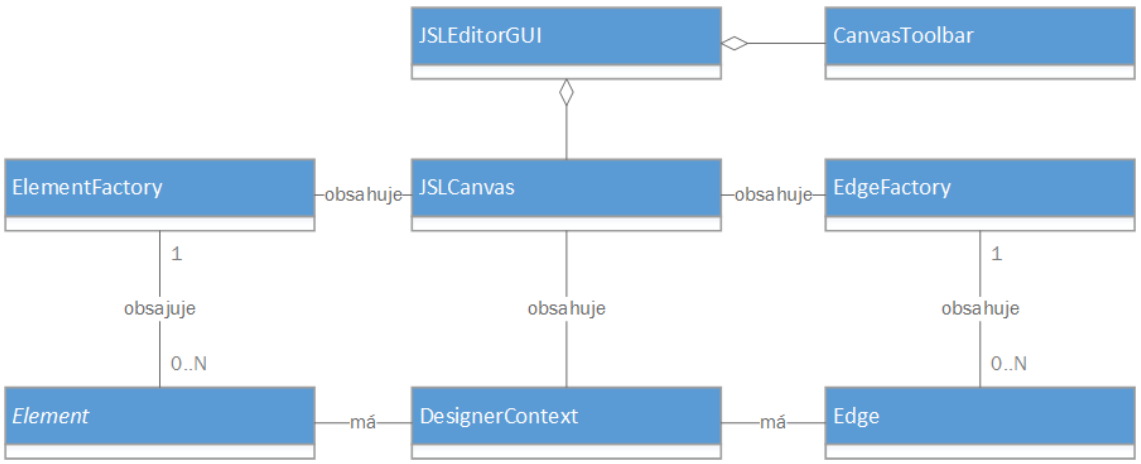
Grafický editor súborov

Predtým ako budú popísané jednotlivé grafické prvky grafického dizajnéra, je potrebné spomenúť spôsob, akým tento dizajnér vzniká. V sekcii č. 3.3 bola spomenutá možnosť rozšírení pre zásuvný modul. Grafický editor je v konfiguračnom súbore `plugin.xml` zaregistrovaný ako rozšírenie `<fileEditorProvider/>`. Povinnosťou je uviesť triedu, ktorá predstavuje poskytovateľa novo-vytvoreného editoru súborov (`fileEditorProvider`). V nej je možné špecifikovať názov editoru, ktorý bude zobrazený v spodnej časti obrazovky na karte na prepínanie aktívnych editorov. Ďalej je v metóde `accept` uvedený typ súborov, s ktorým sa má daný grafický editor asociovať (`.jsl`) a v metóde `getPolicy` je nastavená politika, podľa ktorej má byť karta s grafickým editorom umiestnená v závislosti na pôvodný editor (v tomto prípade je umiestnená za pôvodný editor).

Grafický editor (`JSLFileGrahpicEditor`) vytvorený v triede poskytovateľa editora súborov musí implementovať triedu `FileEditor`. Obsahuje v sebe dve zaujímavé metódy `selectNotify` a `deselectNotify`, ktoré sú vyvolané pri zmene editora, či už z textového na grafický alebo naopak. Sú spojené s generovaním súborov a s načítavaním dát zo súborov. Bližšie spomenuté budú v sekcií č. 5.3. Poslednou metódou tejto triedy, ktorá bude spomenutá je metóda `getEditor`. IDE pri prepínaní editorov volá túto metódu a očakáva od nej vrátenie grafického prvku predstavujúceho samotný editor súborov. Tento grafický prvok je inštancia triedy `JSLEditorGui` a obsahuje v sebe dva grafické panely. Jedným je panel nástrojov plátna (trieda `CanvasToolbar`) a druhým je hlavný panel - plátno (trieda `JSLCanvas`).

Diagram tried

Nasledujúci diagram tried na obrázku č. 5.1 znázorňuje štruktúru niektorých vybraných tried v balíku `designer.ui.editor`. Z dôvodu zvýšenia prehľadnosti obrázku a ušetrenia miesta je diagram veľmi zjednodušený a slúži hlavne na ukážku a pochopenie vzťahov medzi zobrazenými triedami, a nie na vysvetlenie ich funkcionality.



Obrázek 5.1: Zjednodušený diagram tried grafického editoru

Canvas

Z pohľadu zobrazenia a úpravy diagramu ide o hlavnú triedu. Ako je možné vidieť na obrázku č. 5.1, obsahuje v sebe inštancie tried `ElementFactory` a `EdgeFactory`, ktoré v sebe zhromažďujú operácie spojené s jednotlivými elementmi a hranami medzi nimi, vrátane ich vytvárania a mazania. Bližšie budú popísané ďalej v tejto kapitole. Ďalej sa stará o správu návrhárskych kontextov (na obr. 5.1 trieda `DesignerContext`) a aktuálne zobrazeného kontextu. Všetky aktuálne dostupné kontexty uchováva v kolekcii typu `ArrayList<DesignerContext>`. Aktívny kontext je navyše dostupný v privátnom atribúte `activeContext`.

Trieda `Canvas` rozširuje triedu grafickej komponenty `JPanel`. Pomocou preťaženia metódy `paintComponent` z nadradenej triedy, ktorá sa stará o vykreslenie samotného panelu a za využitia knižnice `AWT` sa na plátno vykresľujú jednotlivé elementy a hrany.

Obsahuje aj trojicu privátnych podtried: `MouseHandler`, `MouseMotionHandler` a `CanvasDropTargerLitener`. Ako z ich názvu vyplýva, význam každej z nich je spojený so zachytávaním akcií myši alebo iného ukazovacieho zariadenia. Prvá z nich zachytáva udalosti pre kliknutie, stlačenie a uvoľnenie myši. Reakciou na tieto udalosti môže byť vytvorenie novej hrany, označenie elementu/hrany pod kurzorom, zmazanie elementu/hrany a ďalšie, ktorých podrobný popis sa z dôvodu obmedzeného rozsahu nezmestí do obsahu tejto práce. Trieda `MouseMotionHandler` zachytáva a reaguje na udalosti pohybu a ťahania kurzora po plátne. Reakciou môže byť posun celého plátna, posun jednotlivých elementov alebo zvýrazňovanie elementov/hrán aktuálne nachádzajúcich sa pod kurzorom myši. Posledná trieda zo spomenutej trojice sa stará o vytváranie nových elementov na plátne. Odchytáva udalosti, ktoré vznikajú pri geste `DnD` po pretiahnutí vytváraného prvku z palety nástrojov a "pustení" tohoto prvku nad plátnom.

ElementFactory

Trieda `ElementFactory` by sa dala jednoducho popísať ako továreň na elementy spojená zo skladom vytvorených elementov. Vytvorené elementy uchováva v kolekcii `elements` typu `ArrayList<Elements>`. Koreňový `Job` element je uložený v atribúte `rootElement`. Okrem vytvárania a mazania elementov obsahuje aj ďalšie dôležité metódy pomáhajúce k celkovej funkčnosti pluginu. Napríklad získavanie párového elementu k `Split` elementu, získanie prvku nad ktorým sa aktuálne nachádza kurzor a mnohé ďalšie metódy pracujúce s jednotlivými elementmi. Za spomenutie ešte stojí metóda `canAddChildToParent`, ktorej úlohou je rozhodnúť či danému rodičovskému elementu (parameter `possibleParent`) je možné vložiť požadovaný element (parameter `possibleChild`).

EdgeFactory

Rovnako ako trieda `ElementFactory` aj trieda `EdgeFactory` sa dá označiť ako továreň a zároveň sklad existujúcich hrán. Obsahuje v sebe však len metódy spojené s vytváraním hrán, mazaním hrán a metódu vracajúcu hranu, ktorá sa nachádza "pod" kurzorom. Zoznam všetkých vytvorených hrán je uložený v kolekcii `edges` typu `ArrayList<Edge>`.

Element

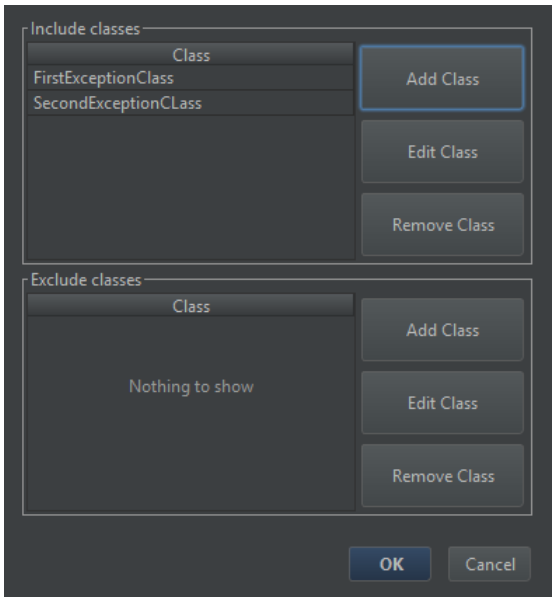
Element reprezentuje každý prvok vykreslený na plátne, bez ohľadu na jeho typ alebo zastúpenie vo výslednom `JDF` súbore. Rozširuje triedu `BaseElement`, ktorá je súčasťou špecifikácie (viď sekcia 5.2). Obsahuje v sebe atribúty, ktoré sú spojené s vlastnosťami, pri

ktorých nezáleží na type elementu. Napríklad atribút `position` typu `Point`, ktorý uchováva aktuálnu pozíciu daného prvku. Metóda `getAttachPoint` vracia bod pripojenia hrany pre daný element v závislosti na jeho relatívnej pozícií voči elementu. Zároveň si uchováva informáciu o priamych predchodcoch a následníkoch (`parents` a `children`) v kolekcii typu `ArrayList<>`. Nachádza sa v nej atribút `isSelected` hovoriaci o tom, či je element v danom momente označený. Táto informácia je využívaná plátnom pre zvýraznenie označeného prvku.

Abstraktnými metódami zaväzuje dediace triedy implementovať niektoré funkcie. Keďže elementy sa líšia vo veľkosti, ale aj ich grafickej reprezentácií, musia zdedené triedy implementovať metódy `getWidth`, `getHeight` a `draw`, ktorá vykreslí odpovedajúcu grafickú reprezentáciu. Posledné dve z abstraktných metód sú: `getType`, ktorá vracia textový reťazec s typom elementu a `canAdd` s jediným textovým parametrom `possibleChild`, ktorá rozhodne o tom, či za aktuálnym elementom môže nasledovať element typu určeným týmto parametrom.

Panel vlastností

Panel vlastností je implementovaný ako preddefinovaná `Swing` komponenta prostredia IDEA. To zaručuje, že panel bude vyzerať a správať sa rovnako ako samotné `IDE`. Napríklad pri zmene farebnej schémy IDE bude táto zmena aplikovaná aj na panel vlastností. Je zaregistrovaný z kódu v triede grafického editora `JSLFileGraphicEditor`. To zaručuje, že panel bude dostupný len pre grafický editor a nebude možné ho zobraziť pri žiadnom inom editore. Pri registrovaní panelu je implicitne nastavené jeho ukotvenie na pravú stranu.



Obrázek 5.2: Ukážka editoru výnimiek

Samotný obsah panelu predstavuje trieda `PropertiesPanel`. Je to grafická komponenta rozširujúca triedu `JPanel`. Jeho jedinou grafickou komponentou je tabuľka `PropertiesTable`. Tá rozširuje triedu `JTable` a preťažuje metódy `getCellRenderer` a `getCellEditor` z dôvodu implementovania netriviálnych editorov, tak isto, ako aj potreby zobrazenia hodnôt zložitejších dátových typov. Na obrázku č. 5.1 je ukážka editoru, ktorý je použitý pre tie bunky tabuľky, ktoré v sebe obsahujú výnimky (viď sekcia 2.3.2). Existuje viac netriviálnych editorov a ich implementácia sa nachádza v balíku `designer.ui.properies.editor`. Zobrazovanie týchto zložitejších dátových typov je realizované pomocou výpisu počtu prvkov definovaných pre danú vlastnosť (pre spomínané výnimky je to počet tried výnimiek, napr. "*2 classes set*").

Komunikácia medzi panelom vlastností a plátnom je realizovaná pomocou využitia vnútornej infraštruktúry IntelliJ IDEA pre zasielanie správ (viac informácií na [3]). Plátno zasiela správu o zmene označeného prvku na plátne (element, hrana, koreňový Job Element) a panel vlastností odoberá správy s témou danou rozhraním `SelectionChangedNotifier`.

Samotný panel vykonáva základnú kontrolu pri úprave dát v tabuľke, ako napr. duplicitu identifikátorov elementov a používateľa na ne rázne upozorňuje.

Paleta

Registrovanie komponenty palety elementov je riešené rovnakým spôsobom, ako pri paneli vlastností. Za spomenutie stojí trieda `PaletteComponent`, ktorá reprezentuje vybraný prvok zobrazený na palete. Dôležité je, že implementuje rozhrania `DragGestureListener` a `DragSourceListener`. Obsahuje podtriedu `TransferableText`, ktorá implementuje rozhranie `Transferable`. Predstavuje zdroj dát prenášaných pomocou `DnD` gesta. V tomto prípade je to textový reťazec obsahujúci typ prenášaného elementu.

5.2 Špecifikácia elementov

Hlavným účelom tried obsiahnutých v balíku `specification` je implementácia vlastností elementov definičného súboru popísaných v kapitole č. 2. V hierarchii elementov sa najvyššie nachádza koreňový `Job` Element. Všetky ostatné elementy sú v ňom uložené v kolekcii `elements` typu `ArrayList<Elements>` poprípade vnorené elementom uloženým v tejto kolekcii (napr. elementom typu `Flow`).

Tieto triedy zväčša obsahujú len atribúty reprezentujúce jednotlivé vlastnosti, vnorené elementy a metódy pracujúce s týmito vlastnosťami. Každá z tried musí obsahovať prázdny konštruktor, ktorý je vyžadovaný architektúrou `JAXB`.

5.2.1 JAXB anotácie

Architektúra `JAXB` slúži na mapovanie tried v Jave na ich XML reprezentáciu. Dovoľuje tak isto opätovné vytvorenie tried z ich XML reprezentácie, a teda tento prevod je obojsmerný. Aby bolo možné jednotlivé triedy previesť na XML zápis, sú anotované pomocou jednoduchých anotácií. Atribúty sú označené ako `@XmlAttribute`, elementy ako `@XmlElement`, množiny elementov ako `@XmlElements` a koreňové elementy ako `@XmlRootElement`. Pomocou anotácie `@XmlAccessorType` a jej hodnoty `XmlAccessType.NONE` je pre každú triedu určené, že serializovať sa budú len anotované atribúty.

Koncové prechodové elementy

O niečo zložitejšiu implementáciu vyžadovali triedy koncových prechodových elementov `EndTrainsition`, `StopTransition` a `FailTransition`. Inštancie týchto tried nie sú priamo serializované do XML. Existujú z dôvodu, že k ľubovoľnému z týchto koncových elementov môžu viesť hrany z viacerých elementov. To znamená, že hodnota atribútu `on`, ktorý hovorí po akej hodnote bude nasledovať práve daný koncový prechodový prvok, musí byť nastavená pre každý jeden z elementov zvlášť. Z toho dôvodu tieto triedy obsahujú slovník výskytov jednotlivých `Stop`/`End`/`Fail` prvkov, kde kľúč je samotný element z ktorého vedie hrana k danému koncovému prechodu a hodnota je `Stop`, `End` alebo `Fail` element, ktorý už je serializovaný do XML. To umožňuje mať vytvorený napr. jeden grafický prvok s identifikátorom “End_0“ a hodnotu atribútu `exit-status` totožnou pre všetky jeho výskyty, zatiaľ čo hodnota atribútu `on` sa môže líšiť.

Prídavné definície

V balíku `specification.definitions` sú triedy predstavujúce extra definície prvkov potrebné pre správny chod pluginu. Hlavná trieda je `Definition`, ktorá je rovnako ako trieda elementu `Job` anotovaná ako koreňový element pre serializovanie obsahu do XML formátu pomocou JAXB architektúry. Obsahuje v sebe list špecifikácií elementov (trieda `ElementSpec`) a list prídavných elementov (trieda `AdditionalElement`).

5.3 Ukladanie a načítavanie zo súborov

Operácie spojené s načítavaním a ukladaním dát do súborov sú implementované v triedach balíka `codeGeneration.xml`. Konkrétne ide o tieto tri triedy: `JSLFileGenerator`, `DefinitionFileGenerator` a `JobFileGenerator`. V najbližších riadkoch je stručne vysvetlený ich význam.

Generátor súborov `.jsl` je implementovaný triedou `JSLFileGenerator`. Funguje v dvoch režimoch. V prvom režime za pomoci architektúry `JAXB` mapuje vopred anotované triedy balíka `specifiaction` s koreňovým `Job` elementom do ich serializovanej podoby do formátu XML a ukladá ich práve do `.jsl` súboru. Pri tomto kroku nie sú potrebné žiadne kontroly oproti `XSD`[2] schéme, ani sémantické kontroly. Grafický editor sa stará o validitu spôsobom, že používateľovi povolí vykonať len úpravy, ktoré nie sú v nijakom rozpore so schémou a povolí vytvoriť medzi prvkami len prechody, ktoré sú v povolené jazykom `JSL`. Druhý režim je spätné vytváranie štruktúry tried s koreňovým `Job` elementom z ich textovej reprezentácie, ktorú môže používateľ ľubovoľne upravovať v textovom editore.

Validácia súboru pri otváraní grafického editoru je riešená v triede `JSLCanvas` metódou `loadJobDiagram`. Najprv je vykonaná validácia oproti Java EE 7 schéme pre dávkové spracovanie popisujúcej jazyk `JSL` rozšírenej o atribút `id` pri prvkoch `Stop`, `End` a `Fail`. Následne je vykonaná kontrola "next" referencií, a teda kontrola, či prvok odkazovaný hodnotou atribútu/elementu `next` existuje v rámci daného návrhárskeho kontextu (nie je možné odkazovať sa na prvok "zakrytý" pomocou vnorenia nejakému `Flow` elementu, rovnako ako prvku `Split`).

Trieda `DefinitionFileGenerator` sa stará o generovanie a spätné načítavanie definičného `.jsd` súboru pomocou rovnakej technológie, ako pri prvom popísanom generátore. Avšak v tomto prípade odpadá kontrola tohoto súboru, keďže je považovaný za vnútorný súbor generovaný zásuvným modulom a používateľ si má byť pri zasahovaní do konfiguračného súboru vedomý možných následkov. V prípade absencie tohoto súboru plugin pokračuje vo svojom chode a chýbajúce informácie o pozícii si vygeneruje sám.

Posledný spomínaný je generátor výsledného dávkového súboru. Obsahuje jedinú metódu `generate`, ktorá vykonáva transformáciu súboru `.jsl` na výsledný súbor umiestnený v koreňovom adresári projektu v zložke `META-INF/batch-jobs`. Pod transformáciou sa myslí odstránenie atribútu `id` z koncových prechodových elementov, ktorý nie je popísaný jazykom `JSL`.

5.4 Textový editor

Ako textový editor dávkového .jsl súboru je využitý štandardný editor prostredia IDEA. Jeho výhodou je vstavaná funkcionalita zvýrazňovania syntaxe, refaktorizácie kódu a inte-

[2] *XSD* - súčasný štandardný jazyk určený na popisovanie povolenej schémy (štruktúry) XML dokumentov

ligentného dopĺňania kódu pre jazyk `JSL`. Ako bolo spomenuté v tejto práci, obsah súboru `.jsl` upravovaného týmto editorom je v jazyku `JSL`, rošírenom o atribút `id` pre koncové prechodové prvky. Implementácia takejto zmeny je jednoduchá. Stačí dynamicky rozšíriť existujúci DOM model o spomínaný atribút pomocou vstavaného nástroja prostredia IDEA (`DomExtender`). Bohužiaľ, DOM model pre jazyk JSL nie je v rámci IntelliJ IDEA zverejnený pod otvorenou licenciou. Z toho dôvodu je využitý štandardný editor bez rozšírenia `DOM` modelu aj za cenu zvýraznenia spomínaného atribútu `id` ako chybného.

Kapitola 6

Testovanie

Zásuvný modul bol testovaný na operačnom systéme Windows 8.1 Pro v 64 bitovej verzii a na operačnom systéme Linux Ubuntu 14.04 rovnako v 64 bitovej verzii. Funkčnosť naprieč rozdielnymi operačnými systémami by mala byť zabezpečená samotnou prenositeľnosťou prostredia IDEA, ale grafické komponenty súpravy nástrojov Swing môžu vyzerať rozdielne naprieč platformami. Z toho dôvodu bol zásuvný modul testovaný na oboch spomínaných operačných systémoch.

Testovanie funkcionality zásuvného modulu bolo realizované na veľkom množstve dávkových súborov s rôznymi kombináciami využitých vlastností prvkov. Pre príklad bude uvedených 5 testovacích sád súborov s rozličnými dávkovými úlohami (viď príloha A).

Prvý testovací súbor je prevzatý z dokumentu The Java EE 7 Tutorial [4]. Ide o jednoduchý príkad dávkového súboru s dvomi prvkami `Step`, ktorého úlohou je zaznamenávať vykonané hovory a k nim odpovedajúce účty. Navyše k súboru `.jsl` neexistuje odpovedajúci definičný `.jsd` súbor.

Druhý súbor je zameraný na otestovanie rôznych možností definovania nasledujúceho "next" elementu. Obsahuje tri príklady prvkov `Step` pre každú z možností: jediný `next` atribút, jeden `next` element s uvedenou hodnotu, po ktorej prechod nastane a element s viacerými `next` elementmi.

V treťom súbore vzniká viacero návrhárskych kontextov vďaka dvom `Flow` elementom a im vnoreným elementom. Ďalej je uvedený jeden `Split` element, ktorý obsahuje spomenuté dva `Flow` elementy. Jeden z nich obsahuje ukončovací `Fail` prvok.

Štvrtá testovacia sada elementov obsahuje vzájomne zanorené `Flow` prvky vo viacerých úrovniach pre otestovanie hlbšieho zanorenia. Tak isto je otestované jednoduché definovanie prvkov `listeners`, `checkpoint-algorithm` ale aj `partition`, konkrétne prvku `plan` a jeho vlastností.

Posledný súbor je naozaj komplikovanou dávkou úloh, ktorý kombinuje takmer všetky dostupné nastavenia prvkov.

Všetky uvedené súbory boli zásuvným modulom spracované bez zistených problémov. Neboli pozorované žiadne obmedzenia z pohľadu dĺžky spracovania. Rovnako vykresľovanie zložitejších štruktúr bolo bezproblémové. Avšak pri rozsiahlejších diagramoch by sa hodila možnosť editovania celých skupín elementov, čo by mohlo byť predmetom budúceho rozšírenia. Ďalej pri povolenom rozdeľovaní kroku na partície pri `Step` elemente, pri definovaní vlastností týchto partícií a následnom zrušení týchto partícií sú nastavené údaje stratené. Návrhom na zlepšenie by bolo uloženie týchto dát v definičnom `.jsd` súbore, pretože opätovné nastavenie veľkého počtu vlastností prvku `partition` môže byť pracné.

Kapitola 7

Záver

Cieľom tejto bakalárskej práce bolo vytvorenie zásuvného modulu do vybraného vývojového prostredia. Po zvážení všetkých možností bolo vybrané práve vývojové prostredie IntelliJ IDEA. Vyvíjaný zásuvný modul by mal fungovať ako nástroj umožňujúci graficky editovať štruktúru dávkových úloh v Java EE 7. Dôležitou požiadavkou bola možnosť obojstrannej transformácie zmien medzi grafickou a textovou reprezentáciou štruktúry dávkových úloh v jazyku JSL.

V úvode som sa musel zoznámiť s architektúrou Batching API a tak isto so spôsobom jej využitia v Java EE, pre ktoré je dostupná od verzie 7. Musel som sa dôkladne zamerať na jednotlivé elementy tvoriace výslednú štruktúru dávkových úloh. Následne som venoval dlhší čas naštudovaniu architektúry vývojového prostredia IntelliJ IDEA, s ktorým som predtým nemal žiadne skúsenosti. Pod dohľadom vedúceho práce a s využitím priebežných konzultácií som vypracoval návrh výsledného zásuvného modulu. Dôraz bol kladený na jednoduchosť a intuitívnosť spojenú so zachovaním základnej idey a využitia postupov pri práci s nástrojom, ktoré sú bežné pre zásuvné moduly podobného zamerania.

Výsledkom realizácie návrhu je nástroj, ktorý výrazne zjednodušuje programátorovi prácu spojenú so základným návrhom štruktúry dávkových úloh a navyše umožňuje aj pokročilé možnosti úprav vlastností jednotlivých elementov. V podstate pokrýva celú škálu úkonov spojených s vytváraním a úpravou dávkových súborov.

7.1 Ďalšie možnosti práce

V predchádzajúcej kapitole boli spomenuté rozšírenia plynúce z testovania zásuvného modulu. Ďalším vhodným rozšírením, ktoré by programátorovi uľahčilo značný kus práce by bola možnosť vygenerovania tried implementujúcich jednotlivé elementy. Užitočná by bola aj možnosť presúvania celých podgrafov, v ktorých by nebol detekovaný žiadny cyklus. Podobne by bolo vhodné implementovať aj možnosť jednoducho meniť dizajnérsky kontext pre celé podgrafy. Možným zvýšením komfortu pri práci s elementmi na plátne by bolo pridanie rozpoznávania umiestnenia okolitých elementov a prenášaný element prichytávať na zaujímavé pozície (rovno pod, vedľa atď.).

Vytvorenie pokročilého grafického nástroja vyžaduje veľké množstvo vykonanej práce a tak isto poskytuje neustále možnosti, ako vytvorený nástroj zdokonaľovať a rozširovať jeho funkcionalitu. Je zverejnený pod open-source licenciou, a preto je cesta k jeho ďalšiemu zdokonaľovaniu otvorená.

Literatura

[1] FIELDS, Duane K., et al. *IntelliJ IDEA in action*. London: Pearson Education [distributor], 2006. 516 s. ISBN 978-193-2394-443.

[2] JEREMOV, D. *IntelliJ IDEA Action System* [online]. 2010, [cit. 2015-05-15]. Dostupné z: https://confluence.jetbrains.com/display/IDEADEV/IntelliJ+IDEA+Action+System.

[3] ZHDANOV, D. *IntelliJ IDEA Messaging infrastructure* [online]. 2011, [cit. 2015-05-15]. Dostupné z: https://confluence.jetbrains.com/display/IDEADEV/IntelliJ+IDEA+Messaging+infrastructure.

[4] JENDROCK, E., et al. *The Java EE 7 Tutorial* [online]. 2014, 980 s. [cit. 2015-05-16]. Dostupné z: http://docs.oracle.com/javaee/7/JEETT.pdf.

[5] VIGNOLA, C. *JSR 352 Batch Applications for the Java PlatformIntelliJ IDEA Action System*. 2012, 116 s.

[6] GUPTA, A. *Java EE 7 essentials*. 2013, 343 s. ISBN 14-493-7017-9.

[7] *IntelliJ IDEA 14.1.1 Help: Project* [online]. 2015, [cit. 2015-05-16]. Dostupné z: https://www.jetbrains.com/idea/help/project.html.

Příloha A

Obsah CD

Priložené CD obsahuje nasledujúce súbory a adresáre:

- /text/bp.pdf - elektronická verzia písomnej správy
- /text/tex/ - zdrojové súbory textu bakalárskej práce
- /src/ - zdrojové súbory zásuvného modulu
- /plugin/ - obsahuje .jar súbor s funkčným zásuvným modulom
- /manual/manual.pdf - návod na inštaláciu a jednoduchý popis používania
- /batch-samples/ - ukážkové dávkové úlohy