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

Bakalářská práce

Nástroj pro grafický popis procesů vývoje software

Plzeň 2017 Václav Janoch

Prohlášení

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

V Plzni dne 3. května 2017

Václav Janoch

Poděkování

Rád bych poděkoval Ing. Petru Píchovi za vstřícnost, trpělivost, odborné rady a cenné připomínky, které mi pomohly tuto bakalářskou práci vypracovat.

Abstract

The purpose of this bachelor thesis is to create a graphical tool for administration of the processes of software development, based on SPADe datamodel. SPADe (Software Process Antipatterns Detectror). SPADe is a tool developed at the University of West Bohemia in Pilsen and it is used for data collection pro Application Lifecycle Management tools. It then searches for patterns and anti-patterns in the data. In the first part of the thesis, there are requirements on the application along with an analysis of existing applications for software development process administration. The second part describes the usable resources for the implementation of the application, description of the implementation itself and a user manual for the application.

Abstrakt

Cílem této bakalářské práce je vytvořit grafický nástroj pro správu procesů vývoje software (Aplikaci), založený na datovém modelu SPADe. SPADe je nástroj vyvíjený na Západočeské Univerzitě v Plzi a slouží ke sběru dat z ALM nástrojů, ve kterých dále hledá patterny a anti-patterny. V první části práce jsou sepsány požadavky na aplikaci spolu s analýzou již existujících aplikací pro správu procesů vývoje software. Druhá část práce popisuje prostředky využitelné pro implementaci aplikace, popis samotné implementace a uživatelský manuál k aplikaci.

Obsah

1 Úvod

Cílem této práce je vytvořit grafický nástroj pro popis procesů (dále jen grafický nástroj) podle datového modelu nástroje Software Process Anti-pattern Detector (SPADe).

SPADe je nástroj umožňující získávání dat o procesech vývoje software, ze kterých následně vyhledává informace o procesních vzorech (patternech) a často opakovaných chybách (anti-patternech). Z těchto dat dále vytváří statistiky pro projektové managery. V textu práce se nachází jeho podrobnější popis, vysvětlení důvodů pro vznik nástroje a popis jeho vlastností. Napojení této práce na SPADe a důvod jejího vzniku, spolu s funkčními požadavky na grafický nástroj.

V další části práce je stručný popis a porovnání dvou nástrojů pro tvorbu modelových procesů, Eclipse Process Composer (EPF Composer) a Rational Method Composer (RMC). Vysvětlení důvodů vedoucích k nemožnosti využití těchto nástrojů pro SPADe a popis funkčních vlastností možných pro převzetí do grafického nástroje.

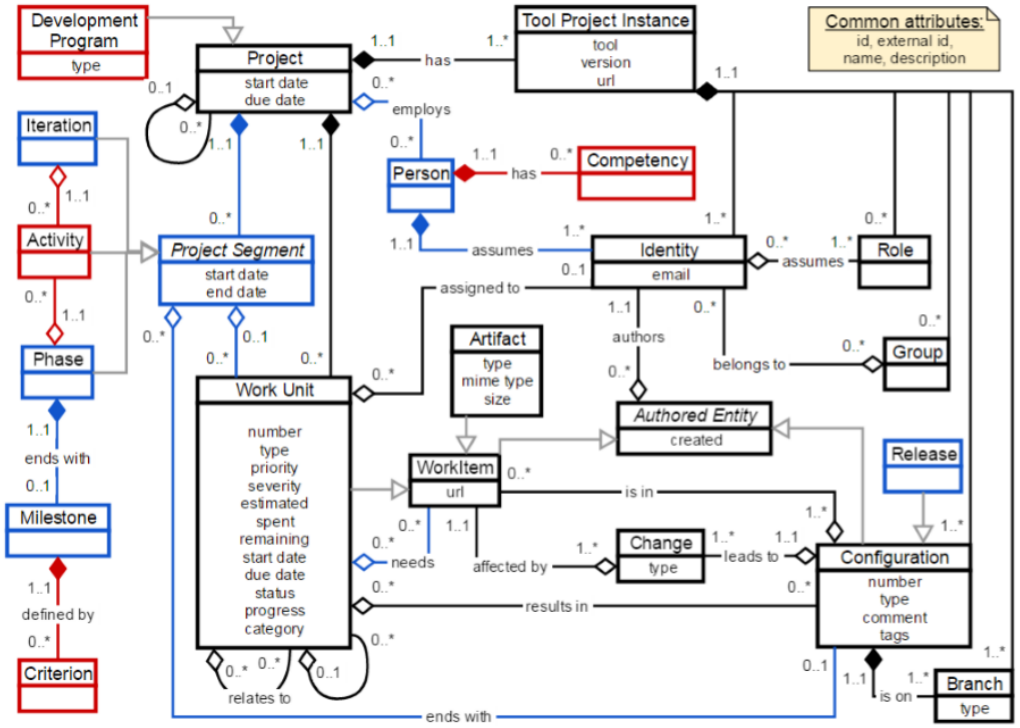
Aplikace implementována v rámci této práce umožňuje zpracovávat procesy uložené pomocí XML. V textu práce se nachází krátký popis XML a jsou zde podrobněji popsány základy obecných metod jeho zpracování, jednotlivé rozdělení parserů a jejich vlastností. V další části je popsáno zpracování XML samotným programovacím jazykem Java. Následuje popis frameworku JavaFX, pomocí kterého je vytvořeno grafické rozhraní nástroje. Návaznost na předchozí nástroje tvorby Graphic User Interface (GUI).

V poslední části práce se nachází popis implementace grafického nástroje spolu s jeho uživatelským manuálem a popisem testování funkčnosti aplikace.

2 Nástroj SPADe

Software Process Anti-pattern Detector (SPADE) je nástroj vyvíjený na Západočeské univerzitě v Plzni (ZČU), Fakultě aplikovaných věd (FAV), Katedře informatiky a výpočetní techniky (KIV). Účelem nástroje je dolování dat o projektech vývoje software z Application Lifecycle Management (ALM) nástrojů a jejich následné ukládání do jednotného datového skladu a identifikace anti-patternů (často opakovaných chyb) projektového řízení, viz [15]. SPADe by měl následně informaci o výskytu anti-patternu předat dále ve srozumitelné formě projektovému manažerovi nebo vedoucímu vývojového týmu a zároveň s ní poskytnout rady jak anti-patrerny odstranit či jinak řešit na základě zkušeností z již uzavřených projektů.

Tato práce umožní rozšíření nástroje SPADe o grafický editor modelů procesů, popřípadě anti-patternů. Pro zpracování dat bude využíván datový model SPADe (doménová forma, viz obrázek 2.1), viz [14]. Modely budou zpracovány pomocí jazyka XML, jehož správné složení bude validováno vytvořeným XSD schématem.



Obrázek 2.1: Doménový datový model databáze nástroje SPADe

3 Funkční požadavky na nástroj

Modelátor bude napsán v programovacím jazyce Java, ve kterém se zpracují i XML dokumenty s anti-paterny nebo procesy. Grafická část programu bude zpracována frameworkem JavaFX.

Mezi hlavní funkční požadavky patří vytváření a vkládání segmentů Activity(skupinu spřízněných úkolů se společným cílem, viz [14]), Iteration, Phase, viz [16] a elementů Work Unit(entita zachycující ticket/issue v nástroji pro změnové řízení), Change, Artifact na kreslící plátno, po kterém je lze libovolně přemisťovat. Pro tyto prvky vytvořit editační formuláře, umožňující vyplnění informací o daném prvku, vyvolané dvojklikem na ikonku vykreslenou na plátně. Umožnit vytváření dvou typů relací mezi vybranými prvky plátna představující elementy. Neorientovanou relaci mezi elementy Change a Artifact a dále pak relaci, u které lze zvolit pojmenování a vyobrazení směru spojení mezi elementy Work Unit. Následně pak možnost odstranění určité relace dvojiklikem na ni. Diagram závislostí segmentů a elementů na obrázku 3.1.

Dalším požadavkem je vytváření drag and drop elementů i ve formulářích s informacemi. V kořenovém elementu, představujícím celý projekt, vytvářet segmenty zmíněné výše a element Work Unit. V těchto segmentech vytvářet pouze Work Unit s možností vytvoření relace mezi více prvky. V elementu Configuration umožnit vytváření pouze Change a Artifact, mezi kterými lze následně vytvářet neorientovanou spojnici. Navazujícím požadavkem bylo omezit vkládání nesprávných prvků do vnitřních pláten formulářů a s tím související funkčnosti, kopírování, vyjmutí, odstranění prvku z plátna a případně jeho vložení do plátna. Následným požadavkem bylo umožnit volbu existence a neexistenci elementů Work Unit, Change a Artifact, dále pak vytvořit barevné rozlišení mezi těmito typy prvků v projektu.

Dalším požadavkem bylo vytvářet jednoduchým způsobem ostatní elementy procesu vývoje, jako jsou Milestone, Criterion, Role, Tag, Configuration-Person-Relation(entita popisující vztah mezi Role a Configuration) a Branch, spolu s přehledovou tabulkou a možností smazání dat z ní. Přenášet jména

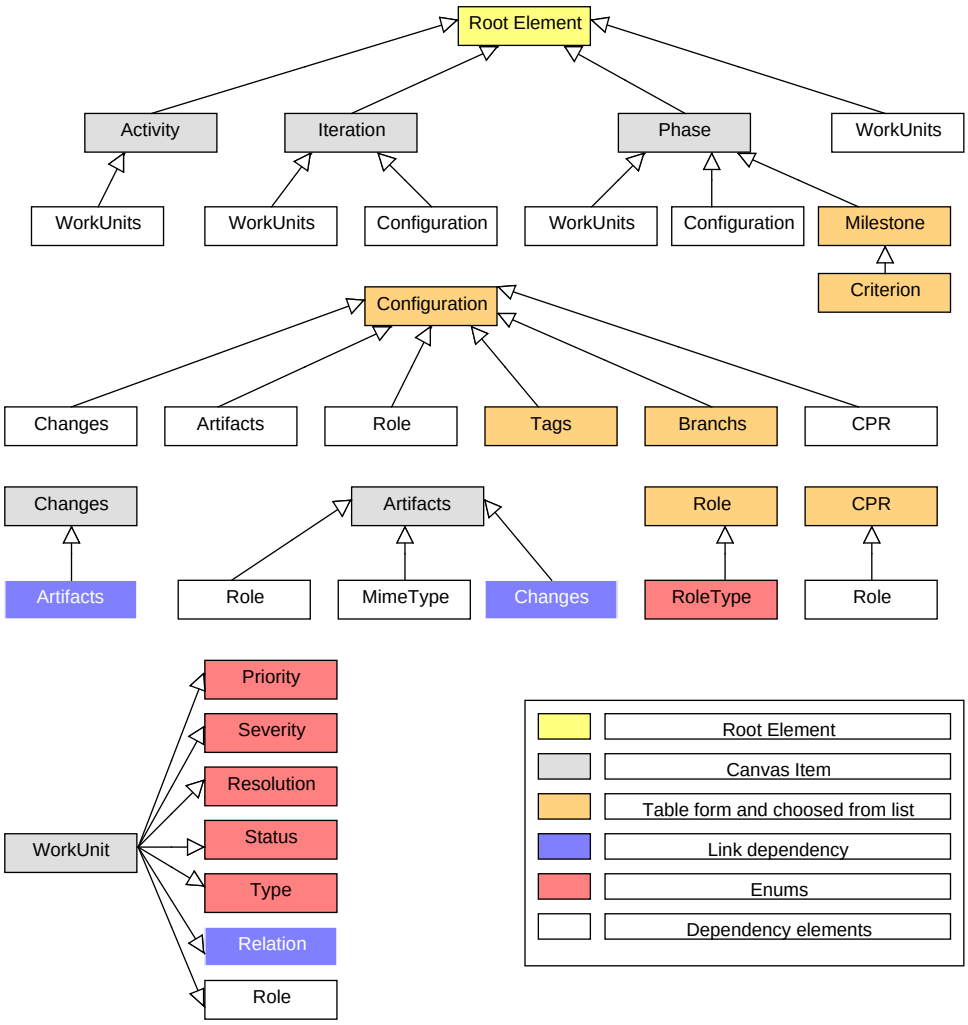
daných elementů do výběrových seznamů uvnitř ostatních segmentů a elementů.

Poslední požadavek na funkci formulářů je automatické mapování výčtových typů Priority, Severy, Resolution, Relation, Status, Type mezi Class a Super Class , umožnění volby pouze typu Super Class[1]. Spolu s tou funkčností, mít možnost definovat ve všech modelech (a následných XML) explicitní rozlišení hodnoty 0, prázdný řetězec, null.

Podle XSD schématu, vytvořeného dle datového modelu nástroje SPADe, umožnit ukládání vytvořených modelů do XML formátu, stejně tak jako zpětné načítání vytvořených procesů z XML dokumentu do aplikace, za pomoci XSD schématu, umožnit validaci vytvořeného modelu oproti XSD schématu s vypsáním krátkého popisu případného problému.

Jako další požadavek bylo tvořit kontextové menu s položkami New, Open, Save a Validate pro volání funkcí popsaných o odstavec výš.

[1]Jedná se o sjednocené klasifikační schéma výčtových hodnot napříč různými zdroji dat (ALM nástroji)



Obrázek 3.1: Diagram závislostí segmentů a elementů procesu vývoje

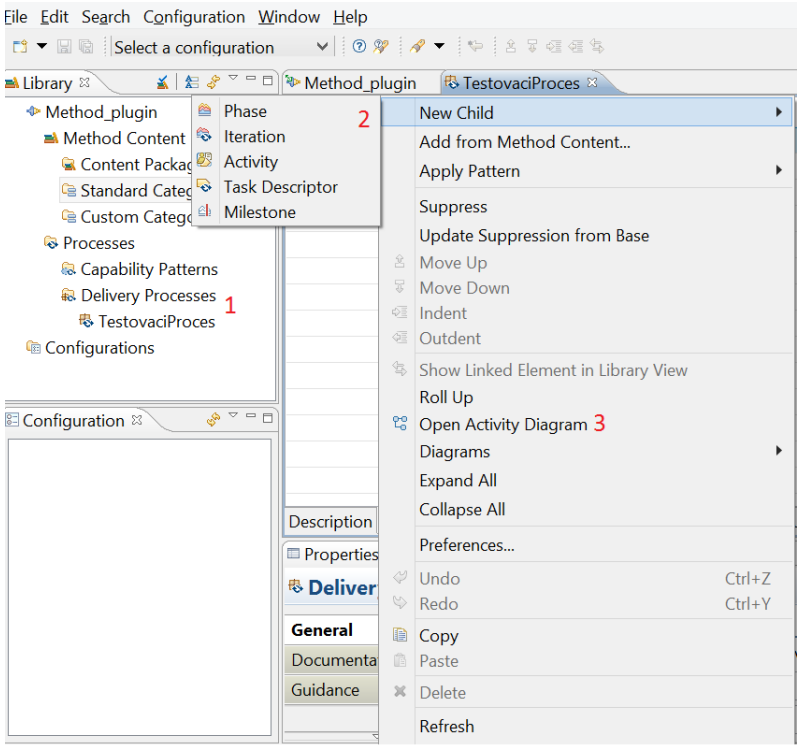
4 Existující nástroje pro tvorbu procesu

V této kapitole budou popsány dva existující specializované softwarové nástroje pro modelování a definování procesů vývoje software, které byly zvažovány pro využití ve SPADe projektu. A to Eclipse Process Framework (EPF) Composer a IBM Rational Method Composer (RMC). Tyto nástroje vznikly za účelem efektivnějšího, přehlednějšího a srozumitelnějšího způsobu modelování, úpravy a uzpůsobování procesů vývoje software obsahujících velké množství elementů pro účely konkrétních projektů nebo organizací.

4.1 Eclipse Proces Framework Composer

Eclipse Proces Framework Composer (EPF) je framework pro správu softwarových procesů, vytváření metod procesů, zprávu knihoven, konfigurace a zveřejnění procesů, viz [7]. Podporuje iterativní, agilní a inkrementální vývoj, použitelný ve velkém množství vývojových platforem a aplikací. EPF je primárně určen pro projektové manažery nebo vedoucí vývojových skupin, kteří pomocí frameworku mohou hromadně upravovat vlastnosti projektu nebo ho vyexportovat do HTML podoby. EPF také umožňuje uživateli použít grafický editor modelů pro návrh procesu. Grafický nástroj využívá také funkci drag and drop, tedy jednotlivé prvky procesu umožňuje jednoduchým přetažením na plátno umístit do modelovaného procesu. U těchto prvků umožňuje měnit jejich formát, velikost písma, barvu textu atd. V kontextu práce je nejzajímavější právě tato modelovací část jako inspirace pro vytvářený nástroj.

Pro spuštění drag and drop editoru je potřeba vytvořit tzv. knihovnu library, do které se vytvoří nová procesní metoda a v této metodě lze následně vytvořit modelový proces. Editor lze spustit volbou Open Activity Diagram(3), viz obrázek 4.1. Po načtení editoru modelu je vytvořeno plátno, na které se po vybrání elementu nacházejícího se v pravé nabídce nazvané Pallete dají tyto elementy jednoduchým klikem vytvořit, nabízené elementy na obrázku 4.2. Editační okno umístěné do spodní části obrazovky slouží k vyplňování informací o jednotlivých elementech pomocí vstupních polí, viz obrázek 4.3. Výsledný namodelovaný proces lze jednoduchým způsobem



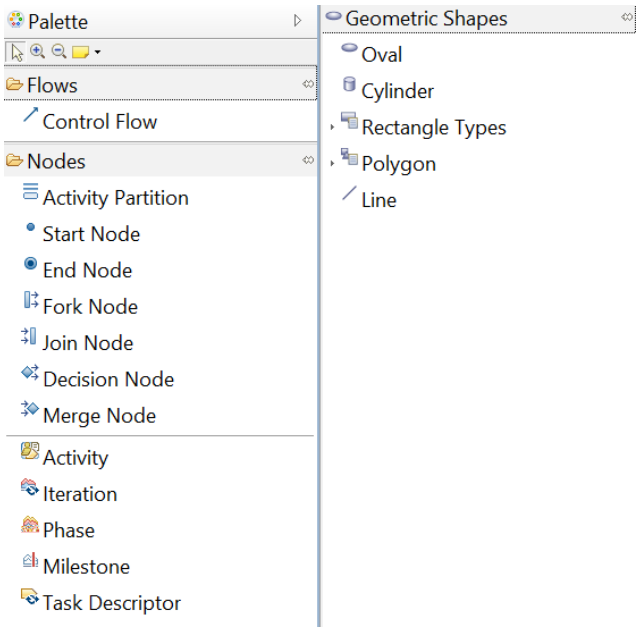
Obrázek 4.1: Spuštěni grafického návrháře

exportovat do HTML formy. To umožňuje záložka Configuration v horním menu a možnost Publish. Je požadováno zadání jména HTML souboru a volby static WEB, po vyplnění těchto údajů a stisknutí Finish se na zadaném místě vytvoří HTML model. Struktura HTML stránky je na obrázku 4.4.

4.2 IBM Rationl Method Composer

„Produkt IBM Rational Method Composer je pružná platforma pro správu procesů s nástrojem pro tvorbu metod a knihovnou procesů, jež v rámci podniku napomáhá implementovat měřitelná zlepšení, návrhy systémů a procesy poskytování softwaru. Nástroje Rational Method Composer umožňují vytvářet, upravovat, spravovat a publikovat popisy procesů. Knihovny procesů a postupů poskytují nejlepší postupy, které můžete přímo využívat v dodané podobě, nebo podle potřeby upravovat v rámci vytváření vlastních procesů.“[1], více informací viz [3]

RMC je v mnohém podobný EPF, který je v podstatě jeho zjednodušenou, open-source formou. Je distribuován pomocí vývojového prostředí Eclipse,

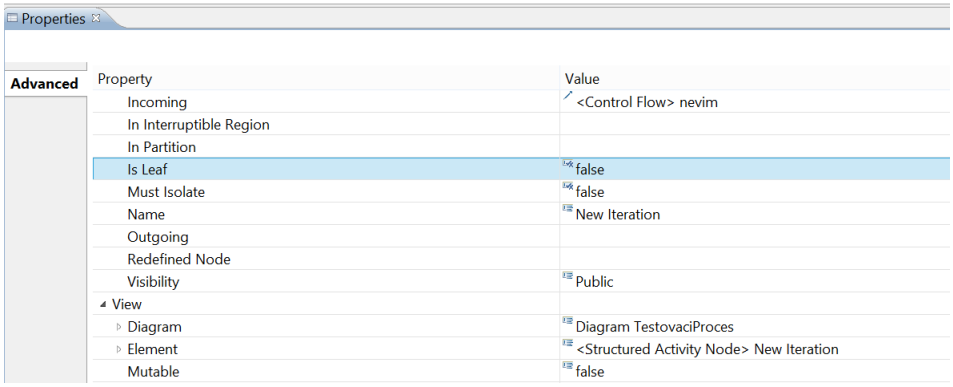


Obrázek 4.2: Nabídka Pallete

do něhož je přímo integrován. Distribuce obsahuje knihovnu s kompletními popisy několika procesů, úkolů, rolí a dalších elementů, které mohou uživatelé RMC použít k vytvoření vlastních modelů procesů. Stejně jako u EPF Composeru se i zde bude text zabývat pouze grafickým editorem procesů, který poslouží jako inspirace pro implemetaci aplikace, která je předmětem této práce.

Logika zacházení s RMC je velmi podobná ovládání EPF Composeru. Do zvolené knihovny se vytvoří Method plugin, ve kterém se vytvoří daný proces. V nově vytvořeném procesu existuje možnost vytvoření různých elementů například Activity, Task Descriptor a další.

Při vytváření některých elementů je nutno daný element definovat za pomoci dvou kroků. V případě Task Descriptor musí být nejprve definován samotný Task, ze kterého lze následně vytvářet Descriptor jako jeho upravitelnou kopii, viz [13] Tyto elementy lze vytvořit i pomocí grafického editoru procesů. Editor je spuštěn po zvolení Open Activity Diagram viz obrázek 4.5. Editor je tvořen plátnem v centrální části obrazovky a seznamem dostupných elementů, který se nachází v pravé části v nabídce s označením Pallete. RMC umožnuje export výsledného modelu procesu vyexportovat třemi různými způsoby. V tomto spočívá jedna z odlišností od EPF Composeru, který dokáže export pouze do HTML. Naproti tomu RMC umožňuje export i do

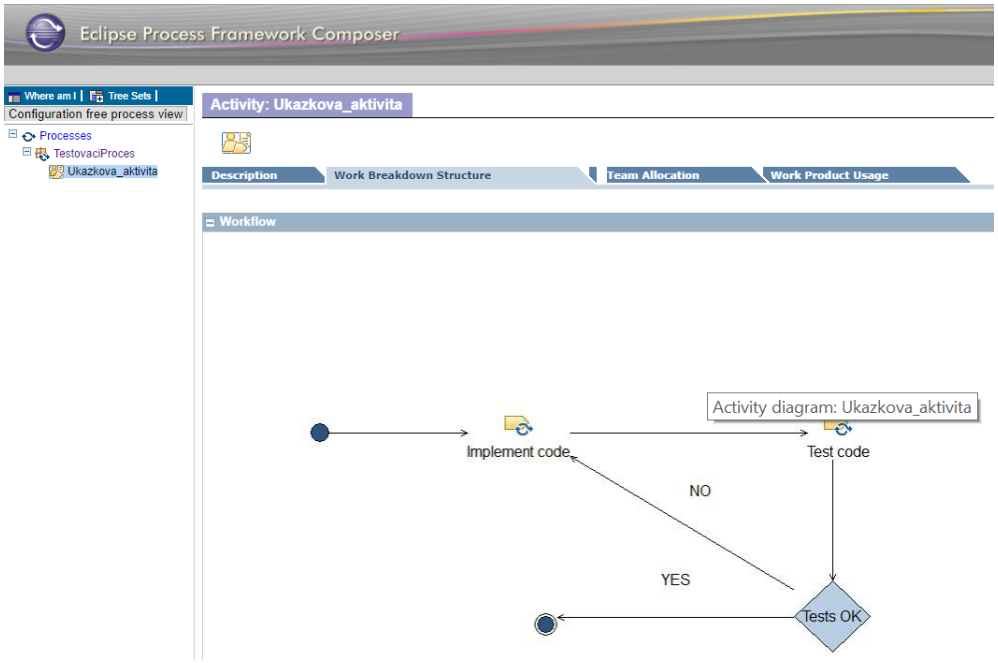


Obrázek 4.3: Editační okno

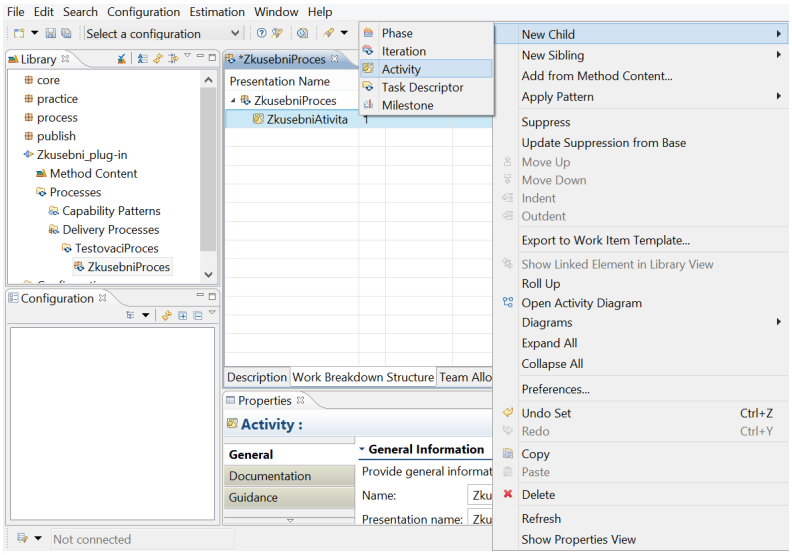
formátů Adobe PDF nebo Microsoft Word. Export do těchto dvou formátů může trvat značně dlouhou dobu v závislosti na velikosti modelu, editační okno viz obrázek 4.6.

4.3 Důvody nemožnosti využití existujících nástrojů

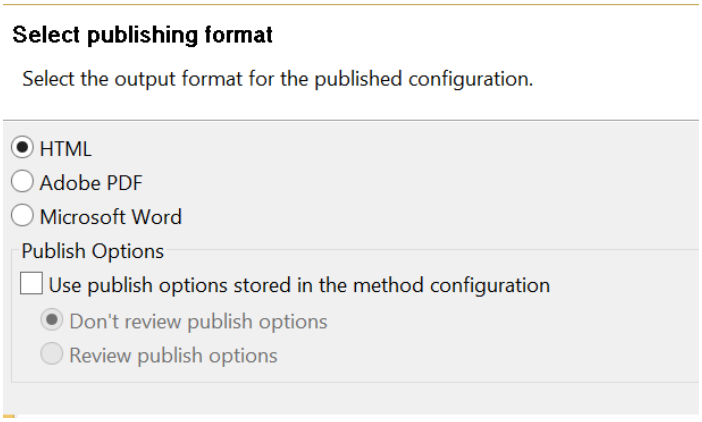
Široká škála využití těchto nástrojů, znemožňuje jejich přímé využití s nástrojem SPADe. Oba nástroje zmíněné v předešlých kapitolách mají odlišné modely oproti datovému modelu SPADe. Nástroje umožňují využívání velké množiny nadstavbových funkcí, nepotřebných při modelování datového modelu SPADe, stejně jako velké množství kroků pro vytvoření grafického editoru modelů. Dále pak při modelování procesu dle SPADe je potřeba vytvářet jednak patterny, tak anti-patterny, vyznačující neexistenci některých prvků, což RMC ani EPF Composer neumožňují. Možné vlastnosti pro převzetí jsou: vytváření prvků plátna pomocí drag and drop funkce prvku nebo jednoduchým klikem na daný prvek, vyvolání editačního okna po kliku na prvek.



Obrázek 4.4: Webová prezentace procesu



Obrázek 4.5: Prostředí RMC



Obrázek 4.6: Export v RMC

5 Analytická část

5.1 Popis XML, DTD a XSD

Jazyk eXtensible Markup Language (XML) patří do oblasti značkovacích jazyků pro uchování a zpracování dat ve formě textových dokumentů. Předchůdcem XML je jazyk SGML (Standard Generalized Markup Language) a velmi se podobá celosvětově používanému jazyku Hypertext Markup Language (HTML), který z něho vychází. Je standardem World Wide Web Consortium (W3C) a jedním z nejdůležitějších formátů výměny strukturovaných dat (užívá se mj. u webových služeb), viz [17]. Nejvíce ceněnou vlastností je platformní nezávislost XML způsobená faktem, že obsah XML dokumentu je pouze textová informace. Jeden z nejvýznamnějších rozdílů oproti HTML je možnost definice vlastních značek včetně atributů a omezení hodnot. Tyto značky mají pevnou gramatiku, to znamená, že je při jejich definici nutné striktně dodržovat několik základních pravidel.

Document Type Definition (DTD) patří do skupiny jazyků pro popis schématu (struktury) XML dokumentu. To znamená, že vymezuje množinu jednotlivých značek (elementů) a atributů. Popisuje, jak mohou být značky navzájem uspořádány a vnořeny. DTD je vhodný pro textově orientované dokumenty, méně pak pro datově orientované dokumenty. Mezi největší výhody DTD patří jeho kompatibilita s většinou aplikací, tato výhoda je dána spoluprací s XML skoro od samého začátku. DTD popis není tvořen samotným XML, ale specifickou syntaxí. Tato syntaxe neumožňuje popsat datové typy, jejich rozsahy atd. DTD nemá žádnou možnost práce se jmennými prostory, a proto je velice nevhodný pro velmi obsáhlé XML dokumenty, viz [8].

XML Schema Definiton (XSD) je v současnosti nejvýznamnější schémový jazyk pro XML. Výhoda XSD oproti DTD spočívá ve využívání XML konvence, což umožňuje velmi ceněnou vlastnost, a to validaci proti sobě samému. Proto lze zkontrolovat samotné složení a obsah elementů v XSD. XSD dále podporuje definování datových typů i s nadefinováním jejich rozsahů, dále umožňuje podporu jmenných prostorů. Nejvhodnější napojení XSD schématu na XML dokument je uvést ho v kořenovém elementu XML, viz [11] Následujícím způsobem:

```
<xmlns:xsi=http://www.w3.org/2001/XMLSchema–instancexsi:
    noNamespaceSchemaLocation="naseCestaKXSD.xsd">
```

Tento postup je obdobný jako u DTD. Další napojení XSD lze uskutečnit pomocí ručního výběru nebo automatickým připojením schématu podle jmenného prostoru a jména kořenového elementu.

5.2 Způsoby zpracování XML

Programovací jazyky umožňují zpracování XML souboru pomocí parserů. Parsery jsou do jazyka většinou implementovány pomocí knihoven. Do obecných vlastností parseru by měla patřit kontrola správné strukturovanosti (well-form). Tato kontrola umožní lokalizovat daný problém v syntaxi XML dokumentu a přesně na něj upozornit. Dále by měla být umožněna validace pomocí DTD nebo XSD. Parser přes rozhraní Application Programming Interface (API) vytváří infoset, neboli abstraktní model XML dokumentu. Pomocí modelu se nepracuje s XML jako s obyčejným textovým souborem, ale s jednotlivými koncepty XML, jako jsou elementy, atributy a jejich hodnoty, viz [8].

Rozhraní lze rozdělit na dva typy. Prvním typem je proudové čtení. Tento typ parseru postupně čte XML soubor a vyvolává události nad každou ucelenou částí. Při využití takového rozhraní je v programu nutno implementovat obsluhu těchto událostí. Sekvenční zpracování souboru znamená, že při použití proudového čtení XML se nelze vracet. Všechny načtené údaje se musí průběžně zpracovávat nebo ukládat do námi vytvořené struktury v paměti. Dalším problémem tohoto druhu zpracování je potřeba velkého množství pomocných proměnných nebo vlastní nadstavby. Nejznámějším představitelem tohoto druhu zpracování je parser SAX.

Dalším druhem zpracování je stromová reprezentace dokumentu, kdy se XML dokument celý načte do stromové struktury v paměti. Tento způsob zpracování přináší značné výhody oproti první možnosti. Za velkou výhodu lze považovat možnost vytvářet nový XML soubor, nebo měnit již existující. Při zpracování údajů je nejvíce užitečná možnost libovolně se pohybovat po infosetu, který se nachází celý uložený v paměti pomocí stromové struktury. Nevýhodou tohoto parseru je pomalé načítání do paměti a velká paměťová náročnost pro uchování dat ve stromu. Proto tento způsob není vhodný pro zpracovávání velkých XML souborů. Základním představitelem toho typu zpracování je parser Document Object Model (DOM) od společnosti W3C.

Parsery se také dělí do skupiny dle vlastnosti validace pomocí XSD nebo DTD. Pokud parser patří do skupiny nevalidujících parserů, tak ani pokud je zadáno XSD nebo DTD schéma, není na tato schémata při zpracování brán ohled.

5.3 Zpracování v Java

Pro práci s XML lze v programovacím jazyce Java použít několik rozhraní parserů. Jedním z nich je například rozhraní Java API for XML Processing (JAXP). Rozhraní JAXP, začleněné do Java Core API, je jedno z nejdůležitějších API. JAXP sjednocuje způsoby používání již existujících parserů, umožňuje programátorovi jednotným způsobem využívat různé metody zpracování i různé typy parserů a odstiňuje ho od drobných odlišností parserů a jejich verzí. Pro sofistikovanější zpracování XML souborů lze pomocí JAXP změnit vlastnosti parseru, a to ve třech úrovních, viz [8].

Do první úrovně spadá například zapnutí, respektive vypnutí validace pomocí XSD nebo DTD a také zapnutí možnosti využívat jmenné prostory. Tato úroveň je přenositelná mezi jednotlivými parsery a nezáleží, který je zrovna používán. Při zapnutí validace pomocí XSD schématu je použito XSD, které je připojeno do XML dokumentu, viz následující segment kódu.

```
saxPF.setValidating(true);
saxPF.setNamespaceAware(false);
```

Druhá úroveň ovlivňuje samotné chování parseru. Nastavení jednotlivých vlastností se provede pomocí URL, které musí být přesné a nestačí tak pouze použít jméno dané vlastnosti. Tato úroveň také spadá do kategorie přenositelných a je nezávislá na parseru, viz následující segment kódu.

```
saxPF.setFeature("http://xml.org/sax/features/namespace",true)
saxPF.setFeature("namespace",true)
```

U třetí úrovně je nutné znát parser, u kterého budou měněny jeho vlastnosti, jelikož je nepřenositelná a nastavuje vlastnosti tomuto danému parseru. Tato úroveň by neměla být měněna bez vážnějších důvodů.

```
saxPF.setFeature("http://apache.org/xml/features/
validation/schema",true);
```

Druhým rozhraním je Java Document Object Model (JDOM), viz [2]. Toto rozhraní vzniklo pouze pro Javu jako snaha o zjednodušení obecného DOM. JDOM využívá vlastnosti Javy jako například přetěžování metod nebo kolekce. Pracuje tak, jak by uživatel Javy očekával, viz [8]. Například pro

vyvolání obsahu elementu stačí nad tímto elementem zavolat metodu getTex(). JDOM umožňuje rychlou práci se soubory XML i nízkou paměťovou náročnost pro operace nad soubory.

5.3.1 Proudové zpracování XML

SAX

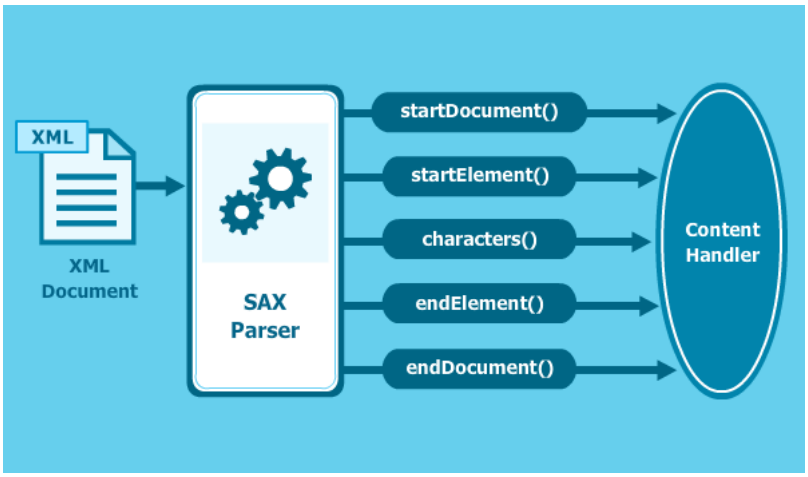
Java umožňuje práci se Simple API for XML (SAX) přes rozhraní JAXP a využitím parseru Xerxes a balíků org.xml.sax. Objekt SAX lze použít jako samotný verifikátor pro dodržení správného formátu XML, také mu lze jednoduchým způsobem předložit XSD nebo DTD soubor pro validaci dat obsažených v XML souboru. Objekt lze použít ke čtení XML dokumentů, ale nikoli pro jejich vytváření nebo úpravu. SAX stejně jako ostatní parsery reaguje na tři typy výjimek:

- výskyt chyby při parsování souboru,
- výjimky vzniklé při nastavování vlastností,
- výjimky vzniklé při čtení souboru.

Omezení SAX spočívá v nutnosti použití textového bufferu, do kterého se ukládají hodnoty elementů, protože hodnota může přijít po částech, nikoli nutně jako celek, například místo hodnoty 1000 uchovávané v některém z elementů může být načtena pouze hodnota 100. Také je nutné testovat, zda je parser uvnitř konkrétního elementu nebo mimo, například kvůli znakům odřádkování. SAX je typu push-parser, to znamená, že připravené metody jsou volány samotným parserem. Problém může nastat při zpracování velkého XML souboru, ve kterém se využívá větší množství elementů, pro které je nutno napsat stejné množství konstrukcí pro větvení programu na jejich vyhodnocení. Proces zpracování XML pomocí SAX je zobrazen na obrázku 5.1.

StAX

Streaming APIs for XML (StAX) je rychlé a jednoduché rozhraní pro sekvenční čtení a zápis XML souborů. StAX umožňuje pracovat dvěma způsoby. První je kurzorový, rychlý a nízko úrovňový, využívající rozhraní XMLStreamReader a XMLStreamWriter. Druhý způsob je událostní. Oproti kurzoro-



Obrázek 5.1: Proces zpracování XML pomocí SAX

vému pojetí umožňuje sofistikovanější přístup a je reprezentován rozhraními XMLEventReader a XMLEventWriter.

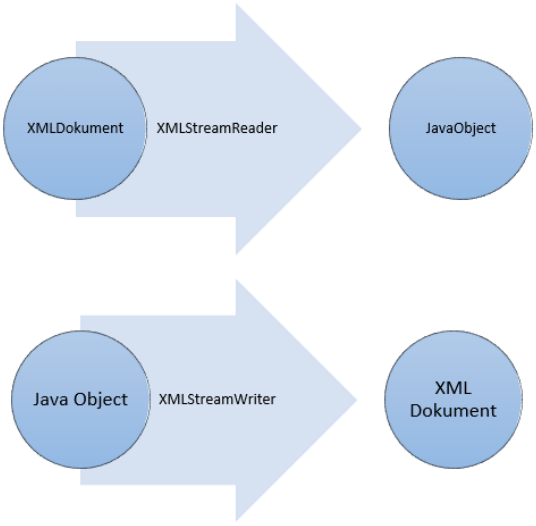
Výhodou oproti SAX je možnost dokument vytvářet a zapisovat do něho. SAX využívá při čtení souboru třech vzájemně provázaných metod, kdežto u StAX je zdrojový kód na jednom místě. Není zapotřebí pomocný buffer pro ukládání hodnot elementů, protože obsah elementů je vracen atomicky.

StAX je typu pull-parser, to znamená, že XML soubor se čte na žádost programátora. Výhodou oproti způsobu zpracování DOM je práce s proudem (stream), což umožňuje zpracovávat i velké dokumenty. Stejně jako u SAX umožňuje Java práci se StAX přes rozhraní JAXP. Samotný objekt StAX lze použít jako verifikátor pro dodržení správného formátu XML. Chybová hlášení jsou stejná u obou parserů, neboť jsou začleněna do rozhraní JAXP. Výjimka je také vyvolána, pokud je načítána hodnota atributu před čtením hodnoty elementu.

Při čtení dokumentu lze data načítat postupně až tehdy, když je potřebujeme. To je další výhoda oproti SAX, kde se musí přečíst celý XML dokument najednou. StAX si pamatuje pozici, ze které se naposledy četlo, a dále umožňuje čtení kdykoliv ukončit.

StAX se hodí pro postupné generování XML souborů. Oproti DOM zde nejsme omezeni velikostí paměti. Pro vytvoření souboru je nutné vytvořit zapisovač implementující rozhraní XMLStreamWriter, viz obrázek 5.2. Na začátku lze nastavit výstupní kódování a využívat metody pro automatický

převod, např. „&” na „&amp” a další. Pro odřádkování a odsazení elementů je nutné tyto úkony udělat ručně. Ruční odsazování závisí na platformě.



Obrázek 5.2: Proces zpracovnání XML pomocí StAX

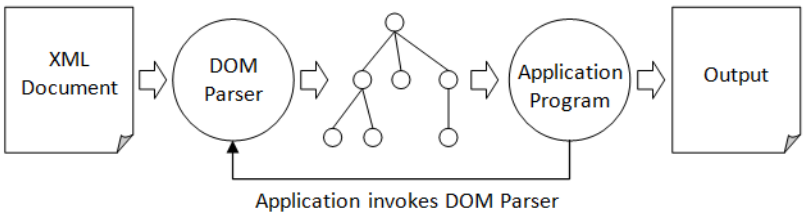
5.3.2 Práce se stromovou strukturou dokumentu

DOM

Document Object Model (DOM) načte celý XML dokument do paměti, kde vytvoří stromovo objektovou strukturu. Jednotlivým objektům se pak říká uzly. DOM umožňuje zápis a vytváření nových XML souborů díky všem potřebným datům uloženým najednou v paměti. Také rozhraní DOM je používáno ve spolupráci s JAXP. Rozhraní popisující jednotlivé uzly se nachází v org.w2c.dom.

Nejpoužívanější typy uzlů jsou Node pro základní prvek, Document pro kořenový uzel, Attr pro atributy, Element pro jednotlivé elementy a Text pro hodnoty elementů. Proces zpracování XML pomocí DOM je zobrazen na obrázku 5.3. DOM umožňuje také využití dvou dalších kolekcí pro uschování skupiny dalších uzlů. NodeList (podobný seznamu z java.util.List) uspořádá elementy do seznamu, do kterého se pak přistupuje pomocí indexu. Druhá je kolekce NamedNodeMap představující mapu pro uložení jmen atributů a jejich hodnot. Přístup do mapy probíhá typicky pomocí jmen atributů. Stejně jako u SAX lze DOM použít pro verifikaci XML dokumentu.

Data uložená v paměti lze zpracovat více možnostmi oproti SAX. Při častém pohybu po jednotlivých uzlech lze použít možnosti jazyka XPath, který tento pohyb usnadní. Pro snadnější zpracování umožňuje DOM odstranit elementy odřádkování nebo komentáře. Pro odstranění odřádkování je nutné mít připravený XSD nebo DTD soubor.

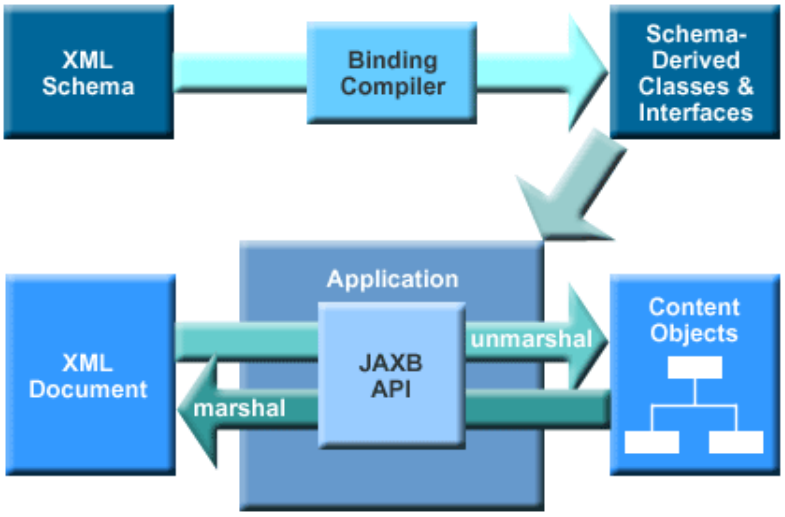


Obrázek 5.3: Proces zpracování XML pomocí DOM

JAXB

Java Architecture for XML Binding (JAXB) má zcela jiný přístup ke zpracování XML než SAX, StAX a DOM. Využívá již jednou vytvořeného XSD schématu pro mapování XML dokumentu na odpovídající Java třídy. Výhodou je, že XML dokument už byl jednou detailně popsán a třídy se dají případně vygenerovat automaticky.

JAXB je vhodný, pokud je dopředu známo XSD schéma, zpracovávaný dokument obsahuje opakující se informace, nebo jsou tyto informace velmi důsledně strukturované, dále pokud XML potřebujeme ve velkém rozsahu měnit nebo načítat. Omezení spočívá v jeho stromové struktuře, díky čemuž není možno načíst příliš velké soubory. Při práci jsme úplně odstíněni od XML formátu, neboť pracujeme pouze s objekty Java tříd. Objekty v paměti lze měnit a výsledný model validovat pomocí XSD. Tento model lze snadno zapsat do XML souboru. Na začátku práce je nutné využít XML to Java Compiler (XJC) pro vytvoření jednotlivých .java souborů odpovídajících většinou jednomu elementu v XML. Schéma zpracování viz obrázek 5.4. Tyto soubory je nutné přeložit pomocí kompilátoru a případně z nich vytvořit jeden .jar soubor. XJC lze spustit např. pomocí nástroje Ant. Jednotlivé elementy mají připravené getry a setry v .java souborech buď pro primitivní datové typy nebo typy z Java Core API.



Obrázek 5.4: Proces zpracování XML pomocí JAXB

5.3.3 Výběr parseru

Při práci s nástrojem pro grafický popis procesů je potřeba dokumenty XML jak načítat, tak i vytvářet a měnit. Proto nejméně vhodná volba pro zpracování je SAX, který dokumenty dokáže pouze načíst. Ostatní tři způsoby zpracování jsou velice podobné, přičemž StAX by byl nejpoužitelnější při zpracování velkých XML dokumentů. Pro tuto práci bylo zvoleno zpracování pomocí JAXB pro jeho objektový přístup ke zpracování XML dokumentů a úplné odstínění od něho. Dalším důvodem pro zvolení JAXB je možnost využití dopředu vypracovaného XSD schématu a také předpoklad, že některé informace budou velmi strukturované. JAXB se hodí i pro následující práci v nástroji pro grafický popis procesů, kde se bude XML ve velkém rozsahu měnit nebo načítat. Značná nevýhoda se skrývá v načítání do stromové struktury v paměti a s tím spojené omezení velikosti XML dokumentu. Toto omezení by nemělo být tak vážné s přihlédnutím ke skutečnosti, že příliš velký dokument obsahuje pravděpodobně špatně definovaný proces vývoje software, pattern nebo anti-pattern.

5.4 JavaFX

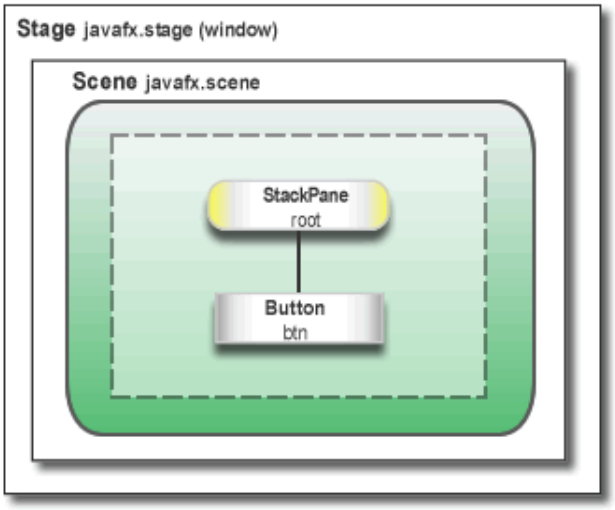
Knihovna JavaFX slouží pro tvorbu Graphical User Interface (GUI), neboli grafických uživatelských rozhraní, pomocí programovacího jazyka Java, viz [9]. Framework JavaFX je součástí jazyka od jeho verze 8, jako náhrada za starší možnosti tvorby GUI. Do předchozích verzí Javy ho lze doinstalovat.

Nejstarší nástroj pro vytváření GUI v Javě je Abstract Window Toolkit (AWT). Jedná se o jednodušší a nejrychlejší nástroj pro GUI díky návaznosti na knihovny operačního systému (OS). AWT bylo následně nahrazováno knihovnou Swing. Swing se stal velmi komplexním díky využívání 30 komponent, u kterých bylo pomalejší vykreslování oproti AWT, kvůli čistému použití Javy. Použitím čisté formy Javy se stal Swing nezávislý na OS, viz [12]. Je zde využíváno velké množství vlastností převzatých z AWT. Swing byl nahrazen právě frameworkem JavaFX. Umožňuje jednodušší práci s událostmi, využití vlastností CSS a HTML. Jména využívaných tříd jsou shodná s pojmenování v knihovně AWT.

5.4.1 Podporované technologie a základní prvky JavaFX

První podporovanou technologií je jazyk FX Markup Language (FXML) pro popis vzhledu aplikace, viz [4]. Tento jazyk je založen na XML a přináší možnost vyžití What You See Is What You Get (WYSIWYG) návrháře designu, například JavaFX Scene builder v některé z jeho verzí. V tomto návrháři lze vytvořit design a uložit jej pomocí FXML. Obsluha událostí musí být napsána v Java kódu. Barevný vzhled aplikace lze měnit pomocí souborů CSS. JavaFX umožnuje zobrazovat a pracovat s prvky HTML5. Také je kompatibilní s komponentami Swingu. Nově je zde i podpora dotykových zařízení a to i více dotykových gest.

Základním prvkem aplikace v JavaFX je třída Stage. Jedná se o kontejner nacházející se na nejvyšší úrovni obvykle reprezentující jedno okno aplikace. Kontejner může obsahovat libovolné množství dalších komponent. Jako jsou například tlačítka, tabulky, stromy, prvky pro vstup od uživatele a případně další kontejnery. Všechny tyto komponenty mají jednoho předka Node, uzel. Node je každý prvek viditelný na scéně. Má své ID může obsahovat třídy pro CSS a lze mu měnit vlastnosti, průhlednost, pozici, rozmazaní spoustu dalších, ukázka rozložení Stage na obrázku 5.5.



Obrázek 5.5: Rozdělení kontejneru Stage

6 Realizace grafického nástroje

V této kapitole bude popsána implementace aplikace, která je výsledkem praktické části této práce (dále jako nástroj). Nástroj byl naprogramován v jazyce Java ve verzi 1.8, od které lze používat framework JavaFX pro tvorbu GUI aplikací. Program byl vyvíjen ve vývojovém prostředí Eclipse Neon a ve spolupráci s verzovacím nástrojem GitHub.

6.1 Struktura aplikace

Zdrojové kódy aplikace jsou rozděleny do jednotlivých balíků. Balíky jsou pojmenované podle funkčnosti tříd, které obsahují.

Balík `abstractform` obsahuje abstraktní třídy, ze kterých následně dědí třídy představující formuláře pro zadání dat o jednotlivých segmentech a elementech vývoje software. Třídy dědící dané abstraktní třídy se nacházejí v balíku `forms`. Ostatní třídy obsahující grafické komponenty lze nalézt v balíku `graphics`, například třídu MenuPanel představující kontextové menu nebo třídu DragAndDropCanvas obsahující kreslící plátno pro jednotlivé drag and drop prvky aplikace. Balík `interface` obsahuje rozhraní, využívaná k sjednocení funkčnosti jednotlivých formulářů. Balík `run` obsahuje třídu Main obsahující metodu main() pro spouštění celého programu. Veškeré třídy tvořící obslužnou část programu se nacházejí v balíku `service`. Například se zde nachází třída Constans se všemi konstantami, které jsou využívány v aplikaci, dále pak třída Control obsluhující metody pro obsluhu jednotlivých prvků kontextového menu. A metody pro převod datumu z LocalDate formátu do XMLGregorianCalendar a naopak. Třídy potřebné pro komponentu TableView jsou umístěny v balíku `tables`. Posledním využívaným balíkem je balík `XML` obsahující třídy určené pro práci s XML soubory.

6.2 Použité knihovny

V této kapitole budou popsány knihovny třetí strany, respektive knihovna vytvořená pomocí XSD schématu pro práci s XML objekty v Java.

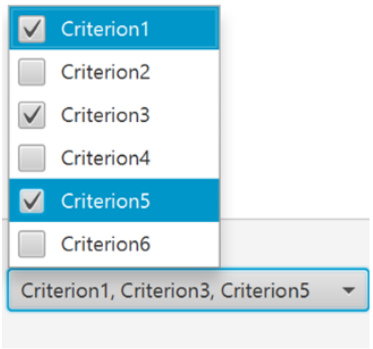
6.2.1 Knihovna SPADEPAC

Tato knihovna je využívána pro práci s třídami představujícími elementy XML souboru namapované pomocí XSD schématu. Knihovna byla vytvořena pomocí programu xjc.exe a skriptu utility Ant (build.xml).

6.2.2 Knihovna ControlsFX

„ControlsFX je open source projekt pro JavaFX, který si klade za cíl poskytnout skutečně kvalitní UI ovládací prvky a další nástroje, které doplňují základní JavaFX distribuci.“ [5].

Z této knihovny je využívána komponenta CheckComboBox umožňující výběr více prvků z jednoho combo boxu, neboť základní varianta frameworku JavaFX tuto komponentu neobsahuje, viz obrázek 6.1. Verze knihovny využívané v aplikaci je 8.40.11.



Obrázek 6.1: Ukázka komponenty CheckComboBox

6.3 Implementace drag and drop

Implementaci drag and drop prvků lze rozdělit do dvou kategorií. Do první kategorie patří přidání segmentu nebo elementu procesu (vykreslení pomocí obrazce) na kreslící plátno. Do druhé kategorie lze zařadit samotné přesouvání komponent, představující jednotlivé prvky, po kreslícím plátně, viz [6].

6.3.1 Přidání prvků na plátno

Prvky Activity, Iteration, Phase představují segmenty vývoje a Work Unit, Change a Artifact představující elementy vývoje software, zobrazené na obrázku 2.1, představujícím datový model SPADe. Jsou reprezentovány tří-

dou `DragText`. Tato třída dědí vlastnosti od třídy Button a využívá metody setDragDetected(), startDragAndDrop(TransferMode.ANY), setDragDone() pro přetažení prvku na plátno. V těchto metodách je použit EventHandler<MouseEvent>. Tento způsob implementace byl zvolen z důvodu umožnění uživateli přidat prvek na plátno jak jednoduchým stisknutím tlačítka, tak přetažením komponenty na plátno, viz ukázka kódu 6.1.

Druhá část tohoto typu drag and drop se nachází ve třídě `DragAndDropCanvas` oddděné od třídy AnorchPane umožňující vytvoření layoutu pro přidávané prvky. Tomuto panelu se zde nastaví metody pro zpracování přetažené komponenty na plátno. Například setOnDragOver() nebo setOnDragDropped() využívající EventHandler<DragEvent>.

```
private void setDragDetected() {

    this.setOnDragDetected(new EventHandler<MouseEvent>() {
      public void handle(MouseEvent event) {

        dragDetected();
        event.consume();
      }
    });
  }
```

Ukázka kódu 6.1: Ukázka metody pro detekci drag and drop

6.3.2 Přesouvání komponent po plátně

Druhým způsobem využití metody drag and drop je přesun komponent po plátně. Tato funkce je celá implementována ve třídě `CanvasItem`. Na rozdíl od předchozího způsobu zde není využíváno rozpoznávání pohybu daného prvku jako takového, ale pouze pohybu myši s daným prvkem. Komponenta má pro tento způsob pohybu implementované dvě metody. První metoda setOnMousePressed umožnuje rozpoznání kliku na daný prvek. Uvnitř EventHandleru se nastaví proměnné třídy, které představující startovní polohu prvku, ze které se následně vypočítává posun prvku po plátnu. Aktuální souřadnice jsou získávány pomocí metody setOnMouseDragged a EventHandler<MouseEvent>. Pro kontrolu pozice komponenty byla využita metoda setOnMouseRelease(), viz ukázka kódu 6.2.

```
public void setDragFromDragPoint(MouseEvent t) {

    double offsetX = t.getSceneX() - orgSceneX;
    double offsetY = t.getSceneY() - orgSceneY;
```

```
    double newTranslateX = orgTranslateX + offsetX;
    double newTranslateY = orgTranslateY + offsetY;
((AnchorPane) (t.getSource())).setTranslateX(newTranslateX);
    ((AnchorPane) (t.getSource())).setTranslateY(newTranslateY);

  }
```

Ukázka kódu 6.2: Ukázka metody pro výpočet posunu

6.4 Implementace zadávacích formulářů

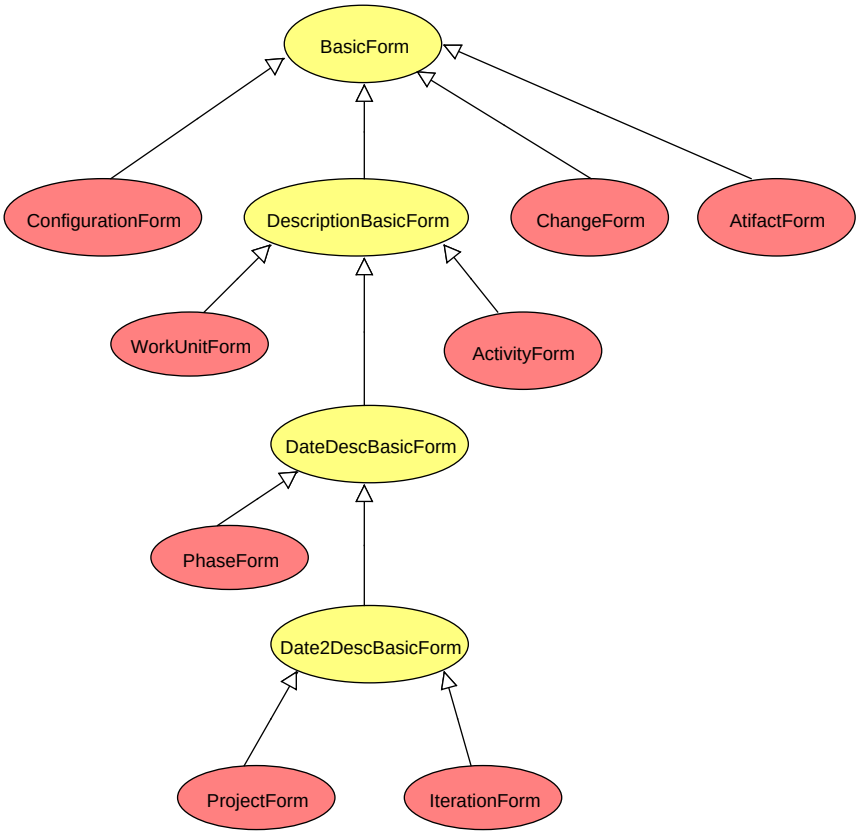
Zadávací formuláře jsou jednou z nejdůležitějších komponent celé aplikace. Pomocí těchto formulářů uživatel zadává veškerá data, která se ukládají do datových struktur a následně se ukládají do XML souboru. Pro implementaci formulářů bylo využito několika vlastností jazyka Java. Pro zjednodušení kódu bylo využito dědičnosti tříd, pro sjednocení funkčnosti a vlastností formulářů pak rozhraní.

6.4.1 Struktura dědění

Velké množství segmentů procesu vývoje má společné datové prvky, a tedy i grafické komponenty. Například jméno, popis nebo datum vytvoření. Proto byly vytvořeny tři struktury dědění tříd se společnými grafickými prvky. První struktura byla vytvořena pro formuláře vázané na komponenty využívané na plátně. Tyto formuláře většinou umožňují vkládaní složitějších dat a jejich následné zpracování. Vytváření jednodušších segmentů a jejich přehled patří do druhé skupiny dědění. Hlavní komponentou využívanou v těchto formulářích je TableView. Ve třetí skupině se nachází formuláře, které jsou tvořeny ze dvou menších formulářů úzce spolu souvisejících.

Hlavní formuláře

Základní třídou v této skupině je třída `BasicForm` oddděná od třídy Stage. Obsahuje hlavní panel, do kterého se vkládají další komponenty, tlačítka, GridPane s prvky pro ukládání dat nebo v případě potřeby umožňuje přidat vlastní plátno. V této třídě je vytvořeno vstupní pole pro zadaní jména. Třída `DescriptionBasicForm` využívající třídu `BasicForm` přidává vstupní pole pro zadaní popisu. Dalšími třídami následujícími v hierarchii dědění jsou třídy `DateDescBasicForm`, která přidává vstupní pole pro zadání data, a od této třídy dědí třída `Date2DescBasicForm`, umožňující zadání druhého data, graf dědičnosti na 6.2.



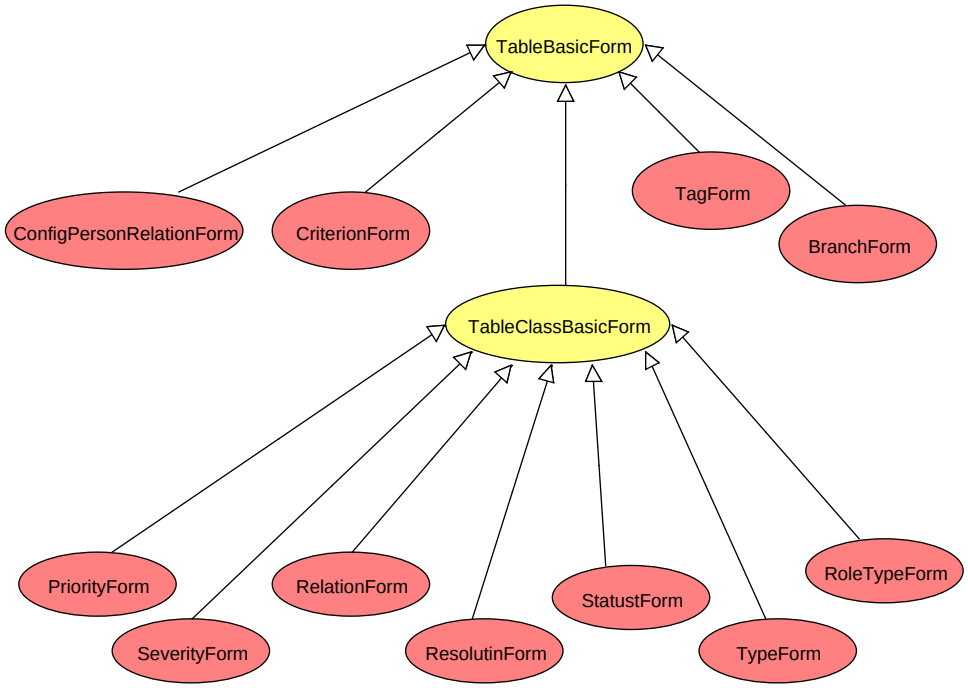
Obrázek 6.2: Přehled dědičnosti hlavních formulářů

Formuláře s TableView

První třídou v hierarchii této skupiny je třída `TableBasicForm`, která obsahuje hlavní panel, do kterého je vložen panel s prvky umožňujícími zadání dat do tabulky, potažmo do datových struktur pro možnost výběru z ostatních formulářů. Vlastní tabulku pro reprezentaci dat implementuje každá třída dědící tuto třídu sama. Výjimkou jsou formuláře umožňující vytvoření metadat, například pro Work Unit. Zde jsou třídy odděděny od třídy `TableClassBasicForm`, která obsahuje TableView společné pro všechny třídy a dědí z třídy `TableBasicForm`, diagram dědění tabulkových formulářů na obrázku 6.3.

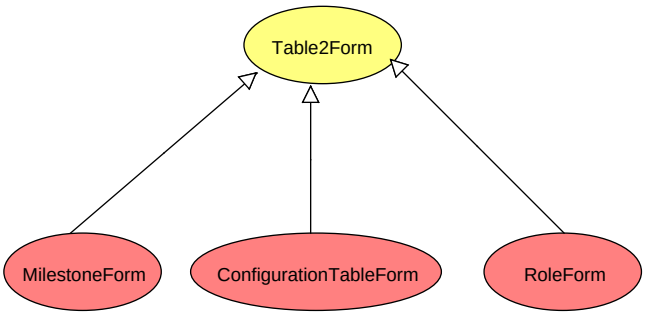
Dvojité formuláře

Třídy oddděděné z třídy `Table2BasicForm` mají schopnost zobrazit dva formuláře v jednom okně. Může se jednat jak o formulář oddděděný z linie hlavních formulářů, tak i o formuláře děděné od tabulkové linie, ukázka linie na



Obrázek 6.3: Přehled dědičnosti tabulkových formulářů

obrázku 6.4.



Obrázek 6.4: Přehled dědičnosti dvojitých formulářů

6.5 Implementace XML

Aplikace umožňuje vytvářet a ukládat nové procesy vývoje software, ale lze v ní i upravovat již vytvořené procesy uložené do XML formátu. Proto bylo potřeba implementovat dva typy práce s XML. Jeden pro uložení vytvoře-

ného procesu v paměti do XML formátu a druhý pro opačný postup načtení dat z XML souboru do paměti.

6.5.1 Ukládání dat do XML

Ukládání objektového modelu z paměti do XML je implementováno ve třídě `ProcessGenerator` instancí třídy Marshaller, které lze pomocí metody marshall() předat objekt představující kořenový element obsahující ostatní elementy vytvořené pomocí formulářů a stream obsahující soubor pro uložení datového modelu, viz ukázka kódu 6.3.

Plnění kořenového a dalších elementů, probíhá ve třídě `FillForms`. Z této třídy lze zavolat metody vykonávající naplnění elementů ze zadávacích formulářů. Pro identifikaci jednotlivých elementů uchovávaných v seznamech a k nim patřícím formulářům byla vytvořena třída `IdCreator`, ve které se vytváří ID. Tyto ID se ukládají do třídy `CanvasItem` představující jednotlivé prvky vývoje procesu na plátně, viz [8].

```
jc = JAXBContext.newInstance(Constans.LIBRARY);
   validator = new ProcessValidator(jc);
   of = new ObjectFactory();

   marshaller = jc.createMarshaller();
   marshaller.setProperty(Marshaller.JAXB_ENCODING, "UTF8");
   marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
       Boolean.TRUE);     marshaller.setProperty(Marshaller.
       JAXB_NO_NAMESPACE_SCHEMA_LOCATION, Constans.XSDNAME);

rootElement = of.createPoject(project);
marshaller.setSchema(validator.getSchemaXSD());
marshaller.marshal(rootElement, new FileOutputStream(file));
```

Ukázka kódu 6.3: Ukázka metody pro uložení XML

6.5.2 Načítání dat z XML

Načítání objektového modelu z XML souboru do paměti je implementováno také ve třídě `ProcessGenerator`. Instancí třídy UnMarshall, metodou unmarshall vracející referenci na objekt představující kořenový element. Tento element je následně zpracován ve třídě `FillFormsXML`. Zde lze pomocí metod obsluhujících jednotlivé objekty elementů, naplnit všechny potřebné seznamy a formuláře. Opět s využitím třídy `IdCreator`.

6.5.3 Validace XML

Validace formátu XML probíhá jak při načítání objektového modelu z XML do paměti, tak hlavně pokud je nějaký model ukládán. Pro validaci byla vytvořena třída `ProcessValidator` obsahující instanci třídy `Validator` z balíku javax.xml.validation. Validator dále využívá třídu `ValidatorErrors`, implementující rozhraní ErrorHandler a umožňuje příjemnější výpis nalezené chyby v modelu. Výjimka, kterou vrátí Validator, je následně zpracována a převedena do chybového hlášení zobrazeného pomocí třídy `Alert`.

6.6 Zajímavé třídy a části kódu

6.6.1 Třída Control

Za nejvýznamnější třídu z pohledu vykonávání obslužných činností lze požadovat třídu `Control`. Tato třída například umožňuje konverzi data vybraného z DatePickeru, v některém z formulářů, vracejícího datum ve formátu LocalDate do formátu XMLGregorianCalendar metodou convertDate(LocalDate Ldate). Formát XMLGregorianCalendar je využit v XSD schématu pro definování konkrétního data. Při načítání elementů z XML umožňuje opačnou konverzi pomocí metody convertDateFromXML(XMLGregorianCalendar xmlDate).

Třída dále obsahuje metody využívané v kontextovém menu aplikace. Metodu openFile() volanou po stisku volby open, znamenající načtení již vytvořeného procesu v XML do paměti. Nejdříve je zavoláno standardní okno pro výběr daného souboru komponentou FileChooser() a následně jsou volány metody třídy `FillFormsXML` pro zpracování všech elementů nacházejících se v XML souboru. Metodu saveFile() lze použít pro uložení objektového modelu v paměti. Tato metoda je volána obsluhou události po zvolení Save v hlavním menu. Zde stejně jako v předchozím případě je zavoláno okno operačního systému pro výběr, respektive zadání jména souboru pomocí FileChooser a jeho uložení. Výběr formátu pro uložení byl omezen pouze na formát XML, dále je předán kořenový element třídě `ProcessGenerator` pro uložení dat do XML souboru, viz ukázka kódu 6.4.

```
public void saveFile() {
    if (file == null || firstSave) {
      saveAsFile();
      firstSave = false;
    } else {
      procesGener.saveProcess(file, project);
```

```
    }
  }
```

Ukázka kódu 6.4: Ukázka metody pro obsluhu uložení

Metody createForm(CanvasItem item, BasicForm form) a createFormFromXML(CanvasItem item, BasicForm form) slouží k rozhodnutí, o jaký typ segmentu procesu se jedná a následně zavolají metody pro vytvoření daného prvku představující tento segment v aplikaci, případně ho zobrazí na plátně v nadřazeném formuláři. Rozhodování o typu segmentu je založeno na porovnání hodnot výčtového typu `SegmentType`.

6.6.2 Implementace komponenty na plátně

V této části bude popsána grafická podoba posuvné komponenty na kreslícím plátně, nikoliv její drag and drop schopnost popsána v kapitole 6.3.2. Tento grafický prvek je tvořen třídou `InfoBoxSegment`. Výslednou podobou prvku tvoří dva obdélníky typu Rectangle poskládané nad sebe. Popis typu segmentu, který daný prvek představuje, je vytvořen pomocí třídy Text, stejně tak jméno prvku. Využití třídy Text pro výpis textu, umožnilo dynamickou změnu velikosti spodního obdélníku, využitím metody getLayoutBounds().getWidth() sloužící jak pro výpočet šířky, tak i výšky prvku. Metoda setWrappingWidth() umožnuje nastavení automatického zalamování textu po určité velikosti obsahu, viz ukázka kódu 6.5.

```
name.setWrappingWidth(Constans.maxCanvasItemWidth);

    if (width > length && width < Constans.maxCanvasItemWidth) {
      repaintBox(width, height);

    } else if (width > Constans.maxCanvasItemWidth) {
      int count = countHeightBotomRectangle((int) width);
      repaintBox(Constans.maxCanvasItemWidth, count * height);
    }
```

Ukázka kódu 6.5: Ukázka metody pro výpočet velikosti prvku plátna

Jméno segmentu a další informace o něm lze zadat do formuláře vyvolaného dvojklikem na prvek. Obsluhu události myši nad prvkem obstarává třída `CanvasItem` obsahující instanci třídy InfoBoxSegment, pomocí MouseEvent a metody getClickCount(). Třída CanvasItem také umožňuje pohyb po plátně již popsaný v kapitole 6.3.2.

6.6.3 Zpracování výběru z ComboBoxů

Klasický ComboBox

Pro výběr již definovaných segmentů procesu v nadřazených formulářích byla zvolena komponenta ComboBox. Tato komponenta je schopna využívat datovou strukturu ObservableList pro automatické přidávání položek do výběrového seznamu. Pro jednotlivé druhy segmentů byly vytvořeny ObservableListy ve třídě SegmentLists. Tyto seznamy uchovávají datový typ String představující ve většině případů vytvořené jméno segmentu uživatelem.

Obsluhu výběru dané položky umožňuje posluchač ChangeListener<Number>() navázaný na konkrétní ComboBox. Uvnitř této vnitřní třídy je volána metoda change(), ve které se získává pořadí vybrané položky a následně předané globální proměnné pro konečné zpracování formuláře, viz ukázka kódu 6.7.

```
ChangeListener<Number> milestoneListener = new ChangeListener<
    Number>() {
     @Override
public void changed(ObservableValue<? extends Number> observable
    , Number oldValue, Number newValue) {
       milestoneIndex = newValue.intValue();
     }
  };
```

Ukázka kódu 6.6: Ukázka metody pro nastavení ChangeListeneru

CheckComboBox

V některých případech bylo potřeba naplnit vazbu 1:N případně N:M a proto byla zvolena komponenta CheckComboBox umožňující výběr většího počtu nabízených voleb. Tato komponenta se nevyskytuje v základním frameworku JavaFX, a proto byla použita knihovna třetí strany controlsfx-8.40.1, viz [5]. Princip zadání ObserverListu je u CheckComboBoxu stejný jako v předchozím případě u klasického ComboBoxu. Liší se v používání typu posluchače výběru prvků. Zde je využívána třída ListChangeListener a metoda onChange. Uvnitř metody je získáván seznam indexů vybraných prvků, ale i seznam se jmény daných prvků pro přidání do TableView, viz ukázka kódu **??**.

```
criteriaCB.getCheckModel().getCheckedItems().addListener(new
    ListChangeListener<String>() {
```

```
public void onChanged(ListChangeListener.Change<? extends String
    > c) {
criterionIndex = criteriaCB.getCheckModel().getCheckedIndices();
criterionArray = criteriaCB.getCheckModel().getCheckedItems();

  }
});
```

Ukázka kódu 6.7: Ukázka metody pro nastavení ListChangeListeneru

7 Testování

Aplikace byla testována v průběhu vývoje za pomoci jednotkových testů. Pro vytvoření jednotkových testů bylo využito vývojové prostředí Eclipse s knihovnami JUnit 4 a TestFX ve verzi 4.0.1. Výsledná aplikace prošla všemi funkčními testy. Jednotkové testy pokrývají 60% funkčnosti aplikace. Větší počet funkčních testů nebyl možný z důvodu neúplné znalosti možností testovací knihovny TestFX.

Pro další testování byla zvolena pěti členná skupina testerů (vybraných uživatelů) s různými schopnostmi využívání PC, například projektant SW, studentka gymnázia, hotelový manager, kteří dostali již hotovou aplikaci, na které měli za úkol vyzkoušet její vlastnosti nebo alespoň její nejdůležitější funkce jako jsou například:

- vytvoření procesu vývoje,
- kopírování, vkládání a odstraňování jednotlivých segmentů,
- uložení a načtení vytvořeného procesu,
- vyzkoušení drag and drop funkčnosti.

7.1 Výsledky testování

Během testování byla testery zjištěna možnost vytvoření relace u některého z elementů umožňujícího propojení, například Change, samo se sebou. Tato chyba byla opravena rozšířením logiky řízení relací o zákaz spojení prvku se stejnou identifikací jako počáteční prvek.

Jako druhý problém, byla zjištěna možnost přetažení drag and drop prvku kreslícího plátna mimo jeho plochu. Tento problém byl vyřešen porovnáním polohy prvku s hranicemi plátna po dokončení tahu s daným prvkem. Pokud se prvek nachází mimo hranice plátna, je automaticky stažen na jeho nejbližší okraj. Testeři měli také poznámky ke grafické části aplikace. Některé byly zapracované do výsledné podoby aplikace, například zvětšení potvrzovacích tlačítek uvnitř formulářů, vytvoření tlačítka pro smazání prvku z

tabulky nebo přepracování některých hlášení. Velká množina těchto připomínek plynula z neúplné znalosti problematiky.

Hlavní problémy odhalené uživatelským testováním byly odstraněny. Uživatelé považují aplikaci za přehlednou a intuitivní.

8 Možná budoucí rozšíření

Aplikaci lze samozřejmě dále rozšiřovat a upravovat. K již implementovaným částem aplikace by mohly přibýt i další funkcionality. Mezi možná rozšíření, která by mohla být implementována, patří:

- přehledový strom v levé části úvodní obrazovky,
- uživatelsky přívětivější výpis výsledku validace,
- možnost měnit barevné schéma aplikace,
- možnost přesouvání, kopírování a vkládání více prvků,
- vylepšení vytváření relací.

9 Závěr

Cílem této bakalářské práce bylo navrhnout, vytvořit a otestovat rozšíření nástroje SPADe o grafický editor modelů pro namodelování procesů, patternů a anti-patternů. Práce obsahuje porovnání dvou nástrojů pro tvorbu popisu procesů vývoje software IBM RMC a EPF Composer. Tyto nástroje posloužily jako inspirace při vývoji vlastní aplikace, jelikož jejich velká funkcionalita a nemožnost vytváření anti-patternů (vyznačení neexistence konkrétního segmentu) neumožňuje jejich přímé využití nástrojem SPADe. Dále bylo potřeba prozkoumat procesní metamodel nástroje SPADe a následně podle tohoto modelu vytvořit XSD schéma, které umožňuje zachycení vytvořených modelů do XML formátu. Z tohoto důvodu byly zkoumány možnosti programovacího jazyka Java pro práci s XML soubory. Jednotlivé parsery pro zpracování elementů a jejich rozdělení podle způsobu zpracování. Byl vybrán parser JAXB. Dále byl zkoumán framework JavaFX sloužící k vytvoření GUI v jazyce Java a předešlé možnosti vytváření GUI. Tímto byly splněny 1. a 2. bod zadání v kapitolách 3 a 4.

V rámci práce jsem implementoval funkcionalitu popsanou v kapitole 5. Vytvořené XSD schéma obsahuje popisy datových struktur jednotlivých prvků procesu vývoje software, pro ukládání konkrétních hodnot, ale také slouží jako validační soubor pro ověření XML modelu. V grafickém prostředí lze vytvářet proces vývoje software jak pomocí drag and drop komponent, libovolně přemístitelných po plátně, tak vytváření jednotlivých částí pomocí přehledových tabulek. Mezi některými prvky obsaženými na kreslícím plátně lze vytvářet relace jak pojmenované, tak bezejmenné. Tyto prvky lze pomocí kontextového menu kopírovat, vkládat do různých prvků případně je odstranit z plátna. Aplikace umožňuje vytvořený proces, jak ukládat do XML souboru, tak z něho následně číst. Aplikace je funkční a odpovídá požadavkům, čímž je splněn 3. bod zadání.

Práce se potýká s několika technickými nedostatky jako například nemožnost přemisťovat větší množství prvků plátna navzájem stejně jako jejich vykopírování či vložení. Dále pak jednodušší grafické prvky jak uvnitř formulářů nebo pro vytváření relací. Některé tyto nedostatky by mohly být napraveny implementací možných rozšíření aplikace popsaných v kapitole 8.

Aplikace byla testována pomocí testovacích knihoven JUnit 4 TestFX určené pro testování GUI vytvořených ve frameworku JavaFX. Výsledná aplikace byla předána 5 nezávislým testerům s různými schopnostmi ovládání PC. Popisy některých nalezených problémů se nacházejí v kapitole 7. Tímto byl splněn 4. bod zadání.

Práce umožní následně další rozšíření nástroje SPADe o generování dotazů a hledání modelovaných vzorů v databázových systémech. Bude zapracována do výsledného grafického editoru nástroje SPADe, jako jeho další funkcionalita.

Hlavním cílem práce bylo vytvoření grafického nástroje pro popis procesu vývoje software. Tento cíl práce byl splněn. Aplikace umožňuje základním způsobem pracovat s drag and drop prvky, dále pak editaci jednotlivých segmentů a elementů, které tyto prvky představují. Ukládání a načítání již vytvořených modelů a případně jejich validaci vůči popisu v XSD schématu.

Literatura

[1] *IBM - Rational Method Composer* [online]. 2013. Dostupné z: http://www-03.ibm.com/software/products/cz/cs/rmc/.

[2] *JDOM* [online]. Dostupné z: http://www.jdom.org/.

[3] Chromík, E. *Oborový projekt - Popis procesu ASWI v Rational Method Composer,*. 2011.

[4] Fedortsova, I. *Why Use FXML* [online]. 2014. Dostupné z: http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDCHIBE.

[5] Giles, J. *ControlsFX* [online]. 2017. Dostupné z: http://fxexperience.com/controlsfx/.

[6] Graff, J. *Drag-and-Drop* [online]. 2015. Dostupné z: https://monograff76.wordpress.com/2015/02/24/developing-a-drag-and-drop-ui-in-javafx-part-4-drag-and-drop/.

[7] Haumer, D. P. *Eclipse Process Framework Composer* [online]. 2007. Dostupné z: https://eclipse.org/epf/general/EPFComposerOverviewPart1.pdf.

[8] Herout, P. *Java a XML*. KOPP, 2012. ISBN 978-80-7232-307-4.

[9] Iverson, D. *Pro JavaFX 2. s.l.* Apress, 2012. ISBN 978-1-4302-6872-7.

[10] Jakob, M. *JavaFX Dialogs* [online]. 2014. Dostupné z: http://code.makery.ch/blog/javafx-dialogs-official/.

[11] Kosek, J. *XML schémata* [online]. 2014. Dostupné z: http://www.kosek.cz/xml/schema/wxs.html#wxs-atributy.

[12] Lipka, R. *Úvod do JavaFX* [online]. 2016.

[13] Pícha, P. Popis softwarových procesů použitelný pro nástroje řízení projektů. Master's thesis, ZCU, 2013.

[14] Pícha, P. Datový model nástroje SPADe a jeho mapování na projektová data z ALM nástrojů. 2016.

[15] Petr Pícha, P. B. ALM Tool Data Usage in Software Process Metamodeling. 2016.

[16] Petr Pícha, P. B. Towards Architect's Activity Detection Through a Common Model for Project Pattern Analysis. 2016.

[17] W3Schools. *XML Tutorial* [online]. Dostupné z: `https://www.w3schools.com/`.

Seznam použitých zkratek a výrazů

- SPADe – Software Process Anti-pattern Detector
- XML – eXtensible Markup Language
- RMC – Rational Method Composer
- EPF – Eclipse Process Framework
- API - Application Programming Interface
- JAXP - Java API for XML Processing
- SAX - Simple API for XML
- StAX - Streaming APIs for XML
- DOM - Document Object Model
- JAXB - Java Architecture for XML Binding
- GUI - Graphic User Interface
- SGML - Standard Generalized Markup Language
- HTML - Hypertext Markup Language
- DTD - Document Type Definition
- XJC - XML to Java Compiler
- AWT - Abstract Window Toolkit
- OS - Operačních systému

- CSS - Cascading Style Sheets
- FXML – Fx Markup Language
- Zcu – Západočeská Univerzita
- Fav – Fakulta aplikovaných věd

Seznam obrázků

Seznam příloh

A Obsah CD

Zde je rozepsán obsah CD nosiče přiloženého k této bakalářské práci.

- složka **src** obsahuje zdrojové kódy aplikace rozdělené do balíků.

- složka **text** obsahuje tento dokument ve formátu pdf spolu s jeho zdrojovými LaTeXsoubory, ve složce **Latex**.

- složka **dokumentace** obsahuje JavaDoc ke zdrojovým kódům

- složka **aplikace** obsahuje spustitelný .jar soubor s aplikací, XSD schéma popisu datového modelu a příkladový proces vývoje software

- soubor **README.txt** obsahuje tento rozpis obsahu CD nosiče

B Uživatelský manuál

Uživatelský manuál obsahuje informace o instalaci aplikace, požadavcích pro její spuštění, její funkcionalitu, popis ovládání a chybových hlášení.

B.1 Instalace aplikace

Aplikaci nelze spustit bez nainstalovaného běhového prostředí Java JRE nejméně verze 8. Se staršími verzemi nebude aplikace vůbec fungovat. Nejnovější verzi prostředí lze stáhnout ze stránek společnosti Oracle.

Aplikace se nijak neinstaluje, jednoduše se spustí po kliknutí na JAR soubor přiložený k tomuto dokumentu, nebo je k získání v několika verzích na stránkách verzovacího nástroje GitHub, konkrétně na adrese[1].

B.2 Rozložení aplikace

B.2.1 Hlavní obrazovka

Hlavní obrazovka je také úvodní obrazovkou aplikace, viz obrázek B.1. Ta umožňuje tři typy funkčnosti, v horní části se nachází kontextové menu popsané v kapitole B.2.3. Prostřední část, označená červenou, představuje pás tlačítek. Tento pás obsahuje dvě řady tlačítek, horní řada slouží k vyvolání formulářů patřících k danému elementu vývoje procesu, popsaných v kapitole B.3.1 a B.3.2. Spodní řada tlačítek slouží k vytvoření pohyblivého prvku na plátně. To lze provést klasickým stiskem tlačítka nebo přetažením tlačítka na plátno. Plátno je umístěno do spodní části obrazovky, označené žlutou barvou. Plátno slouží k rozmístění vytvořených segmentů patřících do základní skupiny projektu. Objekty na plátně lze libovolně přemisťovat a skládat.

B.2.2 Prvek plátna

Prvek obsahuje v horní části identifikaci s názvem segmentu či elementu, který představuje, a ve spodní části se nachází jméno vytvořené uživatelem. Velikost spodního obdélníku je automaticky měněna na základě délky

[1] github.com/VenaJanoch/BakalarkaSPADEUI/releases

textu zadaného uživatelem. Vyvolání formuláře proběhne po dvojkliku levým tlačítkem myši. Pravím tlačítkem je vyvoláno kontextové menu umožňující kopírování daného objektu, jeho odstranění a vyjmutí z plátna. Identifikaci jednotlivých objektů lze zjistit i po najetí na objekt. Ukázka segmentu Phase na obrázku B.2.

B.2.3 Menu

Menu je složené z kořenové položky, obsahující další podpoložky. Hlavní položka má označení File, zde se nachází možnost New pro vytvoření nového projektu, Open... pro načtení již vytvořeného procesu uloženého v XML, Save uložení procesu, Save as... uložení procesu s výběrem názvu souboru. Validate umožní ruční kontrolu procesu. A Close umožňující ukončení celé aplikace, viz obrázek B.3.

B.3 Druhy formulářů

Formuláře umožnují jednoduché zadávání konkrétních informací o jednotlivých segmentech procesu vývoje software do vstupních polí, nebo výběr již předdefinovaného elementu z nabídky. K využití jsou tři typy formulářů popsaných dále v této kapitole.

B.3.1 Formuláře s tabulkou

Tyto formuláře poskytují dva druhy funkčnosti. Prvním druhem tabulkového formuláře je formulář pro vytvoření prvku mapovaného z Class a Super Class. Druhý je určen pro jednoduché objekty, například přidání Tag do Configuration.

První typ formuláře je využíván pro mapování Class a Super Class výčtových typů Work Unit a Role na jednotný prvek, který lze dále vybrat v složitějších typech formulářů. Formulář obsahuje tabulku pro snadný přehled již namapovaných prvků. Pro identifikaci formuláře je umístěn název s typem elementu nad danou tabulku. Tabulka neumožňuje další editaci políček v ní zobrazených, ale pouze jejich odstranění klávesou Delete. Po stisku Delete se zobrazí okno s výzvou, zda uživatel chce opravdu data z tabulky smazat, konkrétní podoba výzvy je popsána v kapitole B.6.1. Přidání mapovaných objektů do tabulky, respektive přímo do projektu, je umožněno pomocí vstupních polí umístěných ve spodní části formuláře. Vstupní pole pro zadání jména objektu je následováno výběrovým seznamem typu Class

patřícího pod daný element. Po vybrání položky ze seznamu se automaticky zvolí hodnota v seznamu představujícím Super Class. Formuláře existují pro určení výčtových typů Priority, Severity, Relation, Resolution, Status a Type pro Work Unit. A dále u elementu Role pro typ Role-Type. Ukázka formuláře Priority na obrázku B.5.

B.3.2 Dvojí zobrazení formuláře

Elementy úzce spolu související jsou vloženy do jednoho společného formuláře. Ve formuláři proto lze nalézt dvě tabulky s přehledem již vytvořených elementů. Tabulky neumožňují další editaci políček v nich zobrazených, ale pouze jejich odstranění klávesou Delete. Po stisku Delete se zobrazí okno s výzvou, zda uživatel chce opravdu data z tabulky smazat (konkrétní podoba výzvy je popsána v kapitole B.6.1. Ke každé tabulce existují vstupní pole pro vytvoření dat do nových segmentů. Data se přidají tlačítkem Add. Vstupní pole a tabulka jednoho elementu jsou od druhého elementu odděleny oddělovačem, pro snazší orientaci jsou také nad tabulkami umístěny názvy elementů. Tlačítko OK vyskytující se v pravé spodní části obrazovky slouží k zavření formuláře. Příklad daného typu formuláře se nachází na obrázku B.6.

B.3.3 Složitější formuláře

Formuláře kořenových adresářů

Pro objekty patřící do skupiny kořenových segmentů projektu slouží formuláře se složitějším uspořádáním obsahu. Do této oblasti patří segmenty Phase, Activity, Iteration, element Configuration a do nich přidávané elementy Work Unit, Change, Artifact.

Formulář kořenového typu obsahuje dvě části, v levé části se vyskytují vstupní pole pro zadání informací o daném segmentu nebo elementu jako například name, description, popřípadě určité datum z kalendáře. Dále pak výběrové seznamy předem vytvořených elementů popsaných v kapitole B.3.1 a B.3.2.

V pravé části se nachází vnitřní plátno formuláře, na které lze přidávat další množinu komponent umístěnou nad samotným plátnem. Tyto komponenty mají stejné vlastnosti jako komponenty přidávané na plátno v hlavní obrazovce popsané v kapitole B.2.2. Příklad formuláře, viz obrázek B.4.

Pomocné formuláře kořenových adresářů

Komponenty představující Work Unit, Change a Artifact jsou vyplňovány přes klasický formulář s vstupními poli a potvrzovacím tlačítkem OK pro uložení dat a zavření formuláře, viz obrázek B.7.

Formulář segmentu Configuration

Tento formulář představuje nejsložitější složení obsahu. Jsou zde využívány formuláře zmíněné v předešlých kapitolách o formulářích. V levé části se vyskytuje formulář kořenového typu pro vytvoření Configuration. Zde je ze začátku klasické složení kořenového formuláře, ke kterému je přidáno tlačítko pro vytvoření Tag uvnitř Configuration. Uprostřed lze využít plátno pro přidání elementů Change a Artifact umístěné nad samotným plátnem. Tyto komponenty mají stejné vlastnosti jako komponenty přidávané na plátno v hlavní obrazovce popsané v kapitole B.2.2. Vedle již zmíněných segmentů se vyskytuje tlačítko pro vytvoření spojení mezi objekty více popsané v kapitole B.4. Tlačítkem Add je přidána Configuration do přehledové tabulky v pravé části. Již uložený formulář je nahrazen opět prázdným pro vytvoření nové Configuration. Tabulka v pravé části má standartní funkčnost tabulky popsané v kapitole B.3.1. Rozdíl je pouze v možnosti znovu vyvolání vstupních polí prvku, uloženého v tabulce, pro další editaci. Tuto funkci lze vyvolat klikem na konkrétní řádek tabulky. Vstupní pole jsou následně vyobrazena v levé části obrazovky, ukázka viz obrázek B.8.

B.4 Propojování objektů

Propojování objektů slouží k vytvoření vazby mezi konkrétními objekty. Aplikace umožňuje vytvářet vazbu mezi elementy Change a Artifact a dále pak mezi dvěma Work Unit.

Vazbu mezi objekty lze vytvořit v režimu kreslení čar, do kterého se přepneme stiskem tlačítka označeného šipkou a umístěného nad kreslícím plátnem. Režim kreslení poznáme dle změny kursoru ze šipky na křížek nebo zvýrazněním tlačítka označeného šipkou. Kliknutím na některý prvek umístěný na plátně vybereme nejprve počáteční prvek, ze kterého se vykreslí čára a až po kliknutí na druhý prvek je čára vykreslena. Režim kreslení ukončíme opětovným stisknutím tlačítka označeného šipkou nebo klávesou ESC. Již nakreslenou čáru lze odstranit dvojitým klikem na ni. Ukázka režimu kreslení na obrázku B.9.

Propojení mezi Change a Artifact se vyskytuje pouze ve formuláři pro vyplnění segmentu Configuration, viz obrázek B.10

Propojení Work Unit umožnuje zvolení relace mezi danými dvěma prvky. Směr relace je označen šipkou u koncového prvku. Uprostřed čáry představující spojení se vyskytuje výběrový seznam pro zvolení názvu relace, viz obrázek B.11.

B.5 Klávesové zkratky

Aplikace reaguje na různé kombinace stisku kláves klávesnice.

- Klávesa ESC -> vypnutí režimu kreslení, zavření editačních formulářů,
- Delete -> smazání prvku z plátna či tabulky.
- Kombinací Ctrl + n -> vytvoření nového projektu,
- Ctrl + o -> open,
- Ctrl + s -> save,
- Ctrl + shift + s -> save as,
- Alt + f4 -> ukončení aplikace,
- Ctrl + c -> kopírování prvku plátna,
- Ctrl + v -> vložení prvku do plátna,
- Ctrl + x -> vyjmutí prvku z plátna,
- Crtl + f -> pro validaci procesu.

B.6 Aplikační hlášení

Hlášení jsou zobrazována na principu vyskakujícího okna. Toto okno ve většině hlášení, kromě informování o počtu mazaných položek z tabulky, je rozděleno do dvou částí. V horní části hlášení je popsána nastalá chyba. Spodní část udává jednoduchý návod, jak chybu napravit. S oknem, které hlášení vyvolalo, se nedá pracovat, dokud není zmáčknuto potvrzovací tlačítko okně hlášení nebo pokud není okno jinak zavřené.

B.6.1 Tabulková hlášení

Do této skupiny spadají hlášení týkající se mazání prvků z přehledových tabulek segmentů, viz [10].

První hlášení je uživateli zobrazeno, pokud není v tabulce s výčtem elementů vybrána žádná položka pro smazání, viz obrázek B.12.

Druhý formát hlášení je zobrazen, pokud je uživatelem zvolena alespoň jedna položka z tabulky a stisknuto tlačítko nebo klávesa Delete. Informační okno obsahuje kontrolní seznam označených položek ke smazání. Pokud uživatel souhlasí se smazáním, stiskne tlačítko OK, v opačném případě Cancel, a označené položky zůstanou v dané tabulce, viz obrázek B.12.

B.6.2 Zavření formuláře

Pokud je některý z formulářů zavírán jiným způsobem než pomocí potvrzovacího tlačítka OK. Je uživatel informován, zda chce okno opravdu zavřít a neukládat data z formuláře, uložit změny nebo případně zůstat v daném formuláři, viz obrázek B.15.

B.6.3 Zavření Aplikace

Při jakémkoliv způsobu vypnutí aplikace je uživatel informován o ukončení programu a možností uložení stávajícího stavu projektu. Projekt lze uložit stiskem tlačítka Save umístěného do levé spodní části okna, po uložení je aplikace vypnuta. Zavření aplikace lze odvolat tlačítkem Cancel, případně zavřít bez ukládání tlačítkem OK, ukázka hlášení na obrázku B.14.

B.7 Validace projektu

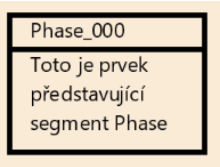
Validaci projektu může každý uživatel provést dle vlastní potřeby volbou umístěnou v menu hlavní obrazovky File položkou Validate. Na výskyt chyby v obsahu procesu vývoje software je uživatel upozorněn hlášením s informací o problému, viz obrázek B.17. Při úspěšné validaci, hlášení obsahuje informaci o úspěchu validace, viz obrázek B.16.

Automatická validace probíhá při každém ukládání vytvořeného procesu nebo jeho načtení ze souboru do aplikace. Pokud validace proběhne úspěšně, soubor je vytvořen pod zadaným jménem, případně jsou data načtena z již vytvořeného souboru bez dalších informací pro uživatele. Pří výskytu chyby umožňuje chybové hlášení obsahující popis chyby danou akci vykonat i s vyskytlou chybou. Pro tuto akci slouží tlačítko Accept, pro odvolání tlačítko Cancel, viz obrázek B.18.

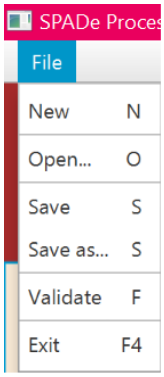
B.8 Práce se soubory

Aplikace umožňuje zpracovávat jak již uložené soubory typu XML pro jejich další úpravu, tak ukládat nově vytvořený proces rovněž do XML. Obě tyto akce lze vyvolat z menu popsaného v kapitole B.2.3. V obou případech je vyvoláno klasické okno operačního systému, pro výběr souboru k načtení nebo zadání jména a vybrání složky pro uložení. Typ souboru pro načtení i uložení je automaticky XML. Toto nastavení nelze změnit na jiné typy souborů.

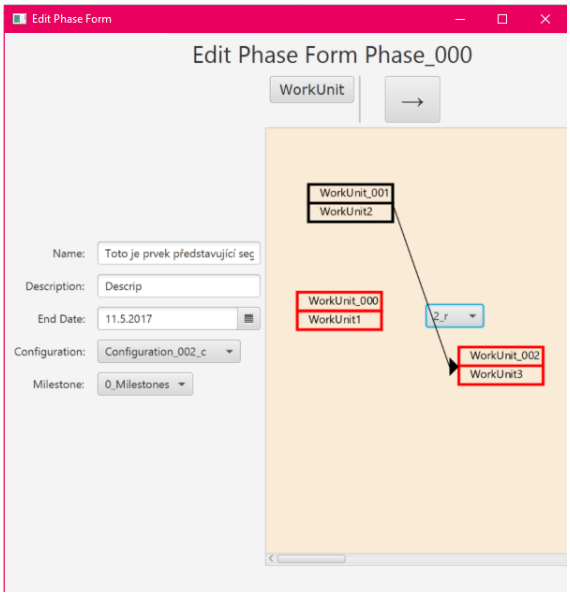
Obrázek B.1: Ukázka hlavní obrazovky



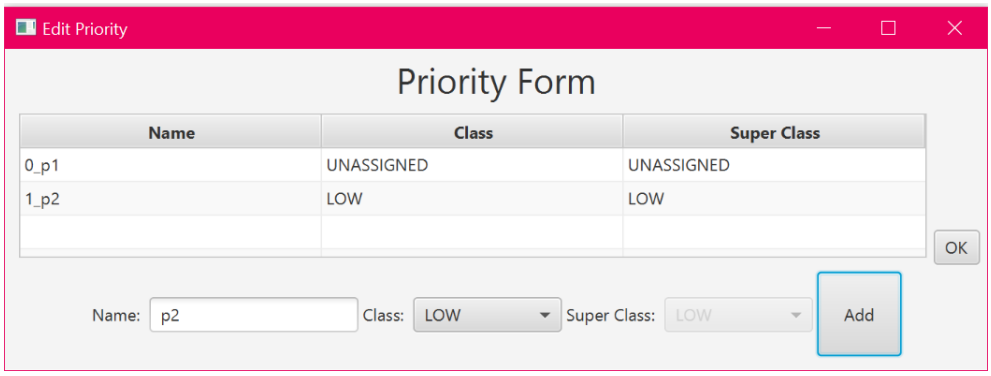
Obrázek B.2: Ukázka aktivního prvku plátna



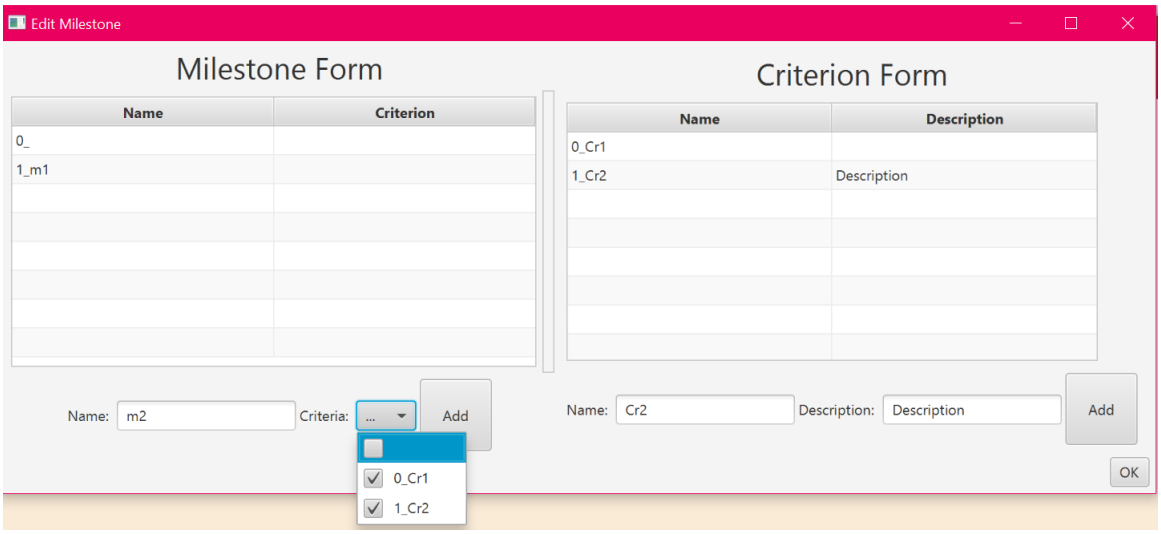
Obrázek B.3: Hlavní menu úvodní obrazovky



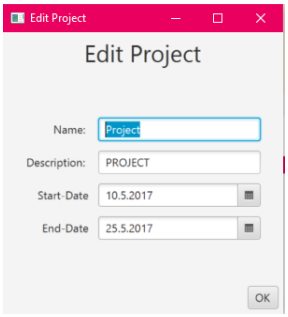
Obrázek B.4: Ukázka formuláře kořenového elementu (Segment Phase)



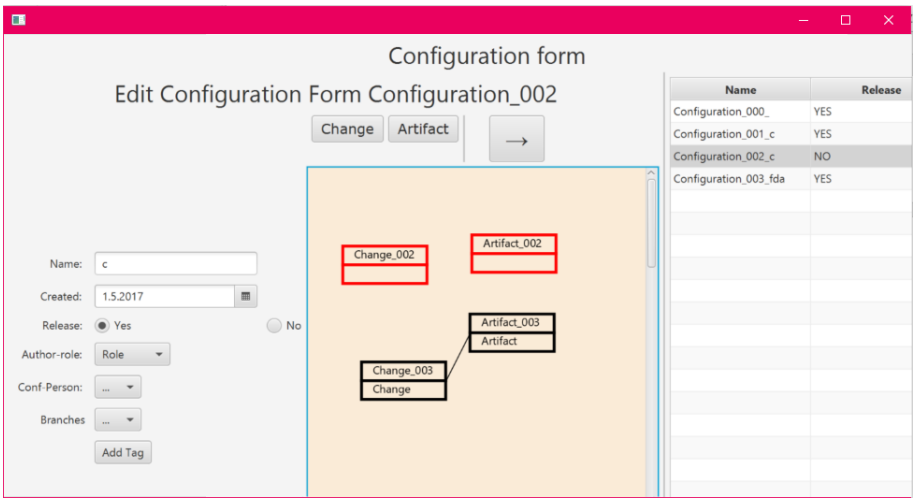
Obrázek B.5: Příklad tabulkového formuláře (Výčtový typ Priority)



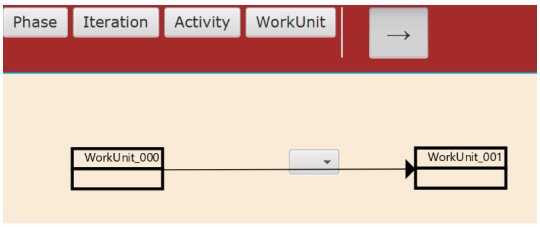
Obrázek B.6: Ukázka formuláře se dvěma tabulkami (Element Milestone)



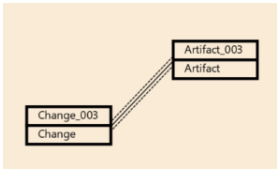
Obrázek B.7: Ukázka klasického formuláře



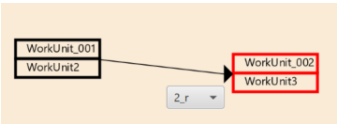
Obrázek B.8: Ukázka formuláře pro element Configurace



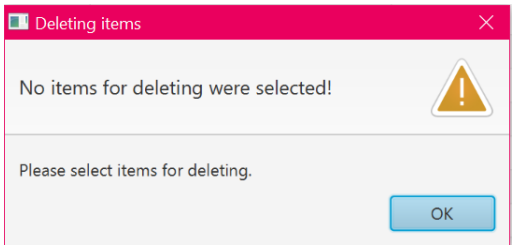
Obrázek B.9: Ukázka režimu kreslení



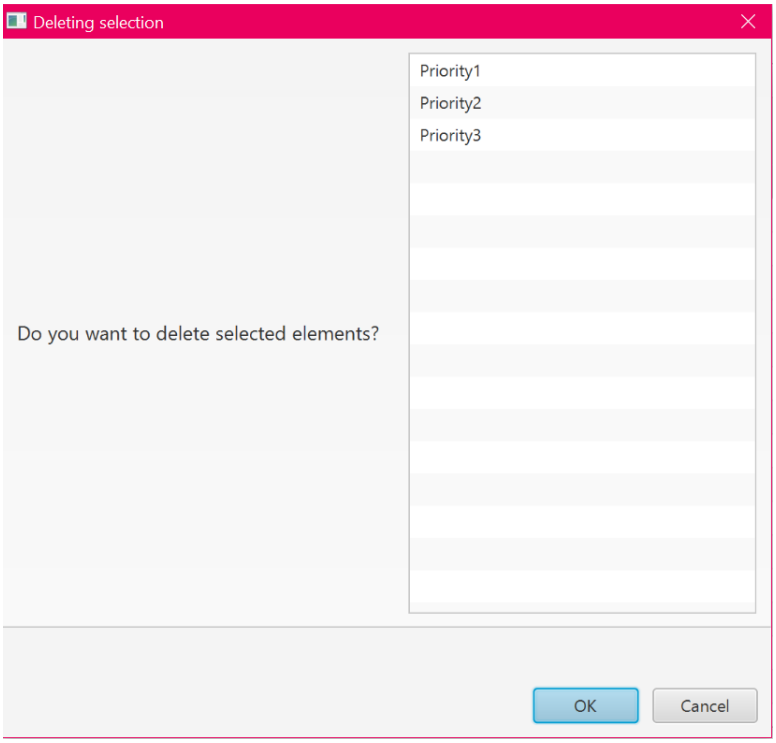
Obrázek B.10: Ukázka propojení prvků plátna



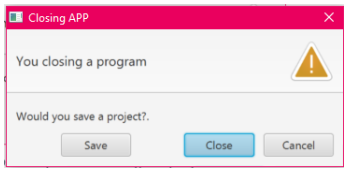
Obrázek B.11: Ukázka vytvoření relace mezi Work Unit



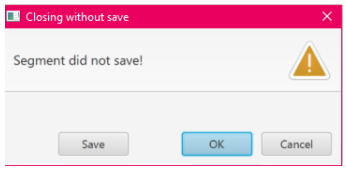
Obrázek B.12: Chybové hlášení o nevybrání položky z tabulky



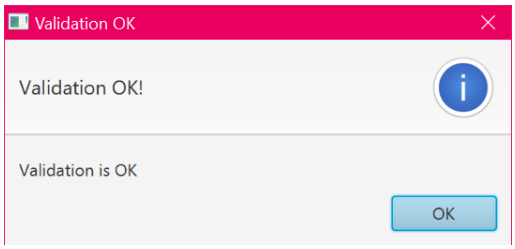
Obrázek B.13: Hlášení s informací o vybraných položkách z tabulky



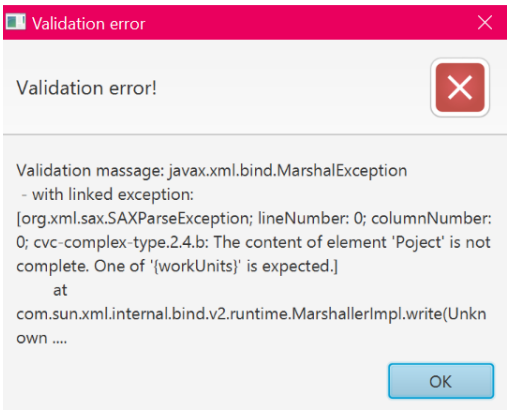
Obrázek B.14: Hlášení o uzavření programu s možností uložení



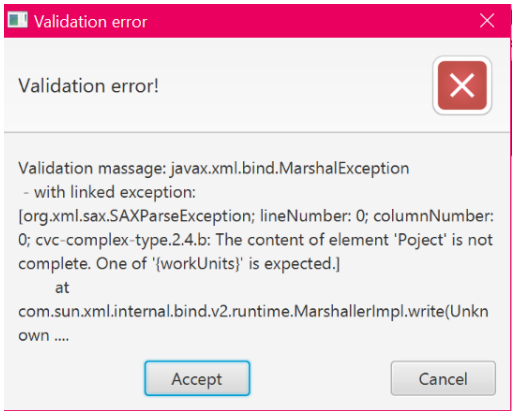
Obrázek B.15: Hlášení o zavření formuláře bez uložení



Obrázek B.16: Úspěšná Validace procesu



Obrázek B.17: Hlášení chyby po ruční validaci



Obrázek B.18: Hlášení chyby při ukládání procesu