VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

SYSTÉM PRO SPRÁVU KONVERZÍ DAT

DATA TRANSFORMATION MANAGER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ROMAN NEDELKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK KOČÍ, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/20118/2016/xnedel07

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Nedelka Roman**
Obor: Informační technologie
Téma: **Systém pro správu konverzí dat**
Data Transformation Manager
Kategorie: Web

Pokyny:

1. Prostudujte problematiku tvorby webových služeb, které jsou postaveny nad technologií Java, a zpracování XML dokumentů.
2. Analyzujte práci s daty a konverze dat ve firmě AIS Servis, s.r.o. Identifikujte procesy spojené s konverzemi dat a slabá místa stávajícího řešení.
3. Navrhněte koncept správy konverze dat, která bude parametrizována prostřednictvím XML dokumentů. Dokumenty definují, mimo jiné, metody konverze a způsob odeslání výsledku (email, datová schránka, apod.) Systém umožňuje definovat XML dokumenty, provádět konverze popsané těmito dokumenty a zpracovávat statistiky využití konverzí.
4. Navržený systém implementujte jako vícevrstvou architekturu, která bude zahrnovat databázový systém, byznys vrstvu EJB a webové služby SOAP.
5. Připravte vhodnou testovací sadu a vyhodnoťte splnění požadavků na systém.

Literatura:

- Oracle. The Java EE 6 Tutorial. https://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf, dostupné říjen 2016.

Pro udělení zápočtu za první semestr je požadováno:

- První tři body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kočí Radek, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Cieľom tejto bakalárskej práce bolo vytvoriť systém pre správu konverzií dát, ktorý bude tvoriť časť procesu tlače v dvoch českých poisťovniach. Ide o konverzie dát z databázy do výsledných dokumentov podľa rôznych šablón (napríklad z údajov o zákazníkovi vo formáte XML sa vytvorí zmluva vo formáte PDF). Systém je navrhnutý ako viac-vrstvová aplikácia (aplikačný server, webové rozhranie, perzistentné dáta) a implementovaný v Jave, resp. Jave EE. Pre prenos a spracovanie dát sú použité technológie založené na XML. Systém bol otestovaný a je pripravený na zaradenie do prevádzky. Časom by mal úplne nahradiť podobné, aktuálne používané systémy, ktoré sú zastarané a ťažko udržateľné a prispieť tak zvýšeniu efektivity procesu tlače.

Abstract

The objective of this bachelor's thesis was to create a data transformation manager, which will be a part of printing process in two Czech insurance companies. It is a transformation, when data from database are transformed to output documents using various templates (for example: information about a customer stored in a XML document is transformed to a contract in a PDF file format). The manager is designed as a multi-tier application (application server, web interface, persistent data) and implemented in Java or more precisely Java EE. XML-based technologies are used for the data transmission and the data processing. The manager was tested and is ready for use. It should eventually replace familiar, currently used systems, which are outdated and difficult to maintain and contribute to increase effectiveness of a printing process.

Kľúčové slová

XML, XML schéma, XSD, JAXB, webové služby, SOAP, REST, WSDL, Enterprise JavaBeans, konverzia dát, databáza, JPQL, webové služby, Java

Keywords

XML, XML schema, XSD, JAXB, web services, SOAP, REST, WSDL, Enterprise JavaBeans, data transformation, database, JPQL, web services, Java

Citácia

NEDELKA, Roman. *Systém pro správu konverzí dat.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Kočí, Ph.D.

Systém pro správu konverzí dat

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Radka Kočího, Ph.D. Ďalšie informácie mi poskytol konzultant Vojtěch Žáček z firmy AIS Servis, s.r.o. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Roman Nedelka
17. mája 2017

Poďakovanie

Rád by som poďakoval vedúcemu tejto práce Ing. Radkovi Kočímu, Ph.D. za pomoc a venovaný čas počas jej vypracovávania. Ďalej chcem poďakovať firme AIS Servis, s.r.o. za to, že mi umožnili vypracovať a použiť firemný projekt ako bakalársku prácu. Veľká vďaka patrí aj Vojtěchovi Žáčkovi za cenné rady, ktoré mi pri vypracovávaní veľmi pomohli.

Obsah

Kapitola 1

Úvod

V súčasnosti je jedným z populárnych nástrojov pre tvorbu podnikového softvéru jazyk *Java*, konkrétne jeho rozšírenie *Enterprise Edition*[1] (ďalej len Java EE), určené, ako už z názvu vyplýva, práve pre tento účel. Tento nástroj bol použitý aj pri tvorbe softvéru popisovaného v tejto práci. Ide o viac-vrstvovú aplikáciu – logika na aplikačnom serveri sprístupnená cez webové rozhranie a práca s perzistentnými dátami – spojenú s tlačou dokumentov v dvoch českých poisťovniach.

Tlač predstavuje komplikovaný proces zahŕňajúci v sebe množstvo spolupracujúcich, na sebe nezávislých modulov. Jedná sa o moduly pre generovanie dokumentov s údajmi o zákazníkoch, ich ukladanie a načítavanie z databázy, vytváranie zmlúv podľa rôznych šablón, automatizované spúšťanie tlačí, kontroly, či tlač prebehla bez problémov apod. Softvér popísaný v tejto práci spravuje časť takejto tlače spojenú s konverziami dokumentov. Ide prevažne o konverzie, kedy sa zo vstupných údajov o zákazníkoch z databázy podľa šablón vytvoria výstupné dokumenty (zmluvy, upomienky, oznámenia, tabuľky atď.) a následne sa tieto dokumenty zašlú zákazníkom, vytlačia alebo archivujú.

Účelom vyvíjaného systému je nahradiť podobný, aktuálne používaný systém, ktorý postupom času od svojho vzniku zastaral a vo všeobecnosti zefektívniť a uľahčiť prácu spojenú s konverziami resp. tlačou.

Táto práca okrem úvodnej časti pozostáva z ďalších piatich kapitol. V kapitole 2 je popísaná analýza systému, náčrt jeho konceptu a dôvody použitia zvolených technológií pri jeho tvorbe. V kapitole 3 sú podrobne popísané technológie a postupy použité pri tvorbe systému. Ďalšia kapitola (4) rozoberá návrh systému, tj. jednotlivé časti tvoriace systém a ich vzájomnú spoluprácu. Kapitola 5 obsahuje popis implementačných detailov. V poslednej časti (6) sú zhrnuté výsledky práce, očakávania a možné vylepšenia aplikovateľné v budúcnosti.

[1] voľne preložené *podnikové vydanie*

Kapitola 2

Analýza systému pre správu konverzií dát

V tejto kapitole sú rozobraté hlavné dôvody potreby prechodu zo starého systému na nový, porovnanie týchto systémov, náčrt konceptu nového systému, očakávania a zvolené technológie pre jeho tvorbu. V ďalšom texte bude systém pre správu konverzii dát označovaný názvom *Waybill*, zdedeným od svojho predchodcu. Druhým často používaným pojmom bude *průvodka*, ktorý bude označovať vstupný XML[1] súbor riadiaci konverzie. Tento názov je rovnako zdedený. Príklad zjednodušenej průvodky je v prílohe A.

2.1 Aktuálny systém pre správu konverzií dát

Aktuálne používaný, teda starý Waybill, vznikol na prelome rokov 2009 a 2010. Jeho hlavnými nedostatkami sú zastaranosť a zlá architektúra.

Zastaranosť

Momentálne používaný Waybill funguje pomocou neaktuálnych verzií rôznych technológií, často bez oficiálnej podpory. Tento fakt je spôsobený neprispôsobovaním sa modernizáciám týchto technológií. Dlhodobé ignorovanie relatívne drobných zmien, ktoré so sebou priniesla každá nová aktualizácia v priebehu pár mesiacov či rokov, vo výsledku viedlo k ťažko udržateľnému kódu, ktorý je nutné prerobiť, resp. nahradiť. Súčasná verzia je implementovaná v Jave 6, ktorá bola oficiálne zverejnená v roku 2006 a ktorej posledná aktualizácia prístupná verejnosti sa datuje k začiatku roku 2013. Rovnako používa aj neaktuálnu verziu Javy EE, konkrétne Java EE 5, vydanú v roku 2006. Taktiež beží na zastaranom aplikačnom serveri bez podpory, vydanom v roku 2006 a poslednou aktualizáciou v roku 2009 – JBoss AS 5. Všetky tri spomenuté technológie majú svoje o niekoľko stupňov novšie verzie.

Zlý návrh

Najpodstatnejším nedostatkom sa však s odstupom času ukázal byť návrh pôvodného Waybillu – išlo o koncept „všetko na jednom mieste“. Snahou bolo vyrobiť robustný softvér, ktorý v sebe zahŕňa obrovské množstvo funkcionality, pričom bola opomínaná modularita – jednotlivé časti boli husto prepojené, na sebe závislé a nefungovali jedna bez druhej. Takmer

[1] *eXtensible Markup Language* – rozšíriteľný značkovací jazyk

každá zmena v ľubovolnom module si vyžiadala zásahy v moduloch, ktoré s ňou relatívne vôbec nesúviseli. V tomto smere bude koncept nového Waybillu diametrálne odlišný.

Využívanosť

Využívanie Waybillu od svojho vzniku každým rokom stúpa a v posledných dvoch rokoch (2015, 2016) bol tento nárast extrémny. Kým v prvom roku svojho používania sa podieľal na cca 70 tis. tlačiach, o tri roky neskôr to bolo viac než 0,5 mil. a v roku 2016 viac než 1,5 mil. tlačí. Stúpajúci trend sa očakáva aj v najbližších rokoch, a preto je prechod z osemročného, zastaraného a nešťastne navrhnutého systému na nový, viacmenej nevyhnutný. Teoreticky by bolo možné starý Waybill upraviť a prispôsobiť novým verziám používaných technológií, ale bol by to komplikovaný proces, ktorý by spotreboval veľké množstvo zdrojov a času a v porovnaní s implementáciou nového systému by to bolo neefektívne. Keďže zásadnejším problémom a hlavným dôvodom novej implementácie je spomínaný koncept systému, táto možnosť aj tak odpadá.

2.2 Nový systém pre správu konverzií dát

Nový Waybill bude postavený na aktuálnych verziách Javy, Javy EE a aplikačného serveru WildFly (premenovaný JBoss), viď. tabuľka 2.1. Ide o verzie z posledných mesiacov či rokov s oficiálnou podporou. V budúcnosti teda bude snaha nespraviť rovnakú chybu ako v pôvodnom systéme a priebežne analyzovať nové verzie týchto technológií a v prípade potreby sa im prispôsobiť.

Čo sa týka konceptu nového systému, je pojatý úplne inak ako ten pôvodný. Nejde o systém zahŕňajúci v sebe všetko, naopak veľký dôraz je kladený na modularitu. Nový Waybill bude akýmsi jadrom väčšieho celku, tj. bude ním implementovaná hlavná funkcionalita a bude volať externé samostatne fungujúce moduly. Moduly s rôznymi funkciami teda nebudú na jednom mieste, ale Waybill ich bude mať za úlohu pospájať a vytvoriť logický, dobre pracujúci celok.

Očakáva sa od neho, že bude kvalitnou a stabilnou náhradou svojho predchodcu, uľahčí prácu viacerým vývojárom zapojeným v procese tlače a celkovo tento proces zefektívni.

	Java verzia	**Java EE verzia**	**Aplikačný server**
starý Waybill	6	5	JBoss 5
nový Waybill	8	7	WildFly 9

Tabuľka 2.1: Porovnanie použitých technológií a ich verzií v starom a novom systéme.

2.2.1 Koncept systému

Koncept Waybillu sa dá skrátene popísať nasledovne: viac-vrstvová aplikácia tvoriaca časť procesu tlače, ktorá komunikuje so svojim okolím (s ostatnými modulmi zapojenými v tomto procese) cez webové rozhranie a pracuje s perzistentnými dátami.

Waybill používa upravený model viac-vrstvových distribuovaných aplikácií, typických pre Javu EE, podrobne v kapitolách 3 a 4. Skladá sa z troch vrstiev: biznis vrstva[2], práca s perzistentnými dátami a rozhranie pre klientov systému.

[2] jadro aplikácie, hlavná funkcionalita

Jadro aplikácie

Biznis logiku tvoria samotné konverzie. Konverzie prebiehajú prevažne pomocou externých modulov. Externými modulmi sa v tomto kontexte rozumejú firemné aplikácie sprístupnené cez webové rozhranie, konkrétne cez webové služby, podrobne o webových službách v sekcii 3.5. Niektoré konverzie sú naprogramované priamo vo Waybille, lebo vo firme pre ne zatiaľ neexistujú vlastné systémy, no v budúcnosti by mali byť taktiež prerobené na externé aplikácie.

Sprístupnenie pre klientov

Rovnako ako externé moduly, ktoré Waybill používa, aj jeho funkcionalita je sprístupnená cez webové služby. Poskytuje dve rozhrania pre spracovanie: cez prvé klienti posielajú všetky vstupné dáta potrebné pre jednu konkrétnu konverziu popísanú v priloženej průvodke a cez druhé posielajú len požiadavku na počet spracovaní, kedy Waybill dáta z databázy načíta sám a následne spracuje.

Práca s perzistentnými dátami

Práca s perzistentnými dátami by sa dala rozdeliť na dve časti. Prvá bude pracovať so štatistikami využitia Waybillu, čiže ukladanie údajov ako dátumy a časy volania systému, dĺžka spracovania alebo spojenie s konkrétnou tlačou. Tieto údaje budú následne poskytnuté cez webové rozhranie.

Druhú časť predstavuje práca s dátami, ktoré bude Waybill načítavať a spracovávať. Ide o vstupné súbory pre konverzie obsahujúce napríklad údaje o zákazníkovi a průvodka riadiaca spracovanie.

2.2.2 Zvolené technológie pre tvorbu systému

Systém je postavený na dvoch hlavných technológiách: Java a XML. Použitie programovacieho jazyka Java je podmienené tým, že firemné oddelenie, vrámci ktorého je softvér vyvíjaný, používa rovnako jazyk Java. Čiže všetci vývojári, ktorí nejakým spôsobom prídu do styku s daným systémom (konzultácie o systéme, prípadne v budúcnosti preberanie systému), budú „javisti“, a teda použitie iného by bolo menej efektívne.

Pre prenos dát som zvolil XML. Ide o bežný formát pre prenos dát medzi aplikáciami, ktorý ma v Jave množstvo podporných funkcií a keďže XML zaručuje multiplatformovosť, v porovnaní s binárnymi súbormi napríklad odpadajú problémy s endianitou. Tiež bol pred začatím implementácie k dispozícií XSD[3] súbor popisujúci XML dokument riadiaci celý proces, čiže bolo možné jednoducho vygenerovať odpovedajúci objektový model a pracovať s ním namiesto práce priamo s XML dokumentom, viď. kapitoly 4 a 5.

2.3 Princíp prenosu a spracovania dát v novom systéme

Java je prenosný programovací jazyk a XML prenosný formát pre popis dát – sú nezávislé na platforme a v tomto smere teda so sebou veľmi dobre kooperujú [3]. Jednotlivé konverzie sú riadené XML súborom, ktorý definuje vstupy, výstupy, typy konverzií apod. Na začiatku spracovania je nutné overiť správnosť tohoto súboru a nejakým spôsobom ho spracovať.

[3] *XML Schema Definition*

2.3.1 Overovanie správnosti XML dokumentov

Overovanie správnosti XML dokumentov predstavuje proces odhaľovania chýb za účelom vyhnutia sa práce s nekonzistentnými dátami, ktoré môžu spôsobiť nechcené správanie systému [8]. Toto overovanie predstavuje dve úrovne kontroly týchto dokumentov.

Prvou je kontrola, ktorá overuje správnosť štruktúry dokumentu podľa všeobecnej špecifikácie jazyka. Musia byť splnené kritéria ako napr. každý element musí mať koncovú značku alebo každý dokument musí mať koreňový element. Dokumenty spĺňajúce tieto kritéria sú správne štrukturované (z angl. *well-formed*) [3].

Pri druhej sa overuje, či je dokument validný oproti schéme. Schémy, alebo tiež schémové jazyky, slúžia pre detailnú kontrolu, kedy sa nekontroluje len všeobecná správnosť štruktúry, ale aj splnenie rôznych obmedzení (z angl. *restrictions*). Tieto sú definované v súboroch napísaných v odpovedajúcom jazyku a prakticky popisujú novú, užívateľom definovanú syntax dokumentov. Napríklad môže ísť o obmedzenia dátových typov atribútov, prípustná množina podelementov k danému elementu apod. Okrem validácie je možné použiť schémy napr. k definovaniu nových značkovacích jazykov či pre kooperáciu s programovacími jazykmi, kedy je možné podľa dátového modelu vytvoriť odpovedajúci objektový model (z angl. *data binding*) [3].

Medzi najznámejších zástupcov schémových jazykov patria napr. *Document Type Definition* (*DTD*), *Relax NG* či *XML Schema Definition* (*XSD*) [3]. Vo Waybille bude použitý jazyk XSD, ktorý je v súčasnosti najrozšírenejší, a to hlavne vďaka jeho veľkej podpore (je W3C[4] štandardom a podporuje ho väčšina komerčných firiem).

XML Schema Definition

Ide o komplexný, v súčasnosti najpoužívanejší schémový jazyk a to aj napriek niektorým jeho nedostatkom ako napr. pomerne zložitá špecifikácia či „ukecanosť“ (schémy v danom jazyku sú syntakticky pomerne zložité aj pre veľmi jednoduché XML dokumenty). XSD súbory sú písane v XML a je ich teda možné validovať oproti samým sebe. Pripojenie XSD k XML sa definuje v koreňovom elemente XML dokumentu [3].

Základný princíp vytvárania XSD súborov je vytváranie nových dátových typov. Tie môžu byť buď jednoduché (odvodené od základných dátových typov napr. celé čísla, logický dátový typ apod.) alebo zložené (odvodené z jednoduchých). Postup vytvárania môže byť napr. taký, že sa vytvoria jednoduché dátové typy pre všetky atribúty a koncové elementy a z týchto typov sa potom vytvoria zložené dátové typy pre každý element až po úroveň koreňového [3]. Táto metóda vytvárania sa nazýva *metóda slepého Benátčana* a je len jednou z viacerých [6]. Ďalej je možné definovať veľké množstvo obmedzení, napr. počet opakovaní výskytu daného elementu, obmedzenie desatinných miest čísel, vymenovanie hodnôt atď.

Dôležitou vlastnosťou XSD, použitou aj v tomto systéme, je možnosť využitia pri data bindingu.

2.3.2 Spracovanie XML v Jave

Medzi najčastejšie akcie vykonávane pri spracovávaní XML patrí načítanie alebo generovanie nových elementov, zápis do XML dokumentu apod. Existujú rôzne nástroje, ktoré rozdelia XML dokumenty na jednotlivé časti (tzv. *parsovanie*) za nás a pripravia nám dokument na programové spracovanie – nazývajú sa parsery [3].

[4] *World Wide Web Consortium*

Parsery sú schopné čítať vstupné XML súbory či prúdy, automaticky vykonávajú validáciu a extrahujú názvy elementov, atribútov a ich hodnôt, prípadne ďalšie doplnkové informácie. Následne spracované XML ponúknu cez určité rozhranie užívateľovi, ktorý môže pracovať s jeho konkrétnymi časťami. Základne rozhrania sú dve: prúdové spracovanie a práca so stromovou reprezentáciou dokumentu [3].

Základné rozhrania pre spracovanie XML

Prúdové spracovanie je riadené udalosťami. Princíp je taký, že sa postupne načítava vstupné XML a pre jeho jednotlivé časti sú vytvárané udalosti, ktoré treba obslúžiť. Toto spracovanie je veľmi rýchle a pamäťovo nenáročné, ale ide o sekvenčné spracovanie a jeho hlavnou nevýhodou je, že pri čítaní sa nedá vracať. Typickým predstaviteľom je *SAX* (*Simple API for XML*) [3].

Pri práci so stromovou štruktúrou sa celé XML naraz načíta do pamäte. Pri tomto spracovaní je možné sa ľubovoľne vracať a okrem čítania XML dokumentu do neho aj zapisovať. Nevýhoda oproti prúdovému spracovaniu spočíva v pomalšom načítavaní a väčšej pamäťovej náročnosti. Základným predstaviteľom je *DOM* (*Document Object Model*) [3].

V Jave existuje viacero balíkov poskytyjúcich implementácie jednotlivých parserov, jedným z nich je napr. *JAXP* (*Java API for XML Processing*), ktorý zjednocuje používanie viacerých rozhraní, medzi ktorými je DOM aj SAX [3].

Parsery bývajú implicitne využívané v rôznych nástrojoch pre prácu s XML, ako napríklad *JAXB* (*Java Architecture for XML Binding*), pomocou ktorého môžeme namapovať XML dokument na objektový model v Jave. Aby mohlo mapovanie prebehnúť, musí sa XML najprv nejakým spôsobom spracovať a vtedy sa použije konkrétna parsovacia technológia [3].

Mapovanie XML dokumentu do objektovej reprezentácie v Jave

Rozhranie JAXB umožňuje mapovanie XML dokumentu na Java triedy pomocou XSD súboru. Využíva sa skutočnost, že XML dokument už bol detailne popísaný pomocou XSD a nie je teda nutné písať odpovedajúce triedy v Jave – tieto triedy vznikajú automatickým generovaním. To prebieha spravidla na začiatku práce a používa sa naň *xjc* (*XML to Java Compiler*). Vygenerujú sa Java triedy odpovedajúce zloženým typom z XSD a premenné daných tried odpovedajúce jednoduchým typom. Takisto vznikne továrenská trieda *ObjectFactory.java*, nesúca informácie o vygenerovaných triedach. Následne je možné načítať ľubovolný XML dokument (validný) do vygenerovaného objektového modelu (tzv. *unmarshalling*) alebo naopak objektový model zapísať do XML dokumentu (tzv. *marshalling*) [3].

Vo Waybille bude takýmto spôsobom vygenerovaná objektová reprezentácia průvodky. Na tieto objekty sa pri každom procese konverzií namapuje konkrétna průvodka, podľa ktorej sa potom programovo riadi celý proces konverzií.

Kapitola 3

Technológie použité pri tvorbe systému pre správu konverzií dát

V tejto kapitole sú popísané zvolené technológie pre realizáciu systému. Sú tu rozobraté možnosti implementácie biznis logiky, sprístupnenia tejto logiky klientom a práca s perzistentnými dátami.

3.1 Programovací jazyk Java a Java platforma

Java je programovací jazyk a zároveň platforma. Java ako programovací jazyk je vysoko úrovňový objektovo-orientovaný jazyk so špecifickou štruktúrou a štýlom. Java ako platforma je prostredie, v ktorom bežia programy naprogramované v Jave. Existuje viacero Java platforiem, napr. *Java Platform, Standard Edition (Java SE)* alebo *Java Platform, Enterprise Edition (Java EE)*. Všetky platformy pozostávajú z JVM[1] a z API[2] [4].

Java SE poskytuje jadro funkcionality programovacieho jazyka Java – definuje všetko od základných dátových typov až po triedy pracujúce s grafickým užívateľským rozhraním či s databázou. Keď sa v nejakej súvislosti hovorí o programovacom jazyku Java, väčšinou ide o Java SE API [4].

Java EE je postavená nad Javou SE. Poskytuje API a prostredie pre vývoj a prevádzku mohutných, viac-vrstvových, škálovateľných, spoľahlivých a bezpečných aplikácií [2].

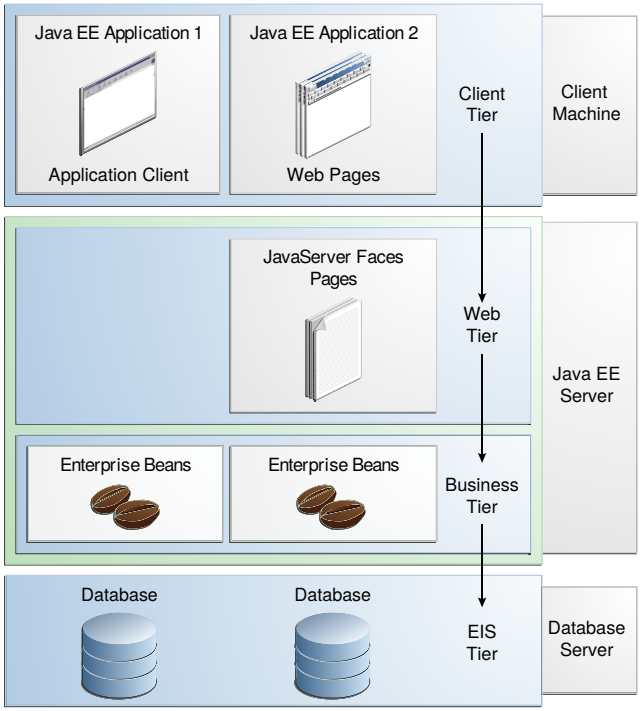
3.2 Model Java EE aplikácií

Tento systém používa model viac-vrstvových distribuovaných aplikácií, resp. jeho miernu modifikáciu, obr. 3.1. Jednotlivé vrstvy sú samostatne pracujúce, nezávislé celky, ktoré sa nachádzajú v rozličných miestach v sieti a sú navzájom prepojené.

Java EE server

Java EE server (často označovaný ako aplikačný server) je serverová aplikácia, ktorá implementuje Java EE rozhrania a poskytuje Java EE služby [2]. Existuje viacero Java EE

[1] *Java Virtual Machine* (virtuálny stroj) – zaisťuje väzbu na hardvér a interpretuje bajtkód
[2] *Application Programming Interface* (aplikačné programové rozhranie) – knižničné triedy



Obr. 3.1: Architektúra distribuovaných viac-vrstvových Java EE aplikácií (prebraté z [5]). Webová a biznis vrstva na Java EE serveri spolu tvoria strednú vrstvu, ktorá spracováva klientské požiadavky a pracuje s dátami z dátovej vrstvy.

serverov. Tento systém používa WildFly 8, vytvorený firmou JBoss a momentálne vyvíjaný firmou Red Hat. Ide o voľne dostupný softvér s LGPL[3] licenciou.

Klientská vrstva

Java EE klient je zvyčajne buď webový alebo aplikačný. Webový klient sa skladá z webových stránok a prehliadača. Ide o tzv. *thin* klienta, ktorý väčšinou nevykonáva zložitejšie operácie, akými sú dopytovanie databázy či vykonávanie komplexnej biznis logiky. Webový klienti sú prepojení s webovou vrstvou aplikačného serveru. Aplikační klienti pristupujú priamo k biznis vrstve [5].

3.3 Komponenty Java EE aplikácií

Java EE aplikácie sú zostavené komponentami. *Java EE komponent* je samostatná jednotka napísaná v Jave, prekladaná rovnako ako ostatné programy napísané v Jave. Rozdielom oproti štandardným triedam v Jave je napríklad kontrola správnosti komponentov oproti Java EE špecifikácii alebo ich nasadzovanie do obehu, kde sú spúšťané a spravované Java EE serverom [5].

Java EE špecifikácia definuje nasledujúce komponenty:

- aplikační klienti a *applety*[4] sú komponenty na klientskej vrstve

[3] GNU Lesser General Public License
[4] malé klientské aplikácie bežiace v JVM vrámci webového prehliadača

- *Java Servlet*, *JavaServer Faces* a *JavaServer Pages* sú webové komponenty na vrstve s aplikačným serverom
- *Enterprise JavaBeans* (ďalej len EJB) sú biznis komponenty na vrstve s aplikačným serverom

Kontajnery

Vo všeobecnosti je veľmi zložité viac-vrstvové aplikácie naprogramovať, lebo musia obsahovať množstvo komplikovaného kódu napr. pre spracovanie transakcií, riadenia stavu aplikácie či prácu s vláknami. V Jave EE však existujú kontajnery, ktoré takéto a podobné procesy uľahčujú. Kontajnery tvoria rozhranie medzi komponentami a „nižšou“ funkcionalitou, ktorú poskytuje platforma pre daný komponent. Aby mohli jednotlivé komponenty vykonávať svoje funkcie, musia byť najprv preložené do Java EE modulu a umiestené do svojho kontajneru. V tomto procese sú zahrnuté rôzne nastavenia spojené s kontajnermi a komponentami, ktoré napr. zaručujú správu transakcií, bezpečnosť či komunikáciu medzi klientom a EJB [5].

3.4 Enterprise JavaBeans

Enterprise JavaBean je komponent na strane serveru, ktorý obaľuje biznis logiku aplikácie. Biznis logika predstavuje podstatu aplikácie. Ide o komponenty zľahčujúce tvorbu rozsiahlych distribuovaných aplikácií. EJB kontajner napr. uľahčuje prácu vývojárom automatickým spracovaním transakcií a inými službami na úrovni systému. Keďže obsahujú logiku aplikácie, odbremeňujú od tohoto problému vývojárov prezentačnej vrstvy, čiže klienti nemusia pristupovať k databáze či vykonávať iné funkcie poskytované cez EJB. Ide o prenosné komponenty, ktoré využívajú štandardné API a môžu bežať na akomkoľvek aplikačnom serveri, ktorý ich podporuje [5].

Rozlišujú sa dva základné typy EJB: *session* a *message-driven*. Výraz session sa bude v tomto texte často vyskytovať. V tomto kontexte nemá doslovný preklad a je možné si pod ním predsaviť niečo ako interakciu/spojenie v určitom čase. Message-driven beany nebudú v systéme použité a ďalej budú rozobraté len session beany.

Ide o klasickú triedu v Jave, obohatenú o anotácie[5].

3.4.1 Session Bean

Session beany zapúzdrujú biznis logiku, ktorá môže byť klientom programovo vyvolaná lokálne, vzdialene alebo cez webové služby – pre sprístupnenie aplikácie nasadenej na serveri klient vyvoláva metódy implementované v session beanoch. Existujú tri typy session beanov: stavové (z angl. *stateful*), bezstavové (z angl. *stateless*) a singleton.

Stateful Session Bean

Stateful session bean je biznis objekt, ktorý nesie informácie o určitom stave. Ide o stav objektu, ktorý je daný hodnotami jeho inštančných premenných – inštančné premenné reprezentujú stav unikátnej klient/bean session. Klient volá biznis metódy daného objektu, tým môže meniť jeho inštančné premenné, čiže jeho stav. Klient takýmto spôsobom komunikuje s beanom, a preto sa takémuto stavu často hovorí konverzačný stav. Pre klienta

[5] anotácie predstavujú doplnkové informácie o programe pre prekladač, aplikačný server apod.

je na serveri vytvorená inštancia beanu, ktorá je spojená s daným klientom a pri každom ďalšom volaní metód z tohoto beanu sa použijú dáta (stav) z poslednej interakcie s klientom. Takýto typ beanu nie je zdieľaný, tj. vrámci jednej session môže spolupracovať len s jedným klientom. Keď klient ukončí prácu s takýmto beanom, bean už nie je s daným klientom v žiadnom vzťahu. Stav je uchovávaný len počas session a po nej zaniká. Trieda predstavujúca takýto bean musí byt anotovaná ako `@Stateful` [5].

Príkladom použitia môže byť napr. košík v e-shope, kedy je treba si pamätať aké rôzne položky si zákazník vybral, upravil, vymazal atď. a na základe toho určovať ďalšie akcie.

Stateless Session Bean

Tento typ beanu neudržiava s klientom konverzačný stav, tj. medzi obsluhami jednotlivých požiadavok neuchováva informácie pre klienta. Takýto typ beanu pri obsluhe požiadavky klienta (pri vykonávaní volanej metódy) môže obsahovať hodnoty inštančných premenných špecifických pre daného klienta, ale len po dobu obsluhy danej požiadavky. Keď obsluha skončí (skončí vykonávanie metódy), stav sa neukladá/neudržiava. Stateless beany sú na serveri uložené v tzv. *poole*, z ktorého sa pri obsluhe požiadavky odoberú a po skončení obsluhy vrátia späť. Keď zrovna nejaký stateless bean neobsluhuje požiadavku, sú všetky stateless beany v poole rovnaké a EJB kontajner priraďuje ľubovolnému klientovi ľubovolný bean [5].

Trieda predstavujúca takýto bean musí byt anotovaná ako `@Stateless`.

Singleton Session Bean

Inštancia singleton session beanu je v aplikácii vytváraná jedenkrát a existuje do konca jej životného cyklu. Udržiava stav medzi jednotlivými invokáciami klientmi, no neudržuje stav medzi behmi serveru resp. opätovnými nasadeniami aplikácie na server. Singleton session beany sú navrhnuté pre udalosti, pri ktorých jedna inštancia enterprise beanu môže byť súbežne sprístupňovaná viacerým klientom. Funkcionalitou sú podobné stateless session beanom, no narozdiel od poolu viacerých stateless beanov, z ktorých ktorýkoľvek môže klientovi odpovedať, singleton session bean je v aplikácii len jeden. Aplikácie používajúce singleton session beany môžu definovať napr. rôzne inicializácie pri štarte servera či uvoľňovanie zdrojov pred vypnutím servera [5].

Trieda predstavujúca takýto bean musí byt anotovaná ako `@Singleton`.

3.4.2 Sprístupnenie EJB

V systéme sú použité dva typy rozhraní: lokálne a vzdialené. Keď k EJB pristupujú klienti vrámci rovnakého JVM, napríklad klientom EJB je iné EJB na tom istom aplikačnom serveri, k prístupu sa používa lokálne rozhranie. Toto rozhranie má anotáciu `@Local`. Pri vzdialených rozhraniach (anotácia `@Remote`) klient beží na inom JVM ako bean, ku ktorému pristupuje. Pre sprístupnenie beanov cez vzdialene rozhrania je použitý *JNDI*[6] mechanizmus. Pre implementáciu vyhľadávania a sprístupňovania rozhraní pomocou JNDI bola prevzatá firemná trieda.

[6] *Java Naming and Directory Interface*, viac na http://www.oracle.com/technetwork/java/jndi/index.html

3.5 Webové služby

Webové služby sú komponenty webových aplikácií poskytujúce možnosti spolupráce medzi aplikáciami na rôznych platformách. Ich oporou sú HTTP a XML.

Z konceptuálneho hľadiska je služba softvérový komponent poskytovaný cez nejaký koncový bod. Poskytovateľ (z angl. *provider*, dá sa predstaviť ako server) a konzument (z angl. *consumer*, dá sa predstaviť ako klient) služby medzi sebou komunikujú pomocou požiadavok a odpovedí v podobe samostatných dokumentov obsahujúcich detaily komunikácie [5].

Z technického hľadiska existuje viacero implementácií webových služieb – dve základné sú založené na princípoch *SOAP* (*Simple Object Access Protocol*) a *REST* (*Representational State Transfer*) [5]. Pre prácu s webovými službami Java EE poskytuje rozhrania *JAX-WS* (*Java API for XML Web Services*) resp. *JAX-RS* (*Java API for RESTful Web Services*). V ďalšom texte sa budú vyskytovať skrátené označenia SOAP a REST webové služby.

3.5.1 Webové služby založené na SOAP

Tento typ webových služieb z hľadiska architektúry zahrňuje tri jednotky [1]:

- *poskytovateľ služby*, ktorý danú službu vytvára a publikuje jej popis v registri služieb
- *register služieb*, ktorý umožňuje online sprístupnenie služby
- *žiadateľ služby*, ktorý nájde službu dopytovaním registru, následne získa jej popis, vytvorí väzbu na jej implementáciu a začne ju používať

Komunikácia (operácia) medzi týmito jednotkami prebieha pomocou SOAP a popis webovej služby je publikovaný pomocou *WSDL* (*Web Services Description Language*).

SOAP

SOAP je štandardom W3C. Ide o protokol pre sprístupňovanie webových služieb založený na XML. Slúži na komunikáciu medzi aplikáciami a definuje formát pre posielanie a prijímanie správ. Je nezávislý na platforme. SOAP správy sú zvyčajne prenášané cez HTTP [1].

WSDL

Rovnako ako SOAP je štandardom W3C a založený na XML. Poskytuje informácie ako používať danú webovú službu vrátane popisu metód služby a vytvorenia väzby na implementáciu [1].

3.5.2 Webové služby založené na REST

Tento typ webovej služby používa architektúru klient-server. REST neobmedzuje klient-server komunikáciu na špecifický protokol, no najviac používa HTTP. REST webové služby môžu byť popísané jazykom *WADL* (*Web Application Description Language*). WADL súbory popisujú požiadavky, ktoré môžu byť službe adresované, URI[7] služby a dáta, ktoré služba očakáva a obsluhuje [1].

REST sa spolieha na tri hlavné princípy: adresovatelnosť, jednotné rozhranie a bezstavovosť. Pre adresovatelnosť REST používa prácu so zdrojmi, ktoré sú identifikované cez URI. Zdroj je akákoľvek forma informácie, ktorá môže byť pomenovaná a odkazovaná (napr.

[7] Uniform Resource Identifier

dokument, riadok databázy, výsledok vyhľadávania apod.). Pri použití HTTP sú zdroje prístupné cez HTTP štandard a jednotné rozhranie. Sú podporované štyri základné operácie: vytvorenie, čítanie, aktualizácia a mazanie, ktoré sú implementované použitím HTTP metód POST, GET, PUT a DELETE. Čo sa týka bezstavovosti, REST požiadavky obsahujú všetky informácie potrebné pre server, aby mohol vykonať požadované akcie. Server sa nikdy nespolieha na informácie z predchádzajúcich požiadavok pre odpoveď na aktuálnu požiadavku [1].

3.5.3 Porovnanie webových služieb

Tento odstavec vychádza z informácií uvedených v zdroji [9]. Porovnanie REST a SOAP webových služieb je uvádzané z viacerých hľadísk, pričom pri výbere je treba zvoliť, ktoré je pre nás prioritné. Z hľadiska ceny na vývoj je REST služba lacnejšia, pretože je založená na jednoduchšej technologickej infraštruktúre, kde webové služby môžu byť implementované s minimálnym použitím vývojových nástrojov. Taktiež pri porovnaní rýchlosti odozvy, REST preukázal kratšie časy odozvy a lepšiu prenosovú kapacitu ako SOAP. Čo sa týka použitej pamäte pre prenos zdrojov, tak aj v tomto ohľade má REST výhodu nad SOAP. Jednou z výhod SOAP oproti REST je menšia chybovosť, keďže SOAP obsahuje vstavanú funkcionalitu pre spracovávanie chybových stavov. Ďalšou a hlavnou výhodou SOAP je väčšia spoľahlivosť a bezpečnosť. SOAP využíva štandard WS-Security pre podpisovanie a šifrovanie správ, čím sa prenos dát stáva bezpečnejší.

3.6 Práca s perzistentnými dátami

Waybill pracuje s relačnou databázou[8]. Archivuje štatistiky svojho používania a vyberá z databázy dokumenty potrebné pre konverzie. V Jave existuje viacero nástrojov pre prácu s perzistentnými dátami. Pri práci s EJB a aplikačným serverom je vhodné použiť *Java Persistence API*[9].

3.6.1 Objektovo-relačné mapovanie

Objektovo-relačné mapovanie (ďalej len ORM) poskytuje spomenuté JPA. Ide o proces mapovania prvkov databázy na prvky v Jave – umožňuje teda pracovať s databázou programovo v Jave. Základným komponentom JPA je *entita*, ktorá predstavuje nejaký perzistentný objekt, v relačnej databáze typicky tabuľku. Inštancia triedy v Jave reprezentujúca takúto entitu potom predstavuje jeden riadok/záznam danej tabuľky. Pre takéto mapovanie sa v triedach používajú špeciálne anotácie konkretizujúce vlastnosti entít. Nástroj pre spracovanie entít pomocou JPA sa nazýva správca entít (z angl. *entity manager*) [5].

3.6.2 Správca entít

S entitami sa pracuje pomocou správcu entít. Každý správca je spojený s nejakým perzistentným obsahom – spravuje entity nejakého dátového úložiska, spravidla tabuľky databázy. Rozhranie pomocou ktorého správca spravuje entity, obsahuje metódy napr. pre vytvorenie dotazu, odstránenie inštancie entity, vyhľadanie podľa primárneho kľúča apod.

[8] relačná databáza – dáta v sú v nej organizované v súlade s relačným modelom dát a ten sa nazýva relačný, pretože dáta sú podľa tohoto modelu logicky štrukturované do relácií (tabuliek), tj. relácie v relačnom modeli označuje tabuľku – názov *relačná* teda v tomto prípade nesúvisí s reláciami medzi tabuľkami [10]

[9] voľne preložené *Java aplikačné rozhranie pre prácu s persistentnými dátami*

Správca je previazaný s perzistentnými dátami pomocou perzistentných jednotiek. Tieto jednotky a ich nastavenia sú definované v samostatnom dokumente v niektorom module aplikácie, napr. v module s EJB. Je v ňom napr. definovaný jednoznačný názov jednotky alebo tzv. *provider*, ktorý špecifikuje konkrétnu implementáciu JPA správcu (vo Waybille ide o *Hibernate*[10]) [5].

3.6.3 Dopytovanie entít

Jednou z možností dopytovania entít je použitie jazyka *JPQL* (*The Java Persistence Query Language*). Tento jazyk definuje univerzálne dopyty, ktoré sú nezávisle od typu dátovej vrstvy – čiže umožňuje programovo v Jave napr. dopytovať rôzne databázové systémy rovnakým spôsobom. Štruktúra tohoto jazyka je veľmi podobná klasickému dopytovaciemu jazyku *SQL*[11]. Nepracuje sa teda priamo s databázou ale s objektami v Jave, ktoré odpovedajú tabuľkám databázy.

Dopytovať je možné dynamicky pomocou metódy `EntityManager.createQuery`, keď sú dopyty definované vrámci biznis logiky aplikácie. Druhým spôsob je statický pomocou metódy `EntityManager.createNamedQuery`, kedy sú dopyty definované priamo v triede entity a neskôr volané pomocou definovaného mena [5].

3.6.4 Transakčné spracovanie pri volaní biznis logiky

Aby bola zaručené správne operovanie s dátami (napr. ošetrenie súbežného prístupu k dátam alebo čiastočne spracovaných dát pri zlyhaní systému), používajú sa transakcie, resp. transakčné spracovanie.

Databázová transakcia je logická jednotka vykonávania programu pracujúceho s dátami v databáze. Je tvorená postupnosťou databázových operácií, ktoré čítajú a modifikujú dáta v databázi. Musí spĺňať štyri základné vlastnosti [10]:

- *atomičnosť* – transakcia predstavuje z hľadiska vykonávania nedeliteľný celok
- *izolovanosť* – je ošetrený súbežný prístup viacerých transakcií k rovnakým dátam
- *konzistentnosť* – izolovaná transakcia neporušuje konzistenciu databázy, čiže ak bola databáza v konzistentnom stave pred zahájením transakcie, bude v konzistentnom stave i po skončení transakcie
- *trvalosť* – potom, čo transakcia úspešne skončí, budú mať všetky zmeny v databáze, ktoré transakcia vykonala, trvalý charakter a to aj pri výpadku systému

Transakcia môže skončiť úspešne – zmeny sa premietnu do databázy a transakcia je v stave *potvrdená* (z angl. *commit*). Pri neúspešnom vykonávaní sa musia všetky čiastočné zmeny vrátiť a databáza musí byť v stave, v akom bola pred začiatkom vykonávania transakcie – táto operácia sa nazýva *rollback* [10].

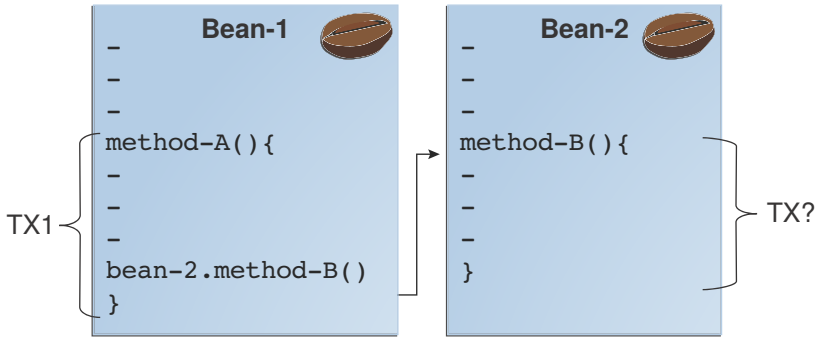
Java EE ponúka dva varianty prístupu k transakčnému spracovaniu: transakcie spracovávané kontajnerom a transakcie spracovávané beanom. Obidva prístupy pracujú so session, aj message-driven beanmi. Ak nie je explicitne špecifikovaný typ spracovania, použije sa implicitné nastavenie a teda spracovávanie kontajnerom. V systéme je použitý a tu bude popísaný len prístup s kontajnerom.

[10] Hibernate je aplikačný rámec pre ORM a jednou z implementácií JPA, viac na http://hibernate.org/
[11] *Structured Query Language* – štruktúrovaný dopytovací jazyk

Transakcie spracovávané kontajnerom

Ide o spracovávanie EJB kontajnerom, ktorý automaticky určuje hranice transakcie (začiatok, potvrdenie) – nie je nutné explicitne v zdrojovom kóde označovať začiatok a koniec transakcie. Spojenie s transakciami sa špecifikuje metódam (eventuálne celej triede) pomocou transakčných atribútov v podobe anotácií, prípadne sú použité implicitné nastavenia. Kontajner spúšťa transakciu typicky v okamihu pred začatím vykonávania EJB metódy a potvrdzuje ju (v prípade, že nenastane rollback) v okamihu pred ukončením danej metódy [5].

Obrázok 3.2 ilustruje príklad, keď nejaký bean obsahuje metódu A, ktorá je v samostatnej transakcii a táto metóda volá metódu B iného EJB. V takomto prípade nastáva otázka, či má metóda B bežať vrámci transakcie metódy A alebo má byť vykonávaná v novej, samostatnej transakcii. Odpoveďou na túto otázku je atribút transakcie metódy B.



Obr. 3.2: Pole pôsobnosti transakcií (prebraté z [5]).

Tieto atribúty definujú pole pôsobnosti transakcií. Klient môže volať biznis metódu z miesta (väčšinou z metódy iného beanu), ktoré je alebo nie je v transakcii – vlastnosti transakcie volanej metódy potom závisia od typu atribútu, viď. obrázok 3.3.

Pri tomto spôsobe spracovania existujú dva spôsoby rollbacku. Prvým je automatický rollback transakcie kontajnerom, ak je vyhodená systémová výnimka. Druhým je explicitná inštrukcia predaná beanom kontajneru pomocou metódy `setRollbackOnly` rozhrania `EJBContext`. Pri vyhodení aplikačnej výnimky beanom, rollback nie je automatický, ale môže byt započatý volaním tejto metódy.

Transakcie spracovávané beanom používajú explicitné značenie hraníc transakcie vnútri beanu, a teda vyžadujú viacero zdrojového kódu. Tento typ spracovania sa v systéme nepoužíva a nebude ďalej rozoberaný.

Transaction Attribute	Client's Transaction	Business Method's Transaction
`Required`	None	T2
`Required`	T1	T1
`RequiresNew`	None	T2
`RequiresNew`	T1	T2
`Mandatory`	None	Error
`Mandatory`	T1	T1
`NotSupported`	None	None
`NotSupported`	T1	None
`Supports`	None	None
`Supports`	T1	T1
`Never`	None	None
`Never`	T1	Error

Obr. 3.3: Transakčné atribúty a ich polia pôsobnosti (prebraté z [5]). T1 a T2 predstavujú transakcie spravované EJB kontajnerom. Stĺpec *Client's Transaction* predstavuje transakciu spojenú s klientom, ktorý volá EJB metódu – vo väčšine prípadov je klientom iný enterprise bean. Posledný stĺpec popisuje transakciu volanej biznis metódy v závislosti od zvoleného atribútu. Transakcia T2 je spustená kontajnerom v okamihu pred začatím vykonávania metódy.

Kapitola 4

Návrh systému pre správu konverzií dát

V tejto kapitole je podrobne rozobratý návrh systému. Je tu opísaná architektúra systému, spracovanie vstupných dát a následné riadenie konverzií a princíp konverzií.

4.1 Architektúra systému pre správu konverzií dát

Systém používa mierne upravený model viac-vrstvových distribuovaných aplikácií popísaný v kapitole 3. Na aplikačnom serveri je biznis vrstva s EJB pracujúca s databázou. Webová vrstva, cez ktorú bude aplikácia vystavená klientom, sa nachádza vo webovom kontajneri, ktorý komunikuje s aplikačným serverom. Aplikácia je vystavená klientom cez webové služby.

4.1.1 Zvolené typy EJB pre realizáciu biznis logiky systému

Jedno zavolanie systému je nezávislé od iných volaní, tj. systém neuchováva medzi jednotlivými klientskými požiadavkami žiadne informácie. Klient pošle požiadavku a v nej všetky potrebné informácie pre systém a ten ju spracuje nezávisle na ostatných. Pre takýto typ funkcionality sú v systéme použité bezstavové session beany.

Mimo bezstavových beanov sú použité aj dva singleton session beany. Jeden slúži na načítanie kľúčov pre digitálne podpisovanie PDF súborov a druhý funguje ako indikátor možnosti spustenia nového spracovania (ak práve prebieha spracovanie, nie je možné spustiť nové).

4.1.2 Rozhrania systému poskytované klientom

Systém poskytuje dva, resp. tri spôsoby sprístupnenia svojej funkcionality. Ide o dve hlavné rozhrania pre spustenie spracovania a tretie, menej využívané, slúžiace pre zisťovanie štatistík využívania. Všetky tri rozhrania sú webové a implementované pomocou webových služieb.

Rozhrania pre spustenie spracovania

Prvé rozhranie je implementované pomocou webovej služby založenej na SOAP. Klient v ňom posiela vstupné súbory, ktoré majú byť skonvertované a průvodku riadiacu dané

spracovanie. Systém buď vráti správu o úspešnom spracovaní alebo vyhodí vlastnú výnimku s popisom chyby.

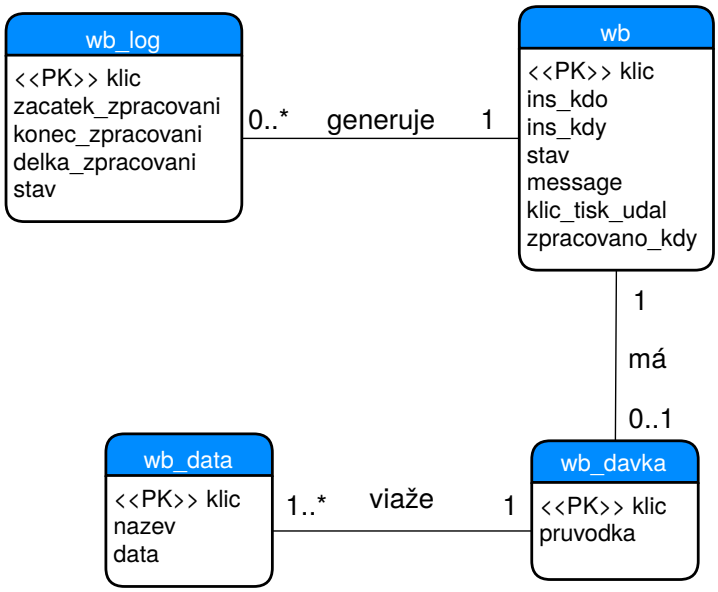
Pre druhé rozhranie je použitá REST webová služba. Klient neposiela žiadne súbory, iba požiadavku na počet spracovaní (celé číslo). Aplikačný server sám vyberie súbory pre konverzie a průvodku z databázy.

Rozhranie pre zistenie štatistík využitia

Toto rozhranie je realizované pomocou REST webovej služby. Klient v požiadavke predá dátum, prípadne časový interval a systém vráti štatistku z daného obdobia – ide o dopytovanie záznamov v tabuľke s logmi. Štatistika bude zahŕňať počty spracovaní (celkom, úspešné a neúspešné), časy spracovaní (najkratší, najdlhší a priemerný) a detailné informácie o neúspešných spracovaniach.

4.1.3 Práca systému s perzistentnými dátami

Waybill pracuje s relačnou databázou. Tabuľky (entity), s ktorými pracuje sú znázornené v ER diagrame[1] na obrázku 4.1. Najpodstatnejšiu časť modelu tvorí tabuľka *wb*, ktorá predstavuje údaje o samotnej tlači a všetky ostatné tabuľky zapojené v procese tlače s ňou buď priamo, alebo nepriamo súvisia. Waybill okrem nej pracuje s tabuľkami reprezentujúcimi vstupné súbory, průvodky a logy. Do tabuľky *wb* vstupuje a vystupuje množstvo ďalších referencií, ale vo Waybille nemajú využitie a preto tu nie sú spomenuté.



Obr. 4.1: ER diagram popisujúci vzťahy medzi tabuľkami používanými v systéme. Tabuľka *wb* predstavujúca tlačovú požiadavku je naviazaná na tabuľku *wb_davka*, ktorá obsahuje informácie o průvodke. Táto tabuľka je naviazaná na *wb_data*, ktorá predstavuje vstupný súbor pre konverzie definovaný v danej průvodke. Vo *wb_log* sú uložené štatistky o jednotlivých spracovaniach.

[1] entitno-vzťahový (z ang. *entity-relationship*) diagram – nástroj pre konceptuálne modelovanie relačných databázií

4.2 Konverzie dokumentov v systéme

Konverzie dokumentov predstavujú hlavný účel systému. Ide o transformácie XML dokumentov do výstupných formátov (PDF, XLS, XML apod.). Tieto XML dokumenty predstavujú nejaké štrukturované informácie, napríklad údaje o osobách alebo firmách. Výstupné dokumenty týmto informáciám potom dodajú kontext a formu, tj. napríklad zmluva s údajmi o zákazníkovi, upomienka apod. Výstupné dokumenty vznikajú podľa šablón.

Waybill používa dva prístupy ku konvertovaniu. Prvým (primárnym) je použitie externých modulov (firemné aplikácie) a druhým je vlastná implementácia konvertorov. V budúcnosti bude snaha o prechod na použitie čisto externých modulov.

Využitie externých modulov

Externé moduly, ktoré systém používa, sú sprístupnené ako webové služby založené na SOAP a WSDL. Waybill bude vystupovať ako klient týchto webových služieb. Súbory potrebné pre ich sprístupnenie budú generované podľa publikovaných WSDL súborov a budú súčasťou zdrojového kódu Waybillu.

Takýmto spôsobom budú realizované konverzie z XML na PDF a XLS, ktoré budú tvoriť najväčšiu časť konverzií.

Vlastná implementácia konverzií

Konverzie vytvorené takýmto spôsobom budú využívané menej. Ide o konverzie z XML na textové, XML alebo HTML súbory, použitím XSL[2] transformácií.

4.3 Spracovanie vstupného XML a riadenie konverzií

Každý proces konverzií je riadený vstupnou průvodkou, ktorá musí spĺňať určité pravidlá, mať určitú formu. Tieto pravidlá definuje XSD súbor (príloha B), podľa ktorého sa na začiatku implementácie vygeneruje objektový model průvodky a začlení sa do zdrojového kódu projektu. Na začiatku každého spracovania sa potom průvodka namapuje na vygenerovaný objektový model (unmarshalling) a ďalej riadi spracovanie/konverzie.

Validácia průvodky a dát pre konverzie

Najviac nevyžiadaného správania systému bude nastávať pri nekonzistentnosti vstupných dát. Prichádzajúca průvodka a dáta od klienta, ako aj průvodka a dáta vybrané z databázy, budú teda na začiatku spracovania detailne validované. Po časti implementujúcej samotné konverzie pôjde o druhý najzásadnejší modul systému.

Pri mapovaní průvodky na objektový model na začiatku spracovania prebehnú implicitné kontroly na správnu štruktúrovanosť XML dokumentu a validácia oproti XSD. Pre detailnejšie kontroly bude implementovaný samostatný modul, ktorý bude kontrolovať správnosť jednotlivých častí tvoriacich průvodku (*sources*, *methods*, *converts*...) a vstupných dát pre konverzie. Prebehnú tu kontroly ako napr. prítomnosť vstupných dát tak, ako sú definované v průvodke (časť *declarations-source*), správnosť atribútov pri danom type konverzie (časť *declarations-method*) alebo prítomnosť definície, určujúcej, čo sa má s výsledkom spraviť (časť *actions-send*).

[2] *eXtensible Stylesheet Language*, viac info na https://www.w3schools.com/xml/xsl_intro.asp

Pripravenie dát pre konverzie

V prípade správnosti vstupných dát sa môže prejsť k samotným konverziám. Podľa informácií z průvodky sa musia nejakým spôsobom najprv pripraviť dáta, ktoré budú do konverzií vstupovať.

Zdroje definované v průvodke (*declarations-source*) sa nájdu medzi zdrojmi pre konverzie zo vstupu, načíta sa ich obsah a uložia sa do pamäte pre ďalšie použitie. K týmto zdrojom sa bude neskôr (pri konverziách) pristupovať cez ich jednoznačné ID, takže k ich uloženiu je vhodné použiť hešovaciu tabuľku, ktorej kľúčom bude ID zdroja a hodnotou obsah súboru.

Podobným spôsobom sa z deklaračnej časti spracujú metódy. Metódy určujú typy konverzií a doplnkové informácie pre daný typ konverzie (napr. názov šablóny pre transformáciu, názov elektronického podpisu pre PDF). Pre každú metódu sa vytvorí objekt pre ňu špecifický, v ktorom bude implementovaná odpovedajúca konverzia. Rovnako ako zdroje, budú metódy sprístupňované pomocou ID, takže sa opäť použije hešovacia tabuľka, kde na ID metódy bude namapovaný objekt, v ktorom je implementovaná funkcionalita pre danú konverziu.

Vykonanie konverzií a spracovanie výsledkov

Pre lepšiu pochopenie nasledujúceho textu je dobré si najprv pozrieť prílohu A. Po úspešnom pripravení zdrojov a metód možno pristúpiť k samotným konverziám. Časť *convert* v sekcii s akciami definuje ID zdroja a ID metódy, podľa ktorých sa načítajú údaje z hešovacích tabuliek. Zdroj sa pošle na konverziu do daného objektu metódy a prebehne konverzia. Výsledok sa uloží do tabuľky zdrojov, no kľúčom bude v tomto prípade ID danej konverzie (výsledok konverzie sa akoby sám stáva zdrojom).

V časti *send* je potom definované, ako výsledok konverzie spracovať. K výsledku konverzie pridáva ID cieľa (*declarations-target*). Cieľom je väčšinou emailová adresa, na ktorú sa ako príloha pošle skonvertovaný súbor.

4.4 Ošetrenie súbežných volaní systému

Pri volaní systému cez REST rozhranie by mohlo dôjsť k súbežnému prístupu k rovnakým dátam v databáze a tým pádom k chybám. Tento potenciálny problém je vyriešený pomocou príznaku v singleton session beane, ktorý je inicializovaný pri štarte serveru alebo pri nasadení aplikácie na server. Pri každom klientskom volaní sa skontroluje hodnota príznaku a podľa toho sa povolí/nepovolí nové spracovanie. V prípade, že sa spracovanie povolí, ďalšie spracovanie je zakázané zmenou príznaku a bude umožnené až po dobehnutí toho aktuálneho.

Taktiež je zaistené transakčné spracovanie pomocou EJB kontajnera.

Kapitola 5

Implementácia systému pre správu konverzií dát

V tejto kapitole sú popísané implementačné detaily – zvolené technológie, postupy, aplikačné rámce, knižnice apod. Prehľad najhlavnejších technológií je v tabuľke 5.1.

	názov	**url**
operačný systém	*Ubuntu*	https://www.ubuntu.com/
vývojové prostredie	*NetBeans IDE*	https://netbeans.org/
verzovací systém	*Apache Subversion*	https://subversion.apache.org/

Tabuľka 5.1: Prehľad použitých nástrojov a odkazy na ich oficiálne WWW stránky.

Generovanie zdrojového kódu

Na začiatku implementácie bolo vygenerovaných a začlenených do projektu niekoľko súborov. Vygenerované súbory tvoria cca 25-30 % všetkého zdrojového kódu. Takýmto spôsobom boli do projektu začlené implementácie klientov webových služieb vygenerovaných pomocou nástroja *wsimport*[1]. Tento nástroj vygeneruje klienta webovej služby pomocou jej WSDL súboru. Takto boli vygenerovaní klienti pre niektoré konverzie dokumentov a odoslanie emailu/SMS. Ďalej bola vygenerovaná objektová reprezentácia průvodky pomocou nástroja xjc. Toto generovanie prebieha pomocou XSD súboru průvodky. Pre generovanie boli použité jednoduché skripty v bashi. Pre vygenerovanie objektov reprezentujúcich tabuľky databázy bol použitý firemný nástroj aj skript.

5.1 Rozdelenie systému na moduly

Systém je z hľadiska architektúry rozdelený na vrstvy, ktorým vo vývojovom prostredí projektu zhruba odpovedajú moduly. Vzájomné prepojenie týchto modulov, definície rôznych závislostí či zostavenie aplikácie[2] (ďalej len z angl. *build*) zabezpečuje nástroj *Maven*[3]. Projekt sa skladá zo štyroch modulov:

- modul *waybill-ejb* – tvorí jadro projektu, obsahuje EJB

[1] viac info na https://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html
[2] preklad + doplnkové úlohy ako napr. vykonávanie testov alebo vytváranie archívov (JAR, WAR atď.)
[3] viac info na https://maven.apache.org/what-is-maven.html

- modul *waybill-ejb-client* – tvorí klienta EJB, obsahuje vzdialené rozhrania k EJB
- modul *waybill-ws* – obsahuje implementáciu webových služieb
- a modul *waybill-ear* – spája moduly a vytvára z nich výslednú aplikáciu

Po builde aplikácie vzniká z modulov *waybill-ejb* a *waybill-ejb-client* JAR (Java ARchive), z modulu *waybill-ws* vzniká WAR (Web application ARchive) a z modulu *waybill-ear* vzniká EAR (Enterprise Application aRchive)). EAR archív sa umiestni na aplikačný server a WAR archív do webového kontajnera. JAR je formát používaný pre distribúciu Java programov. Archívy WAR a EAR sú od neho odvodené.

Modul *waybill-ejb*

Predstavuje biznis vrstvu aplikácie. Obsahuje implementáciu EJB a ich lokálne rozhrania. Sú tu implementované jednotlivé konverzie, spracovanie průvodky, odoslanie výsledkov apod. Pracuje s databázou, sú v ňom vygenerované objekty tabuliek.

Modul *waybill-ejb-client*

Tvorí klienta EJB, obsahuje vzdialené rozhrania k EJB. Je možné cez neho vystaviť aplikáciu „von“. Tvorí spojenie medzi EJB a webovou službou. Tvorí ho taký kód, ktorý je používaný mimo EJB (prípadne aj v EJB, aj na rozhraní), napr. výnimky, vymenované typy (*enum*) alebo aj modul pre validáciu, ktorý je momentálne použitý v EJB, ale v budúcnosti sa možno presunie na rozhranie.

Modul *waybill-ws*

Obsahuje implementáciu webových služieb – tvorí rozhranie aplikácie.

Modul *waybill-ear*

Tento modul neobsahuje žiadnu implementáciu, tvorí výslednú aplikáciu spojením niektorých predošlých modulov.

5.2 Implementácia rozhraní aplikácie

Systém obsahuje dve rozhrania pre spracovanie (SOAP a REST) a jedno rozhranie pre zisťovanie štatistík využitia (REST). Pri posielaní niektorých objektov z webových služieb do EJB, tj. na rôzne miesta v sieti, na odlišné stroje, je kvôli zvýšeniu výkonu využitá serializácia, resp. deserializácia, teda konverzia objektu na postupnosť bajtov a naopak.

5.2.1 SOAP rozhranie

Vstupom SOAP webovej služby je objekt `InputDataInfo` obsahujúci id průvodky a objekt `InputData` predstavujúci průvodku a vstupné dáta. Dáta aj průvodka sú dátového typu `DataHandler`. Pri SOAP webových službách ide o bežný dátový typ, ktorý poskytuje jednotné rozhranie pre dáta dostupné v rozličných formátoch či zdrojoch. Výsledkom tejto webovej služby je buď výnimka s popisom chyby, alebo odpoveď v podobe objektu `WaybillResult`, ktorý informuje o úspešnom spracovaní.

Klient volajúci túto webovú službu musí do požiadavky pribaliť HTTP hlavičky s menom a heslom. Meno a heslo predstavujú akúsi poistku, aby napr. jedno firemné oddelenie nevolalo aplikáciu nad dátami, nad ktorými by nemalo. Ak bude v budúcnosti viacero inštancií aplikácie bežať na viacerých serveroch, budú mať rovnaké rozhrania a pri nesprávnych volaniach by mohlo dôjsť k chybám.

5.2.2 REST rozhranie

Pri požiadavke na spracovanie sa metóde z REST rozhrania predá celé číslo `n`, predstavujúce požadovaný počet spracovaní. Toto číslo sa predá do EJB, ktoré spracuje prvých `n` nespracovaných záznamov v tabuľke wb. Odpoveď webovej služby je formáte JSON[4] a nesie v sebe štyri údaje: počet spracovaní z požiadavky, počet dostupných záznamov na spracovanie, počet úspešných spracovaní, stav a správa výsledku.

Pri požiadavkách na zistenie štatistík používania sa metóde z rozhrania predá dátum, prípadne časový interval a odpoveďou je reťazec s údajmi. Interval zo vstupu sa predá do EJB, kde sa hodnoty predajú do príkazu `SELECT` a odpoveďou služby je teda výsledok daného dopytu nad tabuľkou s logmi. Zatiaľ ide o čisto informačný charakter webovej služby, pokiaľ sa v budúcnosti na základe odpovede bude treba vrámci iných procesov nejako rozhodovať, reťazec sa nahradí formátom JSON.

5.2.3 Validácia vstupných dát a průvodky

Po načítaní vstupných dát z webovej služby alebo z databázy sa z nich vytvorí objekt `DataContainer`, ktorý obsahuje objekt průvodky `Waybill` a zoznam zdrojov a ďalej sa v programe pracuje už len s týmto objektom. Implementácia validácie reprezentovaná triedou `Validator` je umiestnená v module `waybill-ejb-client` a teda môže byť volaná ako v EJB, tak vo webových službách. Obsahuje jedinú metódu `validate(DataContainer dataContainer)`, ktorá volá metódy čiastkových validátorov (pre zdroje, metódy atď.). Tieto čiastkové validátory prechádzajú jednotlivé premenné (objekty) objektu `Waybill`. Okrem kontrol, či daný objekt obsahuje nejaké deklarácie, tie či obsahujú zdroje apod., sú najpodstatnejšími kontroly parametrov. Z hľadiska XML súboru průvodky ide o najzanorenejšie značky, ktoré sú pri každej deklarácii a akcii. Konštrukciou `for-each` sa prechádza zoznam parametrov pri každej deklarácii a akcii a vykonávajú sa kontroly výskytu potrebných/nadbytočných parametrov, či obsahujú prípustné hodnoty apod.

5.3 Implementácia konverzií a spracovanie výsledku

Ako už bolo spomenuté, sú použité 2 spôsoby konvertovania dokumentov: využitie externých modulov a vlastná implementácia konverzií. Každá konverzia má pre svoju implementáciu vlastnú triedu. Tieto triedy sú potomkami triedy `Transformator`, ktorá má jedinú abstraktnú metódu `transform(byte[] bytes)`. Každý potomok teda musí prekryť (*override*) túto metódu a vytvoriť vlastnú implementáciu, odpovedajúcu danej konverzii. Polymorfizmus je potom využitý pri načítavaní z hešovacej tabuľky metód (sú v nej uložené inštancie potomkov) a následnej konverzii zdroja.

[4] *JavaScript Object Notation*, viac info na http://www.json.org/

5.3.1 Využitie externých modulov

Tieto moduly sú sprístupnené vrámci firemného intranetu cez SOAP API. Na začiatku implementácie bol vygenerovaný klient webovej služby, ktorá poskytuje konverzie z XML na PDF a XLS. Triedy `TransformatorPDF` a `TransformatorXLS` (potomkovia triedy `Transformator`) v metóde `transform(byte[] bytes)` volajú metódu webovej služby pre konverziu s príslušnými parametrami z průvodky (zdroj pre konverziu v bajtoch, reťazec s názvom jrxml šablóny, reťazec s formátom výstupu – PDF/XLS atd.). Výstup metódy predstavuje výsledný PDF/XLS súbor reprezentovaný ako pole bajtov.

Po transformácii na PDF ešte nasleduje pridanie digitálne podpisu. Zdrojový kód predstavujúci implementáciu digitálneho podpisovania bol prebratý z príkladov na stránke http://developers.itextpdf.com.

5.3.2 Vlastná implementácia konverzií

Ide o XSL transformácie a vytváranie ZIP archívov s heslom. Pre vytvorenie ZIP archívu s heslom bol prebratý kód zo stránky http://javaprecursor.blogspot.cz/2013/09/how-to-make-zip-file-password-protected.html. Túto transformáciu predstavuje trieda `TransformatorZIP`.

XSL transformácie využívajú XSL šablóny uložené v module EJB v adresári `src/main/resources/META-INF/xsl`, ktorý je súčasťou výsledného JAR archívu. Pri týchto transformáciách boli použité triedy z balíku `java.xml.transform` a vychádzali z príkladov v zdroji [3]. Implemetácia tejto konverzie je v triede `TransformatorXSL`.

5.3.3 Spracovanie výsledku konverzie

Takmer všetky výstupy sa odosielajú emailom a archivujú. Toto odoslanie zabezpečuje webová služba, ktorej klient bol vygenerovaný na začiatku implementácie. Objekt, ktorý je vstupom metódy pre odoslanie sa naplní prílohami (výstupy konverzií uložené v hešovacej tabuľke) a informáciami z průvodky (odosielateľ, prijímateľ, priorita atď.). Pri odoslaní emailu prebieha automatická archivácia emailu aj s jeho prílohami do archivačného systému, takže je možné sa k nim spätne dostať.

Druhým, veľmi málo používaným spôsobom, je odoslanie SMS. Odoslanie zabezpečuje rovnaká webová služba. Nejde o spracovanie výsledkov ako také (cez SMS nepošlem PDF), ale skôr slúži na operácie ako napr. odoslanie textovej pripomienky zákazníkovi. Je implementovaná aj otestovaná validácia tohoto modulu podľa dát v průvodke. Modul je pripravený na použitie, no reálne zatiaľ použitý/otestovaný nebol, lebo SMS sú spoplatnené a zatiaľ ani neexistujú průvodky k tomuto modulu, ide o plány do budúcna.

5.3.4 Riadenia spracovania

Implementácia spracovania popísaného v sekcii 4.3 prebieha podobne ako implementácia validácie. Pre každu časť průvodky existuje vlastná trieda, ktorá konštrukciou `for-each` spracováva zoznam parametrov každej deklarácie a akcie.

5.4 Implemetácia práce s databázou

Pre vytvorenie tabuliek, vloženie a úpravu dát bol použitý databázový klient s grafickým uživateľským rozhraním, *SQuirreL*[5]. Bol použitý databázový server *Informix*[6] od IBM, ktorý používa mierne upravený jazyk SQL.

Na začiatku implementácie boli vygenerované triedy reprezentujúce tabuľky databázy. Tieto triedy su dvojakého typu: *JPA* triedy/objekty a *DTO* triedy/objekty. JPA (Java Persistence API) objekty sú používané v biznis logike, obsahujú anotácie a pracujú priamo s databázou. Pre prácu s nimi je použitý jazyk JPQL. DTO (Data Transfer Object) objekty sú obyčajné objekty bez anotácií, obsahujúce len premenné a k nim *gettery* a *settery*. Slúžia pre prácu so záznamami tabuľkami mimo aplikačného serveru, tj. napr vo webovej službe. Ku každej JPA triede existuje odpovedajúca DTO trieda. Pri práci so záznamom tabuľky, ktorý je posielaný von z EJB modulu, prípadne prichádza do EJB modulu zvonku, je nutné namapovať objekt jedného typu na druhý. Pre tento účel slúži trieda `BeanMapper`, prebratá z firmy.

5.5 Odhaľovanie a analýza chybových stavov systému

V tejto sekcii je popísané testovanie a logovanie, dva základné postupy použité pre otestovanie systému a analýzu jeho neočakávaných stavov.

5.5.1 Testovanie systému pre správu konverzií dát

Systém bol otestovaný a je pripravený na použitie. Najbližšie odhalenie nedostatkov sa očakáva po zaradení do prevádzky. Táto časť popisuje spôsoby, akými prebiehalo testovanie.

Jednotkové testy

Pre jednotkové testovanie bol použitý nástroj *JUnit*[7]. Testy boli zamerané predovšetkým na modul overovania správnosti průvodky a vstupných dát (nie je súčasťou EJB), ktorých nekonzistentnosť spôsobuje najviac problémov.

Netestovanie EJB

Testovanie EJB je vo všeobecnosti veľmi zložité a jedno z riešení napr. vyžaduje použitie softvéru tretích strán. V tomto systéme testovanie EJB nie je veľmi podstatné, keďže väčšina logiky je sprístupnená cez webové služby.

Testovanie webových služieb

Pre testovanie webových služieb bol použitý nástroj *SoapUI*. Ako je uvedené v zdroji [7], jedná sa o svetovo najpoužívanejší otvorený softvér pre testovanie SOAP a REST API.

Poskytuje jednoduché grafické užívateľské rozhranie, ktoré umožňuje nasimulovať ľubovolný testovací prípad. Je napríklad možné podľa WSDL resp. WADL súboru vygenerovať požiadavku SOAP resp. REST webovej služby, naplniť ju údajmi, spustiť a skontrolovať odpoveď služby.

[5] viac info na http://squirrel-sql.sourceforge.net/

[6] viac info na https://www.ibm.com/analytics/us/en/technology/informix/

[7] aplikačný rámec pre jednotkové testovanie pre programovací jazyk Java, viac na http://junit.org/junit4/

5.5.2 Logovanie

Pre pohodlnejšiu analýzu chybových stavov a ladenie programu je v každom module použité podrobné logovanie. Logy je možné spätne dohľadať v súboroch aplikačného serveru aj webového kontajneru. Pre logovanie bol použitý nástroj *log4j*[8]. Log4j poskytuje viaceré úrovne logovania, z ktorých boli použité tri: *info, debug, error*.

Info pre základné informácie ako začatie spracovania, dáta putujúce cez rozhranie, výsledok spracovania, *SessionDTO*. SessionDTO je objekt, ktorý jednoznačne identifikuje jeden chod programu/jedno spracovanie a slúži pre lepšiu orientáciu, dohľadanie chyby v logoch. Vzniká na začiatku spracovania volaním metódy `java.util.Date.getTime()`, ktorá vráti počet milisekúnd, ktoré ubehli od času: 1. január, 1970, 00:00:00 GMT.

Debug loguje podrobnejšie informácie ako *info*. Ide v podstate o akékoľvek údaje, ktoré môžu byť nejakým spôsobom užitočné pri hľadaní chyby, napríklad údaje o vstupoch a výstupoch väčšiny metód, stave premenných po nejakých zložitejších operáciach apod.

Error slúži pre zaznamenávanie chybových stavov. Väčšinou ide zalogovanie zachytenej výnimky s konkretizujúcim popisom.

[8] viac info na https://logging.apache.org/log4j/2.x/

Kapitola 6

Záver

Na systéme sa pracovalo nepretržite takmer štyri mesiace a počas vývoja nenastali žiadne závažné problémy. Keďže vývoj nebol ohraničený nijakým záväzným termínom dokončenia, ak aj nejaké problémy nastali, bol čas ich vo firme v kľude konzultovať a vyriešiť. Za hlavný prínos vytvoreného systému považujem jeho reálnu využiteľnosť – nahradí systém používaný v 2 veľkých českých poisťovniach, ktorý sa používa pri tisíckach tlačí denne.

Do termínu odovzdania bakalárskej práce sa bohužial nestihlo nasadiť systém do prevádzky, a to z toho dôvodu, že ešte nebol úplne dokončený zásadný modul spolupracujúci so systémom, vyvíjaný iným firemným oddelením. Aplikácia však bola otestovaná na reálnych dátach a v najbližších mesiacoch by sa mala začať používať.

Počas vývoja sa ukázalo, že v budúcnosti bude možno aplikáciu potrebné vystaviť cez iný typ rozhrania, ako bolo pôvodne plánované, a tak aplikácia poskytuje mimo pôvodne plánovaného SOAP rozhrania aj REST rozhranie. Ak by bolo v budúcnosti potrebné zvýšiť rýchlosť spracovania systému, je možné nahradiť sekvenčné spracovanie paralelným, použitím viacvláknového prístupu.

Literatúra

[1] Belqasmi, F.; Singh, J.; Melhem, S. Y. B.: SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing. *IEEE Internet Computing*, ročník 16, 2012: s. 54–46.

[2] Evans, I.: *Your First Cup: An Introduction to the Java EE Platform*. Apr 2012, [Online; navštívené 06.02.2017].
URL http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf

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

[4] Herout, P.: *Učebnice jazyka Java*. KOPP, 2013, ISBN 978-80-7232-398-2.

[5] Jendrock, E.; Cervera-Navarro, R.; Evans, I.; aj.: *The Java EE Tutorial*. Sep 2014, [Online; navštívené 09.02.2017].
URL https://docs.oracle.com/javaee/7/JEETT.pdf

[6] Khan, A.; Sum, M.: *Introducing Design Patterns in XML Schemas*. Nov 2006, [Online; navštívené 01.02.2017].
URL http://www.oracle.com/technetwork/java/design-patterns-142138.html

[7] Kolektív autorov: *SoapUI Open Source*. [Online; navštívené 20.03.2017].
URL https://www.soapui.org/open-source.html

[8] Kosek, J.: *XML schémata*. Nov 2014, [Online; navštívené 19.02.2017].
URL http://www.kosek.cz/xml/schema/index.html

[9] Tihomirovs, J.; Grabis, J.: Comparison of SOAP and REST Based Web Services Using Software Evaluation Metrics. *De Gruyter*, ročník 19, 2016: s. 92–97.

[10] Zendulka, J.; Rudolfová, I.: *Databázové systémy*. Jul 2006, [Online; navštívené 20.03.2017].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php.cs?file=%2Fcourse%2FIDS-IT%2Ftexts%2FIDS_predn.pdf&cid=10302

Prílohy

Príloha A

Průvodka

Zjednodušená průvodka bez hlavičky, menných priestorov a s niektorými vynechanými parametrami. Takto vyzerá kostra každej průvodky – má definované 2 hlavné oblasti: *deklarácie* a *akcie*. V sekcii s deklaráciami sú definované *zdroje*, ktoré sú vstupmi konverzií, *metódy*, ktoré definujú typy konverzií a *ciele*, ktoré definujú ako a kam sa výsledok odošle/uloží. Sekcia s akciami konkrétne definuje, ktoré metódy sa použijú na ktoré zdroje (konverzie), a čo sa s výsledkom konverzií urobí (kam konkrétne sa odošle).

Táto zjednodušená průvodka definuje 1 zdroj, 1 metódu a 1 ciel. V akciách je potom definované, že na zdroj *source_1*, predstavujúci súbor *file.xml*, sa má aplikovať metóda *method_1* predstavujúca konverziu na PDF a výsledok tejto konverzie sa má odoslať na ciel *target_1*, čiže emailom na danú adresu.

```
<waybill id="1">
  <declarations>
    <source id="source_1">
      <param name="name">file.xml</param>
    </source>
    <method id="method_1">
      <param name="type">PDF</param>
    </method>
    <target id="target_1">
      <param name="type">email</param>
      <param name="to"><![CDATA[<roman.nedelka@ais-servis.cz>]]></param>
    </target>
  </declarations>
  <actions>
    <convert id="convert_1">
      <param name="method">method_1</param>
      <param name="source">source_1</param>
    </convert>
    <send id="send_1">
      <param name="target">target_1</param>
      <param name="source">convert_1</param>
    </send>
  </actions>
</waybill>
```

Príloha B

XSD súbor průvodky

XSD súbor definuje, že koreňový element Waybill sa skladá z deklarácií a akcií. Deklarácie môžu obsahovať zdroje, metódy a ciele. Akcie môžu obsahovať značky pre konverzie a odoslanie. Všetky typy deklarácií a akcií môžu obsahovať zoznam parametrov.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4    targetNamespace="http://xml.ais-servis.cz/schema/waybill.xsd"
5    elementFormDefault="qualified" xmlns:tns="http://xml.ais-servis.cz/
         schema/waybill.xsd">
6
7    <xsd:complexType name="declarations">
8      <xsd:sequence>
9        <xsd:element name="source" type="tns:declaration" minOccurs="0"
             maxOccurs="unbounded"/>
10       <xsd:element name="method" type="tns:declaration" minOccurs="0"
             maxOccurs="unbounded"/>
11       <xsd:element name="target" minOccurs="0" maxOccurs="unbounded" type=
             "tns:declaration"/>
12     </xsd:sequence>
13   </xsd:complexType>
14
15   <xsd:complexType name="actions">
16     <xsd:sequence>
17       <xsd:element name="convert" minOccurs="0" maxOccurs="unbounded" type
             ="tns:action"/>
18       <xsd:element name="send" minOccurs="0" maxOccurs="unbounded" type="
             tns:action"/>
19     </xsd:sequence>
20   </xsd:complexType>
21
22   <xsd:complexType name="declaration">
23     <xsd:sequence maxOccurs="unbounded">
24       <xsd:element name="param" type="tns:param"/>
25     </xsd:sequence>
26     <xsd:attribute name="id" type="xsd:string"/>
27   </xsd:complexType>
28
29   <xsd:complexType name="action">
30     <xsd:sequence minOccurs="0" maxOccurs="unbounded">
31       <xsd:element name="param" type="tns:param"/>
32     </xsd:sequence>
```

```
33      <xsd:attribute name="id" type="xsd:string"/>
34    </xsd:complexType>
35
36    <xsd:element name="waybill">
37      <xsd:complexType>
38        <xsd:sequence>
39          <xsd:element name="declarations" type="tns:declarations"/>
40          <xsd:element name="actions" type="tns:actions"/>
41        </xsd:sequence>
42        <xsd:attribute name="id" type="xsd:string"/>
43      </xsd:complexType>
44    </xsd:element>
45
46    <xsd:complexType name="param">
47      <xsd:simpleContent>
48        <xsd:extension base="xsd:string">
49          <xsd:attribute name="name" type="xsd:string" />
50        </xsd:extension>
51      </xsd:simpleContent>
52    </xsd:complexType>
53
54  </xsd:schema>
```

Príloha C

Obsah CD

Priložené CD obsahuje nasledujúce súbory a adresáre:

- **Read.me** – readme súbor
- **manual.txt** – užívateľská príručka
- **System-pro-zpravu-konverzi.pdf** – technická správa vo formáte PDF
- **doku** – adresár so zdrojovými súbormi pre tvorbu technickej správy
- **src** – adresár so zdrojovými kódmi programu