VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

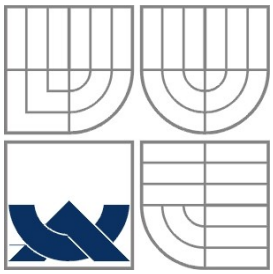
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZVOJ NÁSTROJE NA PODPORU VÝVOJE INTERAKTIVNÍCH APLIKACÍ

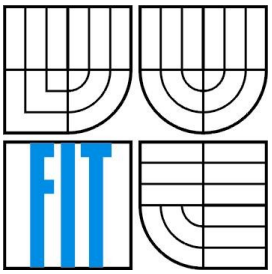
DIPLOMOVÁ PRÁCE
MASTER'S thesis

AUTOR PRÁCE Bc. Michal Rozsypálek
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZVOJ NÁSTROJE NA PODPORU VÝVOJE INTERAKTIVNÍCH APLIKACÍ

ADVANCEMENT OF TOOL FOR DEVELOPMENT OF INTERACTIVE APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S thesis

AUTOR PRÁCE
AUTHOR

Bc. Michal Rozsypálek

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. Jaroslav Zendulka, CSc.

BRNO 2008

Zadání diplomové práce/5458/2007/xrozsy02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání diplomové práce

Řešitel: **Rozsypálek Michal, Bc.**
Obor: Informační systémy
Téma: **Rozvoj nástroje na podporu vývoje interaktivních aplikací**
Kategorie: Softwarové inženýrství

Pokyny:

1. Seznamte se se stávající implementací webového nástroje Process Inspector, použitými technologiemi a strukturou databáze. Dále se seznamte s aplikací Business Process Visual Architect.
2. Nastudujte stávající realizaci komunikace a vizualizaci dat nástroje Process Inspector aplikací Business Process Visual Architect prostřednictvím technologie XML.
3. Navrhněte redesign nástroje Process Inspector včetně uživatelského rozhranní a datového modelu. V rámci redesignu rozšiřte datový model a exportní funkce tak, aby obsahovaly i informace o grafické prezentaci modelu v aplikaci Business Process Visual Architect.
4. Implementujte návrh v programovacím jazyce C# s použitím databázové technologie Microsoft SQL Server 2005.
5. Ověřte funkčnost nástroje při praktickém použití.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Business Process Visual ARCHITECT. Visual Paradigm. Dostupné na http://www.visual-paradigm.com/product/bpva/.
- Fiedler, Z.: Nástroj na podporu vývoje interaktivních aplikací. Diplomová práce. FIT VUT v Brně, 2006.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 až 3.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% 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, DVR-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zendulka Jaroslav, doc. Ing., CSc.**, UIFS FIT VUT
Konzultant: Fiedler Zdeněk, Ing., Allium
Datum zadání: 24. září 2007
Datum odevzdání: 19. května 2008

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

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

Abstrakt

Cílem práce je rozvoj nástroje Process Inspector pro získávání dat a modelování vnitrofiremních procesů. Pro rozvoj nástroje bylo rozhodnuto o přechodu na jiné technologie, nad kterými je nástroj vybudován. Hlavními body je přechod na databázi MSSQL a implementace v prostředí .NET Framework. Při rozvoji nástroje je důraz kladen na zpříjemnění uživatelského rozhraní. V neposlední řadě důležitým aspektem rozvoje nástroje je komunikace s produktem Business Precess Visual Architect.

Klíčová slova

Modelování procesů, Process Inspector, Business Process Visual Architect, MSSQL, ASP.NET, NHibernate, Dependence Injection

Abstract

The aim of the project is development of the tool Process Inspector for business process modeling and collecting informations from stakeholders. For new solution was estimated supporting other technologies. Microsoft MsSQL database server and .Net Framework offer great oportunities to do that. On the other hand there are user graphic interface requirements. Develop tool with intuitive interface and friendly face. Communication with other tools is important feature, we will discuss Business Process Visual Architect communication.

Keywords

Business process modeling, Process Inspector, Business Process Visual Architect, MSSQL, ASP.NET, NHibernate, Dependence Injection

Rozvoj nástroje na podporu vývoje interaktivních aplikací

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením:

Doc. Ing. Jaroslav Zendulka, Csc.

Další informace mi poskytli:

Ing. Zdeněk Opršal, STI Software Technology Institut, a.s.

Ing. Zdeněk Fiedler, STI Software Technology Institut, a.s.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Rozsypálek
Datum

Poděkování

Rád bych též všem výše zmíněným poděkoval za pomoc, kterou mi poskytli, a za neocenitelné rady, které mi pomohly při zpracování této práce.

Obsah

1 Úvod

Jednou z oblastí informačního softwaru jsou ERP systémy. ERP (Enterprise Resource Planning) je systém pro plánování výroby ve společnosti při sledování potřebných zdrojů. ERP systém je začleněn do každé části podniku a velmi výrazně ovlivňuje chod firmy. Do systému jsou integrovány vnitro-firemní procesy, které je nezřídka potřeba změnit, aby byly navzájem kompatibilní a účinně podporovatelné.

ERP systémy dokázaly svůj přínos u velkých firem a organizací, a není třeba diskutovat o tom, zda se to vyplatí. I když původní nasazení ERP systému nedokázalo zcela naplnit očekávání uživatelů, musíme konstatovat, že se postupně z nástroje pro budování konkurenční výhody stal nástroj, bez kterého je úspěch na trhu téměř nemožný.

V této práci se budeme zabývat rozvojem nástroje pro tvorbu interaktivní aplikace. Problematiku návrhu interaktivních aplikací můžeme najít například v práci ing. Fiedlera [1]. Výsledné řešení je zaměřeno na již zmíněné ERP systémy. Cílem této práce je rozvoj nástroje pro modelování vnitro-firemních procesů s ohledem na komunikaci s již hotovými nástroji a navržení intuitivního uživatelského rozhraní.

Ve druhé kapitole se seznámíme se stávajícím řešením nástroje Process Inspector. Budeme se zaměřovat na analýzu cílů, pro které byl nástroj vyvinut, jeho uživatelským rozhraním, použitými technologiemi a také jeho databázovou strukturou. V třetí kapitole se zaměříme na nástroj Business Process Visual Architect. Tento nástroj používá pro modelování vnitrofiremních procesů notaci Business Process Modeling Notation, která je v kapitole zmíněna. V další kapitole si přestavíme důležité technologie a návrhové vzory, které budou využity při implementaci nového řešení. V další kapitole si popíšeme požadavky na rozvoj nástroje. Tyto požadavky budou následně doplněny popisem implementovaného řešení. Testováním, zhodnocením odvedené práce a vyslovením podnětů pro další rozvoj nástroje zakončíme tuto práci. Součástí práce jsou přílohy průvodce instalací, stručná uživatelská příručka.

2 Process Inspector

V této kapitole se budeme zabývat nástrojem Process Inspektor, použitými technologiemi a databázovou strukturou. Budeme se hlavně zaměřovat na použitelnost stávajícího řešení a analýzu požadavků kladených na tento nástroj.

2.1 Hlavní modelovací entity

Abychom byli schopni provádět rozvoj nástroje pro vývoj interaktivních aplikací, je nutné se s tímto nástrojem seznámit. Process Inspector je nástroj pro modelování vnitrofiremních procesů. Je systematicky navržen, aby splňoval základní požadavky jako funkčnost, jednoduchost, spravovatelnost. Pro jednoduchost nasazení v zákazníkově doméně a jednodušší spravovatelnost aplikace byla zvolena varianta webové aplikace. Nástroj splnil základní požadavky, které na něj byly kladeny, těžko ale odolává novým požadavkům.

Jak jsme již uvedli, nástroj slouží pro modelování procesů. Proto je vodné, abychom začali rozborem hlavního prvků, kterým je proces. V nástroji můžeme modelovat více typů procesů, přičemž počet typů procesů není omezen. Z praktického hlediska se ukázalo následujících pět typů procesů obecně použitelných ve většině implementacích :

- Administrativní – lidé, zdroje, peníze
- Korekční – neshody
- Inovativní – znalost
- Materiál a technologie – materiál, čas
- Anonym

Na omezení, která klademe na procesy můžeme nahlížet z několika pohledů. Nejdůležitějším omezením je modelování pouze vnitrofiremních procesů. Nemodelujeme tedy procesy mimo rozsah společnosti.

Za druhý nejdůležitější prvek nástroje můžeme považovat dokument. I pro dokument platí, že může být daného typu a při modelování musíme dodržovat určitá omezení. Nejprve se podívejme na typy dokumentů. Zde můžeme opět říci, že počet typů dokumentů není omezen, avšak v praxi používáme pouze dva typy:

- Vstup – dokument je připojen ke vstupu některého procesu
- Podpora – dokument slouží jako podpůrný materiál při modelování vnitrofiremních procesů

Omezení pro dokumenty platí následující. Dokument nemusí být produkován žádným procesem a také nemusí být zpracováván žádným z procesů. Takovéto dokumenty nazýváme chybějící dokumenty. Další možností je dokument, který není produkován vnitrofiremním procesem,

ale je zpracováván alespoň jedním procesem. Takového dokumenty nazýváme externí dokumenty a takovým dokumentem může být například faktura od dodavatele. Další skupinou dokumentů, pro které nemáme speciální označení, jsou dokumenty produkovány vnitrofiremním procesem a to právě jedním. Není možné, aby byl proces produkován více než jedním procesem. Počet procesů, které zpracovávají dokument (neboli jej mají připojený na svůj vstup) je libovolný včetně nuly. Poslední skupinu dokumentů nazýváme chybějící dokumenty. To jsou takové dokumenty, které nejsou produkovány žádným vnitrofiremním procesem. Můžeme tedy říci, že je to sjednocení osamocených dokumentů a externích dokumentů.

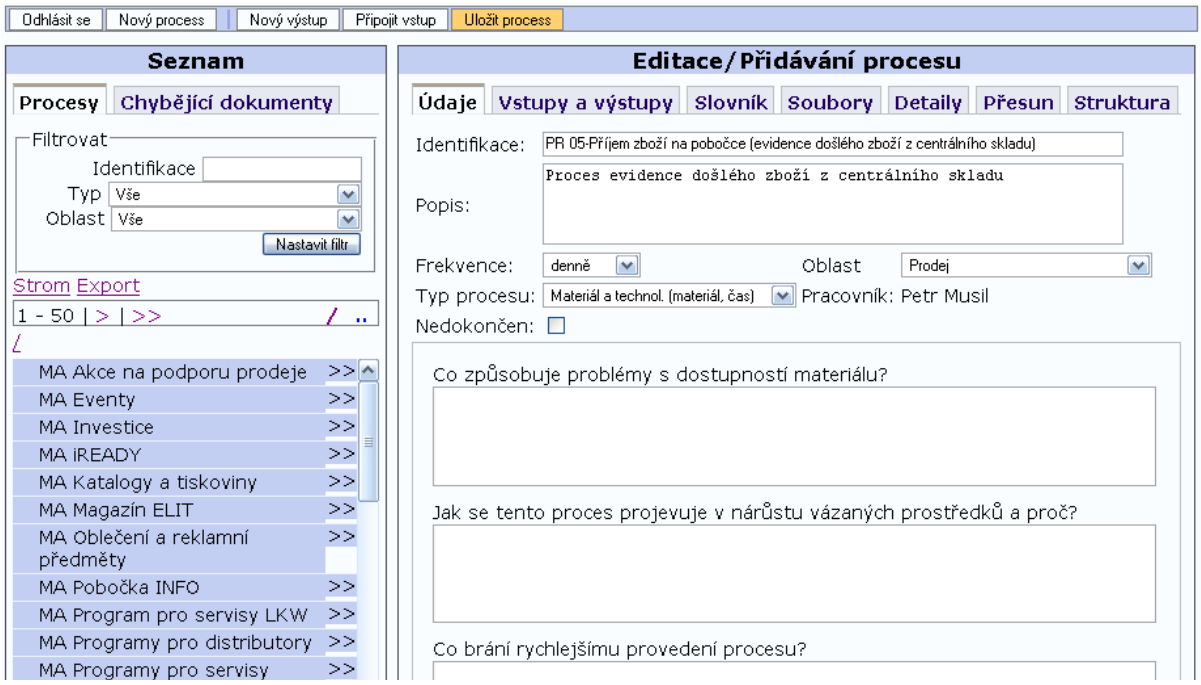
Poznamenejme, že při modelování vnitrofiremních procesů volíme strategii budování procesů, které provádíme a dokumentů, které k tomu potřebujeme. V této strategii se předpokládá, že v počátků modelování dostaneme velkou množinu chybějících dokumentů. Většinou půjde ale o dokumenty, které jsou produkovány uvnitř firmy a budou tedy připojeny na výstup jednoho z procesů probíhajícího v jiné části firmy (tímto vyloučíme dokument z množiny chybějících dokumentů).

2.2 Uživatelský mini-manuál

Jak jsme již uvedli v minulé kapitole, jedním z požadavků na rozvoj nástroje Process Inspector je přepracování uživatelského rozhraní. Pro hledání možných částí pro rozvoj bude nejlépe, když se první seznámíme s nynějším uživatelským rozhraním.

Přihlašování do systémů je řešeno pomocí jména a příjmení. Z bezpečnostního hlediska se to může zdát jako velká bezpečnostní hrozba, musíme si ovšem uvědomit, že cílem přihlašování není autorizace ale autentizace. To znamená, že cílem je informovat kdo jsme a ne, zda máme oprávnění. Zabezpečení je implementováno na nižší úrovní ISO/OSI modelu a to pomocí povolených IP adres.

Po přihlášení k práci s nástrojem je nám zobrazeno hlavní okno viz následující obrázek (Obr. 1). To je rozděleno do tří částí. První část tvoří tenká příkazová lišta umístěna v horní části, která slouží pro zobrazování funkčních tlačítek. Po přihlášení je nám zobrazeno pouze Odhlásit a Nový proces. Počet se však mění podle vybraného detailu. Tím se dostáváme ke zbylým dvou částem nazvaných Seznam a Detail. Seznam se nám zobrazuje v levé části okna, má pevnou šířku a výšku. Při zvolení položky v seznamu je nám zobrazen detail položky v poslední části, která zabírá zbývající část okna.



Obr. 1: Hlavní okno nástroje Process Inspector

2.2.1 Část Seznam

Nyní si popišme možnosti nástroje poskytované v levé části nazývané Seznam. Na první záložce můžeme najít seskupeny informace o procesech. Zde můžeme vyhledávat v názvech procesů či je dále filtrovat dle typu nebo oblasti. Pokud není aplikován žádný filtr, je nám zobrazen seznam kořenových procesů (dělený dle stránkování padesáti položek na stránku). U tohoto výpisu procesů se chvíli zdržíme. Umožňuje nám totiž více informací, než by se mohlo na první pohled zdát.

Na prvním místě je to zanoření podprocesů do procesu. Pomocí výběru „>>“ u procesu se můžeme dostávat na vnořené procesy, návrat zpět do nadřazeného procesu provedeme pomocí odkazů „/“ pro návrat do kořene procesů, nebo „..“ do prvního nadřazeného procesů. Důležité je také zobrazování zobrazování vstupních a výstupních dokumentů pro vybraný proces.

Teď nám ještě zbývá zmínit dva odkazy na záložce pro procesy. První je „Strom“. Tento odkaz nám vygeneruje do části detailu kompletní strukturu modelovaných procesů, jejich vstupní a výstupní dokumenty. Druhým odkazem je „Export“, jež nám generuje opět kompletní strukturu modelovaných procesů. Tentokrát je ovšem výstupní formát XML podle specifikace pro import dat v nástroji Business Process Visual Architect.

Druhou záložkou v části Seznam jsou Chybějící dokumenty. Tento seznam je tvořen dokumenty podle definice, jak jsme si uvedli v kapitole 2.1. Cílem seznamu je zobrazení pouze externích dokumentů, které nejsou produkovány uvnitř firmy.

2.2.2 Část Detail

V této části obrazovky jsou nám zobrazovány podrobné informace dle požadavků uživatele.

- detail procesu
  - údaje
  - vstupy a výstupy
  - slovník
  - soubory
  - Detaily (popis anotace)
  - Přesun
  - Struktura
- detail dokumentu
  - údaje
  - slovník
  - soubory
  - detaily

2.3 Použité technologie

Při analyzování použitých technologií se budeme zaměřovat na obecné postupy a techniky vývoje informačních systémů. Nebude hodnotit vhodnost použité implementace nebo výkonnost zvolené platformy.

Jak jsme již zmínili, nástroj je implementován jako webová aplikace. Tato volba umožňuje snazší správu nasazené verze aplikace. Oproti klasickému řešení klient/server nemusíme zákazníkovi poskytovat žádný instalační balíček nebo řešit distribuci nových verzí. Uživateli aplikace stačí pouze prohlížeč internetových stránek a přístup k internetu. Další výhodou je zkvalitnění servisu, protože technik nemusí jezdit ke klientovi, ale aplikaci opraví na jednom centrálním místě.

Pro implementaci stávajícího řešení byl použit jazyk PHP a pro ukládání dat byl zvolen databázový systém MySQL. Tyto technologie umožňují nasazení aplikace bez platformového omezení. Jak jsme ale uvedli v prvním odstavci této kapitoly, další rozbor této stránky použité technologie nebude cílem analýzy, proto je nebudeme dále rozvádět. Je to ovšem nedílná součást, a je proto důležité, abychom tyto informace alespoň zmínili.

2.3.1 Návrhové vzory

Webová aplikace Process Inspector je navržena podle návrhového vzoru Model – View – Controller. Tato informace byla zjištěna z analýzy zdrojových kódů.

2.3.2 Model – View – Controller

Návrhový vzor Model – View – Controller (MVC) rozděluje aplikaci do tří nezávislých komponent, jimiž jsou

- Model – datový model aplikace,
- View – uživatelské rozhraní,
- Controller – řídící logiku.

I když je MVC chápán jako návrhový vzor, zabývá se mnohem více návrhem architektury než klasické návrhové vzory. Můžeme jej tedy také nazývat architektonický vzor.

Model je doménově specifická reprezentace informací, se kterými aplikace pracuje. Doménová logika přidává význam pro data aplikace. Mnoho aplikací používá mechanismus pro persistentní ukládání dat, jako je třeba databáze. MVC vzor nemá specifickou vrstvu aplikace pro přístup k datům, protože je předpokládáno zapouzdření v komponentě model.

View komponenta zobrazuje model aplikace ve formulářích, které umožňují interakci, typicky jsou to prvky uživatelského rozhraní. Pro jeden model aplikace může existovat více View komponent dle potřeb koncového uživatele.

Controller se stará o zpracování a odpověď na události, typicky jsou to události na uživatelské operace, a také může provádět změny v modelu aplikace.

Pro lepší představu jak tento architektonický vzor funguje, uveďme si typickou posloupnost operací a událostí.

1. Uživatel provede nějakou akci v uživatelském rozhraní (například stiskne tlačítko).
2. Controller zpracuje tuto vstupní událost z uživatelského rozhraní.
3. Controller přistupuje k modelu aplikace a provádí v něm případné změny, podle akce uživatele aplikace.
4. Controller přistupuje ke komponentě View, která zobrazí aktualizovaná data v modelu uživateli. K datům modelu přistupuje nepřímo, avšak komponenta Model nemá do komponenty View žádný přístup. To umožňuje přidávat další komponenty View nezávisle na modelu aplikace. Pouze vytváříme nové komponenty View podle potřeb uživatele.
5. Uživatelské rozhraní je připraveno na další akci uživatele.

2.3.3 Testování aplikace

Součástí stávající implementace nástroje Process Inspector je sada testů pro věření správné funkčnosti a docílení vyšší kvality. Jak jsme již uvedli, k implementaci není žádná dokumentace, a tudíž musíme opět sadu použitých testů zjistit metodou reverzního inženýrství ze zdrojových kódů.

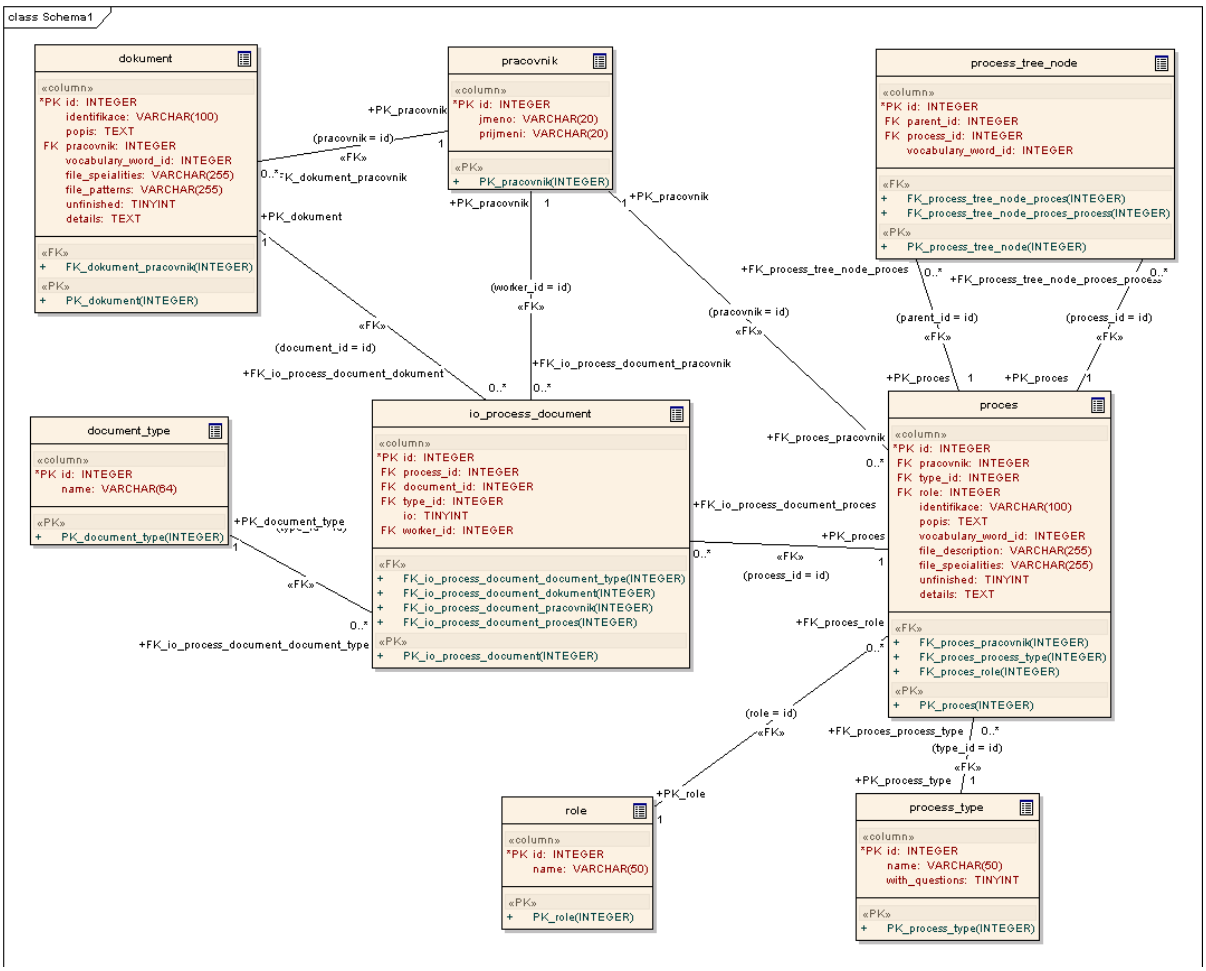
Zjistili jsme, že byl použit obecný framework pro testy. Implementovány byly ale pouze testy modelu aplikace. Ty byly zaměřeny na persistenci dat do databáze a na jejich opětovné načtení. Nebyl implementován žádný test pro kontrolu funkčnosti komponenty Controller nebo View. Zhodnotíme-li tedy přínos implementovaných testů, můžeme říci, že je nedostatečný.

2.4 Datové struktury nástroje Process Inspector

Součástí analýzy nástroje Process Inspector je určitě seznámení se s použitými datovými strukturami. Stále platí, že k nástroji není dostupná dokumentace a bylo nutné analyzovat databázi naplněnou vzorovými daty. Na základě modelovaných procesů v nástroji Process Inspector a odpovídajících dat v tabulkách databáze MySQL byla rekonstruována část datové struktury (viz Obr. 2).

Poznamenejme, že u typů tabulek MyISAM použitých v databázi MySQL není implementována integrita cizího klíče. Tento nedostatek znamená důkladnější analýzu pro sestavení použité datové struktury.

Další důležitou informací, o které bychom si měli říci, je zaměření se pouze na hlavní část datové struktury nástroje. Důraz je kladen na data potřebná pro export ze systému pro nové řešení.



Obr. 2: Datový model nástroje Process Inspector

Prvním důležitým rysem datového struktury je jednotné pojmenovávání sloupce, který slouží jako primární klíč. Pokud je v databázi zapotřebí integritního omezení unikátnosti, je to realizováno pomocí vlastního indexu.

Jak jsme uvedli výše, nejdůležitějšími dvěma entitami nástroje jsou proces a dokument. Tyto dvě entity mají svou reprezentaci v datové struktuře. Tabulka pro ukládání dat o dokumentu má název dokument a tabulka pro ukládání procesů má název proces.

Dále jsme si uváděli vztah mezi dokumentem a procesem. Tento vztah je realizován tabulkou io_process_document, které jej dále rozšiřuje o atributy typ a pracovník. Kontrola omezení pro generování dokumentu pouze jedním procesem je prováděna na úrovni aplikace. Sloupec io určuje, zda je dokument procesem produkován nebo zpracováván.

Další důležitou tabulkou je pracovník. Tato tabulka obsahuje pouze tři sloupce a to id, jmeno a prijmeni a obsahuje seznam uživatelů nástroje. Tato tabulka se stává důležitou tím, že se v databázi zaznamenává, kdo provádí modifikace v modelu. Tyto informace jsou realizovány v databázi pomocí cizích klíčů v referenčních tabulkách. Tabulka process_tree_node realizuje vztah zanoření procesů.

Vazba je realizována pomocí sloupců parrant_id a process_id v této tabulce. První sloupec odkazuje na id procesu, do kterého bude proces zanořen, druhý sloupec odkazuje na proces, který má být zanořen. Tento vztah je rozšířen atributem pro slovo ze slovníku, které má vystihnout smysl vazby.

Posledními tabulkami z datové struktury, které nám zbývá popsat, jsou role, document_type a process_type. Role je tabulka se seznamem oblastí, do kterých může být proces zařazen. Jak jsme uvedli, toto třídění procesů je z důvodu lepší orientace uživatele mezi procesy z pohledu zařazení procesu. Tabulky s typem procesů je podobná tabulce role. Typy procesů ale slouží k třídění procesů z dalšího pohledu a to dle částí, se kterými proces pracuje.

3 Business Process Visual Architect

V této kapitole se budeme zabývat nástrojem Business Process Visual Architect (dále jen BP-VA) a jeho možnostmi. Nejprve si představíme hlavní výhody nástroje, dále si představíme notaci, kterou nástroj používá, potom se budeme zabývat exportem a importem dat do nástroje a nakonec si ukážeme příklad naimportovaného modelu.

Než začneme s vlastním popisem BP-VA, řekneme si motivaci pro jeho použití. Jak jsme zmínili, Process Inspector je jednoduchý nástroj určený pro koncové uživatele. Pro potřeby analytiků jsou ale jeho možnosti hodně omezené, a proto vyhledávají jiné alternativy. Pro náš případ byl zvolen nástroj BP-VA.

3.1 Popis nástroje BP-VA

BP-VA je rozsáhlým nástrojem pro modelování komplexních systémů, který poskytuje nejvíce rozsáhlou podporu Business Process Modeling Notation. Umožňuje diagram analýzy požadavků, ER diagram, diagram toku dat a digram procesů. Pro naše účely se budeme dále zabývat pouze modelování procesů. Poznamenejme také, že nástroj je dostupný v plně funkční zkušební 30 denní verzi zdarma ke stažení. Na stejných stránkách můžeme najít také uživatelský manuál [4], který je dobře zpracován a poskytuje množství příkladů.

Nedívejme se na BP-VA pouze jako na nástroj pro vytváření digramů notace BPMN, ale podívejme se také na několik hlavních výhod, jež umožňují modelování procesů být snazším, rychlejším a lepším:

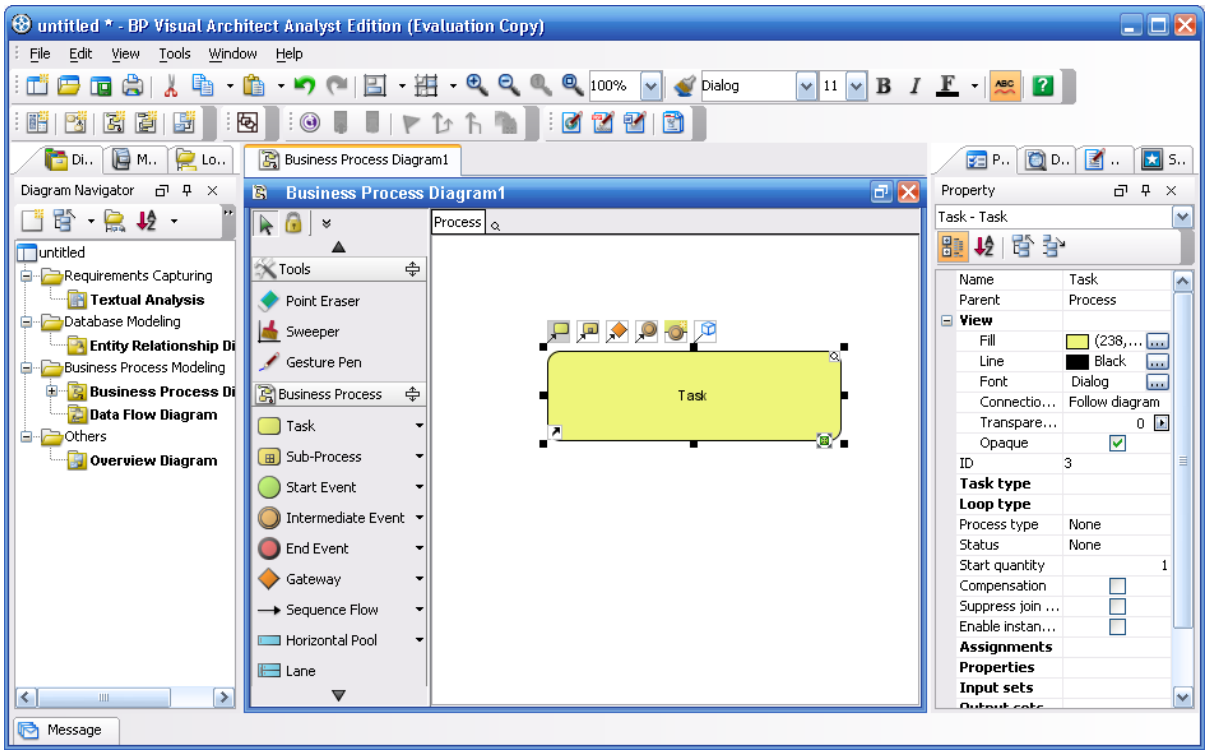
Intuitivní uživatelské rozhraní – v BP-VA jsou projekty, jednotlivé zdroje informací a modelovací nástroje dobře organizovány do přemístitelných oken tak, že si můžeme zobrazit, skrýt nebo přesunout okna tak, abychom si vytvořili modelovací prostředí podle vlastních preferencí. Většina operací vztahujících se na modelování a diagram mohou být provedena buď přímou editací nebo přes kontextové menu a nebo přes tabulku s vlastnostmi. To umožňuje radiálně snížit čas potřebný pro modelování v porovnání s klasickými modelovacími nástroji s modálními okny.

Ve snaze docílit ještě větší efektivity při modelování, nástroj BP-VA přijal zdrojově orientované rozhraní. Znamená to, že kolem aktivního prvku diagramu jsou zobrazena funkční tlačítka. Každé takovéto tlačítko poskytuje funkcionalitu, kterou bychom potřebovali nejčastěji jako vytváření spojení na nové nebo existující prvky, otevírání specifikace modelu nebo změnu velikosti prvku. Umístění funkčních tlačítek kolem aktivního prvku umožní snížení času potřebného pro nalezení prvku v menu a přemístění myši na jeho pozici a zvolení požadované funkčnosti.

Kompletní podpora specifikace a notace BPMN – BP-VA dodržuje poslední verzi notace BPMN od specifikace modelu až po různé možnosti prezentace. Můžeme jednoduše modelovat vnitrofiremní procesy kreslením v Business Process Diagrams a prezentovat je dále svým partnerům, kteří rozumí notaci BPMN.

Inteligentní validace notace a asistent pro modelování – je další klíčovou výhodou nástroje. V průběhu modifikace modelu je automaticky kontrolováno, zda námi modelovaný systém dodržuje notaci. Pokud se snažíme vytvořit nedovolené spojení mezi dvěma prvky, je nám zobrazeno chybové hlášení. Pokud jsou prvky přemísťovány mezi „bazény“ (vysvětlení pojmu bazén uvedeme v následující kapitole), které má za následek nekorektní tok informací, budou tyto informace analyzovány a zobrazeny nám na obrazovce včetně návrhů, jak rychle problém vyřešit.

Některé z výše uvedených výhod můžeme vidět na obrázku Obr. 3.



Obr. 3: Snímek obrazovky nástroje BP-VA

3.2 Business Process Modeling Notation

Business Process Modeling Notation (BPMN) [3] je navržena pro podporu modelování návrhů, které mohou být aplikovány na vnitrofiremní procesy. Znamená to, že modelování ostatních potřeb je mimo rozsah této notace. Uveďme si příklady, které lze modelovat pomocí BPMN:

- Struktura organizace a její zdroje

- Modelování dat a informací
- Strategie
- Vnitrofiremní pravidla

I když můžeme pomocí BPMN modelovat toky dat nebo asociace dat a aktivit není to model toku dat.

Modelování procesů používáme pro znázornění komunikace různých typů informací pro různý typ příjemců. BPMN je navržena tak, aby umožnila modelování mnoho druhů modelů a poskytovala možnost vytvoření end-to-end procesů. Strukturované prvky umožňují čtenáři diagramu jednoduše rozpoznat mezi různými sekcemi. Rozeznáváme tři druhy pod-modelů v end-to-end modelu.

3.2.1.1 Vnitřní procesy

Vnitřní procesy jsou takové, které popisují organizaci a jsou to takové typy, které byly obecně nazvány tok dat. Jednoduchý proces může být spojen s jedním nebo více dokumenty. Pokud používáme terminologii „bazénů“, můžeme vnitřní procesy definovat jako ty, které jsou obsaženy pouze v jednom bazénu. Tok řízení musí zůstat pouze v jednom bazénu, tok zpráv může překročit hranici bazénu pro znázornění interakcí mezi oddělenými vnitřními procesy.

3.2.1.2 Veřejné procesy

Jsou to takové procesy, které znázorňují interakci vnitřních procesů s ostatními procesy nebo účastníky. Pouze ty aktivity, které mohou být použity mimo vnitřní procesy společně s mechanismem toku řízení jsou obsaženy ve těchto modelech.

3.2.1.3 Souhrnné procesy

Souhrnné procesy zohledňují vztahy mezi jednou nebo více objekty. Tyto vztahy jsou definovány jako posloupnosti aktivit. Souhrnný proces může být zobrazen jako dva veřejné procesy, které spolu navzájem komunikují.

3.2.2 Diagramy v BPMN

V této kapitole si představíme grafické objekty a vztahy mezi nimi pro modelování BPMN. Cílem BPMN je poskytnout analytikům jednoduchou a přizpůsobitelnou notaci. Tento požadavek může být těžko dosažitelný spolu s dalším požadavkem, aby notace poskytovala možnost popisu komplexních procesů. Pro dosažení těchto požadavků dělíme seznam grafických objektů do dvou skupin. První je to skupina základních prvků, které budou podporovat požadavky jednoduché notace. Většinu procesů můžeme adekvátně modelovat pomocí těchto prvků. Druhý je kompletní seznam prvků, včetně základních, které poskytují mohutnou notaci pro popis více rozsáhlých systémů.

3.2.2.1 Sada základní prvků

Měli bychom zdůraznit, že jeden z požadavků pro vývoj notace je vytvoření jednoduchého mechanismu pro vytváření modelů procesů a současně bychom byli schopni modelovat komplexní vazby procesů. Přístup pro vyřešení těchto rozporných požadavků bylo zvoleno rozdělení grafických částí notace do dvou skupin. To poskytuje malou sadu kategorií notací tak, že čtenář BPMN diagramů může jednoduše rozpoznat prvky a porozumět významu diagramu. K základním kategoriím prvků můžou být přidány další variace a informace, které splní požadavek na modelování komplexních systémů bez dramatického vzhledu diagramu. Základní 4 kategorie prvků jsou:

Objekty toku jsou hlavními grafickými prvky diagramu, které definují chování procesů. Rozlišujeme tři objekty toku:

- událost,
- aktivita,
- brána.

Objekty pro spojení nám slouží pro vzájemné spojení objektů toku nebo s dalšími informacemi. Opět rozlišujeme tři objekty pro spojení:

- sekvenční spojení,
- spojení zasíláním zpráv,
- asociace.

Objekty seskupování (Swimlanes) nám pomáhají určovat hranice uvnitř modelu a seskupovat objekty do tříd. Máme k dispozici dva grafické prvky:

- Bazén – tvoří jednu skupinu.
- Pruh – je obsažen v bazénu a představuje pouze jeho část.

Vlastní objekty používáme pro dodatečné informace o procesech. Standardizovány jsou pouze tři objekty, ale nástroj pro modelování umožňuje přidat tolik objektů, kolik je potřeba.

- Datový objekt
- Skupina
- Anotace

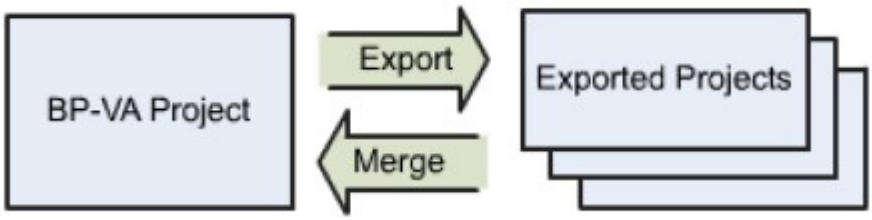
3.3 Exportování a importování projektů

Protože součástí zadání je nastudování komunikace nástroje PI a BP-VA, považujeme za důležité zmínit, jakým způsobem je komunikace řízena v nástroji BP-VA.

Funkce exportování projektu byla do nástroje BP-VA přidána z důvodu správy velkých projektů. Jestliže spravujete obrovský projekt, který obsahuje spoustu diagramů, můžete využít funkce exportu dokumentu, rozdělení do několika menších projektů. Každý exportovaný projekt

obsahuje všechny potřebné informace o diagramech, prvcích modelu a modifikacích v projektu, které se netýkají nadřazeného modelu, ze kterého jsou odvozeny.

Po exportování projektu v něm můžeme provádět libovolné změny a poté je opět sloučit s původním model pomocí funkce pro import dat, jak je znázorněno na Obr. 4.



Obr. 4: Způsob exportování a slučování projektů v BP-VA

Výše jsme uvedli první možnost, jak lze využít funkce exportování dat. Další možností, ke které lze funkce použít je zálohování dat. V kombinaci s dalšími nástroji pro správu verzí, jako je Subversion nebo CVS, můžeme sledovat změny, které byly v modelu prováděny.

Pro naše potřeby se ale zaměříme pouze na možnost, že nástroj umožňuje export a import dat. To nám dává volné ruce k rozšíření nástroje Process Inspector pro komunikaci s tímto nástrojem. O podrobnějším popisu komunikace se můžeme dočíst v další kapitole.

3.4 Komunikace technologií XML

V této krátké kapitole se budeme zabývat stávající komunikací nástrojů PI a BP-VA pomocí technologie XML. Více o možnostech Extensible Markup Language (XML) se můžeme dozvědět ve specifikaci [5]. Protože se jedná o komunikaci dvou nezávislých nástrojů, nezahrnujeme rozbor komunikace jako podkapitolu k žádnému z nástrojů, ale vyčlenili jsme si pro ni vlastní kapitolu. Podklady k této kapitole jsme našli v zadání pro export dat z nástroje PI [2].

3.4.1 Popis komunikace

Process Inspector byl navrhován s cílem vytvořit jednoduchý nástroj pro běžné uživatele zákazníkovi domény. Tito uživatelé dobře rozumí problematice své domény, většinou ale nerozumí problematice modelování procesů. Tento nástroj má překonávat propast mezi analytiky systému a jeho uživateli. Proto s cílem uchovat jednoduchý nástroj, nebyly dále jeho možnosti hluboce rozvíjeny, ale byl zvolen způsob propojení nástroje Process Inspector s nástrojem Business Process Visual Architect. Tento komplexní nástroj je navržen pro analytiky systémů, kteří se v dané

problematice orientují. Cílem komunikace je tedy ponechat jednoduché prostředí pro běžné uživatele, s možností upravit model systému v komplexním nástroji.

Ve stávající verzi nástroje Process Inspector je implementována částešně odchozí komunikace směrem do nástroje BP-VA. Nástroj BP-VA umožňuje obousměrnou komunikaci pomocí technologie XML. Pro exportování dat z nástroje PI se musíme přihlásit. V levé části hlavního okna je nám zobrazen odkaz, pro generování XML souboru. Soubor uložíme na disk svého PC a provedeme import v nástroji BP-VA, jak jsme popsali výše.

V současné verzi komunikace není implementováno grafické rozmísťování prvků v diagramech. Tento fakt je způsoben ukládáním pouze textových informací v nástroji PI. Z nástroje PI exportujeme čtyři typy prvků notace BPMN:

- task – listový proces
- subprocess – všechny nelistové procesy, tj. Takové, které mají alespoň jednoho potomka
- data object – dokument
- association – orientovaná vazba.

V levé části jsou názvy v notaci BPMN a na pravé straně jsou názvy z Process Inspectoru. Jsou exportovány i chybějící dokumenty a jsou označeny odlišnou barvou (červenou). Zcela izolované dokumenty jsou také exportovány v červené barvě, nebudou pro ně ale existovat žádné vazby.

BP-VA používá jazykové řazení (tj. 1, 10, 2, 3, …), pokud má být dodrženo matematické řazení, je třeba v názvech používat úvodní nuly (tj. 01, 02, 03, …). Z toho vyplývá, že je nutné zjistit předem počet nul, na kolik je potřeba čísla doplňovat.

3.4.2 Formát dat pro komunikaci

Formát dat exportovaných z nástroje BP-VA nemá veřejně dostupnou dokumentaci. Pro nastudování formátu bylo nutné vytvořit vzorový model v nástroji BP-VA, provést export dat do XML souboru a poté provedena analýza vyexportovaného dokumentu.

Z analýzy byly zjištěny následující informace.

Hlavním uzlem XML souboru je „Project“. Tento uzel má pro export povinný atribut „Name“, který je nastavován vždy podle instance aplikace Process Inspector.

Modely jsou v nástroji BP-VA definovány pomocí dvou dílčích popisů – je to popis modelu a jeho grafická reprezentace. Tato informace byla odvozena ze struktury exportovaného XML souboru, který obsahuje dvě větve uzlů. Uzel „Models“ obsahuje informace o popisu modelu a uzel „Diagrams“ popisuje grafickou podobu diagramů.

Uzel „Models“ může obsahovat vnořené uzly se jménem „Model“. Každý uzel „Model“ reprezentuje navržený model systému s možností rozdělení do dalších vnořených modelů. Tato možnost je realizována pomocí uzlu „ChildModels“, kde mohou být uvedeny opět další modely.

Povinné vlastnosti modelu jsou ve struktuře realizovány jako atributy uzlu „Model“, nepovinné vlastnosti jsou ukládány v samostatném uzlu „ModelProperties“.

Struktura pro ukládání informací o diagramech v XML souboru je podobná. Uzel „Diagrams“ obsahuje seznam diagramů, realizovaných pomocí uzlu „Diagram“. Uzel „Diagram“ má ve svých atributech uloženy své základní vlastnosti. Další rozšiřující vlastnosti jsou uloženy ve vnořeném uzlu „DiagramProperties“. Informace o prvcích modelu umístěných na daný diagram jsou uloženy v uzlu „Shapes“. Každý prvek diagramu má vlastní uzel „Shape“ s vlastnosti uloženými ve vnořených uzlech. Za zmínku stojí atribut „model“, který obsahuje hodnotu ID pro odkazovaný prvek v modelu systému. Posledním vnořeným uzlem diagramu je uzel „Connectors“ s informacemi o propojení jednotlivých prvku diagramu.

Export jednoho listového procesu. Jak jsme uvedli, listový proces bude převeden na objekt typu Task v BPMN. Duálně se vždy daný objekt zapisuje to uzlu „Models“ a jeho grafická interpretace je v uzlu „Digram/Shapes“. Popisy jsou navzájem propojeny prostřednictvím ID. Uzel „Shape/Model“ se odkazuje na ID modelu, které je unikátní v rámci celého XML dokumentu. Z toho vyplývá, že model i shape mají vlastní unikátní ID. V exportu grafické interpretace je nutné definovat rozměry samotného tvaru a jeho popisku. Protože nástroj PI nemá informaci o vzhledu objektů, jsou tyto hodnoty plněny výchozími hodnotami doporučenými v dokumentaci BP-VA.

Dokumenty z nástroje PI jsou exportovány jako objekty typu „data object“ v BPMN. Opět platí, že informace o dokumentech jsou ukládána do uzlu „Models“ a grafická interpretace do uzlu „Diagrams“. Pro chybějící dokumenty nastavujeme barvu v uzlu „FillColor“.

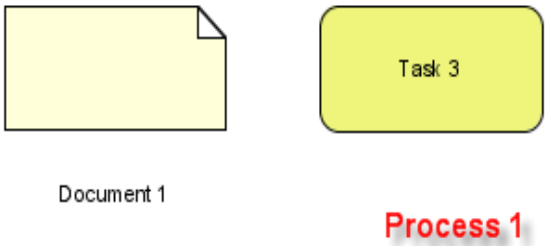
Přidávání vazeb modelu mezi procesy a dokumenty je realizováno pomocí objektu „association“ bez nutnosti uvádět název vazby. Modely typu association mají navíc uzel „ModelRefProperty“ zanořen v uzlu „ModelProperties“. Tyto uzly umožňují definovat orientovanou vazbu, která musí být explicitně nastavena v uzlu „ModelProperties/StringProperty“. Grafická interpretace vazby je v uzlu „Connectors“ a odkazy zde uvedené jsou na prvky diagramu (Shapes) a nikoliv prvky modelu (Models).

Export vnořených procesů. Posledním prvkem exportu, neboli prvkem komunikace nástrojů, jsou podprocesy, realizované objekty „subprocess“ v BPMN. Subprocess je v rámci modelu jako samostatný objekt a navíc má vlastní diagram. Subprocess je tedy v diagramu zobrazen jako tvar (uzel Shape) ale současně existuje druhý diagram s názvem shodným se subprocessem, který definuje uspořádání entit v práci vnořeného procesu. Jedná se tedy o hierarchickou strukturu, která může mít libovolný počet zanoření.

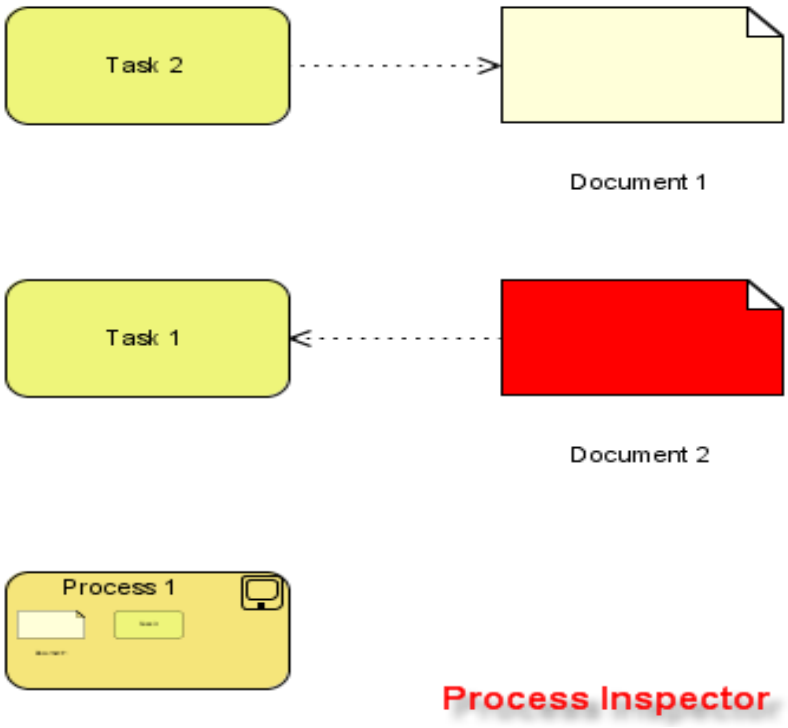
Pokud si to celé shrneme, můžeme říci. Objekty komunikace jsou definovány pouze jednou a to v uzlu „Models“. Mohou ale vystupovat ve více diagramech, propojené pomocí unikátních ID,

případně mohou být zobrazeny vícekrát i v rámci jednoho diagramu (jako více objektů tvaru „shape“ s různým ID). Pro komunikaci je ale stanoveno pravidlo, že daný objekt se bude vyskytovat v rámci jednoho diagramu pouze jedenkrát.

Na níže uvedených obrázcích Obr. 5 a Obr. 6 vidíme příklady exportovaných modelů do nástroje BP-VA. Prvky diagramů jsou rozmístěny automatickou funkcí.



Obr. 5: Model exportovaný do nástroje BP-VA s využitím komunikace přes Xml



Obr. 6: Druhý model exportovaný do nástroje BP-VA s využitím komunikace přes Xml

4 Technologie a návrhové vzory

V následující kapitole si představíme několik důležitých technologií a návrhových vzorů, které jsou pro vybudování nového řešení nástroje Process Inspector stavebními kameny. Prvním stavebním kamenem je volba technologie. Zde jsme zvolili ASP.NET. V první z podkapitol se budeme snažit přiblížit několik důležitých částí této technologie, jež budou použity v implementaci. Po zvolení technologie řešení přistupuje volba návrhu architektury aplikace. V této části si představíme návrhový vzor Model-View-Presenter.

Po volbě architektury aplikace se dostáváme k seznamu dílčích návrhových vzorů, které budeme chtít v aplikaci používat. V práci se nebudeme snažit o výpis všech známých návrhových vzorů nebo o jejich porovnávání. Ukážeme si, jak vybrané vzory fungují, co je jejich cílem a další kapitole popisu implementace si ukážeme praktické použití.

Mimo návrhové vzory si představíme několik důležitých projektů, které nám při sestavení aplikace výrazně pomáhaly. Můžeme se tedy těšit na představení projektů jako je Nhibernate, Log4Net, Caste Project.

4.1 ASP.NET

ASP.NET je framework pro vývoj webových aplikací od společnosti Microsoft, který umožňuje vývojářům vytvářet dynamické webové aplikace, webové služby i webové stránky. Je to volně dostupný framework a jsou k němu i volně stažitelné nástroje umožňující rychlý vývoj. V této podkapitole se zaměříme pouze na webové aplikace, kde jsou hlavní částí stránky a uživatelské prvky. V další kapitole si ukážeme, že tyto dva prvky budou hlavními aktéry pro návrhový vzor Model-View-Presenter.

Nejdůležitějším prvkem ASP.NET je webová stránka, neboli soubor s příponou apsx. Tyto soubory typicky obsahují jako statické (X)HTML tagy, tak tagy prvků nebo komponent vytvořených uživatelem, kde může uživatel opět umístit statický nebo dynamický obsah. Dynamický obsah je umístěn do bloku, jako je tento **<% -- dynamický obsah -- %>** . Pro tvorbu dynamického obsahu je ale Microsoftem doporučena technika kódu na pozadí. Toho je docíleno vytvořením druhého souboru s příponou cs. V tomto souboru se pracuje s objektovým modelem webové stránky a je tedy možné upravit požadovanou funkčnost stránky například obsluhou událostí nebo přepisováním metod.

Jak jsme již zmínili, důležitou částí pro tvorbu webové stránky v ASP.NET jsou prvky definované uživatelem zvané „user controls“. Jde o vytvoření znovu použitelné části stránky, kterou je možné umístit do více stránek. Vývojáři je umožněno rozšíření prvku o metody, vlastnosti,

události. Odpálení událostí na uživatelských prvcích je mechanismus jak může probublat informace z prvku na stránku, kde je umístěn.

4.1.1 NHibernate

I když služba NHibernate není svázána pouze pro ASP.NET, bereme ji zde jako jezdnu z použitých technologií. NHibernate je portovaný projekt Hibernate Core pro Javu do .Net Framework. Poskytuje persistenci objektů prostředí .Net do relačních databází. Pomocí XML souboru popisujeme entity a vztahy mezi nimy. NHiberante automaticky generuje SQL dotazy pro jejich načítání a ukládání. Alternativou pro XML konfigurační soubor mohou být popisy pomocí atributů přímo ve zdrojovém kódu.

NHibernate podporuje transparentní persistenci, neboli naše objekty nemusí být vázány na omezení cizích modelů. Objekty k persistenci nemusejí implementovat žádné speciální rozhraní, nebo dědit z žádné bázové třídy. To nám poskytuje vytvoření vlastního modelu tříd využitím pouze možností .NET (CLR) objektů, podle stylu pro objektově orientovaný návrh.

4.2 Model View Presenter

Model-View-Presenter (MVP) [6] je architektura, která byla poprvé použita v IBM u systému Taligent v rode 1990. Myšlenka architektury byla dále publikována a popsána vývojáři Doplhin Smalltalk. I když původní představy nebyly úplně rozvinuty, hlavní myšlenka se stala populární.

Výhody přístupu MVP vidíme při porovnání rozdílných vlastností pro dva hlavní směry architektury UI. Na jedné straně jsou to architektura formulářů a prvků, jako hlavní proud vývoje UI aplikací. Na druhé straně je to MVC a jeho odvozené architektury. První zmíněná architektura formulářů a prvků poskytuje jednoduchou, snadno pochopitelnou cestu návrhu. Výsledkem jsou jednoduché a znovupoužitelné prvky uživatelského rozhraní. Co je zde problém, a v čem je síla druhého směru MVC, je oddělení prezentačního kódu a samozřejmě oddělení kontextu programového používání domény modelu. MVP můžeme vidět jako krok v spojení těchto dvou přístupů a výběr toho lepšího z obou přístupů.

4.2.1 Popis MVP

View můžeme definovat jako rozhraní, které bude Presenter používat pro načítání nebo posílání dat do nebo z Modelu. Implementace View vytváření instanci objektu Presenter a jako referenci poskytuje sám sebe (formální parametr konstruktoru Presenteru je rozhraní View, zatím co parametr konstruktoru je konkrétní instance třídy View). Pokud jsou vyvolány metody nebo události View, nedělají nic jiného, než volají metody Presenteru, které nemají žádné parametry a nevracejí žádné

hodnoty. Presenter potom získá data z instance View, pomocí vlastností rozhraní View, kterou si uložil jako lokální proměnou při volání konstruktoru. Potom Presenter mění stav modelu voláním metod modelu a nakonec Presenter nastaví data ve View pomocí jeho View rozhraní.

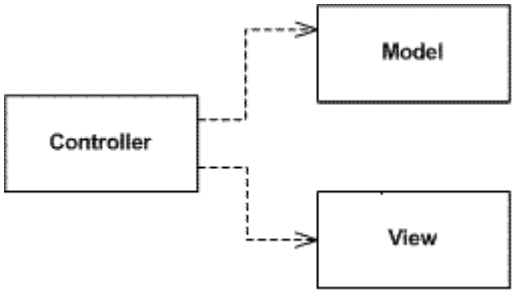
4.2.2 Passive View

Ve článku Martina Fowlera se můžeme dočíst o jeho pohnutkách o rozdělení původního návrhového vzoru Model View Presenter do dvou nezávislých větví. První je Pasive View a druhá je Supervising Controller. K tomuto rozhodnutí došel po delší studiích a v návaznosti na ohlasy k jeho článku o původním návrhovém vzoru v kontextu UI architektury.

Na úvod si uveďme jednou větou M. Fowlera, co to vlastně Pasive View je : „*A screen and components with all application specific behavior extracted into a controller so that the widgets have their state controlled entirely by controller.“* .T textu vyplývá, že stav aplikace (obrazovka, a všechny aplikačně závislé prvky) mají jejich chování extrahováno do Controlleru.

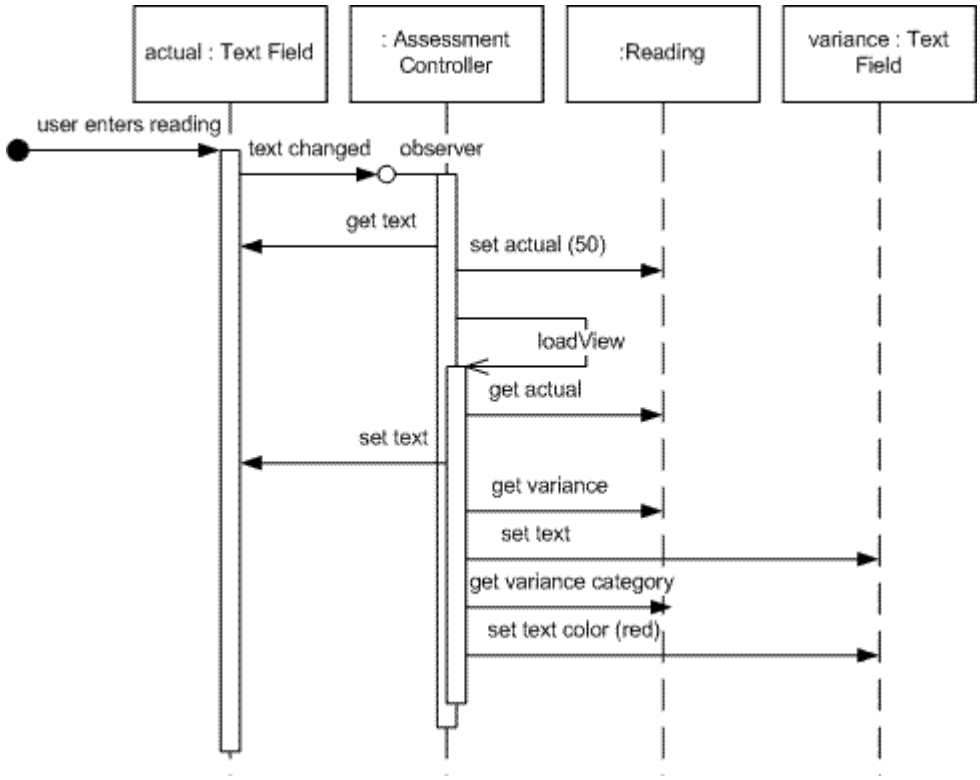
Věčný problém při budování klientských systémů jsou komplikace při jejich testování. Při návrhu většiny klientských frameworků nebyl tento aspekt brán v potaz a tak je pro nás testování takových aplikací značně náročné. Pasive View se snaží tomuto vyhnout tím, že redukuje chování UI aplikace na naprosté minimum využívání Controlleru, které nezpracovávají pouze vstupy od uživatele, ale také aktualizují data zpět do View. Tohle umožňuje zaměřit se při testování na Controlleru s cílem menšího rizika, že budou problémy ve View.

Model Passive View je jen další z variací na model-view-controller a model-view-presenter. Jako u jiných, je uživatelské rozhraní rozděleno do View, který zodpovídá za zobrazování a Controller, který zodpovídá za reakce na uživatelské operace. Základní změny v Pasive-View je myšlence, že je zcela pasivní a není zodpovědný za aktualizace sama sebe v modelu. Výsledkem celé logiky View je v Controlleru. Důsledkem toho je odstranění závislosti mezi View a Model (viz Obr. 7).



Obr. 7: Na rozdíl od většiny MVC konfigurací, Pasive-View nemá žádné závislostí mezi View a Model

Na další obrázku Obr. 8 můžeme sekvenční diagram pro případ zadání dat uživatelem. Na rozdíl od původního MVP, zde se Controller stará o veškerou práci jak by měl View zobrazovat data z Modelu. Textové pole získává data od uživatele. Ihned na to předává řízení dále do Controlleru, jako je tomu v klasickém MVP sekvenci. Controller na to aktualizuje model, a zařídí znovu-načtení View daty z Modelu. Tato metoda zahrnuje načtení všech dat z modelu a použije je k aktualizování View. Tento příklad ukazuje synchronizaci, kdy jakákoliv změna znamená kompletní znovu-načtení View.



Obr. 8: Všechny změny v editování textu ve View jsou zpracovány Controllerem

Hlavním cílem, a důvodem proč použít Pasive View, je testování. Je velice přínosné nahradit View za objekty z frameworku Double Test a tak docílit testování Controlleru bez nutnosti interakce s UI frameworkem.

4.3 Inversion of Control (Dependence Injection)

Jednou z věcí okolo OpenSource řešení, je vytváření alternativ k nejpoužívanějším technologiím. Mnoho z toho vzniká pouze jako reakce na nové věci v hlavním proudu, ale stále mnoho z nich je výzkumná činnost se spoustou zajímavých podmětů. Hlavním problémem, který je potřeba řešit, je

tedy jak sloučit několik různých částí v jedné aplikaci. Typický příklad je „Jak sloučit architekturu Controlleru pro web s rozhraním do databáze, když byly vytvořeny různými týmy s pouze lehkými znalostmi projektů navzájem“. Cílem takového frameworku je sloučit obecné schopnosti komponent z různých vrstev aplikace. Podrobnosti můžeme najít v dokumentu publikovaném M. Flowlerem [7].

4.3.1 Úvod ke vzoru

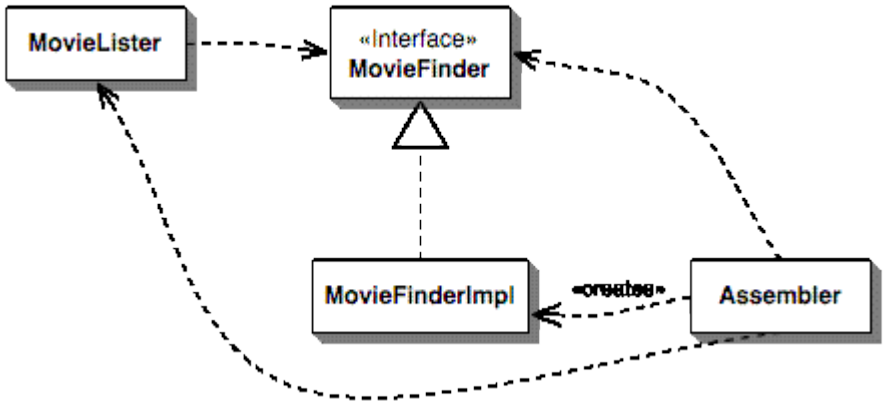
Před výkladem, jak tento vzor funguje, si upřesněme několik pojmů, které budeme pro výklad potřebovat. Komponenta je kousek softwarové aplikace, kterou chceme používat bez změn v části kódu, která není pod správou autora komponenty. Pod částí bez změn rozumíme, že využívající aplikace nemění zdrojový kód komponenty, ačkoliv může měnit chování komponenty nebo rozšiřovat její funkčnost, dle toho, co umožní autor komponenty.

Služba je podobná komponentě v části, že používána cizí aplikací. Hlavním rozdílem je předpoklad lokálního používání komponenty (assembly, dll apod). Služby bude použita vzdáleně přes cizí rozhraní a to jak synchronně, či asynchronně (RPC, web service, socket apod). Příklady, které si dále uvedeme, budou ukázány na službách, ale jsou stejný princip může být použit pro komponenty.

Protože Inversion of Control je společnou charakteristikou pro frameworků, zaměříme se pouze na problematiku Dependence Injection. Upřesněme také, i když se budeme zabývat různými formami řešení závislostí, je více způsobů, jak lze obecně odstínit implementaci z naší aplikace.

4.3.2 Formy Dependence Injection

Základní myšlenkou Dependence Injection je oddělit objekt, překladač, který dokáže poskytnout objektu posluchač implementaci požadovaného rozhraní. Myšlenka je patrná z obrázku níže (viz Obr. 9).



Obr. 9: Závislosti pro Dependence Injector

Existují tři hlavní styly. Jména, která pro ně budeme používat jsou Závislost Konstrutkoru, Závislost nastavení a Závislost rozhraní.

Závislost konstruktoru – při deklarování konstruktoru objektu definujeme všechny závislosti, které objekt potřebuje. IoC kontejner pro jeho vytvoření musí tyto závislosti vyhledat a poskytnout je konstruktoru. Tento způsob závislosti považujeme na nejprůhlednější. Instanci nového objektu nemůžeme vytvořit bez jeho závislosti. Existují ale příklady, kdy vyřešení závislosti nemůžeme provést v dobře vytváření nové instance. Pak můžeme využít další možnosti.

Závislost nastavení – závislé objekty jsou do objektu nastavovány pomocí metod (nebo vlastností). Vytvoření instance třídy můžeme provést bez předchozí závislosti (nic nám ale nebrání využít Závislosti konstruktoru). Po jejich vytvoření jsou závislosti (služby nebo komponenty) poskytnuty posluchači assemblerem.

Závislost rozhraní – poslední, z hlavních stylů, využívá implementaci rozhraní. Objekt posluchač implementuje rozhraní. Objekt assemler řeší vyhledání a poskytnutí závislostí přes toto rozhraní.

5 ProcessInspector II

V původním návrhu nástroje Process Inspector (dále jen PI) byl opomenut důležitý požadavek na intuitivnost uživatelského rozhraní. Uživatelé stávajícího nástroje Process Inspector musí být důkladně proškoleni, aby byli schopni ovládat tento nástroj. Hlavní nedostatky můžeme vidět v nepředvídatelnost na stisk klávesy Enter nebo ztrátu informace při změně stránky. Další nedostatek původního nástroje můžeme vidět v chybném zobrazování informací. Musíme měnit výchozí nastavení prohlížečů, aby zobrazené informace byly přehledné a ucelené.

Důležitým bodem použitelnosti nástroje je komunikace s ostatními nástroji pro modelování vnitrofiremních procesů. Rozvoj nástroje musí probíhat s návazností na rozšiřování komunikace s nástrojem Business Process Visual Architect.

5.1 Požadavky na nové řešení

Z požadavků na původní systém a nových požadavků jsme sestavili seznam nový požadavků na nástroj Process Inspector.

5.1.1 Funkčnost

Nástroj by měl pokrývat funkčnost stávající implementace. Znamená to modelování vnitrofiremních procesů na základě dvou důležitých objektů a to dokument a proces.

5.1.2 Spravovatelnost

Jako velkou výhodu stávajícího řešení vidíme vývoj nástroje jako webovou aplikaci. Toto řešení nám umožňuje kladení minimálních nároků na systémové požadavky uživatelů a současně nám umožňuje jednoduchou spravovatelnost nástroje. Jakmile dojde k migraci na nově implementované řešení, uživatelé systému si nemusí instalovat žádné aktualizační baličky. Uživatelé budou moci rovnou používat nový nástroj, jako používali stávající.

5.1.3 Jednoduchost uživatelského rozhraní a intuitivní ovládání

Jak jsme již zmínili, jedním s hlavních požadavků na nový nástroj je srozumitelnost údajů zobrazených uživateli. Uživatel nástroje musí být srozuměn v každém okamžiku, kde se nachází, s jakými objekty pracuje. Tento požadavek bychom mohli nazvat předvídatelnost operací. Nástroj se

musí v každém okamžiku chovat predikovatelně. Uživatel by měl tušit, jakou operaci provede před tím, než ji opravdu provede. Tento požadavek vidíme jako dosti náročný a těžko hodnotitelný. Souvisí totiž se subjektivním pocitem každého z uživatelů.

5.1.4 Správa společností

Důležitým požadavkem rozšíření je správa více společností v jedné databázi. Cílem je možnost pouze změnou nastavení nástroje přidat další projekt (v reálném světě máme pro každého zákazníka jeden projekt). Nástroj by měl umět přihlašování ke konkrétnímu projektu, bez možnosti zjištění jaké projekty jsou v nástroji modelovány. Měl by také umožnit přihlašování s výběrem projektu, ke kterému se přihlásit.

5.1.5 Komunikace s ostatními nástroji

Komunikace s ostatními nástroji používanými procesními analytiky. Tento požadavek souvisí s jednoduchostí nástroje. Nástroj by měl být navržen pouze pro operace nutné k modelování vnitrofiremních procesů ze strany domény uživatelů. Uživatelé systému znají problematiku své domény a proto pro jejich účely jim stačí jednoduchý nástroj pro popsání svých požadavků. Na druhé stráně máme analytiky informačních systémů, pro které takto jednoduchý nástroj nemůže pokrýt veškeré jejich požadavky. Z toho důvodu je zde požadavek na komunikaci nástroje PI s nástroji používanými analytiky.

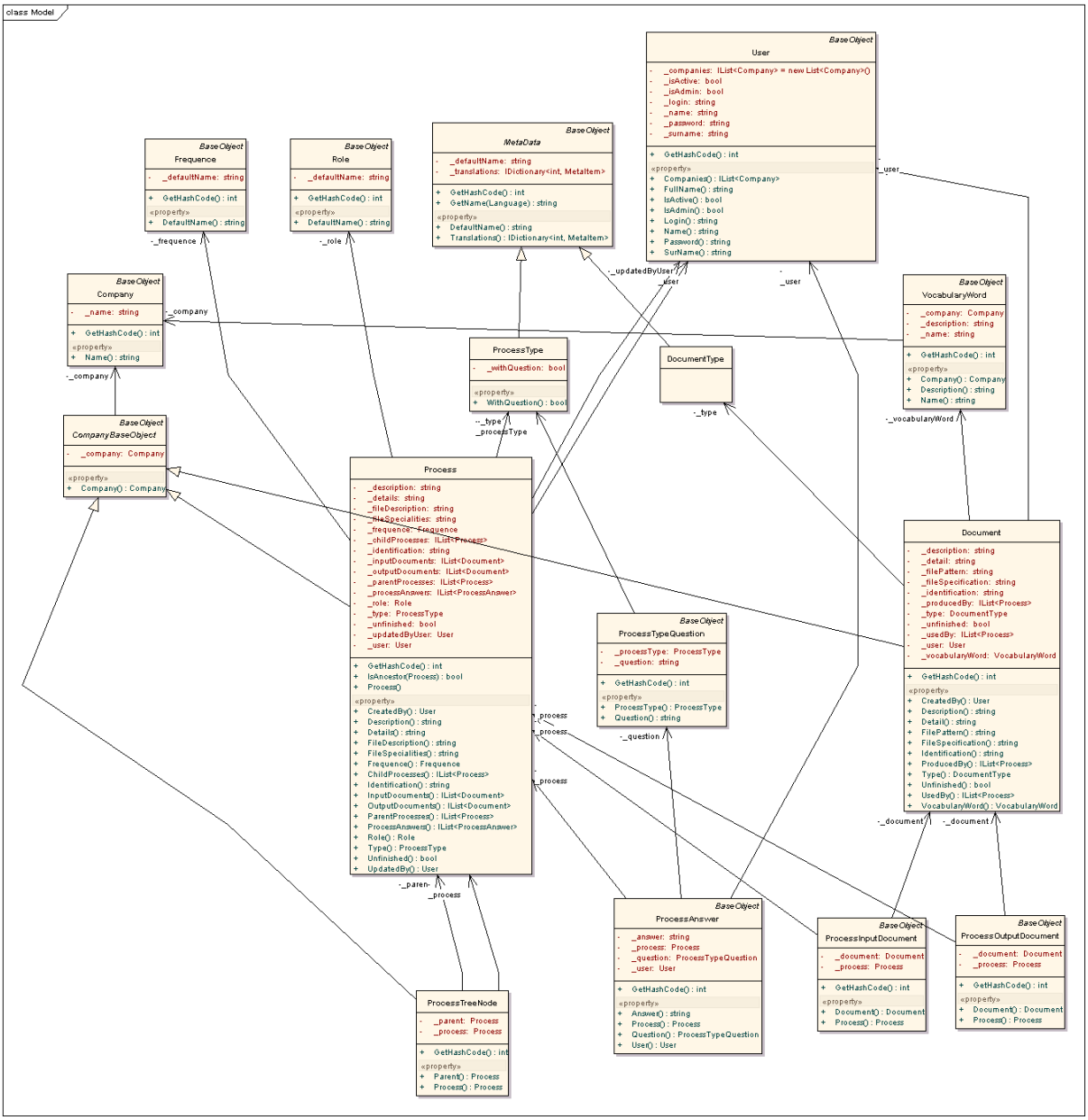
Byl také stanoven požadavek na implementaci komunikace s konkrétním nástroje Business Process Visual Architect (dále jen BP-VA). Tento nástroj umožňuje výměnu dat pomocí technologie XML. Formát pro výměnu dat není veřejně dostupný, můžeme jej ale analyzovat a vyhodnotit.

5.2 Diagram tříd nového řešení

Z výše uvedených požadavků a konzultací s odborníky z praxe jsme sestavili diagram tříd nového řešení (viz Obr. 10). Seznam názvů tříd a jejich krátký popis můžeme najít pod diagramem. Hlavní změnou je reflektování požadavku na spravování více projektů, v reálném prostředí více společností, v jedné databázi. To zapříčinilo umístění hlavní třídy „Company“, na které jsou závislé další dvě třídy „Process“ a „Document“. Vazba nám tedy umožňuje zařadit jeden proces nebo dokument pouze do jedné společnosti, současně nám také dovoluje v jedné společnosti evidovat více procesů nebo společností. Stejná pravidla jako pro „Process“ a „Document“ platí pro třídu „VocabularyWord“.

Pro další vztahy, které budeme popisovat, je nutné zmínit návrh uživatelského rozhraní. Zde bude platit, že uživatel si před prací (při přihlášení do nástroje) bude muset zvolit společnost, kterou chce spravovat. Z tohoto omezení jsme vyvodili několik zjednodušení diagramu tříd nástroje. Pro

vzájemné vztahy mezi entitami závislých na konkrétní společnosti nebudeme uvádět, pro kterou společnost se vztah vázán. Jedná se o vztahy nadřazeného a podřízeného procesu, vstupního a výstupního dokumentu procesu a také vazeb dokumentů na slovníkové pojmy.



Obr. 10: Diagram tříd nástroje PI

- **Company** – projekt realizovaný v nástroji PI. V reálném světě má každý zákazník, odpovídá společnosti, svůj vlastní projekt. Odtud je název společnost.
- **CompanyBaseObject** – abstraktní třída pro další třídy v diagramu, které jsou závislé na společnosti.
- **Process** – vnitrofiremní proces ve společnosti.

- **Document** – dokument vytvořený pro společnost. Název pro dokument se může měnit dle jeho vazeb. Pokud dokument není produkován procesem vazbou „ProcedBy“ jde o chybějící dokument.
- **User** – uživatel spravující společnost. Pro třídy Process a Document je svázán asociacemi pro vytvoření, poslední editaci. Pro třídu ProcessAnswer je vázán informací, kdo odpověděl na otázky k procesu.
- **ProcessTreeNode** – třída realizuje stromovou strukturu procesů. Vazba Process identifikuje process a společně s vazbou Child svazuje vnořené procesy.
- **ProcessInputDocument** – třída poskytuje vazbu vstupního dokumentu procesu.
- **ProcessOutputDocument** – třída poskytuje vazbu výstupního dokumentu procesu.
- **ProcessType** – třída představuje typ procesu
- **ProcessTypeQuestion** – třída představuje otázku svázanou k procesu podle jeho typu. Uživatel vyplňuje pouze otázky, které jsou vztaženy k vybranému typu procesu.
- **ProcessAnswer** – odpověď na otázky vztažené k procesu.
- **Role** – třída představuje oblast působnosti procesu.
- **Frequence** – třída představuje frekvenci opakování procesu.
- **DocumentType** – třída představuje typ dokumentu.
- **Metadata** – třída je připravena pro realizaci vícejazykové mutace nástroje.
    - **VocabularyWord** – třída představuje frázi ve slovníku.

5.3 Rozvržení projektu

V této kapitole se seznámíme s implementací nového řešení. Projekt byl implementován v IDE nástroji Microsoft Visual Studio 2005 SP1. To nám dává možnost výběru mezi webovou aplikací a website projektem. Řešení bylo rozvrženo do sedmi projektů. Prvním vytvořeným projektem byl Utils, nad kterým jsou postaveny všechny ostatní projekty. Druhým projektem byl Core, kde můžeme najít model řešení, včetně rozhraní pro přístup k datům. Dalším projektem je Presenter, který obsahuje Controllery aplikace a projekt Data, kde jsou implementována rozhraní pro přístup k datům. BP-VA projekt obsahuje části aplikace spojené s komunikací s nástrojem BP-VA. Na těchto projektech jsou závislé poslední dva projekty a Web, kde jsou View aplikace a projekt Test, kde můžeme najít testy jednotlivých vrstev aplikace.

5.3.1 ProcessInspector.Utils

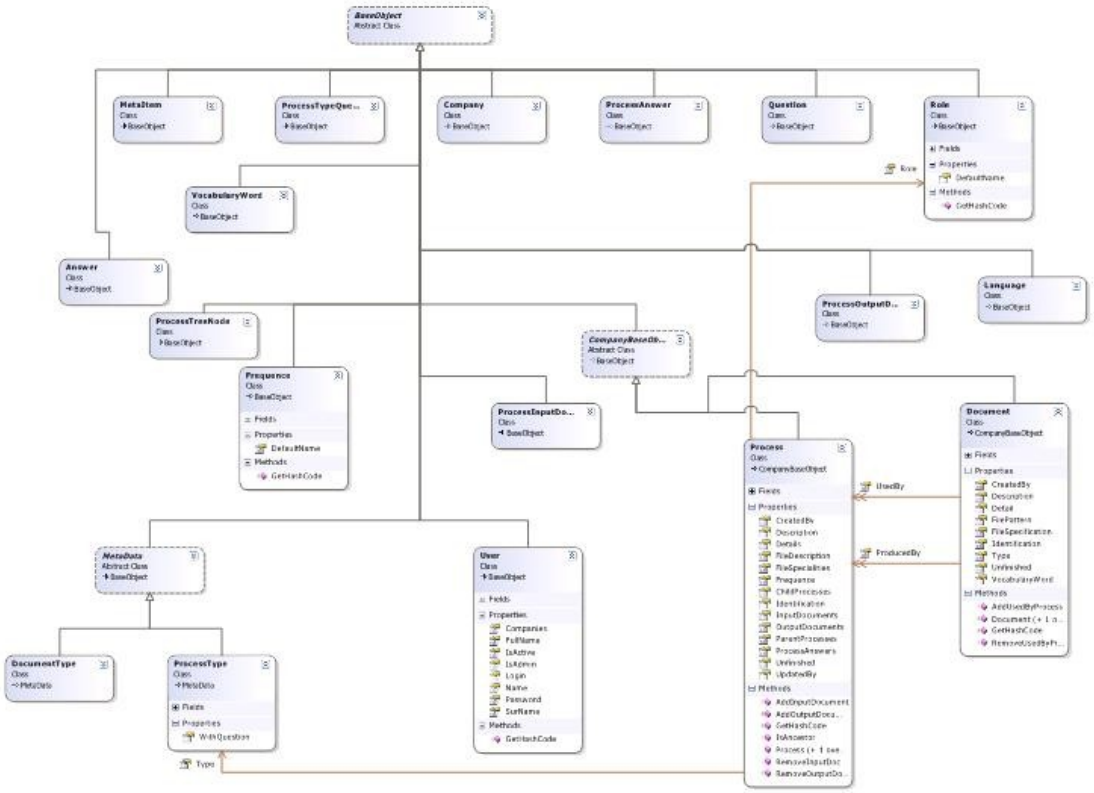
Třídy použitelné nezávisle na projektu nebo jeho vrstvě jsou umístěny do toto projektu. Typicky je to třída pro logování, třída pro metodiku vývoje stylem „designed by contract“. První se podívejme na třídu pro logování.

Třída vychází z volně dostupného řešení Log4Net [8]. Důvod vytvoření vlastní třídy je umístění závislostí na projekty 3. stran do jednoho místa. To nám umožňuje, v případě nutnosti výměny komponenty, změnu kódu pouze na jednom místě aplikace. Knihovna pro logování nabízí několik metod pro informování dle úrovně důležitosti zprávy. Těchto úrovní je 5 (Info, Warn, Debug, Error, Fatal). V konfiguračním souboru knihovny log4net můžeme nastavit jak úroveň důležitosti zpráv, tak i způsob dalšího šíření zprávy. Pro naše účely jsme využili zapisování zpráv do souboru a zasílání chyb emailem. Důležitou vlastností logovací knihovny je možnost vytváření objektu pro odchytávání zpráv dle řetězce nebo typu identifikující cílový objekt. Tuto vlastnost můžeme využít pro oddělení chyb v komunikaci o nástrojem BP-VA od chyb v uživatelském rozhraní či chyb v načítání dat z databáze.

Druhou důležitou statickou třídou je „Check“. Tato třída poskytuje metody pro kontrolu aktuálního stavu aplikace. Pokud tedy v aplikaci máme místo kódu, kde vyžadujeme nějaké vstupy, je doporučeno je zkontrolovat a k tomu jsou přizpůsobeny zmíněné metody. Do projektu jsou také zahrnuty části, které jsou sice specifické pouze pro jeden projekt, ovšem jsou znovu použitelné u druhého řešení. Cílem tohoto projektu je tedy možnost migrace jako základ pro jiné řešení bez nutnosti změny kódu.

5.3.2 ProcessInspector.Core

Projekt ProcessInspector.Core (dále jen Core) obsahuje část aplikace Model. Jsou zde vytvořeny třídy, odvozené z entit ER diagramu, využívané v aplikaci včetně vzájemných závislostí. Pro lepší přehled nad všemi třídami modelu byl vytvořen diagram tříd (viz Obr. 11) ve vývojovém nástroji Microsoft Visual Studio 2005. Diagram vytvořený v tomto nástroji nerespektuje pravidla UML a proto jej nazýváme pouze náhledem na třídy.



Obr. 11: Náhled na diagram tříd nového řešení nástroje PI

Z náhledu můžeme vidět alespoň částečně uspořádání tříd. Jak je dobrým zvykem, všechny objekty modelu by měly mít společného předka. Zde je tímto předkem abstraktní třída „BaseClass“. Protože každý prvek modelu je v našem případě uložen v databázi, obsahuje bázová třída vlastnost „Id“ identifikující instanci třídy v databázi. Dále třída vyžaduje implementaci metody „GetHashCode“, který má za cíl identifikovat instanci objektu třídy bez přiděleného „Id“ z databáze.

Druhou důležitou částí projektu Core, je definice rozhraní pro přístup k datům. Definovaná rozhraní jsou vztažena ke konkrétním třídám modelu. Každé rozhraní dědí od bázového rozhraní, stejně po vzoru tříd. Bázové rozhraní je definováno genericky a pro konkrétní třídy vyžaduje implementaci základních metod jako je vytvoření, uložení, načtení nebo načtení seznamu.

Na závěr popisu projektu Core uveďme, že jsou v adresáři Hbm vytvořeny konfigurační soubory definující mapování prvků třídy na relační databázi pro systém NHibernate.

5.3.3 ProcessInspector.Presenter

Chování nástroje je implementováno v tomto projektu. Pro entity, které je třeba v modelu měnit, je vytvořena složka s Controllerem a rozhraními pro View.

Při vývoji jsme se od přesné definice chování presenterů a view odchýlili k vlastním inovacím. Hlavní změnou je konstruktor bez parametru view. Odkaz na view je předáván jako parametr volaných metod. Důsledkem je využívání jedné třídy presenter pro více tříd view.

Na komunikaci s nástrojem BP-VA můžeme vidět využití Inversion Of Control. Bylo definováno jednoduché rozhraní, které musí implementovat stroj obsluhující komunikaci. Metody jsou pouze dvě, Export a Import. Komunikace byla navržena na pro každou společnost zvlášť. Proto obě metody vyžadují parametr objektu společnosti. Metoda pro import vyžaduje další parametr xml dokumentu s informacemi o společnosti. Metoda export takovýto objekt vrací.

5.3.4 ProcessInspector.Web

Část View je implementována v tomto projektu. V této části řešení se dozvíme nejvíce informací spojených s vlastním používání nástroje. Začneme požadavkem na standardní přihlašovací okno pro každou společnost.

Tento požadavek byl splněn za pomoci vlastního HTTP modulu a stanovení formátu přihlašovací adresy. Vlastní modul reaguje na událost začátku požadavku na server. Tato událost se u ASP.NET aplikací týká všech požadavků na webový server jako jsou obrázky, soubory s kaskádovými styly, javascriptové soubory, ale také i datové soubory zip, rar, avi, či swf. Proto je třeba reagovat pouze na požadavky na přihlašovací adresu, kde vycházíme z definovaného formátu. Byl sestaven regulární výraz, který kontroluje podobnost cesty požadavku a v případě, že vyhovuje, převezme se z adresy název společnosti a přepíše se adresa požadavku na serveru na přihlašovací stránku s parametrem společnosti. Po krásné teorii je čas na ukázkový příklad. Uvažujme o společnosti s názvem VUT a aplikací Process Inspector umístěnou na doméně www.example.com. V takovém případě by byla adresa http://www.example.com/VUT.pi. Z adresy modul pomocí regulárního výrazu převezme název a přepíše požadavek na serveru na adresu http://www.example.com/Login.aspx?company=VUT.

Užití HTTP handleru nám pomohlo vyřešit požadavek vzniklý v průběhu vývoje. Pro zadávání detailního popisu dokumentu a procesu bylo třeba použít editor umožňující zvýraznění textů. Byl nalezen free editor openwysiwyg. Editor má ve svém zdrojovém kódu odkazy na doplňující soubory. Tyto odkazy je možné zapsat dvěma způsoby, relativně nebo absolutně. Protože editor využíváme v aplikaci na více místech, není možné předpokládat platnou relativní adresu. Využití absolutní cesty by nám ale znemožnilo nasazení aplikace na zanořených adresách. Proto bylo rozhodnuto využití handleru, který vrací obsah zdrojového kódu editoru. Při výpisu jsou cesty programově normalizovány na absolutní formát.

Lokalizace byla dalším požadavkem na nové řešení. Předpokládáme, že aplikaci bude možné využívat i pro zahraniční partery. Díky společnosti Allium s.r.o. máme vyhlídky na využití aplikace v polštině, angličtině a češtině.

Další částí projektu je obsluha komunikace s nástrojem BP-VA. Díky návrhového vzoru Model View Presenter je funkční část komunikace odstíněna fyzického přenesení dat. To je zatím ponecháno na ručním zpracování člověkem.

5.3.5 ProcessInspector.BPVA

Tento projekt obsahuje implementaci komunikace nástroje Process Inspector s nástrojem Business Process Visual Architect. Rozsah problematiky komunikace si vyžádal vlastní kapitolu, na kterou se zde odvoláváme. Na projektu si ukážeme využití návrhového vzoru Dependence Injection.

Základní části projektu je třída „CommunicationEngine“. Tat je vytvářena pomocí Inversion Of Control kontejneru a využívá jeho vlastnosti „Constructor Injection“. Požadována je služba pro práci s daty. Samotná třída implementuje rozhraní „IBPVAEngine“ a slouží jako služba pro ostatní uživatele kontejneru. Praktickou ukázku si ukážeme na Windsor Container. Agreguje inversion of control kontejner, který rozšiřuje o škálovatelnost a konfiguraci. Získání instance objektu zpracovávajícího rozhraní „IBPVAEngine“ získáme příkazy:

```
IWindsorContainer winCont = new WindsorContainer(new XmlInterpreter());
IBPVAEngine eng = winCont.Resolve(typeof(IBPVAEngine)) as IBPVAEngine;
```

5.3.6 ProcessInspector.Data

Využití služby NHibernate [9] pro persistenci objektů v relačních databázích si představíme v projektu ProcessInspector.Data (dále jen Data). V projektu vycházíme ze třídy implementované v projektu Utils. Pro vytvoření instancí tříd jsme využily návrhového vzoru factory.

Třída „AbstractNHibernateDao“ implementuje většinu funkcí pro přístup k datům využitých v aplikaci. Třída je implementovány pomocí generik, dostupných v jazyce C# 2.0, a tak radikálně snížila množství zdrojového kódu psaného pro konkrétní třídy. Jak je dobrým zvykem, ukážeme si konkrétní příklad.

```
public class ProcessTypeDao : BaseDao<Core.Model.ProcessType>
      , Core.DataInterface.IProcessTypeDao {
      public ProcessTypeDao(string sessionConfigPath)
            : base(sessionConfigPath) { }
}
```

5.3.7 ProcessInspector.Test

Při vývoji aplikace si autor vyzkoušel metodu vývoje zvanou Test Driven Development (dále jen TDD). Je to metoda řízení vývoje testy. Frameworků, které nabízí potřebné zázemí pro vytváření a provádění testů, je mnoho. V našem případě byl zvolen Nunit. Pro každý projekt řešení bylo sestaveno několik testů, pro vyzkoušení metodiky TDD.

5.4 Komunikace s BP-VA

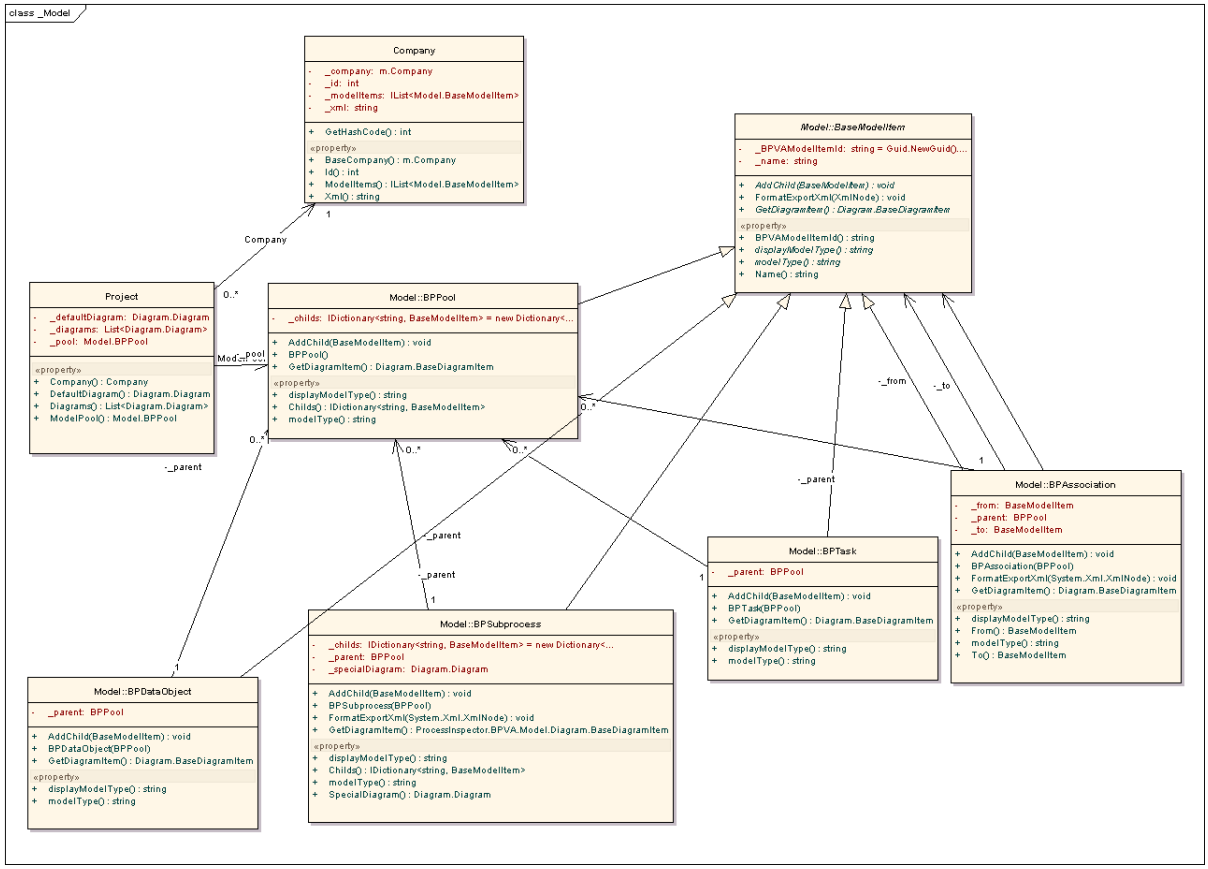
5.4.1 ER diagram modelu pro komunikaci

V následujících odstavcích se budeme snažit přiblížit návrh komunikace. Metodou reverzního inženýrství byl analyzován formát Xml dokumentu exportovaného nástrojem BP-VA. Pro tento formát byl vytvořen objektový model sestávající se ze dvou částí. První část popisuje model dat zadaných do modelu a druhá část popisuje zobrazování modelu. Na obrázku níže (viz Obr 12.) vidíme část nazvanou Model. Protože se bavíme o modelovacím nástroji, očekáváme tedy, že nástroj ukládání informace o modelech. Pro naše účely budeme pracovat pouze se čtyřmi prvky pro modelování.

Diagram tříd modelu pro komunikaci byl navržen následovně. Projekt se může vztahovat pouze na jednu společnost. Pro objekty modelu byla vytvořena abstraktní třída ze které musí všechny ostatní prvky modelu dědit. Abychom byli schopni postihnout hierarchickou strukturu modelů, byla vytvořena třída „BPPool“ realizující skupinu modelů. Třída obsahuje seznam vnořených prvků bázového typu, tudíž může obsahovat i další vnořené skupiny modelů.

Pro procesy modelované v nástroji PI jsme definovali dvě třídy v modelu komunikace s nástrojem BP-VA. První třída odpovídá všem nelistovým procesům s názvem „BPSubProcess“. Třída obsahuje seznam vnořených prvků, stejně jako „BPPool“. Na rozdíl od třídy „BPVA“obsahuje informace o procesu z nástroje PI. Druhou třídou, navrženou pro komunikaci, budeme realizovat výměnu informace o listových procesech v nástroji PI. Tato třída nemůže obsahovat již vnořené prvky a její název jsme zvolili „BPTask“.

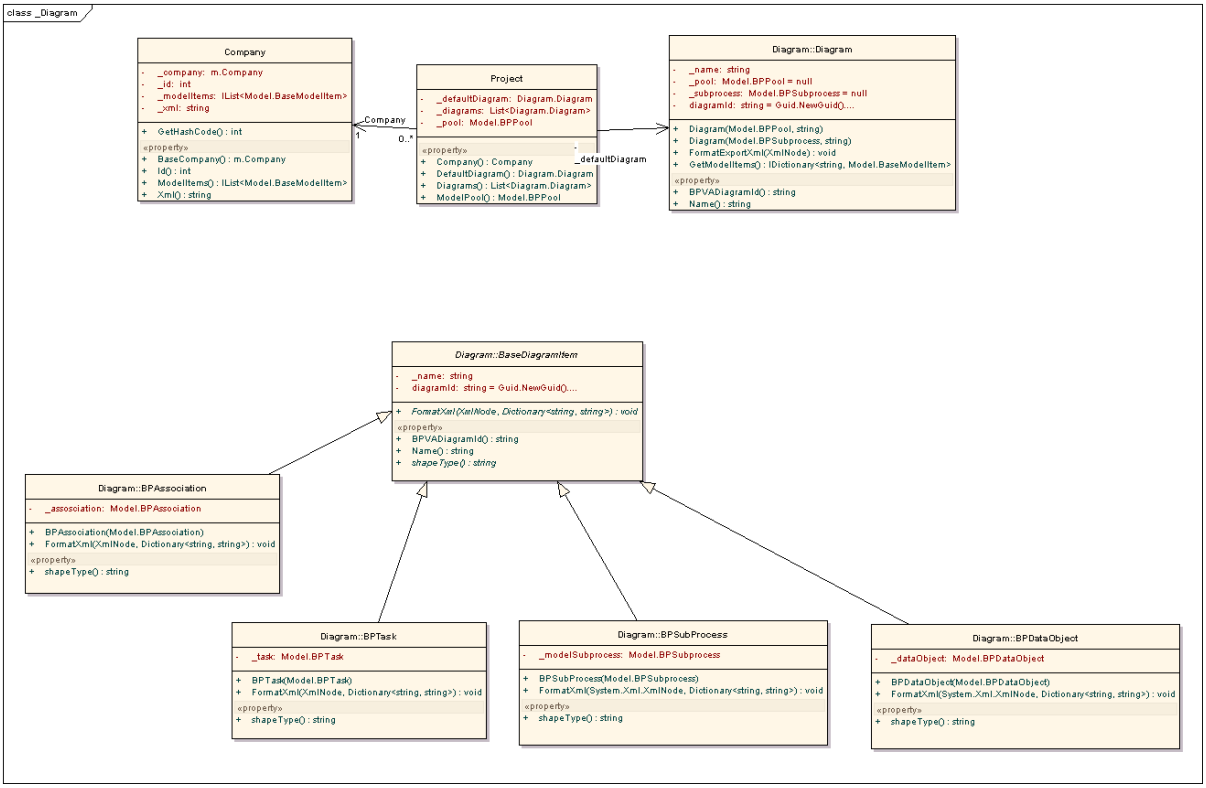
Pro dokumenty z nástroje PI jsme vytvořili třídu „BPDataObject“. Stejně jako třídy „BPTask“, nemůže tato třída obsahovat vnořené prvky. Poslední třídu navrženou pro komunikaci jsme pojmenovali „BPAssociation“. Úloha této třídy je realizovat vazby mezi procesy a mezi procesy a dokumenty v nástroji PI. Vazby, se kterými je třída navržena, poskytují možnosti vytvoření vazby mezi libovolnými třídami modelu komunikace.



Obr. 12: Diagram třídy pro část komunikace s informacemi o modelech

Druhý diagram tříd (viz Obr. 13) znázorňuje vazby tříd nesoucí informaci o grafické podobě modelů. Diagram vychází z diagramu tříd modelu komunikace. Třída projekt byla rozšířena o dvě vazby. První vazba je výchozí diagram, ve kterém budou graficky reprezentovány třídy typu „BPTask“, „BPSubprocess“, „BPDataObject“ a „BPAssociation“. Do diagramu jsme tedy přidali novou třídu „Diagram“ plnící funkci kontejneru. Druhou vazbou, kterou jsme třídu „Project“ rozšířili je seznam tříd typu „Diagram“. Do tohoto seznamu bude pro každou instanci třídy „BPSubprocess“ z modelu vytvořena nová instance třídy „Diagram“ obsahující grafiky reprezentované třídy typu „BPTask“, „BPSubprocess“ a „BPDataObject“ a „BPAssociation“.

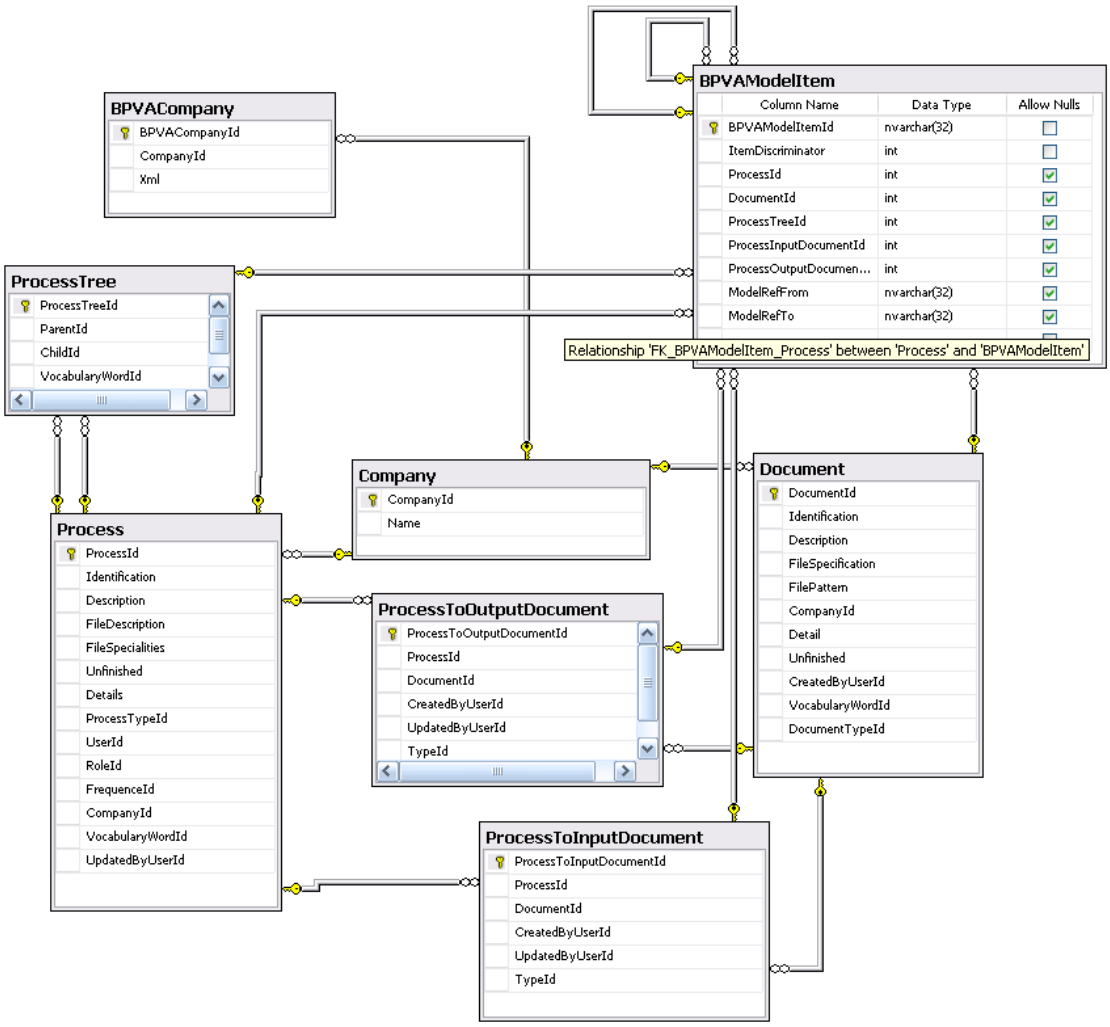
Další třídy diagramu jsou stejně pojmenovány, jako třídy z diagramu modelu. Třídy je možno vytvořit pouze pro konkrétní instanci třídy z modelu. Protože v nástroji PI nemáme informaci o grafické podobě modelu, a ani neplánujeme takovéto informace do nástroje zadávat, musíme informaci definovat až při exportu výchozími hodnotami doporučenými v dokumentaci BP-VA.



Obr. 13: Diagram tříd grafické reprezentace modelů

5.4.2 Rozšíření datového modelu

Rozšíření datového modelu bylo nedílnou součástí implementace komunikace. Byla rozšířena jedna tabulka a byla přidána nová tabulka. Do tabulky „Company“ byl přidán nový sloupeček textového typu s velkým počtem znaků (alespoň 10^8 znaků). Nová tabulka obsahuje sloupec primárního klíče o délce 32 znaků pro možnost ukládání identifikátoru GUID. Tato tabulka je provázána cizími klíči na tabulky třídy, které jsou součástí komunikace. Sloupečky ModelRefFrom a ModelRefTo jsou doplněny pro informaci provázání vazeb v modelu komunikace. Detaily datového modelu můžeme najít na obrázku níže (viz Obr. 14)



Obr. 14: Rozšíření modelu o tabulky pro komunikaci s BP-VA

5.5 Testování

Testování aplikace probíhalo na několika úrovních. Postupně si je všechny představíme a zhodnotíme výsledky, kterých bylo dosaženo. V první podkapitole se zaměříme na výsledky programového testování pomocí NUnit frameworku. Dále si představíme výkonnostní testy webového nástroje. V poslední podkapitole uvedeme několik poznatků z uživatelského testování aplikace.

5.5.1 Test Driven Development

V projektu ProcessInspector.Test bylo implementováno testování jednotlivých částí nástroje. Projděme si výsledky, jakých jsme dosáhli.

Pro testování vrstvy pro přístup k datům nástroje můžeme říci, že nám testy nijak výrazně nepomohly. Služba NHibernate pro mapování objektů do relační databáze nám většinu chyb nahlásila již při pokusu o kompilování mapovacího souboru. Rozhraní pro přístup k datům nebylo v průběhu vývoje výryzně rozšiřováno a tak počet nových metod pro testování nebyl velký. Můžeme říci, že úspěšně zkompilovaný mapovací soubor odpovídal bezchybné funkčnosti objektu.

Pro testování modelu aplikace bylo vytvořeno několik hlavních testů pro nejdůležitější objekty. Můžeme říci, že testování správnosti modifikace stavu objektů, které v aplikaci nejsou měněny je zbytečné. Jakmile bude do aplikace přidán požadavek na modifikaci dalších objektů, budou přidány testy, které požadovanou funkčnost zkontrolují.

Testování presenteru aplikace bylo nejnáročnější částí testování. Ve většině případů bylo náročnější vymyslet prostředí pro testování, než byla implementace samotné funkčnosti presenterů. Po úspěšném sestavení testů ale můžeme konstatovat jejich přínos. Rozšiřování modelu nemělo při vývoji větší vliv na podobu testů. Úspěšné provedení testů nám přitom garantovalo zachování správné funkčnosti pro již implementované fuknce.

Poslední testy kontrolují funkčnost View. Pro tyto testy můžeme uvést, že kontrolování správnosti zobrazení UI je nejvíce obtížná část. V tomto směru jsme udělali velký kompromis a za správnou kontrolu považujeme úspěšně vypsanou stránku, tedy že je v provozu.

5.5.2 Testování výkonnosti

Nástroj byl podroben několika výkonovým testům. Testy byly zaměřeny na části systému, které považujeme za kritické. Těmito částmi je počet procesů, dokumentů a doba nutná pro komunikaci s nástrojem BP-VA.

Do databáze nástroje jsme tedy naimportovali konkrétní projekt z praxe. Bylo naimportováno 324 procesů, 375dokuentů, 240 vazeb mezi vnořenými procesy a 754 vazeb vstupních a výstupních dokumentů. Následně byla společnost 100 krát duplikována pomocí databázových skritpů. Testy probíhaly měřením času zpracování požadavku na serveru. Z testování vyšel nástroj jako plně funkční a použitelný v běžném provozu.

5.5.3 Uživatelské testování

Nástroj byl nasazen na reálný projekt v praxi. Testování uživateli právě probíhá. Z prvních připomínek můžeme vyvodit následující poznatky.

Rozvoj nástroje nástroje v oblasti jednoduchosti použití bude nadále probíhat. Tento fakt je převážně dán obecnou problematikou modelování a ne tak nástrojem samotným. V nástroji byla implementována řada vylepšení, která uživatelům poskytují přehlednější pohled na modelovaný projekt.

Z uživatelského testování taky vyplynuly nové požadavky na rozšiřování systému. Hlavní oblastí je rozšiřování slovníkových pojmů. Testy odhalily problém běžných uživatelů vybrat si mezi správnými pojmy.

Druhým bodem dalšího rozšiřování bude zjištění relevantních doplňujících otázek. Musíme konstatovat, že jsme se při nasazení nástroje snažili docílit bohatého slovníku frází. Byly tak vytvořeny otázky bez informačního přínosu pouze s cílem vytvořit bohatý slovník.

6 Závěr

V úvodu práce byl zmíněn požadavek na nástroje pro vývoj informačních systémů. Představili jsme si dvě implementace nástrojů. První nástroj Process Inspector jsme zhodnotili jako jednoduchý, snadno použitelný. Museli jsme ale také konstatovat několik nedostatků, které bránili uživatelům jeho používání. Druhá implementace Business Process Visual Architect je komplexním řešením pro modelování systémů určený pro systémové analytiky. Nástroj přichází s dalšími výhodami, jako jsou vytváření zkratek pro nejčastěji používané funkce nebo značně propracované grafické zobrazování modelů.

Cílem práce je implementace nového řešení, které skloubí výhody obou nástrojů. Byly popsány konkrétní požadavky, které byly konzultovány s odborníky z praxe. Z požadavků vyplynulo, že není třeba měnit dosavadní architekturu nástroje Process Inspector, ale hlavním bodem zlepšení nástroje byla použitelnost uživateli. Dalším důležitým bodem pro implementaci nového řešení je volba technologií Microsoft SQL Server, ASP.NET.

Obsah práce i samotná implementace se odrazila ve snaze autora o seznámení se s návrhovými vzory, jejich možným variantami a praktickým využití. Technologie a návrhové vzory, se kterými jsme se seznámili, vycházejí z požadavků a návrhu řešení. Cílem bylo představení nejdůležitějších částí použitých v implementaci.

Po sestavení požadavků na základě analýzy bylo implementováno nové řešení nástroje Process Inspector. Nástroj byl vyvíjen metodou řízení vývoje testy (TDD), jež umožnila testovat nástroj z funkčního hlediska již v průběhu vývoje. Implementovaný nástroj byl následně testován z hlediska rychlosti a stability. Ve všech kriteriích byl nástroj shledán plně použitelným. Další testování, které neustále probíhá, je nasazení v reálném provozu. Za pomoci firmy Allium s.r.o. byl nástroj nasazen na konkrétní projekt, který umožní vyhodnocení výsledků testování použitelnosti.

Z prvních poznatků můžeme konstatovat, že nástroj splnil požadavky, které jsme na něj kladli. V blízké budoucnosti budou zapracovány připomínky z testování v reálném provozu. Další rozvoj aplikace vidíme v rozšíření komunikace s nástrojem Business Process Visual Architect. Neméně důležitý bod, je budování a rozšiřování databáze slovníku a otázek, ve kterých nástroj nabízí podměty pro další návazné práce v této problematice.

Literatura

[1] Fiedler, Z.: Nástroj na podporu vývoje interaktivních aplikací [Diplomová práce 2006] Brno, VUT v Brně, Fakulta informačních technologií, Božetěchova 2

[2] Allium, s.r.o., Náměstí republiky 366/1, 614 00 Brno
PI2BPVA, 2007

[3] Object Management Group, Business Process Modelig Notation Specification
http://www.bpmn.org/, 2006

[4] Visual Paradigm, Business Process Visual Architect User's Guide
http://www.visual-paradigm.com/documentation/, 2007

[5] Extensible Markup Language (XML)
http://www.w3.org/XML/, 2003

[6] Retirement note for Model View Presenter Pattern
http://martinfowler.com/eaaDev/ModelViewPresenter.html, 2006

[7] Inversion of Control Containers and the Dependency Injection pattern
http://martinfowler.com/articles/injection.html, 2004

[8] Apache log4net
http://logging.apache.org/log4net/index.html, 2004

[9] NHibernate for .NET
http://www.hibernate.org/343.html, 2007

7 Přílohy

A. Průvodce instalací

Hlavním předpokladem využívání nástroje je instalace. Nástroj byl implementován pro technologii ASP.NET, u které předpokládáme instalaci na platformě operačního systému Windows. Ujistíme se, že máme nainstalovány následující programy:

- Internetová informační služba 6.0 (IIS)
- Microsoft .Net Framework 2.0
- MSSQL Server 2005

Instalaci databáze provede ve třech krocích

1. Na MSSQL serveru vytvoříme novou databázi.
2. Vytvoříme nového uživatele a nastavíme mu oprávnění na novou databázi.
3. Připojíme se k serveru k nově vytvořené databázi a spustíme soubor „install.sql“ z instalačního adresáře.

Instalaci aplikaci provedeme provedením následujících kroků.

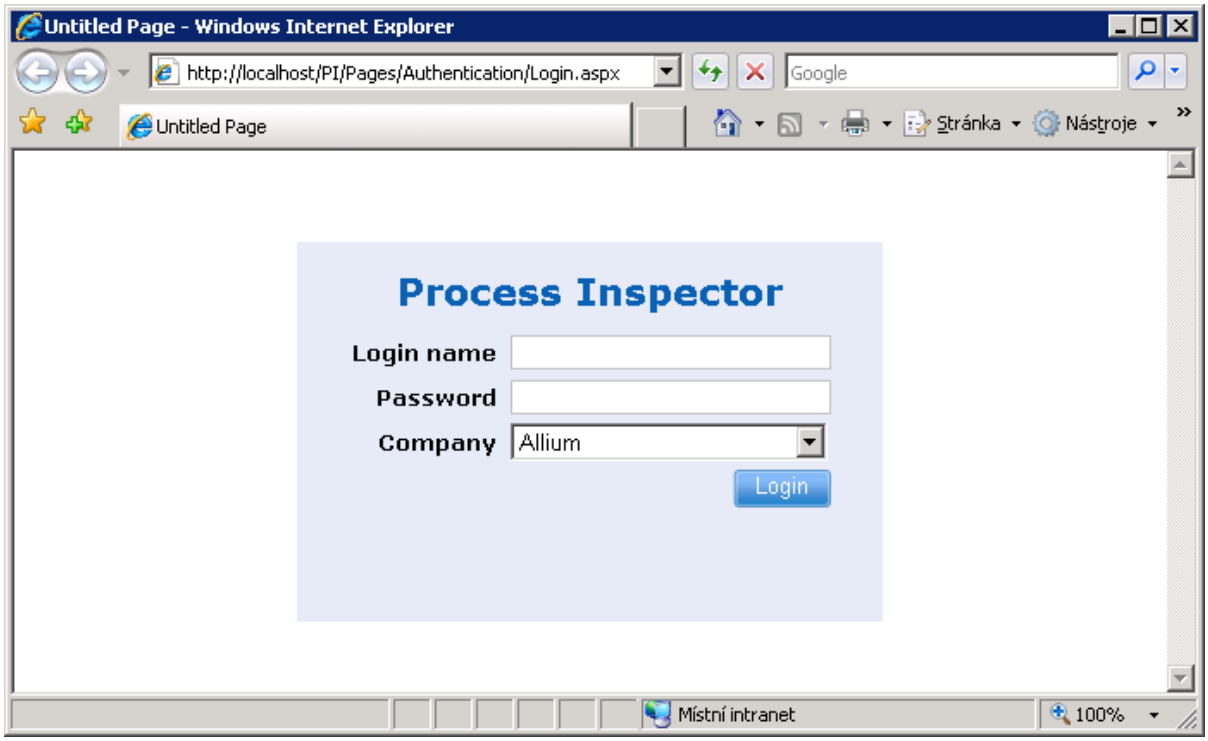
1. Vytvoříme si adresář, kam hodláme aplikaci nainstalovat.
2. Zkontrolujeme oprávnění na čtení a zápis pro uživatele, pod kterým bude aplikace spuštěna.
3. Spustíme Internetovou informační službu a založíme nový webový adresář.
4. Pro vytvořenou položku zkontrolujeme následující nastavení ve vlastnostech. V menu zvolíme Akce a Vlastnosti.
    - Na záložce Domovský adresář stiskneme tlačítko Konfigurace. V seznamu Mapování aplikace přidáme příponu „.pi“ (program volíme knihovnu aspnet_isapi.dll nainstalovaného Microsoft .NET Frameworku 2.0)
    - Na záložce ASP.NET se ujistíme, že je vybrána volba 2.0
5. Nastavíme konfigurační soubory aplikace. První konfigurační soubor pro WinsdorContainer se jmenuje components.config z adresáře Config. Zde změníme cestu ke konfiguračnímu souboru pro NHibernate editací obsahu uzlů <sessionFactoryConfigPath>. Konfiguraci připojení do databáze nastavíme v souboru nhibernate.config ve stejném adresáři. Zde musíme nastavit connectionstring a nastavit výchozí schema pro databázi. Volitelnou částí je nastavení logování. Ve výchozím nastavení bude aplikace zapisovat chybové hlášení do souboru Error.log.
6. Aplikací můžeme spustit zadáním adresy do prohlížeče.

B. Uživatelská příručka

Uživatelská příručka programu Process Inspector verze 1.0 (dále jen nástroj). V této příručce jsou vysvětleny základní operace a postupy práce z pohledu koncových uživatelů. Předpokládanými znalostmi je základní práce na PC.

B.1. Spuštění a přihlášení

Pro spuštění aplikace je nutné mít nainstalovaní prohlížeč webových stránek. Nástroj byl testován na běžně využívaných prohlížečích (IE 6, IE7, FireFox 2.0.0.14, Opera, Safari). Přihlašovací body aplikace byly navrženy dva. První pro přímé přihlášení ke společnosti, druhé s možností výběru společnosti. Na obrázku níže (viz Obr. 15) vidíme přihlášení s možností volby společnosti. Zadáme přihlašovací jméno a příjmení.



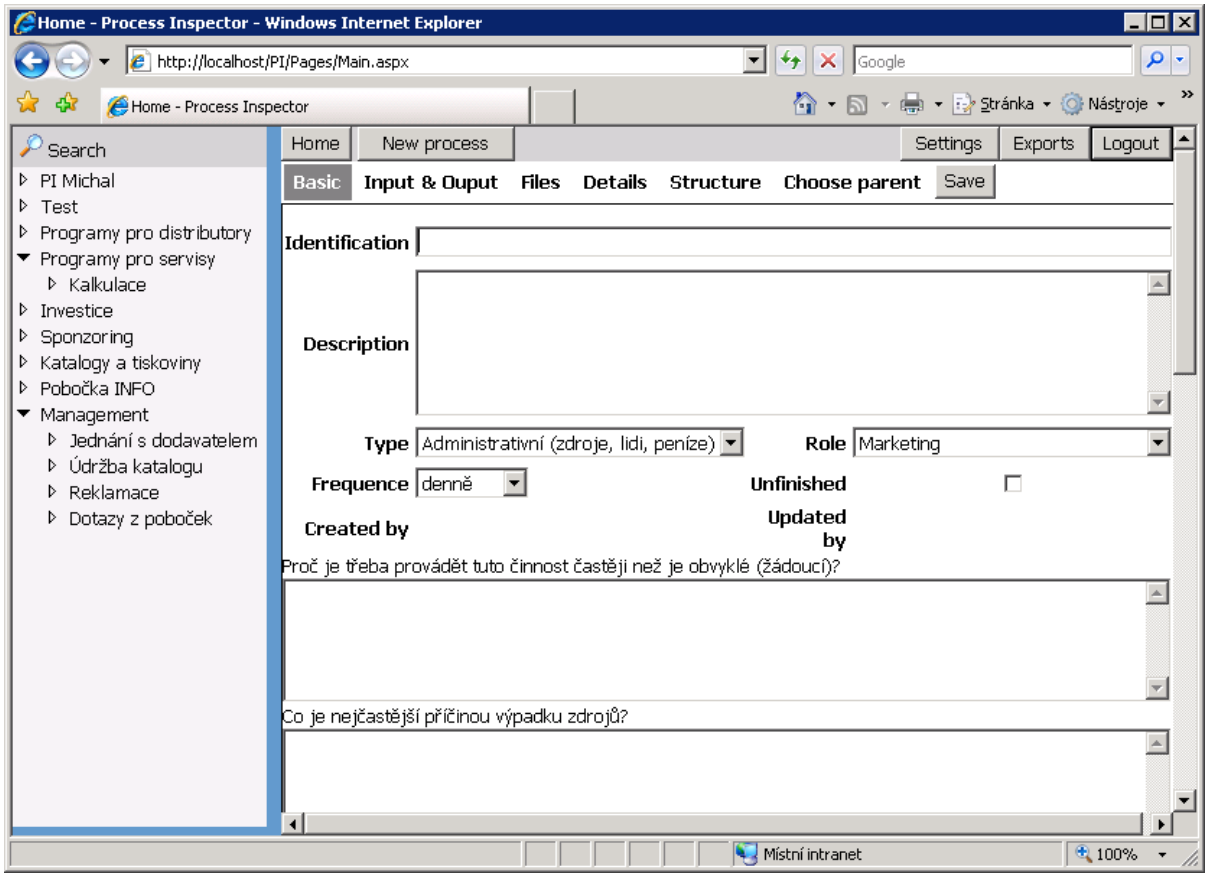
Obr. 15: Přihlašovací okno včetně volby společnosti

Poznamenejme, že přístup ke stránkách aplikace je chráněn autentizačním modulem v ASP.NET. Ten všechny neautorizované požadavky přesměruje na přihlašovací stránku „Login.aspx“ bez parametru názvu společnosti. V takovém případě se do aplikace nepřihlásíme.

B.2. Vytvoření procesu

S cílem vytvoření přehledného nástroje byla funkční tlačítka umístěna do horní části obrazovky. Na hlavní stránce nástroje stiskneme tlačítko „New process“. Zobrazí se nám stejný formulář jako pro

editaci existujícího procesu. Velké množství informací ukládaných o procesu mělo za následek jejich rozdělení do skupin. Mezi skupinami se můžeme přepínat pomocí záložek na straně klienta. Na straně klienta je také filtrován seznam otázek k procesu podle jeho typu. Využíváním klientského skriptování se snažíme posunout uživatelské rozhraní na vyšší úroveň.



Obr. 16: Hlavní okno aplikace s detailem procesu

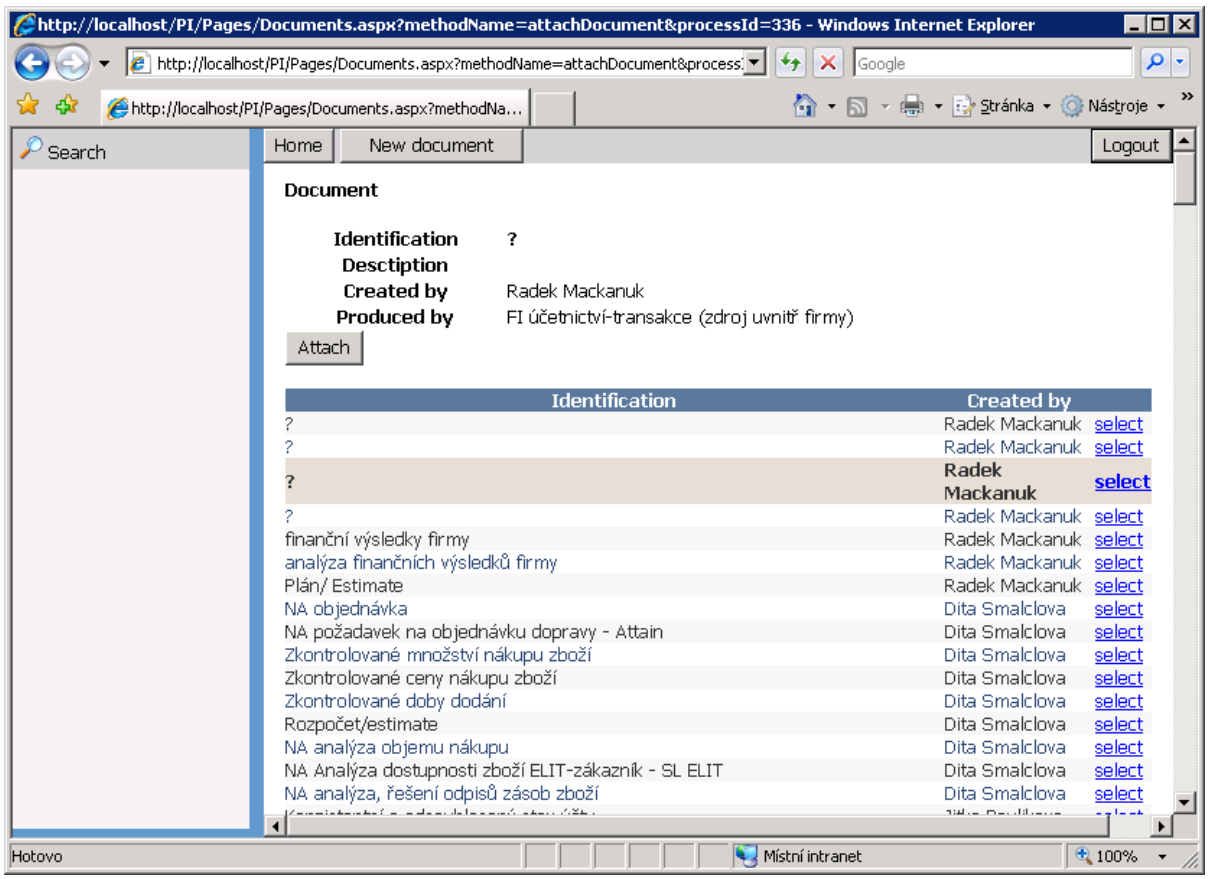
B.3. Smazaní procesu a dokumentu

Tato operace není v nástroji implementována. Je to jeden z požadavků přejatých z minulé verze. Důvod, proč nejde proces nebo dokument smazat je podmět, za kterým byli vytvořeni. I když se můžou některé procesy nebo dokumenty zdát zbytečné, jejich uchování považujeme za přínosné. Abychom byli schopni odlišit aktuálně nepotřebné položky, byla stanovena konvence pro jejich pojmenování. Název položky je znak otazník.

B.4. Připojení dokumentu

Připojení dokumentu k procesu provedeme na záložce Input & Output detailu procesu. Stiskneme tlačítko Attach Document a nástroj nám zobrazí seznam všech dokumentů společnosti. V seznamu vidíme název dokumentu a jméno uživatele, který jej vytvořil. Výběr provedeme na konci řádku u

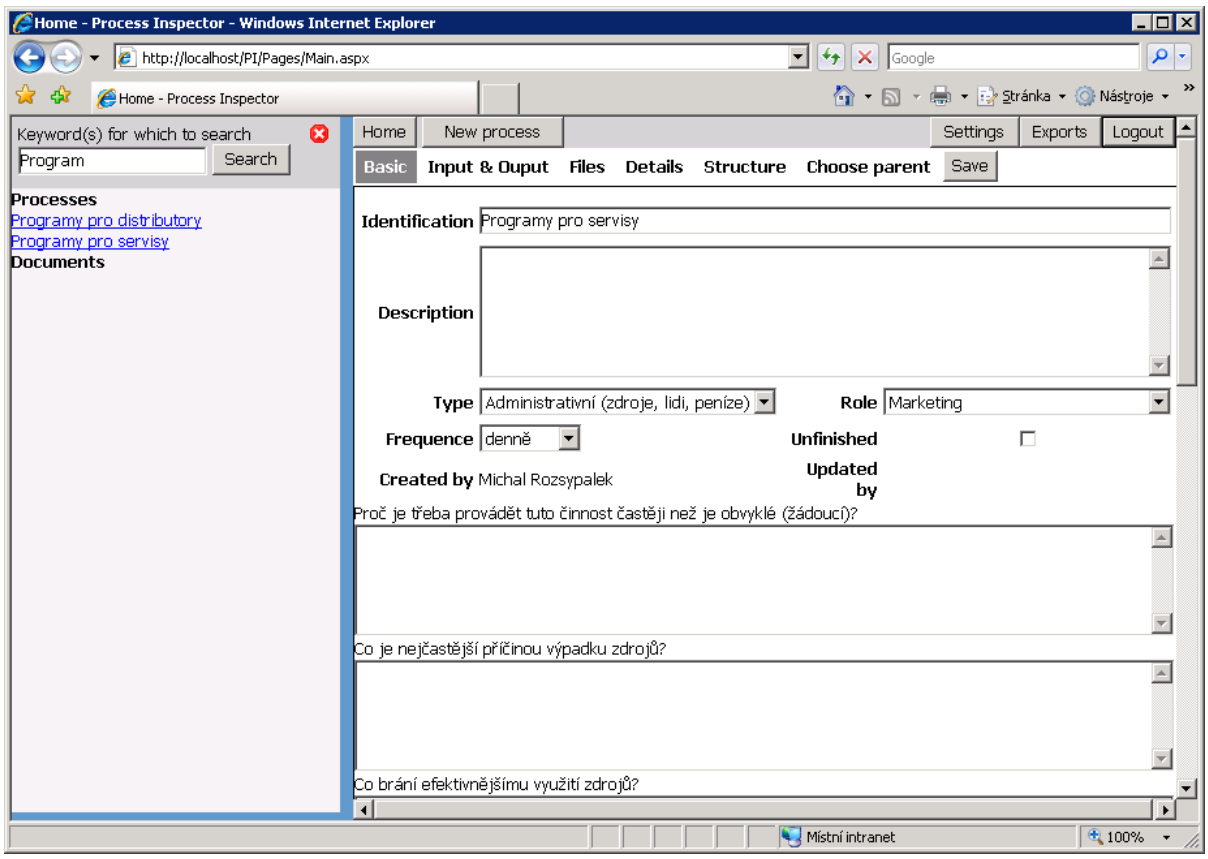
příslušného dokumentu. V záhlaví stránky jsou nám zobrazeny doplňující informace o dokumentu. Tlačítkem Attach připojíme dokument k procesu. Strom procesů není zobrazen záměrně.



Obr. 17: Připojení dokumentu

B.5. Vyhledávání

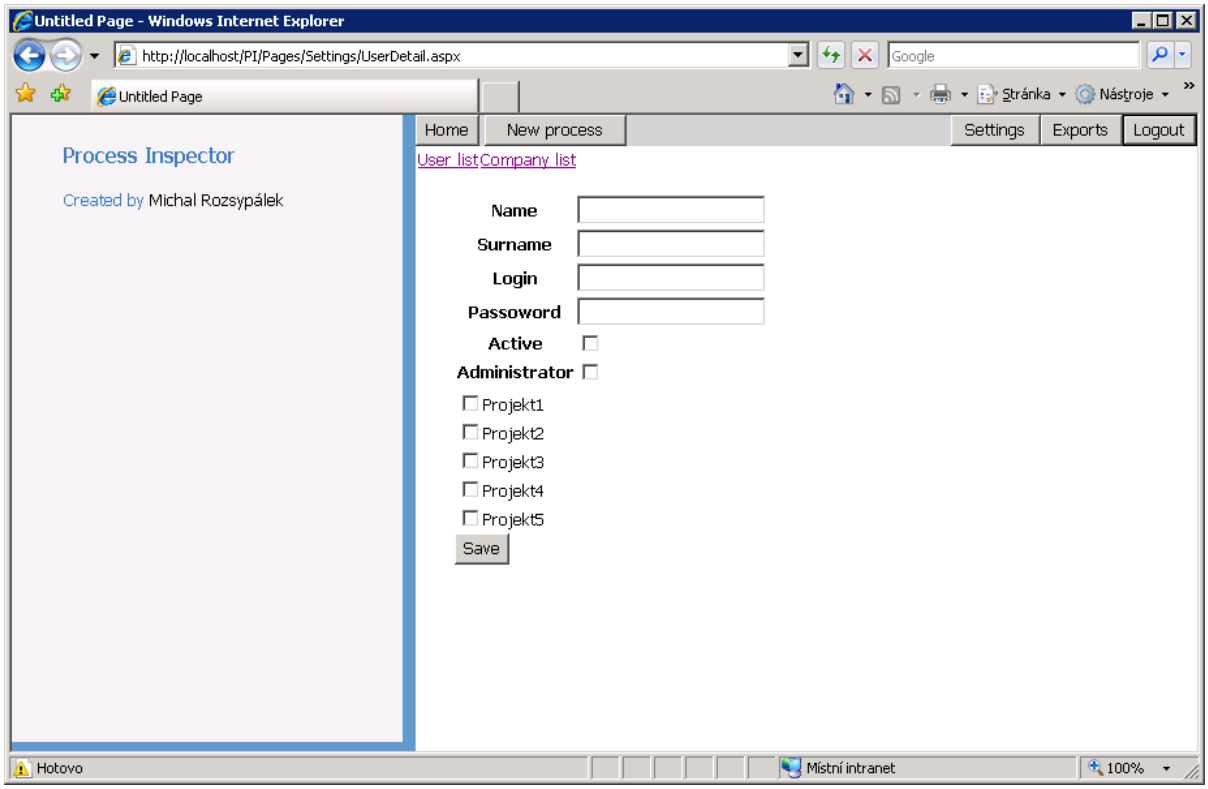
Vyhledávání je jednou z nejdůležitějších vlastností dobrého nástroje. V levé části obrazovky klikneme na obrázek lupy. Zobrazí se nám pole pro vložení hledané fráze. Nástroj vyhledává v identifikaci procesů a dokumentů současně. Zadáme frázi a stiskneme Search. Výsledek hledání v dokumentech a procesích je zobrazen odděleně. Kliknutím na identifikaci procesu nebo dokumentu zobrazíme jeho detail.



Obr. 18: Vyhledávání v dokumentech a procesech

B.6. Přidání uživatele

Poslední z nejdůležitějších operací běžného používání nástroje zbývá přidání a odebrání uživatele. Na hlavní stránce nástroje stiskneme tlačítko Settings. Na další stránce klikneme na odkaz User list a ještě jednou klikneme, tentokrát na tlačítko New user nebo příkaz Edit u příslušného řádku uživatele. Na formuláři máme možnost zadat základní informace o uživateli a zadat jeho oprávnění na projekty.



Obr. 19: Formulář detailu uživatele