VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

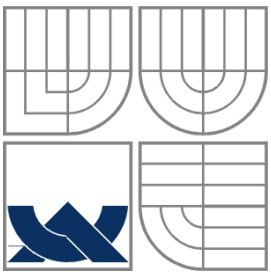
RÁMEC PRO RYCHLÝ VÝVOJ GUI KLIENTSKÝCH APLIKACÍ POST-RELAČNÍCH DATABÁZÍ

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

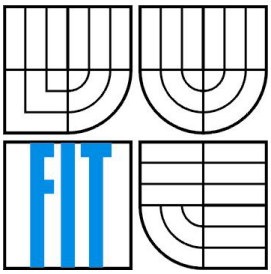
AUTOR PRÁCE
AUTHOR

BC. LUKÁŠ MAREK

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

RÁMEC PRO RYCHLÝ VÝVOJ GUI KLIENTSKÝCH APLIKACÍ POST-RELAČNÍCH DATABÁZÍ

A FRAMEWORK FOR RAPID DEVELOPMENT OF GUI CLIENTS OF POST-RELATIONAL DATABASES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. LUKÁŠ MAREK

VEDOUCÍ PRÁCE
SUPERVISOR

RNDR. MAREK RYCHLÝ PH.D.

BRNO 2013

Abstrakt

Tato práce se zabývá návrhem a implementací rámce pro rychlý vývoj aplikací pracujících s multimediálními, prostorovými a časovými databázemi. Seznamuje čtenáře s použitými technologiemi a návrhem samotného rámce, který obsahuje tři grafické komponenty, které je možné použít v Java aplikacích používajících grafickou knihovnu Java Swing.

Abstract

This thesis deals with design and implementation of a framework for rapid application development that use multimedia, spatial and temporal databases. Introduces readers with applied technologies and framework design. Framework design contains three graphical components that can be used in Java applications using Java Swing graphics library.

Klíčová slova

rámec, Oracle, post-relační, databáze, multimediální, prostorová, časová, Java, rychlý vývoj aplikací, RAD, MVC

Keywords

framework, Oracle, post-relational, database, multimedia, spatial, temporal, Java, rapid application development, RAD, MVC

Citace

Lukáš Marek: Rámec pro rychlý vývoj GUI klientských aplikací post-relačních databází, diplomová práce, Brno, FIT VUT v Brně, 2013

Rámec pro rychlý vývoj GUI klientských aplikací post-relačních databází

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením RNDr. Marka Rychlého, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Marek
21. 5. 2013

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce panu RNDr. Marku Rychlému, Ph.D., který se mnou diplomovou práci konzultoval.

Obsah

1 Úvod

Tato diplomová práce pojednává o možnostech vytvoření rámce pro rychlý vývoj aplikací (Rapid Application Development, RAD) s grafickým uživatelským rozhraním (GUI). Rámec má pracovat s post-relačními databázemi a to konkrétně s multimediálními, prostorovými a temporálními. Tato diplomová práce vznikla proto, aby při vývoji aplikací pracujícími s těmito databázemi bylo možné co nejrychleji vytvořit grafické rozhraní a rozhraní pro práci s daty a při vývoji se zaměřit hlavně na komunikaci s databází a práci s databázovými daty. Diplomová práce popisuje jednotlivé technologie použité pří návrhu a později i při implementaci, teoretický návrh rámce a samotnou implementaci.

Druhá kapitola obsahuje informace o všech použitých technologiích, jsou zde popsány post-relační databáze a jazyk SQL pro komunikací s nimi, rychlý způsob vývoje aplikací (Rapid Application Development). Dále je zde popsán jazyk Java, který je použit pro implementaci rámce a vývojové prostředí použité jak pro implementaci tak následně i pro import nových komponent vytvořených v rámci. V poslední podkapitole je popsán návrhový vzor MVC, který používají nové komponenty.

Třetí kapitola popisuje teoretický návrh rámce a jeho nových komponent. Je popsán způsob vytváření nových komponent v rámci Swing hierarchie a způsob vytvoření nových grafických komponent tak, aby je bylo možné vkládat v GUI Builderu vybraného vývojového prostředí. Dále je popsán návrh jednotlivých komponent a rozhraní každé komponenty, které bude nutné implementovat v klientských aplikacích.

2 Teoretický úvod

Tato kapitola popisuje specifikaci rámce, příklad kdy daný rámec lze použít na zjednodušení implementace a popis všech použitých technologií a postupů při návrhu a implementaci rámce.

2.1 Specifikace

Rámec pro rychlý vývoj aplikací je určen na urychlení vývoje aplikací s grafickým uživatelským rozhraním s možností zobrazování a editace dat z post-relačních databází.

Jako příklad je možné uvést komplexní aplikaci na zobrazování a editaci dat z období stěhování národů. Aplikace by mohla sloužit jako studijní materiál, který podrobně zobrazí nejen výpis jednotlivých přesunů národů, ale také veškeré informace co se dané historické látky týkají. Aplikace by byla rozdělena na tří části. V první části by byla zobrazena mapa osídlených území s možností časového posunu, která by interaktivně zobrazovala všechny přesuny podle nastaveného časového rozmezí. V druhé části by byly zobrazeny jednotlivé objekty zobrazené na mapě, ale byli by zobrazeny a uspořádány v tabulce podle časové příslušnosti s možností filtrace, aby bylo možné sledovat časové posloupnosti a editovat časové údaje. Ve třetí části by byli k jednotlivým objektům na mapě možně ukládat a následně zobrazovat multimediální data. Tato část by detailněji popisovala dané objekty a poskytovala dodatečné informace ve formě obrázků, videí a dokumentů.

Každá z částí aplikace by pracovala s jiným typem databáze a proto je možné aplikaci rozdělit do tří víceméně nezávislých částí. Pro usnadnění implementace podobných aplikací a hlavně pro urychlení vývoje jednotlivých částí má sloužit rámec, který bude výstupem této diplomové práce. Rámec bude implementovat třídy pro jednotlivé části, které mohou sloužit jako základ při vývoji. Jednotlivé třídy pro komponenty budou implementovat grafické zobrazení, logiku a budou nabízet rozhraní pro třídu, která bude načítat data za databáze.

2.2 Post-relační databáze

Post-relační databáze jsou databázové systémy, které už nevystačí se základním relačním schématem a bez přímé podpory na implementační úrovni je zpracování jiných dat neefektivní. Jedná se tedy o databázový systém rozšířený o nějakou specializaci na databázové úrovni [6].

Příklady post- relačních databázových systémů:

- Objektově orientované databáze - rozšířeny o objektový model dat a vazby
- Multimediální databáze - rozšířeny o funkce pro práci s multimediálním obsahem
- Prostorové databáze - rozšířeny o práci s prostorovými objekty a vztahy mezi nimi
- Deduktivní databáze - rozšířeny o funkce pro analýzu dat

- Temporální databáze - rozšířeny o temporální logiku
- Aktivní databáze - rozšířeny o aktivní pravidla

2.2.1 Databáze Oracle

Účelem databáze je ukládat a poskytovat informace. Databázový server je klíčový při správě dat. Obecně platí, že server spolehlivě spravuje velké množství dat ve víceuživatelském prostředí a umožňuje paralelní přístup uživatelů k datům. Databáze zároveň poskytují vysoký výkon při čtení dat, zabraňují neoprávněnému přístupu a poskytují efektivní řešení při obnovování dat, pokud došlo k selhání systému. Databáze Oracle je objektově-relační databázový systém vyvinutý společností Oracle Corporation. Poskytuje moderní multiplatformní databázový systém s velice pokročilými možnostmi zpracování dat, vysokým výkonem a snadnou škálovatelností [2].

2.2.2 Multimediální databáze

Multimediální databáze je databázový systém, který dokáže spravovat multimediální data. Multimediální data jsou nestrukturovaná data vyznačující se velkým objemem (jedna položka může mít desítky až tisíce megabajtů). Dělíme je na vizuální (2D obrázky, 3D modely, video, textové a smíšené dokumenty) a audio data. Tato data obsahují často také metadata (např. Exif u obrázků, ID3 tag u MP3). Databáze při vkládání různých typů dat analyzuje a uloží informace, podle kterých lze vyhledávat a editovat vložená data. Na správu těchto informací slouží SŘMMBD (systém řízení multimediální báze dat), jedná se o programovou vrstvu spravující různé typy dat odpovídající různým médiím potenciálně reprezentované v různých formátech.

Pro úspěšnou činnost by měl SŘMMBD poskytovat tyto možnosti:

- jednotný způsob dotazování nad daty různých typů, v různých formátech z různých datových zdrojů (relační databáze, objektové databáze, souborový systém apod.) a typů médií
- získat a prezentovat spojitě mediální objekty (např. video) z paměťového zařízení
- uživateli definovat strukturu a požadovaný obsah výsledku dotazu. Výsledek dotazu je obecně opět nějaký multimediální objekt
- doručit prezentaci výsledku dotazu klientovi (aplikace, výstupní zařízení) způsobem, který zajišťuje dostatečně kvalitní prezentaci (např. výsledkem dotazu je proud video a audio dat)

Multimediální databáze disponuje funkcemi na správu multimediálních dat. Zaručují konzistenci, souběžnost a integritu dat, tyto funkce jsou nutné pro bezpečnost a dostupnost dat. Poskytují funkce pro manipulaci s multimediálními daty (např. změna velikosti, transformace). Dotazování a získávání informací z kolekcí uložených dat může probíhat podle popisu, podle multimediálního obsahu nebo podle vlastností objektů. Příkladem databázového dotazu v takovémto systému je nalezení nejpodobnějších obrázků k danému obrázku (tzv. podobnostní vyhledávání).

2.2.3 Prostorová databáze

Prostorové (spatial) databázové systémy, přesněji databázové systémy s podporou ukládání, zpracování a manipulace s prostorovými daty, jsou případem pokročilého databázového systému. Jejich DDL a DML (viz. kapitola 2.2.5) zahrnují prostorové datové typy. Prostorové datové typy jsou podpořeny i na implementační úrovni, takže je možné efektivně provádět operace indexace, vyhledávání, spojování, atd.

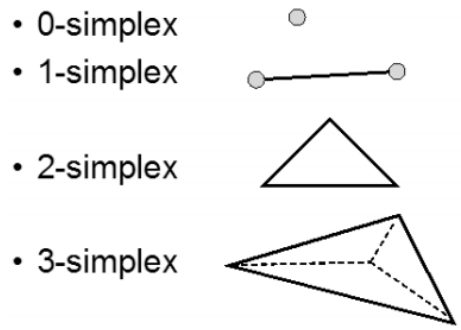
Prostorové databáze jsou především databázové systémy schopné spravovat data, která se váží k určitému prostoru, bez ohledu na to, jak veliký ten prostor je. Z toho je možné odvodit i charakteristiku podpůrné technologie, je to schopnost spravovat velké množství relativně jednoduchých geometrických objektů. V prostorových databázích tedy nalezneme množiny entit z určitého prostoru , u kterých je zřejmá identifikace, umístění a vztah k okolí.

V prostorové databázi je třeba ukládat geometrické objekty, skupiny objektů a také polohu vůči jiným objektům. Mezi základní objekty patří:

- bod - města, křižovatky
- lomená úsečka - řeky, silnice, vedení
- uzavřená lomená úsečka (polygon) - ohraničení oblastí
- oblast (vyplněná uzavřená lomená úsečka) - les, jezero, město

Tyto objekty je potřeba ukládat v databázi velice přesně a protože v databázi, a obecně v počítačích, lze ukládat desetinná čísla pouze do určité přesnosti musí se data ukládat jiným způsobem. Základní přístupy ukládání dat v prostorových databázích, které tento problém řeší jsou simplexy a úplné deskriptory.

Způsob ukládání pomocí simplexů je založeno na skládání jednoduchých geometrických entit pro vytvoření složitějších. Simplex je nejmenší nevyplněný objekt dané dimenze, 0-simplex je bod, 1-simplex úsečka, 2-simplex trojúhelník, 3-simplex je čtyřstěn atd., jednoduché simplexy jsou vidět na obrázku 2.1. Každý d-simplex se skládá z d+1 simplexů rozměru d-1. Takové tvořící elementy se potom nazývají styky (faces). Kombinace simplexů do složitějších struktur je povolena jen tehdy, pokud průnik libovolných dvou simplexů je styk.



Obrázek 2.1: Simplexy

Způsob ukládání pomocí úplných deskriptorů (realms) je založen na kompletním popisu modelované oblasti. Deskriptory (realms) jsou vlastně jakýmsi souhrnným popisem všech objektů v databázi, jedná se o množinu bodů, úseček, případně vyšších celků, které mají tyto vlastnosti:

- každý (koncový) bod je bodem sítě
- každý koncový bod úsečky (složitějšího útvaru) je bodem sítě
- žádný vnitřní bod úsečky (složitějšího útvaru) není zaznamenán v síti
- žádné dvě úsečky (složitější útvary) nemají ani průsečík ani se nepřekrývají

Deskriptory však musejí na implementační úrovni řešit problémy s číselnou reprezentací a to se ne vždy daří zcela uspokojivě, aplikační data často obsahují průsečíky vnitřních bodů grafických elementů (úseček), ty však ale nemusí být v síti. Jednoduchá řešení v tomto případě neexistují.

2.2.4 Temporální databáze

Temporální databáze je druh databáze, která mimo statických dat dokáže uchovávat také temporální data bez ztráty jejich temporálního charakteru. Databáze obsahuje časové údaje označované jako valid-time (čas platnosti dat vzhledem k reálnému času) a transaction-time (čas, kdy byla data přítomna v databázi). Oba tyto časy tvoří bitemporální model dat.

V temporální databázích jsou data, která obsahují časovou složku. Při ukládání takovýchto dat do relační databáze je třeba tuto časovou složku modelovat jako další atributy v n-tici a pro relační databázi nemá tato časová složka žádný speciální význam. Naproti tomu u temporální databáze jsou časové složky záznamu definovány zvlášť a databáze s nimi pracuje odděleně od běžných atributů n-tice. U temporálních dat je možné evidovat dva typy časové informace. Prvním typem je čas platnosti dat. Tento čas určuje, v jakém okamžiku nebo v jaké době byla data platná vzhledem k reálnému světu. Druhým typem je čas transakce. Tento čas určuje, kdy byla daná data přítomna v databázi

Temporální databáze nacházejí využití v mnoha odvětvích lidské činnosti, kde je vhodné nebo dokonce nutné uchovávat časové a historické informace o datech. Jedná se například o:

- finanční data - Bankovní ústavy evidují klientské účty. Stavy účtu a prováděné transakce jsou temporálními daty u kterých je nutné udržovat časové údaje. Musí být možné prohlížet historii stavu účtu nebo výpisy transakcí za určité časové období.
- medicínská data - Údaje o pacientech jako prodělané nemoci, průběh teploty pacienta nebo dávkování léků jsou temporálními daty u kterých je třeba udržovat informaci o časech.
- katastrální data - Je třeba evidovat vlastníky pozemků, přepisy, změny katastrálních map. Všechny tyto údaje jsou vázány na data a časy.
- meteorologická data - Teplota, tlak vzduchu, vlhkost vzduchu se mění v závislosti na čase a je třeba tyto údaje nějak uchovávat.

2.2.5 Jazyk SQL

SQL (Structured Query Language) je databázový jazyk určený k definici a manipulaci s daty v relačních databázových systémech a pro správu relačních databázových systémů. Protože relační databáze neobsahují funkce pro práci specializovanými daty jako jsou multimediální data, prostorové objekty a časové údaje, je nutné rozšířit jazyk SQL o funkce a struktury pracující s těmito typy dat.

Jazyk SQL byl vytvořen v období 1970 až 1980. Stal se ANSI standardem (American National Standards Institute) v roce 1986 a ISO standardem (International Organization for Standards) v roce 1987. Během pár let se projevily některé nedostatky, jazyk byl několikrát upraven a byli přidány nové funkce. V roce 1992 vznikla verze SQL92, která je standardem dodnes. I přes standardizaci existují rozdíly v implementaci mezi nejpoužívanějšími databázovými systému, hlavně kvůli neúplné specifikaci a různým rozšířením jazyka.

SQL patří mezi tzv. deklarativní programovací jazyky, což znamená, že kód jazyka SQL nepíšeme v žádném samostatném programu, ale vkládáme jej do jiného programovacího jazyka. Se samotným jazykem SQL můžeme pracovat pouze v případě, že se terminálem připojíme na SQL server a na příkazový řádek bychom zadávali přímo příkazy jazyka SQL.

SQL se skládá z několika částí. První částí jazyka SQL je jazyk DDL (Data Definition Language). Jedná se o jazyk pro vytváření databázových schémat a katalogů. Způsob ukládání tabulek definuje jazyk SDL (Storage Definition Language). Třetí částí je jazyk VDL (View Definition Language), který je určen pro návrháře a správce. Poslední částí, je jazyk DML (Data Manipulation Language), který obsahuje základní příkazy INSERT, UPDATE, DELETE a SELECT. Jazyk DML je nejpoužívanější částí jazyka SQL [1].

2.3 Rapid Application Development

Rapid Application Development, dále jen RAD, je metodologie vývoje softwaru. Metodologie vývoje softwaru je pojem softwarového inženýrství, kterým označujeme framework používaný pro návrh, plánování a řízení procesu vývoje.

RAD je takový přístup k vývoji software, který používá minimální plánování ve prospěch rychlého vytváření prototypů. Samotný vývoj aplikací pomocí RAD je založen na iterativním vývoji a prototypování. Díky minimálnímu plánování před samotným vývoje je obecně vývoj rychlejší a hlavně umožňuje změny požadavků během vývoje.

V RAD se používají strukturované techniky a prototypování především ke stanovení uživatelských požadavků a návrhu finálního systému. Vývoj začíná s návrhem předběžných datových modelů a modelů podnikových procesů s využitím strukturovaných technik. V další fázi jsou požadavky ověřeny pomocí prototypů a případné zpřesnění požadavků aplikováno v datovém modelu. Tyto fáze se opakují iterativně. RAD vývojový proces může vést ke kompromisům ve funkčnosti a výkonu, výměnou za rychlejší vývoj a snadnější správu aplikací [7].

2.3.1 Historie

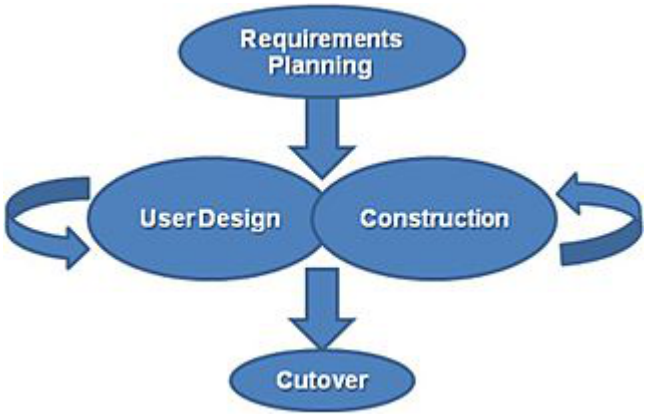
Termín Rapid Application Development (RAD) byl poprvé použít při vývoji a úspěšném nasazení softwaru v polovině roku 1970 firmou New York Telefon Co's Systems Development Center pod vedením Dana Gielana. Po sérii úspěšných implementací tohoto procesu, Gielan přednášel na různých fórech o metodice, praxi a přínosech tohoto procesu.

V roce 1990 James Martin zdokumentoval výklad metodologie v knize RAD, Rapid Application Development. Zanedlouho poté se začal termín používat v širším slova smyslu, zahrnoval celou řadu metod, jejichž cílem bylo urychlení vývoje aplikací, jako je například používání softwarových frameworků různých typů.

RAD byl reakcí na procesy vyvinuté v roce 1970 a 1980, jako je například strukturovaná systémová analýza a další vodopádové modely. První problém u předchozích metod byl ten, že vývoj aplikace trval tak dlouho, že se během vývoje změnily požadavky na systém a to vedlo k vytvoření nevyhovujícího nebo nepoužitelného systému. Dalším problémem bylo to, že fáze analýzy požadavků měla určit všechny kritické požadavky, a to se i v týmu s velmi zkušenými odborníky povedlo jen někdy [8].

2.3.2 Fáze RAD

Při vývoji pomocí metodologie RAD je nutné dodržet fáze vývoje. Model zobrazující dané fáze je na obrázku 2.2 [7][8].



Obrázek 2.2: Rapid Application Development Model

Fáze plánování požadavků (Requirements Planning phase) - kombinuje systémové plánování a fázi systémové analýzy z životního cyklu vývoje systému. Uživatelé, manažeři a zaměstnanci IT se dohodnou na potřebách podniku, rozsahu projektu a systémových požadavcích. Fáze končí když se tým dohodne na klíčových otázkách.

Fáze uživatelského návrhu (User design phase) - uživatelé v této fázi spolupracují se systémovými analytiky a navrhnou modely a prototypy, které představují všechny systémové procesy se vstupy a výstupy. RAD skupina používá nástroje jako Joint Application Development (JAD), které převedou uživatelské požadavky do pracovních modelů. Uživatelský návrh je interaktivní proces, který uživatelům průběžně umožní pochopit, upravit a nakonec schválit pracovní model systému, který splňuje jejich požadavky.

Fáze vývoje (Construction phase) - zaměřuje se na vývoj aplikace, podobně jako v životním cyklu vývoje systému, ale uživatelé se nadále účastní a můžou navrhovat změny nebo vylepšení. Úkoly této fáze jsou programování, kódování (vytváření vzhledu aplikace), integrace a testování systému.

Fáze zavádění (Cutover phase) – podobné poslední fázi v životním cyklu vývoje systému a to realizační fázi. Díky tomu, že je nový systém uveden do provozu mnohem dříve, tak je tato fáze ve srovnání s tradičními metodami kratší. Úkoly této fáze jsou konverze dat, testování, přechod na nový systém a školení uživatelů.

2.4 Java

Java je objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems. První verze byla představena v květnu roku 1995, v roce 2006 Sun Microsystems uvolnil zdrojové kódy Javy a Java je nyní vyvíjena jako open source pod licencí GNU General Public License (GPL).

2.4.1 Java SE

Java SE (Standard Edition) je široce používán platforma pro programování v jazyce Java. Je to platforma používaná k programování desktopových aplikací. Z praktického hlediska se skládá z virtuálního stroje, který musí být použit ke spuštění Java programů, společně se souborem knihoven nutných pro práci se souborovým systémem, sítí, grafickými rozhraními, atd. Je to vlastně Java, tak jak byla vyvíjena od první verze a postupně rozšiřována. Když firma Sun Microsystems zavedla termín platforma Java, bylo třeba původní kolekci API odlišit od ostatních verzí, proto vzniklo toto označení pro standardní destopovou platformu.

Jazyk Java byl vyvinut tak aby splňoval následující požadavky:

- jednoduchý, objektově orientovaný, známá syntaxe
- spolehlivý a bezpečný
- nezávislý na architektuře a přenositelný
- výkonný (rychlí při provádění)
- interpretovaný, s podporou vláken, dynamický

Java je díky velké podobnosti se syntaxí C/C++ velice jednoduchá. Byli odstraněny konstrukce, které programátorům způsobovaly problémy, a které umožňovaly vytvářet chybný kód. Naopak přibyla řada užitečných rozšíření. Java je, až na pár jednoduchých datových typů, plně objektová.

Java je určena pro psaní vysoce spolehlivého softwaru. Z tohoto důvodu neumožňuje některé programátorské konstrukce, které bývají častou příčinou chyb (např. správa paměti, příkaz goto, používání ukazatelů). Používá silnou typovou kontrolu (tzn. veškeré používané proměnné musí mít definovaný svůj datový typ). Kvůli bezpečnosti má vlastnosti, které chrání počítač v síťovém prostředí, na kterém je program zpracováván, před nebezpečnými operacemi nebo napadením vlastního operačního systému nepřátelským kódem.

Je navržena pro podporu aplikací v síti. Podporuje jednoduchou komunikaci po síti, složitější komunikaci na různých vrstvách síťového spojení, práci se vzdálenými soubory a v neposlední řadě umožňuje vytvářet rozsáhlé distribuované klientské aplikace a servery. Vytvořená aplikace běží na libovolném operačním systému nebo libovolné architektuře. Ke spuštění programu je potřeba pouze to, aby byl na dané platformě instalován správný virtuální stroj. Vedle zmíněné nezávislosti na architektuře je jazyk nezávislý i co se týká vlastností základních datových typů (je například explicitně určena vlastnost a velikost každého z primitivních datových typů). Přenositelností se však myslí pouze přenášení v rámci jedné platformy Javy (např. J2SE). Při přenášení mezi platformami Javy je třeba dát pozor na to, že platforma určená pro jednodušší zařízení nemusí podporovat všechny funkce dostupné na platformě pro složitější zařízení a kromě toho může definovat některé vlastní třídy doplňující nějakou speciální funkčnost nebo nahrazující třídy vyšší platformy, které jsou pro nižší platformu příliš komplikované.

Přestože se jedná o jazyk interpretovaný, není ztráta výkonu významná, neboť překladače pracují v režimu „just-in-time“ a do strojového kódu se překládá jen ten kód, který je opravdu zapotřebí. Správa paměti je realizována pomocí automatického Garbage collectoru, který automaticky vyhledává již nepoužívané části paměti a uvolňuje je pro další použití. To bylo v prvních verzích příčinou pomalejšího běhu programů. V posledních verzích je díky novým algoritmům rychlost ve většině případů dostatečná.

Java byla navržena pro nasazení ve vyvíjejícím se prostředí. Knihovna může být dynamicky za chodu rozšiřována o nové třídy a funkce, a to jak z externích zdrojů, tak vlastním programem. Při kompilování se místo skutečného strojového kódu vytváří pouze tzv. Java bytecode. Tento formát je nezávislý na architektuře počítače nebo zařízení. Program pak může pracovat na libovolném počítači nebo zařízení, který má k dispozici interpret Javy (virtuální stroj Javy). V pozdějších verzích Javy nebyl mezikód přímo interpretován, ale před prvním svým provedením dynamicky zkompilován do strojového kódu daného počítače. Tato vlastnost zásadním způsobem zrychlila provádění programů, ale výrazně zpomalila start programů. V současnosti se převážně používají technologie, které mezikód zpočátku interpretují a na základě statistik získaných z této interpretace později provedou překlad často používaných částí do strojového kódu včetně dalších dynamických optimalizací.

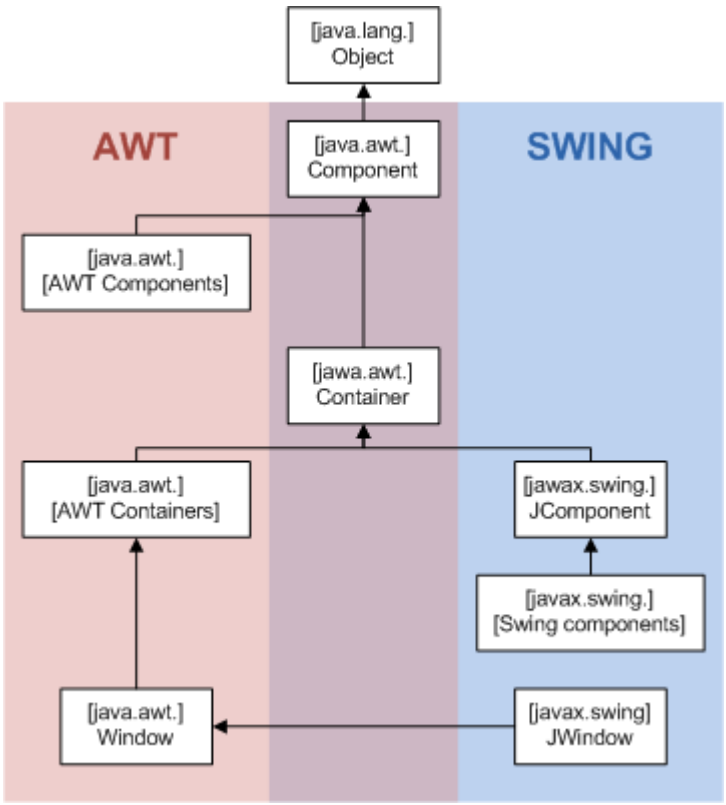
Java je jedním z nejpoužívanějších programovacích jazyků na světě. Díky své přenositelnosti je používán pro programy, které mají pracovat na různých systémech počínaje čipovými kartami (platforma JavaCard), přes mobilní telefony a různá zabudovaná zařízení (platforma Java ME), aplikace pro desktopové počítače (platforma Java SE) až po rozsáhlé distribuované systémy pracující

na řadě spolupracujících počítačů rozprostřené po celém světě (platforma Java EE). Tyto technologie se jako celek nazývají platforma Java. Jednotlivé dílčí části platformy sdílejí společné koncepty, kterými jsou syntaxe jazyka Java, virtuální stroj Javy (Java Virtual Machine) a obdobné API standardních knihoven funkcí [10].

2.4.2 Java Swing

Swing je primární knihovna pro Java GUI (grafické uživatelské rozhraní). Je součástí tříd Java Oracle Foundation (API pro poskytování grafické uživatelské rozhraní pro programy v jazyce Java). Swing je nedílnou součástí Java SE od verze 1.2. Do té doby byl dostupný pouze jako knihovna k samostatnému stažení [11].

Původní nástroj pro tvorbu grafického uživatelského rozhraní byl Abstract Window Toolkit (AWT), vyvíjený firmou Sun Microsystems jako součást Javy. První verze vyšla v roce 1995, spolu s první verzí jazyka Java. Během dalšího vývoje se u AWT projevily chyby v návrhu a další problémy (např. komponenty byly závislé na platformě a tím AWT porušovalo jeden ze základních principů, na kterém je jazyk Java postaven). V roce 1997 Sun Microsystems a Netscape Communications Corporation oznámily svůj záměr vytvořit Java Foundation Classes. Jako základ sloužily Internet Foundation Classes (IFC), vyvinuté Netscape Communications Corporation na konci roku 1996 jako nástroj pro tvorbu grafického uživatelského rozhraní pro Javu. V roce 1997 došlo ke spojení IFC a dalších technologií od Netscape a Sun Microsystems do Java Foundation Classes (JFC), později přejmenován na "Swing".



Obrázek 2.3: Hierarchie tříd

Swing byl vyvinut s cílem poskytnout sofistikovanější sadu komponent GUI než starší AWT. Poskytuje nativní vzhled, který napodobuje vzhled několika platforem, dále je možné připojit vlastní styl, který změní vzhled celé aplikace. Swing má výkonnější a přizpůsobivější komponenty než AWT, kromě známých prvků jako jsou tlačítka, checkboxy a labely nabízí několik pokročilých komponent jako jsou přepínací panely (tabbed panel), scrollovatelné panely (scroll panes), stromy, tabulky a seznamy. Na rozdíl od AWT, nejsou komponenty implementovány specifickým kódem pro platformu, ale jsou napsané v Javě a proto jsou nezávislé na platformě.

Hierarchie tříd grafických komponent Swingu je založena na rodičích z AWT, jak ilustruje obrázek 2.3.

Jako každá třída v Javě, i třídy grafického uživatelského rozhraní jsou potomky třídy Object.

Základním stavebním kamenem všech grafických komponent je třída Component. Instance třídy Component představují objekty, které mají svoji grafickou reprezentaci a mohou být zobrazeny na monitoru a interagovat s uživatelem. Všechno, co Java na monitoru nakreslí, je instance třídy Component. Různé ovládací prvky (tlačítko, zaškrtávátko, seznam, atd…) v AWT jsou definovány jako potomci třídy Component.

Od třídy Component je dále odvozena třída Container (kontejner). Kontejner je grafická komponenta, která v sobě může držet a kreslit ostatní grafické komponenty. Jakým způsobem se komponenty v kontejneru nakreslí, určuje tzv. správce rozložení (implementace rozhraní LayoutManager). V AWT jsou od třídy Container odvozeny různé prvky, např. třída Window nebo Panel.

Základním kamenem knihovny Swing je třída JComponent. Tato třída je rodičem každé swingovské komponenty. Jak je patrno z obrázku 2.3, JComponent je potomkem Container. Tudíž všechny swingovské prvky v sobě mohou držet další komponenty. Potomci třídy JComponent jsou samotné swingovské komponenty, jako je JButton (tlačítko), JList (seznam), JPanel (obecný panel), atd… Všechny swingovské komponenty mají před svým logickým názvem písmeno J, aby byly rozlišitelné od svých protějšků z AWT.

K této obecné hierarchii existují výjimky, a to jsou okna a obecně všechny nejvyšší kontejnery (top-level containers - okna, dialogy, applety). Okna jsou obecně potomci třídy Window, a nemohou proto být potomci třídy JComponent (v Javě nelze dědit od více tříd). Swingovské protějšky k Window a jeho potomků (Frame, Dialog, atd…) jsou definovány jako potomci těchto tříd s „J“ na začátku. Tj. hlavní swingovské okno JFrame je definováno jako potomek Frame.

Swing komponenty jsou implementovány pomocí návrhového vzoru MVC (viz. Kapitola 2.6), kde je uživatelské rozhraní (View) odděleno od logiky programu (Model) řídící logikou (Controller). Tyto tři komponenty bývají často odděleny rozhraními, a tak modifikace jedné části má minimální vliv na ostatní části. Swing také poskytuje abstraktní vrstvu mezi kódovou strukturou a grafickou prezentací komponent.

Ve Swingu má každý grafický objekt (tlačítko, seznam, rozbalovací menu…) svůj model v podobě rozhraní, který je na daném grafickém objektu nezávislý. Swing poskytuje defaultní implementaci modelu pro běžné využití, ale nic uživateli nebrání, aby si naprogramovat objekt s jiným modelem. Toto umožňuje uživateli změnit chování grafické komponenty, ale využívat její vzhled.

Model dále nabízí rozhraní pro komunikaci se zbytkem programu pomocí vyvolávání událostí. Každá komponenta nabízí tzv. listener na různé události, kterýkoli objekt si k ní může přihlásit. Listener je pro danou komponentu definován obvykle jako nějaké rozhraní. Když v komponentě proběhne daná události např. zmáčknutí tlačítka, vybrání položky, minimalizování okna, atd., komponenta všem přihlášeným posluchačům odešle zprávu o provedené akci. Tato technologie přihlašování a odesílání zpráv je založena na návrhovém vzoru Observer.

2.5 GUI

Tato kapitola popisuje použité vývojové prostředí a možnost vytvoření grafických komponent, které by bylo možné vkládat přímo v grafické části vývojového prostředí.

2.5.1 Netbeans

NetBeans začínal jako studentský projekt v roce 1996 v České republice. Cílem bylo v Javě vytvořit integrované vývojové prostředí (IDE) pro Javu, podobné jako bylo Delphi. Jednalo se o první Java IDE psané v Javě a jeho název byl Xelphi. Prvního vydání se bylo v roce 1997 a protože projekt vzbudil dostatečný zájem, tak se ho vývojáři rozhodly prodávat jako komerční produkt. V roce 1999 se IDE přejmenovalo na NetBeans a byla vydána verze DeveloperX2, která podporovala Swing. Tato verze bylo průlomová, protože byla přepracována tak, aby byla více modulární a víceméně tvoří základ pro IDE jak ho známe dnes. Po vydání JDK 1.3 na podzim roku 1999, které přineslo mnohá výkonnostní vylepšení se NetBeans stal lákavou volbou pro Java vývojáře [4].

Programátoři začaly vyvíjet své aplikace v NetBeans IDE a vytvářeli si vlastní pluginy, které nebyly spojené jen se samotným programováním a tím se funkce IDE začaly rozrůstat. V roce 1999 se o NetBeans také začala zajímat firma Sun Microsystems, která chtěla lepší vývojové nástroje. Později v roce 2000 se vývojáři rozhodli projekt vést jako open source a firma Sun Microsystems projekt sponzorovala. Byl to první open source projekt, který firma sponzorovala až do ledna roku 2010, kdy se Sun Microsystems stal dceřinou společností Oracle. Oracle dál sponzoroval NetBeans a vyhledával nové vývojáře pro práci v NetBeans týmu. NetBeans IDE se stal oficiální IDE pro platformu Java.

Nyní je NetBeans zdarma pro komerční i nekomerční použití. Zdrojový kód je k dispozici každému kdo by ho chtěl použít a upravit, za předpokladu dodržení stanovených podmínek. Díky početné komunitě a možnosti vytváření modulů ve formě pluginů je dnes NetBeans použitelný pro mnoho programovacích jazyků (např. C/C++, PHP, Perl, Python, JavaFX, JavaScript) a frameworků [3].

2.5.2 Nové komponenty v Netbeans

Swing je založený na komponentách, každá komponenta dědí od třídy javax.swing.JComponent a implementuje rozhraní, které je pro daný typ komponenty specifikované. Swing komponenty jsou JavaBeans komponenty, vyhovující specifikacím JavaBeans architektury.

JavaBean je opakovatelně použitelná programová komponenta, se kterou lze vizuálně manipulovat ve vývojových prostředích. JavaBean komponentou může být například jednoduchý prvek uživatelského rozhraní jako je tlačítko nebo editační pole ve formuláři, případně složitější programová komponenta jako třeba tabulkový kalkulátor. Některé komponenty dokonce nemusejí být viditelnými součástmi grafického rozhraní, i když je možné je používat ve vizuálních nástrojích. Typickou JavaBean komponentu lze charakterizovat následujícími vlastnostmi:

- Introspekce - umožňující vývojovým nástrojům analyzovat to, jak komponenta pracuje.
- Přizpůsobivost - možnost nastavení vzhledu a chování komponenty při vývoji aplikace.
- Události - prostředek pro komunikaci mezi komponentami.
- Vlastnosti - nastavitelné hodnoty určené pro přizpůsobení i pro programovou obsluhu komponenty.
- Persistence - možnost uložit přizpůsobenou komponentu a později její stav obnovit.

JavaBean komponenta je implementována jako obyčejná Java třída, přičemž se nepožaduje, aby tato třída byla odvozena z nějaké konkrétní bázové třídy nebo aby implementovala konkrétní rozhraní. V případě vizuální komponenty je tato třída odvozena od třídy javax.swing.JComponent, aby bylo možné ji zařadit do vizuálních kontejnerů.

Při vytváření nových grafických komponent je dobré dodržovat Swing hierarchii tříd a při vytváření nové komponenty dědit z již existující komponenty, díky tomu není třeba implementovat celé rozhraní a je dodržena kompatibilita s ostatními prvky a nezávislost na architektuře.

Dále při vytváření nové grafické komponenty požadujeme, aby programátor mohl danou komponentu vkládat přímo ve vývojovém prostředí NetBeans a nemusel ji nastavovat přímo ve zdrojovém kódu. Toho se nejlépe dosáhne tím, že se z dané komponenty vytvoří JavaBean komponenta (viz. výše), a ta se v NetBeans připojí přes Palette Manager do NetBeans GUI Builderu [12].

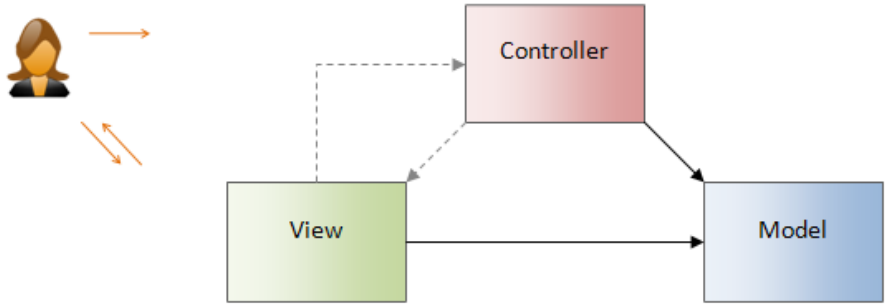
2.6 MVC

Cílem této kapitoly je popsat jak funguje návrhový vzor MVC (Model-View-Controller). Protože MVC se týká architektury aplikací mnohem více než klasický návrhový vzor, je možné ho chápat i jako architektonický vzor [13][14].

2.6.1 Princip

Architektura MVC dělí aplikaci na 3 logické části tak, aby je šlo upravovat samostatně a dopad změn byl na ostatní části co nejmenší. Tyto tři části jsou Model, View a Controller. Na obrázku 2.4 jsou znázorněny vztahy mezi jednotlivými částmi vzoru MVC navzájem a mezi uživatelem. V MVC jsou primárně dvě vazby (znázorněny nepřerušovanou černou šipkou), jsou to vazby z View na Model

a z Controller na Model. View obsahuje vazbu na Model, aby mohl zobrazit jeho data a Controller zase, aby mohl jeho data změnit. Další vazba je mezi Controller a View, a používá se jen tehdy chceme-li, aby mohl Controller určit, které View se má zobrazit. Další nepřímá vazba je mezi View a Controller, je to v případě, kdy View dokáže zachytávat uživatelské vstupy a provádět kontrolu před zpracováním události v Controller. Proto po zpracování vstupu musí mít vazbu na Controller. Tahle vazba je typická u komponentových frameworků (např. Java Swing komponenta dokáže sama zachytávat, když uživatel něco píše a reagovat na to).



Obrázek 2.4: Architektura MVC

2.6.2 Model

Model představuje a spravuje konkrétní data a business logiku nad těmito daty. Odpovídá na požadavky týkající se stavu modelu (většinou View) a na požadavky na změnu aktuálního stavu (většinou Controller). Pokud systém používá databázi k ukládání dat, tak právě model komunikuje s databází a vytváří SQL dotazy.

2.6.3 View

View má za úkol zobrazovat uživateli data z Modelu a má na starosti další prvky uživatelského rozhraní. Je zodpovědný za překreslování a aktualizaci grafické části aplikace pokud se v Modelu změní data. Pokud se jedná o systém založený na komponentách, tak se View muže starat i o kontrolu uživatelského vstupu (typicky Java Swing, Windows Forms, atd.). View může být rozděleno na několik částí a Controller muže vybrat, který View se použije. Každý View může data zobrazovat jiným způsobem.

2.6.4 Controller

Controller má na starosti tok událostí v systému a aplikační logiku, tzn. zpracovává požadavky od uživatele, které následně aplikuje v systému. Zpravidla každý Controller zpravuje jeden popř. více View. Příklad funkce Controlleru: Uživatel klikne na tlačítko odstranit položku, Controller zpracuje požadavek, zavolá v Modelu funkci na odstranění položky a informuje View, že se změnily data v Modelu.

3 Návrh

V této kapitole si ukážeme teoretický návrh komponent, jejich umístění v rámci swing hierarchie, návrh rozhraní jednotlivých komponent a jejich modelů. Rámec pro rychlý vývoj GUI aplikací by měl obsahovat tři komponenty na zobrazování dat z post-relačních databází a to:

- komponentu pracující s daty z multimediální databáze
- komponentu pro zobrazování a práci s daty z prostorové databáze
- komponentu pro práci s daty z temporální databáze

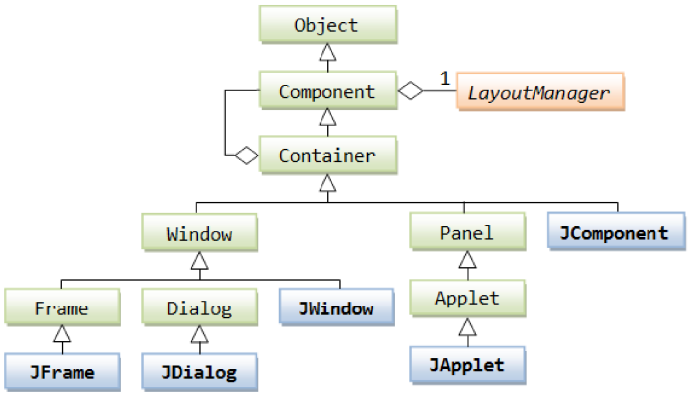
Detailní návrh jednotlivých komponent je v kapitole 3.2.

Komponenty by měly být navrženy pro jednoduché použití při implementaci aplikací. Nejdříve se zaměříme na návrh jednotlivých částí MVC struktury komponenty, následně vytvoříme teoretický návrh jednotlivých komponent a pomocí jazyka UML specifikujeme některé kroky při vývojovém cyklu.

3.1 Návrh hierarchické struktury

V této kapitole si ukážeme hierarchickou strukturu nových komponent a výběr již existující komponenty z Swing hierarchie, která bude sloužit jako nejvyšší rodič nových komponent[9].

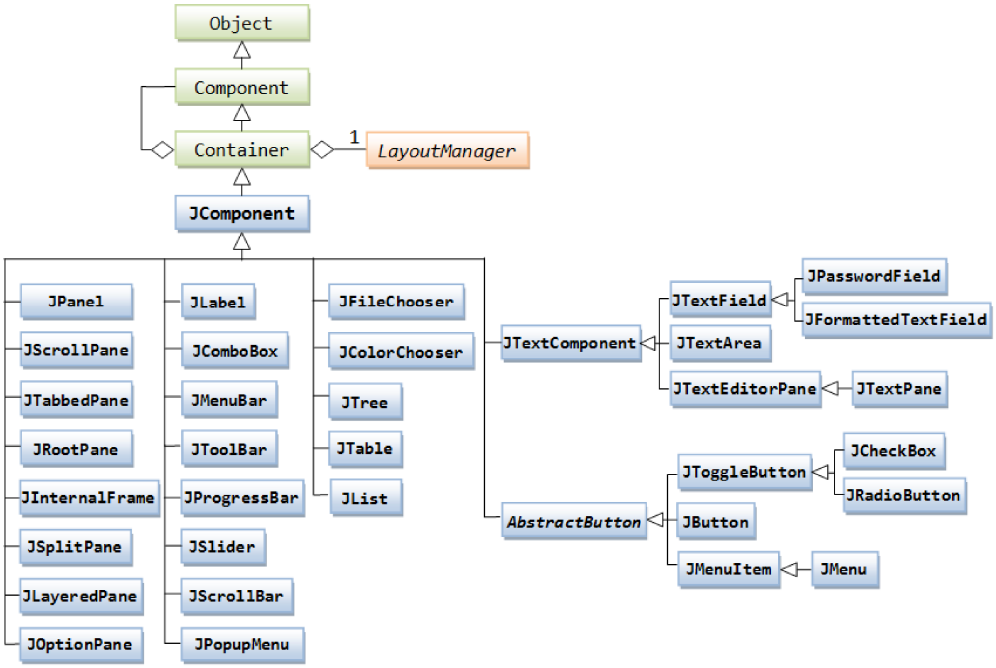
3.1.1 Výběr existující komponenty



Obrázek 3.1: Top-level diagram tříd

Na obrázku 3.1 je zobrazena top-level hierarchie Swing tříd. Každá nová grafická komponenta musí být minimálně potomkem třídy JComponent nebo musí implementovat specifikované rozhraní. Při volbě komponenty, kterou v rámci použijeme jako základ pro nové komponenty zobrazující data

z databáze jsme musely vybrat nejvhodnější komponentu. Protože se jedná o vzhledově rozdílné komponenty musí vybraná Swing komponenta umožňovat vkládání jiných prvků. K tomu se nejlépe hodí Swing komponenta JPanel. Její umístění v hierarchii je vidět na obrázku 3.2.

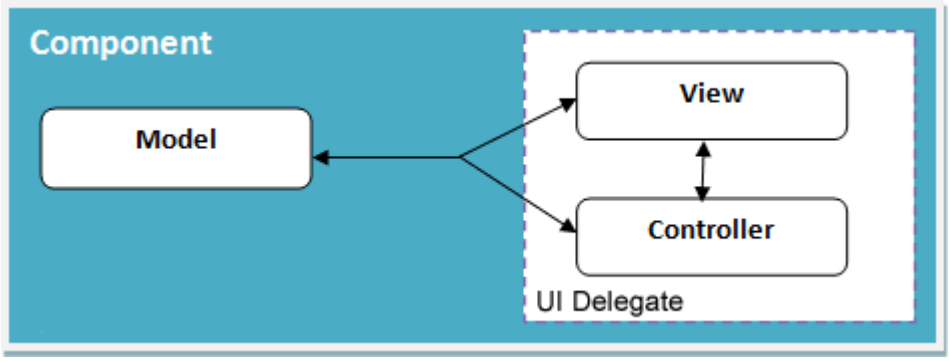


Obrázek 3.2: Detailnější class diagram

Komponentu JPanel jsme vybrali, protože se dá bez problému použít jako základ pro všechny tři typy nových komponent, nejvíce se hodí na vkládání jiných komponent a vkládání pod-komponent lze jednoduše udělat v NetBeans GUI Builderu. Většinu podmínek splňuje i nadřazená třída JComponent, ale do ní by se musely pod-komponenty vkládat manuálně ve zdrojovém kódu.

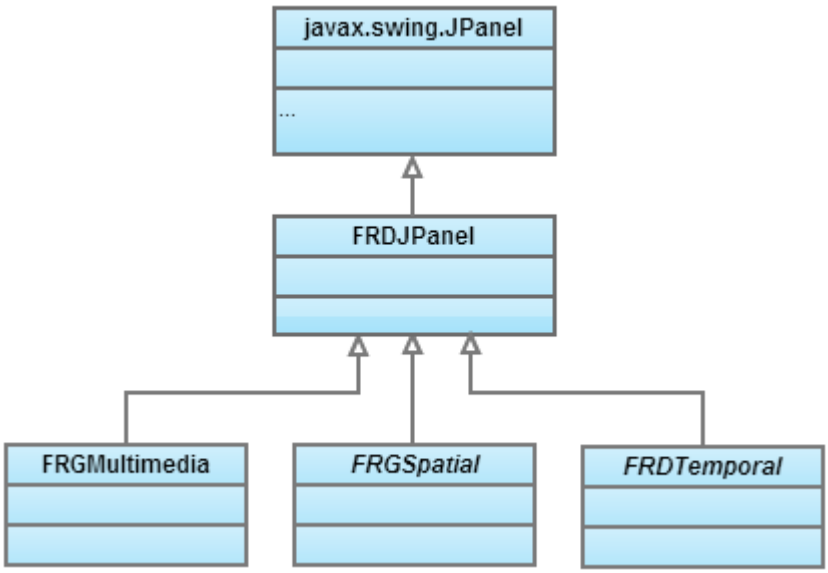
3.1.2 Návrh MVC struktury

Každá Swing komponenta je implementována pomocí MVC návrhového vzoru (viz. Kapitola 2.6). Na obrázku 3.3 je vidět jak je MVC implementováno v Java Swing komponentách. Každá komponenta má část View a Controller ve stejné části zdrojového kódu. Je to dáno hlavně tím, že pokud se grafický vzhled aplikace vytváří pomocí NetBeans GUI Builderu, tak je značná část zdrojového kódu vkládající grafické komponenty (View) chráněna proti přímím úpravám a zdrojový kód pro ovládání událostí (Controller) se musí implementovat pomocí NetBeans funkcí. Část Model je implementována ve zvláštní třídě, ale některé komponenty Model vůbec neobsahují (např. JPanel) a naopak některé komponenty mají Model implementovaný ve více třídách (např. Jlist) [15].



Obrázek 3.3: Java Swing MVC

Návrh MVC struktury pro rámec bude rozdělen na dvě části. První část návrhu je vytvoření hierarchie tříd nových komponent, jak už bylo napsáno v kapitole 3.1.1, je vybrán JPanel jako hlavní třída pro nové komponenty. Základní hierarchie je vidět na obrázku 3.4.



Obrázek 3.4: Základní hierarchie tříd

Část View nových komponent (tj. FRDMultimedia, FRDSpatial, FRDTemporal) bude vytvořena pomocí NetBeans GUI Builderu. Jak už bylo uvedeno, jedná se o rozšíření komponenty JPanel, která bude obsahovat pod-komponenty specifické pro daný typ komponenty. Grafická reprezentace nových komponenty, bude ve výsledku pevně nadefinovaná a neměnná (v případě potřeby je ovšem možno celé View implementovat znovu). Použité typy pod-komponent jsou blíže specifikovány v kapitole 3.2.

Část Controller nových komponent bude implementována dle požadavků při implementaci. Bude se jednat hlavně o změny stavů pod-komponent při uživatelských událostech, následná změna dat zobrazených v grafickém výstupu aplikace a o změny dat v databázi. Detailnější popis Controlleru jednotlivých komponent je v kapitole 3.2.

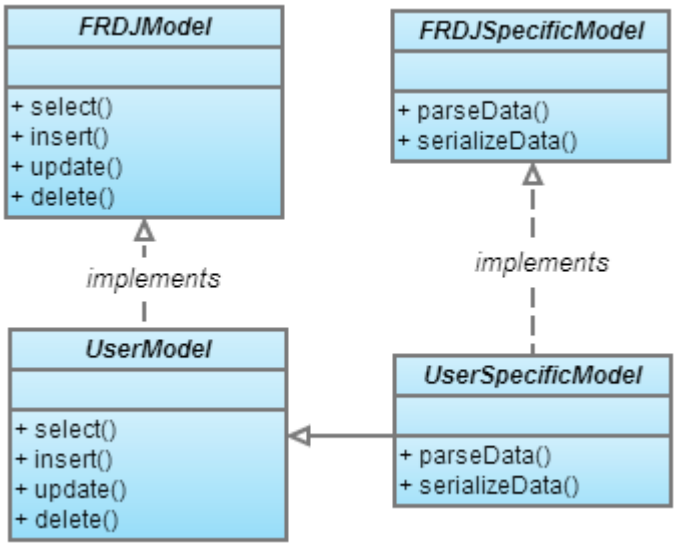
Část Model bude podle požadavků zadání vytvořena pomocí rozhraní, které bude implementováno až v aplikacích, používající daný rámec. Model každé komponenty se bude starat o načítání a ukládání dat do jednotlivých typů databází a bude mít přesně specifikovánu strukturu výstupních dat jednotlivých metod. Návrh rozhraní bude popsán v následující kapitole 3.1.3.

3.1.3 Návrh společných metod a atributů

Jako první zde bude popsán Model. I přes to, že Model bude implementován pouze jako rozhraní, jsou na diagramu tříd na obrázku 3.5 vidět i uživatelské třídy v klientské aplikaci. Je zde vidět názorná ukázka jak by mělo rozhraní fungovat. Třída FRDJModel bude implementovat rozhraní pro metody pro práci s databázemi na úrovní SQL dotazů (co bude jaká funkce dělat je zřejmé z jejího názvu), proto je rozhraní společné pro všechny modely. Naopak třída FRDJSpecificModel bude implementovat rozhraní Modelů jednotlivých typů komponent. Daná třída ve skutečnosti představuje třídy pro tři typy modelů a to konkrétně:

- FRDMultimedialModel
- FRDTemporalModel
- FRDSpatialModel

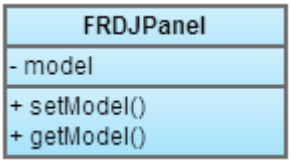
Pro názornost jsou zobrazeny jako jedna třída. Ve výsledných třídách budou názvy funkcí stejné, ale z důvodu načítání a ukládání různých typů dat musí mít dané metody různé typy parametrů a návratových hodnot, proto musí mít každý model vlastní rozhraní. Velice podobně je to i s názvy uživateli implementované třídy UserSpecificModel, s tím rozdílem, že názvy v daných tříd budou v plné režii uživatelů. V diagramu mají v názvu user, aby je bylo možné jednoduše odlišit od tříd nacházejících se v rámci.



Obrázek 3.5: Diagram tříd popisující Model komponent

Funkce parseData bude načítat data z databáze pomocí funkce rodiče (konkrétně funkce select) a následně je vloží do specifické struktury, kterou bude schopen zpracovat Controller dané komponenty. Funkce serializeData naopak dostane data z Controlleru a bude z nich připravovat SQL dotaz a podle typu události zavolá konkrétní funkci rodiče (tj. insert, delete nebo update).

Dále budou popsány společné metody a atributy pro View a Controller. Vzhledem k tomu, že každá nová komponenta bude mít specifické pod-komponenty a tudíž i metody reprezentující funkci Controlleru tak jediné společné atributy a metody jsou vidět ve třídě FRDJPanel na obrázku 3.6. Jedná se o atribut model který bude nutný nastavit přes daný setter při implementaci klientské aplikace. Typ a třída modelu bude určena podle typu komponenty.



Obrázek 3.6: Třída FRDJPanel

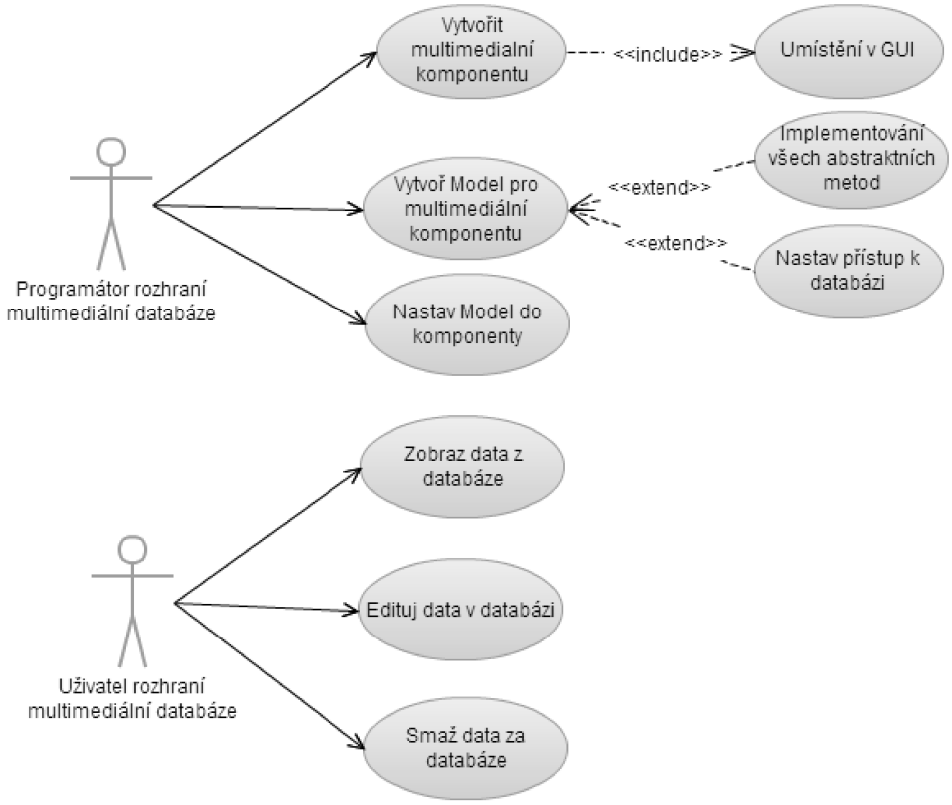
3.2 Návrh komponent

Tato kapitola detailněji popisuje návrh jednotlivých komponent, jedná se zejména o návrh typů grafických komponent, které budou použity v jednotlivých třídách ve výsledném rámci.

3.2.1 Multimediální databáze

Komponenta pro práci s multimediálními daty bude reprezentována třídou FRDMultimedia, tato třída bude implementovat logiku pro zobrazování, vkládání, editaci a mazání multimediálních dat. Data budou vykreslována do grafických boxů. Po vybrání konkrétní položky bude otevřen detail, ve kterém budou vypsány dodatečné informace a další možnosti práce s položkou (např. smazání, rotace, atd). Součástí komponenty bude box na vkládání nových dat a filtrace načtených položek. Na obrázku 3.7 je zobrazen diagram případů užití pro multimediální komponentu (tento diagram by byl pro ostatní komponenty velice podobný, a proto bude popsán pouze u multimediální komponenty). Diagram znázorňuje dva aktéry. První aktér je programátor, který bude daný rámec používat při vývoji nové aplikace (v diagramu pojmenován jako „Programátor rozhraní multimediální databáze“). Tento aktér má případy užití, které souvisejí s vytvořením komponenty a integrací do aplikace, dále vytvořením modelu a implementování všech jeho abstraktních metod. Model slouží na komunikaci aplikace s databází a programátor tento model implementuje sám. Druhý aktér je uživatel výsledné aplikace (pojmenován „Uživatel rozhraní multimediální databáze“). Tento aktér má případy užití týkající se používání aplikace. V diagramu jsou znázorněny pouze případy užití na vkládání, editaci a mazání, ale jedná se pouze o základní teoretické případy užití, protože

programátor implementující výslednou aplikaci může další případy užití vytvořit nebo existující zakázat.



Obrázek 3.7: Diagram případů užití pro multimediální komponentu

3.2.2 Prostorová databáze

Komponenta pro práci s prostorovými daty bude reprezentována třídou FRDSpatial. Bude vykreslovat grafické objekty na pozadí, které půjde měnit (jako příklad lze uvést zobrazení mapy). Komponenta bude implementovat dva režimy, režim editace a režim zobrazování. V režimu zobrazování bude umožněno vybrat konkrétní objekt a zobrazit si k němu bližší informace (propojení na multimediální a temporální komponentu). V režimu editace bude umožněno měnit prostorové objekty (např. v případě lomené přímky přesouvání jednotlivých bodů nebo přesun celých objektů), mazat je a vytvářet nové. Další funkcí bude vytváření nových typů grafických komponent. Díky této funkci bude možné vytvořit typ prostorové komponenty, k tomuto typu přiřadit konkrétní prostorový druh (přímka, bod, oblast, atd.) a daný typ propojit s multimediální databází.

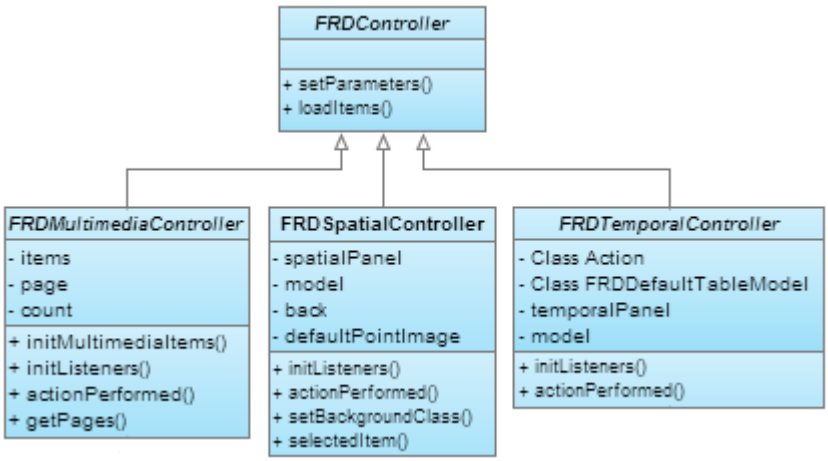
3.2.3 Temporální databáze

Temporální komponenta bude pracovat s časovými údaji jednotlivých prvků. Bude reprezentována třídou FRDTemporal. Hlavní část této komponenty bude tvořena tabulkou a filtrem, který bude použitelný na konkrétnější výpisy dat. Tabulka bude sloužit na výpis jednotlivých časových údajů, které můžou, ale nemusí, být propojeny s prostorovou částí aplikace. Tato komponenta bude hlavně

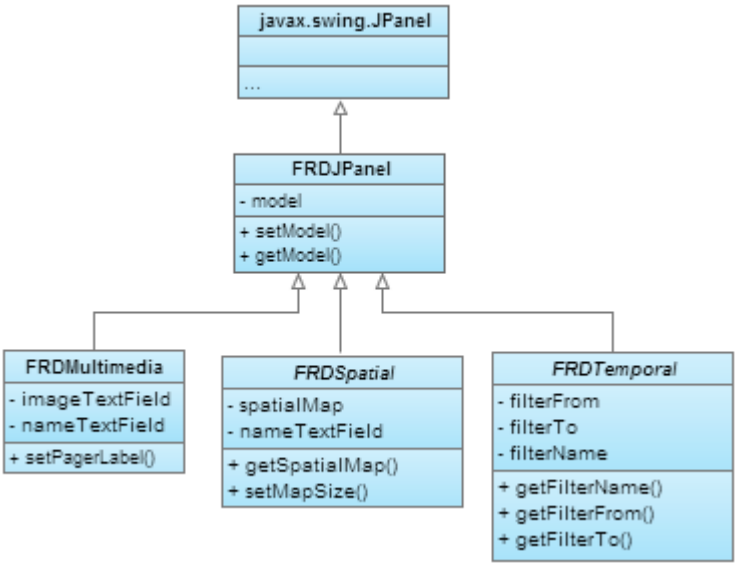
sloužit na přidávání a editaci časových údajů prostorových komponent. Tato část se ideálně hodí na editaci objektů, které s časem mění svoji polohu nebo tvar. Tyto změny budou prakticky zobrazeny v komponentě pro prostorová data. Dále se tato komponenta bude hodit na hromadné editace časových údajů (např. prodloužení času u více komponent).

4 Implementace rámce

V této kapitole se budeme zabývat implementací rámce. Rámec je implementován jako knihovna, která obsahuje všechny třídy a rozhraní, které je možné použít pro rychlý vývoj databázových aplikací. Kapitola popisuje postup při vývoji, odchýlení od původního návrhu a výsledné zhodnocení implementace. Kapitola je rozdělena na tři podkapitoly, každá z nich popisuje část rámce pro práci s daným typem databáze.



Obrázek 4.1: Class diagram pro controllery



Obrázek 4.2: Class diagram pro panely reprezentující view

Při implementaci se objevila nutnost oddělit část controlleru do vlastní třídy, tato třída nebyla v původním návrhu zahrnuta a týká se multimediální, prostorové i časové části rámce. V implementaci návrhového vzoru MVC knihovnou Swing (viz. kapitola 3.1.2) je část controller a view ve stejné části zdrojového kódu, ale z důvodu jednoduší implementace aplikace využívající tento rámec je controller rozdělen. Část controlleru je tedy pořád ve třídě reprezentující view a část je ve třídě obsahující hlavně uživatelský kód (více je popsáno v kapitole popisující implementaci ukázkové aplikace). Každý controller implementuje funkci na načtení konkrétních dat (loadItems()). Tato funkce počítá se zadáním parametrů, které jsou později použity pro filtrování dat v modelu. O zadání těchto parametrů se stará funkce setParameters(), která vrací seznam parametrů v proměnné typu LinkedList<String>. Tuto funkci je možně přeimplementovat ve výsledné aplikaci a usnadňuje tedy zadání parametrů bez nutnosti přeimplementovávat celou funkci loadItems(). Tyto parametry jsou následně předány do modelu. Výsledný class diagram pro controllery je na obrázku 4.1. Protože většina funkcí pro ovládání a zpracování dat z modelu se přesunula do controllerů, nebylo možné dodržet strukturu tříd a jejich funkcí, která bylo prezentována v návrhu. Výsledný class diagram pro panely reprezentující view část je na obrázku 4.2. Také class diagram zobrazující modely byl oproti návrhu upraven. Výsledná podoba diagramu je vidět na obrázku 5.3, který se nachází v kapitole 5.1.

4.1 Implementace multimediální části

Multimediální část implementuje třídu pro zobrazování multimediálních dat FRDMultimedia. Tato třída reprezentuje view a část controlleru návrhového vzoru MVC. Grafika je rozdělena na dvě části. Pravá strana zobrazuje jednotlivé prvky načtené z multimediální databáze. V levé části jsou pak zobrazeny ovládací prvky.

```
@Override
public void addMouseListener(MouseListener listener)
{
  super.addMouseListener(listener);
  for(Component component: this.getComponents())
  {
    if (component instanceof JButton)
      ((JButton)component).addActionListener((ActionListener) listener);
    else
      component.addMouseListener(listener);
  }
}
```

Obrázek 4.3: Předání Listeneru do prvků umístěných v panelu

O vykreslení konkrétních prvků se stará třída FRDMultimediaItem, která je potomkem třídy JPanel. Tato třída zobrazuje obrázek načtený z databáze a uchovává identifikátor pro pozdější editování prvku. V pravé dolní části tohoto panelu je vykresleno tlačítko, které slouží pro zobrazení detailu. Zobrazení okna s detailem není implementováno, je to hlavně proto, že rámec nemůže znát strukturu

databáze a proto je stejně nutné detail implementovat, až ve výsledné aplikaci. Tlačítko pro zobrazení detailu má nastaveno atribut ActionCommand, podle kterého controller rozpozná, o které tlačítko se jedná. Všechny tlačítka v rámci, jejichž operace zpracovává controller, používají na rozpoznání tento atribut. Třída FRDMultimediaItem má nastaven MouseListener pro zpracování akcí myši, ale není implementovaná žádná operace, která by na nějakou akci myši reagovala. Nastavení MouseListeneru je spíše z důvodu předání Listeneru všem prvkům, které jsou v panelu umístěny. Předání tohoto Listeneru provádí funkce addMouseListener(), která je vidět na obrázku 4.3. Při vytváření vlastní podtřídy (hlavně z důvodu přidávání dalších funkčních tlačítek) není tedy potřeba se starat o nastavení Listeneru novým prvkům.

Jak už bylo uvedeno v úvodní části bylo nutné implementovat i třídu reprezentující controller. Tato třída se jmenuje FRDMultimediaController a stará se o jednotlivé uživatelské akce. Controller implementuje rozhraní ActionListener a proto je možné do něj směrovat veškeré uživatelské události (např. kliknutí na tlačítko, interaktivní změna údajů) a následně je zpracovat. Nestandardní operací controlleru je načtení dat z modelu a vložení je do grafických prvků v FRDMultimedia (view). Toto načítání umožňuje jednodušší změnu funkce pro načítání a zobrazování dat ve výsledné aplikaci. Při načítání obrazových dat je načtena jejich velikost a upravena podle velikosti grafického prvku. Při načítání dat, u kterých poměr velikostí neodpovídá poměru velikostí grafického prvku je velikost upravena tak, aby výsledný obraz nebyl deformovaný. Dále controller obsahuje funkce pro stránkování dat z databáze.

Dále tato část rámce obsahuje rozhraní pro model, které je nutné ve výsledné aplikaci implementovat a nastavit do controlleru. Toto rozhraní definuje funkce pro načtení objektu z databáze a uložení objektu. Jedná se o funkce, které budou využity pro načítání a ukládání detailu objektů ve výsledné aplikaci. Samotný rámec tyto funkce nepoužívá, protože okno pro detail se implementuje, až ve výsledné aplikaci.

Pro načítání dat z modelu do controlleru byla vytvořena třída nesoucí veškeré informace nutné pro zobrazování. Vzhledem k tomu, že rámec nezná přesný databázový model, počítá se s tím, že uživatel rámce (programátor) využije tuto třídu jako rodiče pro vlastní třídu, která bude obsahovat veškeré informace nutné pro zobrazení. Více o konkrétním využití je v kapitole 5.1.1.

4.2 Implementace prostorové části

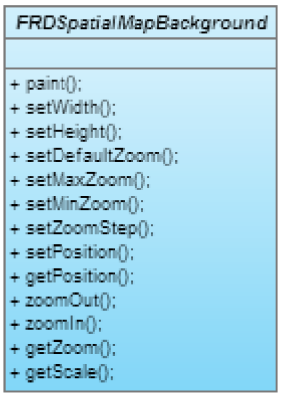
Prostorová část rámce je nejrozsáhlejší z pohledu implementace. Má za úkol zobrazovat a editovat prostorová data. Část view je graficky je rozdělena na ovládací panel a panel zobrazující grafická data. V ovládacím panelu jsou umístěny detailnější informace o vybraném grafickém prvku a ovládací prvky. Panel na vykreslování grafických objektů (FRDSpatialMap) obsahuje třídu pro vykreslení pozadí a seznam objektů načtených z databáze (podrobnější popis funkčnosti níže).

Rozhraní FRDSpatialModel definuje základní funkce, které je nutné implementovat na práci s databází. Jedná se základní vložení grafického objektu, jeho editaci a smazání. Dále je nutné implementovat funkci na načítání všech dat z databáze a vložit je do příslušných tříd (popis tříd reprezentujících konkrétní grafické objekty viz. kapitola 4.2.2).

Třída FRDSpatialController se stará o načtení seznamu grafických objektů z modelu a uložení do třídy FRDSpatialMap, která s nimi pracuje a zobrazuje je. Také se stará o uživatelské akce a o uložení třídy pro vykreslení pozadí.

4.2.1 FRDSpatialMap

Jak už bylo uvedeno, třída FRDSpatialMap zobrazuje prostorové objekty načtené z databáze. Stará se také o vykreslování nedokončených objektů při vytváření nebo editaci. Práce s tímto panelem připomíná práci s jednoduchým vektorovým editorem. Kromě jednotlivých objektů pracuje i s třídou zobrazující pozadí. Protože jako pozadí by bylo možné použít nějakou veřejnou knihovnu (např. Google Street View Map), snažily jsme se vytvořit univerzální rozhraní pro práci s pozadím (FRDSpatialMapBackground). Toto rozhraní definuje funkce pro práci s pozadím, jedná se především o funkce zjišťující aktuální stav mapy. Stav mapy je důležitý pro pozice vykreslovaných grafických objektů. Rozhraní je vidět na obrázku 4.4.



Obrázek 4.4: Rozhraní FRDSpatialMapBackground

Samotný rámec obsahuje třídu FRDSpatialMapImageBackground, která implementuje rozhraní FRDSpatialMapBackground. Tato třída používá jako pozadí obyčejný obrázek a souřadnicový systém je určen velikostí obrázku v pixelech. Tato třída se stará o vykreslení obrázku a v případě používání přiblížení potom vykresluje aktuální výřez obrázku podle aktuální pozice a stupně přiblížení. Třídě je možné nastavit maximální a minimální přiblížení a velikost kroku při přibližování a oddalování. Vykreslení pozadí obstarává funkce paint(), které je jako parametr předán objekt Graphics do kterého je obrázek vykreslen funkcí drawImage().

Třída FRDSpatialMap je implementován jako třída rozšiřující Swing komponentu JPanel. Pro vykreslování jakýchkoli grafických objektů bylo nutné přepsat funkci paintComponent(), která je volána pro vykreslení obsahu Swing komponent. Kreslení do komponent je realizováno pomocí třídy Graphics, která umožňuje nejen do komponent kreslit, ale také definuje parametry kreslení (např. barvu, font). Jako první se zavolá funkce pro vykreslení pozadí (tzn. FRDSpatialMapBackground a jeho metoda paint()). Následně se vykreslí všechny grafické objekty (metoda draw() pro každý objekt), které jsou uložené v seznamu objects. Poté se testuje, zda

je některý z objektů právě editován nebo vytvářen. Pokud ano zavolá se jejich metoda drawEdit() pro vykreslení. Třída FRDSpatialMap implementuje rozhraní MouseListener a MouseMotionListener, takže zadávání nových bodů při akcích myši obstarávají funkce mousePressed(), mouseReleased() a mouseDragged(). Pokud je vybrán konkrétní objekt zadané body zpracovává daný objekt, pokud ne, jsou tyto body použity pro přesouvání pozice výřezu mapy.

4.2.2 Třídy grafických objektů

Vzhledem k tomu, že prostorová databáze umožňuje ukládat různé druhy prostorových objektů, bylo nutné odlišit práci s jednotlivými typy grafických objektů. Proto byla pro univerzálnost vytvořena abstraktní třída FRDSpatialGItem (je vidět na obrázku 4.5), která definuje základní abstraktní funkce pro vykreslování. Jedná se zejména o funkce draw() a drawEdit(). Funkce draw() se stará o vykreslení kompletního objektu a rozlišuje, zda je objekt vybrán nebo ne. Funkce drawEdit() je určena pro postupné vykreslování nového objektu a pro vykreslování editovaného objektu. Další důležitou funkcí je undermouse(), která zjistí podle zadaného bodu zda se grafický objekt nachází pod aktuální pozicí kurzoru a vrací hodnotu true nebo false. Tato funkce se používá při výběru prvku a při jeho editaci. Další funkce slouží pro definování barev, které jsou použity při vykreslování. Jedná se o barvu standardního vykreslení (defaultColor), barvu pro vykreslení vybraného objektu (SelectedColor) a jako poslední je barva objektu pokud je editován (editColor). Jako poslední parametr je v objektu uložena instance třídy FRDSpatialMapBackground (tj. back). Jak už bylo uvedeno výše, tato třída se stará o vykreslování pozadí, takže jako jediná zná aktuální přiblížení v mapě, měřítko a pozici zobrazované mapy. Objekty potřebují tyto parametry pro spočítání své vlastní pozice v mapě.

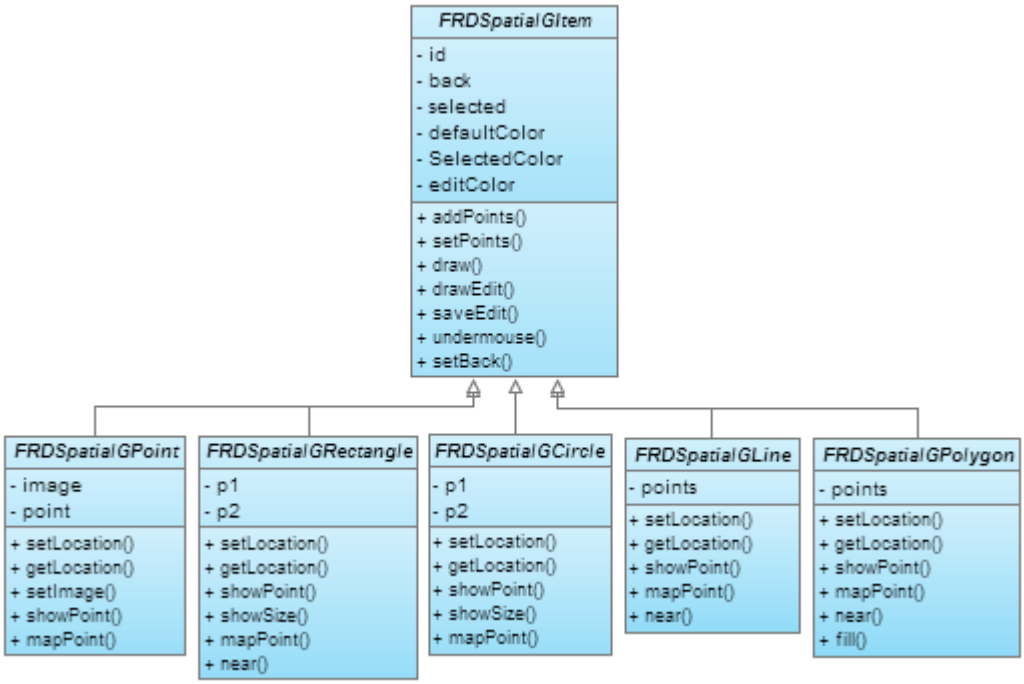
Při přidávání nového grafického objektu se vytvoří instance třídy daného typu a postupně se do objektu přidávají body nesoucí pozici kurzoru při kliknutí myší v mapě. Pro vložení bodu slouží funkce addPoints(). Ta je implementována tak, že pokud objekt nemá uložen žádný bod, tak jej vloží. Pokud ale objekt nějaký bod obsahuje, tak funkce upravuje pouze ten poslední. Díky tomu je možné vidět tu část objektu, která je zrovna vytvářena. Pro konečné uložení bodu slouží funkce setPoints(). Vykreslení při vytváření musí být odlišeno od základního vykreslení, protože některé prvky nelze vykreslit, dokud nejsou kompletně zadané (např. polygon), proto se o vykreslení stará funkce drawEdit().

Editace prvku sestává ze tří kroků. Jako první je nutno vybrat objekt pro editaci, k tomu slouží výše zmíněná funkce undermouse(). Každý objekt tedy sám ověří zda je vykreslen pod kurzorem nebo ne. S tím nastává malý problém pokud se objekty překrývají. Rámec je implementován tak, že vybere pouze první objekt, který se nachází pod kurzorem. Pokud je objekt vybrán, vykreslí se odlišnou barvou nebo se kolem něj vytvoří rámeček (jedná se o objekty vykreslené jako ikona atd.). Druhým krokem je výběr typu editace, o tento výběr se stará přímo funkce drawEdit().

Této funkci jsou předány tři parametry:

- Graphics g
- Point pt1
- Point pt2

Parametr g slouží pro vykreslování, parametr p1 určuje pozici kliknutí a parametr p2 určuje novou pozici kurzoru. Parametr p1 tedy určuje výběr typu editace. Vybraný objekt musí zvolit akci, která proběhne při změně pozice kurzoru. Jako názorný příklad je editace čtverce. Podle pozice bodu p1 určíme zda budeme měnit velikost čtverce (bod p1 se nachází v blízkosti nějakého rohu čtverce), budeme měnit pozici čtverce (p1 se nachází uvnitř čtverce), nebo nebudeme provádět žádnou akci (p1 je mimo čtverec). Pro uložení poslední pozice při editace slouží funkce saveEdit().



Obrázek 4.5: Class diagram zobrazující jednotlivé třídy pro grafické objekty

Základní typy objektů se kterými rámec umí pracovat a implementuje pro ně třídy na vykreslování (v závorce jsou uvedeny názvy tříd):

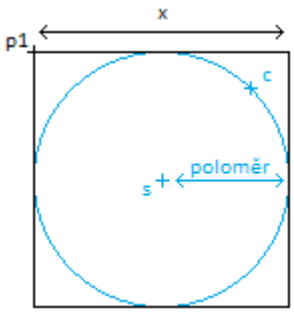
- bod (FRDSpatialGPoint)
- lomená čára (FRDSpatialGLine)
- polygon (FRDSpatialGPolygon)
- kružnice (FRDSpatialGCircle)
- obdélník (FRDSpatialGRectangle)

Class diagram zobrazující jednotlivé třídy je vidět na obrázku 4.5.

FRDSpatialGPoint je v databázi uložen jako bod, ale při zobrazování v mapě je definován svým obrázkem (ikonou). Obrázek se vykreslí tak, aby střed byl na souřadnicích bodu uloženého v databázi. Při editaci bodu je tedy možné změnit pouze jeho pozici na mapě.

FRDSpatialGLine je třída reprezentující čáru nebo přesněji lomenou čáru. Z implementovaných tříd je to jediný grafický objekt, u kterého při vytváření nelze rozpoznat jestli je už dokreslen do konce nebo ne. Pro upřesnění jak je to myšleno uvádíme příklad. Obdélník, kružnice a bod je ukončen při zadání dvou bodů (tzn. hned při prvním „táhnutí“ myši). V případě bodu pouze jeden, ale je možné zadat dva, tak se použije ten, který odpovídá souřadnicím uvolnění tlačítka myši. V případě polygonu končí kreslení zadáním bodu, který přibližně odpovídá souřadnicím prvního bodu polygonu. Ale u lomené čáry nevíme kdy končí, proto to musí určit uživatel. Z toho důvodu musí být v ovládacím panelu aplikace tlačítko ukončující kreslení lomené čáry. Rozpoznání zda je kreslení objektu dokončeno určuje funkce addPoints(), která vrací true v případě, že je objekt dokončen. V případě lomené čáry vrací vždy false. Lomenou čáru můžeme editovat, buď změnou pozice některého z bodů nebo posunutím celého objektu. Jaký druh editace bude aplikován, určuje pozice kurzoru při stisknutí levého tlačítka.

Třída FRDSpatialGPolygon slouží pro uložení polygonu. Polygon (nebo také mnohoúhelník) je omezená část roviny ohraničená uzavřenou lomenou čárou. Je tedy implementován velmi podobně jako lomená čára s tím rozdílem, že poslední bod polygonu musí odpovídat tomu prvnímu. Na vykreslení je použita funkce fillPolygon(). Největší problém při implementace byla funkce undermouse(). Konečné řešení vychází z toho, že vytvoříme prázdný obrázek do kterého daný polygon vykreslíme s nějakou barvou. Následně zjišťujeme, zda je pozice daná kurzorem vybarvená nebo ne. Toto řešení vychází z předpokladu, že známe velikost celého souřadnicového systému a tudíž musí být pro pozadí použita třída FRDSpatialMapImageBackground jinak rámec vyhodí vyjímku. Při implementace vlastní třídy pro pozadí mapy je nutné implementovat i vlastní třídu pro vykreslování polygonu, nebo alespoň přeimplementovat funkci undermouse(). Polygon lze editovat podobně jako lomenou čáru, tzn můžeme změnit pozici některého z bodů nebo přesunout celý polygon. Druh editace opět určuje pozice kurzoru při stisknutí levého tlačítka.



Obrázek 4.6: Souřadnice při vykreslování kruhu

FRDSpatialGCircle reprezentuje kružnici. Kružnice se při vytváření definuje bodem, který určuje souřadnice jejího středu (na obrázku 4.6 jako bod 's') a bodem, jehož vzdálenost od prvního bodu

určuje poloměr kruhu (na stejném obrázku například bod 'c'). Ale při samotném vykreslování je nutné kružnici definovat bodem, který leží v levém horním rohu čtverce (na obrázku 4.6 jako bod 'p1'), který kružnici opisuje a velikostí daného čtverce (velikost na stejném obrázku označena jako 'x'). Proto je nutné při vykreslování přepočítat vstupní body. Zda je kružnice pod kurzorem (v tomhle případě se tedy jedná spíše o kruh) se určuje vzdáleností bodu od středu kružnice. Pokud je vzdálenost menší tak funkce undermouse() vrátí true, jinak false. Kružnice se dá opět editovat dvěma způsoby. Prvním je změna pozice středu a druhým je zadání nového bodu určujícího poloměr kružnice. Další způsob definování pozice kružnice je třemi souřadnicemi, to se používá při uložení v databázi. Více o konkrétním uložení v databázi je v dokumentaci (viz. [2]).

Poslední třídou je FRDSpatialGRectangle. Tato třída reprezentuje obdélník. Mohlo by se zdát, že vykreslení čtverce je nejjednodušší ze všech objektů, ale není to tak. Při vykreslování čtverce je nutné zadat jeho levý horní roh a velikost čtverce jak tomu bylo u kruhu na obrázku 4.6. Při vytváření ale můžeme kreslit z libovolného rohu, a proto je nutné vždy spočítat uvedené parametry ze zadaných bodů. Při výchozím vykreslování to není nic složitého, ale v případě editace je nutné spočítat všechny čtyři body a až podle nich určit, který roh bude editován. Dále je nutné pozici určeného bodu změnit a opět přepočítat zpět na levý horní roh a velikost. Samozřejmě je možné editovat pozici obdélníku. Při určování zda je kurzor v objektu bylo nutné opět spočítat levý horní roh a velikost, protože obdélník může být zadaný libovolnými vrcholy a pozice by se hůř určovala. Pak stačilo určit zda daný bod spadá dovnitř obdélníku nebo ne a vrátit true nebo false.

Díky abstraktní třídě FRDSpatialGItem je jednoduché vytvořit si vlastního potomka reprezentujícího jiný grafický objekt (např. trojúhelník). Kromě implementováni všech abstraktních tříd starajících se o zadávání bodů, vykreslování objektu (draw i drawEdit) a zjišťování zda se objekt nachází pod kurzorem, bude pravděpodobně nutné implementovat funkce pro přepočítaní bodu ze souřadnicového systému aplikace (tj. souřadnicový systém vycházející z velikosti obrázku v případě použití třídy FRDSpatialMapImageBackground) na souřadnicový systém panelu, který grafické objekty zobrazuje. V implementovaných a výše popsaných třídách se jedná o funkce showPoint(), která přepočítá bod ze souřadnicového systému mapy na souřadnice v panelu a funkce mapPoint(), která počítá opačně. Obě funkce berou v potaz aktuální pozici v mapě, měřítko a aktuální zoom.

4.3 Implementace temporální části

Struktura temporální části rámce je velice podobná struktuře multimediální části. FRDTemporal je hlavní implementovaná třída zobrazující data načtená z databáze. Dále rámec obsahuje rozhraní pro model FRDTemporalModel, controller starající se o uživatelské operace FRDTemporalController a třídu FRDTemporalDBItem reprezentující jeden záznam v databázi. Jak už bylo uvedeno v kapitole 4.1, popisující multimediální části, aplikace nezná model databáze, proto není možné přesně specifikovat komunikaci mezi modelem a controllerem. Díky nutnosti zobrazovat nejen rozdílný počet prvků (databázových záznamů), ale i rozdílný počet jejich parametrů (počet sloupečků v databázovém modelu dané tabulky), bylo nutné vytvořit třídu která komunikuje pomocí některého dynamického datového typu. Pro tento účel byl vybrán datový typ LinkedList, který umožňuje vytvořit seznam libovolného typu. Pro univerzálnost byl zvolen datový typ String a převod dat z databáze na jednotlivé datové typy je nutné převést, až při zpracování v controlleru.

FRDTemporalModel je rozhraní, které je nutné ve výsledné aplikaci implementovat. Definuje funkce, které slouží na načítání, editaci a mazání vybraných dat v tabulce temporální dat. Model dále definuje počet a název sloupců v zobrazované tabulce a možnost jejich editace přímo v tabulce.

FRDTemporalController se stará o načítání dat z modelu, naplnění tabulky daty a stará se o uživatelské akce. Rámec implementuje dvě uživatelské akce. První je načtení dat podle parametrů filteru a druhá je smazání vybraných prvků z databáze.

Část view (FRDTemporal) je graficky rozdělena na tři části:

- filter
- tabulka s výslednými záznamy
- ovládací panel

Filtr obsahuje tři položky (jméno, platnost od, platnost do), které by měly být jako základ pro temporální tabulku. Tyto položky filtru se zadávají jako text do grafické komponenty JTextField a proto je možné použít libovolný model časové informace reprezentující platnost (např. rok – 1988, datum 15.2.2012). Tyto grafické komponenty jsou uloženy jako public, díky tomu je jednoduché v controlleru zjistit jejich hodnotu a pracovat s ní. Dále se předpokládá, že ve výsledné aplikaci, kdy je už znám model databáze, se do filteru přidají další požadované položky, které budou specifické pro konkrétní aplikaci a tudíž bude filtrování detailnější a přesnější.

Tabulka s výslednými záznamy je implementována jako JTable. Tabulka je defaultně prázdná, přesně se definuje v controlleru při načítání dat z databáze. V controlleru se také načítá počet a názvy sloupců tabulky, tyto údaje definuje uživatel rámce v modelu. Data v tabulce jsou uložena ve speciální třídě implementující rozhraní TableModel. Při vytváření tabulky v grafickém editoru NetBeans se jedná o konkrétní implementaci a to DefaultTableModel. Tento model neobsahuje jen data tabulky, ale i informace o struktuře tabulky (počet sloupců a jejich názvy) a některé operace s tabulkou. V rámci zobrazování temporálních dat v tabulce jsme implementovali i možnost jednoduché změny údajů přímo v tabulce. Při změně některého údaje se načte nová hodnota,

identifikátor v databázi a název sloupečku pro změnu. Proto bylo nutné nastavit, které sloupce tabulky bude možné editovat a které ne. K tomu slouží výše popsaný model tabulky a konkrétně funkce isCellEditable, kterou bylo nutné přepsat. Pro jednoduchost budoucí implementace byla v controlleru vytvořena třída dědící od DefaultTableModel, která přepisuje funkci isCellEditable. Třída je vidět na obrázku 4.7. Této třídě je možné v konstruktoru předat pole hodnot typu boolean specifikující, které sloupce bude možné editovat. Dále bylo nutné zjistit která buňka tabulky byla editována a její nová hodnota. Při první implementaci se zdálo, že bude stačit vytvořit třídu dědící z TableCellEditor a přepsat její metodu stopCellEditing, ale nebylo to fungující řešení, protože instance TableCellEditor nezná souřadnice dané buňky, tudíž by bylo nutné vždy ukládat do databáze celou tabulku. Jako jediné fungující řešení by bylo použít třídu implementující rozhraní PropertyChangeListener, a tento listener přiřadit k tabulce. Následně by bylo nutné implementovat funkce pro zjištění, kdy začala a skončila editace a souřadnice aktuální vybrané buňky (myšleno jako souřadnice v tabulce). Nakonec jsme použili třídu z již existujícího řešení (viz. [16]). Třída se jmenuje TableCellListener a pracuje přesně jako bylo popsáno výše. Při detekci změny vytvoří událost, kterou je možné pomocí třídy, implementující rozhraní AbstractAction, zachytit a následně zpracovat.

```
protected class FRDDefaultTableModel extends DefaultTableModel
{
  boolean[] canEdit;

  public FRDDefaultTableModel(Object[][] Data, Object[] columns, boolean[] canEdit)
  {
    super(Data, columns);
    this.canEdit = canEdit;
  }

  @Override
  public boolean isCellEditable(int rowIndex, int columnIndex)
  {
    if (canEdit.length > columnIndex)
      return canEdit[columnIndex];
    else
      return false;
  }
}
```

Obrázek 4.7: Třída rozšiřující DefaultTableModel, nastavuje možnost editování sloupců tabulky

Poslední grafickou částí v FRDTemporal je ovládací panel. Ten je implementován jako JPanel, ve kterém je možné umístit libovolné ovládací prvky a uživatelsky definované akce, které je možné zpracovat v controlleru.

5 Implementace aplikace

V této kapitole je popsán vývoj ukázkové aplikace využívající výše popsaný rámec pro rychlý vývoj GUI aplikací využívajících postrelační databáze. Je popsán postup a využití všech částí rámce a implementace všech rozhraní, které jsou nutné správnou funkci aplikace. Dále je popsána možnost rozšíření některých tříd a popis možností při přidávání nových funkcí, které rámec neobsahuje. Rámec je navržen a implementován tak, aby jeho použití bylo jednoduché a rychlé, tudíž implementování základní funkčnosti by nemělo být pro uživatele rámce moc obtížné. Kapitola také obsahuje zhodnocení použitelnosti rámce a porovnání s vývojem bez jeho využití.

Ukázková aplikace reprezentuje databázi protivníků v nejmenované hře. Multimediální část databáze obsahuje obrázek a základní popis jednotlivých druhů. Prostorová databáze zobrazuje jejich umístění na mapě. Data v prostorové části obsahují i informace o času výskytu na jednotlivých souřadnicích. V temporální části jsou pak zobrazeny jednotlivé záznamy z prostorové části u kterých je zobrazena časová informace. Časová osa je definovaná pouze roky a je definovaná v rozmezí 0 až 500.

Grafické okno ukázkové aplikace musí zobrazovat všechny panely. Protože se jedná o panely, které zabírají značnou část aplikace, jsou tyto panely umístěny v grafické komponentě JTabbedPane. Tato komponenta umožňuje jednoduché přepínání mezi jednotlivými vloženými panely. Každá grafická část rámce má nadřazenou třídu Jpanel, takže je možné je přímo vkládat do JTabbedPane. Aby bylo vkládání grafických částí rámce co nejjednodušší, je možné si dané panely přímo vložit do seznamu grafických komponent v NetBeans GUI Builderu. Na přidávání a odebírání nových komponent slouží Palette Manager. Ten lze nalézt v NetBeans v záložce Tools->Palette->Swing/AWT Components. Pro přidání se musí v Palette Manageru kliknout na tlačítko „Add from JAR...“, vybrat knihovnu, ze které chceme vložit novou komponentu. V našem případě se jedná o knihovnu FRDFramework.jar. Po vybrání knihovny jsou v následujícím kroku vidět všechny třídy, které je možné do Palette Manageru přidat. Přidáme tedy FRDMultimedia, FRDSpatial a FRDTemporal nejlépe do složky Beans, aby bylo poznat, které komponenty jsou manuálně přidané. Nyní stačí tyto komponenty přidat do JTabbedPane a nastavit velikost, aby byli všechny komponenty dobře viditelné.

Dalším krokem je inicializace komponent. Každá z komponent reprezentuje view v návrhovém vzoru MVC. Proto je nutné, aby obsahovala vazbu na model. Implementaci modelů musí obstarávat konečná aplikace. Rámec pouze definuje rozhraní, dle kterého bude komunikovat s modely. Každá ze tří výše uvedených tříd musí obsahovat model, který načítá data z příslušné databáze. Proto každá část rámce má definované specifické rozhraní, které je nutné implementovat. Proto dalším krokem při vývoji aplikace bude právě vytvoření tříd reprezentujících dané modely pro všechny tři typy databáze. Detailnější popis samotné implementace modelů bude níže. Pro nastavení modelu do každé komponenty slouží metoda setModel(). U prostorové části je nutné také nastavit velikost panelu pro zobrazení mapy (funkce setMapSize()), inicializovat a nastavit objekt starající se o pozadí (funkce setBackgroundClass()) a volitelně nastavit výchozí ikonu při přidávání grafického objektu reprezentující bod (funkce setDefaultPointImage()).

Dále je nutné vytvořit pro každou část rámce Controller. Rámec obsahuje plně implementovanou třídu pro controller, ale je nutné vytvořit vlastní podtřídy, která se budou v každé části starat o uživatelské akce a ovládání, které budou vytvořeny, až ve výsledné aplikaci. Každému controlleru

je nutné v konstruktoru předat daný panel (FRDMultimedia, FRDSpatial nebo FRDTemporal) a model. Inicializace těchto komponent je vidět na obrázku 5.1. Na obrázku je také vidět třída pro práci s databází a inicializace jednotlivých modelů, popis těchto části je obsahem následující kapitoly.

```
db = database;

//Multimedialni cast
fRDMultimedia1.setModel(new MultimediaModel(db));
mc = new MultimediaController(fRDMultimedia1, fRDMultimedia1.getModel());
mc.loadItems();

//Prostorová část
fRDSpatial1.setModel(new SpatialModel(db));
fRDSpatial1.setMapSize(612, 500);//velikost souradnicoveho systemu urcuje velikost obrazku v pixelech
sc = new SpatialController(fRDSpatial1, fRDSpatial1.getModel());
sc.setBackgroundClass(new FRDSpatialMapImageBackground(getClass().getResource("/data/map.jpg")));
try
{
  sc.setDefaultPointImage(ImageIO.read(getClass().getResource("/data/symbol_cave.png")));
}
catch (IOException e) { System.err.println(e.getMessage()); }
sc.loadItems();

//Temporální část
fRDTemporal1.setModel(new TemporalModel(db));
tc = new TemporalController(fRDTemporal1, fRDTemporal1.getModel());
tc.loadItems();
```

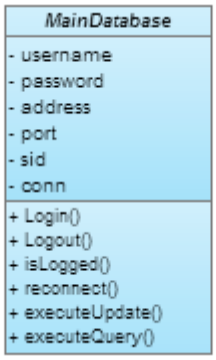
Obrázek 5.1: Inicializace základních komponent rámce

5.1 Databáze a modely

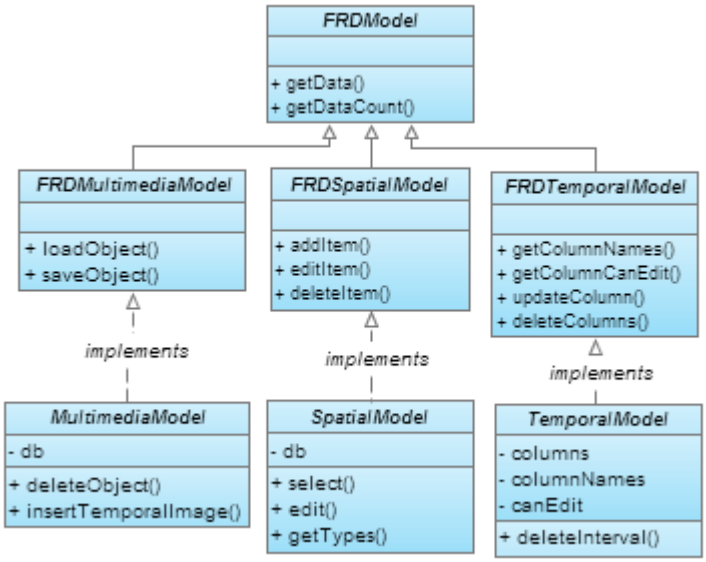
Aplikace používá postrelační databáze k uložení dat. Rámec neobsahuje třídu pracující s databází ani rozhraní pro implementaci této třídy. Proto je nezbytné vytvořit třídu, která se bude starat o připojení a přihlášení k databázi a bude definovat základní komunikaci s databází. V ukázkové aplikaci jsme pro komunikaci s databází vytvořily třídu, která se jmenuje MainDatabase. Obsahuje základní databázové operace, které jsou společné pro všechny modely. Pro připojení a odpojení od databáze slouží funkce login() a logout(). Při připojování se předávají jako parametry uživatelské jméno, heslo, adresa databázového serveru, port a service name. Tato třída se stará nejen o komunikaci s databází, ale také o udržování spojení. Pokud bylo spojení jednou úspěšně navázáno, muže být kdykoli obnoveno funkcí reconnect(). Třída MainDatabase je vidět na obrázku 5.2.

Každý model musí komunikovat s databází, proto při vytváření instance jednotlivých modelů předáváme MainDatabase již v konstruktoru (viz. obrázek 5.1). Aplikace musí být v době vytváření instancí jednotlivých modelů již připojená k databázi (k získání přihlašovacích údajů slouží dialog ConnectionWindow, který je použit před spuštěním samotné aplikace). Každá část rámce má definované rozhraní modelu, které je nutné implementovat. Nyní bude popsána implementace jednotlivých modelů v ukázkové aplikaci. Nejdůležitější funkcí každého modelu je funkce s názvem getData(), která načítá data z konkrétní databáze. Funkce používá typ návratové hodnoty a parametru

LinkedList. Konečný class diagram, zobrazující strukturu rozhraní a tříd pro modely je vidět na obrázku 5.3.



Obrázek 5.2: Třída MainDatabase



Obrázek 5.3: Class diagram zobrazující strukturu tříd a rozhraní pro modely

5.1.1 Model pro multimediální data

Model pro multimediální databázi načítá z databáze záznamy obsahující obrazové data. Načtené obrázky jsou potřeba nejprve načíst jako pole bytů a následně z nich vytvořit obrázek typu ImageIcon. ImageIcon je později v controlleru upravena tak, aby její velikost odpovídala velikosti panelu, ve kterém je zobrazená a následně je převedena na typ BufferedImage. S tímto typem obrázku se lépe pracuje při samotném vykreslování. Další funkcí modelu je načítaní dat dle zvoleného filteru a v ukázkové aplikaci je přidána funkce vyhledávání obrázků dle podobnosti s jiným obrázkem. Model se také stará o stránkování načítaných dat. Stránkování je v databázích Oracle trošku problém, protože v době vytváření jazyka Oracle SQL nebyly ve standardu definovány funkce pro stránkování

(jako např. LIMIT v jazyce MySQL nebo OFFSET/FETCH v jazyce Java SQL). Bylo nutné stránkování vytvořit pomocí dvou subdotazů. Ukázkový SQL dotaz je vidět na obrázku 5.4. Tento SQL dotaz zobrazí záznamy z databáze, které mají uložený obrázek a mají pořadí 11 až 20, tzn. dotaz vrátí 10 záznamů s obrázky. Obrázkem se myslí inicializovaný objekt ORDImage, který nemusí obsahovat konkrétní data, ale musí obsahovat základní strukturu dat. Načtené položky tedy nemusí obsahovat obrázek.

```
SELECT *
FROM
( SELECT R.*, ROWNUM rnum
  FROM
  ( SELECT * FROM monster_tab
    WHERE picture IS NOT NULL
  ) R
  WHERE ROWNUM <= 20
)
WHERE rnum >= 11
```

Obrázek 5.4: SQL dotaz pro stránkování dat

Jak bylo výše uvedeno funkce getData() načítá záznamy i dle podobnosti s jiným obrázkem. Dříve používaná funkce pro porovnání podobnosti OrdImageSignature je v aktuální Oracle multimediální databázi zastaralá a místo ní se používají funkce objektu StillImage. Jedná se o databázový typ reprezentující obrázek stejně jako ORDImage, ale nedefinuje některé funkce pro práci s obrázkem. Proto není možné ho plně nahradit. Práce se StillImage a s funkcemi pro porovnání podobností (nejen podobnost) není implementovaná v aktuální verzi java knihovny na práci s multimediální databází. Proto je nutné veškeré operace provádět pomocí databázových dotazů a nejlépe pomocí funkcí nebo procedur. Model dále implementuje funkce pro načtení detailu, jeho editaci a smazání.

Pro porovnání dvou obrázků jsme vytvořily dvě databázové funkce a dva uživatelské datové typy. Kompletní SQL script je obsahem přílohy 1. První funkce get_si_featurelist() vytvoří objekt StillImage z nahraného obrázku a následně vytvoří strukturu si_featurelist, která je nutná pro definování hodnot pro porovnávání obrázků. Druhá funkce get_similiar() načte nový obrázek z dočasné tabulky (testTable) a pomocí první funkce z něj vytvoří si_featurelist. Dále pak prochází jednotlivé záznamy v databázi a pomocí databázové funkce si_scorebyftrlist(), porovnává podobnosti. Funkci se předá vytvořený si_featurelist a StillImage vytvořený z obrázku každého záznamu v databázi. Funkce si_scorebyftrlist() vrací hodnotu podobnosti, kde menší hodnota znamená vetší podobnost. Nakonec naše funkce vrátí pole identifikátorů záznamů, které se vešli do zadané hranice podobnosti. Toto pole je pak použito v podmínce při načítání dat. Část SQL dotazu využívajícího výše popsanou funkci je možné vidět na obrázku 5.5.

```
AND id IN (SELECT * FROM TABLE (get_similiar))
```

Obrázek 5.5: Část SQL dotazu využívajícího databázovou funkci get_similiar()

Jak bylo uvedeno výše, databázové funkce využívají dočasnou tabulku pro uložení obrázku, podle kterého chce uživatel vyhledávat. O vytvoření této tabulky a vložení nového obrázku je stará funkce insertTemporalImage().

5.1.2 Model pro prostorová data

Model pro prostorovou databázi musí pracovat s daty různých typů. Nejedná se o různé data z pohledu databázové tabulky, ale o prostorová data, která jsou zadána různým počtem bodů a jsou zkrátka definována různými způsoby. Model musí tyto data načíst a podle jednotlivých identifikátorů rozpoznat typ grafického objektu. Kompletní definice datových typů v prostorové databázi je dostupná v dokumentaci (viz. [2]). Data jsou v databázi uložena ve střech strukturách. První obsahuje pole informací o objektu a definuje jeho typ, díky těmto informacím určíme o jaký objekt se jedná. Další definuje souřadnice bodu, používá se pro určení umístění grafického objektu prezentovaného jako bod. Poslední obsahuje seznam bodů definujících umístění grafického objektu. Tento seznam je dynamický, každý objekt obsahuje rozdílný počet bodů a některé jich mohou mít libovolně mnoho (např. se jedná o polygon).

Při načítání dat je tedy nutné u každého záznamu určit typ daného objektu a podle tohoto typu určit počet bodů, kterými je definován. Následně je pro každý grafický objekt vytvořena instance příslušné třídy (viz. kapitola 4.2.2) do které jsou body uloženy. Slouží k tomu funkce setLocation(), která je pro každou třídu definována jinak kvůli rozdílnému počtu bodů. Každému objektu je taky přiřazen identifikátor, aby jej bylo možné použít při dalších databázových operacích. Dále je do každého objektu vložena reference na objekt na zobrazování pozadí. Ten je nutný pro zjišťování informací o aktuálním přiblížení atd. (viz kapitola 4.2.1). Pokud je grafický objet typu bod, je mu navíc přiřazen obrázek, který bude zobrazen na mapě. Právě popsané načítání dat, které aplikace zobrazuje na mapě, obstarává funkce getData(). Tato funkce vrací proměnnou typu LinkedList, která obsahuje seznam všech načtených prvků.

```
else if (item instanceof FRDSpatialGRectangle)
{
  Point[] points = ((FRDSpatialGRectangle)item).getLocation();
  Point p1 = points[0];
  Point p2 = points[1];
  sql = "UPDATE map_objects SET geometry="
      + "SDO_GEOMETRY(2003, NULL,NULL, "
      + "SDO_ELEM_INFO_ARRAY(1, 1003, 3),"
      + "SDO_ORDINATE_ARRAY("+ p1.x +","+ p1.y +", "+ p2.x +","+ p2.y +"))"
      + "WHERE object_id = " + id;
}
```

Obrázek 5.6: Editace pozice čtverce

Malý problém nastává u přidávání a editace prvku (funkce addItem() a editItem()), protože parametrem těchto funkcí je objekt třídy FRDSpatialGItem, musíme napřed zjistit, o který grafický objekt se jedná. Protože nelze objektům přiřadit jednoznačné identifikátory, protože při přidávání

vlastních tříd pro nové grafické objekty by jsme neměly definované nové identifikátory, byli jsme nuceni rozpoznávání o jakou podtřídu se jedná dělat pomocí podmínek za použití konstrukce „instanceof“. Po zjištění o jakou třídu se jedná už nebyl problém načíst konkrétní data a vložit či editovat je v databázi. Konkrétní ukázka implementace je vidět na obrázku 5.6, jedná se o editaci souřadnic čtverce. Každý třída implementuje funkci getLocation, která načte všechny souřadnice objektu.

Model dále implementuje funkce pro načtení a uložení dalších informací o konkrétním objektu (select() a edit()), které nepracují se souřadnicemi, a tak jsou tyto funkce společné pro všechny typy grafických objektů. Poslední funkcí, kterou model implementuje je funkce getTypes(), která načte seznam z databáze multimediálních dat. Tento seznam je použit v ComboBoxu pro určení typu objektu podle multimediální databáze.

5.1.3 Model pro temporální data

Temporální model má za úkol načítat data z databázové tabulky, ve které sou uloženy časové údaje. Konkrétně se jedná o čas platnosti databázových dat. V ukázkové aplikaci je časová osa definována pouze roky, jedná se o celočíselné údaje od 0 do 500. Při standardním načítání, editování a mazání údajů se časovými údaji nijak speciálně nepracuje. Speciální operace s temporálními daty je možné přidat do výsledné aplikace. V ukázkové aplikace je impementována funkce, která odstraní určitý časový úsek u vybraných záznamů.

Temporální model obsahuje definici dat načítaných z tabulky, jedná se o atributy columns a columnNames. Atribut columns obsahuje názvy sloupců v databázové tabulce a první položka značí jednoznačný identifikátor v tabulce. Atribut columnNames definuje textové názvy sloupců, které jsou zobrazeny v záhlaví tabulky. Model také definuje, které sloupce je možné v rámci tabulky editovat, jedná se o atribut canEdit, který je definován jako pole hodnot typu boolean (true znamená, že daná hodnota se může editovat).

Model kromě základního načtení všech dat pomocí funkce getData(), která načítá data i podle zadaného fitru, implementuje také funkci pro editace konkrétní buňky tabulky updateColumn(), která podle databázového identifikátoru edituje záznam v databázi. Dále pak funkci pro smazání vybraných řádků tabulky deleteColumns(), které je předáno pole identifikátorů a výše zmíněnou funkci pro smazání intervalu pro konkrétní id. Tato funkce upraví časy platnosti a v případě potřeby rozdělení intervalů, duplikuje databázový záznam.

5.2 Nové funkce aplikace

Protože rámec neimplementuje všechny požadované funkce (a ani nemůže), je nutné do existujících panelů vložit nové ovládací prvky, reprezentující nové funkce a implementovat ovládání těchto prvků.

Jako ukázku přidání nové funkce aplikace jsme v temporální části aplikace vytvořily novou operaci. Jedná se o smazání intervalu u vybraných záznamů. Pro tuto funkci jsme přidaly grafické prvky do public komponenty jLayeredPane2 umístěné ve FRDTemporal. Konkrétně se jednalo o tří komponenty JLabel, dvě JTextField a jedno tlačítko JButton. Pro přidání komponent jsme ve třídě TemporalController implementovaly funkci AddComponents(), která je volaná z konstruktoru. Tlačítku jsme nastavili atribut ActionCommand pro rozpoznání ve funkci actionPerformed(), ve které jsme také implementovaly kód pro danou funkci. Přidaní nové operace s daty v tabulce je tedy velice rychlé. Funkce pro konkrétní smazání intervalu je implementována v modelu.

Do prostorové části jsme přidaly funkci pro načtení informací o vybraném grafickém objektu. Tyto informace se zobrazí v ovládacím panelu prostorové části aplikace. Nové komponenty jsme opět přidali ve funkci AddComponents(). Jedná se například o komponentu JcomboBox, která je naplněna daty z multimediální databáze. Konkrétně se jedná o název a tato informace určuje typ grafického objektu. Při vybrání objektu v mapě je zavolána funkce selectedItem() v controlleru. Tuto funkci jsme přeimplementovali a doplnily o načtení výše uvedených dat. Při stisku tlačítka Uložit jsou informace vyplněné v grafických komponentách uloženy do databáze.

V multimediální části bylo nutné implementovat funkce pro přidání a editování položek. Pro tento účel byl vytvořen dialog DetailWindow, který dodatečné informace zobrazuje. Dialog je zobrazen při stisku tlačítka detail u některé položky (tehdy je také předvyplněn informacemi z databáze) a nebo při stisku tlačítka Přidat. Další možností rozšíření je přidat funkční tlačítko do konkrétního panelu zobrazující obrázek. Pro tento účel byla vytvořena třída MultimediaItem rozšiřující FRDMultimediaItem. V konstruktoru nové třídy jsme vložily nové grafické tlačítko. Jak již bylo uvedeno v kapitole 4.1, o přidání Listeneru k tlačítku se nemusíme starat. Akci tohoto tlačítka zpracuje funkce actionPerformed v controlleru. Pro inicializaci těchto panelů jsme v controleru přeimplementovali funkci initMultimediaItems, kde jsme do již existujících proměnných místo původní třídy inicializovaly MultimediaItem.

5.3 Zhodnocení

Použití rámce při vývoji aplikace výrazně snižuje dobu, za kterou je aplikace schopna zobrazovat, editovat či mazat data s postrelačních databází. Pro základní funkce implementované v rámci stačí pouze inicializovat jednotlivé komponenty (jak je vidět na obrázku 5.1) a implementovat jednotlivé modely a třídu komunikující s databází. Při vývoji bez použitý rámce zabere nejvíce času implementace panelu zobrazující prostorová data, protože je třeba rozlišit vykreslování jednotlivých grafických objektů.

V multimediální části je díky rámci usnadněno zobrazování dat a přidávání nových funkcí, ale je nutné kompletně implementovat přidávání a editaci. Snížení doby implementace multimediální části není tak výrazné jako u prostorové části.

V temporální části rámce nebylo jednoduché implementovat funkce pracující s daty, protože struktura databáze není při vývoji rámce známá. Implementované funkce bylo tedy nutné zobecnit, aby se dalo pracovat s libovolnými daty načtenými z databáze. Konkrétní funkce na práci s temporálními záznamy je proto nutné implementovat, až ve výsledné aplikaci. Implementace temporální části ve výsledné aplikaci tedy zabere dost času a oproti vývoji bez použití rámce je časový rozdíl nejmenší.

I přes některé neimplementované funkce rámce (hlavně díky neznalosti databázové struktury), je vývoj aplikace mnohem rychlejší a to nejen díky implementovaným funkcím, ale i díky připravenému grafickému vzhledu do kterého je možné umístit nové prvky. Všechny tři části rámce jsou implementovány nezávisle a případné propojení mezi nimi, lze implementovat ve výsledné aplikaci. To umožňuje zvolit, zda chceme použít všechny části nebo jen některé a zbytek si implementovat pomocí vlastních tříd.

Jako rozšíření rámce by bylo možné navrhnout a implementovat obecné funkce pro práci s temporální částí, například s možností využití TSQL2 interpretu nad relační databází (viz. [1]). Jako další rozšíření by mohlo být implementované rozhraní FRDSpatialMapBackground, které by zobrazovalo mapu pomocí externí knihovny (např. Google maps).

Možné rozšíření v ukázkové aplikaci by mohlo být optimalizování funkcí pro porovnávání obrázků v multimediální části, protože v s aktuálními databázovými funkcemi to trvá docela dlouho. Dále pak rozšířit prostorovou část o funkce umožňující výběr barev u konkrétních prvků (popř. ikony u bodu). Temporální část by bylo možné rozšířit o další funkce pracující s časovými údaji.

6 Závěr

Cílem této diplomové práce je vytvořit rámec implementující komponenty, které urychlí vývoj aplikací komunikujících s post-relačními databázemi. Při specifikaci rámce jsme hledaly knihovny, které by mohli podobnou problematiku implementovat, ale žádná knihovna usnadňující vývoj podobných aplikací neexistuje.

Po specifikaci rámce bylo nutné nastudovat technologie, které jsou nutné pro návrh a následnou implementaci rámce. Bylo nutné nastudovat post-relační databáze (konkrétně multimediální, prostorovou a temporální), které nabízí firma Oracle Corporation (viz. kapitola 2.2). Dále bylo nutné nastudovat metodologii vývoje softwaru Rapid Application Development popsaný v kapitole 2.3, a umožnit budoucím uživatelům rámce použít vytvořený rámec pro rychlý vývoj aplikace. Proimplementaci byl zvolen jazyk Java a jeho verze Java SE (kapitola 2.4), která je určená pro desktopové aplikace. S tím souvisí nastudování možností implementace nových komponent tak, aby je bylo možné vkládat přímo ve vývojovém prostředí NetBeans (viz. kapitola 2.5). Nové komponenty bylo nutné zaintegrovat do hierarchie grafické knihovny Java Swing, a protože každá komponenta je implementována podle vzoru MVC, bylo nutné tento vzor nastudovat. Popis MVC je v kapitole 2.6.

Dalším krokem byl samotný návrh rámce, který je popsán v kapitole 3. Rámec obsahuje tři komponenty, každá komponenta pro jeden typ databáze. Bylo nutné zvolit rodičovskou komponentu (tzn. vybrat třídu z knihovny Java Swing) a navrhnout hierarchii nových komponent. Nové komponenty budou implementované podle vzoru MVC. Dle zadání má implementaci metod pracujících s databází provádět až případný uživatel rámce (programátor), proto jsme museli navrhnout rozhraní, podle kterého bude tuto komunikaci implementovat. Jako poslední krok byl návrh samotných komponent. V kapitolách popisující návrh jednotlivých komponent je uvedeno jak budou jednotlivé komponenty fungovat a je zhruba určeno, jaké grafické komponenty na to budou použity. Dále je popsáno použití rámce pomocí diagramu případů užití.

Po nastudování všech použitých technologií nutných k zvládnutí návrhu a implementace a vytvoření návrhu rámce přichází na řadu implementace. Kapitola 4 popisuje implementaci rámce, je rozdělena na tří podkapitoly. Je zde popsán postup při implementaci a rozdíl oproti návrhu. Každá z podkapitol popisuje implementaci jedné části rámce, všech použitých tříd a popis některých funkcí.

Po implementaci rámce bylo nutné vytvořit ukázkovou aplikaci, které daný rámec využívá a zhodnotit rozdíl doby vývoje aplikace oproti vývoji bez rámce. Popis ukázkové aplikace je v kapitole 5. Ukázková aplikace používá všechny třídy rámce a popisuje implementaci jednotlivých rozhraní. Hlavní částí implementace ukázkové aplikace je vytvoření modelů, které načítají data z databáze. Postup při implementaci je popsán v kapitole 5.1. Dále kapitola 5 popisuje přidávání nových funkcí, které rámec neobsahuje (viz. kapitola 5.2) a zhodnocení použitelnosti rámce (kapitola 5.3).

Aplikace i rámec byl vyvíjen na operačním systému Windows 7, ale díky jazyku Java není problém knihovnu i aplikace spustit na ostatních podporovaných operačních systémech. Výsledek této práce bude sloužit jako ukázková aplikace pro vývoj aplikací používajících a zobrazujících data z postrelačních databází. Knihovna vytvořená z implementovaného rámce může být použita pro usnadnění vývoje takovýchto aplikací, a je tedy přínosná na poli vývoje projektů a aplikací nejen v rámci studia na vysoké škole.

Literatura

[1] TOMEK, Jiří. *TSQL2 interpret nad relační databází: Processor of TSQL2 on a Relational Database System*. Brno: Vysoké učení technické, Fakulta informačních technologií, 2009. Dostupné z WWW: <http://www.fit.vutbr.cz/study/DP/DP.php?id=8603&file=t>. Diplomová práce. FIT VUT v Brně.

[2] ORACLE. *Oracle Documentation* [online]. 2012 [cit. 2013-01-07]. Dostupné z WWW: <http://www.oracle.com/technetwork/indexes/documentation/index.html>

[3] NETBEANS. *NetBeans IDE* [online]. 2012 [cit. 2012-12-20]. Dostupné z WWW: <http://netbeans.org>

[4] NETBEANS. *NetBeans* [online]. 2012 [cit. 2013-01-07]. Dostupné z WWW: <http://netbeans.org/about/history.html>

[5] LIU, Yu-Cheng. *Smalltalk, objects and design: The venerable model-view-controller*. Vyd. 1. New York: Excel, 1996, s. 115-126. ISBN 1-58348-490-6.

[6] KOLÁŘ, Dušan. *Pokročilé databázové systémy* [online]. 2012 [cit. 2012-12-27]. Studijní opora. Dostupné z WWW <https://www.fit.vutbr.cz/study/courses/PDB/private/lectures/prednaskyPRD.pdf>

[7] What is Rapid Application Development?. V: *CASEMaker Inc* [online]. 2000 [cit. 2013-01-07]. Dostupné z WWW: <http://www.casemaker.com/download/products/totem/rad_wp.pdf>

[8] Rapid application development: History. V: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012 [cit. 2013-01-07]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Rapid_application_development>

[9] HOCK-CHUAN, Chua. Java Programming Tutorial: Programming Graphical User Interface (GUI). *Nanyang Technological University, Singapore* [online]. 2012 [cit. 2013-01-07]. Dostupné z WWW: <http://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI_2.html>

[10] *Java SE Documentation* [online]. 2012 [cit. 2013-01-07]. Dostupné z WWW: <http://docs.oracle.com/javase/>

[11] Java Swing. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012 [cit. 2012-12-26]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Swing_(Java)>

[12] GROUCHNIKOV, Kirill. How to Write a Custom Swing Component. *Java.net* [online]. 2007 [cit. 2013-01-02]. Dostupné z WWW: <http://today.java.net/pub/a/today/2007/02/22/how-to-write-custom-swing-component.html>

[13] MVC a další prezentační vzory. *Zdrojak.cz* [online]. 2009 [cit. 2013-01-02]. Dostupné z WWW: <http://www.zdrojak.cz/serialy/mvc-a-dalsi-prezentacni-vzory>

[14] NĚMEC, Miroslav. Případová studie na použití architektonických vzorů [online]. 2011 [cit. 2013-01-07]. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Radek Ošlejšek. Dostupné z WWW: <http://is.muni.cz/th/172853/fi_m/>.

[15] Introduction to Java Swing. V: *Zentut.com* [online]. 2012 [cit. 2013-01-07]. Dostupné z WWW: <http://www.zentut.com/java-swing/introduction-to-java-swing/>

[16] Google Code: mygrid-labs. Project Hosting on Google Code [online]. [cit. 2013-05-20]. Dostupné z WWW: <https://code.google.com/p/mygrid-labs/>

Seznam příloh

Příloha 1 : SQL script

```
CREATE OR REPLACE FUNCTION get_si_featurelist (image IN BLOB) RETURN si_featurelist
IS
    l_img_obj   si_stillimage;
    l_avgcolor  si_averagecolor;
    l_colorhist si_colorhistogram;
    l_poscolor  si_positionalcolor;
    l_texture   si_texture;
    l_featurelist   si_featurelist;
BEGIN
    l_img_obj := NEW si_stillimage(image);
    l_avgcolor    := NEW si_averagecolor(l_img_obj);
    l_colorhist   := NEW si_colorhistogram(l_img_obj);
    l_poscolor    := NEW si_positionalcolor(l_img_obj);
    l_texture     := NEW si_texture(l_img_obj);
    l_featurelist := NEW si_featurelist(l_avgcolor, 1, l_colorhist, 1, l_poscolor, 1, l_texture, 1);
    RETURN l_featurelist;
END;
/
CREATE OR REPLACE TYPE array IS object (id number);
/
CREATE OR REPLACE TYPE arrayType IS table of array;
/
CREATE OR REPLACE FUNCTION get_similiar RETURN arrayType IS
    l_emp_tab arrayType := arrayType();
    n integer := 0;
    l_featurelist si_featurelist;
    l_blob BLOB;
BEGIN
    SELECT m.img.source.localdata INTO l_blob FROM testTable m WHERE ROWNUM = 1;
    l_featurelist := get_si_featurelist(l_blob);
    for r in (SELECT m.id FROM monster_tab m
        WHERE m.picture.contentLength IS NOT NULL AND
        si_scorebyftrlist(l_featurelist, new si_stillimage(m.picture.source.localdata)) < 10)
    LOOP
        l_emp_tab.extend;
        n := n + 1;
        l_emp_tab(n) := array(r.id);
  END LOOP;
  RETURN l_emp_tab;
END;
/
```

Příloha 2 : CD

/SRC – Zdrojové kódy programu a knihovny
/DOC – Dokumentace a manuál k programu a knihovně
/DP – Diplomová práce ve formátu ODT a PDF
/BIN – Jar soubory pro knihovnu a ukázkovou aplikace
README.txt – všeobecné informace