VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZŠÍŘENÍ PROJEKTU JENKINS O DYNAMICKÉ WORKFLOW

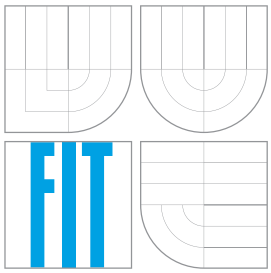
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ SVITÁK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZŠÍŘENÍ PROJEKTU JENKINS O DYNAMICKÉ WORKFLOW

DYNAMIC WORKFLOW EXTENSION FOR JENKINS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JIŘÍ SVITÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK LETKO
SUPERVISOR

BRNO 2012

Abstrakt

Cílem diplomové práce bylo implementovat zásuvný modul pro Jenkins, který umožní řízení spouštění Jenkins úloh pomocí podnikových procesů. Pro podnikové procesy byl použit projekt jBPM 5, který je založen na standardu Business Process Model and Notation 2.0. Pro demonstraci funkčnosti byly navrženy a úspěšně otestovány dva ukázkové podnikové procesy. Výsledkem práce je jBPM zásuvný modul pro Jenkins zveřejněný pod svobodnou licencí.

Abstract

The main of the master's thesis has been to implement a plugin for Jenkins, which enables flow control of launching Jenkins jobs by using business processes. Project jBPM 5, which is based on Business Process Model and Notation 2.0, has been used for business processes. Two sample business processes have been drafted and successfully tested to demonstrate functionality. The main output of this thesis is the jBPM plugin for Jenkins released under free license.

Klíčová slova

Jenkins, Hudson, kontinuální integrace, BPMN, jBPM, podnikový proces, Java, JBoss, Drools

Keywords

Jenkins, Hudson, continuous integration, BPMN, jBPM, business process, Java, JBoss, Drools

Citace

Jiří Sviták: Rozšíření projektu Jenkins o dynamické workflow, diplomová práce, Brno, FIT VUT v Brně, 2012

Rozšíření projektu Jenkins o dynamické workflow

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Zdeňka Letka.

.....
Jiří Sviták
23. května 2012

Poděkování

Děkuji Ing. Zdeňku Letkovi za vedení práce a odborníkům z firmy Red Hat Czech s.r.o. Ing. Vojtěchu Juránkovi, Ph.D. a Bc. Lukáši Petrovickému za jejich rady a připomínky při tvorbě této diplomové práce.

Obsah

1 Úvod **3**

2 Jenkins **5**
2.1 Kontinuální integrace . . . 5
2.2 Architektura systému Jenkins . . . 5
2.3 Tvorba zásuvných modulů . . . 6
2.4 Použití Jenkins v oddělení JBoss Quality Assurance . . . 7

3 Podnikové procesy **9**
3.1 Standard BPMN 2.0 . . . 9
3.2 Projekt jBPM . . . 10
3.3 Události . . . 11
3.4 Aktivity . . . 12
3.5 Větvící konstrukce . . . 14
3.6 Koncepty . . . 14
3.6.1 Procesové proměnné . . . 15
3.6.2 Podmínky . . . 15
3.6.3 Skripty . . . 15
3.7 Drools Guvnor . . . 15
3.8 Designer . . . 16

4 Analýza a návrh zásuvného modulu **17**
4.1 Analýza současných možností . . . 17
4.2 Analýza možností implementace . . . 19
4.3 Návrh zásuvného modulu . . . 20

5 Implementace **22**
5.1 Zásuvný modul . . . 22
5.1.1 JbpmUrlResourceBuilder . . . 23
5.1.2 JenkinsJobWorkItemHandler . . . 24
5.1.3 DescriptorImpl . . . 25
5.1.4 CompleteProcessEventListener . . . 25
5.1.5 Logger . . . 25
5.1.6 WorkItemCause . . . 25
5.2 Interakce s uživatelem . . . 26
5.3 Definice pracovní položky pro spuštění Jenkins úloh . . . 26

6 Testovací podnikové procesy **28**
6.1 Jednoduchý testovací proces 28
6.2 Proces spouštějící reálný projekt 29

7 Problémy **32**
7.1 Problémy s překladem 32
7.2 Náhlá změna repozitáře Mavenu 33
7.3 Tvorba vlastních definic pracovních položek 33
7.4 Mapování procesových proměnných na parametry obslužných úloh 34
7.5 Editory v jBPM Designeru 34
7.6 Konflikt Java překladačů na classpath 34

8 Závěr **36**

A Obsah DVD **40**

Kapitola 1

Úvod

Tato diplomová práce se zabývá studiem serveru pro kontinuální integraci Jenkins, dále standardem Business Process Model and Notation 2.0 (BPMN 2.0) pro podnikové procesy a konkrétní open source implementací jBPM 5. Práce popisuje uvedené technologie, analýzu, návrh a implementaci zásuvného modulu. Výsledkem je implementace zásuvného modulu schopného řídit úlohy nástroje Jenkins pomocí podnikových procesů jBPM a také několik ukázkových testovacích procesů, na kterých byla ověřena funkčnost aplikace.

Nezbytnou součástí současného inženýrství kvality softwaru se stává kontinuální integrace, protože na kvalitu softwaru se klade čím dál tím větší důraz. Kvalita softwaru může být klíčovým elementem v konkurenčním boji. Pro společnost, která má své podnikání založené na distribuci a podpoře svobodného softwaru je kvalita jejích produktů ještě důležitější než u jiných firem. Open source software je stále zatížen nálepkami nekvalitního či nedotaženého softwaru a stále se toto hodnocení projevuje v rozhodnutích mnoha manažerů a architektů počítačového zázemí a infrastruktur v mnoha organizacích. Ovšem open source nezahrnuje pouze komunitní software, ale i plně podporované a vysoce kvalitní produkty. Jako příklad vysoce kvalitního open source produktu lze uvést Red Hat Enterprise Linux[1], na kterém kvůli jeho rychlosti a spolehlivosti běží přední světové finanční burzy – např. New York Stock Exchange, London Stock Exchange [18]. Mimo marketing a osvětové kampaně je tedy nutností také zvyšovat kvalitu softwaru, aby zákazníci měli důvod opouštět proprietární řešení. Nemusí to být pravidlem, ale proprietární software často trpí pomalým opravováním chyb. Hlavně z důvodu, že přístup ke zdrojovým kódům bývá vyhrazen jen určené skupině vývojářů a testerů a není tak dán prostor jiným lidem s jinými znalostmi a s jinými myšlenkovými pochody, kteří nahlíží na daný produkt jinak a mají potenciál najít nové chyby, které původní autoři mohli přehlédnout.

Kontinuální integrace je jeden z postupů extrémního programování a má za cíl usnadnit celý proces testování kvality softwaru. U významných projektů se dnes programuje zároveň s kódem také jeho testovací sada. Testovací sada pak obsahuje testy, které pokrývají co nejvíce poskytované funkcionality. Tyto testy mohou mít podobu od automatizovaného proklikávání webových formulářů až po testování složitých knihovních funkcí expertního systému. Klíčové je zautomatizování celého procesu testování. Například po každém odeslání změn kódu do verzovacího systému se sestavuje nový build projektu, spouští se některá jeho testovací sada a zároveň se aktualizuje programová dokumentace přímo na webových stránkách. Celý proces testování se pak stává transparentnějším i pro projektového manažera. Má lepší přehled o vývoji projektu, práci na něm a může nahlížet do různých statistik.

[1] http://www.redhat.com/rhel/

Jenkins je nejrozšířenější server pro kontinuální integraci, napsán v jazyce Java, ale využívá se na spouštění testovacích úloh pro projekty v libovolném programovacím jazyce. Vznikl na základě programu Hudson [13]. Dodnes se lze setkat s dvojitým značením Jenkins/Hudson. Hudson pochází z firmy Sun Microsystems a jeho vývoj pokračoval pod hlavičkou firmy Oracle, jako jeden z jejích produktů. Jenkins se začal vyvíjet jako nezávislý komunitní projekt. Důvodem pro odštěpení bylo jednání firmy Oracle a neshody s vývojáři právě po převzetí firmy Sun. V průběhu řešení této práce se Oracle vzdal dalšího vývoje Hudsonu, jelikož většina původních vývojářů projekt opustila, a přenechal jej nadaci Eclipse Foundation [7].

Ve firmě Red Hat, konkrétně v divizi JBoss, se Jenkins používá pro automatické spouštění testovacích sad firemních middleware produktů. Psaní těchto testů, jejich spouštění a vyhodnocování má na starosti oddělení inženýrství kvality softwaru. Každý produkt má svůj specializovaný tým, který za tento proces odpovídá. S využitím serveru pro kontinuální integraci se testuje mnoho scénářů použití. Provádí se různé testovací sady pro pokrytí různých oblastí, jako např. testování podporovaných operačních systémů, databází, webových prohlížečů a různých verzích virtuálních strojů pro jazyk Java (JVM). Problematickou částí zůstává vyhodnocování výsledků testů, jelikož vývojář musí testy průběžně kontrolovat. Některé druhy úloh mohou totiž běžet i v řádu hodin a dnů. Dalším neduhem je skutečnost, že pokud v některém bodě začnou úlohy z definované matice testů selhávat, pak není jiné řešení než chybu opravit, zbývající úlohy zrušit a pustit znovu. Tato činnost stojí nějaký čas a znovu se spouští i úlohy, které již dříve proběhly úspěšně. Tím také dochází k plýtvání firemních serverů a jejich výpočetního času, který potřebují používat i jiné týmy. Kapitolou samu pro sebe jsou potíže s alokací sdílených zdrojů, kterými mohou být virtuální počítače s požadovaným operačním systémem či databází.

Cílem této práce je tedy ještě více zautomatizovat celý proces testování a zbavit testery nutnosti stále kontrolovat výsledky úloh. Tohoto lze docílit větším rozvrstvením testovacích sad a řízením jejich spouštění pomocí podnikových procesů. Tato práce směřuje k vytvoření takové úlohy, která umožní spuštění té nejzákladnější testovací sady a na základě jejích výsledků dojde ke spuštění další testovací sady. Ta umožní spuštění další sady atd.

Text práce je členěn následovně: V kapitole 2 je čtenář seznámen s Jenkins, jeho použitím a možnostmi jeho rozšíření. S podnikovými procesy, jejich použitím a implementacemi se čtenář seznámí v kapitole 3. S důvody, proč tyto technologie integrovat a také s předběžnými úvahami, jak by tyto technologie měly spolu fungovat se čtenář seznámení v kapitole 4. Následuje kapitola 5, která se zabývá popisem implementace a výběrem komponent. Demonstrace řešení jak jednoduchých podnikových procesů, tak i ukázky podnikových procesů spouštějících reálné testovací Jenkins úlohy, se nalézají v kapitole věnované testovacím procesům 6. Protože během fází implementace a testování docházelo často různým problémům, které bylo potřeba včas a rozumně řešit, bylo uznáno za vhodné věnovat popisu těchto potíží samostatnou kapitolu 7. V závěru 8 je diskutováno splnění stanovených cílů a celkový přínos práce.

Kapitola 2

Jenkins

Tato kapitola se zabývá serverem pro kontinuální integraci Jenkins. Bude popsán z hlediska jeho použití ve firmě Red Hat, konkrétně v oddělení pro inženýrství kvality softwaru JBoss Quality Assurance (JBoss QA). Dále bude popsána jeho architektura a možnosti vytváření zásuvných modulů.

2.1 Kontinuální integrace

Kontinuální integrace [25] je technika vývoje softwaru spadající do extrémního programování. Vznikla jako reakce na přetrvávající problémy ve fázi integrace tradičního vodopádového modelu vývoje softwaru. Ve své nejjednodušší podobě představuje použití nástroje, který sleduje změny ve verzovacím systému softwarového projektu. Při detekci změny dojde automaticky k překladu zdrojových kódu projektu a spuštění testů. Jestliže se objeví nějaká chyba, systém ihned upozorní vývojáře. Kontinuální integrace však zahrnuje více technik a to sledování historie neúspěšných překladů, automatické sledování kvality kódu, metriky pokrytí kódu testy atd. Tyto údaje mohou být zveřejněny a celý stav procesu vývoje softwaru je pak transparentní.

Cílem kontinuální integrace je snížení rizik při vývoji softwaru poskytováním rychlejší zpětné vazby. Daří se rychleji identifikovat regresní či integrační chyby, což se ve výsledku projevuje menším počtem chyb a rychlejším dodáním produktu.

Jenkins [25] je aplikace pro kontinuální integraci napsaná v jazyce Java. Distribuje se v balíčcích pro distribuce Linuxu, jako `.war` archiv stáhnutelný ze stránek projektu či z repozitáře nástroje Maven. Nasazuje se v aplikačních serverech jako Apache Tomcat [9], JBoss [15] anebo je možné ji spouštět samostatně jednoduchým způsobem `java -jar jenkins.war`, jelikož v sobě obsahuje minimalistický webový kontejner Winstone [16].

Základním kamenem nástroje Jenkins je úloha (job). Úloha představuje soubor kroků vedoucích k nějakému cíli, např. sestavení aplikace. V základu zahrnuje překlad zdrojových kódů a spuštění testů. Existují ale další typy úloh, jako třeba spuštění integračních testů, měření pokrytí kódu testy, generování programové dokumentace nebo dokonce nasazení aplikace na webový server.

2.2 Architektura systému Jenkins

Jenkins se skládá primárně ze sady Java tříd [22], které modelují koncepty sestavovacího systému. Název každé třídy pak odpovídá její funkci a významu, jde například o třídy

`Project`, `Build`, atd. Kořenem objektového modelu je třída `Hudson` a všechny další objekty jsou z něj dosažitelné. Jenkins se dále skládá z rozhraní a tříd, které modelují jednotlivé součásti procesu sestavení, jde například o třídu `SCM` (Software Configuration Management), která umožňuje přístup k verzovacímu systému zdrojových souborů, třídu `Ant` pro sestavení aplikace pomocí stejnojmenného nástroje či třídu `Mailer`, pro zasílání upozornění přes email.

Jenkins má webové rozhraní, které využívá mapování vybraných objektů. Pro jejich mapování na Unified Resource Locator (URL) se používá Stapler [5]. Stapler je knihovna, která umožňuje mapování aplikačních objektů na URL adresy, přičemž toto mapování probíhá automaticky a vytváří intuitivní hierarchii. Singleton `Hudson` je kořenem hierarchie a je tedy navázán ke kořenové ,,/‘‘ URL. Další objekty se navazují podle jejich dosažitelnosti z kořenového objektu. Dále za názvem objektu mohou být jeho metody s případnými argumenty.

Namapované objekty mohou používat více pohledů pro renderování webových stránek. Jenkins používá pro implementaci pohledů Jelly [3], což je nástroj pro převod XML do spustitelného kódu. Mimo jiné lze Jelly využít jako šablonovací systém. Jelly ale v současné době vykazuje znaky mrtvého projektu, takže Jenkins zřejmě do budoucna přejde na jiný šablonovací systém. Existují snahy psát nové moduly bez Jelly.

Ukládání nastavení Jenkins probíhá na souborový systém do jeho domácí složky, kterou lze zjistit ze systémové proměnné `$JENKINS_HOME`. Data objektů se ukládají do adresářů, které kopírují odpovídající objektovou hierarchii. Nestrukturovaná data, jako výstupy konzole, se zaznamenávají jako neformátované textové soubory. Nastavení různých proměnných si Jenkins zapamatuje do Java `.properties` souborů a obsah objektů se perzistuje pomocí technologie XStream [17]. Stav Jenkins objektů včetně objektů zásuvných modulů tak může být snadno uchován. Je třeba dávat pozor na zpětnou kompatibilitu, k tomu lze využít klíčové slovo `transient` pro takový atribut třídy, který se nemá perzistovat. Nebo se dají některé atributy přímo určit jako zpětně kompatibilní a tak se problémům s kompatibilitou vyhnout.

2.3 Tvorba zásuvných modulů

Zásuvné moduly pro nástroj Jenkins se tvoří poměrně snadno, protože se od začátku počítalo s tím, že projekt bude rozšiřitelný a že se objeví spousta zájemců o inovativní rozšíření. Lze využít zásuvný modul Jenkins pro nástroj Maven, se kterým se vygeneruje kostra zásuvného modulu, anebo si lze napsat svůj zásuvný modul od začátku.

Základem pro tvorbu zásuvných modulů jsou body rozšíření (extension points). Zásuvný modul může být třída odvozená od třídy `Plugin`, ale není to nutné. Důležitější je, jestli v sobě obsahuje implementaci bodu rozšíření, díky kterému se pak zásuvný modul v instanci Jenkins zaregistruje. Implementace bodu rozšíření se označuje anotací `@Extension`. Podle typu bodu rozšíření se pak na příslušném místě ve webovém rozhraní Jenkins objeví příslušná položka v příslušném formuláři konfigurace. Zásuvný modul pak má díky objektové hierarchii umožněn přístup do libovolného objektu Jenkins a může tedy získávat libovolné informace nebo volat vybrané metody vykonávající požadovanou akci. Jenkins při svém běhu hledá všechny třídy týkající se konkrétní akce systému, které ve svém nitru implementují nějaký bod rozšíření a u nalezených tříd spustí metodu `perform`. Tímto způsobem se spouští samotný kód zásuvného modulu, který již provádí požadovanou činnost.

Například ukázkový zásuvný modul HelloWorld [23] implementuje jeden krok sestavení (build step, třída `Builder`). Běžná volná Jenkins úloha (free-style project) se skládá ob-

vykle z jednoho nebo více kroků sestavení. Vybíráme-li kroky sestavení, tak při nasazení HelloWorld pluginu, můžeme vybrat právě tento krok sestavení a nakonfigurovat jej. Funguje jednoduše, nastavuje se jen jméno, jakým má tento zásuvný modul pozdravit uživatele ve webové konzoli. Po dokončení konfigurace zbytku Jenkins úlohy je možné tuto úlohu spustit. Její spuštění se kromě přípravných činností dělí na spouštění jednotlivých kroků sestavení v řadě za sebou, jak byly definovány. Jakmile tedy přijde řada na HelloWorld, je vypsán odpovídající pozdrav. Na tomto zásuvném modulu je tedy vidět, jak se nastavuje a spouští vlastní kód v systému Jenkins. Je ukázáno i využití globálního nastavení zásuvného modulu. V globální konfiguraci Jenkins lze nalézt i úsek, který je věnován zásuvnému modulu HelloWorld. Zde se určuje, zda má být pozdrav před oslovením anglicky nebo francouzsky.

2.4 Použití Jenkins v oddělení JBoss Quality Assurance

Pro lepší pochopení práce a její motivace bude stručně zmíněna případová studie použití Jenkins v oddělení JBoss Quality Assurance (JBoss QA) firmy Red Hat. Firma Red Hat založila své podnikání výhradně na open source software a předplatném za podporu. Podpora je poskytována na produktizovaný software, což bývá v základu komunitní projekt rozšířený o podporu a opravy chyb tak, aby byl použitelný u zákazníka. Jedním z klíčových úspěchů této firmy není jen použití netradičního modelu podnikání, ale také velký důraz na vysokou úroveň podpory a kvality softwaru. JBoss QA se zabývá systematickým testováním produktů z oblasti middleware, což zahrnuje spektrum produktů (v závorce je uveden příslušný komunitní projekt):

- Application Platform (JBoss AS),
- Business Rules Management System (Drools, jBPM),
- Data Services Platform (Teiid),
- Developer Studio (JBoss Tools),
- JBoss Operations Network,
- Portal Platform (GateIn),
- SOA Platform (JBoss ESB),
- Web Framework Kit (Richfaces, Seam, Weld),
- a další.

Jde tedy primárně o testování aplikací v jazyce Java. Kromě psaní a pravidelného provádění jednotkových testů, integračních testů a výkonnostních testů se dělají jednorázové testy oprav chyb přímo pro zákazníka pro již nasazený software.

V následující případové studii [20] se popisují důvody, které vedly v tomto oddělení k využití kontinuální integrace založené právě na Jenkins. Jak bylo výše uvedeno, pracuje se s velkým množstvím open source projektů. Navíc se týmy snaží dodržovat pravidlo vydávat nové verze brzo a často. Vzniká tak požadavek na možnost práce s mnoha větvemi aplikace. Každá aplikace pak má specifické požadavky a potřebuje být testována na požadovaných verzích operačních systémů, virtuálních strojů (JVM), prohlížečů, databází a Java

repozitářích (JCR). Vše uvedené je již silným argumentem pro využití kontinuální integrace. Dříve se v JBoss QA využíval CruiseControl [12], ale časem se požadovaly nová vylepšení, která chyběla. Volba padla na tehdejší Hudson. Hudson oproti CruiseControl nabízel nové možnosti:

- webové rozhraní pro konfiguraci a správu více různých projektů,
- běh více úloh najednou v distribuovaném prostředí, což je umožněno komunikačním modelem master-slave,
- snadná integrace mnoha nástrojů jako: Concurrent Versions System (CVS), Apache Subversion (SVN), JUnit, Ant, Maven, Code Coverage, atd.

Jak dále článek [20] píše, spolupráce s komunitou se vyplatila a podařilo se opravit mnoho nedostatků a chyb. JBoss QA má nádale více nápadů a požadavků na vylepšení Jenkins, jedním z nich je i cíl této práce.

Kapitola 3

Podnikové procesy

Tato kapitola obsahuje úvod do problematiky podnikových procesů a stručně popisuje standard Business Process and Model Notation (BPMN) 2.0. Dále se představí engine pro podnikové procesy jBPM ve verzi 5, což je komunitní open source projekt, který implementuje tento standard. Jednotlivé konstrukce standardu BPMN budou popsány na notaci podle projektu jBPM. Kromě samotného engine se čtenář seznámí s nástroji Drools Guvnor a Designer, které úzce spolupracují s jBPM.

3.1 Standard BPMN 2.0

Business Process Model and Notation 2.0 (BPMN 2.0) [24] je diagramový jazyk pro popis modelů podnikových procesů. Je důležitý nejen tím, že nabízí větší možnosti než jiné podobné zápisy pro podnikové procesy, ale hlavně se jedná o průmyslový standard. Tento standard podporují mnozí dodavatelé softwaru a používají jej mimo aplikační vývojáři a doménoví experti. Standard udržuje skupina Object Management Group (OMG), která mimo jiné spravuje standardy jako Unified Modeling Language (UML) či Common Object Request Broker Architecture (CORBA). BPMN 2.0 nenabízí jen jednotné značení procesů pomocí stejných tvarů a symbolů, ale také popisuje jednotnou XML reprezentaci pro diagramy, čímž je dosažena nezávislost na konkrétní implementaci engine. Existují tedy dva základní principy:

- Každý BPMN diagram by měl mít jen jednu jednoznačnou interpretaci. Logika procesu by tedy měla být popsána jasně a kompletně diagramem samotným.
- Každý BPMN diagram by měl mít jednu a pouze jednu XML reprezentaci. Jinak není možné zaručit nezávislost mezi nástroji a aplikacemi.

Bohužel není možné zaručit ani jeden z principů pouhým striktním dodržováním standardu. Uvedené principy lze splnit dodržováním jistých konvencí a pravidel. Tyto konvence lze opět nalézt v literatuře, například v knize *BPMN Method and Style* [24]. V následující sekci bude popsána konkrétní implementace BPMN 2.0 – projekt jBPM 5. Ukázka jednotlivých konstrukcí jazyka BPMN 2.0 bude vysvětlena na obrázcích a konstrukcích používaných v modelovacím nástroji projektu jBPM pro vývojové prostředí Eclipse. Samotné jBPM pak poskytuje běhové prostředí, ve kterém je spuštěn samotný podnikový proces.

3.2 Projekt jBPM

jBPM [26] je flexibilní nástroj pro správu a spouštění podnikových procesů organizace. Snaží se najít průnik mezi požadavky netechnicky vzdělaných lidí a aplikačních programátorů. Tradiční BPM přístupy se často zaměřují na lidi s netechnickým vzděláním. jBPM se naopak pokouší nabídnout dvojí přístup, který vyhovuje zároveň doménovým expertům i vývojářům. jBPM ve verzi 5 podporuje poslední verzi standardizované specifikace BPMN 2.0. jBPM patří do širšího celku Drools a celý projekt nyní nese oficiálně název *Drools and jBPM*. Současné jBPM 5 vzniklo spojením projektu jBPM spolu s projektem Drools Flow. Drools je platforma pro integraci obchodní logiky v podnikovém informačním systému, která obsahuje ještě další součásti:

- Drools Expert – na pravidlech založený expertní systém implementující ReteOO algoritmus, který je určen pro rychlé zpracování mnoha pravidel a faktů v systému, využití např. pro hromadné zpracování žadatelů o hypotéku, řízení cenové politiky, řízení spotřeby, atd.,
- Drools Fusion – systém na zpracování složitých událostí (Complex Event Processing – CEP), využití např. pro algoritmické obchodování, detekci podvodu ve finančních transakcích, atd.,
- Drools Guvnor – webová aplikace na správu obchodních pravidel, podnikových procesů a jiných aktiv,
- Drools Planner – systém pro řešení plánovacích problémů, používá algoritmy pro prohledávání stavového prostoru jakými jsou tabu search, simulované žíhání aj. (využití např. při plánování školních rozvrhů, směnového provozu, atd.).

Drools spolu s jBPM tedy tvoří integrovanou platformu, která je schopna spravovat a řešit v podnikovém informačním systému celou oblast obchodních pravidel, politik a podnikových procesů organizace. Tyto projekty jsou silně provázané, neboť sdílejí společné aplikační rozhraní (tzv. Knowledge API). Tím je usnadněna jejich integrace a spolupráce, např. na základě detekované události v proudu příchozích událostí pomocí Drools Fusion je zahájen příslušný jBPM podnikový proces apod. Samotné jBPM závisí na Drools Expert, protože používá pravidla tohoto expertního systému pro vyhodnocování některých podmínek a omezení. Pro zájemce z komerční sféry pak je k dispozici plně podporovaná produktizovaná verze celého projektu.

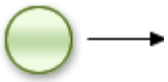
V následujících podsekcích budou popsány základní konstrukce a koncepty jazyka BPMN 2.0 a to na notaci, kterou používá právě jBPM 5. Existují tři základní druhy konstrukcí a to události, aktivity a větvící konstrukce. Události modelují konkrétní událost v systému, značí místa, kde proces začíná, končí, případně kde může vzniknout událost již za běhu procesu. Aktivity představují různé akce, které se mají vykonat během provádění procesu, mají jeden vstupní bod a výstupní bod. Větvící konstrukce, jak již název napovídá, značí místa, kde se mění tok v procesu, například výběrem jedné nebo několika z dalších možných pokračování toku. Všechny tyto konstrukce představují uzly grafu, které jsou propojeny orientovanými hranami a tvoří pak jako celek orientovaný graf. Ukázka takového diagramu je na obrázku 4.1, tato kapitola dále pokračuje výčtem a popisem základních typů konstrukcí [6]. Vysvětleny budou také jednotlivé důležité koncepty, jako jsou procesové a globální proměnné, podmínky a psaní vlastních krátkých kusů kódu přímo v definici podnikového procesu.

3.3 Události

Popis standardu BPMN bude zahájen popisem základních událostí. Každá událost má definovaný svůj jednoznačný identifikátor *Id*, název *Name*, případně další více specifické vlastnosti.

- Startovací událost

  Každý podnikový proces má právě jednu startovací událost, ve které začíná tok procesu. Tato událost nemá žádné vstupy, ale má právě jeden výstup.



- Ukončovací událost

  Tato událost ukončuje proces. Každý proces byl měl mít nejméně jednu ukončovací událost. Má jeden vstup, ale žádné výstupy. Kromě *Id* a *Name* má parametr *Terminate*, který určuje, jestli událost ukončuje pouze větev procesu nebo celý proces. Některé procesy mohou mít totiž více paralelně bežících větví.

- Chybová ukončovací událost

  Chybová událost nastane při výjimečné situaci v procesu. Pokud není definována obslužná rutina pro daný chybový stav, pak je běh podnikového procesu ukončen. Příčina selhání je předávána v proměnných *FaultName* a *FaultVariable*.

- Časovaná událost

  Pracuje jako časovač. Pokud tok procesu dojde do této události, dojde k pozastavení toku na definovaný časový úsek *TimerDelay*. Po jeho uplynutí se pokračuje v toku podnikového procesu. Je možné definovat také opakování časovače parametrem *TimerPeriod*.

- Signálová událost

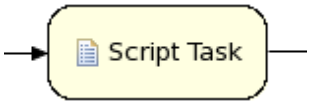
  Je použitelná jako reakce na interní nebo externí událost v systému. V proměnné *EventType* je určen druh naslouchané události a v *VariableName* jsou data spojená s detekovanou událostí.

3.4 Aktivity

Existuje více druhů aktivit (nebo také úloh), každá provádí jinou činnost. Uživatelé si mohou poměrně snadno definovat vlastní aktivity, aby lépe vyhovovaly potřebám uživatele.

- Skriptovaná úloha

  Skriptovaná úloha obsahuje kód v jazyce Java nebo dialektu MVFLEX Expression Language (MVEL) [1]. Tento kód může provádět libovolnou činnost a manipulovat s proměnnými procesy a s globálními proměnnými. Skriptovaná úloha by neměla obsahovat kód, jehož provádění může trvat delší dobu. Měla by tedy provést kód a okamžitě pokračovat v toku procesu.



- Obslužná úloha

  Slouží pro zavolání vzdálené služby, která provede nějakou činnost. Takováto služba může být například odeslání mailu, zaznamenávání činnosti, atd. Je zde nechán prostor pro aplikační programátory, aby si naprogramovali vlastní doménově specifickou činnost ve své aplikaci. Tento typ úlohy je klíčový pro tuto diplomovou práci. Koncept

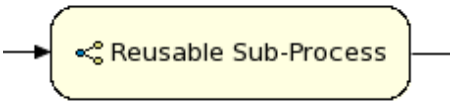
  obslužných úloh nabízí aplikačním programátorům široké možnosti využití. V případě této práce se obslužná úloha použije ke spuštění Jenkins úlohy.

- Uživatelská úloha

  Uživatelská úloha způsobí pozastavení procesu a čeká na odpověď od uživatele. Zpravidla se od uživatele se vyžaduje vyplnění a odeslání nějakého webového formuláře. Jde tedy o speciální případ obslužné úlohy.

- Podproces

  Umožňuje v procesu spustit jiný podnikový proces. V toku hlavního procesu se pokračuje po ukončení činnosti podprocesu. Je tak umožněna znovupoužitelnost některých již hotových procesů.

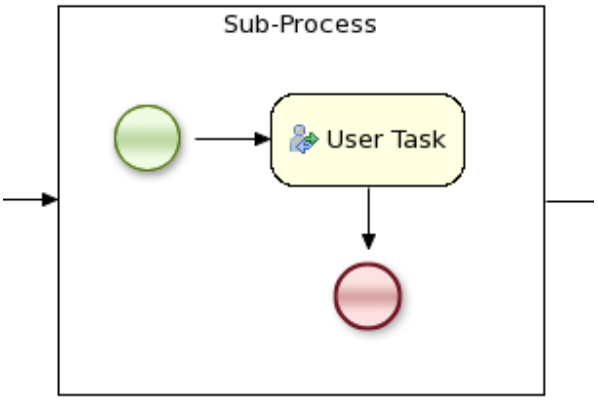


- Úloha pro vyhodnocení obchodních pravidel

  Obsahuje množinu obchodních pravidel, které budou vyhodnoceny expertním systémem. Pravidla je nutno definovat ve vlastních souborech. Popis použitelných pravidel a jejich možnosti přesahuje rámec tohoto semestrálního projektu.

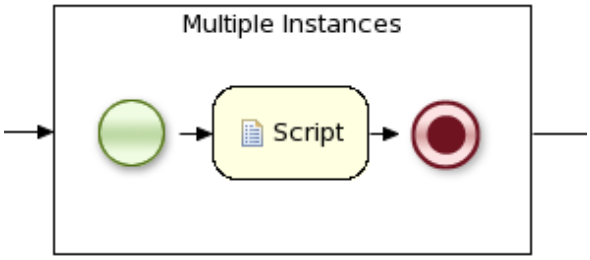
- Vnořený podproces

  Jde o rámec, který představuje aktivitu, která uvnitř sebe má definovaný nový proces. Má také své vnitřní procesové proměnné.



- Podproces s více instancemi

  Je to speciální druh podprocesu, který spustí svůj proces víckrát najednou pro každý element v definované kolekci. V dalším toku pokračuje až po dokončení všech instancí.

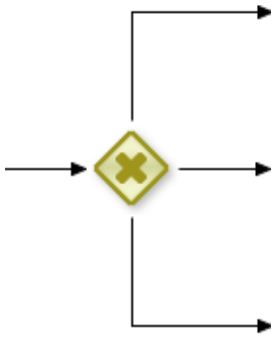


3.5 Větvící konstrukce

Umožňují změnu toku v procesu na základě vyhodnocení podmínky. Umožňují také paralelní spuštění více větví najednou.

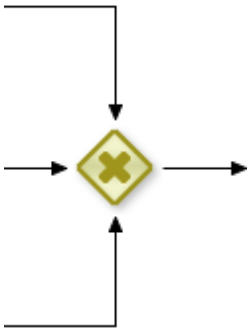
- Rozdělení větví

  Tato brána (gateway) umožňuje rozdělení toku podnikového procesu. Existují tři základní typy těchto bran. Brána typu AND spustí všechny větve najednou paralelně. Brána typu XOR vybere právě jednu větev a to na základě vyhodnocení omezení podle priority, které jsou specifikovány na každé odchozí větvi. Brána typu OR vybere právě ty větve, pro které je daná podmínka vyhodnocena jako pravdivá.



- Spojení větví

  Tato spojovací brána umožňuje synchronizovat spojení toků více větvé do toku jedné větve. Brána typu AND čeká na dokončení všech větví. Bráně typu XOR stačí pro pokračování dokončení pouze jedné větve.



3.6 Koncepty

Samotný výčet použitelných konstrukcí není pro plné pochopení problematiky podnikových procesů dostačující. To samé platí i pro samotné praktické použití podnikových procesů v konkrétní implementaci, v našem případě jBPM.

3.6.1 Procesové proměnné

Procesové proměnné (process variables) jsou základem pro výměnu dat v podnikových procesech. Jejich použití je volitelné, ale v této diplomové práci se bez nedá obejít. Specifikují se již při startu procesové instance. Nejprve se tedy vytváří instance jednotlivých procesových proměnných. Ty se následně umístí do kolekce, která se předá instanci podnikového procesu při jejím odstartování. Procesové proměnné se hodí v mnoha situacích. Nejčastěji se k nim přistupuje z míst, kde se píší skripty. Zde se skripty myslí krátké úseky kódu v jazyce Java nebo MVEL. Lze k nim přistupovat přímo `person.setAge(10);` anebo přes kontext `kcontext.setVariable(variableName, value);`. Procesové proměnné se také mapují jako vstupy nebo výstupy nejrůznějších typů úloh. Lze k nim ještě přistupovat z uzlů událostí.

Existují také globální proměnné, ty se však definují pro celé sezení (Session). V Javě se nastavují pomocí `ksession.setGlobal(name, value);`, přímo v procesu pak přes kontext `kcontext.getKnowledgeRuntime().setGlobal(name,value);`.

3.6.2 Podmínky

Jsou logické výrazy, které jsou typu `Boolean` a nabývají tedy pouze hodnot `true` nebo `false`. Zapisují se v jazyce Java (`return person.getAge() > 20;`), v dialektu MVEL (`return person.age > 20;`) nebo v jazyce pravidel Drools Expert (`Person(age > 20)`). Nejčastěji se podmínky využívají ve větvících bránách, pro každou větev se vyhodnotí výstupní podmínky a tok podnikového procesu pokračuje podle typu brány ve splněné větvi či větvích.

3.6.3 Skripty

Skripty (Action Scripts) jsou krátké kusy kódu určené pro jednoduché výpočty a operace. Používají se ve skriptovaných úlohách či v akcích na začátku nebo po ukončení úlohy. Skriptuje se opět v jazyce Java nebo dialektu MVEL. Pro složitější operace anebo pro úkoly, které zaberou delší čas je lepší používat obslužné úlohy (service tasks).

3.7 Drools Guvnor

Drools Guvnor je webová aplikace sloužící pro centrální správu definic pravidel (rules), podnikových procesů (business processes) a jiných aktiv (assets). Je napsána v jazyce Java z využitím frameworku Google Web Toolkit (GWT). Pro uchovávání všech typů aktiv se využívá dokumentové úložiště založené na standardu Java Content Repository (JCR), které pak běží nad vybranou relační databází. Mezi výhody patří i správa verzí jednotlivých aktiv.

V této práci je Guvnor využit nejen pro správu a ukládání vlastních podnikových procesů, ale také pro správu definic pracovních položek (work items) včetně jejich ikon. Guvnor je určen nejen pro vývojáře, ale vzhledem k dostatku pokročilých editorů s nápovědami i pro netechnicky orientované uživatele, jakými jsou experti na konkrétní doménu, nebo obchodní analytici. Guvnor poskytuje také několik možností integrace s reálnými podnikovými a informačními systémy. Lze využít:

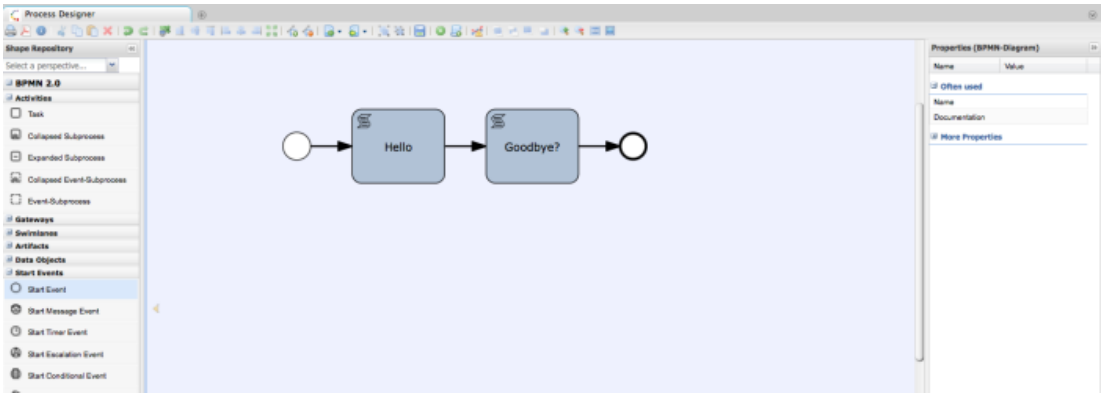
- Webové služby REST,
- Protokol WebDAV,

- Knowledge Agent.

V této práci se pro integraci využije přístup do repozitáře skrz protokol WebDAV. Pro samotný návrh podnikových procesů pro řízení nástroje Jenkins se využije další webová aplikace – Designer, kterou je spouštěna přímo z Guvnoru.

3.8 Designer

Designer je webová aplikace určená pro návrh podnikových procesů. Typicky se spouští automaticky z Guvnoru, když uživatel chce vytvářet nový BPMN 2.0 podnikový proces, či pokud chce již existující proces upravovat. Obsahuje mnoho uživatelsky přívětivých funkcí a nástrojů pro usnadnění návrhu podnikových procesů. Na obrázku 3.1 je snímek obrazovky Designeru. Pro práci jsou nabízeny dva hlavní panely. Levý panel obsahuje minimální a pl-



Obrázek 3.1: Designer – webový nástroj pro návrh podnikových procesů

nou paletu konstrukcí BPMN 2.0, zejména různé typy událostí, úloh a bran. Na plátno, které je uprostřed obrazovky mezi panely, se umísťují BPMN konstrukce přetahováním pomocí chycení a puštění. Většinou stačí umístit na plátno jen *Startovací událost* a zbytek procesu se nakliká pomocí uživatelsky přívětivého rozhraní přímo na plátně. V pravém panelu se nastavují vlastnosti jak samotného procesu, tak i všech ostatních typů konstrukcí. Designer zahrnuje v horní liště také nástroje pro validaci správnosti vytvořených procesových definic, pro migraci definic podnikových procesů vytvořených ve starším ještě nestandardizovaném jBPM 3, pro generování formulářů pro uživatelské úlohy, pro stahování definic obslužných úloh z centrálního repozitáře a pro další funkce.

Kapitola 4

Analýza a návrh zásuvného modulu

Tato kapitola se na základě poznatků získaných v předchozích kapitolách zabývá návrhem samotného zásuvného modulu. Důraz je kladen na konkrétní případy použití, ve kterých má budoucí zásuvný modul své opodstatnění, protože si tak lze lépe uvědomit, co všechno bude potřeba později implementovat. Nejdříve je uvedeno podrobnější zhodnocení stávajících možností pro dynamické řízení úloh v nástroji Jenkins. Následuje analýza možností implementace. Nakonec je proveden konkrétní návrh, který slouží jako základ pro implementaci. V průběhu řešení práce bylo potřeba reagovat na požadavky a měnící se situace, proto návrh byl původně jiný a až později dospěl do stavu popsaném na konci této kapitoly.

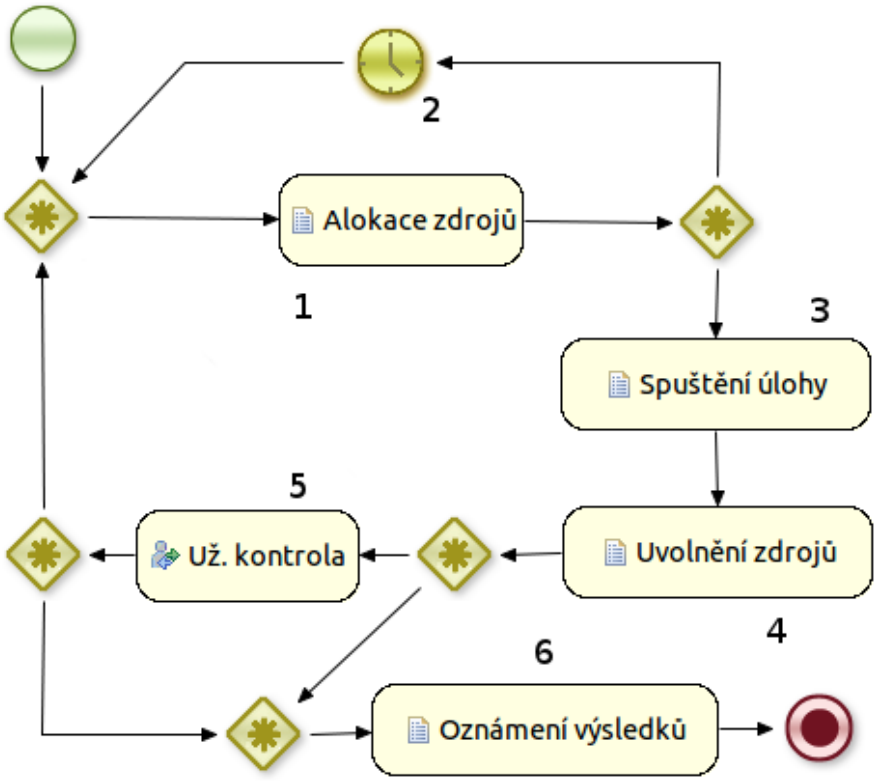
4.1 Analýza současných možností

V současné době Jenkins neobsahuje možnosti pro plnohodnotné řízení úloh pomocí dynamického řízení toku. Existuje pouze možnost jako nakonfigurovat některé úlohy tak, že jsou spuštěny, jakmile se dokončí jiné vybrané úlohy. Naproti tomu využití podnikových procesů má následující výhody [19]:

- Pomocí podnikových procesů lze modelovat teoreticky neomezeně složité řízení toku a to s využitím grafického návrháře takových procesů. Tím je usnadněna reprezentace a čitelnost řízení toku.
- BPMN umožňuje využít uživatelských úkolů (human tasks), které zapojují reakci od uživatele do celého procesu. Jenkins sám o sobě po spuštění úlohy takovou interakci s uživatelem nepodporuje.
- BPMN podporuje také skriptované úkoly (script tasks), které umožňují spuštění libovolného Java či MVEL kódu během procesu.
- Podnikové procesy podporují verzování, takže v případě nasazení nové verze podnikových procesu nejsou běžící procesy nijak zasaženy.
- Řízením toku lze popsat i celý životní cyklus aplikace včetně automatizovaného testování, vydání a nasazení. Jenkins se tak může stát centrálním uzlem informací o celém projektu díky výborné integraci s nástroji na verzování kódu a správu chyb.

Existují dva zásuvné pluginy, které umožňovaly dynamické řízení úloh, ale tyto zásuvné moduly mají zásadní nedostatky. Jedná se o dva několik let neaktivní projekty, které mají každý po jednom autorovi. Jako důsledek svého stáří v nejnovější verzi Jenkins nefungují a navíc pracují se starými verzemi knihoven pro podnikové procesy. Jeden se jmenuje JBPM plugin [19] a pracuje se starou, nepoužívanou a nestandardizovanou verzí jBPM. Další se jmenuje Drools plugin (pro Jenkins) [2] a ten pro změnu pracuje s projektem Drools Flow, který byl opět nestandardizovaný a fakticky zanikl po začlenění jBPM 5 do platformy Drools. Po analýze jejich kódu a výše uvedených problémů se jejich využití a případné přepsání jeví jako příliš komplikované. Proto je žádoucí napsat nový zásuvný modul od začátku.

Dosud byly vyjmenovány důvody pro využití BPMN při řízení sestavovacího procesu v projektu Jenkins. To nejsou ale všechny důvody, neboť poptávka po použitelném řešení vzešla z firmy Red Hat a tato firma má navíc své vlastní požadavky vyplývající z jejího způsobu používání Jenkins při testování softwaru v oddělení JBoss QA.



Obrázek 4.1: Návrh podnikového procesu využitelného v prostředí JBoss QA firmy Red Hat

Na obrázku 4.1 je vidět příklad možného využití v prostředí firmy Red Hat. Celý postup toku procesu bude detailněji vysvětlen v jednotlivých krocích.

1. Po startu podnikového procesu a průchodu první větvící konstrukcí (nebo také brány) se spustí alokace zdrojů. V JBoss QA se používá Jenkins v distribuovaném heterogenním prostředí. Server s Jenkins se označuje jako *Master*. Další servery se nazývají *Slave*. Na těchto virtuálních strojích běží různé operační systémy a jejich různé verze, například Red Hat Enterprise Linux, Windows, Solaris a AIX. Tvoří tak heterogenní prostředí. Před startem se zajišťuje, že je potřebný počet virtuálních strojů volný a

dále, že jsou volné další potřebné virtuální stroje, na kterých například běží specifické databáze. Pokračuje se bodem 2.

2. Pokud alokace zdrojů v požadovaném rozsahu selže, pak se inicializuje časovač a po určitém intervalu je pokus o alokaci zdrojů opakován bodem 1, jinak se pokračuje bodem 3.

3. V případě úspěchu alokace jsou spuštěny samotné úlohy, které spouští testovací sady. Pro jednoduchost je v obrázku spuštění jedné úlohy.

4. Po dokončení testů je potřeba uvolnit zdroje na virtuálních strojích. Serverovou farmu pro testování využívá více týmů najednou a je třeba se chovat ohleduplně a šetřit společnými výpočetními zdroji. V případě selhání testů je zavolána uživatelská úloha v bodě 5. V případě úspěchu se pokračuje bodem 6.

5. Tester zkontroluje stav, pokud je možná rychlá oprava testu či nastavení, tak ji provede a obnoví proces testování, začínáme tedy znovu od bodu 1. Pokud si myslí, že výsledky jsou dostatečně uspokojivé, přejde k bodu 6.

6. Vše dopadlo dobře a odešle se zpráva testerům, že proces testování skončil úspěšně.

Zásuvný modul by tedy měl umožnit spuštění libovolného podnikového procesu a měl by podporovat práci se všemi běžnými BPMN 2.0 konstrukcemi. Mimo to by však měl nabízet navíc aktivitu, která bude spouštět Jenkins úlohu a vracet do procesu její výsledný stav.

4.2 Analýza možností implementace

Na základě analýzy Jenkins, jeho architektury, podpory tvorby zásuvných modulů, standardu BPMN, projektu jBPM[1] a na základě analýzy požadavků odborníků z firmy Red Hat byly promyšleny následující návrhy.

V první uvažované možnosti by základem byla implementace třídy `JbpmJob`, která by vznikla zděděním třídy `Job` v Jenkins. V této třídě by byl implementován bod rozšíření, který by zajistil vykreslení nového typu úlohy vedle již běžně používaných základních úloh. Těmito základními úlohami pro sestavení projektů jsou v současné době:

- Volný projekt (free-style software project)
- Maven projekt (maven2/3 project)
- Projekt s více konfiguracemi

Tato možnost by byla poměrně náročná, vyžadovala by hodně programátorské úsilí, zejména s doprogramováním funkcionality, která je v ostatních úlohách již vyřešená.

V druhé možnosti se nabízí implementovat zásuvný modul jako bod rozšíření (extension point) pro krok sestavení (build step). Tento krok sestavení by byl použitelný v jiných typech úloh, jakými je volný projekt nebo projekt s více konfiguracemi. Jedna úloha by tak mohla mít více kroků sestavení, takže v rámci jedné úlohy by šlo zavolat více akcí, dokonce i více podnikových procesů za sebou. Předchozí možnost by se dala simulovat vytvořením

[1] Existují i jiné workflow systémy jako IBM Business Process Manager Standard [11], Activiti [8], Bonita [10] nebo Intalio [14], ale vzhledem k tomu, že zadavatelem práce je firma Red Hat, je použito její řešení.

volného projektu s jedním krokem sestavení. Proto pro implementaci zásuvného modulu bude vybráno vytvoření nového kroku sestavení.

Spuštěním úlohy by v metodě `perform` byla spuštěna činnost podnikového procesu. Existují dvě možnosti implementace.

1. Podnikový proces by běžel na stejném serveru spolu s Jenkins. Byly by v něm spouštěny jednotlivé úlohy podnikového procesu, třeba i vzdáleně na *Slave* strojích, které by přímo volaly metody Jenkins pro spuštění, pozastavení nebo zastavení jiných úloh definovaných v Jenkins. Stejně tak by skrz API objektového modelu Jenkins získávaly informace o stavu dokončených úloh. Výsledkem by byl zásuvný modul, který by byl kompletně integrovaný v Jenkins. Jelikož navrhovaná definice jBPM úlohy by byla poměrně rozsáhlá, bylo by nutné také dopsat části kódu, které by na stránce úlohy zobrazovaly historii procházení stavů jBPM procesu. Potenciální interakci s uživateli během provádění podnikového procesu by bylo potřeba řešit buď poměrně náročným zásahem do zdrojových kódů Jenkins, případně přinejmenším jejich náročným studiem. Druhou možností je využít pro tyto účely externí aplikace, např. jBPM console z balíku jBPM. Nebo si napsat vlastní externí aplikaci.

2. Podnikový proces by běžel na jiném serveru. Jenkins podporuje komunikaci přes webové služby Representational State Transfer (REST). Externí podnikový proces by tak pomocí webových služeb volal spuštění jednotlivých již definovaných úloh vzdáleně. Vzdáleně by také vyhodnocoval jejich stav, případně by je vzdáleně ukončoval. V případě uživatelských úloh by uživatel určoval další postup pomocí webových formulářů právě na externí aplikaci. Podobně by vypisoval historii procházení stavů či aktuální stav procesu. Výsledkem by pak byla samostatná webová aplikace schopná pomocí webových služeb komunikovat s Jenkins a zásuvný modul, který by inicioval tuto komunikaci.

Po zvážení více kritérií jsem se rozhodl postupovat navrženou variantu číslo jedna. Bude tedy potřeba programovat pouze jeden zásuvný modul bez webové aplikace. Tato varianta tedy bude jednodušší na vývoj, správu i distribuci zdrojových kódů. Pro tvorbu definic podnikových procesů, jejich správu a interakci s uživatelem lze využít externí webové aplikace a služby.

4.3 Návrh zásuvného modulu

Navržený zásuvný modul bude celý integrován do Jenkins a nasazován jako balíček `.hpi`. Balíček .hpi má tuto vnitřní strukturu [21] vygenerovanou při sestavování:

```
plugin.hpi
 +- META-INF
 |    +- MANIFEST.MF
 +- WEB-INF
 |    +- classes
 |    +- lib
 +- (static resources)
```

V návrhu se vycházelo z ukázkového zásuvného modulu, který byl vygenerován nástrojem Maven, konkrétně Jenkins má specializovaný zásuvný modul pro Maven včetně svého vlastního Maven repozitáře. Základem aplikace budou následující Java třídy:

- `CompleteProcessEventListener` – definuje vlastní naslouchání pro události v jBPM, třída je potřebná pro detekci korektního ukončení podnikového procesu,
- `DescriptorImpl` – třída, která se podílí na registraci zásuvného modulu do systému, spravuje globální nastavení zásuvného modulu a zajišťuje validní konfiguraci kroku sestavení v konfiguraci celé Jenkins úlohy,
- `JbpmUrlResourceBuilder` – vychází ze základní třídy vygenerovaného ukázkového zásuvného modulu, je základem celé aplikace, neboť implementuje chování kroku sestavení Jenkins úlohy, tedy spuštění podnikového procesu a odkazuje se na deskriptor, který popisuje konfiguraci tohoto kroku sestavení,
- `JenkinsJobWorkItemHandler` – implementuje obslužnou rutinu, která se provádí, jakmile tok podnikového procesu dojde do pracovní položky, která má za úkol spustit Jenkins úloh, zde je tedy zavolána Jenkins úloha a předán její výsledek dále,
- `Logger` – pomocná třída, která vypisuje na základě nastavení informační zprávy do webové konzole nebo do konzole příkazové řádky,
- `WorkItemCause` – jednoduchá třída pro popis příčiny spuštění Jenkins úlohy.

Nad těmito třídami se budou implementovat formuláře v jazyce Jelly pro konfiguraci kroku sestavení konkrétní Jenkins, dále pro globální konfiguraci. V prvním případě půjde o URL k definici podnikovému procesu a jeho identifikátor. V případě globálního nastavení půjde o příznaky, zda zaznamenávat zprávy zásuvného modulu do webové konzole, případně do konzole příkazové řádky. K položkám těchto formulářů bude napsána nápověda v jazyce HTML, jeden soubor s nápovědou pro každou položku.

Pro návrh podnikových procesů se použije nástroj Designer. Jelikož Designer se obvykle nespouští samostatně, bude používán společně s nástrojem Guvnor pro správu procesových a jiných definic. Guvnor bude použit jako úložiště definic testovacích podnikových procesů, které budou dostupné pomocí rozhraní WebDAV. URL definic podnikových procesů se nastaví v konfiguraci kroku sestavení příslušných Jenkins úloh. V Guvnoru bude také definován nový typ pracovní položky určené pro spouštění Jenkins úloh.

Pro testování funkčnosti budou definovány a popsány dva podnikové procesy. První předvede spuštění jednoduchého podnikového procesu, jenž bude obsahovat pouze volání triviální Jenkins úlohy. Druhý podnikový proces bude spouštět dvě úlohy, přičemž v něm bude přidána ukázka vyhodnocení podmínky. V této podmínce se na základě výsledného stavu první úlohy rozhodne, jestli se proces buď předčasně ukončí nebo se bude pokračovat v toku procesu spuštěním druhé úlohy.

Kapitola 5

Implementace

Tato kapitola popisuje výsledný modul, jak byl implementován včetně popisu všech úskalí a problémů, které bylo nutné pro zdárné dokončení řešit. Základem pro zdárnou implementaci bylo důkladné pochopení fungování zadaných technologií, tedy Jenkins a jBPM. Studium Jenkins z jeho tutoriálů není pro pochopení problematiky dostatečné. Jedná se o rozsáhlý a rychle se rozvíjející projekt, bylo tedy nutné začít co nejdříve se samotným programováním a zkoušet co a jak funguje. S jBPM byl tento postup podobný. Opět jde o rozsáhlý projekt s velkou dokumentací. Samotné čtení dokumentace ale nestačí. Je potřeba zkoušet si jednotlivé vlastnosti, jak fungují. Navíc je potřeba se seznámit se všemi nástroji a službami, které jBPM používá a včas zavrhnout technologie, které nemá cenu používat. Například jako nevyhovující se ukázal plugin pro Eclipse pro návrh podnikových procesů. Lepší je webový nástroj Designer. Ovšem detaily ohledně návrhu testovacích projektů budou probrány v kapitole 6. Po většinu času se pracovalo s verzí jBPM 5.2, přičemž některé komponenty této sady trpěly značnou nevyzrálostí. Až v ke konci práce vyšla verze 5.3, která s sebou přinášela řešení některých nedostatků. Probíhala zde spolupráce s komunitními vývojáři a některé změny se již během řešení práce promítly v hlavní vývojové větvi projektu. Více o problémech spojených s implementací v kapitole 7.

V kapitole nebude popsána jen samotná implementace v jazyce Java, nýbrž i popis navržených formulářů ve skriptovacím jazyce Jelly včetně popisu definic vlastních, uživatelem definovaných, pracovních položek (work items) v jBPM. Pro zdrojový kód aplikace byl využit verzovací nástroj Git a projekt je hostován na portálu Github na adrese `https://github.com/jsvitak/jbpm-workflow-plugin`, kde je zpřístupněn pod licencí GNU General Public License (GPL) verze 3.

5.1 Zásuvný modul

Nejprve bylo potřeba zjistit, jak spouštět vlastní kód v prostředí Jenkins. Tuto situaci nám významně ulehčuje zásuvný modul pro systém Maven. Maven umožňuje sestavení projektů v jazyce Java, přičemž dokáže snadno spravovat závislosti na knihovnách včetně jejich verzí. Vycházelo se tedy ze třídy, která byla vygenerována pomocí Mavenu. Tento jednoduchý ukázkový zásuvný modul dokáže vypisovat v konzoli Jenkins pozdrav a to buď anglicky nebo francouzsky v závislosti na globálním nastavení. Zároveň také demonstruje všechny klíčové vlastnosti a koncepty. Jde například o ukázku tvorby jednoduchého formuláře, předání této hodnoty, její validace ve formuláři, tvorbu globálního nastavení pro daný zásuvný modul apod.

Pro potřeby této diplomové práce stačilo tento zásuvný modul modifikovat tak, aby místo pozdravu nastartoval podnikový proces, jehož definice ve formátu BPMN2 byla stažena ze specifikovaného URL umístění. Dále bylo potřeba načíst z webového formuláře kromě již zmíněného umístění také identifikátor podnikového procesu. Tento identifikátor je nezbytné znát pro samotné spuštění podnikového procesu. Třída pro integraci v prostředí Jenkins se jmenuje `JbpmUrlResourceBuilder`.

5.1.1 JbpmUrlResourceBuilder

Tato třída je automaticky zaregistrována díky tomu, že obsahuje vnitřní třídu s anotací `@Extension`, čímž je definován již dříve vysvětlený bod rozšíření. Díky němu je při nasazení pluginu do Jenkins toto rozšíření správně rozpoznáno a zaregistrováno. Integrace ve formulářích je závislá na typu implementovaného bodu rozšíření. V případě zásuvného modulu této diplomové práce jde o tzv. krok sestavení (build step). Nebylo potřeba vytvářet vlastní typ Jenkins úlohy (vlastní *job*). Tento krok sestavené tedy bude nabízen jako jedna z možností při přidávání kroků sestavení např. ve volném (free-style) projektu. Jinými kroky sestavení mohou být kupříkladu ty základní výchozí, nabízené již po instalaci Jenkins, např. spuštění Shell skriptu, spuštění cílů nástrojů Ant či Maven apod. Pokud byli v zásuvném modulu definovány i nějaké položky globálního nastavení, pak je možné tyto položky vyplnit v globální konfiguraci v sekci určené pro implementovaný zásuvný modul. Při samotném spuštění celé úlohy je spouštěn postupně jeden krok sestavení za druhým. Jakmile je tímto krokem sestavení zásuvný modul pro podnikové procesy jBPM, pak je spuštěna již výše zmíněná přetížená metoda `perform()`. V parametrech metody jsou předány informace o probíhající úloze, jejíž součástí je prováděný krok sestavení. Využit zde bude pouze parametr typu `Listener`, který umožňuje mimo jiné vypisovat textové zprvávy do webové konzole spuštěné úlohy.

Nejprve je provedena počáteční inicializace třídy `Logger`. Tato třída je určena pro průběžné vypisování informace o stavu zásuvného modulu. Výstup je realizován buď do webové konzole spuštěné úlohy a do konzole příkazové řádky. Záleží zde na globálním nastavení proměnných zásuvného modulu. Dále se nastavuje konfigurace třídy `KnowledgeBuilder`. Jde zejména o nastavení vlastního překladače pro Java kód. Důvod pro potlačení výchozího Eclipse Java překladače, který je využíván jBPM a Drools je uveden v sekci 7.6. Následně je vytvořena instance `KnowledgeBuilder`, do které lze přidávat definice obchodních pravidel a podnikových procesů.

Aby bylo možné načítat data z nějakého umístění URL, bylo potřeba provést autentizaci spojení. Je-li předpokládáno využití nástroje Drools Guvnor pro správu definic podnikových procesů, pak je potřeba do souboru `plugin.properties` definovat přihlašovací údaje – jméno a heslo. Pokud tyto údaje definovány nejsou, je použita výchozí kombinace uživatele *admin* s heslem *admin*. Adresa definice podnikového procesu je získána z proměnné třídy `url`. Tato proměnná spolu s proměnnou třídy `processId` je naplněna v době vytvoření třídy v jejím konstruktoru. Jelikož je konstruktor třídy opatřen anotací `@DataBoundConstructor`, pak je nutné, aby jednotlivé parametry konstruktoru svými názvy odpovídaly identifikátorům ve webovém formuláři konfigurace Jenkins úlohy. Informace z konfiguračních formulářů je perzistována a je tedy načtena z vnitřního úložiště objektů Jenkins, kde je stav objektů serializován do XML.

V případě selhání při přidávání a kompilaci definice podnikového procesu je zachycena informace o chybě a vypsána její příčina v textové podobě. Zároveň je krok sestavení okamžitě ukončen jako neúspěšný. Jestliže je však vše v pořádku, pak nic nebrání vytvoření

`KnowledgeBase`, která má za úkol obsahovat všechny v aplikace používané definice podnikových pravidel. Protože vytváření `KnowledgeBase` je poměrně drahá operace, která se ve větším počtu definic pravidel a procesů již projeví, je vhodné používat více sezení se stavem reprezentovaných třídou `StatefulKnowledgeSession`. Pro účely této diplomové práce toto není tak důležité, protože naše aplikace bude vždy v jednom běhu kroku sestavení pracovat s jedním podnikovým procesem.

V sezení reprezentovaném `StatefulKnowledgeSession` je potřeba zaregistrovat rutinu, která bude implementovat chování pracovní položky v podnikovém procesu, která spouští určenou Jenkins úlohu a předá její výsledek dále podnikovému procesu k vyhodnocení. Také se zde registruje rutina pro obsluhu uživatelských úloh (human tasks), která dává uživateli možnost ve svých podnikových procesech používat uživatelské úlohy. jBPM nabízí webového klienta pro interakci se službou uživatelských úloh – jBPM Console. Je také možné definovat globální proměnné, které jsou potenciálně dostupné všem spuštěným instancím procesů.

Definice vytvořené pracovní položky se jmenuje *JenkinsJob* a název třídy pro implementaci její obsluhy je `JenkinsJobWorkItemHandler`. Podrobněji se popisem implementace této třídy zabývá následující podsekce (5.1.2). Popis pracovní položky je možné provést např. v prostředí Eclipse a ukládat ji jako `.wid` soubor. V této práci byl využit editor pro pracovní položky nástroje Drools Guvnor. Definice pak nebyla uložena v souboru, ale v repozitáři této webové aplikace. Další popis definice pracovní položky pro spouštění Jenkins úloh je v podsekci (5.3).

Jako poslední věc v metodě `perform()` je předávání procesových proměnných. První procesovou proměnnou je asociativní pole *jenkinsJobResults*, kde klíčem je unikátní název Jenkins úlohy a hodnotou je její výsledný stav. Pro naprosto správnou funkčnost zásuvného modulu je potřeba myslet při návrhu podnikového procesu na to, aby se spuštění jedné úlohy neopakovalo v procesu víckrát. Kdyby se nějaká úloha spouštěla na více místech paralelně, docházelo by při vyhodnocování stavu pouze k vyhodnocení posledního, tedy nejaktuálnějšího, výsledku. Navíc to samo v praktickém použití příliš nedává smysl spouštět některou úlohu vícekrát. Dochází tak k chaosu při interpretaci výsledků úloh. Další procesovou proměnnou je *jenkinsLastJobResult*, která je zde pouze pro zjednodušení práce a obsahuje výsledek Jenkins úlohy, která skončila naposledy. Na konci metody je tedy spuštěno provádění podnikového procesu a řízení dalšího toku je předáno jBPM. Protože v podnikovém procesu může běžet více vláken, čeká se na dokončení celého podnikového procesu a to se pozná detekcí přechodu toku procesu do ukončovací události.

5.1.2 JenkinsJobWorkItemHandler

Pro spouštění jednotlivých úloh z podnikového procesu je nutné nejen správně definovat celý podnikový proces, ale je potřeba i správně používat obslužné úlohy (service tasks). V této diplomové práci byl implementován vlastní typ obslužné úlohy a to pracovní položku *JenkinsJob*, viz (5.3). Je-li hotova její definice, pak je možné tuto pracovní položku použít v libovolném podnikovém procesu. V této podsekci bude popsáno fungování obsluhy pro pracovní položku, tedy co se stane, když tok v podnikovém procesu dorazí na místo tohoto typu obslužené úlohy.

Třída `JenkinsJobWorkItemHandler` jako rutina pro obslužnou úlohu musí implementovat rozhraní `WorkItemHandler`. V tomto rozhraní jsou dvě metody. Metoda implementující vlastní chování obslužné rutiny se nazývá `executeWorkItem`. Jako parametry jsou předány objekt s reprezentací pracovní položky `workItem` a objekt správce všech obslužných ru-

tin `workItemManager`. Prvním úkolem je vyextrahovat z předaných vstupních parametrů proměnné, s kterými je možné pracovat. Klíčovou proměnnou, kterou je nutné zjistit, je `jenkinsJobName`, tedy název Jenkins úlohy ke spuštění. Druhou proměnnou je kolekce výsledků všech obslužných úloh v celém podnikovém procesu.

Jakmile je znám název Jenkins úlohy, pak se podle něj vyhledá odpovídající projekt v aktuální instanci Jenkins. V kódu je místo třídy `Jenkins` ještě použita třída `Hudson` (včetně balíčků) z důvodů zpětné kompatibility. Vyhledaná úloha je obecně typu `AbstractProject`. Jenkins má složitou hierarchii tříd a úroveň generických typů dosahuje poměrně velké složitosti. Je-li tedy nalezen objekt typu `AbstractProject`, pak je naplánováno jeho spuštění metodou `scheduleBuild`. Je však potřeba specifikovat vlastní typ důvodu pro spuštění úlohy. Zde je tento důvod reprezentován třídou `WorkItemCause`, která je poměrně malá obsahuje pouze textový popis příčiny spuštění úlohy. Naplánováním úlohy je získán `Future` objekt. Úloha je spuštěna, má li instance Jenkins k dispozici systémové zdroje ke spuštění. Na lokálním počítači toto není potřeba řešit, ale při běhu mnoha úloh na mnoha vzdálených strojích se úlohy řadí do fronty. V synchronizovaném bloku se počká na dokončení výpočtu `Future` objektu a předá se výsledek doběhnuvší úlohy. Tento výsledek je přidán do kolekce výsledků všech úloh a do proměnné pro aktuálně poslední výsledek. Tyto proměnné jsou pak předány jako výstupní parametry pracovní položky a správci rutin obslužných úloh `workItemManager` je oznámeno dokončení výpočtu.

Metoda `abortWorkItem()` nebyla implementována. Je určena pro kód, který bude řešit náhlé vynucené ukončení probíhající obslužné rutiny a je na uživateli, aby tuto metodu naprogramoval dle svých potřeb.

5.1.3 DescriptorImpl

`DescriptorImpl` je třída vnořená do třídy `JbpmUrlResourceBuilder`, jež implementuje bod rozšíření (extension point) a to díky anotaci `@Extension`. Jsou v ní zejména persistentní proměnné pro globální nastavení zásuvného modulu a metody pro jejich načítání a ukládání. Dále pak validátory pro proměnné pro popis konkrétního podnikového procesu, které se pro každou Jenkins úlohu liší – `url` a `processId`. Je zde také popisek kroku sestavení, pod kterým je nabídnut uživateli při konfiguraci Jenkins úlohy. Tato třída je nedílnou součástí třídy `JbpmUrlResourceBuilder`.

5.1.4 CompleteProcessEventListener

Třída pro naslouchání událostí v jBPM, konkrétně naslouchá pro událost ukončení procesu. Vzhledem k principu fungování jBPM je toto korektní způsob, jak hlavní vlákno počká na dokončení toku v podnikovém procesu, kde mohou ještě běžet jiná vlákna.

5.1.5 Logger

`Logger` je pomocná třída pro zaznamenávání zpráv o stavu zásuvného modulu. Zapisuje do webové konzole i do konzole příkazové řádky na základě globální konfigurace pluginu. Díky prefixu `JBPM:` jsou její zprávy lépe rozlišitelné.

5.1.6 WorkItemCause

Tato triviální metoda pouze definuje textový popis příčiny spuštění Jenkins úlohy. Její použití je ale nezbytné, je použita pro volání metody `scheduleBuild` třídy `AbstractProject`,

která naplánuje spuštění Jenkins úlohy nalezené podle názvu.

5.2 Interakce s uživatelem

Jak již bylo uvedeno v sekci 2.2 Jenkins používá pro tvorbu formulářů skriptovací jazyk Jelly. Jedná se o jazyk poněkud nestandardní a jako projekt již málo používaný, ale pro jednoduché políčka ve formulářích bylo jeho použití plně dostačující. Stačilo napsat pouze tři soubory `.jelly`:

- `config.jelly` – políčka pro vyplnění URL umístění definice BPMN 2.0 podnikového procesu a identifikátoru tohoto procesu,
- `global.jelly` – políčka pro globální konfiguraci, je zde možnost zakázat zaznamenávání do webové konzole nebo do konzole příkazové řádky,
- `index.jelly` – obsahuje pouze text s popiskem pro obrazovku se seznamem instalovaných zásuvných modulů.

Mimo psaní formulářů bylo potřeba napsat i vnitřní programovou nápovědu s popisem jednotlivých políček. Tato nápověda byla ve formátu `.html` a název souboru bylo nutné volit obezřetně a pojmenovat tak, aby tuto nápovědu dokázal Jenkins najít:

- `help-processId.html`
- `help-url.html`
- `help-globalConfig.html`

Z názvu souborů je patrné, jakou nápovědu pro které položky obsahují.

5.3 Definice pracovní položky pro spuštění Jenkins úloh

Aby bylo možné v podnikovém procesu spouštět Jenkins úlohy, bylo nutné pro tyto účely definovat vlastní pracovní položku (work item), v terminologii jBPM označovanou jako obslužná úloha (service task). V Guvnoru je potřeba definovat tyto uživatelské pracovní položky pro každý balíček (package) zvlášť. V balíčku jsou definice pracovních položek k nalezení pod záložkou *WorkItemDefinition*. Problémem zde byl pouhý textový editor bez nápovědy při psaní, či bez validátoru. K dispozici byl jen nástroj pro generování jistých dílčích konstrukcí, jakými jsou vstupní parametry, výstupní parametry, importy závislostí a případně adresy na ikony. Nešlo zde předejít uložení vadné definice podnikové položky, což pak mohlo způsobovat problémy během návrhu a zbytečně tyto nepříjemnosti zpomalovaly práci. Tuto problematiku ještě rozebírá sekce 7.3. Tvorbu pracovních položek nabízí i jBPM plugin pro Eclipse, ale práce s ním byla shledána jako nevyhovující. Jedním z důvodů byla velmi špatná uživatelská přívětivost, neexistovaly zde ani jednoduché generátory dílčích kusů kódu jako v Designeru a jBPM Eclipse návrhář podnikových procesů není tak vyspělý jako Designer. Dále definice pracovních položek `.wid` a podnikových procesů `.bpmn2` nešlo snadno distribuovat běžící webové aplikaci Jenkins, neboť jakožto soubory byly uloženy na disku někde ve struktuře Eclipse projektu. Následující kus zdrojového kódu ukazuje, jak vypadá definice pracovní položky pro spouštění Jenkins úloh vytvořená v této diplomové práci, viz 5.1. Vlastnost *name* je v systému důležitá, protože podle tohoto názvu je regis-

```
import org.drools.process.core.datatype.impl.type.StringDataType;
import org.drools.process.core.datatype.impl.type.ObjectDataType;
[
  [
    "name" : "JenkinsJob",
    "displayName" : "Jenkins Job",
    "parameters" : [
      "jenkinsJobNameInput" : new StringDataType(),
      "jenkinsJobResultsInput" : new ObjectDataType(),
    ],
    "results" : [
      "jenkinsJobResultsOutput" : new ObjectDataType(),
      "jenkinsLastJobResultOutput" : new ObjectDataType(),
    ],
    "icon" : "http://localhost:8080/drools-guvnor/rest/packages/
      org.jenkinsci.plugins.jbpm/assets/JenkinsJobIcon/binary",
  ]
]
```

Obrázek 5.1: Definice pracovní položky *JenkinsJob* pro spouštění Jenkins úloh implementována v této práci

trován příslušná oblužná rutina (handler). Výčty parametrů a výsledků popisují vstupní respektive výstupní argumenty pracovní položky. Tyto argumenty se v Designeru mapují na procesové proměnné, nebo je do nich přímo přiřazena hodnota, například pro argument *jenkinsJobNameInput* se přímo přiřazuje řetězec s názvem úlohy, jež se má vyhledat a spustit. Ikona *icon* má již pouze dekorativní charakter, ale je užitečná pro snadnější rozlišení úloh při návrhu procesu. V této práci je pro pracovní položku použita přímo ikona z loga projektu Jenkins. Ikona se nahrává do Guvnoru pod záložkou *Other assets, documentation.*

Kapitola 6

Testovací podnikové procesy

Pro testování funkčnosti a nakonec i pro výsledné předvedení vlastností implementované aplikace bylo vytvořeno několik podnikových procesů. Tyto podnikové procesy modelují spuštění ať už jedné testovací Jenkins úlohy anebo spuštění úlohy s jednoduchým rozhodováním založeném na ruční definici podmínek. Vybrané procesy budou rozebrány v následujících sekcích, včetně podnikového procesu, který byl požadován v zadání, tedy spuštění reálné základní a regresní testovací sady.

6.1 Jednoduchý testovací proces

Počáteční testování probíhalo na jednoduchém podnikovém procesu obsahujícím pouze jednu Jenkins úlohu *hello-job*. Úloha *hello-job* měla jen jeden krok sestavení, který spouštěl Shell skript, jenž vypisoval do konzole jednoduchý pozdrav. V obrazovce této úlohy byla i její historie spuštění. Podle času a zvyšujícího se pořadového čísla bylo možné poznat, jestli a kdy byla úloha spuštěna. Při nahlédnutí do konzole úlohy bylo možné na prvním řádku vidět, kdo tuto úlohu spustil, jestli anonymní či registrovaný uživatel Jenkins anebo to byl jBPM workflow plugin. Dále se pak v konzoli nalézala informace o spuštění skriptu a samotný vypsaný pozdrav.

Naopak úloha, která spouštěla podnikový proces, se jmenovala *test-hello*. Tato úloha byla stejně jako předešlá typu *free-style software project* a měla jen jeden krok sestavení (build step), jímž byl zásuvný modul implementovaný v této práci. Bylo potřeba určit URL k `.bpmn2` definici podnikového procesu a také jeho procesový identifikátor (process ID). Na obrázku 6.1 lze vidět, jak vypadá tento podnikový proces ve vzhledu a barvách s jakými pracuje aktuální Designer ve verzi 2.2.

Obrázek 6.1: Jednoduchý testovací podnikový proces spouštějící triviální *hello-job*

Při návrhu podnikového procesu *hello* se nastavovaly v konfiguraci dvě procesové proměnné, konkrétně *jenkinsJobResults*, což je asociativní pole s aktuálními výsledky všech

úloh v podnikovém procesu, a *jenkinsLastJobResult*, jež představuje aktuální výsledek naposled ukončené Jenkins úlohy. V konfiguraci procesu lze nastavit ještě import pro datovou třídu `Result`, které slouží pro reprezentaci výsledku Jenkins úlohy. Tento import se využije v místech, kde se krátkými úseky Java kódu programuje nějaká podmínka či omezení.

Ve vlastnostech samotné pracovní položky se specifikovaly vstupní a výstupní parametry a jejich datové typy. Pro tuto činnost však čistá instalace jBPM 5.2 nabízí naprosto nedostatečné prostředky. Lze použít jen textový editor, ale přesnou syntax zápisu nelze nikde dohledat, což činí takový editor nepoužitelným. Než bylo těsně před dokončením práce vydáno jBPM 5.3, bylo nutné použít alespoň aktualizovanou verzi nástroje Designer. Více o tomto problému viz sekce 7.5.

Po definici procesových proměnných spolu se vstupními a výstupními parametry se pokračovalo přiřazováním dat mezi sebou, tedy mapováním procesových proměnných na vstupní parametry pracovních položek a naopak, případně přiřazováním konstant. Jelikož v převážné době řešení práce se používalo jBPM 5.2, které trpělo problémy spojenými s mapováním procesových proměnných, bylo potřeba dodržovat jisté konvence v názvech, aby nedocházelo ke kolizím a následným ztrátám definic procesů. Více o tomto problému pojednává sekce 7.4.

Závěrem byl vyplněn název *Name* pracovní položky *JenkinsJob*. Tento název je zobrazen v diagramu na položce, úloha z této kapitoly se jmenuje *hello-job*. Test proběhl zdárně, jak se lze přesvědčit na obrázku 6.2. Na obrázku je vidět, jak ve výsledku vypadá Jenkins spolu s úspěšně dokončenou úlohou, která má jako jeden krok sestavení spuštění zásuvného modulu pro jBPM. V levé části obrázku se nalézá nabídka pro manipulaci s během úlohy *test-hello* s pořadovým číslem 35. Pravá část obsahuje výstup webové konzole, kde zprávy s prefixem `JBPM:` generuje zásuvný modul pro jBPM.

Obrázek 6.2: Ukázka Jenkins spolu s jBPM zásuvným modulem

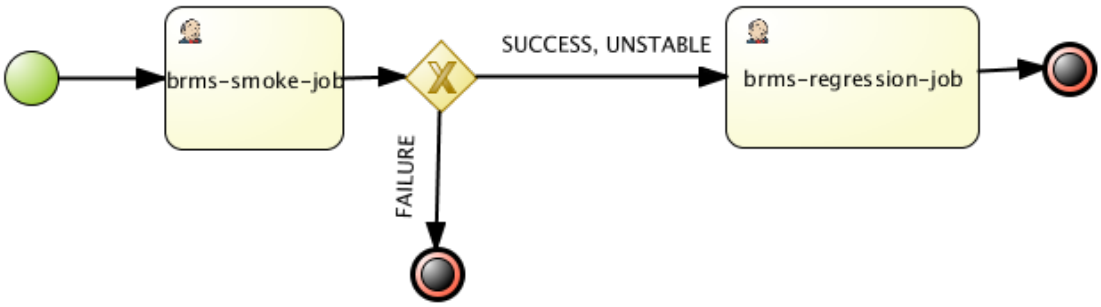
6.2 Proces spouštějící reálný projekt

Dle požadavků zadání bylo potřeba vykonat testovací plán pro produkt Business Rules Management System (BRMS) 5.2, kde tento testovací plán obsahoval základní a regresní testovací sadu. Podnikový proces spouštějící dvě úlohy bylo možno demonstrovat na jakýchkoli úlohách, na demonstraci funkčnosti zásuvného modulu by to nemělo vliv. Ačkoliv

tedy na spouštěných úlohách nezáleží (mohlo se jednat i o úlohy typu *hello-job* z předcházejícího příkladu), bylo potřeba nakonfigurovat pro splnění zadání reálné Jenkins úlohy pro testování BRMS.

Prvním krokem tedy bylo správně zprovoznit tyto úlohy ručním přenesením jejich konfigurace z firemní instance Jenkins na lokální testovací instanci. Do lokální instance Jenkins, spouštěné pomocí nástroje Maven, bylo nejprve nutné doinstalovat dodatečné zásuvné moduly – Multiple SCMs plugin, Git plugin a Groovy plugin. Jak již názvy pluginů napovídají, používalo se více verzovacích systémů v jedné úloze a to Subversion pro stažení konfiguračních Groovy skriptů a Git pro stažení zdrojových kódů interní testovací sady QA týmu. Binární kód produktu BRMS 5.2 se stahoval z úložiště umístěném na firemním Jenkins serveru. Dále se specifikovaly možnosti nastavení úlohy, které se pak vyplňují ve formuláři před spuštěním úlohy. Zásuvný modul spouští Jenkins úlohy přes API, žádný formulář se nevyplňuje, jsou tedy použity výchozí možnosti z nabídek. Ve formuláři při startu úlohy se vybírala verze produktu – *BRMS_VERSION*. Dále pak verze testovací sady *TEST_BRANCH*, skupiny testů, které se mají pustit (*includes*) a naopak skupiny testů, které se mají vynechat (*excludes*). Samotná úloha má pouze jeden krok sestavení, který spustí Groovy skript `bre-regression.groovy` pro tuto úlohu. Groovy skript tedy připravuje prostředí pro spuštění úlohy, v našem případě hlavně nastavení vlastností (properties) a parametrů pro Maven, jímž je spouštěna QA testovací sada.

Byly vytvořeny dvě testovací úlohy *brms-smoke-test* a *brms-regression-test*, které spouštějí testovací sadu týkající se hlavně funkcionality engine BRMS 5.2. Jiné testy však mohou vyžadovat spuštění webového kontejneru pro testování webových aplikací nebo pro testování integrace více služeb, případně mohou vyžadovat jinou, složitější, konfiguraci. Testování reálných testovacích sad trvalo déle, protože se stahovaly poměrně velké balíky dat přes síť. V lokální instanci Jenkins byly tyto úlohy úspěšně otestovány samostatně, později jejich spuštěním společně v podnikovém procesu, viz obrázek 6.3.



Obrázek 6.3: Testovací podnikový proces spouštějící reálné úlohy – *brms-smoke-job* a *brms-regression-job*

Úloha *brms-smoke-job* testovala základní funkce jako načítání základních typů zdrojových souborů, vybranou minimální funkcionalitu, byla podmnožinou druhé úlohy *brms-regression-job*. Tato úloha pokrývala již všechny testy engine zaměřené na jeho funkcionalitu a na opravy hlášených chyb. Podnikový proces znázorněný na obrázku 6.3 se liší od testovacího procesu z předchozí sekce přidáním práce s procesovými proměnnými a s podmínkami. Každé pracovní položce se nastavily názvy jejich odpovídajících Jenkins úloh. Na vstupu i na výstupu se namapovaly obě proměnné pro zpracování výsledků Jenkins úloh – *jenkinsJobResults* a *jenkinsLastJobResult*. Větvící podmínky se naprogramovaly v jazyce Java.

Jenkins definuje několik druhů výsledků:

- SUCCESS – úloha proběhla bez chyb,
- UNSTABLE – úloha měla nějaké chyby, ale nebyly zásadní, například selhaly některé testy,
- FAILURE – úloha selhala,
- NOT_BUILD – úlohu se nepodařilo sestavit, např. některá fáze úlohy selhala,
- ABORTED – úloha byla manuálně ukončena.

Podnikový proces spustil *brms-smoke-job* a v případě, že jeho stav výsledku byl SUCCESS nebo UNSTABLE, pak pokračoval spuštěním *brms-regression-job*. Jinak pro stav FAILURE a horší došlo k ukončení procesu. Reálným testem prošla základní úloha s výsledkem SUCCESS a regresní úloha s výsledkem UNSTABLE, protože několik testů z této testovací sady selhalo. Výsledná podmínka pro první popsanou větev vypadala následovně:

```
return ((Result)(kcontext.getVariable("jenkinsLastJobResult")))
        .isBetterThan(Result.FAILURE);
```

Testování požadovaného reálného podnikového procesu dle zadání dopadlo úspěšně. Podařilo se spustit podnikový proces, který byl schopen volat úlohy nástroje Jenkins. Samotná vnitřní činnost Jenkins úloh již není důležitá, lze je považovat za tzv. černou skříňku. Důležitá je možnost jejich spuštění a zjištění jejich výsledného stavu, což se podařilo demonstrovat.

Kapitola 7

Problémy

Při samotné implementaci a testování vznikala spousta problémů. Někdy se je podařilo poměrně snadno vyřešit, jindy bylo potřeba problém nahlásit vývojářům, zejména jBPM, a urgovat vyřešení problému. Cílem této kapitoly je ukázat, že vývoj open-source projektů je velmi bouřlivý a někdy trochu hůře předvídatelný. Často se stává, že nějaká součást projektu se přestane udržovat a bývá zdrojem problémů v budoucnu. Na programátora jsou kladeny vyšší požadavky, zejména na jeho flexibilitu a schopnost rychle řešit nové problematické situace. Jde o hledání řešení na různých fórech, protože dokumentace často neodráží aktuální realitu. Na tyto odpovědi se nedá spolehnout, je potřeba zkoušet různá řešení. Člověk je také často nucen komunikovat s vývojáři projektů, které používá ve svém vlastním projektu. Tato komunikace zahrnuje kladení dotazů na IRC kanálech, hlášení problémů do systému pro hlášení chyb, jako je například Bugzilla. Lze narazit na problém, který programátorovi může znemožnit pokračování ve svém projektu. Stává se tak, že řešitel je v dokončení své práce závislý na práci a čase jiných lidí.

7.1 Problémy s překladem

Pro překlad projektu byl ze začátku používán Jenkins plugin pro Maven. Překlad se tedy spouštěl pomocí příkazu `mvn hpi:hpi`, který měl vytvořit zkompilovaný *.hpi* soubor. Bohužel časem byl tento překlad nespolehlivý a bylo odhaleno, že tento zásuvný modul pro Maven není nějakou dobu udržovaný a kvalitní. Náhodně nefungující překlady se tedy podařilo vyřešit pomocí překladu standardním způsobem `mvn clean package -DskipTests`, jenž také sestavuje výsledný `.hpi` soubor, ale standardním způsobem. Definice systémové proměnné `skipTests` je zde nutná, neboť spouštění automatických testů také nefungovalo a končilo chybami. Tyto automatické testy vygenerované spolu ze základním pluginem lze tedy považovat za další věc nedotaženou do konce. Jak je vidět problémů je zde hodně, je to způsobeno slabou aktualizací některých návodů pro tvorbu zásuvných modulů a také některých součástí nástroje Jenkins. Jenkins Maven plugin byl využíván již jen pro testování. Testování probíhalo pomocí příkazu `mvn hpi:run -Djetty.port=8090`. Tento Maven příkaz spustí instanci Jenkins ve webovém kontejneru Winstone na portu 8090. Verze Jenkins je specifikována v konfiguračním souboru `pom.xml`. Maven zajistí, že v případě potřeby je potřebná verze stažena z internetu a spuštěna. Mimo jiné je důležité určit si i port, na kterém bude instance Jenkins naslouchat, protože na portu 8080 byla již často spuštěná instance aplikačního serveru JBoss. Tento aplikační server je dodáván s balíkem jBPM a je využíván jako kontejner pro webové aplikace, jako je Drools Guvnor pro správu definic pod-

nikových procesů, včetně webového nástroje Designer pro samotný návrh a úpravu definic podnikových procesů.

7.2 Náhlá změna repozitáře Mavenu

Implementovaný zásuvný modul používá pro sestavení nástroj Maven. Soubor `pom.xml` je konfigurační soubor projektu pro Maven a jsou v něm specifikovány cíle překladu, závislosti na knihovnách a repozitáře, ze kterých je možné požadované knihovny stáhnout. Jednoho dne projekt náhle nešel přeložit, závislosti se nemohli stahovat. Toto bylo způsobeno jedním z mnoha výpadků repozitáře aplikačního serveru Glassfish firmy Oracle `http://maven.glassfish.org/content/groups/public/`, který byl používán z historických důvodů, neboť původní Hudson spadal pod firmu Oracle. Po zjištění problému ve vývojářské diskuzi bylo potřeba přejít na nový vlastní nezávislý repozitář projektu Jenkins `http://repo.jenkins-ci.org/public/`, který se rozhodli vývojáři Jenkins založit po přetrvávajících problémech s původním repozitářem.

7.3 Tvorba vlastních definic pracovních položek

Pro tvorbu vlastních definic pracovních položek (work item definitions) obsahuje Guvnor specializovaný editor. Chceme-li novou vlastní definici, začínáme v editoru s již výchozí ukázkovou definicí *MyTask*, viz obrázek 7.1. Tato definice toho obsahuje hodně na to,

Obrázek 7.1: Výchozí definice pracovní položky *MyTask* včetně editoru

abychom podle ní byli schopni napsat vlastní definici. Bohužel toto není plně dostačující řešení, chyběla zde komplexnější nápověda. V levém panelu máme nabídku základních prvků, na které když klikneme, dojde k vygenerování jejich kódu na místo kurzoru v editoru pracovní položky. Takové řešení je ale nedostačující, jestliže vyplníme něco špatně, tak se nám naše nová položka nenabídne v paletě Designeru a chybu musíme hledat vlastními silami. Nahlásil jsem tedy požadavek na vylepšení tohoto editoru do systému Bu-

gzilla https://bugzilla.redhat.com/show_bug.cgi?id=807693. Systém Bugzilla firmy Red Hat je využíván primárně pro její produkty, ale zde se oprava chyby promítne i do zdrojových kódů jBPM 5, takže lze očekávat, že ve verzi 5.3, jež byla vydána krátce před dokončením práce, je již tento problém vyřešen.

7.4 Mapování procesových proměnných na parametry obslužných úloh

Při používání webového nástroje Designer, jenž je součástí projektu jBPM, často docházelo k problémům při práci s procesovými proměnnými, což bylo způsobeno nevyzrálostí celého projektu. Tento problém se týkal mapování proměnných procesu na vstupní a výstupní parametry obslužných úloh (service task). Pro spouštění Jenkins úlohy bylo potřeba předat název spouštěné úlohy a také kolekci výsledků jednotlivých úloh. Na výstupu byla tato kolekce aktualizována o výsledek aktuální úlohy. Na výstupu je také předáván poslední aktuální výsledek, pokud nám stačí jen tento poslední výsledek. S ním si například vystačíme u jednodušších podnikových procesů, kde nespouštíme požadované Jenkins úlohy paralelně najednou. Bohužel je potřeba dodržovat konvenci názvů vstupních a výstupních parametrů a procesních proměnných. Je potřeba odlišit jejich názvy, například přidáním `Input` na konec názvu vstupních parametrů a přidáním `Output` na konec názvu výstupních parametrů. Pokud se vstupní anebo výstupní parametry jmenují stejně jako na ně mapovaná procesní proměnná, pak při uložení podnikového procesu dojde k chybě a ztratíme celou definici podnikového procesu. Tento neduh je velmi obtěžující pro uživatele a brání efektivní práci. Jediným řešením je tedy si dávat pozor a dodržovat navrženou konvenci. Tento problém tedy byl nahlášen na adrese https://bugzilla.redhat.com/show_bug.cgi?id=819884 a v příští verzi jBPM by měl být již opraven.

7.5 Editory v jBPM Designeru

V základní verzi Designeru pro návrh procesů balíku jBPM 5.2 byla naprosto nedostatečná podpora webových editorů pro specifikaci proměnných procesu, vstupních a výstupních parametrů úloh a také pro jejich mapování. Byl k dispozici pouze obyčejný textový editor. Bez jakékoli nápovědy syntaxe bylo pro nezasvěceného člověka téměř nemožné napsat tyto definice správně, neboť nevěděl jak. Protože během tvorby práce byla hlavní verzí k dispozici jBPM 5.2 a nepřibližovalo se tehdy vydání další verze, bylo nutné řešit tento problém alespoň vylepšením Designeru. Po konzultaci s hlavním vývojářem Designeru mi bylo doporučeno používat jeho vývojovou, průběžně vydávanou, verzi ze stránek http://people.redhat.com/tsurdilo/designer/master/. Tento postup jistě není ukázkový, ale byl nutný. Po nasazení .war archivu Designeru z uvedených stránek do lokální instalace jBPM se podařilo tuto nepříjemnost eliminovat.

V závěru řešení práce došlo k vydání nové verze jBPM 5.3, na kterou byl celý projekt migrován. Mimo opravy mnoha jiných chyb byl vyřešen i tento problém s Designerem.

7.6 Konflikt Java překladačů na classpath

Spouštění prvních testovacích úloh narazilo na problémy, protože jakmile aplikace zvládala spouštět úlohy na běžící instanci Jenkins, bylo potřeba výsledky běhu těchto úloh zpracovat a vyhodnotit. Kromě výše zmíněných problémů s mapováním procesových proměnných na

parametry úloh byl problém se spouštěním Java kódu ve skriptovaných úlohách a hlavně v podmínkách větvících bran. Bez nich nebylo možné efektivně provádět rozhodování v podnikovém procesu. Mvel dialekt ovšem fungoval, ale ne všude, jen ve skriptovaných úlohách. V podmínkách větvících konstrukcí nikoliv, jelikož v nich nebyl dialekt Mvel na výběr, podmínky bylo možné definovat pouze v jazyce Java či jako Drools Expert pravidla. Naštěstí se problém podařilo vyřešit. Překladač Javy, který používá Drools Expert (a na kterém je závislé i jBPM) kolidoval s překladačem Javy, který používal webový kontejner Winstone, kde běžela instance Jenkins. Jednalo se překladač Eclipse Java Compiler. Postupů, jak problém řešit, bylo více, například si izolovat classloading v nastavení aplikačního serveru. Bylo ale zvoleno jednodušší řešení, kterým byla definice vlastního překladače, který má Drools KnowledgeBuilder používat. Tímto vybraným překladačem je sice jednodušší, ale pro naše potřeby plně dostačující překladač Janino [4]. Bylo také pro něj potřeba dodefinovat závislost v projektovém konfiguračním Maven souboru `pom.xml`.

Kapitola 8

Závěr

Cílem této práce bylo nastudovat technologie Jenkins a jBPM a hlavně implementovat zásuvný modul, jenž umožní integraci těchto dvou technologií, tedy možnost řízení úloh nástroje pro kontinuální integraci Jenkins pomocí podnikových procesů. Taktéž bylo cílem dostatečné otestování výsledného řešení a to pomocí podnikového procesu, který pracuje s testovacími úlohami z reálné firemní praxe.

Tyto cíle zadání se podařilo splnit. Seznámil jsem se s nástrojem Jenkins pro kontinuální integraci a to nejen jako uživatel, ale i s jeho kódem a filozofií. Podařilo se mi získávat cenné informace od široké komunity vývojářů tohoto nástroje. Dále jsem se podrobně seznámil s nástrojem pro podnikové procesy jBPM. A to nejen s fungováním samotného engine, nýbrž i s celou řadou doprovodných nástrojů, jako jsou Drools Guvnor, Designer, jBPM Eclipse plugin a další. Samotné jBPM se ukázalo jako poměrně nevyzrálý projekt s nepříjemnými nedostatky, ale pro potřeby této práce nakonec jako dostačující. Nejprve jsem pracoval s jBPM 5.2, kde byly tyto problémy znát nejvíce, později ke konci řešení práce byla k dispozici o něco vyzrálejší verze jBPM 5.3.

Navrhl jsem možné varianty řešení výsledné aplikace a zvážil jejich nedostatky. Vybral jsem jednu z variant, která se po zvážení pro a proti ukázala jako nejschůdnější a implementoval jsem ji. Při implementaci jsem se musel potýkat s různými problémy a situacemi, které mě při práci potkávaly. S některými jsem se po dlouhém snažení dokázal vypořádat sám hledáním řešení na internetu, s jinými jsem musel žádat o pomoc zkušenější vývojáře. Výsledná aplikace je nabídnuta ostatním ke stažení na serveru Github pod licencí GNU GPL verze 3. Každý zájemce tedy může svobodně aplikaci využívat anebo přispět svým kódem ke zlepšení aplikace.

Zapojil jsem se aktivně do spolupráce s komunitou, ať již aktivním dotazováním na diskuzních kanálech anebo hlášením chyb a spoluprací při jejich opravování. Některé chyby se stihly v projektu jBPM opravit, již během řešení této práce, jiným chybám bylo potřeba se vyhnout, pokud to bylo možné.

Aplikaci jsem otestoval na několika triviálních a několika reálných testovacích úlohách. I při testování jsem musel řešit a překonávat problémy spojené zejména se správnou konfigurací všech částí. Při návrhu testovacích procesů jsem se potýkal s problémy zejména s nástroji z balíku jBPM a popisu řešení vybraných problémů jsem věnoval celou kapitolu.

Zásuvný modul má ještě prostor pro další vylepšení. Vzhledem k tomu, že ve firmě Red Hat je velký zájem na dokončení této práce, budu v rámci svého zaměstnání dále pracovat na tomto projektu a udržovat jej do budoucna. Počítá se s ním v oddělením JBoss QA firmy Red Hat, měl by zefektivnit práci Quality Assurance inženýrů. Ovšem očekávám, že najde využití i v jiných firmách a organizacích, které používají Jenkins. Momentálně

je zde několik konkrétních oblastí, které by mohly být vylepšeny. Jednou z prvních je vytvoření testovacích procesů s uživatelskými úlohami, které zapojí do podnikového procesu rozhodování uživatelů. Např. když selžou některé testy testovací úlohy, tak je QA inženýr dotázán, zda-li se má pokračovat dále v toku podnikového procesu anebo mají být nejprve prohlédnuty neúspěšné testy a po jejich případné opravě testy spuštěny znovu. Dalším bodem, který stojí za pozornost, je přidání podpory persistence stavu instancí podnikových procesů, pracovních položek a celých sezení. V případě výpadku serveru tak nedojde ke ztrátě dat aktuálního procesu, ale je možno stav procesu obnovit z databáze. Poslední věcí, která nebyla dostatečně otestována je spuštění více úloh najednou. Například spustíme základní testovací sadu a při jejím úspěchu rozvětvíme tok procesu na regresní testovací sadu, integrační testovací sadu a testující sadu pro webovou aplikaci, které jsou spuštěny paralelně.

Hlavní cíle práce se tedy podařilo splnit a největším přínosem je zásuvný modul dostupný komunitě uživatelů nástroje Jenkins, u kterého je velmi vysoká pravděpodobnost, že se bude v praxi používat a že bude dále udržován a rozvíjen. Dalším významným přínosem práce je přispění do rozvoje komunitních open source projektů, zejména jBPM.

Literatura

[1] MVEL [online]. http://mvel.codehaus.org/, 2006 [cit. 2012-05-21].

[2] Drools Plugin [online]. https://wiki.jenkins-ci.org/display/JENKINS/Drools+Plugin, 2009 [cit. 2012-05-21].

[3] Jelly: Executable XML [online]. http://commons.apache.org/jelly/, 2010-01-05 [cit. 2011-12-30].

[4] Janino [online]. http://docs.codehaus.org/display/JANINO/Home, 2010-08-19 [cit. 2012-05-21].

[5] What is Stapler? [online]. http://stapler.java.net/what-is.html, 2011 [cit. 2011-12-30].

[6] Chapter 5. Core engine: Basics [online]. http://docs.jboss.org/jbpm/v5.2/userguide/ch05.html, 2011 [cit. 2012-01-05].

[7] Oracle Submits Proposal to Make Hudson an Eclipse Foundation Project [online]. http://www.oracle.com/us/corporate/press/393483, 2011 [cit. 2012-05-21].

[8] Activiti [online]. http://activiti.org/, 2012 [cit. 2012-05-21].

[9] Apache Tomcat [online]. http://tomcat.apache.org/, 2012 [cit. 2012-05-21].

[10] Bonita Open Solution [online]. http://www.bonitasoft.com/products/bonita-open-solution-open-source-bpm, 2012 [cit. 2012-05-21].

[11] Business Process Manager Standard [online]. http://www-01.ibm.com/software/integration/business-process-manager/standard/, 2012 [cit. 2012-05-21].

[12] CruiseControl [online]. http://cruisecontrol.sourceforge.net/, 2012 [cit. 2012-05-21].

[13] Hudson Continuous Integration [online]. http://www.eclipse.org/hudson/, 2012 [cit. 2012-05-21].

[14] Intalio BPMS [online]. http://bpms.intalio.com/, 2012 [cit. 2012-05-21].

[15] JBoss Application Server [online]. http://www.jboss.org/jbossas, 2012 [cit. 2012-05-21].

[16] Winstone Servlet Container [online]. http://winstone.sourceforge.net/, 2012 [cit. 2012-05-21].

[17] XStream [online]. http://xstream.codehaus.org/, 2012 [cit. 2012-05-21].

[18] EISCHMANN, J.: Linux na světových burzách. *Open source a praxe*, ročník 1, duben 2011, ISSN 1804-8560.

[19] HUYBRECHTS, T.: JBPM Plugin [online]. https://wiki.jenkins-ci.org/display/JENKINS/JBPM+Plugin, 2008 [cit. 2011-12-31].

[20] JHA, P.: Taking Continuous Integration to a Greater Height Using Hudson [online]. http://jboss-qa.blogspot.com/2007/10/taking-continuous-integration-to.html, 2007-10-15 [cit. 2012-01-01].

[21] KAWAGUCHI, K.: Plugin structure [online]. https://wiki.jenkins-ci.org/display/JENKINS/Plugin+structure, 2008-11-19 [cit. 2012-01-03].

[22] KAWAGUCHI, K.: Architecture - Jenkins [online]. https://wiki.jenkins-ci.org/display/JENKINS/Architecture, 2011-06-07 [cit. 2011-12-30].

[23] KAWAGUCHI, K.: Plugin tutorial [online]. https://wiki.jenkins-ci.org/display/JENKINS/Plugin+tutorial, 2011 [cit. 2011-12-31].

[24] SILVER, B.: *BPMN Method and Style*. Cody-Cassidy Press, druhé vydání, 2011, ISBN 978-0-9823681-1-4.

[25] SMART, J. F.: *Jenkins: The Definitive Guide*. Wakaleo Consulting, 2010, ISBN 978-1-4493-0535-2.

[26] VERLAENEN, K.: jBPM [online]. http://www.jboss.org/jbpm, 2011 [cit. 2011-12-27].

Dodatek A

Obsah DVD

jbpm5.3	složka s projektem jBPM 5.3 včetně všech svých součástí a databáze definic podnikových procesů a definic pracovních položek
jbpm-workflow-plugin	složka obsahující zdrojové kódy zásuvného modulu, včetně složky `work`, která obsahuje testovací instalaci Jenkins spolu s konfiguracemi ukázkových úloh
tex	složka se zdrojovými soubory technické zprávy pro sázecí systém Latex
pdf	složka s technickou zprávou v elektronické verzi s aktivními odkazy a ve verzi pro tisk
doc.html	soubor s manuálem obsahujícím podrobný popis složek a návod ke zprovoznění ukázek