VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

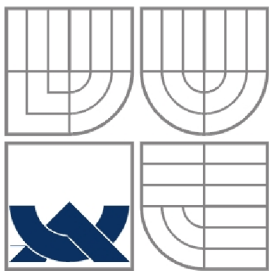
# DISTRIBUOVANÝ INFORMAČNÍ SYSTÉM ZALOŽENÝ NA SÉMANTICKÝCH TECHNOLOGIÍCH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

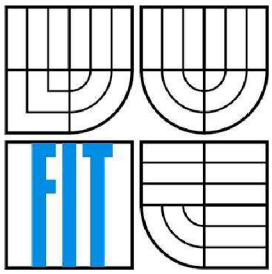
AUTOR PRÁCE
AUTHOR

BC. JAN HAVLENA

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# DISTRIBUOVANÝ INFORMAČNÍ SYSTÉM ZALOŽENÝ NA SÉMANTICKÝCH TECHNOLOGIÍCH

DISTRIBUTED INFORMATION SYSTEM BASED ON SEMANTIC TECHNOLOGY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. JAN HAVLENA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. RADEK BURGET, PH.D.

BRNO 2010

*Zadání diplomové práce/10723/2009/xhavle02*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav informačních systémů Akademický rok 2009/2010

# Zadání diplomové práce

Řešitel: **Havlena Jan, Bc.**
Obor: Informační systémy
Téma: **Distribuovaný informační systém založený na sémantických technologiích**
**Distributed Information System Based on Semantic Technology**
Kategorie: Web

Pokyny:

1. Seznamte se s pojmem ontologie, způsobem návrhu ontologií a s existujícími prostředky pro jejich reprezentaci.
2. Prostudujte technologie sémantického webu s ohledem na výměnu informací mezi informačními systémy. Soustřeďte se na RDF a související technologie.
3. Navrhněte architekturu distribuovaného informačního systému umožňujícího zadávání, procházení a sdílení informací pomocí ontologií.
4. Na základě konzultace s vedoucím navrhněte ontologii popisující účast osob na různých konferencích a seminářích. Využijte v maximální míře již existující ontologie.
5. Implementujte navržený informační systém s ohledem na zpracování informací reprezentovaných pomocí navržené ontologie.
6. Proveďte testování systému.
7. Zhodnoťte dosažené výsledky.

Literatura:

- Hák, L.: Sdílení dat mezi informačními systémy založené na ontologiích, diplomová práce, Brno, FIT VUT v Brně, 2009
- Svátek, V.: Ontologie a sémantický web. Dokument dostupný online na URL: http://nb.vse.cz/~svatek/onto-www.pdf
- Svátek, V., Labský, M.: Objektové modely a znalostní ontologie. Dokument dostupný online na URL: http://nb.vse.cz/~svatek/obj03fi.pdf
- Svátek, V., Vacura, M.: Ontologické inženýrství. Dokument dostupný online na URL: http://nb.vse.cz/~svatek/dkon07final.pdf

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 až 3

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

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

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

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

doc. Dr. Ing. Dušan Kolář
*vedoucí ústavu*

# Abstrakt

Tato diplomová práce se zabývá návrhem a implementací distribuovaného informačního systému, jehož distribuce dat je založena na sémantických technologiích. Jsou v ní rozebírány technologie sémantického webu se zaměřením na výměnu informací mezi informačními systémy a s tím související pojmy, zejména ontologie, ontologické jazyky a technologie RDF. Dále je v práci popsán návrh vlastní ontologie, pomocí níž jsou data předávaná mezi systémy popsána, a technologie použité pro implementaci distribuovaného informačního systému. Nejvýznamnějšími z nich jsou JavaServer Faces a Sesame.



# Klíčová slova

Distribuovaný informační systém, sémantické technologie, sémantické weby, ontologie, ontologické jazyky, RDF, OWL, JavaServer Faces, Sesame.



# Abstract

This master's thesis deals with the design of a distributed information system, where the data distribution is based on semantic technologies. The project analyzes the semantic web technologies with the focus on information exchange between information systems and the related terms, mainly ontologies, ontology languages and the Resource description framework. Furthermore, there is described a proposal an ontology which is used to describe the data exchanged between the systems and the technologies used to implement distributed information system. The most important of them are Java Server Faces and Sesame.



# Keywords

Distributed information system, semantic technology, semantic web, ontology, ontology language, RDF, OWL, JavaServer Faces, Sesame.



# Citace

Havlena Jan: Distribuovaný informační systém založený na sémantických technologiích, diplomová práce, Brno, FIT VUT v Brně, 2010

# Distribuovaný informační systém založený na sémantických technologiích

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Havlena
20.5.2010

## Poděkování

Tímto chci poděkovat vedoucímu své diplomové práce Ing. Radku Burgetovi, Ph.D. za odborné vedení, jeho hodnotné rady, připomínky a konzultace, které mi pomohly k vypracování této práce.

# Obsah

# 1 Úvod

V dnešní době, kdy informace jako takové nabývají stále větší důležitosti a jejich množství se stále zvyšuje, je třeba mít tyto informace pod pomyslnou kontrolou. S přibývajícími zkušenostmi odborníků jsou vyvíjeny nové vhodnější formy reprezentace dat, z nichž některé se stávají novými standardy. Nových technologií je třeba hlavně z důvodu rychlého růstu dat na světové síti internetu. Tato data jsou do této chvíle reprezentována různými způsoby, což může vést k nejednoznačnosti jejich sémantiky. Jejich strojové zpracování se stává obtížným. Tomuto by měl předejít nový standardizovaný způsob jejich reprezentace, který vymezí datům jejich sémantiku a díky němuž bude možné data jednoduše prohledávat a provádět s nimi další strojové zpracování. Jednou z technologií, které tento způsob reprezentace dat v rámci webového prostředí umožňují, je sémantický web.

Sémantický web, je web, který obsahuje nejen data, ale i jejich sémantiku. Jeho základním charakterem je založení na xml formátu a ontologiích. Podrobněji bude popsán v následující práci. Do zmíněné skupiny strojového zpracování dat patří i získávání a distribuce dat v prostředí informačních systémů, kde je také výhodné využití sémantických technologií. Na tuto problematiku, tedy předávání informací mezi informačními systémy prostřednictvím technologie sémantického webu a technologie s tím související, se zaměřuje tato práce.

Součástí práce je i vytvoření praktické aplikace distribuovaného informačního systému využívající sémantických technologií pro sdílení dat. V tomto dokumentu se budeme zabývat jejím návrhem a implementací.

Druhá kapitola této práce pojednává o ontologiích, které jsou základem sémantického webu. Nejprve je rozebrán samotný význam slova ontologie. Poté jsou zde rozebrány prvky struktury ontologií a nejvýznamnější členění ontologií. V závěru kapitoly jsou popsány některé z existujících prostředků pro reprezentaci ontologií.

V třetí kapitole se dostáváme k samotným sémantickým technologiím a sémantickému webu. Popisuji zde s tím související RDF, založený na tvrzení. Dále charakterizujeme RDF Schema, což je první jazyk vycházející z RDF. Následně se zabýváme jazykem OWL, popisujeme jeho vlastnosti, strukturu a konstrukcí jeho syntaxe, ve které hraje svou roli i jazyk RDF.

Čtvrtá kapitola se zabývá návrhem distribuovaného informačního systému, založeném na sémantických technologiích. Nejprve jsou zde popsány konkrétní požadavky na systém, které jsou zanalyzovány a shrnuty. Následuje navržení architektury. Dále je navrženo ukládání dat v systému. V poslední části této kapitoly je podrobně popsán mechanismus distribuce dat založený na sémantických technologiích.

Pátá kapitola představuje některé již existující ontologie a posléze se zabývá návrhem ontologie používané v rámci distribuce dat mezi informačními systémy.

Šestá kapitola se zabývá implementací navrženého distribuovaného informačního systému. Kapitola je rozdělena do dvou podkapitol. V první z nich jsou představeny technologie, které jsou použity pro implementaci systému. Jsou to především architektura MVC, framework JavaServer Faces a sémantické úložiště Sesame 2. V druhé části je popsána samotná implementace systému. Jsou v ní rozebrány vytvořené implementační celky. Sedmá kapitola popisuje postup testování systému.

Osmá závěrečná kapitola shrnuje celou práci. Zabývá se zhodnocením dosažených výsledků, přínosu pro řešitele a možností nasazení vytvořeného distribuovaného informačního systému pro správu seminářů a konferencí do praxe.

# 2 Ontologie

V této kapitole se budeme nejprve zabývat samotným významem pojmu ontologie. Následně si představíme základní prvky struktury ontologií, některé z významnějších způsobů jejich členění a několik existujících prostředků vytvořených pro její reprezentaci. Informace k této kapitole, pokud nebude v příslušné části uveden jiný zdroj, byly čerpány z [1].

Pojem ontologie má svůj kořen ve filosofii. Ontologie z pohledu filozofie však neznamená to samé jako ontologie, která je předmětem ontologického inženýrství. Filozofický význam slova je chápán jako nauka o „bytí“, eventuálně soubor znalostí vůči člověku nezávisle popisující objekty, jevy a zákonitosti světa. Druhým pojetím významu slova ontologie je popis existence objektů a vztahů mezi nimi, tedy reprezentace znalostního systému. Z tohoto pojetí je odvozena ontologie, kterou se budeme zabývat v další části textu, a kterou z našeho pohledu chápeme jako to, co může být reprezentováno v informačním systému. K podrobnějšímu popisu se dostaneme v další části dokumentu.

Jednou z oblastí použití ontologií je proces vytváření znalostních aplikací, kde jsou z hlediska znalostního inženýrství ontologie chápány jako abstraktní popisy znalostního systému. Tyto popisy jsou na finální reprezentaci a implementaci znalostí relativně nezávislé. Důležitou vlastností těchto popisů vytvářejících modely je to, že mohou být sdíleny více procesy v jedné aplikaci a použity v různých projektech, které mezi sebou nemají žádnou vazbu. Jedná se tedy o způsob použití ontologií jako nástroje pro sdílení významu pojmů v určité oblasti zahrnující její opakované použití.

V rámci použití ontologií je důležitá formalizace. Jedná se o standardizaci používaných pojmů s přesně definovanou syntaxí. Pokud by nebyla tato vlastnost při vytváření a používání ontologií brána v potaz, vznikaly by různé ontologie popisující stejné doménové oblasti různým způsobem, což by mohlo vést k neurčitosti sémantiky v ontologiích definovaných pojmů. Je tedy důležité, aby komunity pracující s ontologiemi byly navzájem v kontaktu [2].

Účel použití ontologií je možné rozčlenit do následujících okruhů. Prvním z možných využití ontologií je podpora porozumění mezi lidmi. Jedná se například o způsob ulehčení komunikace mezi experty a znalostními inženýry. Další příležitostí využití je podpora komunikace mezi počítačovými systémy. Posledním zmíněným okruhem použití je usnadnění návrhu znalostně-orientovaných aplikací. V těchto uvedených scénářích má ontologie široké spektrum uplatnění, při řešení jejich problémových částí.

## 2.1 Prvky ontologií

Základní struktura ontologií je ve všech projektech, jazycích a nástrojích relativně stejná pro všechny typy ontologií. Není tomu tak však i u používaných terminologií, které se značně liší, což znesnadňuje orientaci. V následujícím textu popíšeme základní prvky tvořící ontologie a terminologickou odlišnost používaných pojmů.

**Třídy** jsou základem znalostních ontologií. Jsou jimi definovány množiny konkrétních objektů. V některých formalismech odpovídá třída termínu **konceptu**, **kategorie** a **rámci**, který reprezentuje základní součásti v systémech umělé inteligence. Ontologické třídy na rozdíl od tříd používaných například v objektově orientovaných programovacích jazycích nezahrnují procedurální metody. Na množině tříd může být použita hierarchie, také stromová struktura a v praxi hojně využívaná dědičnost.

**Individuum** je naproti třídě odrazem konkrétního reálného objektu světa. Jako rovnocenný termín individua je často uváděn termín instance. **Instance** je však vždy svázána s nějakou třídou, což je v rozporu s definicí individua, které se může v ontologii nacházet i bez vazby na konkrétní třídu. Samotné rozhodnutí, zda bude entita obsažená v ontologii, modelována prostřednictvím třídy nebo individua, závisí na úhlu pohledu modelujícího na modelovanou entitu.

**Relace** jsou stejně jako v databázových modelech velice významnou složkou ontologií. Dalšími druhy relací jsou **funkce**, **sloty**, **vlastnosti**, **role** a **atributy**. Relace také mohou vytvářet hierarchická seskupení, podobně jako třídy. Hierarchie je realizována jako vztah množiny n-tic argumentů podřazené relace, která je podmnožinou n-tic argumentů nadřazené relace. Jako příklad můžeme uvést binární relace má-otce a má-předka.

V ontologiích je možné slotům přiřazovat **vlastnosti**, ty můžeme vyjádřit jako **meta-sloty**. Mezi nejčastější meta-sloty patří hierarchický vztah tranzitivní binární relace podřízeného a nadřízeného slotu. Další možnou binární relací, kterou vyjadřují meta-sloty je inverzní relace. K unárním relacím, které patří k zbývajícím vyjádřením relačních vztahů meta-slotů, náleží symetrie, tranzitivita, funkčnost či inverzní funkčnost. Do vlastností slotů dále patří definiční obor a obor hodnot, které jsou vymezeny prostřednictvím konkrétních tříd. Zmíněné vlastnosti přísluší mezi globální omezení, která se vztahují na sloty bez ohledu na jeho použití. V některých případech je třeba vymezit hodnotu slotu na konkrétní třídu. Zejména se jedná o omezení oboru hodnot eventuálně kardinality. Takováto vlastnost přísluší mezi lokální omezení a označuje se jako **facety**.

Argumenty relací mohou být, kromě samotného popisu vztahů mezi n-ticí objektů, i **primitivní hodnoty**, které žádnému objektu neodpovídají. V tomto případě se nejedná o objektové sloty, ale o sloty **dato-typové**. Tyto sloty mají vymezený obor hodnot výpočtem, číselným intervalem nebo základním datovým typem jakou jsou například string, integer nebo float.

**Axiomy** či **pravidla** jsou logické, výrokové formule vyjadřující například ekvivalenci, substituci tříd či relací, disjunktnost tříd, rozklad třídy na podtřídy a podobně. Axiomy mohou vystupovat zcela samostatně nebo být součástí tříd, podle interpretace konkrétního jazyka. Mimo axiomů se v ontologiích nachází ještě výrazy, které určují příslušnost k třídám nebo relacím.

Ontologie obsahují mimo znalostních konstruktů další **souhrnné údaje**. Tyto údaje jsou nejčastěji umísťovány do hlavičky. Patří sem zejména odkazy importovaných ontologií. Dále dokumentační položky, které většinou informují o autorovi, verzi, času vytvoření a dalších. Logickou interpretaci tyto údaje neovlivňují.

Existují dva problémy komplikující tvorbu a využití ontologií. Jedná se o **problém rozsahu**, který je způsoben velkým množstvím pojmů, jimiž lze vyjádřit požadovanou doménu. Je tedy třeba vybírat takové pojmy, které jsou jasně spjaté s jádrem ontologie. Dalším z toho důvodu vzniklou vlastností při navrhování ontologie je její rozdělení do více nezávislých modelů.

Druhým problémem je **problém interakce**, který souvisí se způsobem používání, na kterém je ontologie závislá, se také někdy nazývá nutnost kompromisu mezi použitelností a znovupoužitelností ontologie. Čím je ontologie specifičtější pro danou aplikaci, tím je hůře znovupoužitelná.

## 2.2 Členění ontologií

Ontologie je pojem, který se vztahuje k mnoha různorodým informačním celkům, v nichž se ontologie člení různými způsoby. Členění odvíjející se od historického použití ontologií v různých oborech je následující:

- **Terminologie či lexikální ontologie** používané v oborech orientovaných na textové zdroje, jako je knihovnictví. Tyto ontologie jsou přirovnávány slovníkům tezaurus.

- **Informační ontologie** rozvíjející databázové konceptuální schémata. Jedná se o nástavby nad primitivními strukturovanými zdroji, které vytvářejí abstraktní koncepty potřebné pro pojmové dotazování. Oproti běžným nástrojům zlepšují možnosti kontroly integrity.
- **Znalostní ontologie** vztahující se k oblasti umělé inteligence a reprezentace znalostí s tím spojené. Ontologie zde představují logické teorie a vazby k reálným objektům, které jsou ve srovnání s informačními ontologiemi pojaty volně.

Jedním ze základních rozdělení ontologií je členění podle předmětu formalizace. Toto členění může mít různé varianty, odvíjející se od pojetí autora. My se budeme zabývat pouze hlavními skupinami.

- **Doménová ontologie** je nejvíce používaná ontologie, zabývající se vždy určitou oblastí, která je vymezena v širším rozsahu jako například celá problematika fungování firmy, nebo v užším rozsahu jako například část funkce firmy jako je proces poskytování úvěrů.
- **Genetické ontologie** jsou ontologie někdy nazývané jako vyšší úrovně. Zachycují obecné znalosti, které nejsou spjaty s konkrétní oblastí. Jedná se například o zachycení problematiky času, či struktury objektů.
- **Úlohové ontologie** neboli znalostní ontologie se nezaobírají znalostmi z reálného světa, ale snaží se znalosti odvozovat. Využití těchto ontologií je hlavně v oboru znalostních úloh a metod jejich řešení. Příkladem použití těchto modelů může být diagnostika, konfigurace nebo plánování.
- **Abstraktní ontologie** jsou specifickým druhem ontologií, které obecně zahrnují jak doménovou, tak úhlovou část. Tento typ ontologií bývá upraven pro konkrétní potřeby aplikací.

I přesto že je důležitou vlastností ontologii její formalizace, jsou používány i ontologie poloformání a neformální, z čehož je odvozeno rozdělení ontologií podle míry jejich formalizace. Jedná se tedy o ontologie formální, poloformální a neformální. Nižší úroveň formality je způsobena použitím přirozeného jazyka. Neformální ontologie je však přítomna i v ontologii formální, kde jsou konstrukty často doplňovány dokumentacemi, ve kterých je možné použití přirozeného jazyka.

## 2.3 Existující prostředky pro reprezentaci ontologií

Pro reprezentaci ontologií bylo od počátku ontologického inženýrství vyvinuto několik desítek jazyků. V rámci seznámení s ontologiemi považuji za vhodné stručně představit několik významnějších, v průběhu ontologického inženýrství vzniklých ontologických jazyků. Informace zmíněné v této podkapitole byly čerpány z [1,3].

- **Cyc** - Jedná se o jeden z prvních pokusů zachycení znalostí o světě ve velkém rozsahu. Usiluje o shromáždění všeobecných znalostí, které by ve znalostních systémech fungovaly komplementárně ke znalostem experimentním, a znemožňovaly vznik absurdních stavů. Pro formální reprezentaci využívá svůj vlastní jazyk CycL, jehož základní notace vychází z funkcionálního jazyka LISP. CycL má plnou vyjadřovací sílu predikátového kalkulu, který kombinuje s prvky rámcových jazyků.
- **Ontolingua** - Byla od začátku koncipována pro výměnu ontologických informací mezi systémy, které interně používají vlastní reprezentaci. Ontolingua ale také pro svou rozšířenost plnila i roli jazyka pro tvorbu ontologií nezávisle na konkrétním znalostním systému. Tento jazyk je koncipován jako nadstavba jazyku KIF (Knowledge Interchange Format). KIF je varianta predikátového kalkulu, který stejně jako Cyc využívá syntaxe LISP. Základní konstrukty jazyka

Ontolingua jsou definice tříd, relací a funkcí. Vymezující podmínky pro příslušnost instancí jsou vyjádřeny v KIF.

- **OCML** – Interpret tohoto jazyka je založen na algoritmech pro prologovské dotazování a dědění v hierarchii tříd. Třídy jsou však důsledně chápány jako unární respektive binární relace, tedy primární (vnitřní) reprezentací jsou Hornovy klauzule. Interpret je implementován v prostředí CommonLisp. To také z části přispělo k nerozšíření OCML mimo Open University, kde vznikl.
- **SHOE** – Jedná se o první jazyk, který vznikl pro účely přidání sémantiky k webovým stránkám. Umožňuje začlenit do zdrojového kódu webových stránek metadata o objektech, jichž se tyto stránky týkají, a samotné ontologie, které definují sémantiku těchto metadat. SHOE zachycuje pouze třídy bez odlišení faced. Je v něm možné definovat relace s libovolnou aritou.
- **DAML+OIL** – Jedná se o jazyk, jehož základem jsou pojmové třídy reprezentované svým jménem nebo třídy anonymní reprezentované určitým logickým výrazem. Pro tvorbu logických výrazů vymezujících třídy se používají konstruktory, jež je možno skládat a tím vytvářet složité výrazy. Náplň ontologie tvoří axiomy. Ty jsou vybudované nad výrazy, které reprezentují třídy.
- **OWL** – Jedná se o modernějšího nástupce DAML+OIL. Je koncipován jako webový sémantický jazyk. Je součástí sémantického webu. Existuje ve třech formách OWL lite, OWL DL a OWL Full. Tomuto jazyku je věnována zvláštní kapitola 3.5.

# 3 Technologie sémantického webu

## 3.1 Sémantický web

Postupem času se stala světová internetová síť jedním z největších zdrojů informací na světě. Aby bylo možné tyto informace jednoduše zpracovávat a využívat, tedy aby byla možnost i strojového zpracování informací v této sféře, je třeba dát těmto nestrukturovaným informacím nějakou standardní formu, která by obsahovala i sémantiku těchto dat. Z toho důvodu začaly být pro reprezentaci webových dat využívány i vyvíjející se ontologické jazyky, které doplňují sémantiku k datům na webových stránkách.

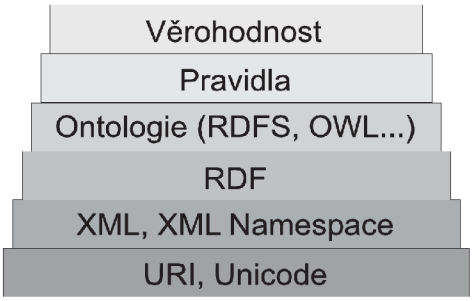
První oficiální představení sémantického webu bylo v květnu roku 2001, kdy Tim Berners-Lee, společně se svými spolupracovníky konsorcia W3C, upozornil na neuspořádanost dat na internetu, reprezentovaných změtí webových stránek, která stále roste a kde je stále obtížnější nalézt relevantní informace. Budoucnost spatřoval v postupném převodu stávajícího webu do takzvaného sémantického webu [2].

Standardizovaný popis webových zdrojů je základním aspektem sémantického webu. Zdroji se rozumí vše dosažitelné pomocí internetu, tedy textové dokumenty, obrázky, zvukové soubory, videosekvence a podobné. Každý zdroj má být vybaven stejnými charakteristickými údaji jako jsou autor, typ zdroje, klíčová slova a podobné. Tyto údaje by umožnily pracovat s internetovou sítí jako s relační databází a dotazovat se na její obsah prostřednictvím jazyků podobných SQL. Vyplývající velkou výhodou by byla vysoká přesnost a relevantnost odpovědí na vyhledávací dotaz [2].

Takový způsob reprezentace dat je možné a výhodné využít v různých IT odvětvích pracující s daty, kdy data není třeba před uložením pro pozdější správnou sémantickou reprezentaci nijak transformovat, protože jejich sémantika je již v nich obsažena, čímž je výhodné použít dotazovací jazyk na tomto způsobu reprezentace dat prostřednictvím sémantických technologií založené.

## 3.2 Koncepce sémantického webu

Sémantický web je koncipován jako souhrn několika na sebe navazujících vrstev. Jedno jeho možné schéma je představeno na obrázku 3.1. Adresování je založeno na URI. XML je použit jako universální syntaktický standard. Primární datová struktura pro uchovávání informací je RDF, které stojí o úroveň výše. Dalším úrovní je ontologie, které informacím dodávají sémantiku. Ontologie mohou být používány pro odvozování. Jejich možnosti jsou však omezené. Konzistence ontologií neboli splnitelnost tříd a příslušnost individuí ke třídám lze testovat prostřednictvím algoritmů, které lze poměrně dobře implementovat. Kromě fakt, která pro individua vyplývají z jejich příslušnosti ke třídě, nelze z ontologií vyvozovat žádná další. Z tohoto důvodu se pracuje na vývoji další pravidlové vrstvy, která bude vycházet ze zkušeností s deduktivními databázemi a s věcnými pravidly v informačních systémech aktivovanými pomocí událostí. Poslední nevyšší vrstvou je vrstva, která má za úkol ověření věrohodnosti nalezených a odvozovaných informací [4].




**Obrázek 3.1 Schéma vrstev sémantického webu [5]**

## 3.3 RDF

Ve druhé polovině 90. let se rozvíjel směr, který usiloval o spojení ontologií a vznikajících webových standardů HTML, XML a později RDF. Za první ontologický jazyk lze pokládat SHOE nesoucí ontologii prostřednictvím speciálních HTML značek, poté vznikly jazyky na bázi XML, kterými jsou XOL a OML. Následoval vznik RDF, které bylo srovnáváno s XML v oblasti syntaktického rámce pro reprezentaci strojově zpracovatelných dat na WWW. Diskuze dospěla k závěru, že RDF je vhodnější z důvodu své modularity a formálně-logické interpretovatelnosti. Jazyk RDF se tedy stal základní úrovní reprezentace ontologií. Jedním z důvodů bylo syntaktické sjednocení webových informací a ontologií definujících jejich význam [4].

RDF je zkratka pro Resource Description Framework tedy framework pro popis zdrojů. RDF je skupinou W3C doporučen pro reprezentaci struktury webových metadat. Základním prvkem je RDF trojice. Tato trojice umožňuje definovat tvrzení ve tvaru *Subjekt-Predikát-Objekt*. Základním prvkem RDF je zdroj, který je identifikován svou URI. URI může být přiřazeno jakémukoli objektu, pojmu nebo například osobě. Zdroj může vystupovat jak v roli subjektu, tak i objektu. Predikát odpovídá sledované vlastnosti subjektu a objekt je hodnotou této vlastnosti. Objekt může být reprezentován jak zdrojem, tak také literálem, což je primitivní datová hodnota (například textová). Jednotlivá tvrzení jsou v RDF nezávislá na ostatních. Tvrzení lze propojovat pomocí URI zdrojů. Tvrzení jsou tedy navzájem nezávislá fakta, která je možno uložit přímo na webové stránky jako jejich „znalostní anotace“, nebo do specializovaných veřejných databází [3,5].

Datový model RDF nemá stanovenou konkrétní reprezentaci, kterou bude znázorňován. Možnými reprezentacemi jsou například grafická, reprezentace prostřednictvím slov a další. Základní syntaxí reprezentace doporučené skupinou W3C je reprezentace založená na XML formátu. Jedná se o takzvanou serializaci, v níž jsou jednotlivé prvky RDF řazeny specifickým způsobem do XML elementů a atributů [3].

Dalšími alternativními způsoby zápisu RDF jsou N3 notace a Turtle. N3 notace je lidmi jednoduše čitelný zkratkovitý popis RDF modelů, který není založený na XML serializaci. Turtle (Terse RDF Triple Language) je zjednodušená forma N3. Tato forma zápisu není standardizovaná [2].

Jak již bylo zmíněno, RDF lze vyjádřit i graficky pomocí grafu. Konkrétně se jedná o ohodnocený orientovaný multigraf, kde jsou subjekty a objekty reprezentovány uzly a predikáty orientovanými hranami. Dva uzly lze spojit jednou i více hranami [2,3].

Silným prostředkem v rámci RDF je tvrzení o tvrzení neboli reifikace. Je založeno na možnosti chápání celého tvrzení jako zdroj, o kterém lze vypovídat dalšími tvrzeními. Důvodem pro vznik tohoto prostředku v RDF byla zejména potřeba usuzovat o věrohodnosti jednotlivých tvrzení na základě důvěry v osobu, která tato tvrzení sestavila. Tímto se tato tvrzení o tvrzení dostávají na stejnou úroveň jako tvrzení základní bez potřeby vytváření zvláštního syntaktického aparátu [3].

RDF je jazykem ze strukturální stránky poměrně jednoduchý. Jeho formálně logické vlastnosti však takové již nejsou. Je to způsobeno především možností chápání celého tvrzení jako zdroj a přiřazováním mu hodnoty vlastností. RDF, který má vyjadřovací sílu nižší než predikátový kalkul první třídu, povoluje konstrukce libovolně vyšších řádů [5].

## 3.4 RDF Schema

RDF schema nebo také označované RDFS je první sémantický jazyk orientovaný na RDF. Byl vytvořen skupinou W3C v roce 1999. Jedná se o nástavbu nad RDF, která doplňuje do jeho struktury hlavní konstrukce z objektových (rámcových) systémů. Těmi jsou třídy a binární sloty, kterým je umožněno stanovit definiční obor a obor hodnot. Nad třídami a sloty je možné definovat hierarchii. Třídám z RDFS je umožněno přiřazovat zdroje z RDF jako jejich instance. Pro toto přiřazení slouží atribut *type*.

Pomocí RDFS je možno zachytit sémantiku obsahu stránek, čímž splňuje požadavky webových návrhářů. Vzhledem k tradičním ontologickým jazykům neumožňuje specifikovat podmínky příslušnosti ke třídám neboli lokální omezení a zcela postrádá datové typy. Plná verze RDFS obsahuje také možnosti reifikace [1].

## 3.5 OWL

OWL byl vytvořen pod hlavičkou W3C Ontology Working Group na základě zkušeností s využíváním ontologického jazyka DAIM+OIL, který vznikl z důvodu nedostatečných možností RDFS pro popis komplexnějších tříd zejména nemožností specifikace podmínek příslušnosti ke třídám a chybějící definici datových typů.

Zkratku OWL lze vyložit jako „Web Ontology Language“. Byl vytvořen pro získávání informací prostřednictvím aplikací. Na rozdíl od jazyků RDF, RDFS, XML má větší možnosti vyjádření sémantiky a významu obsahu. Návrh OWL byl koncipován směrem k použití jako webový ontologický jazyk. Z důvodu jeho doporučení konsorciem W3C se stal součástí sémantického webu. OWL používá *Případové* a *Požadavkové* dokumenty, které ontologiím poskytují více detailů, motivuje potřebu pro ontologickou síť jazyka při podmínkách šesti používaných případů a formuluje cíle designu, požadavky a cíle pro OWL [1,3]. Další informace v rámci jazyka OWL, pokud nebude v příslušné části uveden jinak, byly čerpány z [5].

OWL existuje ve třech formách. Jedná se o tři více-méně odlišné jazyky OWL lite, OWL DL a OWL Full, jejichž použití je určené specifickým společenstvím programátorů a uživatelů. U těchto forem je stupňována vyjadřovací síla, čímž je také stupňována náročnost na odvozovací prostředky. Informace týkající se popisu konkrétních verzí OWL byly čerpány z [3].

**OWL Lite**

OWL Lite má nižší formální složitost než OWL DL a OWL Full. Je jednodušší než ostatní verze, čímž je také jednodušší vytváření podporujících nástrojů tohoto jazyka. Zaměřuje se především na uživatele, kterým k vyjádření postačuje pouze hierarchie tříd a jednoduchá omezení. Mezi podporující omezení patří například omezení kardinality, jenž umožňuje vytvářet mohutnosti o velikosti 0 a 1.

**OWL DL**

V této verzi znamená zkratka DL description logic neboli logika popisu. V OWL DL jsou zahrnuty všechny konstrukce jazyka OWL, které je však možno použít pouze s danými omezeními. Verze se zaměřuje na uživatele, kteří žádají maximální výraznost při co nejnižší výpočetní náročnosti, kdy jsou

i přes to všechny závěry zcela vypočitatelné, a bezespornost, tedy všechny výpočty skončí v reálném čase.

**OWL Full**

Jedná se o verzi, jež má maximální výraznost a syntaktickou volnost stejně jako RDF, ale nezaručuje náročnost výpočtů. Je určena uživatelům, kteří tyto maximální vyjadřovací schopnosti vyžadují. Pro příklad uvedeme možnost třídy být zároveň kolekce individualit a současně individualita v jejich vlastní správě. Z důvodu rozsahu je velice nepravděpodobný vznik softwaru, který bude zahrnovat každý rys verze OWL Full.

OWL se skládá z hlavičky a vlastního obsahu. Hlavička může obsahovat například informace o verzi či odkazy na ontologie, které jsou v ní implicitně obsaženy. Obsah může být definován sadou axiomů, které definují třídy, individua, vlastnosti a vazby mezi nimi.

### 3.5.1 Struktura

**Třídy**

V jazyku OWL je možné pracovat s dvěma typy tříd, kterými jsou třídy pojmenované a třídy anonymní.

Pojmenovaná třída je identifikována svým názvem, který je URI (Uniform Resource Identifier). Jedná se o nepostradatelnou vlastnost pro konkrétní použití ontologií v otevřeném webovém prostředí.

Anonymní třída odpovídá logickému výrazu nad pojmenovanými nebo také anonymními třídami. Pro jakoukoli dvojici tříd lze vyslovit axiomy substituce, ekvivalence a disjunktnosti. V praxi se nejčastěji používá axiom substituce dvou pojmenovaných tříd, který odpovídá v běžném chápání speciálnější či obecnější třídě a axiom substituce pojmenované a anonymní třídy, která je vymezena prostřednictvím hodnoty určité vlastnosti. Anonymní třída je například třída objektů, které mají vlastnost „*být vlastněn*“ vůči alespoň jedné instanci třídy *osoba*. Pojmenovaná třída *být vlastněn* je potom ekvivalentní k třídě *být* a výše uvedené anonymní třídě.

**Individua**

Individuum nebo také instance či objekt je vždy identifikován prostřednictvím URI. Existence individua není závislá na žádné třídě. Může být přiřazováno jak k jedné, tak i k více třídám zároveň. Dvě individua je možno rozlišit pomocí axiomů. Individua se běžně do ontologií nezačleňují, využívají se hlavně v případech, kdy je pomocí nich třeba definovat určitá třída.

Například pokud chceme třídu *převod nemovitosti* vymezit jako množinu transakcí provedených podle jistého zákona sbírky, musíme tento zákon v ontologii definovat jako individuum, které je instancí třídy *zákon*.

**Vlastnosti**

Také vlastnosti jsou definovány prostřednictvím URI. Vlastnosti nejsou definovány v rámci určité třídy, ale jde o binární relaci nad množinou individuí. Axiomy vlastností mohou určovat jejich definiční obor a obor hodnot, dále také jejich obecné matematické charakteristiky jako je určení zda se jedná o tranzitivní, symetrické a či nebo funkční relace, vztah substituce, ekvivalence a či nebo inverzity dvojice vlastností.

**Dědičnost**

Omezení hodnot vlastností se dědí z obecnější třídy na speciálnější, jak již tomu napovídá logická podstata ontologií. Stejně tak se dědí matematické charakteristiky i obory argumentů obecnějších vlastností na vlastnosti specifičtější.

### 3.5.2 Syntaxe

Základní syntaxí OWL je XML. Nejedná se však o nativní vyjádření, jelikož je mezi stromovou strukturou XML a strukturou OWL, kterou jsme popsali výše, vložena úroveň jazyka RDF spolu s jeho vlastní sémantickou nástavbou RDF Schema.

RDF je syntaktickým prostředkem pro zápis OWL, ale z druhé strany OWL sémanticky rozšiřuje RDF. RDF schema umožňuje zavést nad zdroji a vlastnostmi hierarchickou strukturu tříd. Jedná se o ontologický jazyk, odlišuje se od OWL chudší vyjadřovací schopností a nedokonalou formální sémantikou.

Pro příklad je na obrázku 3.2 uvedeno, jak by mohla v syntaxi XML/RDF vypadat část ontologie definující *byt v osobním vlastnictví*.

```
<owl:Class rdf:ID="Byt_v_osobnim_vlastnictvi">
    <owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="\#Byt" />
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty
                rdf:resource="\#veVlastnictvi" />
                <owl:someValuesFrom rdf:resource="\Osoba" />
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:equivalentClass>
...
</owl:Class>
```

**Obrázek 3.2 Příklad owl [5]**

Ve výše uvedeném příkladě na obrázku 3.2 je definice této pojmové třídy založena na ekvivalenci vůči anonymní třídě, která je vymezena substitucí vůči dalším dvěma anonymním třídám. V této definici je odkazováno na tři jmenné podprostory, kterými jsou RDF u atributů ID a resource, RDFS v rámci substituce tříd a OWL. Tento vrstvený přístup je výhodný z důvodu, kdy jsou i aplikace, které neznají OWL ale pouze RDF či RDFS, schopny z ontologie získat určité informace. Ontologie v OWL jsou ze syntaktické stránky korektním komplexem tvrzení RDF. V XML syntaxi RDF není trojice *Subjekt-Predikát-Objekt* dobře viditelná. V případě OWL někdy ani nemá samostatně význam.

# 4 Návrh architektury distribuovaného informačního systému

## 4.1 Neformální specifikace

Byl zadán požadavek na vytvoření distribuovaného informačního systému, pomocí něhož bude možné jednoduše spravovat konference a semináře vysokoškolských výzkumných skupin a informace s nimi související. Skupiny chtějí tento systém také využít pro distribuování svých konferencí a seminářů ostatním skupinám. Stejně tak chtějí mít možnost získávat informace o konferencích a seminářích pořádaných skupinami jinými.

Systém musí umožnit spravovat jednotlivé konference, semináře a informace s nimi spojené, kterými jsou název, tematický obsah, na který bude událost zaměřena, skupinu, která tuto událost pořádá, datum a čas uskutečnění, datum a čas ukončení, místo konání, přednášející na konferenci či semináři a možnost zaznamenání dalších údajů, jako například informace o návaznosti na některou jinou již uskutečněnou či teprve v budoucnu realizovanou konferenci či seminář, v podobě poznámky. Uživatel bude mít také možnost nastavení vlastnosti konference či semináře určující schválení její distribuce i ostatním skupinám. Systém bude také obsahovat základní informace o příslušné vysokoškolské skupině.

Uživatel bude mít možnost spravovat tyto konference, semináře a informace s nimi spojené. Bude si tedy moci zobrazovat informace, přidávat, upravovat a mazat konference či seminář. Dále by bylo dobré, aby měl uživatel možnost v konferencích a seminářích vyhledávat.

Systém může být doplněn o další výpisy konajících se událostí seřazených podle několika kritérií: podle názvu události, podle data uskutečnění, podle tematického zaměření, podle pořádajících skupin. Tyto informace by měl systém zobrazit v grafické podobě, například v tabulce.

Systém by měl umožnit správu základních uživatelských rolí, to je uživatel, oprávněný uživatel a administrátor. Uživatel bude mít v systému právo prohlížet události, vypisovat o nich veškeré informace a měnit své osobní údaje. Oprávněný uživatel bude disponovat s právy na úkony spojené s administrací událostí, kterými jsou prohlížení událostí v systému, správa událostí v systému, zveřejnění zadaných událostí v systému pro další skupiny, zadání odběru konferencí a seminářů jiných skupin. Také bude mít oprávnění měnit svoje údaje a základní informace o svojí skupině. Administrátor bude mít stejná práva pro vykonávání úkonů v systému jako oprávněný uživatel, plus právo pro správu uživatelských rolí, obsahující přidání, úpravu a mazání uživatelských rolí v systému.

**Požadavky na architekturu**

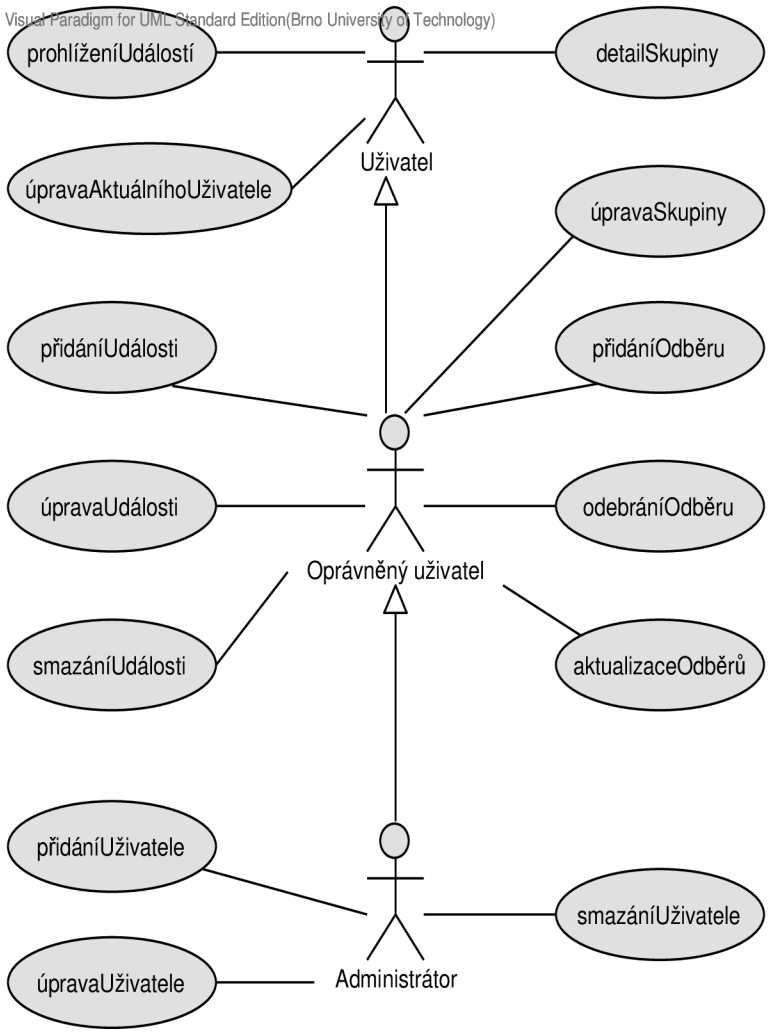
Komunikace systému s uživatelem bude realizována prostřednictvím webového rozhraní. Distribuce dat mezi jednotlivými systémy bude uskutečněna sémantickými technologiemi. Každý systém bude umožňovat vyvolání připojení na další vybrané systémy se sémantickou reprezentací dat, tato data následně zpracovávat a aktualizovat si jimi svoje temporální informace s událostmi získanými od jiných systémů.

## 4.2 Analýza

Z neformální specifikace vyplývají následující požadavky na systém. Byly vyvozeny případy užití zobrazené v diagramu případů užití na obrázku 4.1. Architektura systému a návrh uložení dat je rozebrán v následující kapitole *Návrh.*

Systém musí dokázat spravovat tři role uživatelů:

- **Uživatel:** Reprezentuje aktéra, který si může v systému procházet události, zobrazovat si o nich detailní informace a měnit svoje osobní údaje. Na jiné úkony v systému nemá práva.
- **Oprávněný uživatel:** Reprezentuje aktéra, který může spravovat události v systému. Jedná se o prohlížení, přidávání, úpravu a odstraňování událostí ze systému. V úpravě zadaných událostí nastavuje jejich zveřejnění v rámci jejich distribuce jiným skupinám (systémům). Spravuje také odběr událostí, jejich zadávání, úpravu a stahování. Také má oprávnění měnit své údaje a základní informace o svojí skupině.
- **Administrátor**: Disponuje se stejnými právy na úkony v systému jako oprávněný uživatel, rozšířenými o správu uživatelských rolí, která zahrnuje přidání, úpravu a mazání uživatelů v systému.



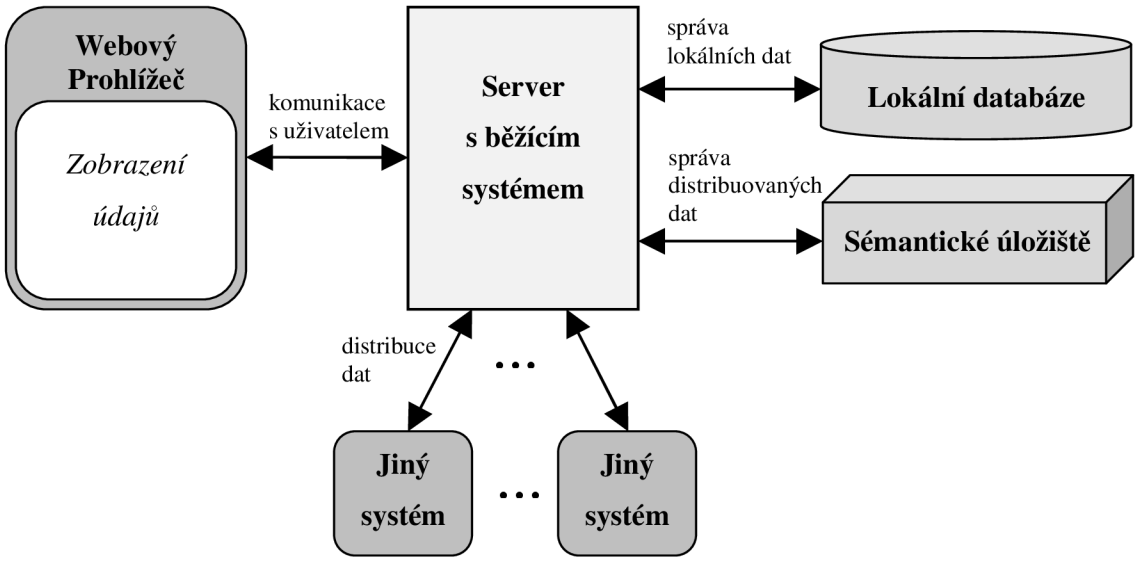

**Obrázek 4.1 Diagram případů užití**

## 4.3 Návrh

Tato část práce se zabývá návrhem architektury distribuovaného informačního systému vycházející z neformální specifikace. Obecné schéma tohoto systému je zobrazeno na obrázku 4.2. Základními rysy tohoto systému je jeho webové rozhraní a distribuce dat založená na sémantických technologiích.

Navrhovaný systém komunikuje s uživatelem prostřednictvím webových stránek, bude se tedy jednat o webový informační systém. Tento systém bude komunikovat na principu požadavek – odpověď. Pro komunikaci bude používán protokol http, který je bezstavový.

Jako architektura bude vycházet ze softwarové architektury MVC, rozdělující architekturu systému na tři části, kterými jsou datový model aplikace, uživatelské rozhraní a řídící logika. Tyto části jsou nezávislými komponentami, kde modifikace některé z nich má minimální vliv na ostatní.




**Obrázek 4.2 Obecná schéma systému**

Z důvodu distribuce dat založené na sémantických webech bude samotné uložení dat v systému rozděleno do dvou částí. První bude uložení dat týkajících se událostí zadaných přímo v systému a dat vztahujících se k administraci a funkčnosti systému, které budou uloženy v relační databázi ve struktuře popsané v následující podkapitole. V druhém případě se bude jednat o uložení dat týkajících se informací o událostech získaných prostřednictvím distribuce z jiných systémů. Uložení těchto dat bude realizováno pomocí sémantických technologií. K jeho podrobnému popisu se dostaneme taktéž v následující podkapitole.

Distribuce dat bude probíhat prostřednictvím sémantických technologií. Komunikace mezi systémy v rámci distribuce dat bude realizována pomocí protokolu http. Data mezi systémy budou přenášena v ontologickém formátu owl. Tato data budou ukládána do sémantického úložiště. Odkud budou v případě potřeby získávána. Podrobný popis distribučního mechanismu je v kapitole 4.3.4.

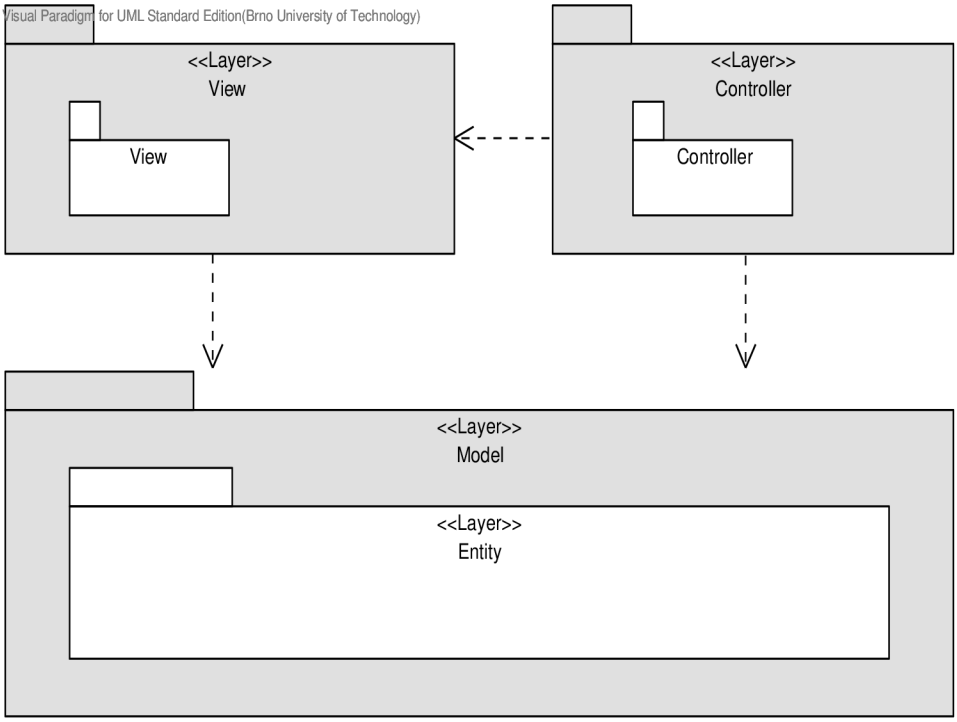
### 4.3.1 Návrh architektury aplikace

Na základě vybrané softwarové architektury byla navržena architektura realizovaného systému, která je znázorněna na obrázku 4.3. Systém je rozdělen do několika částí. Vrstva *View* převádí data do podoby, ve které budou prezentovány uživateli.

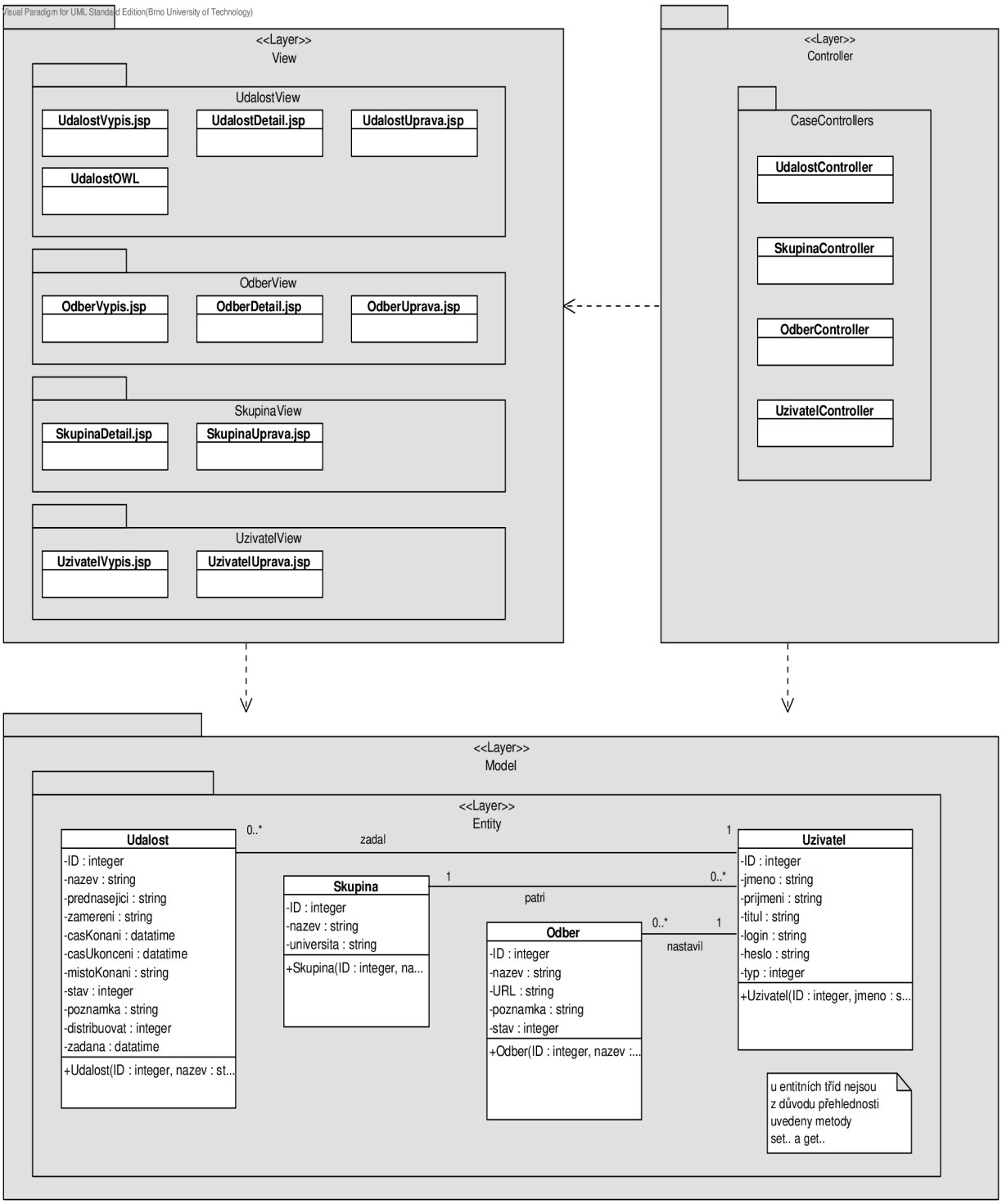
Vrstva *Controller* obsluhuje události ve většině případů vyvolané uživatelem. V rámci obsluhy těchto událostí používá i ostatní vrstvy *View* a *Model*. V této vrstvě budou také vykonávány úkony spojené se správou sémantického úložiště.

Vrstva *Model* obstarává manipulaci s daty vyvolanou prostřednictvím vrstvy *Controller*. Komponenta *Entity* představuje samotná data a *Mediator* jsou nástroje pro práci s těmito daty.

Každá z uvedených komponent architektury obsahuje jednotlivé třídy realizující uvedenou funkčnost. Toto podrobnější rozvržení základních tříd navrhovaného systému je znázorněno prostřednictvím diagramu návrhových tříd na obrázku 4.4.



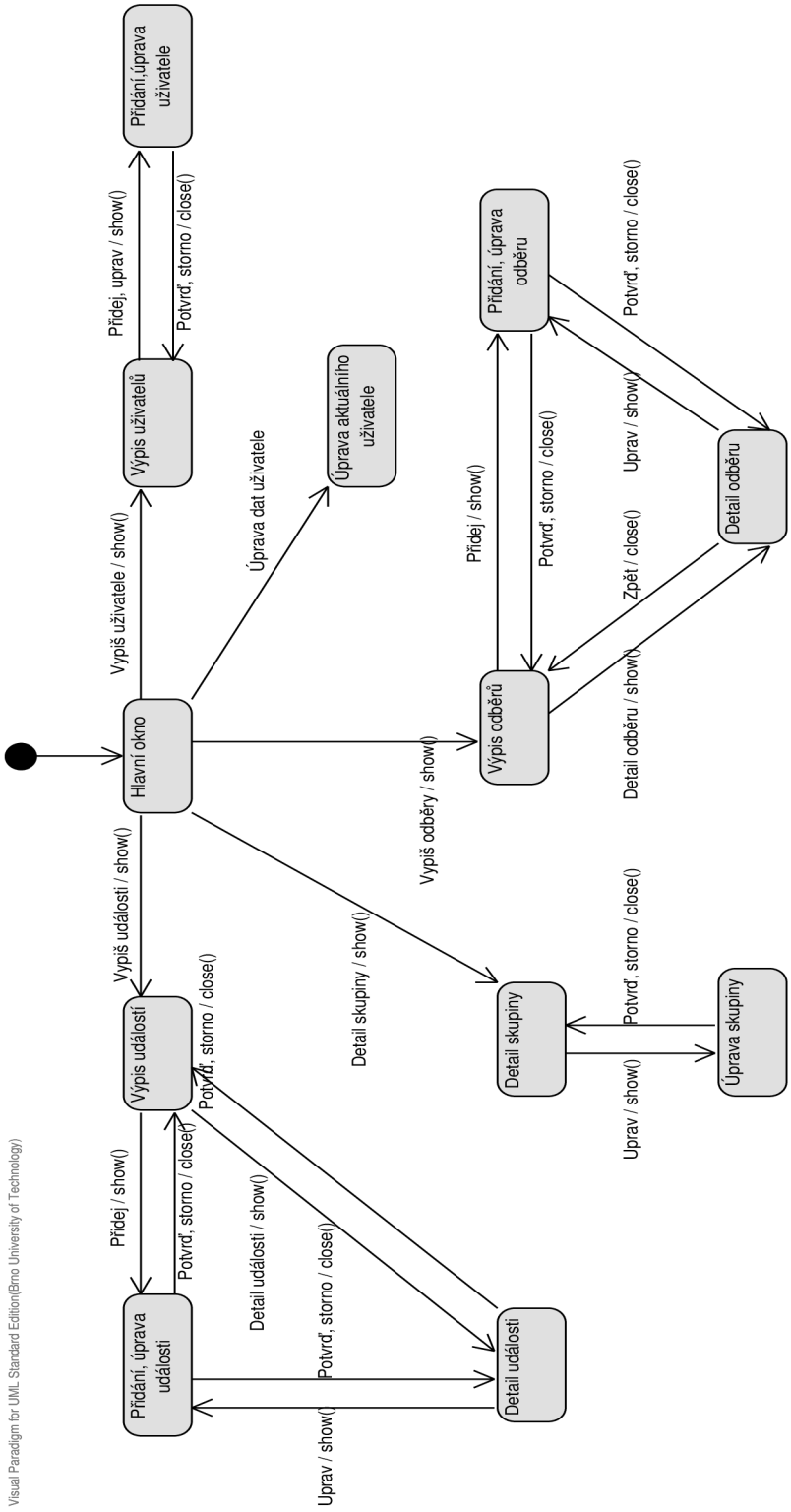

**Obrázek 4.3 Návrh architektury aplikace**




**Obrázek 4.4 Diagram návrhových tříd**

Bylo také třeba navrhnout rozložení obrazovek systému a jejich návaznost, kterou reprezentuje stavový diagram návaznosti obrazovek uvedený na obrázku 4.5. Základní výchozí okno systému, do kterého se uživatel dostane po přihlášení, je v diagramu reprezentováno oknem *Hlavní okno*. Z tohoto

okna se uživatel bude moci dostat do oken Výpis událostí, Výpis odběrů, Detail skupiny, Úprava aktuálního uživatele a Výpis uživatelů. V těchto oknech budou přístupné další možnosti práce s příslušnými daty.



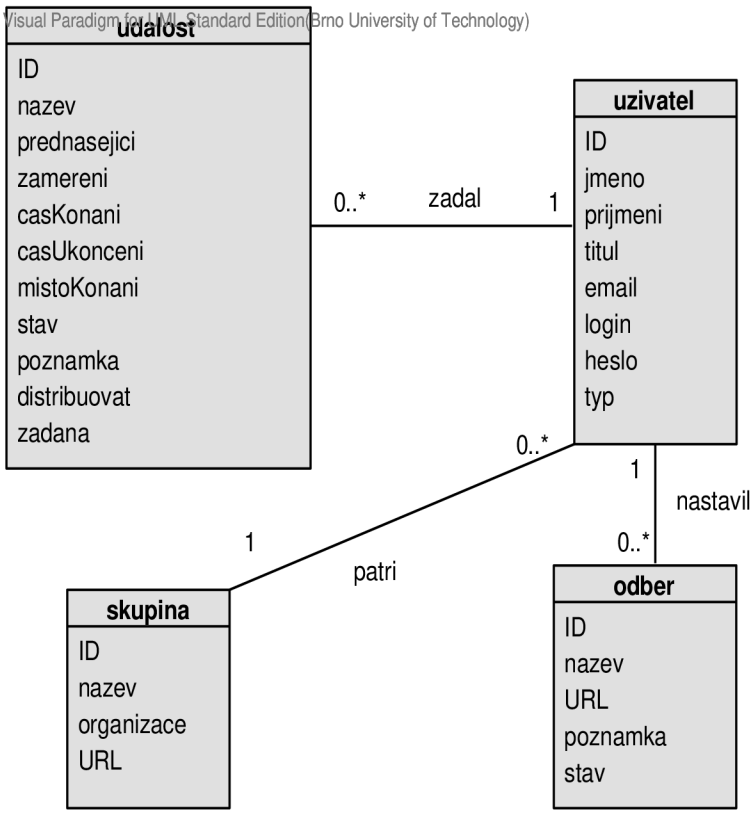

**Obrázek 4.5 Diagram návaznosti obrazovek**

### 4.3.2 Návrh databáze

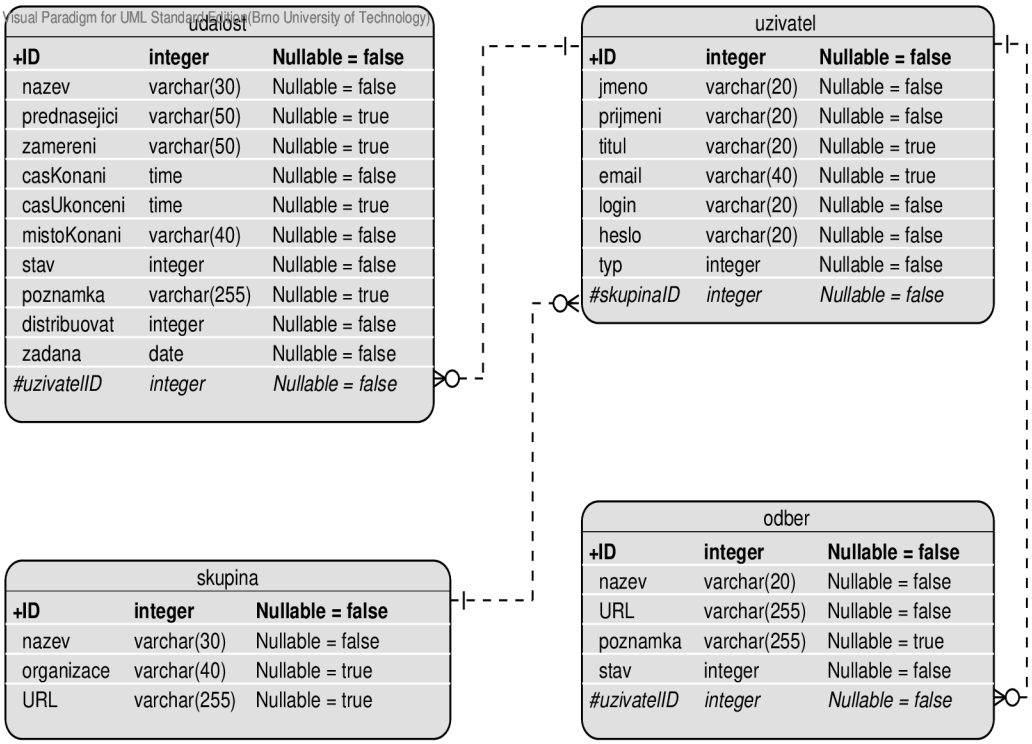
Nyní se budu zabývat samotným návrhem rozložení dat v databázi, která bude sloužit pro uložení dat událostí zadaných přímo v systému. Z neformální specifikace distribuovaného systému byly vyvozeny informace, které musí být ukládány v systému. Dalšími informacemi, které budou taktéž ukládány v systému, jsou data vyvozená ze specifikace funkčnosti uvedené v neformální specifikaci a data potřebná pro správné implementační fungování systému. Shrnutím všech těchto informací byly navrženy čtyři tabulky pro ukládání dat. Jsou to tyto:

- ***Událost*** – ukládá data související s konferencemi a semináři uloženými v systému.
- ***Skupina*** – tabulka uložení informací o skupině v rámci níž jsou data do systému zadávána.
- ***Odběry*** – tabulka uchování informací o zadaných odběrech ze systémů jiných skupin, které mají být realizovány.
- ***Uživatel*** – je to tabulka nesoucí informace o uživatelech systému.

Následuje uvedení diagramů, v nichž je znázorněna statická struktura systému. Na obrázku 4.6 je to konceptuální diagram tříd zobrazující statickou strukturu systému prostřednictvím tříd a vztahů mezi nimi. Na dalším obrázku 4.7 je znázorněn návrh schématu databáze prostřednictvím ER diagramu.




**Obrázek 4.6 Konceptuální diagram tříd**




**Obrázek 4.7 Návrh schématu databáze**

### 4.3.3 Uložení dat získaných prostřednictvím distribuce

Distribuce dat v navrhovaném informačním systému bude realizována prostřednictvím sémantických technologií. Pro uložení dat v sémantickém formátu bude, z důvodu ulehčení ukládání, dotazování, přístupu k nim a především lehké změny obsažených informací a jejich formátu, výhodné použít úložišť k tomu určených. Pro reprezentaci distribuovaných dat bude použit formát RDF, z čehož vyplývá, že budou použity RDF úložiště.

#### 4.3.3.1 RDF úložiště

Existují dva hlavní způsoby uložení dat v rámci RDF úložišť. Jedná se o nativní a databázový způsob uložení. Pro přístup k těmto datům se používají speciální jazyky. V této kapitole byly informace čerpány ze z [6].

**Databázová úložiště**

V případě databázových úložišť se využívá ukládání dat do relační databáze, která je založena na ukládání dat podle fixního schématu představující popis ukládaných dat metadaty. RDF a následně i RDFS data mají velmi dynamické schéma, což je jednou z jejich výhod, ale zároveň to znamená nemožnost přímého mapování na schéma relační databáze. Databázová úložiště lze podle typu mapování rozdělit do tří skupin, kterými jsou nezohledňující schéma, zohledňující schéma a kombinovaná.

**Nativní úložiště**

Nativní úložiště vznikly z důvodů problémů při ukládání sémantických formátů dat do relačních databází. Jedná se o vlastní pro sémantická data vyhovující způsob uložení. Obvykle se jedná o uložení do souboru podle vlastního formátu nebo i do operační paměti.

**Dotazovací jazyky**

K datům, která jsou uložena v RDF úložištích, je třeba nějakým způsobem přistupovat. Z důvodu efektivity úložiště je třeba odstínit způsob uložení od obsahu, což vede k nevhodnosti použití SQL pro dotazování se na RDF data. Z toho důvodu vznikly pro dotazování na RDF data jazyky nové. Jedná se o jazyky inspirované buď jazyky relačních databází tedy SQL, nebo v menší míře založené na pravidlech podobně jako u Prologu. Do prvního zmíněného typu patří například jazyky RQL, SeRQL, RDGL, do druhého například TRIPLE, N3. Tyto jazyky však nejsou standardizovány, a proto se ve větší míře nerozšířily a nespolupracují s mnoha nástroji.

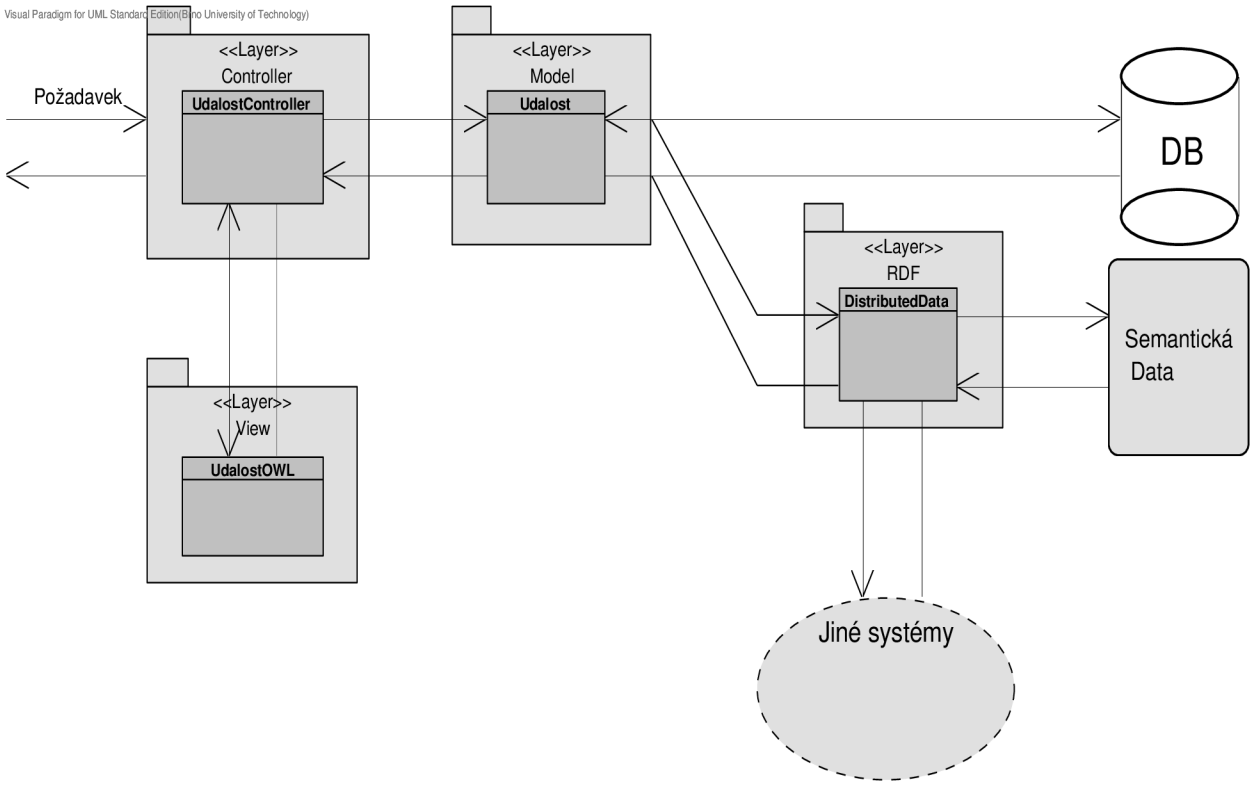
Proto vznikl s cílem standardizovat dotazování RDF dat jazyk SPARQL. Skládá se ze dvou částí, jimiž jsou dotazovací jazyk a protokol. Dotazovací jazyk umožňuje klást dotazy na obsah úložiště a protokol definuje formu dotazu a odpovědi.

#### 4.3.3.2 Sesame 2

Pro uložení sémantických dat získaných prostřednictvím distribuce mezi informačními systémy bude využito úložiště Sesame 2. Toto úložiště bylo vybráno z důvodu jeho možností a vyhovujících vlastností. V potaz byla také vzata jeho rozsáhlá podpora a jeho kvalitní dokumentace. Podrobnější seznámení s tímto úložištěm je obsahem podkapitoly 6.1.4.

### 4.3.4 Mechanismus distribuce dat

Distribuce dat prostřednictvím sémantických webů bude probíhat níže popsaným způsobem. Diagram zpracování distribuovaných dat je zobrazen na obrázku 4.8.




**Obrázek 4.8 Diagram zpracování distribuovaných dat**

**Distribuce dat**

Zde je pro příklad popsán jeden konkrétní případ distribuce. *Controller* dostane přes http požadavek o data, prostřednictvím komponenty *UdalostController* vyvolá komponentu v modelu *Udalost*, která získá příslušná data z databáze. Komponentou v modelu získaná data jsou předána komponentě v *Controlleru*, který předá data příslušné komponentě ve vrstvě *View UdalostOWL*. Ta změní formát těchto dat do formátu owl. Poté pošle data zpět příslušné komponentě *Controlleru*, který data odešle jako odpověď na požadavek od jiného systému.

**Získávání dat**

Vyvolání mechanismu získávání dat může být realizováno automaticky po nějaké době. Nebo také při žádosti o výpis dat, kdy se zkontrolují poslední časy aktualizací odběrů zadaných v systému a pokud od těchto časů uplynula určitá doba, je zavolána aktualizace dat pro příslušné odběry. Další možností je vyvolání prostřednictvím stisku tlačítka *Stáhnout data* v okně odběrů. Následně bude popsán mechanismus distribuce dat na konkrétním příkladě.

*Controller* obdrží zprávu požadující aktualizaci dat, tu zpracuje příslušná komponenta *UdalostController*. Ta dále komunikuje s modelem komponentou *Udalost*, která předá zprávu balíčku *DistributedData* ve vrstvě *RDF*. V balíčku *DistributedData* jsou komponenty provádějící aktualizaci ze zadaných odběrů. Zasílají se požadavky o data na příslušné adresy. Obratem jsou data přijata systémem. Data od jiných systémů jsou ukládána do sémantického úložiště. Z těchto úložišť je možno tato data lehce získávat prostřednictví sémantických dotazů. Po uložení dat do souborů je o dokončení informována komponenta *UdalostController*, která tuto informaci zašle komponentě ve vrstvě *Model* a ta tuto informaci vydistribuje prostřednictvím příslušných technik ve vrstvě *View* až k uživateli, čímž je uživatel uvědoměn, že proces aktualizace databáze byl dokončen.

# 5 Návrh ontologie pořádaných událostí

V této kapitole se budu zabývat návrhem ontologie sloužící pro distribuci údajů o konferencích a seminářích vysokoškolských výzkumných skupin ve vytvářeném distribuovaném informačním systému. Obvyklým formátem pro přenos informací mezi distribuovanými informačními systémy je formát XML, či jiné podobné formáty. Přenášená data v těchto formátech však neobsahují žádnou sémantiku. Tu musí datům dodat až programátor podle příslušné dokumentace. Výhodou využití některého z ontologických formátů v rámci distribuce je jednoznačné určení sémantiky dat. Je tedy možné tyto distribucí poskytovaná data jednoduše zpracovávat a využívat i jinými aplikacemi než mnou vytvářeným distribuovaným informačním systémem.

## 5.1 Existující ontologie

V rámci návrhu nových ontologií je z důvodu jednoduchosti a jednotnosti vhodné využít již existující ontologie. Existující ontologie, které mají specifikovanou sémantiku a jejich využitím se tato sémantika přenáší i na nově vytvářenou ontologii. Přehled některých existujících ontologií se nachází na http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library#OWL_ontologie. Dále popíši existující ontologie použité v navrhnuté ontologii.

### 5.1.1 FOAF

Ontologie *Friend of a Friend* neboli FOAF vznikla z důvodu zjednodušení sdílení informací o lidech a jejich aktivitách. Dále pro zjednodušení přenosu těchto informací mezi webovými stránkami a zjednodušení jejich využívání a zpracování v online prostředí internetu [7].

Samotný základ ontologie položili v polovině roku 2000 Dan Brickley a Libby Miller pod jménem "*experimental linked information project*". V nynější době se jedná o stabilní jádro obsahující množství tříd a vlastností. Její jmenný prostor již nebude pozměňován, jedinými očekávanými změnami jsou změny v rámci rozšíření ontologie o nové vztahy [7].

Z jejího poměrně širokého obsahu bych pro příklad zmínil třídy *Agent*, *Person*, *Project*, *Organzation* či *Group*, sloužící pro uchování informací, které vyjadřují jejich samotná pojmenování. Zajímavými vlastnostmi v rámci tohoto projektu jsou *name*, *suriname*, *title*, *mbox* a několik dalších, které budou zmíněny v následující části textu v rámci popisu navržené ontologie [7].

Jednoduchý příklad zápisu informací *name*, *homepage*, *oppid* a *img* o osobě v jazyce OWL s prostřednictvím ontologie FOAF je uveden na obrázku 5.1. Celková specifikace ontologie je uvedena na adrese http://xmlns.com/foaf/spec/.

```
<foaf:Person rdf:about="#danbri"
             xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:name>Dan Brickley</foaf:name>
  <foaf:homepage rdf:resource="http://danbri.org/" />
  <foaf:openid rdf:resource="http://danbri.org/" />
  <foaf:img rdf:resource="/images/me.jpg" />
</foaf:Person>
```

**Obrázek 5.1 Příklad využití FOAF pro popis osoby [7]**

### 5.1.2 DBpedia

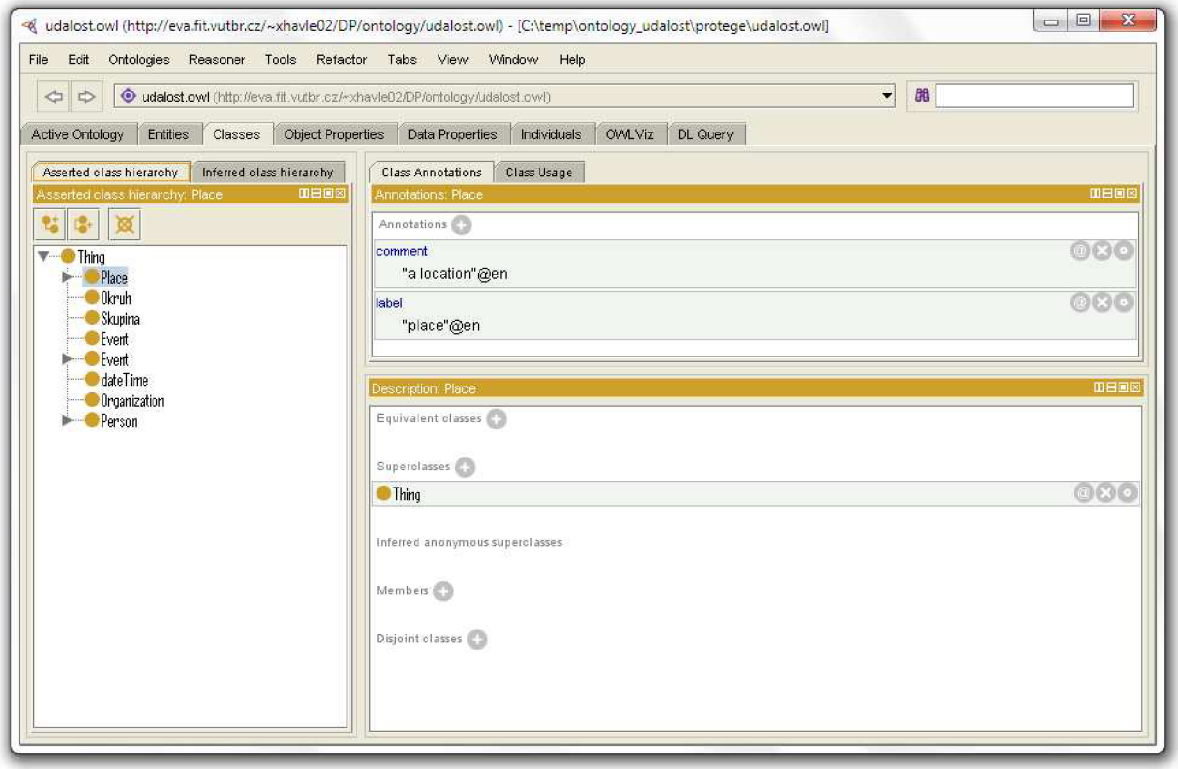
DBpedia je otevřený komunitní projekt. DBpedia se snaží získávat co největší množství informací z Wikipedie. Tento projekt obsahuje vlastní ontologii pod názvem DBpedia ontology. Jedná se o mezioborovou ontologii vytvořenou na základě nejčastěji používaných dotazů ve Wikipedii. Ontologie obsahuje v současnosti 259 tříd popsaných 1200 odlišných vlastností. Pro reprezentaci informací používá RDF. Nyní se v ní nachází asi 1 480 000 instancí. Pro získávání dat z této ontologie je možné použít pro to určené rozhraní s dotazovacím jazykem SPARQL [8].

Znalostní základna DBpedie má oproti stávajícím znalostním zdrojům několik výhod. Zahrnuje znalosti z mnoha oblastí. Obsahuje reálné z více stran potvrzené vědomosti. Automaticky se rozvíjí prostřednictvím změn Wikipedie. Je multijazyčná. Ve srovnání s Wikipedií, dovoluje DBpedia pokládat překvapivé dotazy. Jako například: „Vrať mi všechna města v České republice, která mají více než 10 000 obyvatel.“ [8]

V rámci návrhu ontologie pro přenos dat mezi informačními systémy byla z ontologie DBpedia použita třída ***Place*** popisující místo**.**

## 5.2 Protégé

Pro prvotní seznámení s návrhem ontologií jsem použil nástroj Protégé. Jedná se o open-source editor ontologií a rámcové báze znalostí poskytující sadu nástrojů pro vytváření ontologií. Protégé platforma podporuje dva způsoby modelování. Prvním z nich je modelování pomocí Protégé-frames, druhým je Protége-OWL editor. Ontologie v editoru je možné exportovat do různých formátů zahrnující RDF, RDFS, OWL a XML Schema [9].




**Obrázek 5.2 Nástroj Protégé**

Protégé funguje na různých platformách. Aplikace je založena na jazyce Java. Je možné použití pluginů. Protégé je podporováno silnou komunitou vývojářů a akademickými, vládními a firemními uživateli. Je používám v rozmanitém spektru oblastí. Příkladem je biomedicína, shromažďování zpravodajských informací či podnikové modelování. Ukázka uživatelského rozhraní editoru Protété je na obrázku 5.2 [9].

# 5.3 Návrh

Mým úkolem bylo navrhnout ontologii pro reprezentaci konferencí a seminářů vysokoškolských výzkumných skupin. Informační obsah byl odvozen, stejně jako schéma databázových tabulek, z neformální specifikace systému uvedené v kapitole 4.1 a je tedy téměř totožný s informačním obsahem databázových tabulek *udalost*, *uzivatel* a *skupina,* uvedených ve schématu databáze na obrázku 4.7.

Hlavními informačními prvky ontologie jsou samotná konference a semináře s příslušnými informacemi, uživatel zadávající tuto událost a skupina, která ji organizuje. Tímto rozdělením by bylo, v případě většího rozsahu, pro jednodušší použití možné ontologii rozvrhnout do tří samostatných ontologií. Ale jak dále uvidíme, navrhovanou ontologii není třeba dělit.

Pro navrhnutou ontologii jsem zvolil název *„udalost“,* což zastřešuje konference i semináře. Příponu této ontologie jsem odvodil podle používaného ontologického jazyka *„owl“*. Celý název navrhované ontologie je tedy *„udalost.owl“*. V rámci této kapitoly je popsán návrh ontologie pomocí RDF tvrzení z jazyka OWL a jím odpovídajících RDF grafů.

## 5.3.1 Konference a semináře

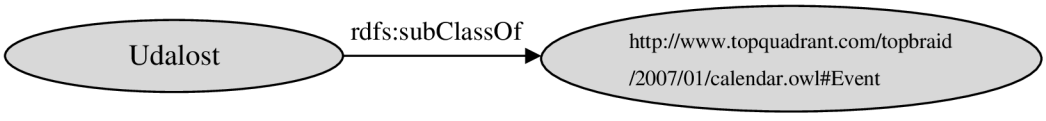
V rámci konferencí a seminářů jsou třeba v ontologii předávat informace o názvu, přednášejícím, zaměření, času konání, času ukončení, místu konání, stavu, poznámku k události a události o uživateli, který konferenci či seminář zadal.

**Jméno, čas konání, čas ukončení**

Pro konference a semináře byla vytvořena nová třída pojmenovaná *Udalost.* Jako předek této třídy byla využita třída ***Event*** z ontologie *calendar.owl*. Tato ontologie je dostupná na URL http://www.topquadrant.com/topbraid/2007/01/calendar.owl.

Třída ***Event*** obsahuje datové vlastnosti *name*, *startTime* a *endTime*. Zděděním třídy událost z třídy ***Event*** bude obsahovat tyto datové vlastnosti i třída ***Udalost***. Na vlastnost *startTime* bude mapována informace o čase konání, *endTime* čase ukončení a na vlastnost *name* informace názvu události. RDF graf zdědění třídy ***Udalost*** je na obrázku 5.3. Owl zápis definice třídy ***Udalost*** je následující:

```
<owl:Class rdf:ID="Udalost">
  <rdfs:subClassOf rdf:resource="http://www.topquadrant.com/
                    topbraid/2007/01/calendar.owl#Event"/>
</rdfs:Class>
```

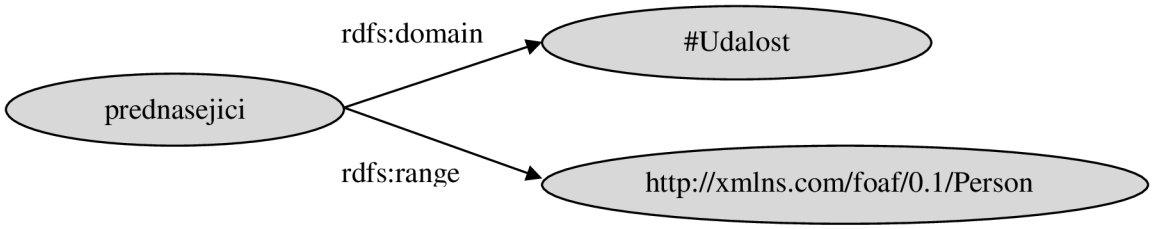



**Obrázek 5.3 RDF graf zdědění třídy *Udalost***

**Přednášející**

Pro přenos informace o přednášejícím je využita výše zmíněná ontologie FOAF, konkrétně v ní obsažená třída ***Person***. Pro vyjádření vztahu mezi třídou ***Udalost*** a ***Person*** jsem definoval objektovou vlastnost třídy ***Udalost*** *prednasejici*, jejíž RDR graf definice je na obrázku 5.4. Owl zápis je následující:

```
<owl:ObjectProperty rdf:ID="prednasejici">
  <rdfs:domain rdf:resource="#Udalost"/>
  <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
</owl:ObjectProperty>
```




**Obrázek 5.4 RDF graf definice vlastnosti *prednasejici***

Díky využití této třídy ***Person*** má přednášející v rámci ontologie definovanou sémantiku osoby. Pro uložení informace o jméně přednášejícího je použita datová vlastnost *name*.

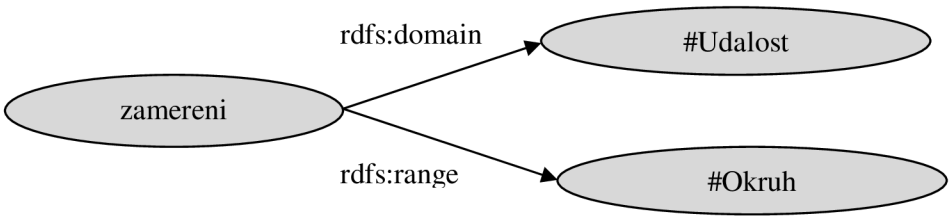
**Zaměření**

Pro přenos informace o zaměření konference či semináře jsem vytvořil samostatnou třídu ***Okruh***, která po zdědění z třídy ***Thing*** obsahuje datovou vlastnost ***name***. Tuto datovou vlastnost používám pro uložení jména zaměření události. Protože je dědění od třídy ***Thing*** explicitní, není ho třeba do definice nové třídy zapisovat. Owl zápis definice třídy ***Okruh*** vypadá následovně:

```
<owl:Class rdf:ID="okruh"/>
```

Pro vyjádření vztahu mezi třídou ***Udalost*** a ***Okruh*** jsem definoval objektovou vlastnost třídy ***Udalost*** *zamereni*. Její definici odpovídající RDF graf je na obrázku 5.5. Owl zápis definice je následující:

```
<owl:ObjectProperty rdf:ID="zamereni">
  <rdfs:domain rdf:resource="#Udalost"/>
  <rdfs:range rdf:resource="#Okruh"/>
</owl:ObjectProperty>
```

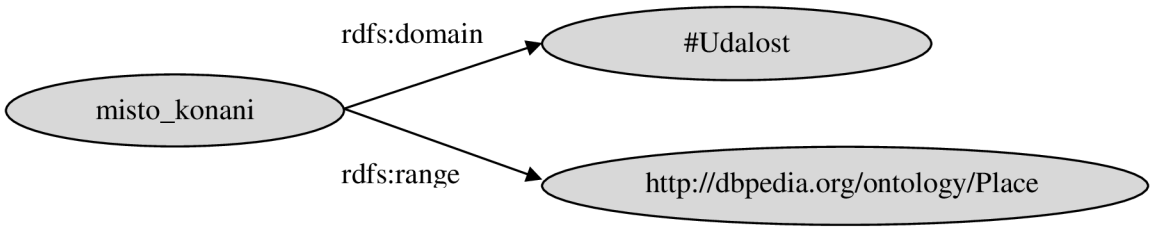



**Obrázek 5.5 RDF graf definice vlastnosti *zamereni***

**Místo konání**

Pro uchování informace o místě konání jsem použil třídu ***Place*** z ontologie *DBpedia,* zmíněné výše. Pro zapsání místa konání je využita datová vlastnost třídy ***Place*** *name*. Vztah mezi třídami ***Place*** a ***Udalost*** je definován objektovou vlastností třídy události *misto_konani.* Její definici odpovídající RDF graf je na obrázku 5.6. Owl zápis definice je následující:

```
<owl:ObjectProperty rdf:ID="misto_konani">
  <rdfs:domain rdf:resource="#Udalost"/>
  <rdfs:range
   rdf:resource="http://dbpedia.org/ontology/Place"/>
</owl:ObjectProperty>
```

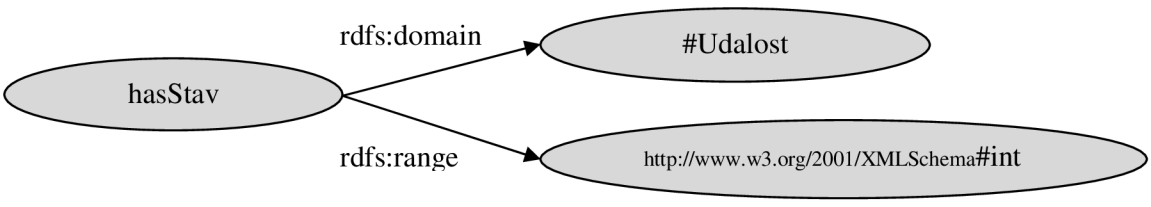



**Obrázek 5.6 RDF graf definice vlastnosti *misto_konani***

**Datové vlastnosti**

Pro zbývající informace v rámci třídy ***Udalost*** jsem použil datové vlastnosti. Tyto vlastnosti jsou uchovány prostřednictvím literálů. Datová vlastnost *hasStav* slouží pro uložení numerické hodnoty reprezentující stav události. Tato hodnota nabývá 1 v případě, že je událost aktuální a 0 v případě, že byla událost zrušena, tedy aktuální již aktuální není. Definici vlastnosti odpovídající RDF graf je na obrázku 5.7. Její owl zápis je následující:

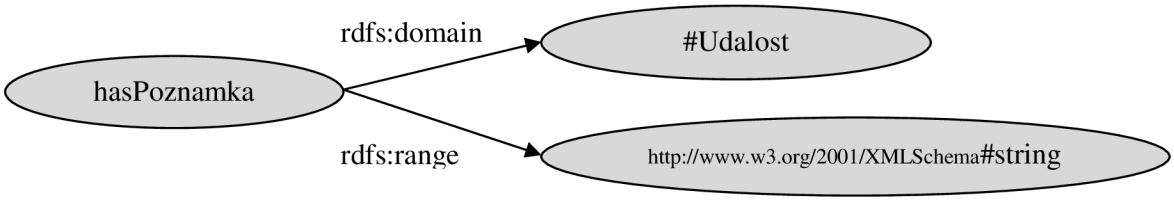
```
<owl:DatatypeProperty rdf:ID="hasStav">
  <rdfs:domain rdf:resource="#Udalost"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>
```




**Obrázek 5.7 RDF graf definice vlastnosti *hasStav***

Další datovou vlastností třídy ***Udalost*** je *hasPoznamka,* uchovávající informaci zadané poznámce ke konkrétní události, která nabývá hodnoty z definičního oboru *string*. Definici vlastnosti odpovídající RDF graf je na obrázku 5.8. Její owl zápis je následující:

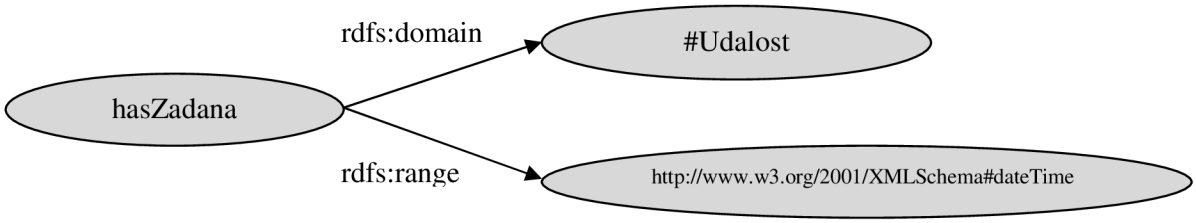
```
<owl:DatatypeProperty rdf:ID="hasPoznamka">
  <rdfs:domain rdf:resource="#Udalost"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```




**Obrázek 5.8 RDF graf definice vlastnosti *hasPoznamka***

Pro uchování informace o datu zadání, či poslední změně informací týkajících se události, slouží datová vlastnost *hasZadana*, která nabývá hodnot z definičního oboru *dateTime*. Její definici odpovídající RDF graf je na obrázku 5.9. Owl zápis definice je následující:

```
<owl:DatatypeProperty rdf:ID="hasZadana">
  <rdfs:domain rdf:resource="#Udalost"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
```

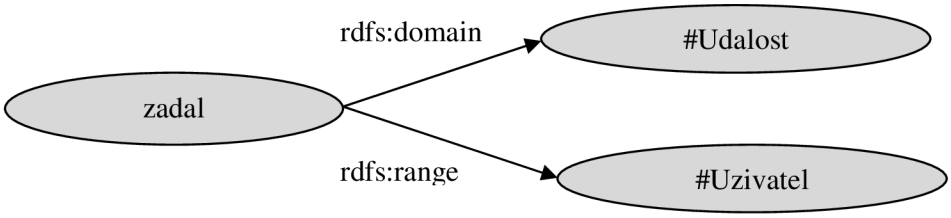



**Obrázek 5.9 RDF graf definice vlastnosti *hasZadana***

### Zadal

Poslední potřebnou přenášenou informací v rámci události je návaznost na osobu, která tuto událost zadala, či poslední editovala. Pro zanesení této informace jsem vytvořil třídě ***Udalost*** objektovou vlastnost *zadal*, jejíž definiční obor je *individum* třídy ***Uzivatel*** popsané v další kapitole. Definici vlastnosti odpovídající RDF graf je na obrázku 5.10. Její owl zápis je následující:

```
<owl:ObjectProperty rdf:ID="zadal">
  <rdfs:domain rdf:resource="#Udalost"/>
  <rdfs:range rdf:resource="#Uzivatel"/>
</owl:ObjectProperty>
```



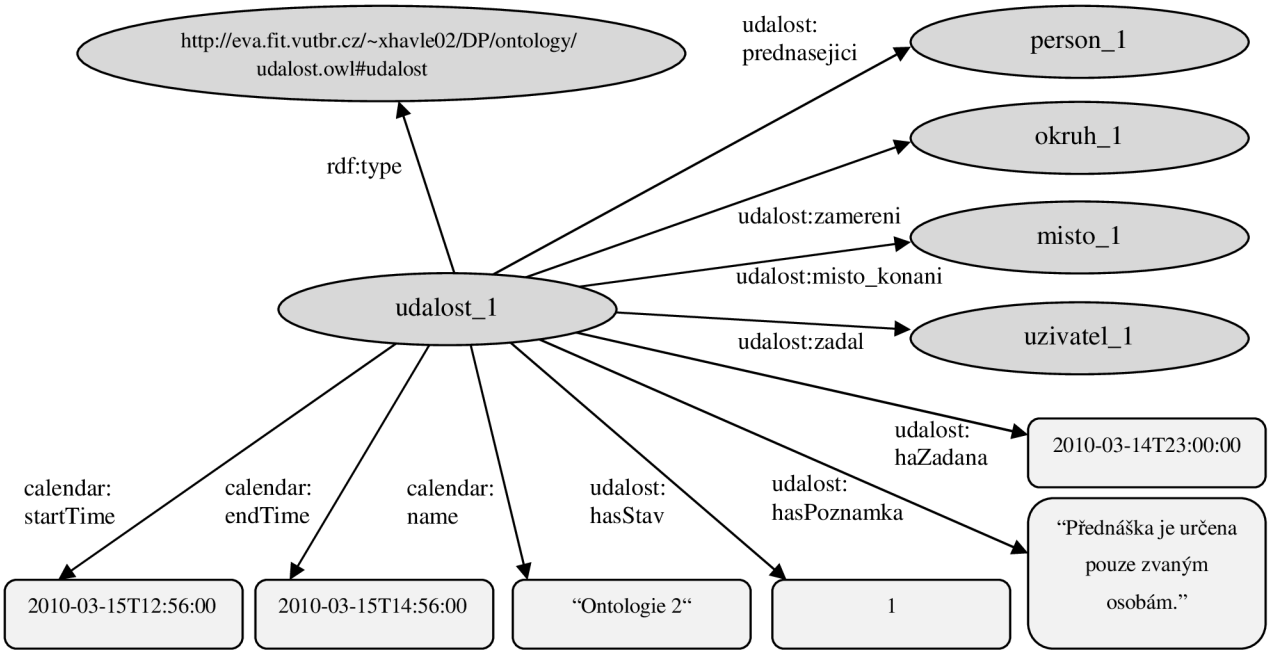

**Obrázek 5.10 RDF graf definice vlastnosti *zadal***

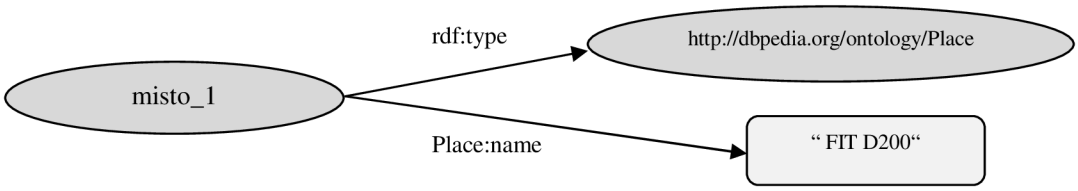
**Příklad instance třídy *Udalost***

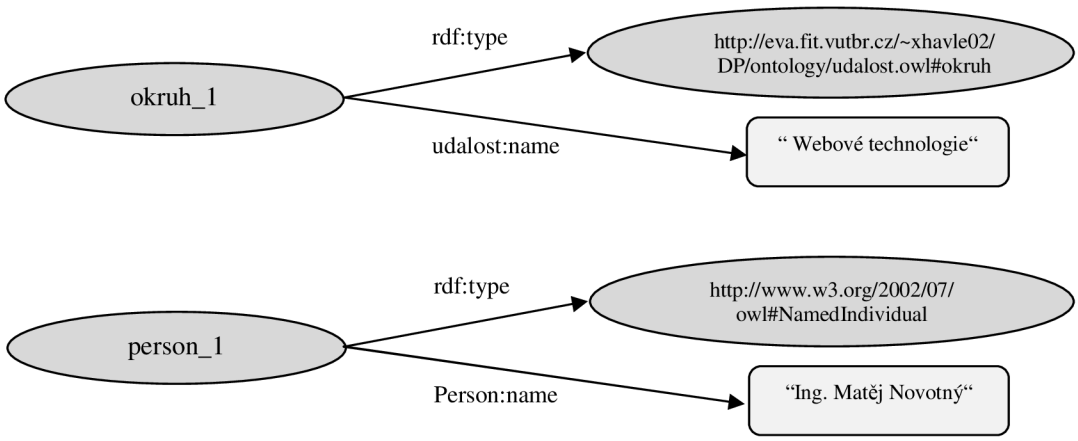
Následuje příklad vytvořeným distribuovaným informačním systémem vygenerované instance třídy *Udalost* společně se souvisejícími instancemi. RDF graf odpovídající tomuto příkladu je na obrázku 5.11.

```
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/"
     xmlns:udalost="http://eva.fit.vutbr.cz/~xhavle02/
                    DP/ontology/udalost.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:Person="http://xmlns.com/foaf/0.1/Person#"
     xmlns:calendar="http://www.topquadrant.com/topbraid/
                     2007/01/calendar.owl#">

<owl:NamedIndividual
 rdf:about="http://localhost:8084/VSSKUPINY/OWLData?sk=1#udalost_1">
   <rdf:type rdf:resource="http://eva.fit.vutbr.cz/~xhavle02/DP/
                           ontology/udalost.owl#udalost"/>
   <udalost:hasStav rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
      1</udalost:hasStav>
   <udalost:hasZadana
    rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
      2010-03-14T23:00:00</udalost:hasZadana>
   <calendar:startTime
    rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
      2010-03-15T12:56:00</calendar:startTime>
   <calendar:endTime
    rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
      2010-03-15T14:56:00</calendar:endTime>
   <calendar:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Ontologie 2</udalost:name>
   <udalost:hasPoznamka
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Přednáška je určena pouze zvaným osobám.
   </udalost:hasPoznamka>
   <udalost:misto_konani
    rdf:resource="http://localhost:8084/VSSKUPINY/OWLData?sk=1#misto_1"/>
   <udalost:zamereni
    rdf:resource="http://localhost:8084/VSSKUPINY/OWLData?sk=1#okruh_1"/>
   <udalost:prednasejici
    rdf:resource="http://localhost:8084/VSSKUPINY/OWLData?sk=1#person_1"/>
   <udalost:zadal
    rdf:resource="http://localhost:8084/VSSKUPINY/OWLData?sk=1#uzivatel_1"/>
</owl:NamedIndividual>
```

```
<owl:NamedIndividual
 rdf:about="http://localhost:8084/VSSKUPINY/OWLData?sk=1#misto_1">
   <rdf:type rdf:resource="http://dbpedia.org/ontology/Place"/>
   <Place:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      FIT D200</Place:name>
</owl:NamedIndividual>

<owl:NamedIndividual
 rdf:about="http://localhost:8084/VSSKUPINY/OWLData?sk=1#okruh_1">
   <rdf:type rdf:resource="http://eva.fit.vutbr.cz/~xhavle02/
                           DP/ontology/udalost.owl#okruh"/>
   <udalost:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Webové technologie</udalost:name>
</owl:NamedIndividual>

<foaf:Person
 rdf:about="http://localhost:8084/VSSKUPINY/OWLData?sk=1#person_1">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
   <Person:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Ing. Matěj Novotný</Person:name>
</foaf:Person>
```





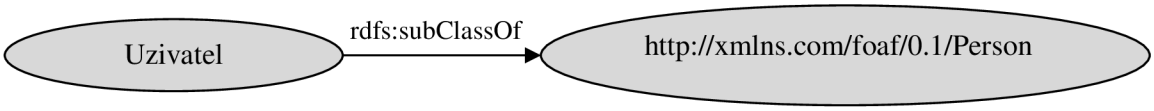

**Obrázek 5.11 RDF graf instance třídy *Udalost***

### 5.3.2 Uživatel

V rámci uživatele jsou třeba v ontologii předávat informace o jméně, příjmení, titulu, emailové adrese a příslušnosti ke skupině.

Pro třídu ***Uzivatel*** bylo výhodné ji zdědit z třídy ***Person*** z ontologie FOAF zmíněné výše. Tato třída obsahuje datovou vlastnost *name*, která je použita pro uchování informace jména. Dále datovou vlastnost *lastName,* optimální pro uchování informace o příjmení, datovou vlastnost *title* pro reprezentaci informace o titulu a datovou vlastnost *mbox* k uchování informace o emailové adrese uživatele. RDF graf zdědění třídy ***Uzivatel*** je na obrázku 5.12. Owl zápis definice třídy ***Udalost*** je následující:

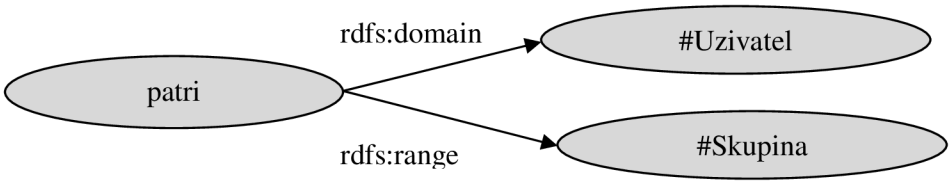
```
<rdfs:Class rdf:ID="Uzivatel">
  <rdfs:subClassOf
   rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
</rdfs:Class>
```




**Obrázek 5.12 RDF graf zdědění třídy *Uzivatel***

Dalším potřebným a uchovávaným údajem v rámci třídy uživatele je vlastnost udávající jeho příslušnost ke skupině. Pro uchování tohoto údaje jsem vytvořil třídě objektovou vlastnost *patri*. Její definici odpovídající RDF graf je na obrázku 5.13. Owl zápis definice je následující:

```
<owl:ObjectProperty rdf:ID="patri">
  <rdfs:domain rdf:resource="#Uzivatel"/>
  <rdfs:range rdf:resource="#Skupina"/>
</owl:ObjectProperty>
```

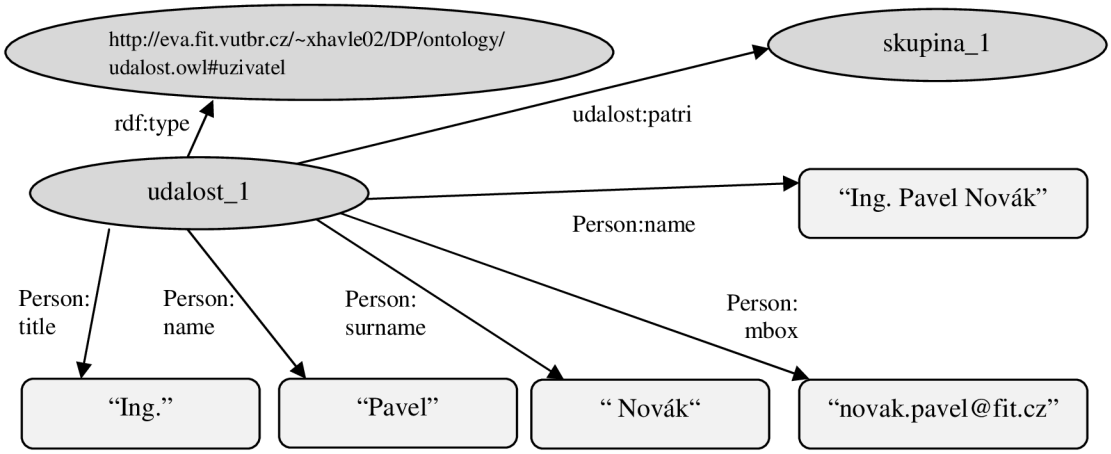



**Obrázek 5.13 RDF graf definice vlastnosti patri**

**Příklad instance třídy *Uzivatel***

Následuje příklad vytvořeným distribuovaným informačním systémem vygenerované instance třídy *Uzivatel*. RDF graf odpovídající tomuto příkladu je na obrázku 5.14.

```
<rdf:RDF xmlns:udalost="http://eva.fit.vutbr.cz/~xhavle02/
                        DP/ontology/udalost.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:Person="http://xmlns.com/foaf/0.1/Person#">

<owl:NamedIndividual
 rdf:about="http://localhost:8084/VSSKUPINY/OWLData?sk=1#uzivatel_1">
   <rdf:type rdf:resource="http://eva.fit.vutbr.cz/~xhavle02/
                           DP/ontology/udalost.owl#uzivatel"/>
   <Person:title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Ing.</Person:title>
   <Person:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Ing. Pavel Novák</Person:name>
   <Person:firstName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Pavel</Person:firstName>
   <Person:mbox rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      novak.pavel@fit.cz</Person:mbox>
   <Person:surname rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Novák</Person:surname>
   <udalost:patri rdf:resource="http://localhost:8084/VSSKUPINY/
                                OWLData?sk=1#skupina_1"/>
</owl:NamedIndividual>
```




**Obrázek 5.14 RDF graf instance třídy *Uzivatel***

### 5.3.3 Skupina

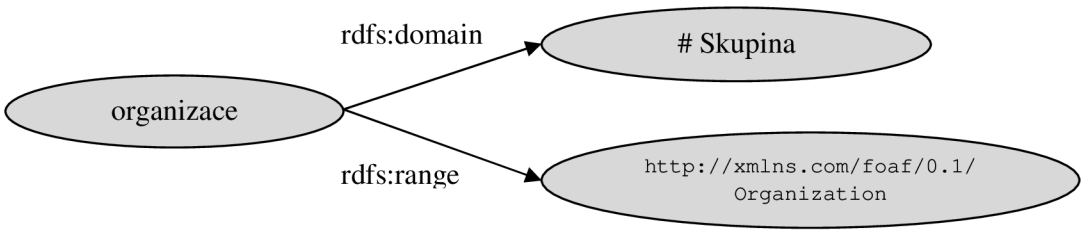
V rámci skupiny je třeba, aby ontologie obsahovala informace o jejím názvu, názvu organizace v rámci níž je skupina provozována a internetovou adresu této organizace.

Pro uchování údajů skupiny v rámci ontologie ***udalost.owl*** jsem vytvořil třídu ***Skupina***. Owl zápis její definice je následující:

```
<owl:Class rdf:ID="Skupina"/>
```

Tato třída je také implicitně zděděna od třídy ***Thing***, z čehož vyplývá obsah datové vlastnosti *name*, která je využita pro uchování informací o jméně skupiny. Pro reprezentaci informací o organizaci a případné internetové adrese organizace, byla použita owl třída ***Organization*** z ontologie FOAF zmíněné již dříve. Pro vyjádření vztahu mezi třídami ***Skupina*** a ***Organization*** jsem prostřednictvím následujícího zápisu přiřadil třídě skupina objektovou vlastnost *organizace*. RDF graf odpovídající tomuto zápisu je na obrázku 5.15.

```
<owl:ObjectProperty rdf:ID="organizace">
  <rdfs:domain rdf:resource="#Skupina"/>
  <rdfs:range rdf:resource="http://xmlns.com/foaf/
                            0.1/Organization"/>
</owl:ObjectProperty>
```




**Obrázek 5.15 RDF graf definice vlastnosti organizace**

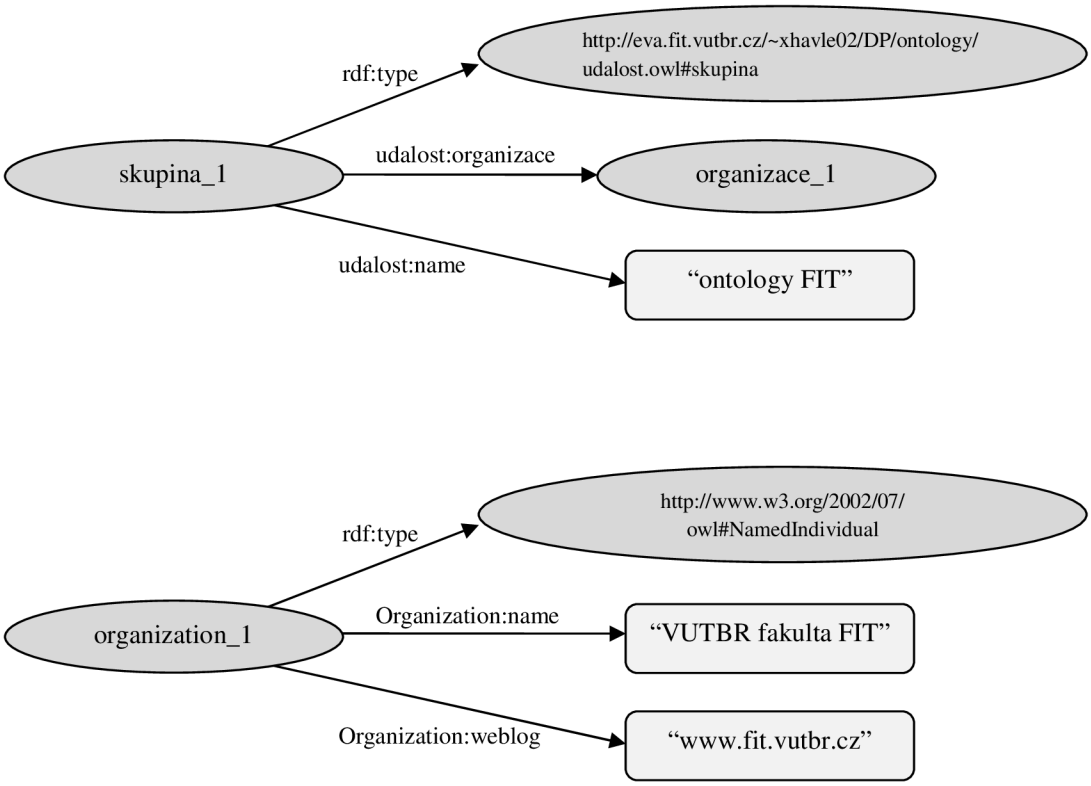
Třída ***Organization*** obsahuje datovou vlastnost *name*, která je v rámci ontologie ***udalost.owl*** použita pro uchování jména organizace a datovou vlastnost *weblog*, která je použita pro uchování URL organizace. Instanci třídy ***Skupina*** jsou přiřazováni uživatelé prostřednictvím objektové vlastnosti *patri* třídy ***Uzivatel***.

#### Příklad instance třídy Skupina

Následuje příklad vytvořeným distribuovaným informačním systémem vygenerované instance třídy *Skupina*. RDF graf odpovídající tomuto příkladu je na obrázku 5.16.

```
<rdf:RDF xmlns:udalost="http://eva.fit.vutbr.cz/~xhavle02/
                        DP/ontology/udalost.owl#"
     xmlns:Organization="http://xmlns.com/foaf/0.1/Organization#"
     xmlns:ontology="http://dbpedia.org/ontology/"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
```

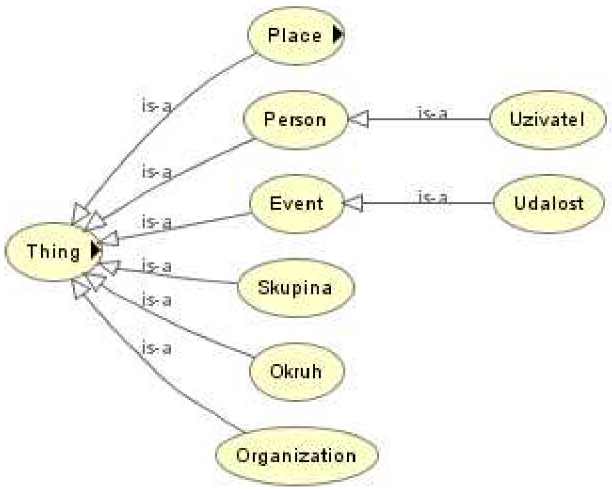
```
<owl:NamedIndividual
 rdf:about="http://localhost:8084/VSSKUPINY/OWLData?sk=1#skupina_1">
   <rdf:type rdf:resource="http://eva.fit.vutbr.cz/~xhavle02/
                           DP/ontology/udalost.owl#skupina"/>
   <udalost:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      ontology FIT</udalost:name>
   <udalost:organizace  rdf:resource="http://localhost:8084/VSSKUPINY/
                                      OWLData?sk=1#organizace_1"/>
</owl:NamedIndividual>

<foaf:Organization
 rdf:about="http://localhost:8084/VSSKUPINY/OWLData?sk=1#organizace_1">
   <rdf:type
    rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
   <Organization:name
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      VUTBR fakulta FIT</Organization:name>
   <Organization:weblog
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      www.fit.vutbr.cz</Organization:weblog>
</foaf:Organization>
```




**Obrázek 5.16 RDF graf instance třídy *Skupina***

## 5.4 Zhodnocení návrhu

Na základě konzultace s vedoucím práce jsem vytvořil výše popsanou ontologii pro účast osob na různých konferencích a seminářích s názvem ***udalost.owl***. Tato ontologie je optimalizována prostřednictvím využití již existujících ontologií. Zahrnuje veškerá potřebná data. Schema hierarchie výsledných tříd, získané prostřednictvím nástroje Protégé, je zobrazeno na obrázku 5.17. Celkový zápis vytvořené ontologie se nachází v příloze 1.




**Obrázek 5.17 Hierarchie tříd ontologie**

# 6 Implementace

V této části se dostáváme k popisu implementace distribuovaného informačního systému, jehož distribuce dat je postavena na sémantických technologiích. Implementace tohoto informačního systému navazuje na jeho návrh popsaný v kapitole 4.

Kapitolu implementace jsem rozčlenil do dvou podkapitol. V první z nich jsem popsal při implementaci použité technologie. V tomto celku se nejprve zabývám základem aplikace, což je její databázová část. Dále popíši základní model aplikace, jímž je MVC. Následovat bude popis aplikačního frameworku JavaServer Faces, který jsem v rámci vytváření distribuovaného informačního systému použil. Posléze popíši používané úložiště pro uchování sémantických dat. V poslední části této podkapitoly zmíním OWL API, které jsem použil pro generování distribučních owl zpráv.

Druhou částí kapitoly implementace je podkapitola popisující implementaci vytvořeného distribuovaného informačního systému. Budou zde rozebrány jednotlivé implementační celky a třídy, z kterých se skládají, popsána jejich funkčnost, vzájemná komunikace a návaznost.

## 6.1 Implementační prostředí

Tato kapitola slouží pro seznámení s vybraným implementačním prostředím, které bylo použito pro vytvoření systému. Pro implementaci byl vybrán programovací jazyk Java. Hlavními důvody jeho výběru byla jeho rozšířenost a tím i podpora realizování navrženého systému. S ohledem na tuto volbu byly vybrány i další, při implementaci použité, programovací technologie a přístupy.

### 6.1.1 Databáze

Základním prvkem každého systému je databáze spravující jeho data. Pro účely tohoto projektu jsem vybral databázový systém MySQL. Informace uvedené v této podkapitole byly čerpány z [10].

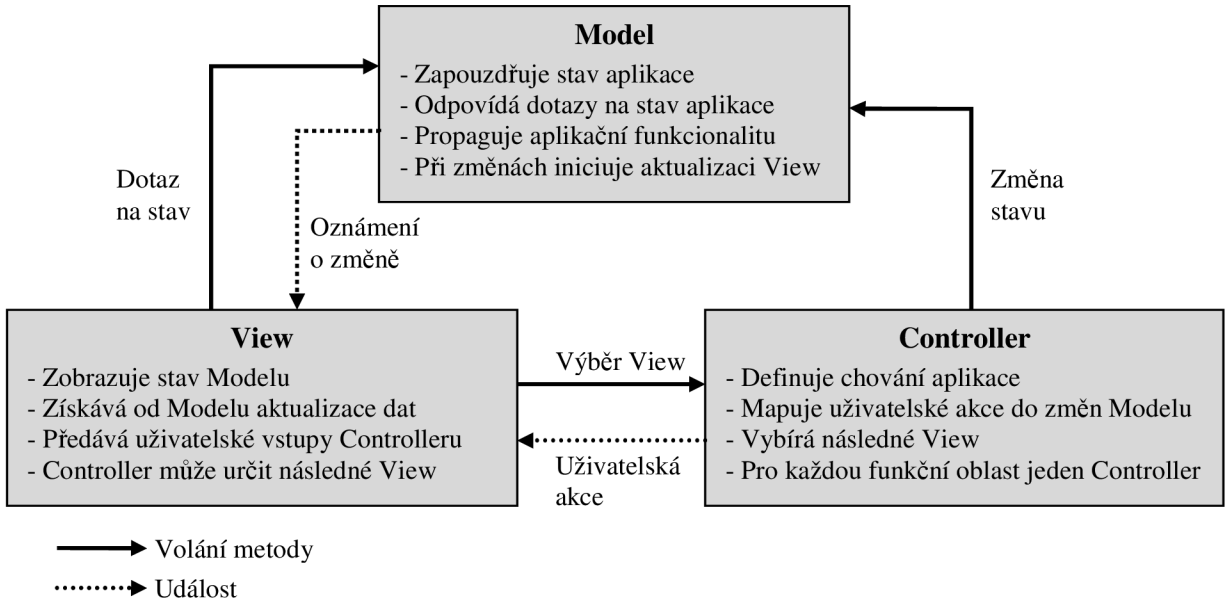
MySQL je databázový systém dostupný jak pod bezplatnou, tak i placenou komerční licencí. Díky své bezplatné licenci, jednoduchému použití a výkonu zastává tento databázový systém v současné době na trhu velký podíl použití. MySQL vytvořila švédská firma MySQL AB. V současné době tuto firmu vlastní společnost Sun Microsystems dceřiná společnost Oracle Corporation. Jedná se o multiplatformní databázi. Pro komunikaci včetně správy dat s databází slouží jazyk SQL.

### 6.1.2 Model view controller

Tato podkapitola se zabývá softwarovou architekturou Model-view-controller, dále zkracováno na MVC, použitou v rámci návrhu a implementace distribuovaného informačního systému. Tuto architekturu jsem použil z důvodu jejích vlastností, které jsou popsány v rámci této kapitoly. Informace uvedené v této podkapitole byly čerpány z [11].

Architektura má své kořeny ve Smalltaku. Zde byla prvotně použita pro mapování tradičního vstupu, zpracování a výstupu do grafického uživatelského rozhraní. Je to architektura použitelná v případě, kdy chceme rozdělit aplikaci na tři nezávislé části. Jedná se o oddělení datového modelu aplikace, uživatelského rozhraní a řídící logiky.

Výhodami tohoto rozdělení je umožnění výměny či modifikace kterékoliv části této třívrstvé architektury s minimálním vlivem na ostatní. Toto rozdělení také umožňuje použít vícenásobné zobrazení pro stejný datový model, což podporuje snadnější implementaci a správu vícenásobného klientského zobrazení. Schéma vztahů a funkcí částí MVC je uvedena na obrázku 6.1.




**Obrázek 6.1 Schéma vztahů a funkcí částí MVC [11]**

Jak již bylo zmíněno výše, softwarová architektura MVC rozděluje aplikaci pro jednoduchost jejich výměny či modifikace na tři části, které jsou v rámci architektury nazvány Model (datový model aplikace), View (uživatelské rozhraní) a Controller (řídící logika). Následuje podrobnější specifikace těchto částí.

**Model**

Model v rámci této softwarové architektury představuje datový a funkční základ aplikace. Jeho rolí v rámci aplikace je správa dat a stavů systému. Tyto úkony zahrnují, jak poskytnutí přístupu k těmto datům a stavům, tak jejich ukládání a aktualizaci. Zastřešuje tedy například správu dat uložených v databázi. Architektura však nespecifikuje konkrétní datové úložiště používané aplikací. O modelu se také hovoří jako o abstrakci procesů probíhajících v reálném světě. V rámci celé softwarové architektury je model zapouzdřeným celkem implementujícím rozhraní pro komunikaci s částí View a částí Controller.

**View**

View, neboli pohled, slouží pro prezentování datového obsahu modelu. Pohled specifikuje, jakým způsobem mají být prezentována data aplikace klientskému systému či uživatelům aplikaci využívající. Dalším úkolem je obsluha reakce na změny datového obsahu v části modelu a jejich promítání do prezentace klientovi. Tohoto může být dosaženo například použitím *puch modelu*, ve kterém se View registruje k datovému modelu, pro zasílání upozornění v případě změny dat. Dalším možným přístupem je *pull model*, v rámci něhož je za aktuálnost zobrazených dat zodpovědný View, který při potřebě aktuálních dat pro prezentaci volá Model pro získání nejnovějších dat. View může sloužit pro přijímání událostí vyvolaných uživatelem. V tomto případě zastupuje View funkci

prostředníka, který tyto události zprostředkovává Controlleru. Stejně jako Model i View implementuje rozhraní pro komunikaci s ostatními částmi Modelem a Controllerem.

**Controller**

Controller realizuje řídící logiku aplikace, tedy její chování při obsluze klientem vyvolaných událostech. Při přijetí klientem aplikace vygenerované události obslouží Controller tuto událost příslušným způsobem. Většinou se jedná o promítnutí události odpovídajících změn, prostřednictvím Modelu do dat či stavu aplikace.

Klientem generované události mohou být vygenerovány prostřednictvím uživatelského rozhraní, například zadáním dat do aplikace. Controller také v rámci příslušné události realizuje vyvolání změny v uživatelském rozhraní v části View. Dalším možným způsobem vyvolání události, na kterou posléze Controller reaguje příslušnou akcí, jsou HTTP požadavky obdržené vrstvou View prostřednictvím *Get* či *Post* metod.

### 6.1.3 JavaServer Faces

Jak jsem již zmínil dříve, při návrhu distribuovaného informačního systému, jsem použil architekturu Model-view-controller. Z této architektury vychází diagram návrhových tříd. Stejně tak se od architektury odvíjel výběr implementačního frameworku, který jsem zvolil framework JavaServer faces. Informace uvedené v této podkapitole byly čerpány z [12].

JavaServer Faces (JSF) je framework pro vytváření serverových uživatelských rozhraní pro webové aplikace. Jak jeho název napovídá, je tento framework založen na jazyce Java. Důvodem pro jeho vznik bylo ulehčení vývoje webových aplikací. Pro vývoj webových aplikací je sice možné použít základní Java Web technologie jako *Java servlets* nebo *JavaServer Pages (JSP)* bez aplikace dalších technologií, ale z důvodu možnosti výskytu problémů vývoje a správy není využití pouze základních technologií výhodné. JavaServer Faces umožňuje se těmto problémům vyhnout.

JavaServer Faces byl vytvořen skupinou *Java Community Process (JCP),* obsahující technologické vedení Sun Microsystems, Oracle, Borland, BEA a IBM, společně se skupinou znalců Javy a webovými experty. První specifikační žádost JavaServer Faces ustanovil v polovině roku 2001 Amy Fowler. V roce 2002 byli pověřeni vedením specifikace Ed Burns a Craig McClanahan. Formální uvedení JavaServer Faces bylo v roce 2004.

JavaServer Faces byl vytvořen pro zjednodušení vývoje a uživatelského rozhraní pro Java Web aplikace v následujícím směru:

- Nabízí komponentně zaměřené, klientsky nezávislé vývojové prostředí, určené k vytváření webových uživatelských rozhraní, zlepšující produktivitu vývojářů a snadnost použití.
- Zjednodušuje přistupování a správu aplikačních dat prostřednictvím webového uživatelského rozhraní.
- Automaticky jednoduchým způsobem spravuje stav uživatelského rozhraní mezi mnohonásobnými dotazy a klienty.
- Zastupuje vývojářský framework, který je přátelský k vývojářům s různorodými dovednostmi.

JSF kombinuje MVC návrhový přístup se silným komponentně zaměřeným uživatelským vývojovým frameworkem, který zjednodušuje J2EE webový vývoj, zatímco využívá existující JSP a servlet technologie.

**UI komponenty**

JSF specifikuje definici balíčků komponent pro uživatelské rozhraní, které mohou být použity standardním způsobem, nebo pro dosažení specifického chování. Je důležité se seznámit s pojmem

„*UI component*“, který bude používán v dalším textu. *UI component* je ve skutečnosti kombinací tří nezávislých elementů, které jsou tvořeny JSF komponentami ve stránce. Konkrétně to jsou tyto:

- Skutečná třída ***UIComponent***, která definuje chování komponent, jako například *UISelectOne,* což dovoluje použít „vybrání jedné z několika“.
- Volitelná třída ***Renderer***, která zajišťuje speciální specifické vykreslování komponent. Například *UISelectOne* komponenta může být vykreslována v HTML, buď jako skupina radio tlačítek, nebo jako selekt menu.
- ***JSP tag***, který spojuje ***Renderer*** s ***UIComponent*** a realizuje jejich použití v JSP jako jeden tag a zbavuje UI komponenty jejich závislosti na HTML, čímž umožní jejich vykreslování v jakémkoliv značkovacím jazyce. Příkladem může být tag *<h:selectOneMenu>*.

Nyní se seznámíme s klíčovým konceptem UI komponent. Standardní UI komponenty jsou specifikovány v rámci JSP tag knihoven „*Core*“ a „*HTML*“. *Core* knihovna tag komponent definuje všeobecné operace webových aplikací jako validaci a konverzi vstupních hodnot a načítání zdrojových balíčků. *HTML* knihovna tag komponent vytváří a vykresluje HTML komponenty. Obsahuje komponenty pro zobrazování jednoduchých textových popisů, vstupních polí, odkazů, tlačítek a dalších včetně složitějších komponent.

JSF je založen na *JavaBean* událostí řízením modelu. Jeho prostřednictvím jsou mapovány komponenty uživatelského rozhraní na *JavaBeany* nacházející se na serverové straně, které jsou registrované v XML souboru *faces-config.xml*. Mapování je zapisováno jednoduchým jazykem. Obsluha mapovaní mezi rozhraním a serverem je prováděna prostřednictvím *JSF regest processing lifecycle*, který v rámci této funkčnosti provádí i další potřebné úkony. Tento mechanismus bude podrobněji popsán dále.

**JSF aplikační architektura**

Nejelegantnějším návrhovým aspektem specifikace JavaServer Faces je jeho kompletní založení na J2EE webových technologiích. Což znamená, že JSF aplikace jsou standardní J2EE aplikace s několika specifickými konfiguracemi.

První specifickou konfigurací je záznam ve webovém aplikačním souboru *web.xml*, který v případě, že je URL specifikováno jako */faces/**, spouští Faces Controller servlet.

Další specifickou JSF konfigurací je soubor *faces-config.xml*, který obsahuje konfiguraci pro všechny prvky JSF aplikace. S tímto souborem je zacházeno stejně jako se souborem *web.xml*. Jeho umístění je většinou ve webové aplikaci ve složce *WEB-INF/*. Přesná struktura tohoto souboru a jeho prvků je detailněji popsána níže.

Dále je třeba, aby byly ve složce *WEB-INF/* umístěny následující knihovny:

- Aktuální JSF knihovny: jsf-api.jar a jsf-impl.jar.
- Dodatečné Apache knihovny: commons-beanutils.jar, commons-collections.jar, commons-digester.jar a commons-logging.jar. Protože nejsou tyto části součásti JSF technologie, počítá JSF s jejich umístěním do adresáře WEB-INF/lib.
- JSTL jar soubory: jstl.jar a standard.jar.

Po správném nastavení J2EE webové aplikace pro JSP, je možné vytvářet zobrazení prostřednictvím JSF. Je však možné použít i prvky JSP. JSF aplikace s použitím JSP se vytvářejí pomocí speciálních pro JSF určených JSP značkovacích knihoven. V případě použití JSP v rámci JSF aplikace je třeba nadefinovat předpis použití příslušných prvků stránky. Následující předpisy jsou pro *Core* a *HTML* knihovny ze Sunovské referenční implementace.

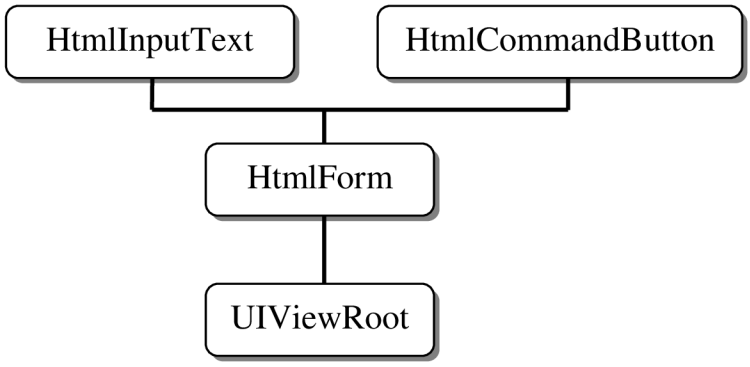
```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

Do těla JSP je třeba přidat <f:view> tag. Tento tag je základní UI komponentou náležející do stromu v paměti serveru, v případě kdy je stránka zobrazována. Pokud bude stránka sloužit pro vstup, je třeba do tagu <f:view> vložit tag <f:form>. Do tagu <f:form> je dále možné vkládat další tagy jako například <f:inputText>, jehož prostřednictvím se na stránce vykreslí vstupní pole, či <f:commandButton>, který v rámci stránky představuje tlačítko.

Vytváření a správa v serverové paměti nacházejícího se stromu JavaServer Faces přímo souvisí s komponentami vloženými ve stránce. Následuje příklad na JSF založené JSP stránky:

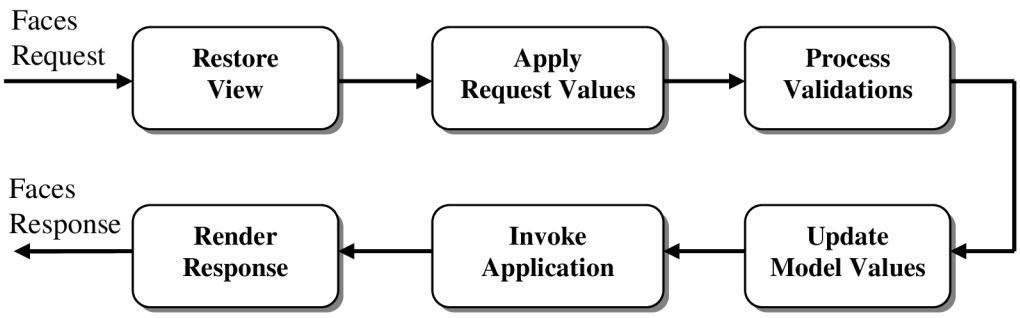
```
<%@ page contentType="text/html" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
  <html>
    <body>
      <h:form>
        <h2>
          A Simple JSF Page
        </h2>
        <h:inputText value="#{modelBean.username}" />
        <h:commandButton value="Click Here" />
      </h:form>
    </body>
  </html>
<f:/view>
```

Jak můžete vidět na obrázku 6.2, JSF komponentový strom na serveru odpovídá UI komponentám ve stránce. Jakmile jsou UI stromu v paměti, je možné komunikovat se serverovou stranou komponenty UI, manipulovat s nimi a měnit jejich vlastnosti na serveru.




**Obrázek 6.2 JSF komponentový strom [12]**

Jak je uvedeno v následujících kapitolách jsou znalosti o interakci s UI komponentovým stromem na straně serveru potřebné pro pokročilé vytváření JSF aplikací. Pro jednoduché JSF aplikace jednoduše umístíme nějaké UI komponenty na stránku, nastavíme nějaké atributy a spoléháme se na JSF zpracování vstupu. Nyní bude následovat podrobnější seznámení procesem zpracování vstupů v JSF známém jako životní cyklus zpracování JSF žádosti.




**Obrázek 6.3 Schéma frameworku Sesame [12]**

**Životní cyklus zpracování JSF žádosti.**

V případě, kdy je volána JSP stránka založená na JSF nebo v případě, když uživatel vyvolá nějakou akci v UI komponentě v JSP stránce založené na JSF, je potřebné porozumět postupu, který je vykonán na serveru za účelem splnění příslušných požadavků pro zobrazení nebo odeslání JSF stránky. Sekvence událostí, které jsou spouštěny během dotazu na JSF stránku se nazývá "*JSF request processing lifecycle*" nebo zkráceně *JSF lifecycle*. Tento cyklus je znázorněn na obrázku 6.3.

Už jsme se dotkli toho, co se stane, když přijde první dotaz na JSF stránku, kdy JSF runtime vytvoří komponentový strom v paměti. Mezi samotnými dotazy, kdy není nic prováděno v rámci aplikace, je komponentový strom většinou skryt. V okamžiku, kdy přijde dotaz je strom okamžitě znovu vytvořen. V případě, kdy jsou v rámci dotazu zaslány vstupní hodnoty, je provedeno jejich zpracovaní a ověření. Po úspěšném ověření hodnot jsou příslušná vstupní pole změněna. Následuje zpracování událostí. V případě nalezení chyb je o těchto chybách podáno hlášení. Jakmile jsou všechny vyvolané události dokončeny a je provedena aktualizace všech příslušných dat, je klientovi odeslána kompletní odpověď.

*JSF lifecycle* je tedy jednoduchý proces, v rámci něhož je vykonávána sekvence událostí, která provádí správu vstupních dat, což vede k ulehčení práce vývojáře, který nemusí tuto obsluhu vyvolané události implementovat sám. Toto je jeden z hlavních odlišností od jiných používaných technologií, kterými jsou například CGI, PHP a Struts, čímž má vývojář možnost lepšího zaměření na jiné části projektu. K obsluze příchozích dotazů se vývojář dostává pouze ve speciálních případech, kdy ji potřebuje specificky upravit.

Jednoduchým příkladem, ve kterém se používá standardní obsluha událostí, je jednoduše namapována UI komponenta vstupního pole na spravovanou vlastnost *Beany*. V tomto případě životní cyklus automaticky aktualizuje hodnotu příslušné vlastnosti *Beany* s hodnotou UI komponenty. Následuje příklad, kde je hodnota pole vstupního textu UI svázaná s „*username*" vlastností spravované *Beany* „*modelBean*" za použití *JSF expresion language* (EL).

```
<h:inputText value="#{modelBean.username}" />
```

Pro povolení odeslání formuláře uživateli a současné zinicializování *JSF lifecycle*, je třeba využít UI komponentu *commandButton*, jehož zápis na stránce vypadá následovně.

```
<h:commandButton value="Click Here" />
```

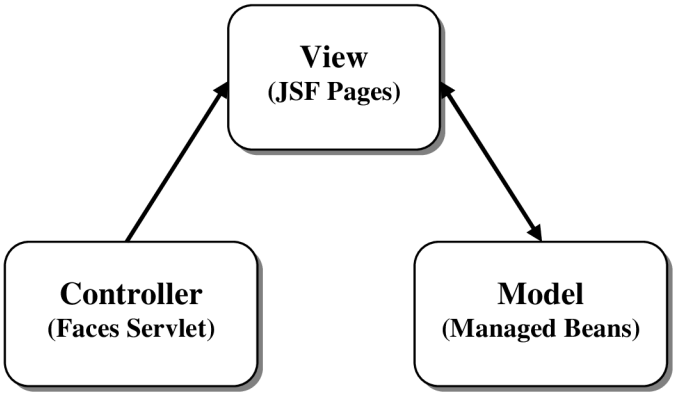
Protože *JSF lifecyklus* využije JavaBean událostní model, uživatel jednoduše klikne na vykreslené command tlačítko v runtime a *JSF lifestyle* automaticky zaktualizuje vlastnost „username" hodnotou zadanou v komponentě intutText.

### JSF navigační model

JSF je založen na architektuře Model-Veiw-Controller. Jak již bylo popsáno v kapitole 6.1.2, je MVC rozděleno do tří oddělených aplikačních komponent.

- Model, který obsahuje aplikační logiku a nevizualizační kód.
- View, který obsahuje veškerý kód zajišťující prezentaci dat a funkčnost uživatelského rozhraní.
- Controller obsluhující v rámci systému uživatelem vyvolaných událostí a včetně generování zpětné odezvy zobrazené uživateli v rámci události.

Tyto tři elementy, které jsou znázorněné na obrázku 6.4, jsou zkombinovány do této architektury, tak aby vytvořily samostatně udržovatelný kód. JavaServer Fases byla od počátku vytvářena tak, aby přesně dodržovala metodologii MVC. Čímž poskytuje prostý způsob oddělení prezentační části (View) od části datové logiky (Model). Obsahuje také *front-end servlet* (Controller), který obsluhuje všechny uživatelem vyvolané události s využitím prezentační části View. Jak již bylo zmíněno část pro zobrazení je vytvářena prostřednictvím pro JSF určených JSP stránek s uživatelskými komponentami. Model je reprezentovaný metodami a nastaveními v takzvaných „*managed beans*“ specifikovaných v *faces-config.xml*.




**Obrázek 6.4 Schéma frameworku Sesame [12]**

### Faces Controller

Nyní se zaměříme na způsob fungování Faces Controlleru v JSF aplikaci. Jak bylo zmíněno již dříve *Fases Controller* je implementován jako servlet, který odpovídá na všechny žádosti přicházející z určité URL, jako /faces/* , která je definována v souboru *web.xml*. Žádost používající vybrané URL se považuje za takzvané „*Faces request*“. Pokud tuto žádost obdrží *Faces Controller*, zpracuje žádost připravením objektu známého jako JSF kontext, který obsahuje všechna dostupná aplikační data, a nasměruje klienta na příslušnou *View* stránku. Pravidla, která používá *Controller* pro směrování podle požadavků, jsou centrálně spravována v souboru *fases-config.xml* a jsou označována jako *JSF Navigation Model*.

*JSF Navigation Model* je elegantní řešení pro uchování všech navigačních pravidel v celé JSF aplikaci. Tím se zlepšuje ovladatelnost celé aplikace, protože tento způsob spravování centrálního navigačního modelu aplikace je snadnější, než obnovování navigačních pravidel podle odkazů na měnících se stránkách. Centrální soubor s navigačními pravidly je ve formátu XML, což mu zajišťuje jednoduchost správy, jak prostřednictvím programů, tak i v textové formě samotným vývojářem.

Následuje příklad jednoduchého navigačního pravidla ze stránky *page1.jsp* na stránku *page2.jsp* prostřednictvím směrovacího řetězce, takzvaného „*outcome*“, v tomto případě *success*.

```
<navigation-rule>
  <from-view-id>/page1.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/page2.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Ještě zbývá vysvětlit, jak je *outcome* určen. Je možné ho definovat dvěma způsoby. První je jeho neměnná definice přímo ve stránce. Druhým je dynamický způsob prostřednictvím jeho návratu z metody, která je spouštěna stiskem tlačítka. Jak již bylo zmíněno, komponenta uživatelského rozhraní může být namapována na vlastnost v spravované *Bean*ě vracející String hodnotu *outcome*, podle které by bylo následně uskutečněno příslušné pravidlo přechodu mezi stránkami. Pro uskutečnění pravidla z příkladu by bylo třeba vrátit řetězec „success“.

V tuto chvíli jsme ve zkratce představili historii a způsob fungování JavaServer Faces, který jsem použil pro vytvoření distribuovaného informačního systému. Popis konkrétně použitých JSF prvků je obsahem podkapitoly 6.2.

### 6.1.4 Sesame 2

Jak jsem již uvedl v kapitole návrhu informačního systému, zvolil jsem pro uložení distribuovaných dat nástroj Sesame 2. V této části se seznámíme s jeho vlastnostmi a možnostmi. Informace v této podkapitole byly čerpány z [13].

**Informace o Sesame 2**

Sesame 2 je open-source Java framework pro skladování, odvozování a dotazování RDF dat. Původně byl vyvinut společností Aduna pro účely EU projektu On-To-Knowledge. V současné době je dále rozvíjen a udržován společností Aduna ve spolupráci s nadací NLnet, vývojáři ze společnosti Ontotex a skupinou dobrovolných vývojářů. Jeho podpora je k dispozici na webových stránkách www.openrdf.org .

Sesame je navržen s ohledem na jeho flexibilitu. V rámci ukládacích mechanismů, rozhraní, RDF souborových formátů, výsledků dotazů a dotazovacích jazyků je tento framework plně rozšiřitelný a konfigurovatelný. Je ho možné použít v různých systémech. Nabízí velkou škálu nástrojů umožňující vývojářům využít možnosti RDF a RDF schemat. Mezi tyto nástroje patří i flexibilní přístupové API podporující lokální i vzdálený přístup. Sesame 2 podporuje dva jazyky pro získávání a ukládání dat do úložiště. Jsou to jazyky SeRQL a SPARQL.

Prostřednictvím Sesame 2 je možno použít několik možností uložení dat. Jedná se o uložení dat v paměti. Uložení dat prostřednictvím nativního úložiště na disku. Úložiště ukládající data v databázi PostgreSQL. Úložiště ukládající data do databáze MySQL. A vzdálené úložiště, které využívá pro ukládání dat server Sesame.

Oproti předchozí verzi Sesame 1 nabízí Sesame 2 mnoho nové funkčnosti a vylepšení. Bohužel však Sesame 2 není z důvodu vykonaných změn, především změn API, zpětně kompatibilní s předešlými verzemi. Nyní zmíním několik nejdůležitějších novinek v Sesame 2, kterými jsou:

- Plné zaměření a kompatibilita s Javou od verze 5.
- Přepracované API úložiště mnohem více zaměřené na použití sesame jako knihovny při vývoji aplikací.
- Úplná podpora transakcí a zpětný návrat změn v případě chyby.
- Podpora dotazovacího jazyka SPARQL.

- Podpora protokolu SPARQL a výsledků SPARQL dotazů ve formátu XML v rámci HTTP protokolu, které jsou v současné době doporučením W3C.

Vlastní Sesame funkčnosti je možné dále rozvíjet prostřednictvím rozšíření a zásuvných modulů. Tato rozšíření jsou dostupná na domovské stránce Sesame. Pochází od externích vývojářů. Jsou to například moduly rozšiřující použiti o další jazyky, další jiná úložiště dat či pluginy pro editory.

**Sesame komponenty**

Nyní si představíme architekturu frameworku Sesame. Její schéma je zobrazeno na obrázku 6.5. Na tomto obrázku jsou znázorněny nejvýznamnější součásti a rozhraní API v Sesame a jejich návaznost na pod nimi ležícími komponentami. Každá komponenta/API je tedy závislé na komponentě/API, nad kterou leží.

Celou nejnižší vrstvu pokrývá ***RDF model***, základ Sesame Frameworku. Jelikož se jedná o RDF orientovaný rámec, jsou všechny části Sesame do určité míry závislé na tomto RDF modelu, který implementuje rozhraní a implementace pro všechny základní RDF entity jako URI, prázdné uzly, litorály a výroky.

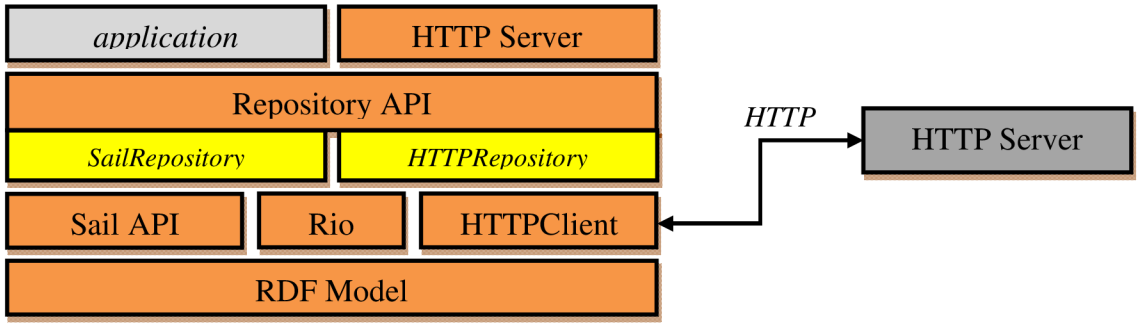
***Rio*** znamenající zkratku RDF I/O. Skládá se ze sady parserů a writerů pro různé formáty RDF souborů. Parsery mohou být použity pro transformaci RDF souborů na sadu výrazů. Pro opačné operace jsou určeny writery. Rio může být použito i nezávisle na zbytku Sesame.

Z anglického Storage And Inference Layer vychází název další komponenty ***Sail Api***. Jedná se o nízkou úroveň systémové API pro RDF uložení a dedukci nad daty. Účelem této vrstvy je získávání dat z úložiště včetně vykonávání dedukčních aplikací nad těmito daty. Tím je umožněno použít různé druhy úložišť. O *Sail API* by se měli zajímat především ti, kteří vyvíjejí *Sail* implementaci. Ostatním postačí znalosti dostačující pro vytvoření a nastavení pouze jednoho *Sail API*. Existuje několik aplikací *Sail API*. Je to například *MemoryStore*, který ukládá RDF data v hlavní paměti nebo *NativeStore* používající pro uložení datové struktury uložené na disku.

Na vyšší úrovni se již nachází ***Repository API***. Tato vrstva nabízí metody pro manipulaci s RDF daty, které jsou určené pro vývojáře. Jejím hlavním úkolem je ulehčení práce vývojářům. *Repository API* obsahuje metody pro ukládání datových souborů, dotazování, získávání a manipulaci s daty. Pro tuto manipulaci s daty existuje několik implementací tohoto API, na obrázku 6.5 jsou znázorněny do této skupiny patříci *SailRepository* a *HTTPRepository*. První slouží pro překlad volání pro použitý *Sail*. Druhý *HTTPResponsitory* nabízí transparentní klient-server komunikaci se Sesame serverem přes HTTP.

Nejvyšší komponentou ve schématu je ***HTTP Server***. Tento server se skládá z řady Java servletů implementujících přístup k Sesame úložišti přes HTTP protokol.

Jak jsem již uvedl, jsou všechny části implementace Sesame veřejně přístupné. Pro vývojáře je nejdůležitější částí *Repository API*. S některými jeho vlastnostmi se ještě seznámíme v další části textu.

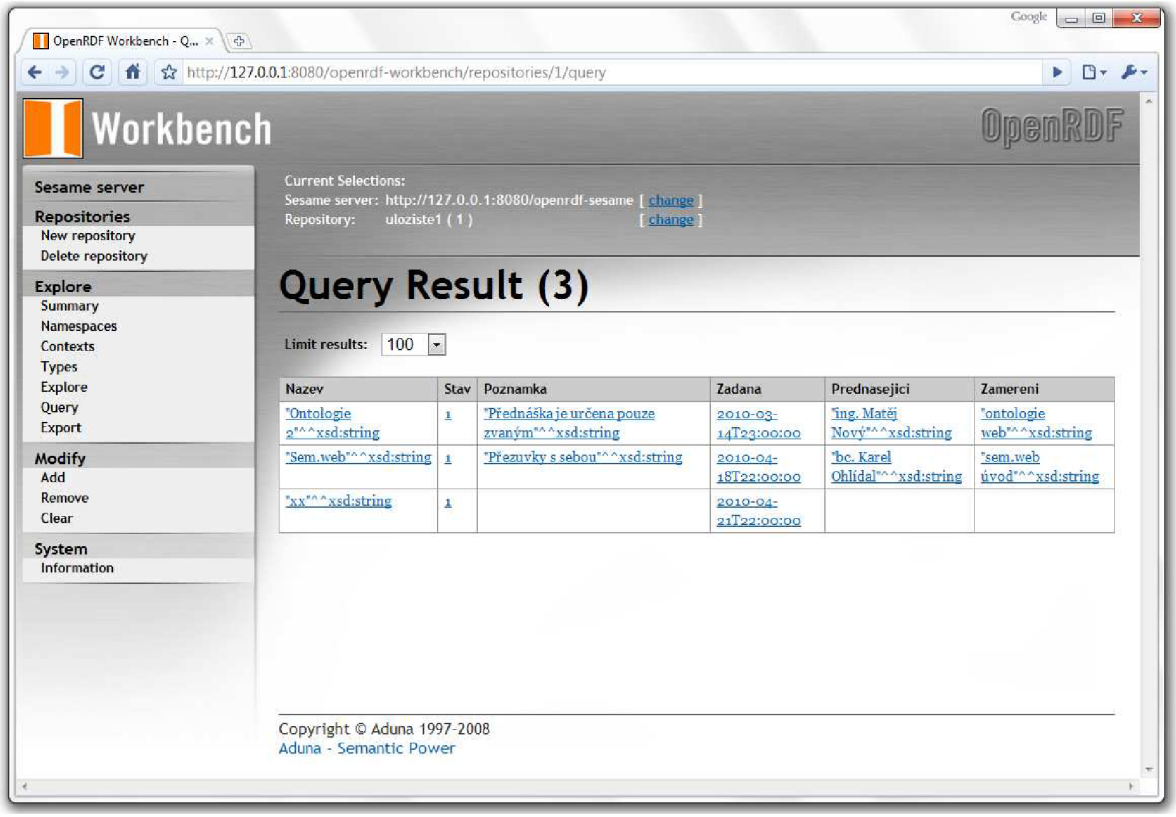



**Obrázek 6.5 Schéma frameworku Sesame [13]**

### Sesame server

Jak již bylo zmíněno Sesame obsahuje několik částí, které je možné využít. Jednou z ní je Sesame server. Pro spuštění Sesame server je třeba mít v systému Javu 5 nebo vyšší. Dalším softwarovým požadavkem je Java Servlet kontejner podporující Java Servlet API 2.4 a Java Server Pages (JSP) 2.0 nebo vyšší. Doporučován je například Apache Tomcat. Ukázka vzhledu Sesame serveru je na obrázku 6.6.

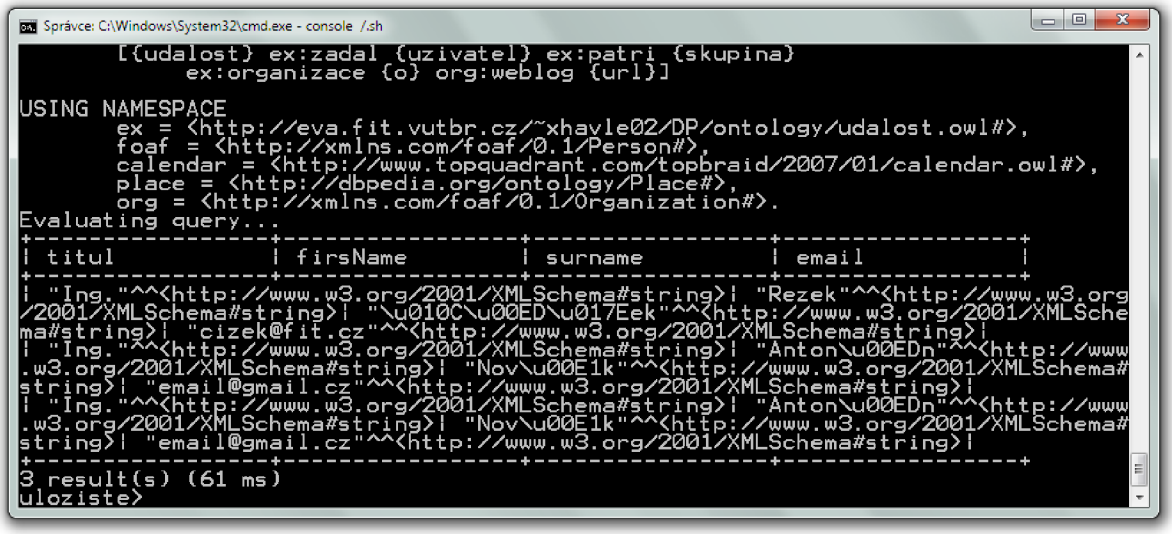
Sesame 2 server existuje ve dvou provedeních Java webové aplikace. Je to *Sesame (HTTP) server* a *OpenRDF Workbench*. *Sesame server* poskytuje HTTP přístup k Sesame úložišti a je určen pro použití v jiných aplikacích. Kromě výpisu logovacích zpráv neposkytuje žádnou pro uživatele orientovanou funkčnost. Na uživatele orientovaná funkčnost je obsažena v *OpenRDF Workbench*, který poskytuje webové rozhraní pro dotazování, aktualizaci a zkoumání úložiště Sesame server.




**Obrázek 6.6 Sesame server**

### Sesame konzole

Další možností pro použití Sesame úložiště je Sesame konzole. Jedná se o takzvanou commad-line aplikaci pro interakci se Sesame. Je ji možno použít v příkazovém řádku v operačním systému Windows nebo v terminále unixových operačních systémů. Jejím prostřednictvím je možné vytvořit a spravovat Sezame úložiště v rámci systému. Ukázka vzhledu Sesame konzole je na obrázku 6.7. Nyní popíši možnosti Sesame konzole prostřednictvím příkladu vytvoření Sesame úložiště.




**Obrázek 6.7 Sesame konsole**

Při vytváření nového úložiště se je nejprve třeba prostřednictvím příkazu *connect* s parametrem umístění úložiště připojit k příslušnému úložišti. Poté je možné prostřednictvím příkazu *create* vytvořit některý z podporovaných typů úložiště, kterými jsou tyto:

- ***memory*** - RDF úložiště umístěné v paměti
- ***memory-rdfs*** - RDF úložiště umístěné v paměti s RDF Schema a odvozením
- ***memory-rdfs-dt*** - RDF úložiště umístěné v paměti s RDF Schema a přímým hierarchickým odvozením
- ***native*** - úložiště v datové struktuře na disku
- ***native-rdfs*** - úložiště v datové struktuře na disku s RDF Schema a odvozením
- ***native-rdfs-dt*** - v datové struktuře na disku s RDF Schema a přímým hierarchickým odvozením
- ***pgsql*** - úložiště, které ukládá data do PostgreSQL databáze
- ***mysql*** - úložiště, které ukládá data do MySQL databáze
- ***remote*** - úložiště, které slouží jako proxy pro úložiště na Sesame serveru

Pro provedení příkazu bude třeba vyplnit několik údajů vytvářeného úložiště. Po vytvoření se lze k úložišti připojit pomocí příkazu *open* s parametrem jména úložiště. Nyní již lze manipulovat s daty v úložišti, a to buď prostřednictvím jazyků SeRQL a SPARQL, nebo například vkládáním RDF souborů s daty příkazem *load*. Další příkazy je možné nastudovat z nápovědy vyvolané prostřednictvím příkazu *help*.

Úložišti je možné nastavit další vlastnosti. Mezi tyto vlastnosti patří například zpoždění synchronizace či nastavení indexace ukládaných dat u nativního úložiště.

### API úložiště

Repository API neboli úložiště API je centrální přístupový bod pro úložiště Sesame. Jedná se o pro vývojáře určený přístupový bod k RDF úložištím, které nabízejí různé metody pro dotazování a aktualizaci dat, přičemž obalují kostrbatou funkčnost, se kterou se tedy programátor nemusí setkávat. V této části se seznámí čtenář se základními informacemi ohledně programování využívajícího Sesame úložiště.

Prvním krokem při jakékoli činnosti, kterou vykonáváme se Sesame úložištěm, je vytvoření objektu *Repository*. Existuje několik jeho implementací.

Třída *org.openrdf.repository.sail.SailRepository* používaná k přístupu k úložišti. *SailRepository* pracuje nad objektem *Sail* pro ukládání a načítání RDF dat. Je důležité si uvědomit, že chování

úložiště je definováno objekty *Sail*, které s ním manipulují. Například úložiště bude podporovat pouze RDF schema nebo OWL sémantiku, pokud *Sail stack* pro to obsahuje inferencer.

*org.openrdf.repository.http.HTTPRepository* je jak název napovídá implementace *Repository*, která se chová jako proxy pro Sesame úložiště k dispozici na vzdáleném serveru Sezame přístupném přes HTTP.

Jak je uvedeno ve specifikaci Sesame úložiště RDF dat může mít několik podob. Pro vytvoření všech těchto úložišť slouží příkaz *new SailRepository(par)* s parametrem určujícím typ vytvářeného úložiště. Pro vytvoření vzdáleného úložiště slouží příkaz *new HTTPRepository (par1, par2)*, jehož parametry jsou adresa příslušného vzdáleného serveru a identifikátor nového úložiště.

Nyní, když již máme představu o tom, jak vytvořit úložiště, si potřebujeme ujasnit, jak můžeme s úložištěm provádět další úkony. V Sesame 2 je toho dosaženo pomocí objektu *RepositoryConnection*, který může být vytvořen pomocí *Repository*. *RepositoryConnection* představuje, jak název napovídá, připojení k reálnému úložišti. Pomocí tohoto připojení můžeme provádět různé operace. Po dokončení potřebných úkonů je třeba toto připojení uzavřít, abychom nezanechávali úložiště, ke kterému jsme připojeni, uzamčené. Což by vedlo k nemožnosti práce s úložištěm prostřednictvím jiných připojení.

Přidání dat do úložiště může být realizováno různými způsoby. Pro zadání dat může být použito vložení dat ze souboru obsahující data v RDF formátu. Data mohou být vkládána samostatně tak i v kolekcích.

Data je také možné spravovat prostřednictvím vytváření, vyhledávání či odstraňování jednotlivých výroků. Tento způsob správy dat také využívá *RepositoryConnection* společně s třídou *ValueFactory*, která umožňuje vytváření výroků.

*Repository API* má několik metod pro vytváření a vyhodnocování dotazů. Rozlišují se tři typy dotazů. *Tuple* dotazy jsou dotazy, jejichž výsledkem je množina n-tic. Kde každá n-tice představuje řešení dotazu. Tento typ dotazu je vhodný pro získávání konkrétních dat z RDF úložiště. Dalším typem dotazu je *grafový dotaz*, který po provedení navrací RDF graf neboli soubor výroků. Tento typ dotazu je užitečný pro získání podgrafů z RDF úložiště dat, které mohou být dále využívány pro dotazování. Posledním typem jsou *boolean dotazy*, jejichž výsledkem je jednoduchá boolean hodnota true nebo false, tedy platí či neplatí. Tento typ dotazu je používán pro ověření přítomnosti nějaké informace v databázi.

Sesame 2 podporuje dva dotazovací jazyky, jimiž jsou SeRQL a SPARQL. Nastíněním používání SeRQL se bude zabývat následující část textu. Specifikace jazyka SPARQL je dostupná online na http://www.w3.org/TR/rdf-sparql-query/ .

Většina výše uvedených technik pracuje na úrovni jednoduchých výroků. Nicméně *Repository API* nabízí několik metod pracujících s kolekcemi výroků, což umožňuje provádět dávkové operace jako například update.

Sesame 2 podporuje pojem *kontextu*, který si lze představit jako způsob, jak lze označit skupinu výroků jedním společným identifikátorem, prostřednictvím něhož se bude možné na tuto skupinu odkazovat. Identifikátorem může být jak prázdný uzel, tak i URL. Typickým využitím kontextu je sledování výrazů podle toho z jakého souboru pocházejí. Příkladem je přidání dat do úložiště z různých souborů. V případě, kdy je některý z těchto souborů aktualizován, je třeba tato data aktualizovat i v úložišti. V této situaci se nabízí kontext, prostřednictvím něhož je možné příslušná data z úložiště odstranit a nahradit je novými aktuálními. Výroky mohou být svázány i s více kontexty zároveň.

*RepositoryConnection* rozhraní podporuje plně transakční mechanismus. Tento mechanismus umožňuje seskupení několika operací a jejich považování za jediný update. Tato skupina operací se poté provádí jako transakce, tedy pokud není některá z operací v rámci transakce úspěšná, neprojeví se změny ani jedné z operací. Operacemi provedené změny budou natrvalo uskutečněny pouze

v případě, kdy budou úspěšně uskutečněny všechny operace, tedy celá transakce. Sdružování operací do transakčních skupin má také za následek snížení četnosti updatů v rámci úložiště, a tím i zrychlení práce úložiště.

**SeRQL**

Sesame RDF Query Language neboli SeRQL je RDF dotazovací jazyk, který je velmi podobný SPARQL, má však odlišnou syntaxi. SeRQL byl původně vyvinut jako lepší alternativa pro dotazovací jazyk RQL a RDQL. Mnoho funkcí z SeRQL lze nalézt v SPARQL a naopak SeRQL přijala některé rysy z SPARQL.

Základními stavebními prvky RDF jsou URI a literály. Proměnné v SeRQL jsou definovány pomocí názvů. Jejich jména musí začínat písmenem nebo podtržítkem. V názvu proměnných nesmí být použita SeRQL klíčová slova. Klíčová slova jsou v SeRQL case-insensitive. Názvy proměnných jsou však case-sensitive.

URI je možné v SeRQL zapsat dvěma způsoby, buď jako úplné URI, kdy musí být URI obklopeno „<“ a „>“. Nebo prostřednictvím zkrácené formy, která se skládá z definované předpony, kterou následuje „:“ a poté řetězec doplňující předponu na celé URI.

Literály se skládají ze tří částí, jimiž jsou popisek, značka jazyka a datový typ. Značka jazyka a datový typ jsou volitelné a není možné, aby se vyskytovaly současně. Nanejvýše jednu z nich může doplňovat popisek. Pro zaznamenání značky jazyka k popisu literálu slouží „@“, pro datový typ pomocí URL „^ ^“. Pro zkrácení URL zápisů lze použít prefix, na který je mapován jmenný prostor. Následuje ukázka zápisu literálů:

```
"Foo"
"Foo"@en
"<foo/>"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns
           #XMLLiteral>
"<foo/>"^^rdf:XMLLiteral
```

SeRQL umožňuje pracovat s prázdnými uzly. Jedná se o uzly, které nejsou reprezentovány ani pomocí URI ani literálů. V dotazech pro prázdné uzly umožňuje použít identifikátory.

Jednou z nejvýznamnějších částí SeRQL jsou *path expressions*. Jsou to výrazy, které odpovídají konkrétní cestě RDF grafem.


**Obrázek 6.8 Základní path expressions [13]**

Na obrázku 6.8 je graficky znázorněna cesta pro osobu, která pracuje ve společnosti, která je typu IT společnost. V notaci SeRQL by tato cesta vypadala následovně.

```
{Person} foo:worksFor {Company} rdf:type {foo:ITCompany}
```

Části obklopené složenými závorkami reprezentují uzly v RDF grafu, části mezi těmito uzly představují hrany grafu. Směr vztahu je v SeRQL výrazech vždy zleva doprava. Zápis lze upravit do dvou kratších výrazů oddělených čárkou.

```
{Person} foo:worksFor {Company},
{Company} rdf:type {foo:ITCompany}
```

V případě, že nemáme zájem o hodnoty v uzlu, můžeme použít prázdný uzel.

```
{Person} foo:worksFor {} rdf:type {foo:ITCompany}
```

Takto vytvářené výrazy pro dosažení žádaného výsledku lze různými způsoby kombinovat. Pokud v rámci výrazu získáváme i hodnotu, která nemusí být vždy definovaná, musíme tuto skutečnost ve výrazu příslušným způsobem definovat. Pro zápis takovéto nepovinné cesty je třeba použít *volitelnou cestu výrazu*. Tuto cestu v zápisu uzavřeme do „[“ a „]“. Příkladem výrazu navracející nepovinnou emailovou adresou je následující.

```
{Person} ex:name {Name};
         ex:age  {Age};
        [ex:email {EmailAddress}]
```

Opět je možné výraz použitím jiného stylu využití čárek či středníků upravit do různých podob. Volitelné cesty výrazu mohou být také vnořené.

SeRQL obsahuje dva koncepty dotazů. První z nich je dotaz vracející tabulku hodnot nebo nastavující proměnné hodnoty. Druhý je dotaz vracející RDF graf. První typ dotazu se nazývá „*select queries*“ a druhý „*construct queries*“.

*Select dotaz* je typicky složen z několika částí. Jsou to klauzule `SELECT`, `FROM`, `FROM CONTEXT`, `WHERE`, `LIMIT`, `OFFSET` a `USING NAMESPACE`. Možná znáte některý z těchto klausulí z databázového dotazovacího jazyka SQL, jejich použití však není zcela stejné. V rámci konstrukce dotazů se jejich části neliší. Výjimku tvoří pouze dotazy začínající klauzulí `CONSTRUCT` místo `SELECT`. Kromě první klauzule, kterou může být `SELECT` nebo `CONTEXT`, jsou další nepovinné.

První klauzule (`SELECT` nebo `CONSTRUCT`) určuje to, co se provádí s výsledky, které jsou nalezeny. V rámci klauzule `SELECT` je možné stanovit konkrétní hodnoty, které mají být navráceny. Naproti tomu se v klauzuli `CONSTRUCT` stanovují výroky, které mají být navráceny.

Klauzule `FROM` specifikuje cestu výrazu, která byla popsána výše. Zde jsou třeba definovat cesty v RDF grafu specifikující vyhledávaná data.

Klauzule `FROM CONTEXT` umožňuje v rámci dotazu stanovit kontext vyhledávaných dat.

`WHERE` určuje další vlastnosti vyhledávaných dat. Jedná se o omezení vztahující se na uzly a konce cest, které není možné vyjádřit prostřednictvím definice cest v rámci klauzule `FROM`.

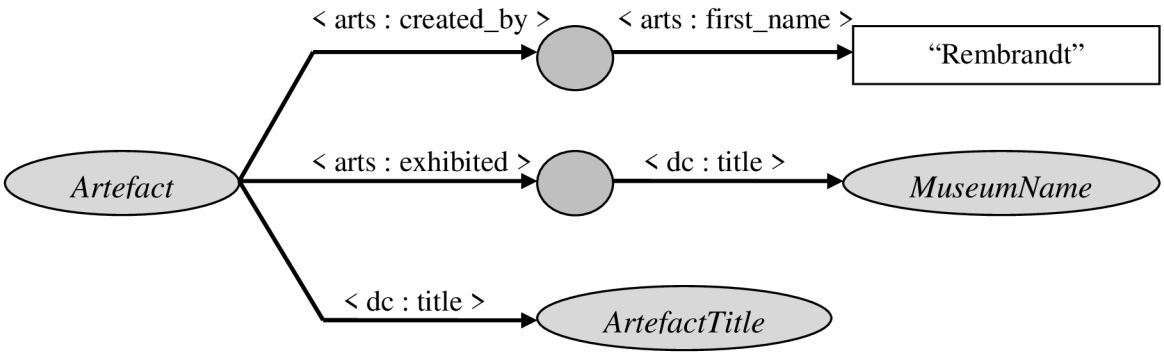
Klauzule `LIMIT` a `OFFSET` je možné použít samostatně nebo v kombinaci s jinými klauzulemi s cílem získání podmnožiny vyhledaných dat odpovídajících ostatním vlastnostem dotazu. Jejich použití se velice podobá použití `LIMIT` a `OFFSET` klauzulím z jazyka SQL. `LIMIT` určuje maximální počet n-tic, které budou navráceny. `OFFSET` určuje, která n-tice bude vrácena jako první přeskočením tolika výsledků, kolik je uvedeno.

Použití `USING NAMESPACE`, je určeno pro deklarování namespace prefixů. Jedná se o mapování předpon použitých v dotazu na jmenné prostory (URI).

Příklad dotazu v jazyce SeRQL je na obrázku 6.9. Na obrázku 6.10 je zobrazena dotazu odpovídající *path expressions*. V rámci tohoto krátkého seznámení s dotazovacím jazykem SeRQL byl čtenář seznámen s jeho hlavními rysy. Nebylo tedy možno obsáhnout všechny informace s tímto jazykem spojené. Podrobněji se může čtenář seznámit s tímto jazykem ve zdroji [13], ze kterého byly tyto informace čerpány.

```
SELECT DISTINCT
   label(ArtefactTitle), MuseumName
FROM
   {Artefact} arts:created_by {} arts:first_name {"Rembrandt"},
   {Artefact} arts:exhibited {} dc:title {MuseumName},
   {Artefact} dc:title {ArtefactTitle}
WHERE
   isLiteral(ArtefactTitle) AND
   lang(ArtefactTitle) = "en" AND
   label(ArtefactTitle) LIKE "*night*"
USING NAMESPACE
   dc   = <http://purl.org/dc/elements/1.0/>,
   arts = <http://example.org/arts/>
```

**Obrázek 6.9 Příklad dotazu v jazyce SeRQL [13]**




**Obrázek 6.10 Path expressions pro příklad dotazu [13]**

### 6.1.5 OWL API

Pro distribuci dat mezi vytvářenými informačními systémy jsou používány sémantické technologie. Konkrétně soubory se sémantickými daty vyjádřenými prostřednictvím jazyka OWL. Vytváření těchto souborů je možné přímým způsobem prostřednictvím skládání řetězců do owl zpráv, nebo použitím ke generování OWL souborů určené aplikace. Požadavkem na tuto aplikaci je možnost jejího začlenění do navrženého systému, především tedy kompatibilita knihoven s Javoským vývojovým prostředím.

Pro vytváření přenášených datových OWL souborů jsem zvolil OWL API. Dále uvedené informace byly čerpány z [14], kde jsou k dispozici i příklady jejího použití. OWL API je v Javě implementované API umožňující vytváření, manipulaci a serializaci OWL ontologií. Jedná se o open source projekt dostupný pod GLPL licencí. Nejnovější verze OWL API se zaměřuje na OWL 2.

První verze API pro OWL 1.0 byla vyvinuta jako součást projektu WonderWeb. Verze OWL API 2.0 byla vyvinuta jako součást projektů CO-ODE a TONES. Verze 3.0 je vyvíjena převážně na Universyty of Manchester.

Funkčnost OWL API je následující:

- API pro OWL 2 a efektivní v paměti referenční implementaci.
- RDF / XML parser a writer
- OWL / XML parser a writer
- OWL Functional Syntax parser a writer
- Turtle parser a writer
- KRSS parser
- OBO Flat file format parser
- Reasoner rozhraní pro práci s reasonery jako FAK++, HermiT, Pellet a Racer

OWL API se skládá ze základního rozhraní poskytující datové struktury, objektů pro manipulaci a změnu OWL dat, vstupně výstupní *Factory* a *Storer* třídy a obalové parsery a renderery. V neposlední řade je součástí i Manager objekt pro spojení funkčnosti standardních komponent.

OWL API je použit v mnoha známých sémantických aplikacích jako například v OWL editorech Protégé 4, Pallet, SWOOP či OntoTrack.

## 6.2 Popis implementovaného systému

V této kapitole bude popsána implementace navrženého distribuovaného informačního systému. V rámci této implementace byly využity technologie uvedené v předchozí podkapitole. Nejprve bude popsána implementace prvků uživatelského rozhraní. Následovat bude popis implementace funkční části, popis implementace databázové části a implementace distribuce dat.

### 6.2.1 Implementace prvků uživatelského rozhraní

Uživatelské rozhraní v rámci architektury MVC vrstva View je implementováno prostřednictvím JSP stránek s příslušnými pro JavaServer Faces určenými tagy. Implementace rozhraní vychází z diagramu návaznosti obrazovek zobrazeného na obrázku 4.5, který byl vytvořen v rámci návrhu informačního systému.

Sceenshot aplikace je zobrazen na obrázku 6.11. Hlavní okno je v implementaci realizováno kombinací stánek ***index.jsp*** s vloženým záhlavím ***header.jsp*** a zápatím ***footer.jsp***. ***index.jsp*** obsahuje základní informace seznamující uživatele s vlastnostmi informačního systému. ***header.jsp*** implementuje zobrazení výběrového navigačního stromu v levé části obrazovky. Zobrazené možnosti navigace jsou upravovány v závislosti na oprávnění přihlášeného uživatele. V případě, že není uživatel přihlášen, obsahuje tento strom pouze odkaz na úvodní stranu systému. ***footer.jsp*** implementuje informace obsahující stránka v pravém sloupci. V případě, kdy není uživatel přihlášen, je zde zobrazena výzva pro přihlášení s odkazem na přihlašovací stránku. Pokud uživatel přihlášen je, jsou zde zobrazeny jeho osobní informace včetně informace o skupině, do které patří. JSP stránky ***header.jsp*** a ***foother.jsp*** jsou vloženy ve všech stránkách systému, čímž je zajištěno zobrazení příslušných informací, které byly popsány výše, v rámci celého systému.

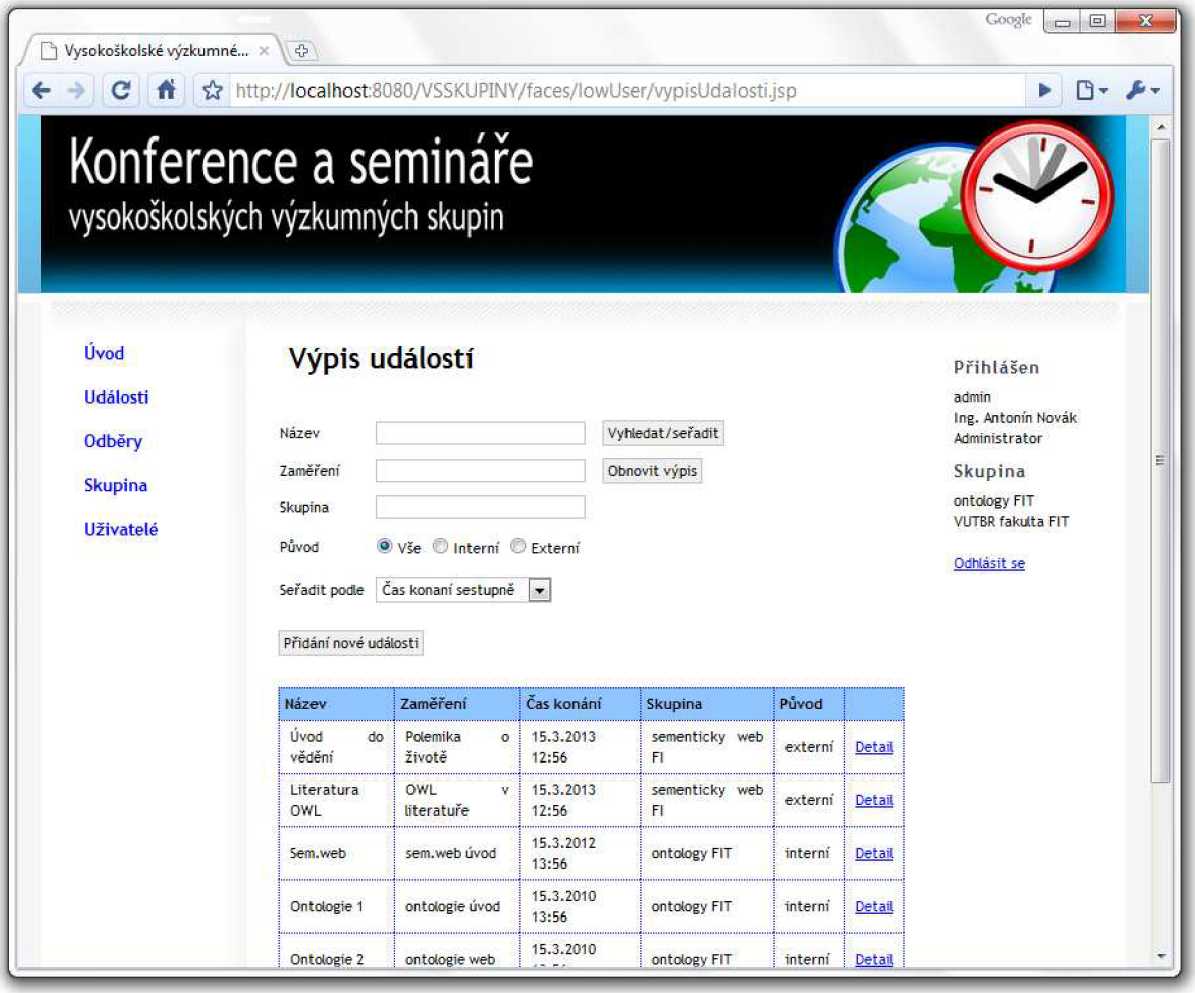
Správa zobrazení informací o konferencích a seminářích je realizována JSP stránkami ***vypisUdalosti.jsp***, ***detailUdalosti.jsp*** a ***pridaniUpravaUdalosti.jsp***. ***vypisUdalosti.jsp*** realizuje výpis veškerých konferencí a seminářů v podobě tabulky, jejich vyhledávání a řazení. Detailní výpis

informací o příslušné konferenci nebo semináři je realizován stránkou ***detailUdalosti.jsp***. Funkčnost přidání či úpravy konference, či semináře je realizována stránkou ***pridaniUpravaUdalosti.jps***.

Informace vztahující se k odběrům jsou v rámci prvků uživatelského rozhraní spravovány prostřednictvím JSP stránek ***vypisOdberu.jsp, detailOdberu.jsp***, ***pridaniUpravaOdberu.jsp. vypisOdberu.jsp*** implementuje zobrazení zadaných odběrů v rámci příslušné skupiny. Detail odběru je zobrazen prostřednictvím stránky ***detailOdberu.jsp***. Zadání a úprava odběru je prezentována JSP stránkou ***pridaniUpravaOdberu.jsp***.

V rámci prezentace a úpravy údajů skupiny jsem implementoval JSP stránku ***detailSkupiny.jsp*** zobrazující údaje o skupině, do které patří přihlášený uživatel, a ***upravaSkupiny.jsp***, která zprostředkovává úpravu jejich údajů.

Poslední skupinou JSP stránek, jsou stránky prezentující funkčnost v rámci správy uživatelů systému. Jsou to JSP stránky ***vypisUzivatelu.jsp*** pro vypsání vyhledávání uživatelů a ***pridaniUpravaUzivatele.jsp*** realizující přidání a úpravu uživatelů systému v rámci příslušné skupiny.




**Obrázek 6.11 Ukázka rozhraní vytvořeného systému**

V rámci filtrace přístupu stránek podle oprávnění aktuálního uživatele jsou JSP stránky umístěny do čtyř adresářů:

- ***public/*** se stránkami přístupnými i nepřihlášenému uživateli *index.jsp*, *header.jsp*, *footer.jsp*, *login.jsp*.

- ***lowUser/*** se stránkami přístupnými uživateli od nejnižšího stupně oprávnění *vypisUdalosti.jsp*, *detailUdalosti.jsp*, *detailSkupiny.jsp* a *pridaniUpravaUzivatele.jsp***.**
- ***authUser/*** se stránkami přístupnými uživateli od středního stupně oprávnění *vypisOdberu.jsp*, *detailOdberu.jsp*, *pridaniUpravaOdberu.jsp*, *pridaniUpravaUdalosti.jsp* a *upravaSkupiny.jsp*.
- ***admin/*** obsahující stránky přístupné pouze uživateli s nejvyšším stupněm oprávnění *vypisUzivatelu.jsp*.

### 6.2.2 Implementace funkční části systému

Implementace funkční části neboli části kontroleru v rámci architektury MVC je implementována prostřednictvím JSF JavaBean nacházejících se v balíčku *com.vsskupiny.back*.

Třída ***UdalostBean.java*** implementuje prostřednictvím JSF Bean funkčnost kontroleru spojenou s konferencemi a semináři. Realizuje získávání dat konferencí a seminářů z vrstvy modelu a jejich předávání komponentám vrstvy View, kterými jsou JSP stránky ***vypisUdalosti.jsp***, ***detaiUdalosti.jsp*** a ***pridaniUpravaUdalosti.jsp***. Taktéž zajišťuje funkčnost spojenou s promítnutím změn zadaných ve vrstvě View do datového úložiště. V rámci zobrazení distribuovaných dat také získává prostřednictvím níže popsané třídy ***RDFRepository.java*** distribucí získaná data a ta společně s daty získanými z databáze zasílá příslušným stránkám uživatelského rozhraní.

Třída ***OdberBean.java*** implementuje prostřednictvím JSF Bean funkčnost kontroleru spojenou s odběry v rámci distribučního mechanismu systému. První částí její funkčnosti je zajištění získávání dat souvisejících s odběry z vrstvy Modelu včetně jejich zasílání JSP stránkám ***vypisOdberu.jsp, detailOdberu.jsp***, ***pridaniUpravaOdberu.jsp*** a promítání změn zadaných ve vrstvě View do datového úložiště. Druhou částí funkčnosti třídy ***OdberBean.java*** je realizace získání dat prostřednictvím třídy ***LoadOWLData.java*** z odběrů v rámci distribuce dat mezi informačními systémy. Tato data jsou prostřednictvím dále popsané třídy ***RDFRepository.java*** ukládána do sémantického úložiště. Realizuje také mazání dat příslušných odběrů ze sémantického úložiště.

Třída ***SkupinaBean.java*** implementuje prostřednictvím JSF Bean funkčnost kontroleru spojenou se správou dat příslušné skupiny. Zajišťuje získávání dat skupiny z vrstvy Modelu včetně jejich zasílání JSP stránkám ***detailSkupiny.jsp*** a ***upravaSkupiny.jsp*** a promítání změn zadaných ve vrstvě View do datového úložiště.

Třída ***UzivatelBean.java*** implementuje prostřednictvím JSF Bean funkčnost kontroleru spojenou se správou uživatelů příslušné skupiny. Zajišťuje získávání dat skupiny z vrstvy Modelu včetně jejich zasílání JSP stránkám ***vypisUzivatelu.jsp*** a ***pridaniUpravaUzivatele.jsp*** a promítání změn zadaných ve vrstvě View do datového úložiště.

Poslední skupinou tříd jsou třídy implementující správu autentizace uživatele systému včetně ověřování oprávnění přístupu k jednotlivým stránkám vrstvy View nacházející se v balíčku *com.vsskupiny.auth*. Jedná se o JSF Bean třídu ***AuthenticationBean.java*** implementující funkčnost spojenou s autentizací uživatele, do níž spadá implementace přihlášení, odhlášení a poskytování informací o přihlášeném uživateli. Dále je to třída ***AuthenticationFilter.java*** realizující filtraci přístupu ke stránkám podle práv aktuálního uživatele. Od této ***AuthenticationFilter.java*** je zděděna třída ***LowUserAuthenticationFilter.java*** filtrující stránky přístupné od uživatele s nejnižším stupněm oprávnění, třída ***AuthUserAuthenticationFilter.java*** filtrující stránky přístupné od uživatele se středním stupněm oprávnění a třída ***AdminAuthenticationFilter.java*** filtrující stránky přístupné pouze uživateli s nejvyšším stupněm oprávnění. Tyto filtry jsou použity v rámci přístupu do adresářů uvedených v podkapitole 6.1.2, v nichž se nachází stránky s příslušným stupněm oprávnění. Je to tedy ***LowUserAuthenticationFilter.java*** pro adresář ***lowUser/***, ***AuthUserAuthenticationFilter.java*** pro adresář ***authUser/*** a ***AdminAuthenticationFilter.java*** pro adresář ***auth/***.

### 6.2.3 Implementace databáze

Pro uložení lokálních dat systému byl použit databázový systém MySQL. Připojení k němu bylo implementováno prostřednictvím kombinace nastavení projektu v Glassfish vývojového prostředí a příslušnou implementací v rámci použitého frameworku JavaServer Faces.

Z pohledu MVC přístupu využitém v rámci návrhu je vrstva Modelu implementována prostřednictvím vygenerovaných tříd v balíčku *com.vsskupiny.data*. Tyto třídy mapují jednotlivé tabulky databáze na objekty, se kterými je v rámci implementace manipulováno. Jsou to tyto:

- ***Udalost.java*** – pro správu dat konferencí a seminářů
- ***Odber.java*** – pro správu dat odběrů
- ***Skupina.java*** – pro správu dat skupiny
- ***Uzivatel.java*** – pro správu dat uživatelů systému

### 6.2.4 Implementace distribuce dat

Mechanismy spojené s distribucí dat jsou implementovány prostřednictvím tříd v balíčku *com.vsskupinz.rdf*.

Třída ***DistributeOWLData.java*** je implementací servletu naslouchajícího na příslušné adrese. Na této adrese zpracovává GET či POST požadavky, podle kterých generuje a na adresu, ze které byl požadavek odeslán, navrací příslušná data v OWL formátu. Pro generování OWL dat využívá OWL API popsané v podkapitole 6.1.5.

Třída ***LoadOWLData.java*** implementuje navázání http spojení a stažení dat z příslušné adresy. Jak bylo zmíněno v 6.2.2 je používána pro stažení OWL dat konferencí a seminářů z jiného systému.

Třída ***RDFRepository.java*** realizuje funkčnost spojenou se sémantickým úložištěm obsahující úkony vytvoření, připojení úložiště, vkládání, získání, aktualizace a mazání dat v úložišti. Pro implementaci úložiště byl použit framework Sesame 2 popsaný v podkapitole 6.1.4. Vkládání dat je realizováno nahráním souboru dat v OWL formátu s kontextem v podobě adresy příslušného odběru. Mazání je realizováno prostřednictvím smazání dat s určitým kontextem. Aktualizace je realizována kombinací mazání a vkládání. Získávání dat konferencí a seminářů je realizováno provedení SeRQL dotazu nad připojeným úložištěm. Zápis tohoto dotazu je následující:

```
SELECT nazev, stav, poznamka, zadana, prednasejici, zamereni,
    casKonani, casUkonceni, mistoKonani,
    titul, firsName, surname, email,
    nazevSk, organizace, url
FROM
    // dotaz na udalost
    {udalost} calendar:name {nazev},
    [{udalost} ex:hasStav {stav}],
    [{udalost} ex:hasPoznamka {poznamka}],
    [{udalost} ex:hasZadana {zadana}],
    [{udalost} ex:prednasejici {p} foaf:name {prednasejici}],
    [{udalost} ex:zamereni {z} ex:name {zamereni}],
    {udalost} calendar:startTime {casKonani},
    [{udalost} calendar:endTime {casUkonceni}],
    [{udalost} ex:misto_konani {place} place:name {mistoKonani}],
    // dotaz na uzivatele
    [{udalost} ex:zadal {uzivatel} foaf:title {titul}],
    [{udalost} ex:zadal {uzivatel} foaf:firstName {firsName}],
```

```
    [{udalost} ex:zadal {uzivatel} foaf:surname {surname}],
    [{udalost} ex:zadal {uzivatel} foaf:mbox {email}],
    // dotaz na skupinu
    {udalost} ex:zadal {uzivatel} ex:patri {skupina}
         ex:name {nazevSk},
    [{udalost} ex:zadal {uzivatel} ex:patri {skupina}
         ex:organizace {o} org:name {organizace}],
    [{udalost} ex:zadal {uzivatel} ex:patri {skupina}
         ex:organizace {o} org:weblog {url}]
WHERE  1=1
    AND nazev LIKE "*searchNazev*" IGNORE CASE
    AND zamereni LIKE "*searchZamereni*" IGNORE CASE
    AND nazevSk LIKE "*searchSkupinaNazev*" IGNORE CASE
USING NAMESPACE
    ex = <http://eva.fit.vutbr.cz/~xhavle02/DP/ontology/udalost.owl#>,
    foaf = <http://xmlns.com/foaf/0.1/Person#>,
    calendar = <http://www.topquadrant.com/topbraid/2007/01/calendar.owl#>,
    place = <http://dbpedia.org/ontology/Place#>,
    org = <http://xmlns.com/foaf/0.1/Organization#>
```

# 7 Testování systému

Cílem testování systému je kontrola správnosti jeho implementace, tedy jeho chování podle stanoveného očekávání. Jedná se o ověření, zda systém splňuje požadavky uvedené ve specifikaci, která je v rámci tohoto dokumentu obsažena v kapitole návrhu distribuovaného informačního systému, konkrétně podkapitole 4.1. Mnou implementovaný systém byl testován ve dvou fázích:

- Testování částí systému během jejich implementace
- Testování dokončeného systému

Testování systému v průběhu implementačních prací sloužilo pro otestování správnosti konkrétní části, kdy byla prostřednictvím jejího testování ověřována správná funkčnost bezprostředně po její implementaci. Toto bezprostřední testování je výhodné, z důvodu umožnění rychlé reakce na chyby implementace a jejich okamžitá korekce.

Poslední částí vývoje v rámci této diplomové práce vytvořeného distribuovaného informačního systému bylo otestování dokončeného informačního systému. Nejprve jsem otestoval veškerou implementovanou funkčnost související s lokální správou dat systému, konkrétně správou dat konferencí a seminářů, odběrů, skupiny, uživatelů a mechanismů přístupových oprávnění. Zahrnující testování správnosti uživatelských vstupů a výstupů včetně jejich korektního zanesení do databáze. Dále správnost omezení úkonů uživatele podle jeho přístupových práv do systému. Následně jsem prostřednictvím několika systémů otestoval funkčnost distribučních mechanismů, zahrnující získání a poskytnutí dat konferencí a seminářů jinému systému, jejich korektní zanesení do sémantického úložiště a následné zobrazení v systému. Při ověřování funkčnosti bylo v implementaci systému nalezeno pouze malé množství funkčních nedostatků, které byly úspěšně opraveny. Nízký počet chyb byl docílen testováním v rámci implementace jednotlivých částí popsaného výše. Systém tedy splňuje zadanou specifikaci.

Data, která jsem použil pro testování, jsou obsažena na CD, které je přílohou této práce. Na tomto CD jsou také veškeré informace spojené s prací se systémem, zahrnující jeho instalaci, systémové požadavky a krátký uživatelský manuál.

# 8 Závěr

Tato práce měla za cíl prozkoumat možnosti použití ontologií v prostředí informačních systémů. Konkrétně se zaměřila na použití ontologií v rámci distribuce dat mezi nimi. S tímto cílem souviselo seznámení se sémantickými technologiemi a následný návrh a implementace distribuovaného informačního systému s ohledem na využití sémantických technologií.

V rámci práce jsme se nejprve seznámili s pojmem ontologie, se způsobem návrhu ontologií a s prostředky pro jejich reprezentaci. V další části jsem se zabýval technologií sémantického webu s ohledem na výměnu informací v rámci informačních systémů. Soustředil jsem se zde především na jazyky RDF a OWL. Následuje část analýzy a návrhu samotného distribuovaného informačního systému umožňujícího zadávání, procházení a sdílení informací pomocí ontologií. Dále je popsán návrh ontologie *udalost.owl*, používané pro přenos dat v rámci distribuce mezi informačními systémy. Následná část práce se zabývá představením při implementaci použitých technologií a popisem implementovaného systému. Práce je zakončena popisem postupu testování vytvořeného distribuovaného informačního systému.

V rámci práce bylo mým vlastním přínosem seznámení se s novou technologií sémantického webu, pojmy a technologiemi s ní souvisejícími. Dále jsem si také obnovil a doplnil znalosti související s návrhem a implementací distribuovaných informačních systémů. Prakticky jsem si vyzkoušel využití sémantických technologií v rámci distribuce dat v informačních systémech. Práce s těmito technologiemi mě zaujala. Líbila se mi především, protože rozšířila mé vědomosti o znalosti nových technologií, které jsem v rámci práce začleňoval do pro mě již známé oblasti návrhu a implementace informačního systému.

Dalším možným rozvojem práce je nasazení vytvořeného distribuovaného systému do praxe. Protože informační systém vznikl především pro demonstrační potřeby, bylo by vhodné jeho funkčnost před vlastním nasazením do praxe zkonzultovat s budoucími uživateli a jejich připomínky zanést do systému. Jednalo by se zejména o upřesnění spravovaných a distribuovaných informací. Či o zautomatizovaní stahování distribucí například po konkrétních časových intervalech.

# Literatura


[1] Svátek, V.: *Ontologie a sémantický web* [online]. DATAKON, Brno, 2002 [cit. 2009-11-07]. Dostupné na URL: <http://nb.vse.cz/~svatek/onto-www.pdf>

[2] Hák, L.: *Sdílení dat mezi informačními systémy založené na ontologiích*, diplomová práce, Brno, FIT VUT v Brně, 2009, 68 s.

[3] Hanyáš, P.: *Sémantický web – tutoriál a demonstrační příklady*, bakalářská práce, Brno, FIT VUT v Brně, 2007, 41 s.

[4] Svátek, V., Vacura, M.: *Ontologické inženýrství* [online]. DATAKON, Brno, 2007 [cit. 2009-11-21]. Dostupné na URL: <http://nb.vse.cz/~svatek/dkon07final.pdf>

[5] Svátek, V., Labský, M.: *Objektové modely a znalostní ontologie* [online]. Katedra informačního a znalostního inženýrství, VŠE, Praha 2003 [cit. 2009-11-14]. Dostupné na URL: <http://nb.vse.cz/~svatek/obj03fi.pdf>

[6] Čížek, P.: *Úložiště znalostí*, diplomová práce, Brno, Fakulta informatiky, Masarykova univerzita, 2007, 50 s.

[7] Brickley, D., Miller,L.: *FOAF Vocabulary Specification 0.97* [online]. 2010 [cit. 2010-04-17]. Dostupné na URL: < http://xmlns.com/foaf/spec/>

[8] Chris Bizer: *DBpedia* [online]. 2010 [cit. 2010-05-07]. Dostupné na URL: <http://dbpedia.org>

[9] Stanford Center for Biomedical Informatics Research: *Protégé* [online]. 2010 [cit. 2010-05-07]. Dostupné na URL: <http://protege.stanford.edu/>

[10] ORACLE: *MySQL* [online]. 2010 [cit. 2010-04-19]. Dostupné na URL: <http://www.mysql.com>

[11] Sun Microsystems: *Model-View-Controller* [online]. 2002 [cit. 2010-04-20]. Dostupné na URL: <http://java.sun.com/blueprints/patterns/MVC-detailed.html>

[12] Shalk, Ch., Burns, E., Holmes, J.: *JavaServer Faces: The Complete Reference*. The McGraw-Hill Companies. United States of America. 2007. s. 864. ISBN 978-0-07-226240-7

[13] Aduna: *User Guide for Sesame 2.2* [online]. 2008 [cit. 2009-12-05]. Dostupné na URL: <http://www.openrdf.org/doc/sesame2/users/>

[14] SourceForge: *OWL API* [online]. 2010 [cit. 2010-03-29]. Dostupné na URL: <http://owlapi.sourceforge.net/documentation.html>

# Seznam příloh

# Příloha 1. Ontologie udalost.owl.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns="http://eva.fit.vutbr.cz/~xhavle02/DP/ontology/udalost.owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://eva.fit.vutbr.cz/~xhavle02/DP/ontology/udalost.owl">

  <!-- ============= Info ============= -->
  <owl:Ontology rdf:about="">
    <owl:versionInfo
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      r. 2010 v 1.0</owl:versionInfo>
    <rdfs:comment
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Vytvořeno v rámci diplomové práce na FIT.VUTBR.CZ </rdfs:comment>
  </owl:Ontology>

  <!-- ============= Class ============= -->
  <rdfs:Class rdf:ID="Uzivatel">
    <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  </rdfs:Class>

  <owl:Class rdf:ID="Skupina"/>

  <owl:Class rdf:ID="Udalost">
    <rdfs:subClassOf rdf:resource="http://www.topquadrant.com/
                        topbraid/2007/01/calendar.owl#Event"/>
  </rdfs:Class>

  <owl:Class rdf:ID="Okruh"/>

  <!-- ============= ObjectProperty ============= -->

  <!-- ============= ObjectProperty of Udalost ============= -->
  <owl:ObjectProperty rdf:ID="zadal">
    <rdfs:domain rdf:resource="#Udalost"/>
    <rdfs:range rdf:resource="#Uzivatel"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="prednasejici">
    <rdfs:domain rdf:resource="#Udalost"/>
    <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="zamereni">
    <rdfs:domain rdf:resource="#Udalost"/>
    <rdfs:range rdf:resource="#Okruh"/>
  </owl:ObjectProperty>
```

```
  <owl:ObjectProperty rdf:ID="misto_konani">
    <rdfs:domain rdf:resource="#Udalost"/>
    <rdfs:range rdf:resource="http://dbpedia.org/ontology/Place"/>
  </owl:ObjectProperty>

  <!-- ============= ObjectProperty of Skupina ============= -->
  <owl:ObjectProperty rdf:ID="organizace">
    <rdfs:domain rdf:resource="#Skupina"/>
    <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Organization"/>
  </owl:ObjectProperty>


  <!-- ============= ObjectProperty of Uzivatel ============= -->
  <owl:ObjectProperty rdf:ID="patri">
    <rdfs:domain rdf:resource="#Uzivatel"/>
    <rdfs:range rdf:resource="#Skupina"/>
  </owl:ObjectProperty>


  <!-- ============= DatatypeProperty ============= -->

  <!-- ============= DatatypeProperty Udalost ============= -->
  <owl:DatatypeProperty rdf:ID="hasStav">
    <rdfs:domain rdf:resource="#Udalost"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasPoznamka">
    <rdfs:domain rdf:resource="#Udalost"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasZadana">
    <rdfs:domain rdf:resource="#Udalost"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
  </owl:DatatypeProperty>

</rdf:RDF>
```

# Příloha 2. CD se zdrojovými soubory.

Obsah přiloženého CD:

- *README.txt* - Soubor s informacemi o obsahu přiloženého cd.
- *DP_Havlena_Jan.pdf* – PDF soubor s textem diplomové práce.
- *DP_Havlena_Jan.doc* - Zdrojový soubor s textem diplomové práce.
- *VSSKUPINY* – Složka s projektem vytvořeného distribuovaného informačního systému.
- *VSSKUPINY.war* – WAR soubor vytvořeného distribuovaného informačního systému.
- *udalost.owl* - Soubor s navrženou ontologií.
- *INSTALL.txt* - Soubor s informacemi o vytvořeném informačním systému a jeho instalaci.
- *create_db.sql* - Soubor se skriptem vytvářejícím databázi s tabulkami pro systém.
- *add_skupina.sql* - Soubor se skriptem pro přidání nové skupiny s uživatelem mající práva administrace této skupiny.
- *test_data.sql* - Soubor pro naplnění databáze systému testovacími daty.
- *distributed_data.owl* - Soubor s příkladem distribuovaných owl dat mezi informačními systémy.
- *manual.pdf* – Soubor s manuálem vytvořeného informačního systému.