TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 1802R022 - Informatika a logistika

Vývoj systému "Primární evidence výpočetní techniky"

System development "Primary IT evidention"

Bakalářská práce

Autor: **David Veltrubský**
Vedoucí práce: Ing. Jan Lisal
Konzultant: Tomáš Arnošt

V Liberci 1. června 2009

Poděkování

Chtěl bych poděkovat Ing. Janu Lisalovi za spolupráci a cenné rady při tvorbě mé bakalářské práce. Tomášovi Arnoštovi za spolupráci na programu a trpělivost, se kterou mi předával své zkušenosti a znalosti a v neposlední řadě pak firmě Trask solutions za zprostředkování možnosti podílet se na tomto projektu.

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **souhlasím** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

V Liberci dne 28 května 2009:

Podpis: .....

Anotace

Cílem mé bakalářské práce je vytvořit program sloužící k evidenci IT majetku středně velké firmy. Program musí umožnit sledování životního cyklu každé položky od nákupu až po vyřazení a archivaci. Důraz je kladen na robustnost sytému a paralelní práci více uživatelů v různých rolích. Nově vytvořený program má za úkol nahradit systém několika nezávislých aplikací, které v současnosti plní tyto úlohy.

Program je realizovaný v jazyce Java.

Klíčová slova:

majetková evidence, Java, Struts

Abstrakt

The goal of my project is to create a program designed for IT property accounts of a middle-sized company. This application has to keep track of an item lifecycle form theirs purchase till disposal and archiving. The emphasis is placed on stability and possibility of parallel work of several users in different roles. The newly created program is meant to replace a system of independent applications that are serving these purposes.

The application is created in Java.

Keywords:

property accounts, Java, Struts

Seznam zkratek

API **Application Programming Interface** - sada metod, dat, tříd a dalších prvků poskytovaných knihovnami či operačním systémem, která slouží k usnadnění tvorby programů.

CGI **Common Gateway Interface** - protokol umožňující propojení externí aplikace s webovým serverem.

CMDB **Configuration Management DataBase** - úložiště informací vážících se ke všem komponentám informačního systému. [10]

HTML **HyperText Markup Language** - značkovací jazyk dominující tvorbě webových stránek.

HTTP **Hypertext Transfer Protocol** - protokol aplikační vrstvy sloužící původně pro přenos hypertextových dokumentů ve formátu HTML. Zdroje, ke kterým má HTTP přistupovat, jsou identifikovány pomocí URL.

HW **Hardware**

JIT **Just In Time** - v souvislosti s jazykem Java - mezikód je místo interpretování nejprve přeložen a ihned poté spuštěn. K překladu přitom dochází až při prvním konkrétním požadavku na volání příslušné části mezikódu. [8]

JSP **Java Server Pages** - technologie pro tvorbu dynamických webových aplikací.

JVM **Java Virtual Machine** - sada počítačových programů a datových struktur, která využívá modul virtuálního stroje ke spuštění dalších počítačových programů a skriptů vytvořených v jazyce Java. Úkolem tohoto modulu je zpracovat pouze tzv. mezikód, který je v Javě označován jako Java bytecode.

LCD **Liquid Crystal Display** - zde myšleno jako jakýkoli monitor.

LDAP **Lightweight Directory Access Protocol** - TCP, client-server protokol určený k práci s adresářovými servery, jedná se o odlehčenou verzi DAP.

MVC **Model View Controller** - návrhový vzor skládající se ze tří složek (datový model, uživatelské rozhraní a vnitřní logika). Tento model je alternativou k tzv. Modelu 1 který rozlišuje pouze dvě složky - data a uživatelské rozhraní s řídící logikou.

NTB **Notebook**

SAP - v použitém kontextu myšleno jako **Enterprise Resource Planning** (ERP) firmy SAP - tedy SW který se stará o množství procesů produkční činnosti podniku.

SQL **Structured Query Language** - databázový jazyk určený k získávání a správě dat relačních databázových systémů.

SW **Software**

UML **Unified Modeling Language** - obecný grafický jazyk určený pro popis systémů.

URL Uniform Resource Locator - identifikátor sloužící k popisu přesné adresy a způsobu zpracování zdroje.

VT Výpočetní technika

XHTML eXtensible Hypertext Markup Language - striktnější a čistší obdoba HTML vycházející z HTML 4.01 a XML.

XML eXtensible Markup Language - jazyk odvozený odvozený ze SGML primárně určný pro platformově nezávislý přenos dat, sloužící také jako specifikace pro vytváření konkrétních značkovacích jazyků.

Obsah

Úvod

Cílem mé bakalářské bylo sestavit webovou aplikaci plnící úlohu primární evidence výpočetní techniky. Toto zadání vzešlo z požadavku klienta, který si takový program objednal pro svou firmu. Jedná se o velkou společnost, pro kterou bylo dále neudržitelné používat pro svou potřebu několik vzájemně nespolupracujících programů. Každý z nich udržoval záznamy jen o určitých položkách a nemohl poskytnout data potřebná pro podporu koncových uživatelů či reporting.

Jedná se tedy o evidenci stolních i přenosných počítačů a jejich periferií. Je třeba sledovat tyto položky od příchodu do firmy, v období, kdy jsou přiděleny konkrétním uživatelům, až po dobu, kdy se vrací zpět do skladu a jsou likvidovány. Dále je třeba zohlednit jednotlivé role, ve kterých budou uživatelé programu vystupovat tak, aby měl každý z nich jednoduchý a přehledný přístup k těm datům, které jsou pro něj relevantní. Na druhou stranu je nutné zabezpečit, aby uživatelé nemohli provádět operace, ke kterým nejsou oprávněni.

Samotný projekt pak probíhal v několika etapách. Nejprve bylo nutné seznámit se s problematikou a osvojit si způsob evidence v zadavatelově firmě, následně vytvořit grafický návrh aplikace a rozvrhnout její členění. V další fázi byl vytvořen datový model. Teprve tehdy bylo možné začít s vlastním vývojem jádra programu. Tato etapa byla završena testováním a dolaďováním programu, aby byla zaručena nejen jeho naprostá spolehlivost, ale také jednoduchost, jednoznačnost a intuitivnost ovládání. Završením celého procesu pak bylo předání celého projektu zákazníkovi.

1 Analytický model

Analýza programu vychází ze série setkání se zástupci klientské firmy. Tím byl vytvořen dokument a UML model jako podklad pro programátorskou činnost. Ačkoli byly tyto podklady na velmi dobré úrovni, našlo se v průběhu práce množství nejasností a nekonzistentností, které si vyžádalo další upřesňování a změny.

Pro pochopení projektu budou nastíněny některé základní body analýzy.

1.1 Požadavky

Požadavky na program jsou rozděleny na funkční (tabulka č.1) a nefunkční (tabulka č.2). Funkční požadavky stanovují, co by měl program umožňovat, definují tedy úkoly, které bude schopen řešit. Nefunkční požadavky pak stanovují především vlastnosti a omezující podmínky daného systému.

Číslo požadavku	**Popis**
FRQ1	**Všeobecné požadavky**
FRQ1.01	Nově vytvořený systém musí být centrálním evidenčním systémem pro účely evidence výpočetní techniky.
FRQ1.02	Nově vytvořený systém musí poskytovat primární evidenci koncových zařízení pro potřeby všech uživatelů a útvarů poskytujících podporu uživatelům koncových zařízení a umožňovat sledování a řízení životního cyklu HW.
	pokračování na další straně

	– pokračování z předešlé stánky
FRQ1.03	Zefektivnit práci uživatelům aplikace tzn. minimalizovat nároky na manuální zadávání dat do aplikace.
FRQ2	**Základní funkčnosti aplikace – aplikace bude nabízet následující funkčnosti:**
FRQ2.01	Vyhledání položky nebo položek výpočetní techniky na základě zadaných hodnot do filtru.
FRQ2.02	Zobrazení vyhledaných položek VT prostřednictvím tabulky.
FRQ2.03	Zobrazení detailu vybrané položky.
FRQ2.04	Import souboru s novými položkami VT.
FRQ2.05	Manuální přidávání nových položek VT.
FRQ2.06	Příjem položek na sklad.
FRQ2.07	Manuální doplnění aktuálně zjištěných HW parametrů dané výpočetní techniky.
FRQ2.08	Aktivování výpočetní techniky tj. umožnit identifikaci, zda je položka VT rezervovaná na konkrétní projekt či nikoli.
FRQ2.09	Přidělení položky VT konkrétní osobě (vlastníkovi).
FRQ2.10	Zavedení položky VT do provozu.
FRQ2.11	Sledování parametrů VT v provozu.
FRQ2.12	Sledování položek VT určené ke svozu na sklad.
FRQ2.13	Umožnit zaznamenat výměnu staré položky VT za novou.
	pokračování na další straně

	– pokračování z předešlé stánky
FRQ2.14	Dovolit zrušit přidělení VT konkrétní osobě, v případě dlouhodobého neuvedení položky VT do provozu.
FRQ2.15	Opravení čísla servisního požadavku, v případě jeho chybného zadání do aplikace.
FRQ2.16	Sledování položek VT určené k odkupu ze stolu.
FRQ2.17	Umožnit vrátit položky VT určené k odkupu ze stolu do předchozího stavu. Bude se týkat pouze těch položek VT, které nebyly do konce následujícího měsíce označeny statusem odkup v systému SAP.
FRQ2.18	Archivování položek VT.
FRQ2.19	Reklasifikování položek VT.
FRQ2.20	Sledování položek VT k opravě.
FRQ2.21	Zaznamenání provedených oprav.
FRQ2.22	Přidělení položky na likvidaci.
FRQ2.23	Sledování počtu SW licencí.
FRQ2.24	Sledování VT v jednotlivých stavech.
FRQ2.25	Namapování uživatele na roli.
FRQ2.26	Umožni návrat položky VT ze stavu archivace do klasifikace.
FRQ2.27	Sledování neopravitelných položek VT.

Tabulka 1: Funkční požadavky

Číslo požadavku	Popis
NRQ1	**Databáze a data**
NRQ1.01	Databáze nově vytvořeného systému musí být SQL Server 2005.
NRQ1.02	Nově vytvořený systém musí poskytovat data pro centrální CMDB.
NRQ1.03	Nově vytvořený systém musí počítat s následujícími objemy ukládaných dat: • evidence PC/NTB do cca 30000 záznamů • evidence komponent/LCD do cca 50000 záznamů • evidence tiskáren/tiskových front do cca 10000 záznamů
NRQ2	**Uživatelé (Autentifikace uživatelů, správa rolí a kapacita)**
NRQ2.01	Autentifikace uživatelů a správa jejich rolí bude probíhat ve formě komunikace se službou ActiveDirectory.
NRQ2.02	Nově vytvořený systém musí být dimenzován na 50 uživatelů, přičemž současně přistupujících uživatelů bude cca 10.
NRQ3	**Uživatelské rozhraní, konektory**
pokračování na další straně	

	– pokračování z předešlé stánky
NRQ3.01	Nově vytvořený systém musí mít uživatelské rozhraní dostupné přes web, jako klientský prohlížeč bude použit IE verze 6 a vyšší.
NRQ3.02	Nově vytvořený systém bude obsahovat 3 konektory pro komunikaci s ostatními systémy. Dva z těchto konektorů budou tzv. online konektory, třetí konektor bude pracovat dávkově.
NRQ4	**Migrace**
NRQ4.01	Bude zabezpečena migrace dat ze stávajících systémů. Zadavatel – připraví „čistá“ data. Vykonovatel – provede migraci a případné odstranění duplicit.

Tabulka 2: Nefunkčních požadavky

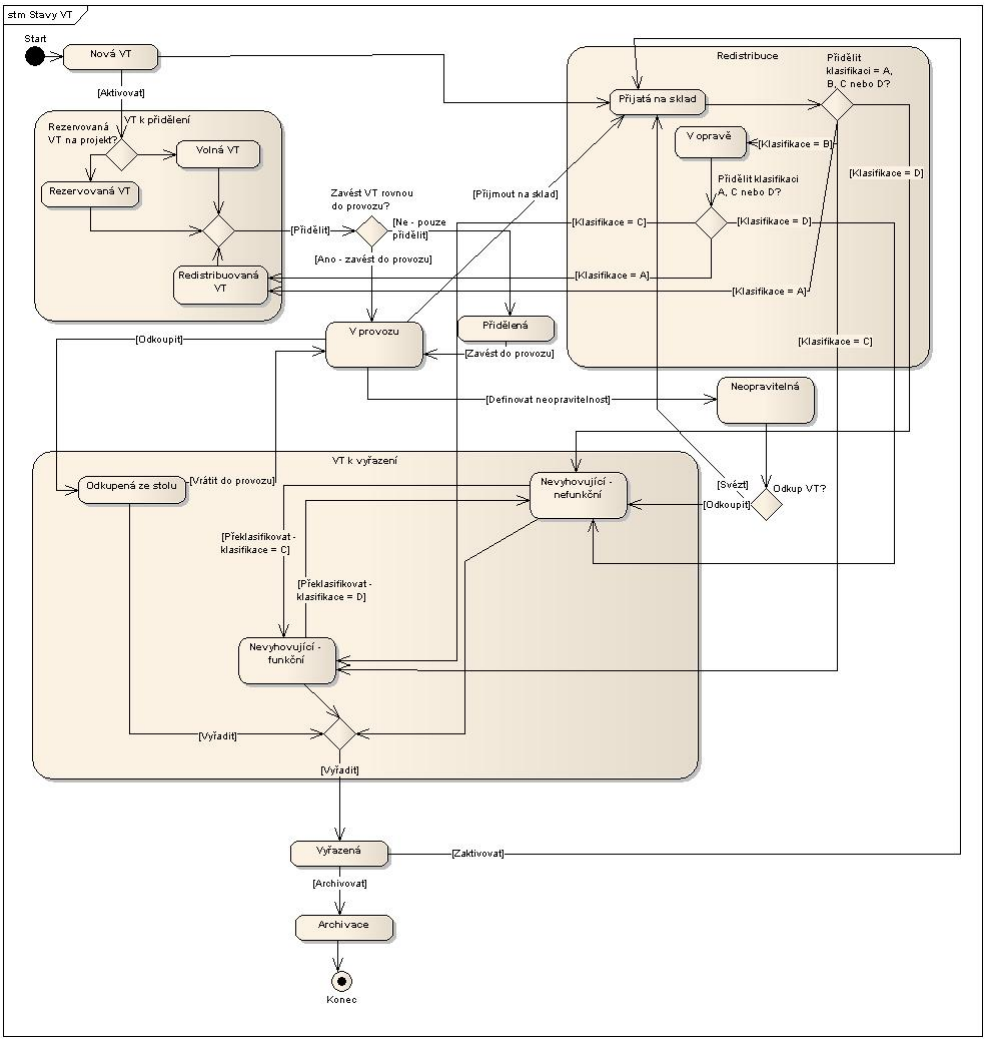
1.2 Životní cyklus

Životní cyklus zachycuje stavy, ve kterých se jednotlivé položky mohou vyskytovat v průběhu jejich existence v podniku. Stavy jsou dány jednak tím, zda je položka na skladě, či zda je přidělělena konkrétní osobě a úkolu. A dále mohou nést informaci o tom, v jaké kondici je konkrétní položka, či jaké je její nejpravděpodobnější užití v budoucnu.

Jednotlivé stavy jsou následující:

1. Nová VT
2. VT k přidělení
3. Přidělená
4. V Provozu
5. Neopravitelná
6. Redistribuce
7. VT k vyřazení
8. Vyřazená
9. Archivace

Podstavy a přechody mezi stavy ilustruje obrázek 1



Obrázek 1: Životní cyklus evidovaných položek

1.3 Uživatelé a role

Vzhledem k velikosti a požadavkům zadavatelské firmy bylo potřeba rozdělit funkčnost aplikace podle jednotlivých rolí, které do ní budou mít přístup. Přiřazení jednotlivých rolí na konkrétní uživatele zajišťuje systém LDAP. V evidenčním programu se uživatelé budou vyskytovat v rolích shrnutých do tabulky č. 3.

Role	**Popis**
Outsourcer	Pracovník skladu, který provádí příjem nové nebo svezené výpočetní techniky na sklad a výdej výpočetní techniky na likvidaci. Dále provádí klasifikaci techniky při příjmu na sklad.
Distributor	Přiděluje konkrétní položku VT určité osobě a ve výjimečných případech ji zavádí přímo do provozu.
EUE	Aktivuje nově zavedené položky VT do systému,.
Redistributor	Pracovník, který provádí příjem VT na svůj sklad, reklasifikaci položek VT, sleduje a edituje VT v provozu, definuje neopravitelnost položky.
User IT	Pracovník, který pouze sleduje VT v provozu.
Help Desk	Pracovník, který pouze sleduje některé údaje VT v provozu. Schvalovatel odkupu Pracovník, který se stará o odkup výpočetní techniky ze stolu.
Opravář	Zodpovídá za opravu výpočetní techniky.
Likvidátor	Provádí likvidaci výpočetní techniky. Definuje, zda u dané výpočetní techniky bude provedena likvidace fyzická nebo likvidace odkupem.

pokračování na další straně

	– pokračování z předešlé stánky
Admin-system	Pracovník, který provádí správu aplikace.
Admin-user	Pracovník, který provádí správu uživatelů aplikace.
Správce	licencí Pracovník, který provádí správu licencí.

Tabulka 3: Role uživatelů v systému

2 Užité nástroje

2.1 Java

Pojem Java je možné chápat ze dvou hledisek. Jednak jako programovací jazyk, nebo jako platformu.

Programovací jazyk - v tomto kontextu je Java chápána jako objektově orientovaný programovací jazyk, vytvořený Jamesem Goslingem (Sun Microsystems, 1995). Původní název toho projektu zněl Oak a primárním zaměřením byl určen pro programování domácích spotřebičů. V době, kdy byl dokončen, se však nesetkal s valným komerčním úspěchem. Teprve po dalších několika letech vývoje a přispění dalších autorů byl přetvořen do podoby, jakou známe dnes a cílem jeho využití se stal především internet. V té době již na trhu byl jiný programovací jazyk stejného jména a proto došlo ke změně v názvu. Java vychází z jazyka C++, který také výrazně připomíná svou syntaxí. Společnost Sun definuje programovací jazyk Java jako jednoduchý, objektově orientovaný, distribuovaný, robustní, bezpečný, na architektuře nezávislý, portabilní, interpretovaný, vysoce výkonný a vícevláknový. [9]

Od jazyka C++ Javu odlišuje několik specifických rysů. Mezi ty nejvýznamější patří:

- silná typová kontrola
- odklon od používání ukazatelů
- odstranění vícenásobné dědičnosti
- absence příkazu goto
- absence globálních proměnných a funkcí
- garbage collector

Veškerý zdrojový kód jayka Java je nejprve uložen do textového souboru s příponou *.java*. Kompilátor *javac* pak tyto zdrojové kódy kompiluje do souborů *.class*. Ty obsahují tzv. bajtový kód (bytecode). Jedná se o zdrojový kód virtuálního stroje Java Virtual Machine (JVM). Teprve JVM je specifický pro různé platformy (kombinace hardwaru a operačních sytémů). Stejný soubor *.class* je tedy možné spustit v MS Windows, Linuxu, Solarisu a dalších operačních systémech.

Platforma Java - jak již bylo uvedeno termín platforma je chápán jako prostředí, umožňující spouštět programy. Většinou je možné ji popsat jako kombinaci HW a SW prostředí. Jednou z výjimek je právě platforma Java. Jedná se o čistě softwarové prostředí, které funguje nad jinými, HW závislými platformami.[11] Zahrnuje dvě hlavní komponenty:

- JVM - java virtual machine
- API - application programing interface

API je rozsáhlou sbírkou hotových SW komponent, členěných do knihoven (balíčků) s příslušnými třídami a rozhraními.

Jasnou nevýhodou, která plyne právě z požadavku na přenositelnost bajtového kódu, je pomalejší běh programu oproti například C++, které své programy rovnou kompiluje do zdrojového kódu závislého na platfomě. Java byla ve svých počátcích čistě interpretovaný jazyk, v průběhu vývoje se však stále víc uplatňuje JIT kompilace. Za běhu se kompilují nejčastěji používané části kódu, jako jsou například cykly.

2.2 Servlety a Java Server Pages

Servlety - původní návrh webových stránek nepočítal s dynamickým obsahem. Aby byl tento nedostatek odstraněn, vznikly různé technologie. Jako první se objevilo CGI (Common Gateway Interface). Jedná se o standard, který umožňuje propojení webových stránek a dalších aplikací. Toto řešení ovšem není zdaleka dokonalé. Potýká se s velkou systémovou náročností, protože pro každý požadavek vzniká nové vlákno a s ním je spojená režie. Další nevýhodou je obtížnost s napojením v různých fázích požadavku.

V zájmu zefektivnění tvorby a běhu dynamických stránek tak vznikaly nové technologie. Jednou z nich jsou servlety. Jedná se o třídy napsané v jazyce Java, které oproti CGI poskytují vyšší rychlost zpracování a nižší systémové nároky, neboť vytvářejí jen jediný proces. Servlety nemohou běžet samostatně na webovém serveru, pro svou práci potřebují prostředí vytvářené webovými kontejnery, jako je například Jakarta Tomcat. Servlety jsou tedy součástí webových aplikací, kde vedle nich nalezneme HTML stránky, obrázky, multimediální obsah, XML a další. Je-li webová aplikace nasazena, kontejner vytvoří a nahraje instance servletů, aby mohl pružně odpovídat na klientské požadavky.

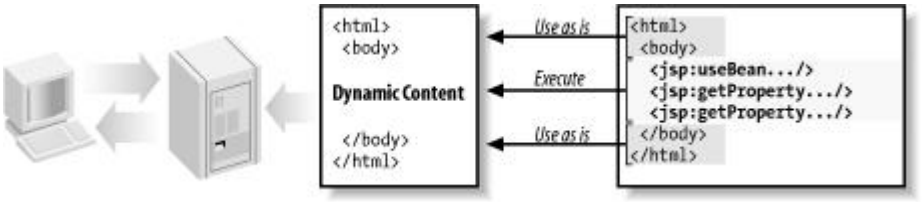
Jednou z hlavních nevýhod servletů je však náročnost na jejich údržbu. I pro drobnou změnu uživatelského prostředí je potřeba, aby vývojář se znalostí javy upravoval kód servletu, znovu ho zkompiloval a nasadil.

Co je to JSP - Java Server Pages - je řešením výše zmiňovaného problému. Dalo by se říct, že JSP je přístup z druhé strany.[7] Místo toho, aby byly HTML tagy generovány v Java kódu, je javovský kód v podobě tzv. scripletů zakomponován do HTML. Zjednodušeně, JSP je technologie pro tvorbu dynamických webových aplikací vzniklá rozšířením servletů. Na rozdíl od prostých HTML stránek, které zůstávají stále stejné, obsah JSP se může měnit na základě mnoha různých podnětů, ať už se jedná o identitu uživatele,

prohlížeče, či výběru, který uživatel provede.[3] JSP vzniká spojením statických HTML tagů, XML značek a scripletů, sloužících právě ke správě a uchovávání dynamických informací.

JSP tedy funguje jako interface. Vývojář vytvoří textový soubor obsahující HTML, XML, XHTML a JSP tagy, které jsou pak JSP kompilátorem (např. Jaspr v Tomcatu) automaticky transformovány do podoby servletu. Vývojář se tímto nově vzniklým kódem nemusí nikdy zabývat. K tomutu překladu dochází při prvním volání a při každém dalším je již používán zkompilovaný servlet.

JSP podporují jak skriptování, tak i přístup k plnohodnotné Javě, nejsu interpretovány, ale kompilovány, čímž zvyšují efektivitu a díky svému javovskému základu sdílejí platformovou nezávislost. Funkci JSP vystihuje obrázek 2, převzatý z [3].



Obrázek 2: Generování dynamického obsahu pomocí JSP

2.3 MVC a Jakarta Struts

JSP umožňuje v zásadě dva druhy přístupu k webovým stránkám. Podle prvního řídí zpracování celého požadavku samotná JSP stránka. Požadavek je zasílán přímo na JSP, ta může spolupracovat s např. JavaBeany, nebo jiným zařízením a nakonec vybere další stránku, na kterou klienta přesměruje. Naproti tomu druhý model počítá s funkcí tzv. řadiče, tedy servletu, který požadavek klienta zachytí, zpracuje a vybere, která JSP bude zobrazena. Ačkoli se tento postup může zdát složitější, výrazně zjednodušuje práci například s autorizací uživatelů či internacionalizací. V tomto modelu je poměrně jasně oddělena vnitřní logika, zpracování vnější formální stránky a zpracování požadavku. Podle toho se nazývá Model-View-Controller (MVC). [4]

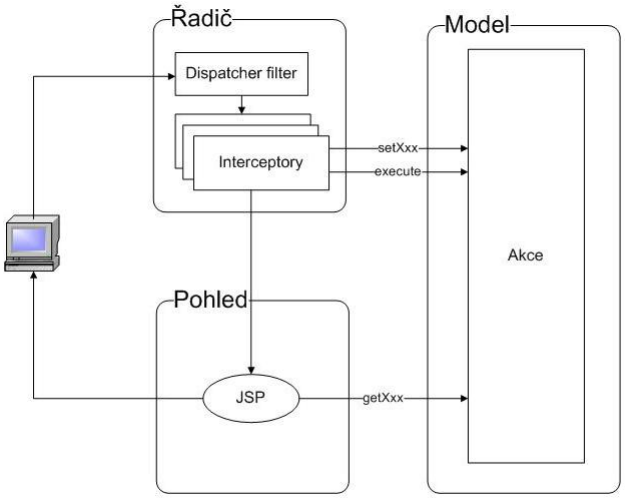
- Model - zodpovídá za správu vnitřní struktury, mohou to být např. běžné objekty jazyka Java spravující data získávaná z databáze
- View - pohled - pohledy bývají HTML stránky nebo JSP. Podle toho vytvářejí statický nebo dynamický náhled na data modelu. Díky tomuto oddělení pohledu od modelu je možné, aby jeden model podporoval několik pohledů (např. pro každý typ klienta) a přitom používal tytéž komponenty.
- Controller - Řadič - bývá obecně servlet, který zpracovává požadavky klienta. Může zajistit jejich pre- a post-processing. Dále je překládá na konkrétní operace a nakonec pomáhá zvolit následný pohled, který vrací klientovi.

Struts je open-source rámec, usnadňující vytváření webových aplikací založených právě na takovémto základě. Tento rámec poskytuje tři klíčové prvky [1]:

- ovladač požadavků - poskytovaný vývojářem aplikace, namapovaný na standartní URI

- ovladač odpovědí - který předá kontrolu dalšímu zdroji schopnému zodpovědět požadavek
- knihovnu tagů - nabízející snadnější a pohodlnější vytváření webových aplikací založených na formulářích

Abychom byli ještě přesnější, Struts je MVC webový rámec založený na akcích. Jeho architekturu ilustruje obrázek 3. Kombinuje tedy servlety a JSP tak, aby každá z těchto komponent dělala to, k čemu je nejvhodnější. V tomto podání je servlet v roli řadiče zajišťujícího centralizovaný bod, který kontroluje všechny klientské požadavky. Jednotlivé URL požadavky mapuje na tzv. akce. Úlohou akcí je pak vykonat specifickou službu pro požadované URL s přístupem k HTTP session, HTTP request a parametrů formuláře. Výsledek je nakonec akcí vrácen přes konfigurační soubory do náležitého JSP.



Obrázek 3: MVC/Struts architektura

3 Implementace

Vytvořený program je webovou aplikací s velice snadným ovládáním. Nevyžaduje téměř žádné zvláštní znalosti, uživatel si vystačí se znalostí problematiky své profese.

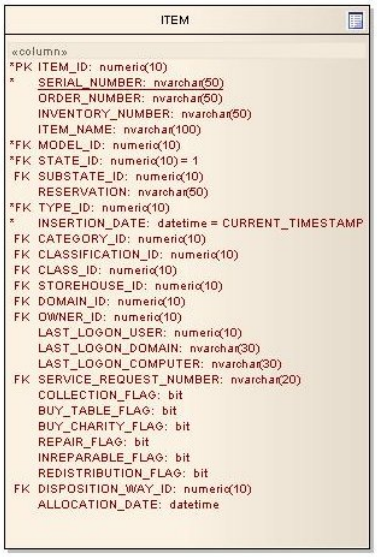
Po spuštění aplikace je uživatel dotázán na své vstupní údaje, podle kterých je mu přidělena role v programu. Následně jsou mu v hlavním menu zobrazeny pouze ty funkce, ke kterým má jeho role přístup. Toho je dosaženo vložením podmínek přímo do servletové stránky menu.

Jednotlivé stránky (funkce programu) jsou vytvořeny pomocí programu Tiles tak, aby bylo možné užít jednotnou šablonu. Odpadá tak nutnost kopírovat hlavičku a patičku každé z nich, stejně jako základní formátování stran.

Celá koncepce členění programu je podřízena rozdělením funkcí v základním menu.

3.1 Datový model

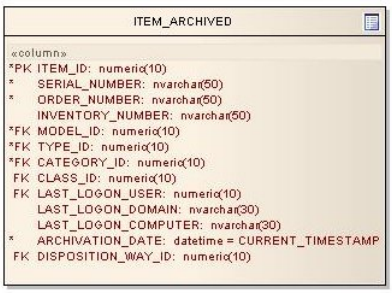
Tabulka ITEM - primárním zdrojem dat pro celý projekt je tabulka ITEM reprezentující jednotlivé kusy HW. Pro účely jednoznačné identifikace položek v databázi slouží primární klíč ITEM_ID a každý prvek je navíc jednoznačně definován svým sériovým číslem. Ostatní sloupce tabulky jsou buď cizí klíče do ostatních tabulek (TYPE_ID, CLASS_ID ...), údaje popisují konkrétní prvek, ale nevyžadují unikátnost v rámci celé databáze (INVENTORY_NUMBER, ORDER_NUMBER ...), nebo se jedná o boolovské příznaky (REPAIR_FLAG, COLLECTION_FLAG). Tabulku ilustruje obázek 4.



Obrázek 4: Tabulka ITEM

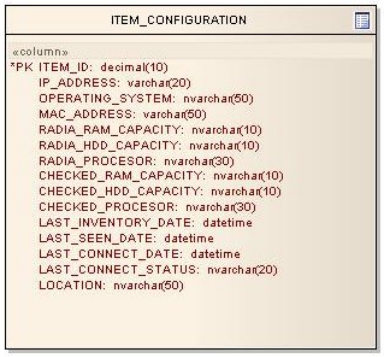
Tabulka ITEM_ARCHIVED - na konci životního cyklu přechazejí položky evidence do stavu *archivovaná*. Právě pro tento účel slouží tabulka, která je výstižně pojmenovaná, ITEM_ARCHIVED. Oproti předchozí postrádá veškeré příznakové sloupce, stejně jako některé cizí klíče k tabulkám, které už nejsou relevantní. Navíc je zde atribut ARCHIVATION_DATE,

který, v souladu se svým názvem, obsahuje datum, kdy byla položka archivována. Tabulku ilustruje obázek 5.



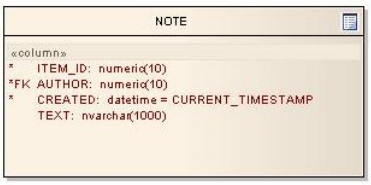
Obrázek 5: Tabulka ITEM_ARCHIVED

Tabulka ITEM_CONFIGURATION - k tabulce ITEM je připojena pomocí klíče ITEM_ID vazbou 1:0..1. Obsahuje data o konfiguraci jednotlivých prvků jednak z automatického reportu programu RADIA a jednak manuálně zadávaná uživatelem ve vhodné roli. Tabulku ilustruje obázek 6.



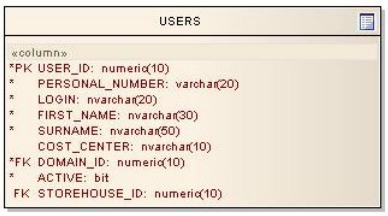
Obrázek 6: Tabulka ITEM_CONFIGURATION

Tabulka NOTE - slouží k uchování poznámek o položkách. S tabulkou ITEM je svázána vazbou 1:0..n klíčem ITEM_ID. obsahuje id autora, text poznámky a čas vytvoření. Tabulku ilustruje obázek 7.



Obrázek 7: Tabulka NOTE

Tabulka USER - reprezentuje uživatele programu, nebo vlastníka položky v evidenci. S tabulkou ITEM je svázána vazbou 1:0..1 klíčem USER_ID. Tabulku ilustruje obázek 8.



Obrázek 8: Tabulka USER

Popisné tabulky - tyto tabulky obsahují id popisovaného prvku, jeho název a popis, popřípadě jiný doplňující údaj. Slouží pro větší efektivitu práce (v databázi by bylo zbytečné tolikrát uchovávat stejná data). Jejich strukturu zachycuje obrázek 9.

Obrázek 9: Popisné tabulky datového modelu

3.2 SQL

Pro uchování údajů v evidenci stejně jako ostatní data potřebná k chodu aplikace je podle zadání od klienta využíván SQLServer2005. V původním návrhu programu bylo počítáno s jedinou třídou, která by zahrnovala všechny možné SQL dotazy pro aplikaci. Po vytvoření prvních několika formulářů však bylo jasné, že by tato třída obsahovala takové množství kódu, kter by ji učinilo zcela nepřehlednou a výrazně tak ztížilo dodatečnou údržbu programu.

Proto nejprve vznikla třída *BaseDatabean*, která zastřešuje ostatní a vytváří neměnnou část dotazu. Od ní byly následně odvozeny podtřídy, které sestavují specifickou část dotazu. Jednotlivé třídy *DataBean* (potomky třídy BaseDataBean), je možné rozdělit do dvou částí. První obsahuje seznam konstant typu string, které jsou tělem SQL dotazu a jsou známy před spuštěním programu. Druhou částí jsou metody, které spojí celý dotaz do jediného řetězce a doplní o údaje odeslané z formuláře.

Pro názornější představu a snazší pochopení funkce uvedu jednu metodu ze třídy *DisposeDataBean* :

```
public PaginatedListImpl<ItemDto> getNotSatisfyingItems (PaginatedListImpl<ItemDto>
 paginatedList, long itemState, long ClassificationId)
{

  String sql = super.createPageableSqlQuery( ItemDto.SORT_COLUMNS,
   ItemDto.DEFAULT_SORT_COLUMN, paginatedList,
   SELECT_NON_SATISFYING_ITEMS );

  Object[] basicSqlParameters = new Object[] {itemState, ClassificationId};
  Object[] allSqlparameters = super.createParametersArray( paginatedList,
   basicSqlParameters, true);

  List<ItemDto> items =  this.mainJdbcTemplate.query ( sql, ItemDto.rowMapper,
   allSqlparameters);
```

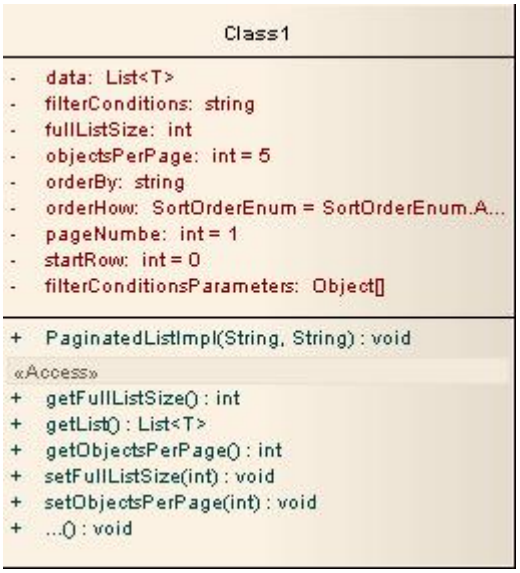
```
    int fullListSize = this.mainJdbcTemplate.queryForInt(
      super.createFullListCountSqlQuery( SELECT_COUNT_NON_SATISFYING_ITEMS, paginatedList),
      super.createParametersArray(paginatedList, basicSqlParameters, false ) );

    paginatedList.setList( items );
    paginatedList.setFullListSize(fullListSize);

    return paginatedList;
  }
```

Tato metoda nejprve volá metody své nadtřídy s parametry, které dostane z formuláře. Z příslušného konstantního řetězce následně vytvoří pole objektů s užitím parametrů, které dostala při svém volání. Poté doplní údaje o filtrech a stránkování a ty společně s údaji o řádcích tabulky předá metodě *mainJdbcTemplate.query*. Nakonec pošle dotaz na počet nalezených prvků, který vrátí jako svůj výsledek.

3.3 JSP a akce

Akce jsou srdcem celé aplikace. V programu jsou rozvrženy tak, aby se kryly s jednotlivými příklady užití v analýze. Skoro každá z akcí zobrazuje nebo upravuje objekt představující tabulku položek z evidence. Bylo tedy vhodné, aby metody určené k práci s tímto objektem byly zahrnuty do rodičovské třídy, od které bude možné odvozovat potomky pro konkrétní úlohy. Touto třídou je *ActionWithTableBaseImpl*, která je sama potomkem třídy *ActionSupport* a rozhraní *ActionWithTable*

Jak již bylo zmíněno, jednotlivé akce pracují s tabulkou položek evidence. Tato tabulka je v programu zahrnuta do třídy *PaginatedListImpl*. Ta obsahuje samotnou tabulku reprezentovanou typem list a rozšiřuje jí o další stavové proměnné, jako je počet zobrazených prvků tabulky, sloupec dle kterého je seřazená, směr seřazení, podmínky, dle kterých byla vybrána, etc. UML reprezentace této třídy je na obrázku 10.



Obrázek 10: třída PaginatedListImpl

Atribut *data* (v aplikaci představuje obsah tabulek v jednotlivých formulářích), definovaný jako list, je nejčastěji tvořen seznamem datových objektů popisujících evidované položky. Koresponduje tedy s tabulkou ITEM z datového modelu. Tento objekt obsahuje pouze data a s nimi spojené accessory. Daty jsou buď jednoduché typy nebo další datové objekty, které opět odpovídají jednotlivým tabulkám.

Detailně se zabývat jednotlivými akcemi by bylo zbytečné, neboť jsou si v mnohém podobné. Jako příklad uvedu a více rozeberu jednu z nich. U ostatních pak jen nastíním jejich úkol, případně uvedu jejich specifika.

3.4 Reklasifikace položky

Úkolem třídy *ClassifyItemAction* je zobrazit prvky z databáze, jimž byla přidělena klasifikace a umožnit její změnu, pakliže ještě nemají jeden z příznaků, který by změně klasifikace bránil.

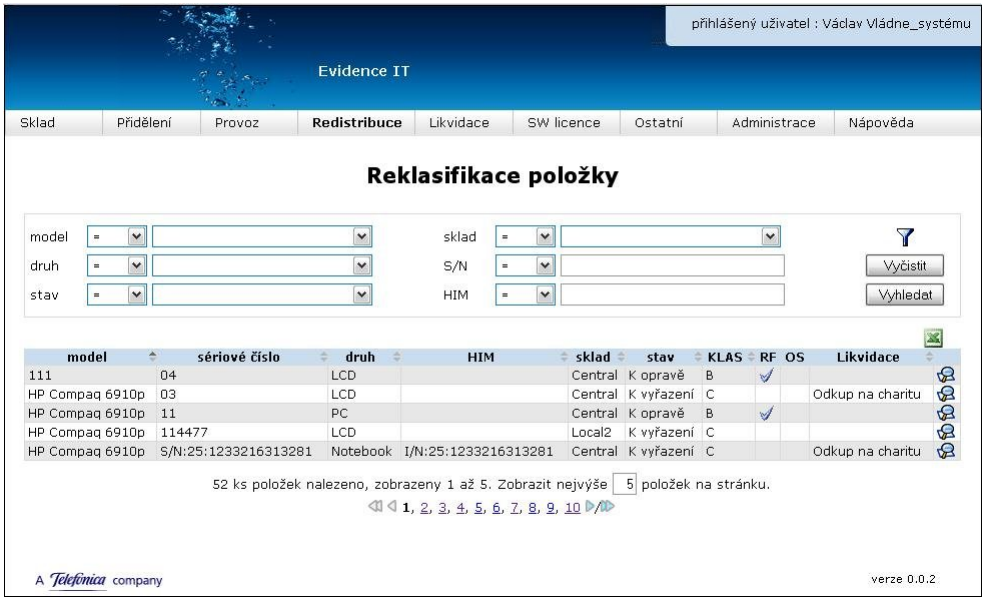
Na začátku je třída inicializována metodou *init()* zděděnou po rodičovské třídě *ActionWithTableBaseImpl*. Ta volá několik abstraktních metod, které

jsou unikátní pro každou akci. První dvě, *getDialogName()* a *getDefaultOrderBy()* dodávají potřebné údaje v podobě textových řetězců konstruktoru třídy *PaginatedListImpl*. Třetí metoda - *readListData()* naplní objekt *PaginatedListImpl* výsledkem vhodného SQL dotazu. Když tak učiní, uloží metoda *init()* objekt *PaginatedListImpl* do HTTP session, aby byl později přístupný i bez opětovného SQL dotazu. Výsledkem celé metody je řetězec „SUCCESS“, který slouží jako identifikátor pro rámec Struts. Ten podle něj zobrazí korespondující JSP. V našem případě jde o *classifyItem.jsp*, což je specifikováno na začátku metody v anotaci. Metoda *init()* třídy *ActionWithTableBaseImpl* a metoda *readListdata()* třídy *ClassifyItemAction* jsou obsahem následujícího příkladu.

```
public String init()
{
  if ( logger.isDebugEnabled() )
  {
    logger.debug( this.getClass().getName() + ".init() started" );
  }

  HttpServletRequest request = ServletActionContext.getRequest();
  HttpSession        session = request.getSession();

  PaginatedListImpl<T> paginatedList = new PaginatedListImpl<T>( this.getDialogName(),
    this.getDefaultOrderBy() );

  paginatedList = this.readListData(paginatedList);

  session.setAttribute(SessionAttributes.PAGINATED_LIST, paginatedList);

  session.setAttribute(SessionAttributes.FILTER_FIELDS_VALUES_MAP,
   new HashMap<FilterField, FilterFieldValueDto>());

  return SUCCESS;
}

public PaginatedListImpl<ItemDto> readListData(PaginatedListImpl<ItemDto> paginatedList)
{
  this.allowedItemStates = (List<ItemStateDto>)
```

```
    this.persistentData.getDemandedItemStates(
    new ItemState[] {ItemState.TO_ASSIGN, ItemState.TO_DISCARD,
    ItemState.REDISTRIBUTION, ItemState.FOR_REPAIRS} );
   return this.dataBean.getItemsToReclassify(paginatedList);
}
```

Výsledné JSP ukazuje obrázek 11.



Obrázek 11: výsledek JSP reklasifikace

Uživatel má nyní možnost provést svou volbu. Pokud si chce například zobrazit detail sledované položky, stiskne symbol v tabulce. Ten nese jako svůj parametr id příslušné položky. Rámec Struts takový požadavek odchytí a volá metodu *showItemDetail()*, která je v následujícím příkladě. Ta z HTTP requestu vybere parametr *item_id* a z HTTP session *paginatedList* (evidencni tabulka). V té podle item_id najde správný řádek, pro nějž chceme zobrazit detail. Alternativou by bylo podle parametru *item_id* najít potřebné údaje v databázi, ale v tom případě by bylo nutné vytvořit nový dotaz a zdržovat se jeho vyřizováním. Nahlédnutí do databáze se však stejně nevyhneme, protože je třeba, krom již získaných dat, nahrát ještě informace o konfig-

uraci zvolené položky a poznámky, které se k ní vážou. Tyto informace jsou uloženy v externích tabulkách a nemusí vždy být dostupné. Proto je SQL dotaz v bloku try-catch. Nakonec jsou získané informace sloučeny do jednoho objektu a metoda vrátí řetězec *„detail“* Podle této návratové hodnoty určí aplikace Struts, že má být použito JSP zobrazující detail položky.

```
public String showItemDetail ()
{

  HttpServletRequest request = ServletActionContext.getRequest();
  HttpSession        session = request.getSession();

  itemId = Long.valueOf( request.getParameter( RequestAttributes.ITEM_ID ) );
  PaginatedListImpl<ItemDto> paginatedList = (PaginatedListImpl<ItemDto>)
    session.getAttribute(SessionAttributes.PAGINATED_LIST);

  List<ItemDto> items = new ArrayList<ItemDto>(paginatedList.getList());

  int i = 0;
  while (items.get(i).getItemId().longValue() != itemId.longValue())
  {
    i++;
  }
  shownItem = items.get(i);

  try
  {
    itemConfiguration = dataBean.getItemConfigurationByItemId(shownItem.getItemId());
  }
  catch (Exception e)
  {    }
  shownItem.setItemConfiguration(itemConfiguration);
  notes = dataBean.getItemNotes(itemId);

  return "detail";
}
```

Toto JSP je především informativní. Navíc, pokud položka ještě nebyla označena k prodeji nebo k opravě, umožní uživateli (pokud na to má jeho role

právo) změnit klasifikaci. Pokud tak uživatel učiní a stiskne tlačítko „Reklasifikovat.“ Spustí tím metodu *itemReclasified()*. Ta nejprve zjistí, zda byla skutečně zadána nová hodnota klasifikace. Pokud ne, přidá do pole vyhrazeného chybovým hláškám varování o tomto problému a vrátí řetězec *„detail“* Struts znovu zobrazí JSP s detailem. Pokud je klasifikace vyplněna, systém podle ní změní stav položky a výsledek uloží včetně nové poznámky (pokud uživatel nějakou vytvořil). Následně vloží do pole určeného informativním hláškám zprávu o úspěšné změně údajů. Nakonec volá metodu *initNext()*. Ta má podobnou úlohu jako metoda *init()*. Na rozdíl od ní ale vybírá data z HTTP session místo toho, aby je brala z databáze. Vrací opět řetězec *„SUCCESS“*, Struts tedy zobrazí původní JSP s tabulkou evidovaných položek. Metoda itemReclasified() je obsahem následujícího příkladu.

```
public String itemReclassified ()
{
  HttpServletRequest request = ServletActionContext.getRequest();

  if ( this.newClassification == null )
  {
    addFieldError("newClassification", CaptionUtil.getCaption(
     "validation.classification.required", request.getLocale()));
    return "detail";
  }
  Long newItemSubState = null;
  if (newClassification.longValue() == Classification.A.classificationId())
  {
    itemState = ItemState.TO_ASSIGN.stateId();
    newItemSubState = ItemSubstate.REDISTRIBUTED.substateId();
  }
  else if (newClassification.longValue() == Classification.B.classificationId())
    itemState = ItemState.FOR_REPAIRS.stateId();
  else itemState = ItemState.TO_DISCARD.stateId();

  dataBean.updateItemStateAndClassification(itemState, newItemSubState,
   newClassification, itemId);
  UserDto currentUser = (UserDto)request.getUserPrincipal();
  dataBean.insertNewNote(itemId, currentUser.getUserId(), newNote);
```

```
    addActionMessage(CaptionUtil.getCaption("message.changesSaved", request.getLocale()));

    return this.initNext();
  }
```

JSP zobrazující klasifikované položky využívá krom rámce Struts ještě Tiles pro vnořování často používaných částí a DisplayTag usnadňující tvorbu tabulek. Takto jsou naimportovány funkce pro zobrazování zpráv a varovných hlášek z rámce Struts a funkce zajišťující korektní chování ukazatele počtu zobrazených položek v tabulce. Dále je naimportována část menu, nadpis a úsek zobrazující filtry. Ten je sice unikátní pro každý formulář, ale jeho separace do samostatného souboru přispěla k přehlednosti kódu. Dále následuje formulář se zobrazením tabulky. Zde už bylo nutné, vzhledem k nutnosti užít grafické vyjádření boolovské hodnoty, vložit kód v Javě. Posledním sloupec tabulky obsahuje link na detail položky realizovaný pomocí přidaného parametru obsahující id položky. V samostatném formuláři je uchovávaný objekt potřebný ke změně počtu zobrazených položek v tabulce. Vygenerování tabulky je díky knihovně Display Tag snadné a přehledné, viz následující příklad:

```
  <display:table uid="itemDtoTable" name="data" class="resultTable"
    requestURI="classifyItem!displayTagService.action" excludedParams="*">
     <%
    String itemId = null;
     boolean RF = false, BTF =  false;
    if (itemDtoTable !=  null ) {
 itemId   = ((ItemDto) itemDtoTable ).getItemId().toString();
 RF   = ((ItemDto) itemDtoTable ).getRepairFlag().booleanValue();
 BTF   = ((ItemDto) itemDtoTable ).getBuyTableFlag().booleanValue();
    }
%>
<display:column titleKey="table.columnHeader.model"
property="itemModel.itemModelName"   sortable="true" />
     <display:column titleKey="table.columnHeader.serialNumber"
     property="serialNumber"              sortable="true" />
        <display:column titleKey="table.columnHeader.itemType"
         property="itemType.typeName"         sortable="true" />
        <display:column titleKey="table.columnHeader.inventoryNumber"
         property="inventoryNumber"        sortable="true" />
        <display:column titleKey="table.columnHeader.storehouse"
         property="storehouse.storehouseName" sortable="true" />
     <display:column titleKey="table.columnHeader.itemState"
     property="itemState.stateName"       sortable="true" />
     <display:column titleKey="table.columnHeader.classification"
     property="classification.classificationName" sortable="true" />
<display:column titleKey="table.columnHeader.repairFlag">
<%if (RF)  { %><img src="img/checkmark_small.GIF" />
<%} else {} %> </display:column>
<display:column titleKey="table.columnHeader.buyTableFlag">
<%if (BTF)  { %><img src="img/checkmark_small.GIF" />
<%} else {} %> </display:column>
<display:column titleKey="table.columnHeader.dispositionWay"
    property="dispositionWay.dispositionWayName" sortable="true" />
  <display:column  >
  <a href="<s:url action="classifyItem!showItemDetail.action"
  includeParams="none" /><%= "?itemId=" + itemId %>" >
         <img src="img/symbol_detail_small.gif "
         title="<s:text name="label.clasify.detail"/>" /></a>
</display:column>
 </display:table>
```

JSP zobrazující detail položky opět importuje funkce pro zobrazování zpráv z rámce Struts. Navíc obsahuje funkce pro "schovávání" některých částí formuláře: funkci *showNotesBox()*, *showOwnerBox()* a funkci, která kontroluje, zda vložená poznámka nepřesahuje kapacitní limit. Všechny údaje, které jsou zde zobrazeny pomocí *s:property*, jsou ještě zopakovány ve formě *s:hidden*. Je to z toho důvodu, že pokud je třeba na tuto stránku ještě jednou přistupit (např. proto, že došlo k chybě při odesílání formuláře) Strats by je znovu nezobrazil. Ač je výsledná stránka poměrně jednoduchá a strohá, zdrojový kód je relativně dlouhý a složitý. To je způsobeno potřebou měnit dle kombinace aktuální uživatelské role a příznaků zobrazené položky.

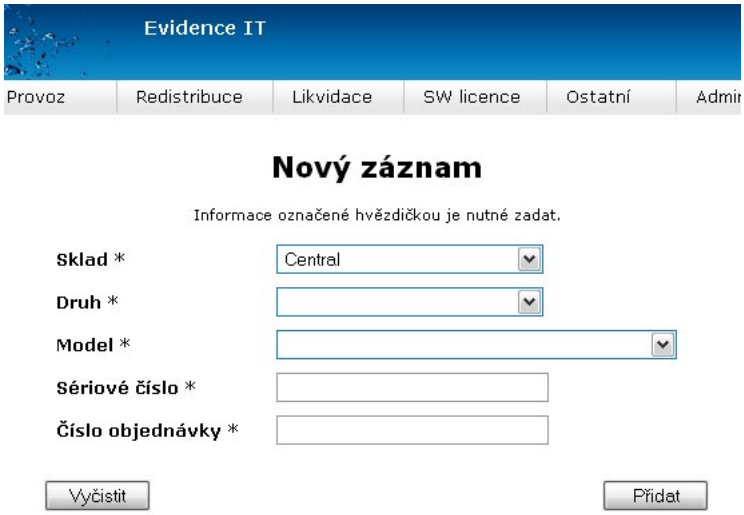
3.5 Realizace životního cyku

Každá evidovaná položka postupuje během „života“ etapami cyklu, které jsou rozberány níže.

Stav: Nová VT - nastává v okamžiku, kdy uživatel v roli outsourcera naimportuje soubor obsahující nové položky VT do systému nebo je do systému manuálně zadá. Poté systém provede logickou kontrolu vstupních dat a ověří existenci položky výpočetní techniky. Pouze v případě, že vše proběhne v pořádku, je nová položka VT založena do systému se stavem „Nová VT“.

Tomuto stavu odpovídá v aplikaci formulář *Přidat novou položku* (viz obrázek 12). Ten je určen k manuálnímu přidávání nových položek do evidence. Po stisknutí tlačítka „Vložit“ provede kontrolu úplnosti zadávaných dat. Pokud je vše v pořádku, je vytvořen přehledný souhrn a uživatel vyzván, aby nově vytvářený prvek potvrdil. Když tak učiní, program zkontroluje, zda by přidáním toho prvku nedošlo k vytvoření duplicitního prvku v databázi. Tuto kontrolu by sice z uživatelského hlediska bylo lepší provést spolu s ostatními před vytvořením souhrnu, ale protože je program určen k souběžné

práci více uživatelů, nebylo by tím zajištěno, že v době, kdy uživatel provádí kontrolu přehledu nového prvku, nedošlo k vytvoření nového prvku s duplicitními údaji. Z hlediska databáze vznikne nový záznam v tabulce ITEM popisující nově zadaný prvek.



Obrázek 12: formulář: Přidat novou položku

Stav:VT k přidělení - se skládá ze tří podstavů:

- Rezervovaná VT
- Volná VT
- Redistribuovaná VT

Položky VT v tomto stavu jsou určeny k přidělení konkrétním osobám do užívání.

Nově zavedené položky VT (tj. všechny položky ve stavu „Nová VT“) musí být aktivovány. Aktivaci provádí manuálně uživatelé v roli EUE, který převádí vybrané položky do stavu *„Rezervovaná VT“* nebo *„Volná VT“*.

Podstav *„Rezervovaná VT“* se vztahuje pouze na ty položky, u kterých je předem známo (již v období objednávky), že jsou rezervovány na konkrétní projekt. Při aktivaci, tj. uvedení položky do stavu *„Rezervovaná VT“*, musí uživatel vyplnit název projektu, na který je položka rezervovaná.

Podstav *„Volná VT“* se týká převážné vetšiny nových položek v systému. Při aktivaci mohou být uživatelem hromadně vybrány a do tohoto stavu uvedeny. Název projektu se v tomto případě neuvádí.

Podstav *„Redistribuovaná VT“* obsahuje položky, které již v minulosti byly přiděleny, užívány a nyní jsou opět redistribuovány. Do tohoto stavu se položky VT dostanou pouze ze stavu *„Redistribuce“*, kde uživatelé v rolích outsourcer, opravář nebo redistributor přidělí vybraným položkám klasifikaci A. Položky, které jsou ve stavu *„Redistribuce“* klasifikovány jako B, C a D se do stavu *„Redistribuovaná VT“* nesmí dostat.

Stav: Přidělená - do tohoto stavu se položka VT dostává, jakmile uživatel v roli distributor provede přidělení konkrétní VT na konkrétní osobu (současný vlastník VT). Dále přepíše číslo servisního požadavku (SP) do systému a provede potvrzení.

Stav: V provozu - do tohoto stavu přechází položka VT z těchto stavů a podstavů:

- Stav: Přidělená
- Podstav: Odkoupená ze stolu
- Stav: VT k přidělení – pouze ve výjimečných případech

K přechodu ze stavu *„Přidělená“* dochází automaticky. Pokud je servisní požadavek shledán ve stavu „Uzavřen“, dojde v systému evidencí VT k automatické změně stavu položky VT na *„V provozu“*.

Přechod ze stavu „Odkoupená ze stolu“ provádí uživatel v roli schvalovatel odkupu, který vybere položku určenou k odkoupení ze stolu a potvrdí její změnu stavu na *„V provozu“*.

Ze stavu *„VT k přidělení“* se položka dostává v případě, kdy je zajištěno okamžité předání VT konkrétní osobě. Proto změnu stavu provádí uživatel v roli distributor, který při přidělování VT konkrétní osobě (vlastníkovi) zvolí příznak na převedení položky VT rovnou do provozu (tj. zatrhne chack box „Výjimka“).

Stav: Redistribuce - v případě, že je položka umístěná na redistribučním skladě, nachází se ve stavu *„Redistribuce“*. Tento stav se skládá z techto podstavů:

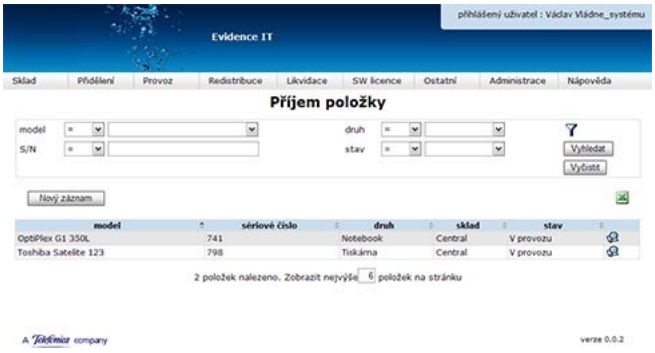
- Přijatá na sklad
- V opravě

Bližší definice podstavů

Podstav *„Přijatá na sklad“* nastává, jakmile uživatel v roli outsourcera nebo redistributora vyhledá v systému S/N dané položky a provede potvrzení jejího přijetí. Nebo také může nastat v případě, kdy položka VT není dosud vedena v aplikaci, ale je nutné ji přijmout na sklad. Outsourcer i redistributor provádí příjem pouze na svůj sklad. Jestliže se položka VT dostala do tohoto podstavu ze stavu *„Neopravitelná“*, bude u této položky VT přednastavena klasifikace na „D“. Jestliže se však položka VT dostala do tohoto podstavu ze stavu *„V provozu“*, klasifikace u ní přednastavena nebude. Přechod z podstavu „Přijatá na sklad“ do následujících stavu nebo podstavu nastává v okamžiku, až outsourcer nebo redistributor doplní do systému aktuálně zjištené HW parametry o položce VT, doplní poznámku, přidelí klasifikaci a provede potvrzení. Na základě přidělené klasifikace se položka dostává do těchto stavů nebo podstavů:

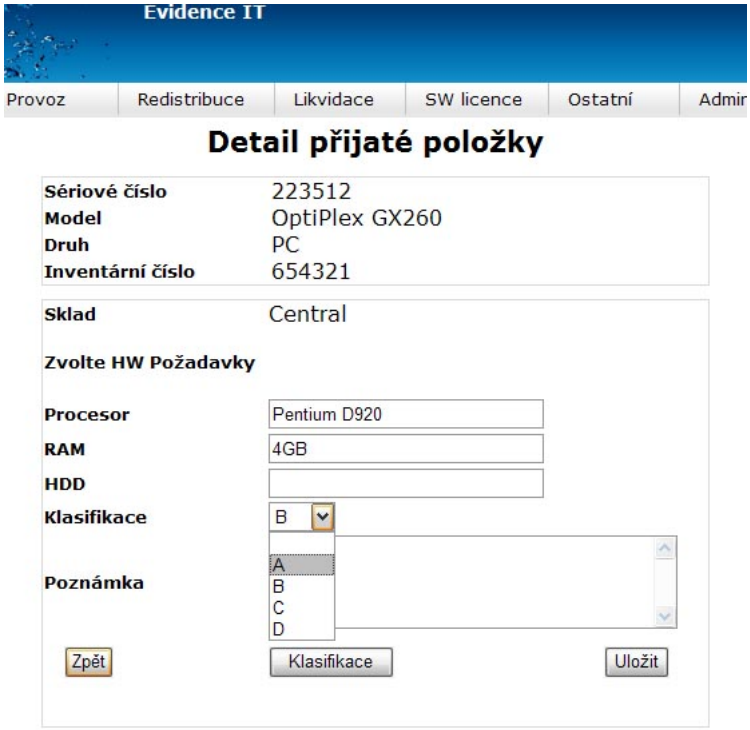
- Klasifikace A = převod položky VT do stavu: *„Redistribuovaná VT“*
- Klasifikace B = převod položky VT do stavu: *„V opravě“*
- Klasifikace C nebo D = převod položky VT do stavu: *„K vyřazení“*

Tyto funkce jsou v programu sdruženy ve formulářích *„Příjem položky* a *„Doplnění HW parametrů položek“* Oba formuláře mají stejný vzhled (viz obrázek 13), liší se pouze ve stavu zobrazených položek.



Obrázek 13: formulář: Příjem položky

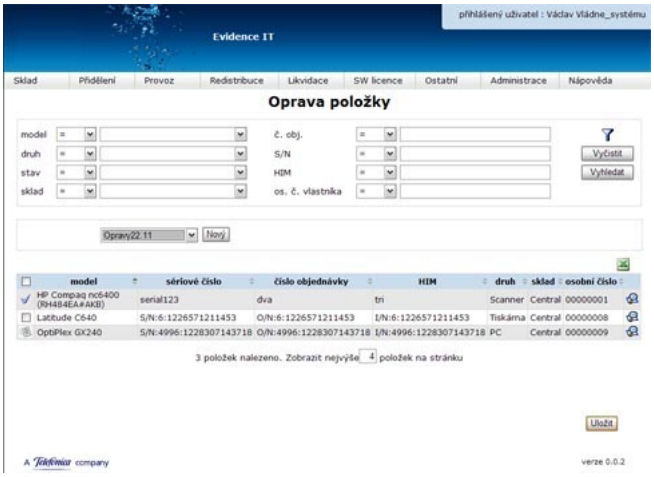
První z nich uživateli zobrazí položky ve stavu *„Neopravitelná“* nebo *„V provozu“*. Uživatel jednu z nich vybere a zobrazí si její detail, nebo stisknutím tlacítka „Nový záznam“ sdělí programu, že bude vytvářet novou položku (viz obrázek 14). Program vygeneruje formulář s detaily položky, na kterém uživatel doplní zjištěné údaje. Stav položky následně buďto uloží, nebo provede její klasifikaci. Uložením je pouze změněn stav položky na *„Přijatá na sklad“*. Položky v tomto stavu jsou pak vidět ve formuláři *„Změnit HW parametry“*. Pokud uživatel stiskne tlačítko „Klasifikace“, provede program kontrolu úplnosti dat a v kladném případě změní stav položky podle zadání.



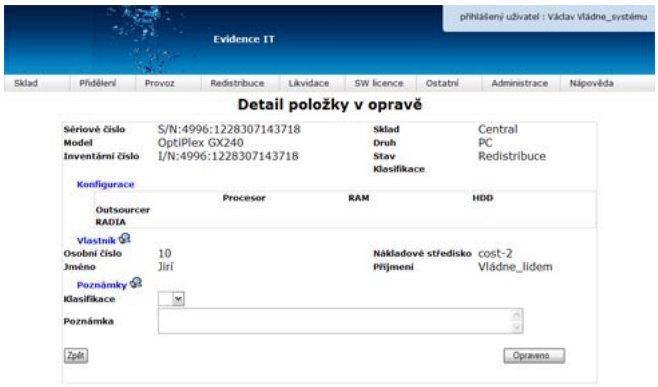
Obrázek 14: formulář pro vkládání aktuální HW konfigurace parametrů

Podstav *„V opravě“* nastává v okamžiku, kdy uživatel v roli outsourcer či redistributor přidělí položce VT klasifikaci B. Přechod z podstavu *„V opravě“* nastává, jakmile uživatel v roli opravář změní u dané položky VT klasifikaci z B na A, C nebo D a provede potvrzení.

Tomuto podstavu odpovídá formulář *„Oprava položky“* obr 15. Ten umožňuje uživateli v roli *„opravář“* vybrat ze zobrazených položek ve stavu *„V opravě“* jednu, zobrazit si její detail, připsat poznámku o opravě a změnit její klasifikaci na A, C nebo D. Podle toho se také změní stav položky tak, jak bylo uvedeno výše. Dále má opravář možnost rezervovat si jednu nebo více položek přidáním do seznamu rezervovaných. Tyto seznamy může libovolně vytvářet, editovat nebo mazat. Ostatním uživatelům ve stejné roli se tyto položky zobrazí s ikonkou seznamu a nemohou býti jimi editovány. Formulář zobrazující detail položky v opravě je na obrázku 16.



Obrázek 15: formulář zobrazující položky v podstavu „V opravě“



Obrázek 16: formulář zobrazující detail opravované položky

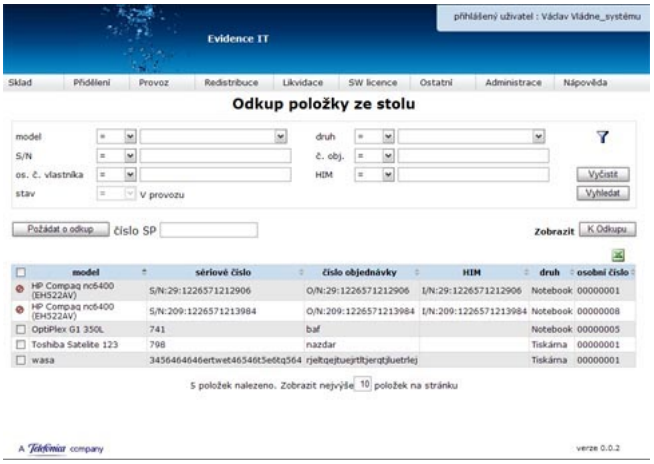
Stav: K Vyřazení - se skládá ze tří podstavů:

- Odkoupená ze stolu
- Nevyhovující – funkční
- Nevyhovující - nefunkční

Do podstavu *„Odkoupená ze stolu“* se položka VT dostává ze stavu:

- V provozu
- Neopravitelná

Přechod ze stavu *„V provozu“* nastává v okamžiku, jakmile uživatel v roli schvalovatel odkupu označí položku VT příznakem „Odkup“ a provede potvrzení. Pro tyto účely slouží formulář *„Odkup položky ze stolu“*, který je na obrázku 3.5. Uživatel zde má možnost přidělit jednomu nebo více prvkům příznak k odkupu a přidělit číslo servisního požadavku. Formulář zobrazuje i položky které jsou z nějakého důvodu k odprodeji nezpůsobilé. Stisknutím tlačítka „K Odkupu“ zobrazí v tabulce ty položky, které již mají příznak k odkupu, ale systémem ještě nebyly vyřazeny z databáze, tj. neproběhlo ještě zpracování příslušného servisního požadavku.



Obrázek 17: formulář: Odkup položky ze stolu

Přechod ze stavu *„Neopravitelná“* nastává v okamžiku, kdy uživatel v roli redistributor definuje, že položka VT bude odkoupena ze stolu (tj. přidělí jí příznak „Odkup“). To se provádí z formulře *„Detail položky určené k likvidaci“* zaškrtnutím příslušného checkboxu.

Do podstavu *„Nevyhovující – funkční“* se položky VT dostanou pouze v případe, když uživatel v roli outsourcer, redistributor nebo likvidátor přidělí vybraným položkám klasifikaci C. V tomto podstavu přiděluje likvidátor vybraným položkám příznaky:

- Prodej charita
- Likvidace odprodejem

Do podstavu *„Nevyhovující – nefunkční“* se položky VT dostanou, jakmile uživatel v roli outsourcer, redistributor nebo likvidátor přidělí vybraným položkám klasifikaci D. V tomto podstavu přiděluje likvidátor vybraným položkám příznaky:

- Likvidace odprodejem
- Fyzická likvidace

Ke správě obou těchto stavů slouží formulář *„Likvidace“*, který zobrazuje položky vždy v jednom ze zmiňovaných stavů. Umožňuje zobrazit detail jednotlivých položek, v němž je možné pro uživatele v rolích likvidátora a redistributora zvolit způsob likvidace jednotlivých položek.

Stav: Vyřazená - do tohoto stavu se položka dostává ze tří podstavů:

- Odkoupená ze stolu
- Nevyhovující - funkční
- Nevyhovující - nefunkční

Položka VT přichází do stavu *„Vyřazená“* z výše vyjmenovaných podstavů vždy automaticky a to na základě zjištění stavu příslušného servisního požadavku k dané položce VT. Pokud je servisní požadavek shledán ve stavu „Uzavřen“, dojde v systému evidencí VT k automatické změně stavu položky

VT na *„Vyřazená“*. V případě, že dojde k chybnému přesunutí položky do stavu *„Vyřazená“*, provede uživatel v roli admin–system její zaktivnění. Takto zaktivněná položka změní svůj stav na podstav *„Přijatá na sklad“* stavu *„Redistribuce“*. Tuto akci může uživatel provést z formuláře *„Aktivace vyřazených položek“*, kde vybere jednu nebo více položek a zmáčkne tlačítko „Aktivovat“.

Stav: Archivace - přechod do tohoto stavu provádí uživatel v roli likvidátor pouze u položek, které jsou ve stavu *„Vyřazená“*. Tuto změnu provádí ve formuláři *„Archivace“*. Ten umožňuje jednak označit jednu nebo více položek ve stavu *„vyřazená“* v zobrazené tabulce a hromadně je převést do stavu *„archivace“*. Položky jsou tím smazány z databázové tabulky ITEM a jsou pro ně vytvořeny nové řádky v tabulce ITEM_ARCHIVED. Tato tabulka neobsahuje všechny sloupce z tabulky ITEM. Některá data jsou tím zcela ztracena a převod zpět již není možný. Formulář *Archivace* tyto položky dovoluje zobrazit, ale nedovoluje provádět s nimi žádné další akce.

3.6 Testování aplikace

Konečná fáze implementace programu byla doprovázena testy, které měly odhlait případné slabiny programu. Pro tento účel bylo vygenerováno třicet tisíc záznamů reprezentujících položky databáze. Ty byly uvedeny do stavu „Nová VT“ nebo „V Provozu“. Dále byly vytvořeny imaginární uživatelé v jednotlivých rolích zmiňovaných v analytické části (viz tabulka 3). Takto velký objem dat byl samozřejmě vytvořen pouze pro zohlednění vlivu velikosti databáze na rychlost programu při kritických operacích, jako je vyhledávání nebp řazení dat. Ta se ukázala jako naprosto dostatečná. Dále se již pracovalo s menším počtem ručně vytvářených položek.

Výsledek pak byl odeslán testerskému oddělení, které mělo za úkol ověřit konzistentnost dat při simulovaném užívání evidence a zhodnitit jenoduchost

a intuitivnost ovládání. Průběžně pak byly zpracovávány všechny připomínky, které z tohoto testování vzešly a program vylepšován, dokud nebyly všechny chyby odstraněny.

Závěr

Program je ve své výsledné podobě schopen plnit všechny požadavky dané zadáním. Aplikace má minimální HW nároky (pro korektní zobrazení je nutné rozlišení alespoň 1024x768) a je odladěný pro užití v prostředí prohlížečů IE (od verze 6) a Mozilla Firefox (od verze 2.0). Žádné výraznější potíže se neobjevovaly ani na jiných prohlížečích. Slabším bodem je fakt, že ke všem evidovaným položkám je přistupováno, jako by se jednalo o počítače, i když jde například o monitory, tiskárny nebo scanery. Jde ale zjednodušení, které bylo zákazníkem schváleno. Náprava tohoto problému by nicméně mohla být vhodnou náplní příští verze programu.

Evidence, která je výsledkem mé práce, zajišťuje vhodným rozdělením do jednotlivých formulářů uživatelům v jednotlivých rolích přehlednost a jednoduchost nutnou k výkonu jejich potřeb. Zároveň ale dovoluje každému uživateli pouze ty akce, ke kterým je oprávněn.

Hlavním a neocenitelným přínosem pro mne byla možnost vyzkoušet si spolupráci na skutečné zakázce v profesionálním prostředí. Účastnil jsem se v omezené míře analýzy programu (v podobě dodatečných připomínek) i návrhu datové struktury. Dále jsem pak vytvořil grafický návrh aplikace tak, aby odpovídal jednotnému image zákazníka a v neposlední řadě jsem vyvíjel vlastní aplikaci, když jsem na kostru programu vytvořenou zkušenějším kolegou doplňoval akce, JSP a metody určené k získávání a ukládání dat z databáze. V samém závěru jsem si ověřil nezbytnost a nepostradatelnost důkladného testování před samotným odevzdáním práce. Pro tyto účely vznikla databáze obsahující více jak třicet tisíc náhodně vytvořených položek, na které byl program otestován. Díky faktu, že testy prováděla třetí osoba, obeznámená s problematikou jen okrajově, bylo možné krom funkčních chyb najít i nedostatky v intuitivnosti ovládání. Takové problémy také tvořily většinu nalezených chyb. I když jsem se s většinou technologií, které byly

použity (Java, Struts, JSP, SQL, Tiles) při vývoji setkal poprvé, podařilo se mi je zvládnout natolik, aby nakonec všchny celky, které jsem vytvářel, tímto testováním zdárně prošly.

Program, vytvořený na základě zadání od klienta, plně splňoval svůj účel, což zástupce firmy, pro kterou byl vyhotoven, potvrdil na posledním setkání.

Reference

[1] *The Apache Software Foundation [online].* 29.04.2009 URL: http://struts.apache.org/

[2] ARRLOW, J., NEUSTADT, I. *UML 2 a Unifikovaný proces vývoje aplikací: Průvodce analýzou a návrhem objektově orientovaného softwaru.* Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[3] BERGSTEN, H. *JavaServer Pages*, Sebastopol: O'Reilly, 3rd Edition, Prosinec 2003. ISBN 0-596-00563-6.

[4] CAVANNES, Ch. *Programujeme Jakarta Struts: tvorba webových aplikací se servlety a stránkami JSP.* Praha: Grada publishing, 2003. ISBN 80-247-0667-9.

[5] ECKEL, B. *Myslíme v jazyku Java.* Praha: Grada publishing, 2001. ISBN 80-247-9010-6.

[6] PERRY, B. W. *Java Servlet & JSP Cookbook.* Sebastopol: O'Reilly, Únor 2004. ISBN 0-596-00572-5.

[7] RAUHLEY, I. *Starting Struts 2.* C4Media Inc, 2007. ISBN 978-1-4303-2033-3.

[8] PETERKA, J., Archiv cz. [online]. 10. 06. 1997 URL: http://www.earchiv.cz/axxxk160/a706k161.php3

[9] *The Java Language: An Overview.* [online]. URL: http://java.sun.com/docs/overviews/java/java-overview-1.html

[10] *Wikipedia, the free encyclopedia* [online]. 19.05.2009 URL: http://en.wikipedia.org/wiki/CMDB

[11] ZAKHOUR, S. et.al. *Java6 Výukový kurz*. Brno: Computer Press, 2007. ISBN 957-80-251-1575-6.