СОФИЙСКИ УНИВЕРСИТЕТ

"СВ. КЛИМЕНТ ОХРИДСКИ"

Факултет по Математика и Информатика

Катедра „Информационни Технологии"

ДИПЛОМНА РАБОТА

на тема:

Уеб приложение за създаване на страници с метаданни за Семантична мрежа

Дипломант: Румяна Ганчева Ангелова, фак. № М-21362

Специалност: Информатика

Специализация: Разпределени системи и мобилни технологии

Научен ръководител: доц. Силвия Илиева

Рецензент: доц. Боян Бончев

София, 2007 г.

Съдържание

1. Увод

World Wide Web (WWW) в момента наподобява слабо картографирана география от огромна маса данни. В нея успяваме да намерим необходимата информация чрез търсене по ключови думи. Но данните са толкова много, че са управляеми само чрез програмни агенти и за да се „картографират" прецизно е необходимо да се създаде машинно-разбираемо описание на съдържанието и възможностите на Web ресурсите. Такива мета-данни трябва да са допълнение към съответните версии на документите, предназначени за хора. По такъв начин се създава надграждащо ниво на интерпретация, което представя семантиката на данните.

Следващата генерация на WWW (Web 2.0) се нарича Семантична мрежа (Web 3.0). Тя включва това разширение на текущата WWW, в което на информацията е предадено добре дефинирано значение. Това се осъществава като към текущите WWW ресурси се добавя семантично съдържание, което се интерпретира машинно. Идеята е да се позволи по-ефективно търсене и откриване, автоматизация на процесите за обработка на намерените данни (например правене на логически изводи, което води до откриване на нови факти), както и интеграция на смисъла на данните от различни приложения. Така програмни агенти могат да действат изцяло от името на човека като се справят със задачи, за които досега е било възможно да бъдат изпълнени само от него (например човека може да направи изводи и от частична информация, понеже я асоциира с факти, които вече са му известни; използва сетивата си, за да възприема информацията; комбинира информацията от различни източници без затруднение дори и да е използвана различна терминология и т.н.).

1.1. Цел на дипломната работа

Целта на дипломната работа е да подпомогне процесът за създаване на уеб страници със семантично съдържание. Тази димпломна работа разглежда реализацията на уеб приложение, което да предоставя удобен графичен интерфейс във вид на лесен за използване уеб редактор. Той трябва да скрива от потребителя сложността на технологиите за семантични мрежи. Използвайки този редактор потребителят ще може да създава уеб страници с текст, графика и съответната семантика към тях без да е необходимо да е специалист в областта на семантичните мрежи.

1.2. Задачи произтичащи от целта

1. Обзор на същността и основните понятия в семантичната мрежа; приложенията на семантичната мрежа и реализации на приложения за Семантична мрежа.
2. Описание на технологиите, които се използват в реализацията на уеб редактора за създаване на уеб страници със семантично съдържание (технологии за семантична мрежа и технологии от Java EE5.0 за разработване на уеб приложения).
3. Реализация на уеб приложението – изисквания, архитектура, дизайн, кодиране и тестване.
4. Предложение за бъдещи насоки за развитие на приложението.

1.3. Практическа полза от реализацията

Съществуват два подхода за налагането на Семантичната мрежа в световното WWW пространство :

- чрез адаптиране на наличната информация към стандартите за семантична мрежа като се прави конвертиране или създаване и асоцииране на мета-данни към текущата информация;
- чрез добавяне на нови данни, които при създаването си съдържат мета-данни.

Тази дипломна работа описва създаването на уеб приложение, което следва втория подход. Синтаксисът на езиците за семантична

мрежа не е тривиален и затова е необходимо да се създават редактори, които да скриват сложността му от потребителите. Много водещи софтуерни компании като Adobe, IBM и други, са разработили или са в процес на разработка на тази идея в редакторите, които предлагат. Повечето от тях използват собствени онтологии с понятия.

Настоящето уеб приложение представлява редактор, който използва различни публикувани във WWW онтологии. Те се достъпват чрез уеб услуги. Това е важна характерна особеност, защото е препоръчително съществуващите онтологии да се използват многократно. Проблемът не е само в загубата на време и ресурси за създаване на онтология, която вече е дефинирана, но и поради това, че се губи връзката между понятията, когато не се използват общоприети онтологии.

Мета-данните, които се създават чрез приложението, се записват в база данни и чрез реализация на стандартизирания интерфейс SPARQL могат да се използват от други приложения, които използват протоколите за Семантична мрежа. Освен това от страниците, които се генерират от приложението, заедно с техните мета-данни се създават XHTML документи с RDFa съдържание, така че да могат да се използват и от текущите агенти, които работят с XHTML.

Друга полезна черта на редактора е, че той е уеб, а не самостоятелно приложение, като по такъв начин е лесно-достъпен за широк кръг от потребители и не е необходимо да се инсталира на системите им.

1.4. Структура на дипломната работа

Глава 2 представя обзор на текущото състояние на Семантичната мрежа, която е разширение на текущата WWW, както и бъдещите тенденции за нейното развитие. Точка 2.1 описва предпоставките за възникването на Семантичната мрежа. Точка 2.2 представя детайлно Семантичната мрежа. В подточка 2.2.1 се разглежда нейната история. Технологиите и спецификациите, които я дефинират са представени в

хронологичен ред, за да се проследи нейното развитие. Подточка 2.2.2 представя технологичния стек на Семантичната мрежа. Там тя е показана такава, каквато трябва да изглежда в завършен вид, а не само текущите нива, които вече са реализирани. Подточка 2.2.3 представя в синтезиран вид начините за съхранение на мета-данните. В точка 2.3 са изброени някои от най-известните приложения за Семантична мрежа в няколко различни насоки - среди за разработка, редактори, системи за управление на съдържание, системи за съхраняване на RDF тройки.

Глава 3 прави преглед на всички технологии, използвани при разработката на уеб редактора.

Точка 3.1 представя основните характеристики на RDF. В подточка 3.1.1 се разглежда понятието уеб ресурс в контекста на Семантичната мрежа, както и идентификацията на тези ресурси спрямо RDF. Подточка 3.1.2 се описва модела на данните в Семантичната мрежа, а именно наредените RDF тройки и тяхното графично представяне. В подточка 3.1.3 се разглежда RDF синтаксиса, който се базира на XML. Описани са основните езикови конструкции на RDF и са разгледани пояснящи примери. Представени са и някои конвенции в описанието на данни чрез RDF/XML синтаксис.

Точка 3.2 представя основните характеристики и примери на RDFa, който е част от XHTML. RDFa е създаден, за да се позволи влагане на RDF съдържание в XHTML.

В подточка 3.3.1 е представен езика OWL и неговите подезици, които предоставят възможност за описание на по-сложни онтологии и техните връзки. Този език е по-изразителен от RDF-S. В подточка 3.3.2 са разгледани основните му езикови конструкции (класове, инстанции и свойства), представени чрез примери.

Точка 3.4 запознава читателя с характеристиките на Java EE 5.0 платформата за разработване на уеб приложения. В подточка 3.4.1 са представени основните характеристики на JSF технологията, която представлява стандартизирана рамка за UI частта на уеб приложенията. В подточка 3.4.2 се разглеждат спецификациите за уеб услуги, които се

предоставят от Java EE 5.0 платформата. Представени са основните механизми за работа с уеб услуги спрямо тези спецификации. В подточка 3.4.3 са разгледани основните характеристики на новата технология в Java EE 5.0 за връзка с база данни, а именно Java Persistence API, която изчиства идеята за обектно-релационното съпоставяне и прави разработката на java persistent частта от приложенията много по-приятна.

Глава 4 разглежда всички етапи от реализацията на приложението. Започва се с изискванията в точка 4.1. След като те са добре дефинирани, в точка 4.2 се разглежда основната архитектура на приложението на компонентно ниво, която се базира на Java EE 5.0 платформата. В подточка 4.2.1 е представен модел Model-2 MVC, на чиято основа се гради Java EE 5.0. Подточка 4.2.2 разглежда всяка от основните концепции на архитектурата на приложението – изгледът, контролерът и моделът. Посочени са също така компонентите и съответните технологии в Java EE 5.0 към всяка от тези три части. В точка 4.3 е разгледан дизайна на приложението. Представени са по-важните java класове и файлове и тяхното взаимодействие. В подточка 4.3.1 са изброени всички JSF файлове и техни особености. Подточка 4.3.2 разглежда всички managed beans, техните атрибути и методи. В подточка 4.3.3 са описани session EJBs, както и как те използват уеб услугите и Java Persistence API. В точка 4.4 са представени средите за разработка, които са използвани за реализацията на приложението. В точка 4.5 са описани подходите за тестване на готовото приложение.

Глава 5 прави обобщение на дипломната работа и се предлагат някои насоки за бъдещо развитие на приложението.

2. Обзор на проблемната област и теоретична обосновка на предлаганото решение

2.1. Възникване на необходимостта от Семантична мрежа

Текущата WWW мрежа представя информацията чрез текст на естествен език (като български, английски, френски и т.н.), графики и мултимедия. Човекът лесно може да я преглежда, да реши кои ключови думи са подходящи и да търси по тях. Той може да прави логически изводи дори и от частична информация, понеже я асоциира с друга, която вече му е известна. Той разбира смисъла на текстовете и графиките като ги види или като чуе мултимедийното приложение. Няма значение каква технология е използвана, за да се извлече и представи информацията на човека – накрая тя винаги е във вид, който се разбира от него. Така той лесно комбинира данните от различни приложения.

Но машините не притежават сетивата и интелекта на хората. Те не могат да разберат смисъла на текст или картинка. Частичната информация за тях е недостатъчна, за да направят цялостен извод. Те не могат да направят асоциации спрямо даден текст или графика, освен ако не е предоставен механизъм за това. Не могат и лесно да комбинират данни от различни приложения, които използват различни технологии например. За всеки две различни технологии трябва да се направи интегриращо приложение, за да е възможно данните да се съвместяват.

Решението е към данните във WWW да се прикрепят мета-данни, които да могат да се обработват от машините. Тези мета-данни трябва да представят информацията еднозначно. Описанието на мета-данните трябва да е достатъчно гъвкаво, за да могат те лесно да се комбинират и свързват. Така е възможно да се конструират логически изводи от съществуващи факти, което ще направи машините мислещи.

2.2. Същност и основни понятия в Семантичната мрежа

2.2.1. Семантична мрежа

В следствие на нарастналата необходимост от създаването на семантичното ниво на WWW, Internet Engineering Task Force и World Wide Web Consortium (W3C) насочват усилията си към специфициране, разработване и внедряване на езици за споделяне на значение. Тези езици осигуряват основата за семантично взаимодействие на приложенията.

През 1997 г. W3C дефинира първата спецификация за Resource Description Framework (RDF). В нея RDF е прост, но мощен изразителен език, базиран на представяне на данните като наредени тройки <субект–предикат-обект> и наименувани чрез Universal Resource Identifiers (URIs). Най-важната черта на тази спецификация е, че описва универсален модел на данните във WWW чрез XML синтаксис. Всяко нещо, което се описва, се представя като уеб ресурс с URI име. През 1999 г. RDF става W3C recommendation, т.е. намира се в последна фаза преди да излезе като стандарт. В този момент се насочва вниманието към спецификацията и се акцентира на това, че нейното широко внедряване ще обогати функционалността и взаимодействието в WWW. RDF поставя началото на представянето на знание в WWW.

През февруари 2004 г. се появява нов W3C recommendation - RDF Schema (RDFS). RDF се ограничава само до представяне на структурата на наредените тройки, които описват уеб ресурсите. Множество от такива дефиниции на ресурси, които описват дадена концепция или предмет, се означава като речник или още наричано онтология. RDFS се базира на RDF спецификацията като добавя възможност за изразяване на структурирани речници, т.е. RDFS се явява минимален език за представяне на онтологии.

След като RDF и RDFS стават все по-популярни сред бизнес компаниите и научните среди, създава се все повече RDF съдържание и се появява необходимост от стандартизиране на достъпа до системите,

които съхраняват тези данни. Езикът, който е разработен, за да изпълни тези изисквания, се нарича SPARQL. В момента той е в последна фаза преди да се обяви като W3C recommendation.

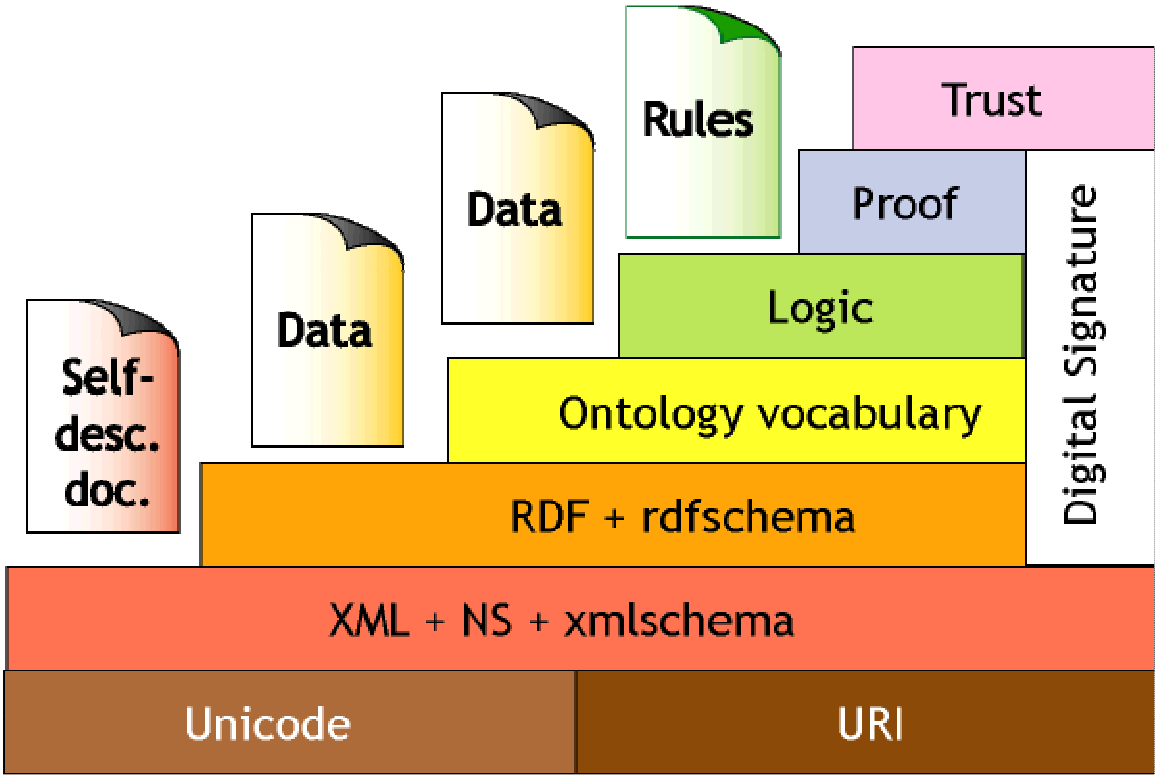
Web Ontology Language (OWL) е следващото разширение, което предоставя по-голяма изразителност на дефинициите от RDFS. От 2004 г. той също вече е W3C recommendation. В спецификацията се представят три версии на OWL, които имат различна степен на изразителна сила и разработчиците могат да избират най-подходящата версия за техните нужди. Основната идея на OWL е да предостави ефективен начин за представяне на онтологии, който да бъде използван от системи, правещи логически изводи. Чрез OWL те могат да определят дали в една онтология няма противоречия, както и дали дадена концепция попада в нея. OWL позволява онтологиите да използват термини от други онтологии и по такъв начин ги прави разпределени.

Работи се в момента и върху включването на мета-данни в самите XHTML документи, така че да не се дублира една и съща информация веднъж представена за хората, и втори път за машините. Спецификацията RDFa, която сега е в последна фаза преди да стане W3C recommendation, е разширение на XHTML, като обобщава атрибутите meta и link, за да се добавя семантика към всички елементи на XHTML документите. Когато се използва RDFa става много лесно извличането на RDF информация от XHTML документи.

Друга технология за Семантичната мрежа, която все още е в начална фаза, се разработва от ноември 2005г. от групата Rule Interchage Format Working Group отново към организацията W3C. Целта на тази група е да специфицира формат (RIF) за правила върху данните, представени чрез RDF и OWL. Този формат може да се използва като общ език за комуникация между различни логически системи. Ако всички логически езици се съпоставят на този нов формат, то правилата написани за едно приложение ще могат да се публикуват, споделят и повторно да се използват от други приложения и логически системи.

2.2.2. Технологичен стек на Семантичната мрежа

Фигура 1 илюстрира технологичния стек на Семантичната мрежа:



Фигура 1. Технологичен стек на Семантичната мрежа

В основата на стека стои концепцията за идентифициране на всички ресурси в Семантичната мрежа чрез URIs, които съдържат Unicode символи. Това позволява да се изразяват URIs с различни езици.

Следващото технологично ниво в Семантичната мрежа са XML спецификациите. Те позволяват на всеки да проектира свой документен формат (xmlschema) и след това да създава документ по този формат. Форматът е предназначен да се чете и разбира от програмите. Като се включва такава възможност, документите стават много по-полезни, понеже се увеличават начините за обработката им. Но с широката употреба на XML идва необходимостта от идентифициране на елементите в XML формата. Затова се включва идеята за XML namespaces, които

също се базират на URIs. Така отпада проблемът с конфликта между имената на XML таговете.

Дотук няма разлика с технологиите, използвани в сегашната WWW. От третото ниво в стека нагоре идват специфичните за Семантичната мрежа технологии. RDF предлага общ модел на всички данни и така те вече могат да се обработват по еднакъв начин. Разликата с предишното ниво е, че с XML технологията данните се представят само в рамките на съответните схеми. В RDF мотото е „всеки може да каже всичко за всичко". Това изглежда като неточно дефиниране на данните, но всъщност е продиктувано от факта, че информацията е разпръсната в мрежата и самите хора имат различно виждане за едни и същи неща. Това е свободата, която мрежата от хора и данни предоставя. RDF се структурира повече чрез схемата на езика RDF Schema.

Следващото технологично ниво в стека на Семантичната мрежа са онтологиите и OWL. Те дефинират значението и връзките между термините, които са описани с помощта на RDF, така че цялата тази маса от информация става по-компактна.

Петото ниво на Семантичната мрежа включва програмите, които могат да правят логически изводи на базата на информацията, описана чрез предишните четири технологии. При публикуване на резултатите от логическите изводи, информацията в мрежата се обогатява. Основната спецификация на това ниво е Rule Interchange Format (RIF).

Следващото ниво са програмите, които на базата на логическите изводи, направени с помощта на предишните нива, могат да правят доказателства.

Последното ниво на Семантичната мрежа е свързано с доверието във верността на информацията. Ако към твърденията в мрежата са прикрепени цифрови подписи, то може да кажем на програмата, която обработва информацията, на кои подписи да вярва и кои да пропуска. Концепцията е подобна като мисленето на човека – за него е важна само информацията, за която знае, че е вярна. На това ниво се появява и т.нар. „мрежа от доверие", така както един човек може да вярва на

няколко човека, те пък от своя страна да вярват на други хора и т.н. Основна идея на тази технология е да дефинира това доверие при уеб ресурсите.

Последните две технологични нива все още се дискутират само на теория. Тяхната реализация предстои в бъдещето, когато ще е създадена стабилна основа на Семантичната мрежа, върху която да може да се надгражда.

2.2.3. Съхранение на мета-данните в семантичната мрежа

Съществуват три основни начина на разположението на тази информация от мета-данни :

- в специални уеб достъпни бази данни и тези бази данни предоставят стандартизиран интерфейс, за да могат външни приложения да използват мета съдържанието;
- в XHTML (Extensible HTML) документи, които комбинират HTML презентационното съдържание с мета данни, описани чрез XML;
- в обикновени XML документи, а презентационното съдържание може да бъде съхранено отделно в други файлове.

Тази дипломна работа представя приложение, което съхранява мета-данни по първите два начина, защото те показват повече детайли от Семантичната мрежа. Бъдещо развитие на приложението може да включи и третия начин за съхранение.

2.3. Разработки за Семантичната мрежа

В тази глава ще бъдат представени някои от по-известните разработки, които допринасят за изграждането на Семантичната мрежа. Не е възможно да се посочат всички, понеже вече проектите са хиляди и продължават да нарастват в най-разнообразни области.

В областта на средите за разработка, редакторите и системите за управление на съдържание за Семантичната мрежа водещи са следните

продукти: XMP (Adobe), SemanticWorks 2006 (Altova), Amilcare (University of Sheffield), Cerebra Server, Cypher, DERI Ontology Management Environment, Graphl, GrOWL, IODT (IBM), Web Ontology Manager (IBM), Semantic Layered Research Platform (IBM), RDF InferEd (Intellidimension), IsaViz, LinKFactory (Language & Computing), M3t4.Studio Semantic Toolkit (Metatomix), Model Futures OWL Editor, Data Spaces Platform (Open Link), OWL verbalizer, pOWL, Semantic Information Router (Profium), RDFe, Semantic Web Client, Seamark Navigator (Siderean), Enterprise Information Integrator (Software AG), Protege (University of Stanford), SWOOP (University of Maryland), Experiment Design Automation (Teranode), Thetus Publisher, TopBraid Composer (Top Quandrant), VisualKii, Enterprise Information Integration (@Semantics).

По-известни от системите за съхраняване на RDF тройки са: Aduna Metadata Server, Boca, D2RQ & D2R Server, AllegroGraph (Franz Inc), RDF Gateway (Intellidimension), Joseki (Jena), Kowari Metastore, Mulgara Semantic Store, Virtuoso (OpenLink), Oracle Spatial 10g, OWLIM, RDFStore, RDF server (RAP), SemWeb for .NET, Sesame, Tucana Suite (Northrop Grumman), YARS/Yet Another RDF Store, 3Store.

3. Преглед на използваните технологии

3.1. RDF

RDF е език за представяне на информация за ресурси в WWW. По-точно основното му предназначение е представяне на мета-данни за тези ресурси. Целите, които си поставят авторите на RDF, са да дефинират:

- обща идентификация на всички ресурси;
- общ модел на данните;
- XML-базиран синтаксис;
- поддръжка на XML Schema типове данни;
- да се позволи всеки да може да прави твърдения за всичко.

3.1.1. Идентификация на ресурсите в RDF

В спецификацията понятието уеб ресурс се обобщава като вече показва не само ресурси, които пряко могат да се достъпят в WWW чрез някой от наличните протоколи, но и такива, за които има информация, но няма директен достъп до тях. Например една уеб страница може да описва даден предмет и тогава според RDF той също е уеб ресурс, а не само HTML документа, който го съдържа. По такъв начин може да се опише цялото съдържание на WWW и тогава то да стане достъпно и разбираемо за машините, а не само за хората, който го преглеждат. RDF предоставя обща рамка за представяне на тази информация, която е независима от технологиите, използвани в уеб приложенията. Така чрез рамката различните приложения могат да обменят помежду си мета-данни без загуба на значение.

За обозначаване на самите ресурси се използват уеб идентификаторите Uniform Resource Identifiers (URIs), които в спецификацията се наричат RDF URI references, заради допълнителните

ограничения, които налага RDF върху URI. Разбира се, всяка друга концепция за идентификатори може да бъде приложена, но URI спецификацията се избира от авторите на RDF, заради това, че тя вече е в основата на досегашната WWW и се използва за съществуващите ресурси. Тъй като Семантичната мрежа е разширение на WWW, то тя продължава същия подход за именуване, но вече приложен и за новите уеб ресурси, които представят мета-данни. Поради това, че WWW е твърде огромна, за да се контролира от която и да е организация, URIs са децентрализирани. Това означава, че не може да се контролира кой ги създава и как се използват. Някои URI схеми, (като http:) зависят от централизирани системи (DNS), но другите схеми са напълно децентрализирани. Така не е нужно разрешение, за да се създава URI. Но наред с факта, че по този начин URI е достатъчно гъвкаво средство за идентификация, създават се и някои проблеми. Заради това, че всеки може да създава URI, неминуемо се стига да ситуацията, че множество различни URIs представят едно и също нещо.

RDF разглежда множество от URIs, описващи даден домейн, като речник. Веднъж описани с URIs в речници, ресурсите могат да се използват многократно, стига да може да се откриват тези речници. Например един потребител може да търси в WWW коментари за всички книги и да създаде среден рейтинг за всяка книга. Тогава той може да публикува информацията с всички рейтинги отново в WWW. Друга уеб страница може да вземе този списък от рейтинги за книги и да създаде страница „класация на книгите по рейтинг”. Това е възможно, ако се използва общ споделен речник за рейтинг, както и споделена група от URIs, идентифициращи книгите. Така се създава взаимно-разбираема и по-мощна (понеже всяко приложение може да дава допълнителни детайли) информационна база за книгите в WWW. Същият принцип важи за огромното количество информация, която хората създават всеки ден за хиляди субекти.

Така не само се спестява труд по създаването на речници, но и се решава идентификационния проблем, при който за едни и същи понятия

се ползват различни термини от различни речници. Дори и ако се описват синоними, фактът, че съответните URIs се използват в общо-достъпното WWW пространство, дава възможност да се дефинира еквивалентност между тези различни URIs. Многократното използване на речници позволява да се описват неща, които отразяват общо разбиране на дадени концепции.

Известна практика за създаване на URIs е да се започне с уеб страница, която описва обекти, на които трябва да се даде семантика. URIs на тези обекти използват URL на страницата като основа, но разработчикът решава какви точно са имената спрямо структурата, която иска да изрази. Важно е да не се забравя, че RDF URI идентифицира ресурс, а не начина на достъп до този ресурс.

3.1.2. Модел на данните в RDF

RDF е предназначен за описание на твърдения за уеб ресурси. За целта RDF дефинира прост модел на данните, при който твърденията се представят като релации между две неща във вид на наредена тройка <субект><предикат><обект>. Субектът и обектът са двете неща в твърдението, а предикатът е релацията между тях. Често предикатът е някакво свойство на субекта, а обекта е стойността на това свойство за дадения субект. Например в твърдението „Колата има червен цвят" субектът е „колата", предикатът е „цвят", а обектът е „червен". Обектът от своя страна може да бъде субект за друго твърдение. Затова за представянето на множество твърдения е удобно да се използва граф. Върховете на графа са субекти или обекти, а дъгите са предикати. Субектите и обектите се означават задължително с RDF URIs, а предикатите могат да са RDF URIs или литерали.

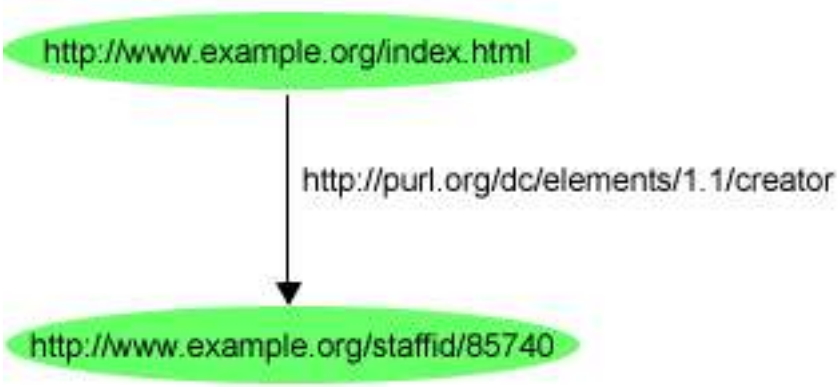
Нека да разгледаме как се описва чрез този модел твърдението *"**http://www.example.org/index.html** has a **creator** whose value is **John Smith**".*

Това твърдение може да се представи чрез следната RDF тройка:

- субект *http://www.example.org/index.html*;
- предикат *http://purl.org/dc/elements/1.1/creator*;
- обект *http://www.example.org/staffid/85740*.

Не е необходимо да се създава ново URI за субекта, понеже той вече идентифицира наличен уеб ресурс. Но за предиката и обекта трябва да се посочат подходящи URIs. В случая URI на предиката се взима от публичната онтология "Dublin Core", която предлага в RDF формат термини за описание на документни обекти (дефинирани са свойствата заглавие, автор, тема, издател и т.н.). За примера е използван URI на термина автор (creator) от тази онтология. Обектът в тройката посочва човека John Smith, който работи в компанията example и има служебен номер 85740. Затова неговото URI е http://www.example.org/staffid/85740. Разбира се всякакви други URIs могат да се измислят, но е задължение на разработчика на RDF съдържание да придава структура на данните.

Визуално графът за това твърдение е представен с Фигура 2:



Фигура 2. Графично представяне на RDF тройка

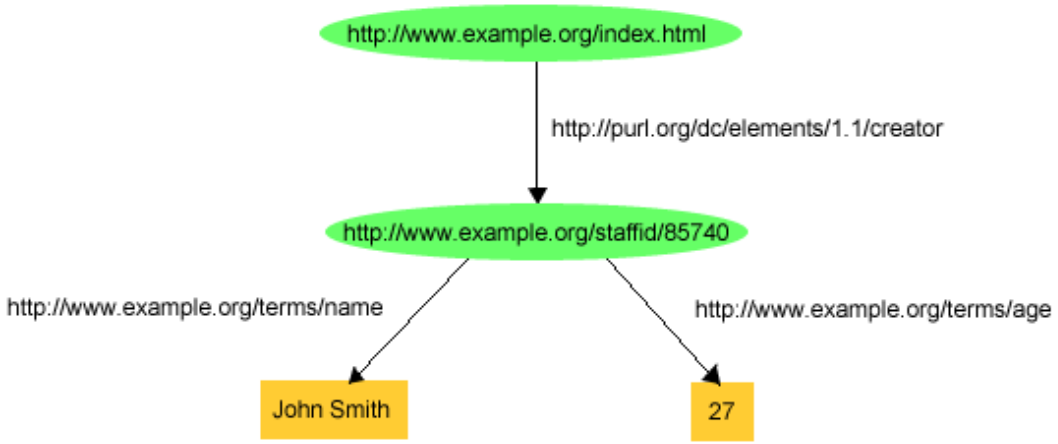
Обектът на една RDF тройка може да се явява субект за друга тройка. Така можем да комбинираме тройките от прости твърдения в по-сложни. В такива случаи обекта не може да бъде изразен като литерал, а само с URI. Например твърдението

*"**http://www.example.org/index.html** has a **creator** whose value is **John Smith. He** is **27 years old**".*

се представя чрез следните три RDF тройки:

- ***http://www.example.org/index.html*** *has a* ***creator*** *whose value is* ***worker 85740.***
  - субект *http://www.example.org/index.html*;
  - предикат *http://purl.org/dc/elements/1.1/creator*;
  - обект *http://www.example.org/staffid/85740*.
- ***Worker 85740*** *has a* ***name*** *whose value is* ***John Smith.***
  - субект *http://www.example.org/staffid/85740*;
  - предикат *http://www.example.org/terms/name*;
  - обект *John Smith*.
- ***Worker 85740*** *has an* ***age*** *whose value is* ***27.***
  - субект *http://www.example.org/staffid/85740*;
  - предикат *http://www.example.org/terms/age*;
  - обект *27*.

Графът на тези RDF тройки е показан на Фигура 3:



Фигура 3. Графично представяне на три свързани RDF тройки

Тук обектите *John Smith* и *27* са обикновени литерали. За предикатите *"name"* и *"age"* са използвани URIs, дефинирани като част от онтологията *http://www.example.org/terms*.

Предикатите в RDF тройките са означават също с URIs, а не с обикновени литерали. Това е важно поради следните причини. Първо, така се различават свойствата, които едно нещо може да има, от

свойствата, които друго нещо може да има, но тези свойства имат еднакви наименования. Например „име" може да е име на човек, но може и да е име на променлива в код. Ако работим със свойствата, които имат съответните URIs

http://www.example.org/terms/name

http://www.domain2.example.org/programming/terms/name

виждаме, че става въпрос за различни свойства с едно и също име.

Освен това, като се използват URI references за идентифициране на свойства, се позволява предикатите да бъдат третирани като ресурси, по същият начин като субектите и обектите. След като свойствата са уеб ресурси, допълнителна информация може да бъде въведена и за тях, което прави езика още по-описателен. Когато се дава допълнителна информация за предикатите, тогава техните URI references се използват като субекти в RDF тройки.

3.1.3. XML-базиран синтаксис в RDF

3.1.3.1. QNames

За да се кодира графа с XML е необходимо върховете и предикатите да се представят с термините в XML – елементи и атрибути. RDF/XML използва XML QNames, за да представи RDF URI references. Всички QNames имат базово namespace име, което е URI reference и късо локално име. Тези QNames може да съдържат префикс, който е асоцииран с namespace URI, следван от : и локалното име. Някои от по-известните QName префикси , използвани в RDF, са:

префикс	**Namespace**
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
dc:	http://purl.org/dc/elements/1.1/
owl:	http://www.w3.org/2002/07/owl#
xsd:	http://www.w3.org/2001/XMLSchema#

Това се прави, за да се скъси RDF URI references на имената на уеб ресурсите. RDF URI references, идентифициращи субекти и предикати, могат да бъдат представени и като XML атрибутни стойности. RDF литерали, които могат да са само върхове-обекти, получават или текстово съдържание на XML елемент или XML атрибутна стойност.

Както вече беше споменато множество от URI references, описващи даден домейн, се разглежда като речник. Всеки URI reference в речника определя термин от този речник. Обикновено URI references в речника са организирани по такъв начин, че да имат общ префикс. Всеки термин от речника се идентифицира с общия префикс, : и локално име, което е уникално за речника. Например организация като example.org може да дефинира два речника. Единият може да е за термини като „creation-date" и „product", които ползва в бизнеса си. Тогава този речник се състои от URI references, започващи с префикса http://www.example.org/terms/. Другият речник може да е за термини, идентифициращи работниците, и да се състои от URI references, започващи с http://www.example.org/staffid/.

Самият RDF използва същият подход, за да дефинира свой речник от термини със специално значение. URI references в този RDF речник започват с http://www.w3.org/1999/02/22-rdf-syntax-ns#, като по конвенция този URI reference се асоциира с QName префикс rdf:. RDF Vocabulary Description Language (RDFS) дефинира допълнително множество от термини, имащи URI references, които започват с http://www.w3.org/2000/01/rdf-schema# и които конвенционално се асоциират с QName префикс rdfs:.

Използването на общи URI префикси осигурява удобен начин за организиране на URI references за свързано множество от термини. Но това е само конвенция. RDF моделът изисква само URI references, но той не се интересува от тяхната структура. В частност RDF не предполага, че може да има връзка между някакви URI references, само защото имат еднакъв префикс. В този смисъл URI references с различни префикси

могат да са част от един речник. Дадена организация може да дефинира речник, използвайки URI references, от които и да са други речници.

Понякога даден URI reference на речник може да се използва като URL на уеб ресурс, който осигурява повече информация за самия речник. Например QName префикса dc: се асоциира обикновено с URI reference http://purl.org/dc/elements/1.1/. Всъщност той се отнася до речника Dublin Core. Достъпвайки този URI reference през Web браузер, се получава допълнителна информация за този речник (по-точно това е една RDF schema). Но това е само конвенция. RDF не предполага, че всяко URI, идентифициращо уеб ресурс, е обект, до който имаме директен достъп.

3.1.3.2. RDF/XML

Представянето на твърдението

http://www.example.org/index.html *has a* ***creation-date*** *whose value is* ***August 16, 1999***

като RDF тройка е

ex:index.html exterms:creation-date "August 16, 1999"

А в RDF/XML формат то се представя така:

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
                xmns:exterms="http://www.example.org/terms/">
3.   <rdf:Description rdf:about="http://www.example.org/index.html">
4.     <exterms:creation-date> August 16, 1999 </exterms:creation-date>
5.   </rdf:Description>
6. </rdf:RDF>
```

Ред 1 <?xml version="1.0"?> е XML декларация, която показва, че по-нататъшното съдържание е XML, а също и каква версия на XML се използва.

С ред 2 започва rdf:RDF елемент. Той показва, че следващото го XML съдържание, започващо тук и завършващо с </rdf:RDF> на ред 6, е предназначено да представя RDF данни. След rdf:RDF е XML namespace декларация, представена с xmlns атрибут на rdf:RDF започващият таг. Тази декларация специфицира, че всички тагове в това съдържание, започващи с rdf: са част от namespace, идентифицирано с URI reference http://www.w3.org/1999/02/22-rdf-syntax-ns#. URI references, започващи с http://www.w3.org/1999/02/22-rdf-syntax-ns#, се използват за термини от RDF речника. След това има друга XML namespace декларация, този път за префикс exterms:. Това е изразено с друг xmlns атрибут на rdf:RDF елемента и специфицира, че URI reference http://www.example.org/terms/ е асоцииран с префикса exterms:. URI references, започващи с http://www.example.org/terms /, се използват за термини от речника, дефиниран от организацията examples. ">" на края на ред 2 показва краят на rdf:RDF започващия таг. Редове 1-2 са задължителни, за да покажат че това е RDF/XML съдържание, както и да посочат namespaces, които се използват в RDF/XML съдържанието.

Редове 3-5 представят RDF/XML вида на даденото твърдение. RDF/XML твърдението има част description, с която показва, че описва някакъв субект, а той се идентифицира с частта about (в този пример about е http://www.example.org/index.html). Започващият таг rdf:Description на ред 3 показва началото на описанието на ресурс и продължава с идентифициране на ресурса чрез атрибута rdf:about, чиято стойност е URI reference на субекта. Ред 4 представя елемента-предикат с таг QName exterms:creation-date. Пълният вид на този URI reference е http://www.example.org/terms/creation-date. Съдържанието на елемента-предикат е вложено в съдържащия го елемент rdf:Description. Той показва, че този предикат се отнася до ресурса, специфициран с rdf:about атрибута на същия този rdf:Description елемент. Ред 5 представя края на rdf:Description елемента.

Ред 6 посочва краят на rdf:RDF елемента, започнат на ред 2.

3.2. RDFa

Много удобно би било, ако в HTML уеб страниците мета-данните се поместят наред със самите данни, за да не се получава дублиране. Освен това се появява нова функционалност като например копиране на структурирана информация между приложения и уеб страници. Когато мета-данните са налични на уеб страницата и еднозначно дефинират съответната информация, може събитие, описано на тази страница да се добави в календара на Microsoft Outlook само с копиране. Друг пример е уеб страница за снимки. Ако към снимките има мета-данни, които дават допълнителни детайли, то търсенето значително се улеснява.

RDFa е синтаксис за изразяване на такива мета-данни, използвайки множество от елементи и атрибути, които влагат RDF в XHTML. Важна цел на RDFa е да постигне влагане на RDF без повтаране на съществуващо XHTML съдържание, когато то е същото като съответните мета-данни. XHTML страница, която съдържа RDFa конструкции е валиден XHTML документ. RDFa използва XHTML съвместими конструкции и разширения, за да изрази RDF съдържанието, а не цели влагане на RDF/XML синтаксиса в XHMTL документи. Когато има ясно специфицирани правила за RDF съдържанието чрез RDFa, то лесно може да бъде извлечено от XHTML и конвертирано до RDF/XML формата, за да бъде използвано от другите технологии за Семантична мрежа.

Нека разгледаме един пример, показващ RDFa синтаксиса в XHTML страница. Първо е описано едно събитие за предстояща конференция, а след това и информация за контакт.

```
html xmlns:cal="http://www.w3.org/2002/12/cal/ical#"
    xmlns:contact="http://www.w3.org/2001/vcard-rdf/3.0#">
...
  <p role="cal:Vevent">
    I'm giving
    <meta property="cal:summary">
      a talk at the XTech Conference about web widgets
    </meta>,
    on
    <meta property="cal:dtstart" content="20060508T1000-0500">
      May 8th at 10am
```

```
      </meta>.
   </p>
...
   <p class="contactinfo" about="http://example.org/staff/jo">
      My name is
      <meta property="contact:fn">
         Jo Smith
      </meta>.
      I'm a
      <meta property="contact:title">
         distinguished web engineer
      </meta>
      at
      <a rel="contact:org" href="http://example.org">
         Example.org
      </a>.
      You can contact me
      <a rel="contact:email" href="mailto:jo@example.org">
         via email
      </a>.
   </p>
...
```

Това позволява на потребителите, които се интересуват от събитието, да копират визуализираната информация от браузера директно в техните календари. За целта се използва речника iCal, чийто URI reference е http://www.w3.org/2002/12/cal/ical#. Аналогично информацията за контакт е описана чрез RDF речника vCard, чийто URI reference е http://www.w3.org/2001/vcard-rdf/3.0# . Дефинирани са 2 XML namespaces за тези речници:

```
<html xmlns:cal="http://www.w3.org/2002/12/cal/ical#"
    xmlns:contact="http://www.w3.org/2001/vcard-rdf/3.0#">
...
```

Мета-данните се описват в параграфи, дефинирани с таг p. В този таг може да се посочи кой RDF URI reference се използва с атрибута about:

```
...
   <p class="contactinfo" about="http://example.org/staff/jo">
      ...everything here pertains to http://example.org/staff/jo...
   </p>
...
```

Ако този атрибут не се използва, се дава служебно RDF URI reference име.

Елементът meta добавя мета-данните, в случая това са предикати и техните стойности. Стойността на предиката може да е текстът, който се визуализира от браузера:

```
I'm giving <meta property="cal:summary">a talk at the XTech Conference about web widgets</meta>,
```

Но ако това не е възможно, тя може да се изрази с атрибута content:

```
<meta property="cal:dtstart" content="20060508T1000-0500">May 8th at 10am</meta>
```

Атрибутът rel се използва за връзка между текущия URI reference, посочен като стойност на атрибута, и URL посочено от href. Например contact:email се асоциира с конкретния mailto:jo@example.org.

След обработка на тези RDFa конструкции се получават следните RDF тройки:

```
_:p0 rdfs:type cal:Vevent;
        cal:summary "a talk at the XTech Conference about web widgets"^^XMLLiteral;
        cal:dtstart "20060508T1000-0500";

<http://example.org/staff/jo>
        contact:fn "Jo Smith"^^XMLLiteral;
        contact:title "distinguished web engineer"^^XMLLiteral;
        contact:org <http://example.org>;
        contact:email <mailto:jo@example.org>.
```

3.3. OWL

3.3.1. Описание на OWL

The Web Ontology Language (OWL) е език за дефиниране и инстанциране на уеб онтологии. Онтологията е термин, зает от философията, който се отнася до науката за описване на нещата в света и как те са свързани помежду си. OWL онтологията може да включва

описание на класове, свойства и техните инстанции. По такава онтология спецификацията за OWL дефинира как да се правят логически изводи, т.е. получават се факти, които не са явно представени в онтологията, но са следствие от семантиката. Тези изводи може да са базирани на един или множество документи, които се комбинират с помощта на OWL механизмите. OWL предполага "open world", т.е. описанията на ресурсите не са ограничени до единствен файл или само в дадена насока. Например клас *C* може да е дефиниран в онтология *C1*, но неговото значение може да се обогатява и в други онтологии. Новата информация в Семантичната мрежа не може да отмени предишна информация. Тя може да е противоречива, но фактите и изводите могат само да се добавят, а не могат да се изтриват. Разработчикът на онтологии трябва да има предвид възможността за такива противоречия. По принцип би трябвало да се създадат програми, които да откриват такива ситуации.

OWL е разширение на RDF-S и има повече възможности за изразяване на значение и семантика от XML, RDF, RDF-S и следователно е по-добър от тях за представяне на семантиката на уеб съдържанието.

OWL езика предоставя три подезика, които са с нарастваща изразителност:

- OWL Lite поддържа йерархична класификация и прости ограничения. Например OWL Lite поддържа ограничението за кардиналност, но стойностите са само 0 и 1.
- OWL DL поддържа максимална изразителност без загуба на изчислителна пълнота (гарантирано е изчислението на всички изводи) и крайност (всички изчисления завършват за крайно време). OWL DL включва всички конструкции на OWL езика с ограничения като разделение на типовете (клас не може да бъде инстанция или свойство, свойство не може да бъде инстанция или клас). Името OWL DL идва от description logics – наука, която се занимава с логика от първи ред.

- OWL Full поддържа максимална изразителност и синтактична свобода на RDF, но без гаранция за изчислителна пълнота или крайност. Например в OWL Full клас може да бъде третиран едновременно както като множество от инстанции, така и като инстанция сам по себе си.

Всеки от тези подезици е разширение на предшественика му в смисъл на това, което може да се изрази и което може да бъде изведено. Следните връзки са валидни:

- Всяка валидна OWL Lite онтология е валидна OWL DL онтология.
- Всяка валидна OWL DL онтология е валидна OWL Full онтология.
- Всяко валидно OWL Lite заключение е валидно OWL DL заключение.
- Всяко валидно OWL DL заключение е валидно OWL Full заключение.

Разработчиците на онтологии, които използват OWL, решават кой от подезиците е най-подходящ за техните нужди. Когато се мигрира от RDF до OWL DL или OWL Lite е необходимо да се провери дали оригиналния RDF документ е съвместим с ограниченията, наложени от OWL DL и OWL Lite.

Сега ще разгледаме един пример с онтологията wine. Нека разгледаме първо речниците, които са използвани. Те се задават в отварящия rdf:RDF таг, като се асоциират с XML namespaces.

```
<rdf:RDF
   xmlns     ="http://www.w3.org/TR/2004/REC-owl-guide-
20040210/wine#"
   xmlns:vin ="http://www.w3.org/TR/2004/REC-owl-guide-
20040210/wine#"
   xml:base  ="http://www.w3.org/TR/2004/REC-owl-guide-
20040210/wine#"
   xmlns:food="http://www.w3.org/TR/2004/REC-owl-guide-
20040210/food#"
   xmlns:owl ="http://www.w3.org/2002/07/owl#"
```

```
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

Първите две декларации идентифицират namespaces, асоциирани с тази онтология. Първата декларация е за namespace по подразбиране, при който имената без префикс се отнасят до текущата онтология. Втората декларация идентифицира namespace на текущата онтология с префикс vin:. Третата декларация идентифицира базовия URI за този документ. Четвъртата декларация идентифицира namespace на онтологията food с префикс food:. Петата декларация специфицира, че в този документ, елементите с префикс owl: трябва да се разбират като отнасящи се до namespace http://www.w3.org/2002/07/owl#. Това е конвенционална OWL декларация, която въвежда OWL речника. OWL зависи от конструкции, дефинирани в RDF, RDFS и XML Schema datatypes. Префиксът rdf: се отнася до namespace http://www.w3.org/1999/02/22-rdf-syntax-ns#. Другите две namespace декларации са rdfs: за RDF Schema и xsd: за XML Schema datatype.

След като namespaces са дефинирани в тага owl:Ontology се дава допълнителна информация за онтологията като коментари, версия и включване на други онтологии.

```
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:priorVersion rdf:resource="http://www.w3.org/TR/2003/PR-owl-
guide-20031215/wine"/>
  <owl:imports rdf:resource="http://www.w3.org/TR/2004/REC-owl-
guide-20040210/food"/>
  <rdfs:label>Wine Ontology</rdfs:label>
   . . .
```

С атрибута rdf:about се дава име на онтологията. Когато стойността на атрибута е „”, което е стандартния случай, името на онтологията е базовото URI на елемента owl:Ontology. Обикновено това е URI на документа, съдържащ онтологията. Изключение е, когато се използва xml:base, за да се зададе базово URI на елемент, което е различно от URI на текущия документ. Атрибутът owl:imports има единствен аргумент, идентифициран от атрибута rdf:resource, който описва онтологията, която

се включва. Разликата между namespace декларацията и owl:import е, че първото предоставя удобен начин за посочване на имена, дефинирани в други онтологии, докато второто показва намерението да се включат твърдения от другата онтология. Когато се включва онтология, всички онтологии, които тя включва, също ще бъдат включени.

OWL изразява онтологична информация, намираща се в множество документи, и затова поддържа свързване на данни от различни източници. Например еквивалентност между различни термини се изразява с атрибута owl:sameAs. Когато такава еквивалентност е установена, информацията от различни източници може да бъде приравнена и да бъде използвана за правене на изводи, които не са пряко налични в нито един от източниците.

3.3.2. Основни елементи в OWL

Основните елементи на една OWL онтология са класовете, свойствата, инстанциите на класове и връзките между тези инстанции. Подобно на обектно-ориентираното програмиране, класовете определят свойствата на инстанциите, които принадлежат към тези класове.

3.3.2.1. Класове

Всяко нещо в OWL света е член на класа owl:Thing. Така всеки клас, който е дефиниран, е неявно подклас на owl:Thing. Класовете се описват чрез елемента owl:Class. В примера са дадени три основни класа: Winery, Region и ConsumableThing.

```
<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>
```

С атрибута rdf:ID се задава името на класа. След тази дефиниция, класът Winery може да бъде посочван в този документ като *#Winery*, например *rdf:resource=#Winery*. Други онтологии могат да посочват това

име чрез пълната му форма *http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Winery*.

Тагът rdfs:subClassOf е основен класификационен конструктор за класове. Той свързва по-специфичен клас с по-общ клас. Този таг е транзитивен. Например с

```
<owl:Class rdf:ID="PotableLiquid">
      <rdfs:subClassOf rdf:resource="#ConsumableThing" />
      ...
</owl:Class>
```

дефинираме, че PortableLiquid е подклас на ConsumableThing.

Дефиницията на клас има две части: име и списък от ограничения. Инстанциите на класа принадлежат на сечението от тези ограничения. Елементът subClassOf е едно от възможните ограничения.

3.3.2.2. Инстанции

Инстанциите са членове на класове. Инстанциите се дефинират с помощта на елемента rdf:type. Ето примерна дефиниция на инстанция от клас Region:

```
<owl:Thing rdf:ID="CentralCoastRegion" />

<owl:Thing rdf:about="#CentralCoastRegion">
  <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

Това съкратено може да се запише и по следния начин:

```
<Region rdf:ID="CentralCoastRegion" />
```

т.е. <име_на_клас rdf:ID=”име_на_инстанция”>.

3.3.2.3. Свойства

Свойствата позволяват да се описват общи релации за членове на класовете, както и специфични факти за инстанциите. Има два типа свойства:

- свойства за типове данни, които описват релациите между инстанции на класове и RDF литерали и XML Schema типове данни;
- свойства за обекти, които описват релации между инстанции на два класа.

Когато се дефинира свойство, т.е. релация, има различни начини за налагане на ограничения. Както релациите в математиката, свойствата се дефинират с домейн (domain) и обхват на стойностите (range). За описание на свойствата се използват таговете owl:DatatypeProperty и owl:ObjectProperty, а за домейните и обхватите съответно rdfs:domain и rdfs:range. Следващият пример показва дефиницията на свойството madeFromGrape, което за обектите от класа Wine получава стойности обекти от класа WineGrape.

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

Едно свойство може да се опише чрез няколко домейна. Тогава неговия домейн е сечението от тези домейни. Това е вярно и за обхватите на стойностите.

В OWL свойствата, подобно на класовете, могат да бъдат организирани в йерархия чрез тага rdfs:subPropertyOf.

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

<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
  ...
</owl:Class>

<owl:ObjectProperty rdf:ID="hasWineDescriptor">
```

```
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range  rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineColor" />
  ...
</owl:ObjectProperty>
```

В този пример се предполага, че класа WineColor е част от класа WineDescriptor. Тогава под-свойството hasColor означава, че ако един обект (вино) има цвят X, то това вино има и стойност „цвят X”на свойството hasWineDescriptor.

Следващият пример показва как се поставя ограничение за наличие на поне едно свойство в дефиницията на клас чрез тага owl:minCardinality.

```
<owl:Class rdf:ID="Wine1">
  <owl:Restriction>
     <owl:onProperty rdf:resource=”#madeFromGrape”/>
     <owl:minCardinality
rdf:datatype=”&xsd;nonNegativeInteger”>1</owl:minCardinality>
  </owl:Restriction>
</owl:Class>
```

Тази дефиниция означава, че всяка инстанция на този клас трябва да участва поне в една релация madeFromGrape.

3.4. JEE 5.0

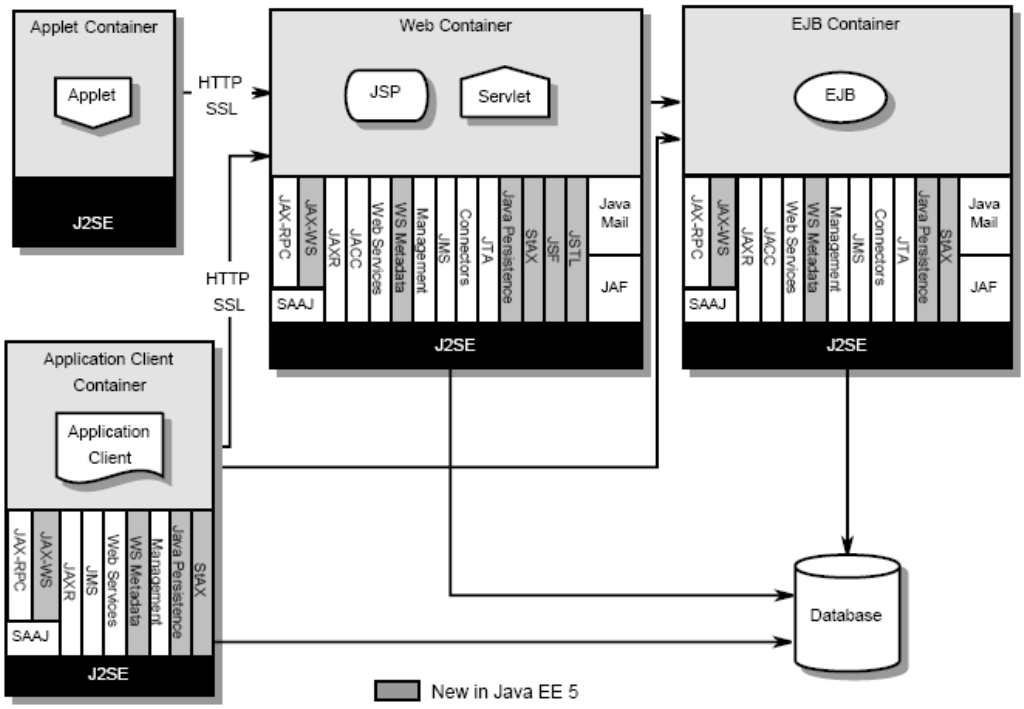
Java Platform, Enterprise Edition (JEE) е спецификация, която дефинира стандартна архитектура за бизнес приложения. Те може да се състоят от следните компоненти:

- Application clients – това са GUI програми на Java, които се изпълняват на десктоп компютри.

- Applets GUI програми на Java, които обикновено се изпълняват в уеб браузер. Те предоставят по-мощен user interface в сравнение с HTML.
- Servlets, Java Server Pages (JSPs), Java Server Faces(JSFs), filters, web event listeners са компоненти, които се изпълняват във web container и отговарят на HTTP заявки от уеб клиенти. Servlets, JSP pages, JSF приложения и filters могат да се използват за генериране на HTML страници, които са презентационната част на приложението и се наричат уеб компоненти. Тези компоненти могат да се използват също така за генериране на XML и други формати, които се обработват от други компоненти. Един вид servlet осигурява поддръжка на уеб услуги по SOAP/HTTP протокола.
- Enterprise JavaBeans (EJB) компонентите се изпълняват в EJB контейнер (container), който е среда поддържаща транзакции. Те обикновено съдържат бизнес логиката на приложението. Могат да предоставят уеб услуги по SOAP/HTTP протокола.

Всеки контейнер предоставя множество услуги (services) на компонентите, за чийто жизнен цикъл се грижи.

Фигура 4 показва архитектурата на стандартно Java EE приложение:



Фигура 4. Архитектура на Java EE 5.0 приложение

Целта на Java EE 5.0 е улесняване разработването на JEE приложения. Широко се използват анотации, въведени в Java езика, версия 5.0. Анотациите намаляват или напълно елиминират нуждата от боравене с JEE deployment descriptors. Една от основните употреби на анотации е да специфицират инжектиране на ресурси в JEE компонентите. Инжектирането спестява използването на JNDI lookup механизмите.

Друга важна промяна е оформянето на JAX-WS технологията, която е наследник на JAX-RPC. JAX-WS използва широко JAXB технологията, за да свързва Java обекти с XML данни. JAX-WS и JAXB са нови за Java EE платформата.

Много важна стъпка е и включването на JSF технологията в Java EE архитектурата. Тя значително опростява разработването на уеб приложения. Подобоно подобрение за бизнес слоя е включването на Java Persistence API, което опростява връзката между Java обекти и базите данни.

Направена е и оптимизация в XML обработката чрез включване на StAX API.

3.4.1. JSF

JavaServer Faces(JSF) е Java-базирана уеб приложна рамка, която опростява разработването на сложен уеб потребителски интерфейс (UI). JSF дефинира UI компонентен модел, който е свързан с жизнения цикъл за обработка на http заявки. Това позволява на Java програмистите да разработват уеб приложения без да се притесняват за http детайлите, както и да интегрират Web UI компоненти чрез събитийно-базиран модел, който широко се използва в Desktop UI приложенията. От друга страна авторите на уеб страниците могат да работят върху "look and feel" аспектите като асемблират готови UI компоненти и разработват само тяхното поведение. Така се намалява максимално нуждата от програмна логика, вградена директно в UI страниците. Също толкова важно следствие от добре дефинирания компонентен модел на JSF рамката е, че могат да се създават мощни програмни среди за разработване, които улесняват създаването на Web приложенията.

JSF е спецификация, която може да бъде реализирана от различни софтуерни доставчици. Тя дефинира множество от основни UI компоненти, които съответстват на елементите в HTML. Но JSF дефинира и Application Programming Interface (API), което позволява създаването на компоненти, които разширяват стандартните или на изцяло нови.

Всеки компонент може да има прикрепени валидатори, които да проверяват въведените от потребителя входни данни и ако те са валидни се продължава останалата бизнес логика. Спрямо действията на потребителя, JSF рамката създава събития и ги праща към съответните компоненти, които са се абонирали за тях. След обработката на тези събития се определя кои страници трябва да бъдат визуализирани като отговор на заявката. На тази стъпка JSF рамката също предоставя улеснен начин за задаване на навигацията между страниците.

JSF не е ограничен само до създаване на HTML страници. За целта се създават "renderers" компоненти отделно от UI компонентите и те определят в какъв формат ще бъдат изходните данни. Така един и същ UI компонент може да е свързан с различни "renderers" и да създава различни изходни данни – например HTML или WML елементи.

3.4.2. Уеб услуги

Java API for XML Web Services (JAX-WS) 2.0 е технология за създаване на уеб услуги и техни клиенти, които си комуникират чрез XML. JAX-WS позволява на програмистите да разработват уеб услуги, базирани на съобщения (message-oriented) или базирани на отдалечено извикване на процедури (RPC-oriented). Тя е наследник на Java API for XML-based RPC 1.1 (JAX-RPC), като новото е, че JAX-WS значително улеснява задачата за разработване на уеб услуги на Java.

В JAX-RPC извикването на уеб услуги се представя чрез XML-базирания протокол SOAP. Той дефинира структурата на съобщението (съдържащо извикването на уеб услуга), кодиращите правила и конвенциите за представяне на заявките към уеб услуги и отговорите от тях. Тези заявки и отговори се предават като SOAP съобщения (XML файлове) по HTTP.

Въпреки че SOAP съобщенията са сложни, JAX-WS API скрива тази сложност от програмиста. От страната на сървъра той трябва само да специфицира операциите, които предоставя уеб услугата чрез дефиниране на методи в интерфейс, написан на Java. След това трябва да кодира един или няколко класа, които реализират тези методи. Клиентските програми са също лесни за кодиране. От страната на клиента трябва да се създаде proxy (локално представяне на уеб услугата) и след това да се извикват методи на това proxy. С JAX-WS програмиста не генерира, нито обработва SOAP съобщения. JAX-WS рамката се грижи за конвертирането на заявките и отговорите между Java API-то и SOAP съобщенията.

С JAX-WS, клиентите на уеб услуги и самите уеб услуги имат голямо предимство: платформената независимост на Java. JAX-WS клиент може да използва уеб услуги, които не са реализирани на Java, както и обратното. Тези гъвкавост е възможна понеже JAX-WS използва технологии, дефинирани от W3C: HTTP, SOAP и Web Service Description Language (WSDL). WSDL специфицира XML формата за описание на уеб услуга като множество от операции, наречени endpoints.

JAX-WS решава някои от проблемите в JAX-RPC 1.1, осигурявайки поддръжка на множество протоколи като SOAP 1.1, SOAP 1.2, XML, както и на средство за поддръжка на допълнителни протоколи по HTTP. JAX-WS използва JAXB 2.0 за свързване на Java обекти с XML данни. Също така поддържа java анотации, с което опростява разработването на уеб услуги и намалява размерите на кода.

3.4.3. Java Persistence API

Java Persistence API е спецификацията от Java EE 5.0, която стандартизира как релационните данни съответстват на обикновени Java обекти ("persistent entities") и как тези обекти се записват в релационна база данни. Тази спецификация опростява entity persistence модела, познат от EJB2.1.

Java Persistence API има следните характеристики:

- Entities (данните на Java, които представят редове в релационната таблица) са Plain Old Java Objects (POJO). За разлика от EJB 2.1 компонентите, които използват container-managed persistence (CMP), entity обектите не са вече специални компоненти с наложени ограничения. Този подход води до по-лесен и по-лек програмен модел.
- Обектно-релационното съответствие е стандартизирано. В EJB 2.1 това съответствие не беше дефинирано и програмистите трябваше да учат специфичното съответствие на всеки доставчик на J2EE платформа. Сега java анотации или XML описания могат да се

използват за специфициране на обектно-релационното съответствие.

- Подържат се наследяване и полиморфизъм. Понеже entities са POJO, entity класа може да наследява друг entity или не-entity клас. Също така не-entity клас може да наследява entity клас. За entities се поддържат и полиморфични асоциации.
- Java Persistence Query Language е базиран на EJB Query Language. Но с Java Persistence Query Language могат да се изразяват заявки на езика за заявки на съответната база данни.
- Поддържат се именовани заявки, които са статични заявки, изразени чрез мета-данни.
- Има прости правила за пакетиране. Понеже entity обектите са прости Java обекти, те могат да бъдат пакетирани навсякъде в приложението. Например в EJB jar, application-client jar, WEB-INF/lib, WEB-INF/classes или в utility JAR в EAR файла.
- Поддръжка на оптимистично заключване с цел оптимизиране на системните ресурси. Това е техника, която избягва заключване, но трябва да се има предвид, че е възможно транзакцията да не приключи успешно, заради колизия с друг потребител.
- Detached entities - понеже entity обектите са POJO, те могат да се сериализират и да се изпратят по мрежата на друг хост, където да се ползват в друг контекст. Затова няма нужда от Data Transfer Objects (DTOs), така популярни от архитектурата на EJB 2.1.
- Използва се стандартно EntityManager API за изпълнение на Create, Read, Update, Delete операциите върху entities.
- Спецификацията дефинира Service Provider Interface (SPI) между Java EE container и база данни. Така лесно могат да се вграждат бази данни от различни доставчици.

4. Реализация на решението

4.1. Изисквания към приложението

Основна задача на приложението е да предостави уеб редактор за създаване на страници със семантично съдържание. За целта е необходимо понятията, които се въвеждат чрез редактора да могат да се асоциират с вече съществуващи термини от речници. Приложението използва уеб услугата на http://www.schemaweb.info/default.aspx, която предоставя множество онтологии от различни области.

Редакторът трябва да има поле за въвеждане на текста на документа и панел с контроли. Основните контроли са:

- File -> New, Open, Close, Save.
- Edit -> Cut, Copy, Paste, Find, Replace.
- Ontology -> Associate, Delete.
- Hyperlink -> Insert, Delete.
- Table -> Insert, Edit, Delete.
- Format -> Text.

File -> New създава нова празна уеб страница. Ако има вече отворена страница, която не е запазена, излиза съобщение дали последните промени да бъдат запазени преди отварянето на новата страница.

File -> Open отваря страница, която вече е създадена и е съхранена в уеб сървъра, на който върви приложението.

File -> Close затваря страницата, която е била отворена в редактора.

File -> Save запазва страницата на уеб сървъра.

Edit -> Cut отрязва маркирания текст и го слага в буфер, така че този текст в последствие да бъде преместен на друго място в документа.

Edit -> Copy копира маркирания текст без да го изтрива и го слага в буфер, така че този текст да може да бъде поместен и на друго място в страницата.

Edit -> Paste копира текста, който е бил поставен в буфера чрез File -> Cut или File -> Copy, на мястото, където се намира текущия маркер в текста.

Ontology -> Associate се използва за асоцииране на даден текст със съществуваща онтология като за целта той трябва да бъде маркиран. В текстово поле се записва ключова дума, по която да се търси подходяща онтология. След като се намери тази онтология се показват всички класове на тази онтология, свойствата на тези класове и съответните стойности. Когато има готови стойности, се избират те. В противен случай е необходимо потребителят да създаде инстанциите като посочи към кои класове принадлежат.

Ontology -> Delete се използва, за да се премахне вече направена асоциация между маркирания текст и конкретна онтология.

Hyperlink -> Insert включва hyperlink в текста. Излиза панел, в който се въвежда името и самия hyperlink.

Hyperlink -> Delete изтрива hyperlink като за целта е необходимо той да е предварително маркиран в текста.

Table -> Insert включва празна таблица в текста.

Table -> Edit позволява променяне на таблицата, която е маркирана чрез посочване на брой редове и колони.

Table -> Delete изтрива таблицата, която е маркирана.

Format -> Text се използва за определяне на формата на маркиран текст. В допълнителен панел се описва какъв да бъде шрифта и големината на текста.

След като е създадена страницата чрез редактора и се натисне Save, тя се обработва на сървъра. Мета-данните се записват в отделна база данни във вид на RDF тройки, така че в бъдеще да може да бъде предоставена чрез SPARQL интерфейс на други програми в Семантичната мрежа. Самата страница се запазва в XHTML 2.0 формат с RDFa съдържание, за да може да бъде визуализирана в браузерите и в същото време мета-данните да се използват от вече наличните програми-

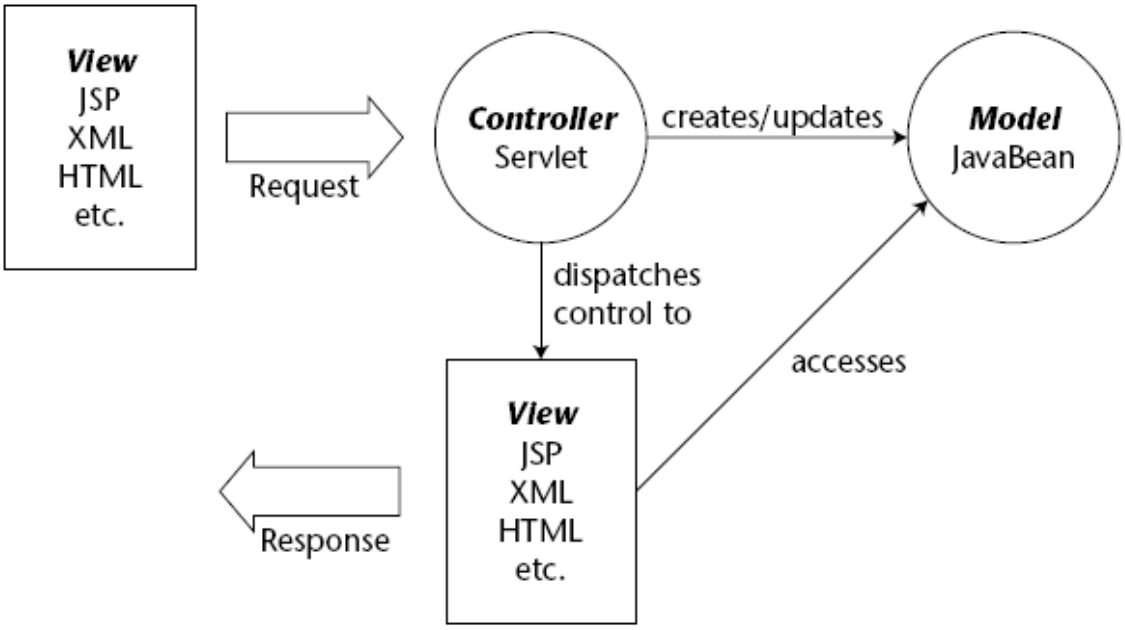
търсачки, които са доразвити да търсят информация и спрямо този формат.

4.2. Архитектура на решението

Архитектурата на приложението се определя от основната архитектура за уеб приложения, предоставена от Java EE 5.0 платформата. Според нея приложенията трябва да следват Model-2 MVC модела. В него е залегнала основната концепция на MVC (Model-View-Controller) модела, но адаптирана за HTTP протокола, за който е характерно, че е без състояние т.е. не пази състоянието между различните заявки.

4.2.1. Model-2 MVC архитектура

Нека се запознаем първо с Model-2 MVC модела. Потребителят използва уеб браузер, за да изпрати заявка за информация от сървъра. Заявката се изпраща на определен URL, който е дефиниран по време на разработката на приложението. На този URL на сървъра слуша controller servlet (Controller/контролер), който получава заявката, взаимодейства с бизнес модела (Model/модел) и определя коя страница (View/изглед) се връща като отговор към потребителя. Обикновено JSP компонент се грижи за самото генериране на страницата. Той може да поиска информация от модела, за да визуализира динамичните данни. Фигура 5 представя Model-2 MVC модела.

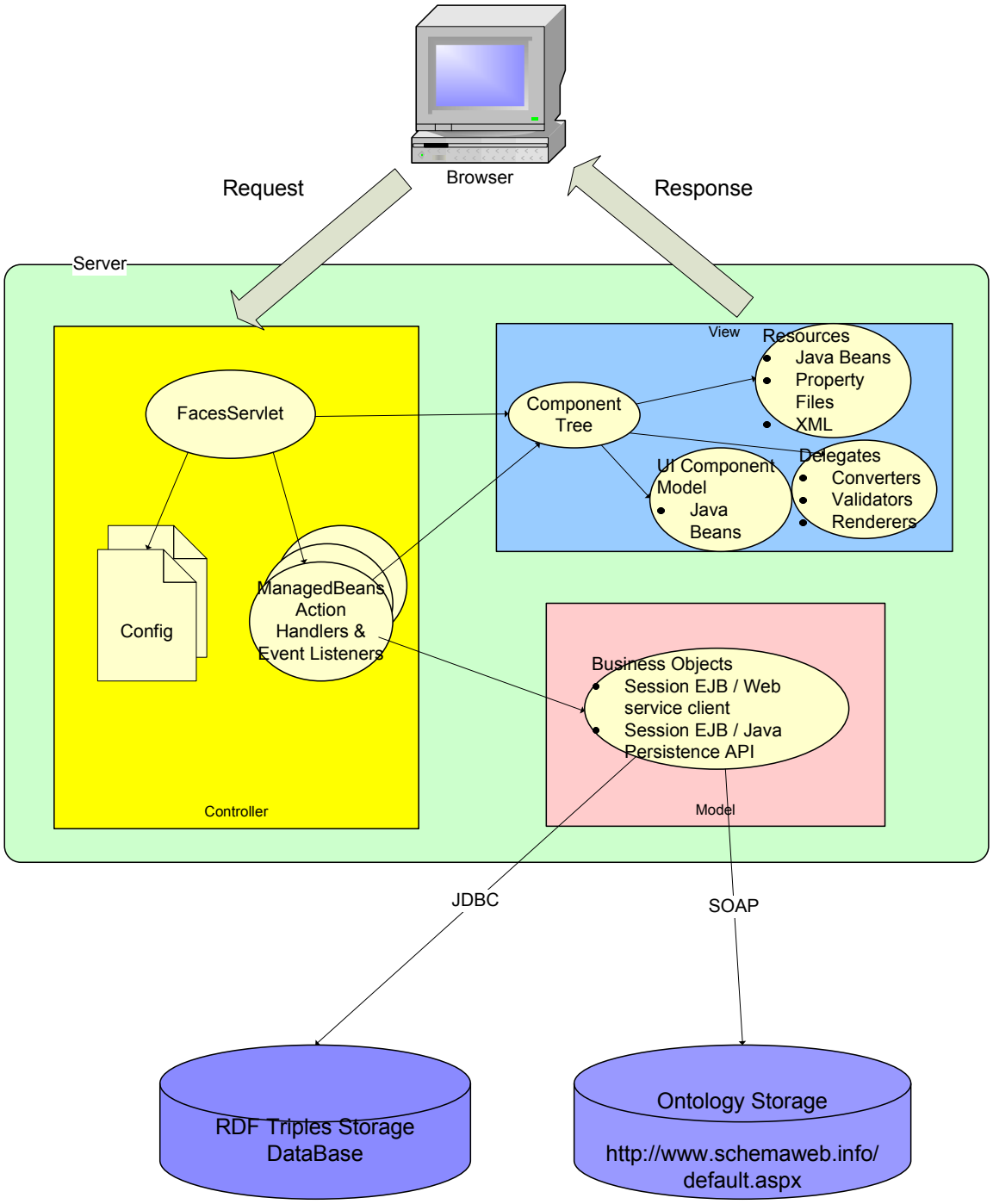


Фигура 5. Model-2 MVC модел

Java EE 5.0 използва технологиите JSF за реализация на изгледа и контролера, а EJB и Java Persistence API за модела. Разглежданото в тази дипломна работа приложение в допълнение използва за бизнес логиката си, т.е. в модела, и уеб услигу за връзка с външния доставчик на онтологии.

4.2.2. Архитектура на приложението

Архитектурата на приложението на компонентно ниво е показана на фигура 6:



Фигура 6. Архитектура на приложението

4.2.2.1. Контролер

Контролерът (Controller) се представя от сървлет (servlet), наречен FacesServlet в JSF спецификацията, един или повече конфигурационни

файлове, както и множество от специални JavaBean компоненти, наречени managed beans. FacesServlet отговаря за получаването на входящите заявки от уеб клиенти и изпълняването на определени операции, необходими за подготовката и предаването на отговора. Връзката между изгледа и модела се осъществява от managed beans компонентите. Те съдържат свойства, които се взимат от потребителската заявка, и event listener методи, които обработват тези свойства и манипулират UI изгледа или изпълняват обработка върху данните от модела на приложението. Managed beans компонентите се асоциират със съответните UI компоненти декларативно чрез JSF expression language (EL). Когато UI компонентът се промени, автоматично се променя и съответния managed bean, с който е асоцииран. Обратното също е вярно. От друга страна managed beans взаимодействат с модела, понеже JSF рамката предоставя достъп до услуги като бази данни, уеб услуги, EJB и други.

4.2.2.2. Изглед

Изгледът (View) на приложението съдържа множество от UI компоненти, делегиращи компоненти (валидатори, визуализиращи и конвертиращи компоненти), както и допълнителни ресурси.

Една страница в JSF се представя като едно компонентно дърво, което съдържа всички UI елементи на страницата (например бутони, календари и т.н.). Връзката между компонентите е родител-деца, т.е. има влагане. Например форма може да съдържа етикет, текстово поле и бутон. Коренният компонент на дървото се нарича "view" и представя цялата страница. Тези UI компоненти се представят като JavaBean компоненти, т.е. имат състояние, свойства, методи и поддръжка от компонентно-базирани среди за разработка (IDE). Тези компоненти не съдържат кода, който ги визуализира, а само свойствата и функционалността на компонента. Визуализацията е отделена в друг вид компоненти. Поради това, че са формирани като самостоятелни

компоненти, могат да се използват вече готови UI компоненти. Повечето среди за разработка на JSF приложения предлагат реализации на основните UI компоненти, и е необходимо само да се асемблират и конфигурират в приложението. Важно е да се отбележи, че те се намират на сървъра, а не на клиента. JSF рамката на сървъра осигурява запазване на състоянието на компонентите между различните http заявки.

Компонентите, които са отговорни за визуализацията на UI компонентите, се наричат „renderers". Те са базирани на определен формат, например HTML, WML, SVG и т.н. Основните им задачи са кодиране (представяне на UI компонента във формат, който се разбира от клиента) и декодиране (извличане на параметрите на http заявката и прилагането на тези стойности върху съответния UI компонент на сървъра). Отделяйки UI компонентите от тяхната визуализация, се позволява да се поддържат няколко вида визуализация едновременно за различни клиенти.

Валидирането на стойностите на UI компонентите се изпълнява от валидиращите компоненти (validators). Те също се намират на сървъра. JSF рамката предлага валидатори по подразбиране за повечето UI компоненти.

Конвертиращите компоненти (converters) се асоциират също към UI компонентите. Те конвертират java обекти към стрингове, за да могат да се визуализират тези обекти на съответния markup език, когато се генерира отговор на http заявката. И обратно – когато се обработва вход от заявката, конверторите превръщат стрингове към обекти.

4.2.2.3. Модел

За реализацията на модела на приложението се използват технологиите session EJB, уеб услуги за достъп до доставчика на онтологии, както и новото Java Persistence API в Java EE 5.0 за данните, които трябва да се съхраняват в базата данни.

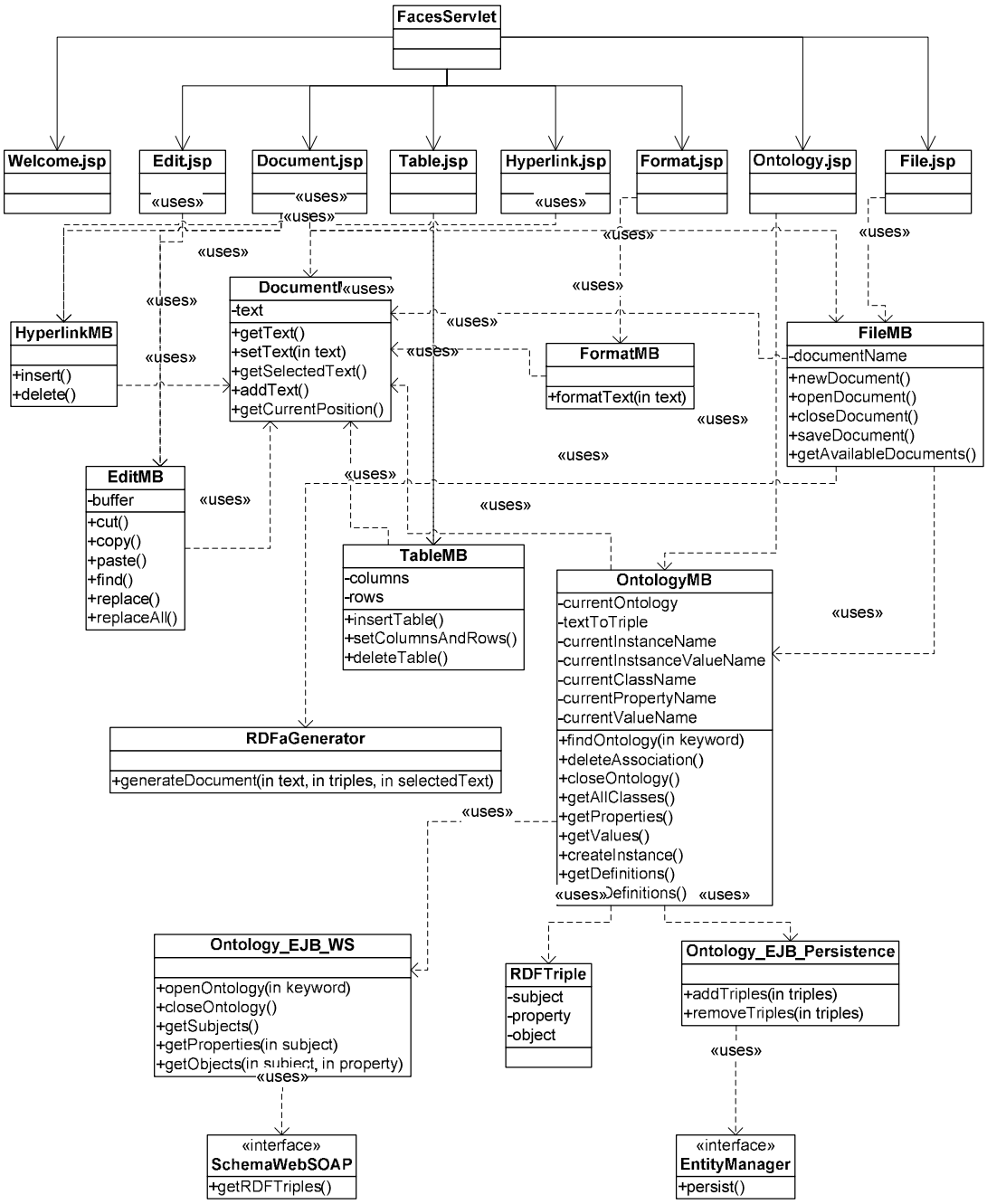
Session EJBs са компоненти, които съдържат логиката на бизнес процесите на приложението. Техният жизнен цикъл се управлява от EJB контейнера на сървъра. Те съществуват само в рамките на сесията за даден потребител и не се съхраняват в база данни, но комуникират с Java Persistence API, което предоставя достъп до базата данни. Има два вида session EJBs – stateless и stateful. Stateless session EJBs нямат състояние между различните заявки от клиента, докато stateful session EJBs пазят това състояние до края на потребителската сесия. Session EBJs използват уеб услуги и Java Persistence API с помощта на java анотации, които са декларативен начин за достъп до определени компоненти и услуги.

Обектите, които се съхраняват в базата, в Java Persistence API се наричат entities. Те са обикновени Java обекти, т.нар. POJOs. Java Persistence API е рамка, която предоставя обектно-релационно съответствие и скрива сложността от работата с базата данни. Session EJBs използват EntityManager класа от Java Persistence API, за да създават, променят и търсят entities в базата данни.

За да могат да се използват уеб услуги, които се предлагат от http://www.schemaweb.info/default.aspx, първо е необходимо да се генерират java proxy класовете от wsdl файла, който описва уеб услугата. След като са генерирани класовете за java клиента на уеб услугата, съответният session EJB включва уеб услугата чрез java анотация и започва извикването на методите.

4.3. Дизайн на решението

Фигура 7 представя UML статична диаграма, която описва дизайна на приложението. Показани са основните класове, които реализират архитектурата. Конфигурационните файлове и помощните класове към тези основни класове не са показани.



Фигура 7. Дизайн на приложението

4.3.1. JSF UI

В основата на диаграмата стои класа javax.faces.webapp.FacesServlet, защото това е единствената входна точка за приложението – потребителите използват уеб браузер, за да могат да работят с него. FacesServlet управлява жизнения цикъл на обработката на http заявката, който се състои от 6 фази:

1. Възстановяване за изгледа – създава дърво от UI компоненти за избраната страница. Ако това дърво е било създадено от предишно извикване на страницата, то не се създава пак, а се намира последното му състояние, което се пази от JSF рамката.
2. Прилагане на стойностите от заявката – подновява стойностите на UI компонентите, за да се изравнят с тези, изпратени чрез заявката. Тук се използват конверторни компоненти. Грешките по време на конвертиране се обработват по специален начин.
3. Валидация – казва на всеки компонент да се валидира. Може да се използват и външни валидатори. Грешките по време на валидация се обработват.
4. Подновяване стойностите на модела – подновява всички стойности на managed beans, асоциирани с UI компонентите. Грешките от тази фаза също се обработват по специален начин.
5. Извикване на всички регистрирани action listeners, включително на тези по подразбиране на компоненти като HtmlCommandButton. Спрямо резултата от тези извиквания се определя следващата страница, която се връща като отговор към потребителя.
6. Визуализиране на отговора – представя избраната страница чрез визуализиращата технология, в случая HTML.

На следващо ниво в диаграмата са JSP файловете, между които се навигира в зависимост от конкретните условия. Всъщност това са JSF файлове, т.е. JSP с JSF taglibs.

Първият JSP файл, който се извиква, когато потребителят се върже към приложението за първи път, е Welcome.jsp. Той визуализира уводно съдържание и бутон за вход.

След като потребителя влезе в системата, FacesServlet навигира към Document.jsp. Този файл визуализира редактора. Той съдържа меню за контролите File, Edit, Ontology, Picture, Hyperlink, Table, както и текстово поле, където потребителя въвежда текста. В този момент само контролът File е активен и в него потребителят избира една от опциите New или Open. Тогава FacesServlet навигира към File.jsp. Ако потребителят е бил избрал New, тогава File.jsp визуализира поле, където да се напише името на новия файл. Ако е избрано Open, тогава File.jsp визуализира списък от всички налични документи, които са създадени с редактора и вече са записани на сървъра и се избира един от тях. След като е направен избора, отново се навигира към Document.jsp, но в него вече всички контроли са активни и може да започне създаването или редактирането на текста.

Контролът Edit има опциите Cut, Copy, Paste, Find, Replace. При първите три не се навигира към друга страница, а промените са в рамките на Document.jsp. При Find и Replace се навигира към Edit.jsp, което показва допълнителен панел със съответните полета. При Find се показва Search бутон и едно текстово поле за въвеждане на текста, който се търси. При Replace се показват две текстови полета за текста, който се търси, и текста, който го замества. При тази операция се появяват също бутоните Find next за следващо срещане на търсения текст, Replace за еднократно заместване и Replace All за заместване на всички срещания. В Edit.jsp има също така бутон Close, при чието натискане JSF рамката навигира отново към Document.jsp.

Следващият основен контрол е Ontology с опциите Associate и Delete. Тези опции са валидни само, ако има маркиран текст в текстовото поле на редактора. При Delete, ако текстът е бил асоцииран преди това с онтология, се премахва тази асоциация. При избиране на Associate JSF рамката навигира към Ontology.jsp. Той визуализира изгледа за избиране

на онтология и дефиниране на инстанция на клас от тази онтология. В горната част на страницата се показват текстът, за който се дефинират мета-данните и текстово поле за въвеждане на ключова дума, по която се търси подходяща онтология. До него има бутон за стартиране на търсенето. Под тях са разположени три списъкови полета. След като е намерена подходяща онтология в първото поле се зарежда списък с всички класове на тази онтология. При избор на даден клас се появяват всички свойства на този клас във второто списъчно поле. При избор на свойство от този списък се появяват всички стойности за това свойство – те могат да са инстанции или класове. Под полетата има две текстови полета. Първото е за име на инстанцията на класа, който е маркиран. Второто е за име на инстанцията на стойността спрямо избраното свойство за този клас. До тях има бутон за създаване на инстанция след като са избрани подходящите клас, свойство и стойност и са дадени съответните имена на инстанцията и нейната стойност спрямо свойството. До полетата има дърво с дефинициите на дадения термин. То има връх с името на инстанцията, която е дефинирана за този термин. Възлите под върха са свойствата на класа на инстанцията. Под тях са възлите за съответните стойности. Те може да принадлежат на други класове, така че по същия начин може да се продължи дефинирането на дървото все по-надолу. В горния ляв край на страницата има бутон “Back to editor” за връщане към редактора. При неговото натискане JSF рамката навигира обратно към Document.jsp.

Следващият контрол в Document.jsp е Hyperlink с опциите Insert и Delete. При избор на Delete се изтрива хипер-връзка, която преди това е била маркирана в тесктовото поле на редактора. При Insert JSF рамката навигира към Hyperlink.jsp. То показва нов панел до текстовото поле на редактора, който съдържа две текстови полета, описващи хипер-връзката. Първото е за текст, който визуализира хипер-връзката. Второто е за самия път на хипер-връзката. Под тях има бутон Create за създаване на хипервръзката на текущата позиция в редактора. До него има бутон

Close за затваряне на този изглед. При неговото натискане JSF рамката навигира към Document.jsp.

Следващият контрол в Document.jsp е Table с опциите Insert, Edit, Delete. При Delete се изтрива таблицата, която преди това е била маркирана в редактора. При избор на Insert или Edit JSF рамката навигира към Table.jsp. При Insert се създава празна таблица в редактора, която е маркирана и допълнително може да се посочат стойности на брой редове и колони. При Edit се дава възможност за редактиране на вече създадена и маркирана таблица. За целта и при двете опции до текстовото поле на редактора се появява панел. Той съдържа две текстови полета – едно за специфициране на брой колони и друго за брой редове. Под тях се намират бутоните Save за запазване на тези стойности и Close за затваряне на този панел. И при двата бутона JSF рамката навигира отново към Document.jsp.

Последният контрол в Document.jsp е Format с опция Text. При него се дава възможност за посочване на формата на маркиран текст в текстовото поле на редактора. При избора на този контрол JSF рамката навигира към Format.jsp. Тази страница съдържа текстовото поле на редактора и до него панел за форматирането на текста. Панелът съдържа три списъчни полета – едно за име на наличните шрифтове, второ за вид на шрифта (regular, bold, italic, bold italic) и третото за големина на шрифта. Под тези полета има бутони Save за запазване на промените и Close за затваряне на този панел. При избор на тези бутони се навигира към Document.jsp.

Контролът File има също така опции Save и Close. При Save документът се записва на сървъра, но остава отворен в редактора. Затова не се навигира към друга страница. При Close документът се затваря и JSF рамката навигира към Welcome.jsp.

4.3.2. JSF Managed Beans

Зад всяко JSP стоят съответни managed beans, които изчисляват резултатите преди да се визуализират с JSP. Всички managed beans в това приложение са дефинирани като сесийни. Комуникацията между самите managed beans се извършва с помощта на се осъществява през API, предоставено от JSF рамката.

4.3.2.1. DocumentMB

Document.jsp използва managed bean с клас DocumentMB. Той има атрибут text, който се асоциира към текста на текстовото поле на редактора чрез JSF техниката „bound value”. Методите на този клас са:

- String getText() – връща текущия текст от текстовото поле на редактора.
- String getSelectedText() – връща маркирания текст от текстовото поле на редактора.
- int getCurrentPosition() – връща текущата позиция в текстовото поле на редактора.
- addText(String text) – добавя текст към вече наличния текст в редактора.
- setText(String text) – заменя текущия текст в редактора с този, който се подава като параметър.

Този managed bean се използва от всички други managed beans, защото те обработват по специфичен начин текста на документа и се нуждаят от достъп до този текст. Първоначално стойността на атрибута text е null. Той се инстанцира, само когато се асоциира с даден документ, т.е. трябва да се избере File -> New за започване на нов или File -> Open за отваряне на съществуващ документ. При File -> Close атрибутът отново получава стойност null.

4.3.2.2. FileMB

FileMB е managed bean, който се използва във File.jsp и Document.jsp. Той се грижи за създаването, отварянето, записването и затварянето на документа. File.jsp го използва при контролите File -> New и File -> Open, а Document.jsp при File -> Save и File -> Close. Има атрибут documentName, който се асоциира с текстовото поле за име на документа във File.jsp. Методите на FileMB са:

- void newDocument() – създава нов документ с име стойността на атрибута documentName в поддиректорията docs на работната директория на приложението. След това извиква DocumentMB.setText(“”), за да може след навигацията към Document.jsp да се визуализира бял екран с празен текст в текстовото поле на редактора.
- String[] getAvailableDocuments() – връща списък от всички налични документи в поддиректорията docs.
- void openDocument() – отваря в текстовото поле на редактора документа с име стойността на атрибута documentName. За целта се използва DocumentMB.setText(String text).
- void saveDocument() – записва промените във файла. След това извиква OntologyMB.saveDefinitions(), за да се запишат всички направени дефиниции в RDF storage. Накрая е необходимо да се генерира съответния RDFa документ. За целта извиква първо DocumentMB.getText() и OntologyMB.getDefinitions() и предава резултатите от тях на RDFaGenerator.generateDocument(String text, Hashtable textToTriple), който генерира RDFa текста и го връща на FileMB. Тогава RDFa документа се записва в поддиректорията rdfa_docs на работната директория на приложението.
- void closeDocument() – атрибутът documentName получава стойност null.

4.3.2.3. EditMB

EditMB е managed bean, който се използва в Document.jsp и Edit.jsp. Document.jsp го използва при контролите Edit -> Cut, Edit -> Copy и Edit -> Paste. Edit.jsp използва EditMB при Edit -> Find, Edit -> Replace и Edit -> ReplaceAll. Атрибутът на този managed bean е String buffer. Той получава стойност само при избор на Edit -> Cut или Edit -> Copy. В началото и след Edit -> Paste стойността му е null.

- void cut() – взима маркирания текст чрез DocumentMB.getSelectedText() и го слага в буфера, т.е. атрибута buffer получава стойност този текст. Взима целия текст чрез DocumentMB.getText() и го променя като отрязва маркираната стойност. След това го подновява в текстовото поле на редактора чрез DocumentMB.setText().
- void copy() – действа по същия начин като void cut(), но с разликата, че не отрязва маркирания текст.
- void paste() – взима текущата позиция в текста чрез DocumentMB.getCurrentPosition(), както и самия текст чрез DocumentMB.getText(). На текущата позиция добавя стринга от буфера и изчиства буфера, т.е. атрибута buffer получава стойност null. Подновява текста в текстовото поле чрез DocumentMB.setText().
- int find(String searchForText) - получава текущия текст чрез DocumentMB.getText(), намира първото срещане на търсения текст и връща позицията му в текста на документа.
- int replace(String searchForText, String replaceWithText) – получава текущия текст чрез DocumentMB.getText(), намира търсения текст в текста на документа и го замества с втория параметър. След това чрез DocumentMB.setText() подновява текста и като резултат новият текст се появява в текстовото поле на редактора.
- void replaceAll(String searchForText, String replaceWithText) – действа по същия начин както int replace(String searchForText,

String replaceWithText), но заменя търсения текст с втория параметър за всичките му срещания в текста.

4.3.2.4. OntologyMB

OntologyMB е managed bean, който се използва от Ontology.jsp. Той се грижи за обработката на онтологиите, с които се асоциира текста. Той предоставя методи за търсене и зареждане на онтология, както и дефиниране на инстанции към съответната онтология. Използва session EJBs Ontology_EJB_WS и Ontology_EJB_Persistence, включвайки ги чрез java анотации. Инстанциите са представени от класа RDFTriple, който съдържа името на инстанцията, класа, към който принадлежи, и множество от свойства и съответните стойности. Тези стойности са също от класа RDFTriple. Атрибутите на този managed bean са String currentOntology, Hashtable textToTriple, String currentInstanceName, String currentInstanceValueName, String currentTreeNode, String currentClassName, String currentPropertyName, String currentValueName. Атрибутът currentInstanceName се асоциира с текстовото поле за име на инстанция, а атрибутът currentValueName с текстовото поле за име на стойност в тройката. CurrentTreeNode се асоциира с текущия избран връх на дървото с дефиниции. CurrentClassName се асоциира с текущия избран клас от списъка с класове, currentPropertyName с текущото избрано свойство и currentValueName с текущата избрана стойност. Ключовете на хеш таблицата са стринговете от текста на документа, за които е било дефинирано значение. Стойностите са съответните инстанции на RDFTriple. Методите на OntologyMB са:

- void findOntology(String keyword) – извиква session bean Ontology_EJB_WS, който връща име на онтологията и то се дава като стойност на атрибута currentOntology.
- String[] getAllClasses() – извиква Ontology_EJB_WS, който връща всички класове, дефинирани в текущата онтология.

- String[] getProperties() – спрямо текущия избран клас, запазен в атрибута currentClassName, извиква Ontology_EJB_WS и той връща всички свойства за този клас.
- String[] getValues() – спрямо текущите избрани клас и свойство, запазени в атрибутите currentClassName и currentPropertyName, извиква Ontology_EJB_WS и той връща всички стойности на тройки с такива субект и предикат.
- void createInstance() – създава инстанция на RDFTriple, която представя дефиницията на една RDF тройка. За тази дефиниция се използват стойностите на атрибутите currentInstanceName, currentClassName, currentPropertyName, currentValueName, currentInstanceValueName. Към първата дефинирана инстанция, т.е. корена на дървото, се създава още една тройка, която има предикат rdfs:comment. Целта е да се запази асоциацията между дефинираната инстанция и текста, който се описва с RDF. Чрез DocumentMB.getSelectedText() се взима маркирания текст от документа и се използва като обект на новата тройка. В допълнение към това маркирания текст и оригиналния RDFTriple обект се поставят в хеш таблицата textToTriple.
- void closeOntology() – извиква се при натискане на бутона "Back to editor". Ontology_EJB_WS изчиства текущото състояние на заредената онтология. Всички атрибути на OntologyMB получават стойност null с изключение на дефинираните тройки.
- Hashtable getDefinitions() – връща атрибута textToTriple.
- void saveDefinitions() – извиква session EJB Ontology_EJB_Persistence, за да запази всички налични дефиниции в базата данни.
- void deleteAssociation() – чрез DocumentMB.getSelectedText() взима текущия маркиран текст и го изтрива от хеш таблицата textToTriple.

4.3.2.5. TableMB

TableMB е managed bean, който се използва в Document.jsp и Table.jsp. Той е предназначен за обработка на таблици в документа. Document.jsp го използва при контрола Table -> Delete. След избор на Table -> Insert или Table -> Edit се навигира към Table.jsp, което използва TableMB. Атрибутите на този managed bean са columns и rows, които се асоциират с текстовите полета за брой колони и брой редове. Методите на TableMB са:

- void insertTable() – чрез DocumentMB.getText() и DocumentMB.getCurrentPosition() взима текста и текущата позиция в него и добавя на тази позиция празна таблица. След това чрез DocumentMB.setText() подновява текста на документа.
- void setColumnsAndRows() – чрез DocumentMB.getText() и DocumentMB.getSelectedText() взима текста и маркираната таблица в него и променя нейните колони и редове спрямо атрибутите columns и rows. След това чрез DocumentMB.setText() подновява текста на документа.
- void deleteTable() – чрез DocumentMB.getText() и DocumentMB.getSelectedText() взима текста и маркираната таблица в него и я изтрива. След това чрез DocumentMB.setText() подновява текста на документа.

4.3.2.6. HyperlinkMB

HyperlinkMB е managed bean, който се използва в Hyperlink.jsp след избор на Hyperlink -> Insert и в Document.jsp при Hyperlink -> Delete. Той се грижи за създаването и изтриването на хипер-връзки в текста. Атрибутите на този managed bean са displayName и hyperlink, асоциирани със съответните текстови полета на Hyperlink.jsp. Методите на HyperlinkMB са:

- void insert() – извиква DocumentMB.getText() и DocumentMB.getCurrentPosition() и включва на тази позиция хипер-

връзката. След това подновява текста на документа като извиква DocumentMB.setText() с параметър новия текст.

- void delete() - извиква DocumentMB.getText() и DocumentMB.getSelectedText(), който съдържа маркираната хипер-връзка и я изтрива. След това подновява текста на документа като извиква DocumentMB.setText() с параметър новия текст.

4.3.2.7. FormatMB

FormatMB е managed bean, който се използва във Format.jsp. Той се грижи за форматирането на текста. Атрибутите му са font, fontStyle и size, асоциирани със съответните текстови полета на Format.jsp. FormatMB има само един метод:

- void format() - извиква DocumentMB.getText() и DocumentMB.getSelectedText() и го форматира спрямо стойностите на атрибутите font, fontStyle и size. След това подновява текста на документа като извиква DocumentMB.setText() с параметър новия текст.

4.3.3. Session EJBs

Следващото ниво в модела на приложението са Session EJBs. Използват се два такива компонента: Ontology_EJB_WS и Ontology_EJB_Persistence.

4.3.3.1. Ontology_EJB_WS

Ontology_EJB_WS използва анотации, за да включи клиент на уеб услуга. По този начин си осигурява достъп до доставчика на онтологии SchemaWeb. Web service endpoint интерфейсът, който се използва е SchemaWebSoap. Има атрибут namespace. Методите на този EJB session bean са:

- String findOntology(String keyword) – намира namespace на онтологията и го запазва в атрибута. Извиква SchemaWebSoap.getRDFTriples() с параметър атрибута, който връща всички RDF тройки дефинирани в тази RDF/XML schema. Тези тройки се обработват и от тях се отделят класовете, свойствата и стойностите.
- String[] getAllClasses() - връщат се всички класове за текущата заредена онтология.
- String[] getAllProperties(String className) – връщат се всички свойства за дадения клас от текущата онтология.
- String[] getAllValues(String className, String propertyName) – връщат се всички стойности за дадените клас и свойство от текущата онтология.
- void closeOntology() – атрибутът ontology получава стойност null.

4.3.3.2. Ontology_EJB_Persistence

Ontology_EJB_Persistence е Session EJB, който се грижи да запази дефинициите на новите RDF тройки, които потребителя създава чрез това приложение. Използва EntityManager интерфейса на Java Persistence API, включвайки го чрез анотации. Entity обектите са инстанциите на RDFTriple класа. Те се записват в таблицата RDF_TRIPLE(SUBJECT, PROPERTY, VALUE). Ontology_EJB_Persistence предоставя само един публичен метод в интерфейса си:

- void saveDefinitions(Hashtable textToTriple) – извлича всички RDFTriple обекти от хеш таблицата и ги записва в базата данни, извиквайки EntityManager.persist(triple).

4.4. Кодиране на решението

За кодирането на уеб приложението са използвани следните средите за разработка Sun Java Studio Creator и SAP NetWeaver Developer Studio.

Sun Java Studio Creator се използва за реализирането на изгледа на приложението. Той е среда за бърза разработка на приложения (Rapid Application Development - RAD), т.е. предоставя панел с множество от готови UI компоненти, които се асемблират в приложението чрез изтегляне с мишката. Всеки компонент има свойства и генерира събития. Комуникацията между компонентите става само чрез генериране и консумиране на събития. Конфигурането на свойствата и събитията на компонентите става изцяло във визуалната среда, която генерира кода за приложението.

SAP NetWeaver Developer Studio се използва за останалата част на приложението – за реализацията на всички managed beans, session EJBs, както и на генерацията на клиентските proxy класове от WSDL за уеб услугата, който се използва. Причината за това е, че приложението трябва да върви върху SAP JEE 5.0 Server и има специфични конфигурационни файлове, които се създават със SAP NetWeaver Developer Studio.

4.5. Тестване на реализацията

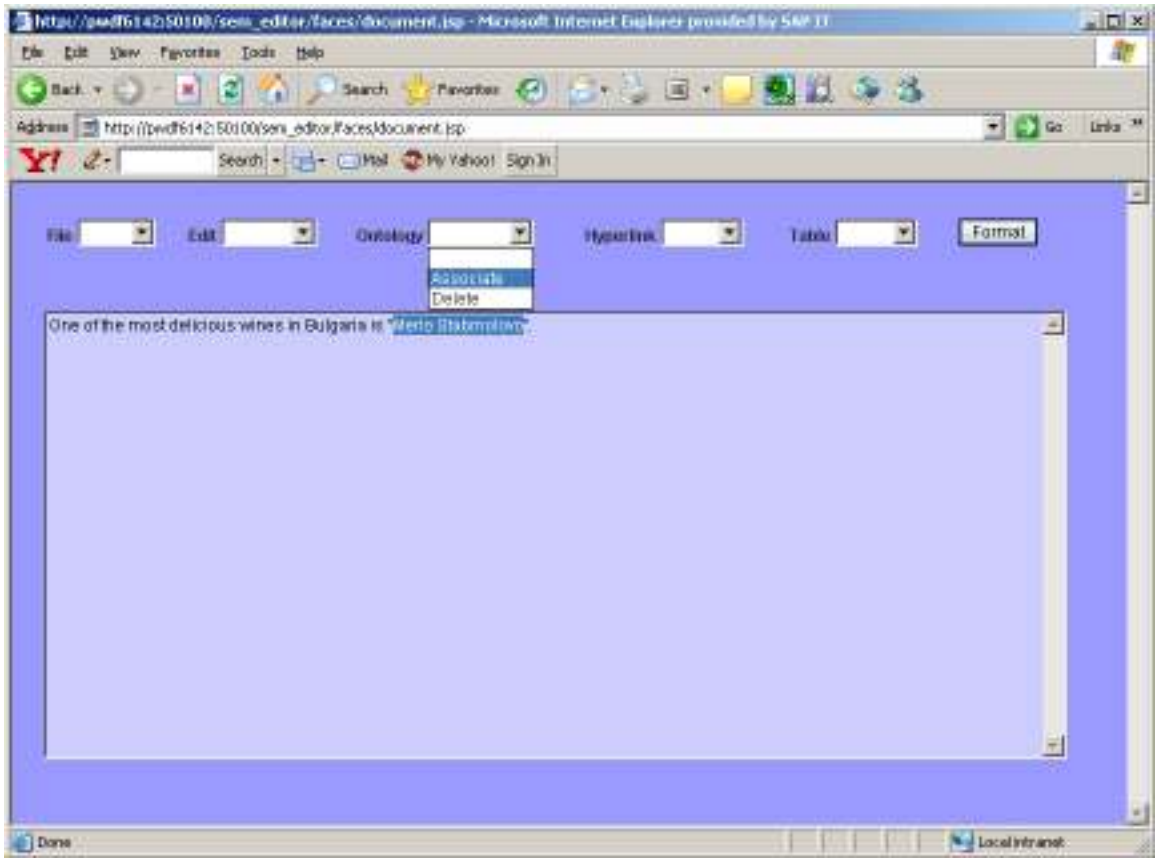
Тестването на приложението се извършва на два етапа – самостоятелно преди да се инсталира на сървъра, и след като се инсталира.

При първия тип се тества кода на приложението извън контекста на сървъра. Това се осъществява с JUnit рамката. Следвайки изискванията на тази рамка, се създават тестови класове. С тяхна помощ се тестват всички методи на класовете на приложението с подходящи параметри, за да се докаже тяхната вярност на най-ниското компонентно ниво – класа.

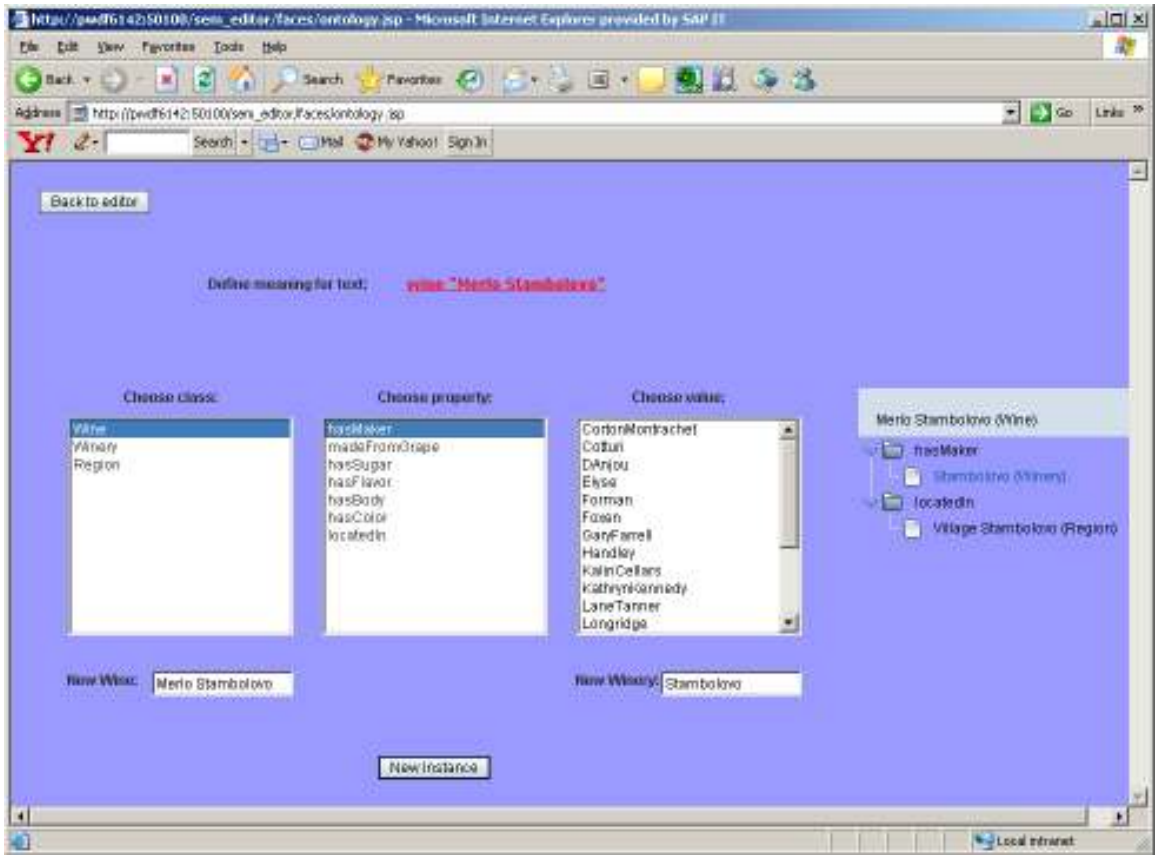
Вторият етап на тестване е след като е инсталирано приложението на сървъра. Тогава може да се извърши тестване на цели сценарии. То се осъществява чрез средата за тестване LoadRunner. Тя предоставя възможност за записване на http скриптове, съдържащи заявките към

приложението. За целта се използва ScriptRecorder приложението на LoadRunner, който стартира уеб браузер. Целият сценарий, който трябва да се тества, се изпълнява през този браузер, а ScriptRecorder записва в скрипт всички http заявки към сървъра и техните отговори. След това този скрипт може да се пуска автоматизирано и да изпълнява сценария от името на потребител, използващ уеб браузер. Нещо повече – данните в скрипта могат да се параметризират, така че той да може да се изпълнява от името на различни потребители. Стартирайки скрипта за максималния брой потребители, които се изискват за това приложение, може да се следи за стабилността на сървъра и приложението. Друго важно свойство на приложението, което също задължително трябва да се провери, е scalability. При него времето за отговор не трябва да зависи от броя потребители. За целта при повече потребители, ако се предоставят повече ресурси (повече инстанции на сървъра), времето трябва да се запази константно. Това свойство отново се проверява с LoadRunner тестовете.

Визуалната част, получена при изпълняване на сценария от уеб браузер е представена на фигура 8 и фигура 9:



Фигура 8. Текстово поле на редактора



Фигура 9. Дефиниране на нова инстанция, спрямо наличната онтология wine.

5. Заключение и бъдещи насоки за развитието на приложението

Тази дипломна работа показа основните концепции на Семантичната мрежа, както и реализацията на приложение, което използва тази технология. Семантичната мрежа предлага големи възможности за машинното обработване на цялата информация във WWW и нейното внедряване и реализацията на приложения са важна стъпка в бъдещото развитие на WWW.

Уеб редакторът, който беше разгледан в тази дипломна работа, може да бъде развиван функционално, както и да бъде оптимизиран.

Тук беше разработен само единия начин, по който уеб редакторът създава страници със семантично съдържание. Това става, като се генерират RDFa документи, които съдържат както HTML, така и разширенията за семантика на съдържанието. Вторият начин за предоставяне на такива документи към външни системи, които работят с технологиите на Семантичната мрежа, е като се предостави реализация на SPARQL интерфейс, с чиято помощ да се предоставят наличните RDF тройки от базата данни на приложението. За този втори начин тука са създадени само предпоставките (всички дефинирани RDF тройки се записват в база данни), но неговата реализация е друго интересно предизвикателство.

Също така графичният интерфейс може да се развие, за да се улесни работата с уеб редактора. Хубаво би било, ако се позволи да се дефинира семантиката на графични и мултимедийни обекти в уеб документите, по същия начин, както това е направено за текста в момента. За целта е необходимо да се включат и други UI технологии освен JSF.

Що се отнася до оптимизацията, която може да се направи, това е използването на AJAX технологията във визуалната част на приложението, която значително ще увеличи бързодействието на приложението.

Речник на използваните термини и съкращения

- World Wide Web (WWW) – система от документи, които са взаимно-свързани чрез хипер-връзки. Тя функционира върху Internet.
- Семантична мрежа (Semantic Web) – еволюция на WWW, в която информацията подлежи на машинна обработка (вместо да се възприема само от хората). Това позволява на браузерите и другите софтуерни агенти да намират, споделят и комбинират информация.
- World Wide Web Consortium (W3C) – основната международна организация за стандартизиране на WWW. Представлява консорциум от членове-организации, които работят заедно целодневно с цел да развиват стандартите във WWW.
- W3C recommendation – последна фаза на развитие на стандарт, разработван от W3C. В този етап стандартът е минал разширен преглед и тестване в теоретични и практически условия и е готов за широка употреба в съответната проблемна област.
- Hypertext Transfer Protocol (HTTP) – протокол за пренос на данни във WWW.
- Uniform Resource Identifier (URI) – компактен низ от символи, използван да идентифицира уеб ресурс.
- Uniform Resource Locator (URL) – подмножество на URI, което специфицира начина на достъп до съответния уеб ресурс, а не само неговото идентифициране във WWW.
- Extensible Markup Language (XML) – markup език, специфициран от W3C, който поддържа широк набор от приложения. Основното му предназначение е да улесни споделянето на данни между различни информационни системи, в частност свързани чрез интернет.
- XML Namespace – стандарт на W3C, осигуряващ уникално наименоване на елементи и атрибути в XML документите, когато те са разпределени в интернет, а не се ползват самостоятелно.

- Qualified name(Qname) – термин, дефиниран в спецификациите XML Schema Part2:Datatypes specification, Namespaces in XML, Namespaces in XML Errata. Стойността на Qname съдържа Namespace URI, local part и prefix.
- Unicode – стандарт, предназначен да позволи символите от всички системи за писане в света да се представят и обработват машинно.
- Литерал - низ от Unicode символи.
- Application Programming Interface (API) – програмен интерфейс, който компютърна система или програма предоставят, за да поддържат протокол на услугите, които предлагат.
- Service Provider Interface (SPI) – софтуерен механизъм за поддръжка на заменяеми компоненти, предоставящи еднакви услуги.
- Resource Description Framework (RDF) – множество спецификации на W3C, които са предназначени за моделиране на знание.
- RDF тройка - основна единица в RDF модела за представяне на минимални твърдения. Изразява се като наредената тройка <субект><предикат><обект>
- Онтология – модел данни, който представя множество от концепции в дадена проблемна област и връзките между тези концепции. Използва се за правене на изводи за обектите в съответната област.
- Dublin Core – онтология, която дефинира термините за цифрови материали като видео, звук, картина, текст и други.
- Resource Description Framework Schema (RDF-S) – разширяем език за представяне на знание, който осигурява основни елементи за описание на онтологии. Онтологиите в RDF се наричат още RDF речници, които трябва да структурират RDF ресурсите.
- Web Ontology Language (OWL) – усложнен език за описание на онтологии. OWL е по-изразителен от RDF-S.
- SPARQL - език за заявки към RDF данни.

- Extensible HyperText Markup Language (XHTML) – markup език, който има същата изразителна сила като HTML, но има по-стриктен синтаксис.
- RDFa – множество от разширения на XHTML, което се специфицира от W3C.
- Rule Interchange Format (RIF) – език за правила върху данните, дефинирани чрез OWL и RDF.
- Java Platform, Enterprise Edition v. 5(Java EE 5) – програмна платформа (част от Java 5 платформата) за разработване на разпределени Java приложения с многослойна архитектура. Те се състоят от софтуерни компоненти, инсталирани върху апликационен сървър.
- JavaServer Pages(JSP), Java Servlet API – Java технология, която позволява на програмистите да генерират динамично HTML, XML или други типове документи в отговор на заявки от уеб компоненти. Тази технология е част от JavaEE 5.0 платформата.
- JavaServer Faces(JSF) – Java-базирана уеб рамка, която опростява разработката на графичен интерфейс за уеб Java EE приложения.
- JavaBean – софтуерен компонент на Java, който следва определени конвенции, за да бъде използван в различни реализации на JavaBean рамки.
- JSF Managed Bean – JavaBean, който се внедрява в JSF рамката и получава контекст, специфичен за JSF.
- Уеб услуга – дефинирана от W3C софтуерна система, предназначена да поддържа взаимодействието между различни системи във WWW. Уеб услугата е API, което се достъпва в интернет и се изпълнява на отдалечена система.
- Simple Object Access Protocol(SOAP) – протокол за обмен на XML-базирани съобщения през компютърна мрежа, обикновено използвайки HTTP. SOAP формира базовото ниво на стека от уеб услуги, осигурявайки основна рамка за съобщения, която се изплолзва от по-горни и абстрактни нива в стека.

- Web Services Description Language(WSDL) – XML-базиран език за описание на модела на уеб услугите. Версия 2.0 се очаква да стане W3C recommendation.
- Java API for XML-based RPC(JAX-RPC) – Java API, което позволява извикване от Java приложение на уеб услуга, ако разполага с нейния WSDL файл.
- Java API for XML Web Services(JAX-WS) – Java API за създаване и извикване на уеб услуги. То наследява JAX-RPC и е част от Java EE 5.
- Java Architecture for XML Binding(JAXB) – позволява на Java програмисти да създават и редактират XML, използвайки Java обекти.
- Java Persistence API – Java рамка, която позволява на програмистите да управляват релационни данни в приложения за Java Platform Standard Edition Java Platform Enterprise Edition.
- Plain Old Java Object (POJO) – термин, представящ идеята, че колкото по-лесен е дизайна на приложение, толкова по-добре. Обектът трябва да е обикновен Java обект, а не специален обект, който трудно се разработва.
- Entity – POJO обект, който представя релация от данни. Java Persistence API предоставя съпоставяне между entity обектите и релационните редове в базата данни.
- Enterprise Java Bean(EJB) – компнонент, който се инсталира на Java EE сървър и се грижи за бизнес логиката на приложението
- Session EJB – EJB компонент, който съществува само в рамките на определена сесия за потребител.
- Java анотация – алтернатива на deployment descriptor, която спестява код по използването на компоненти. Тя служи за инжектиране на различни компоненти и услуги, което се описва в самия код преди да се използват, а не във външни XML файлове.
- JavaServer Pages Standard Tag Library(JSTL, taglib) – компонент на Java EE платформата. Той разширява JSP спецификацията,

добавяйки повече функционалност към JSP таговете за често срещани задачи като XML обработка, изпълнение с условия и цикли, интернационализация и др.

- Model-View-Controller(MVC) – вид дизайнов модел, използван в софтуерното инженерство. Широко се използва в сложни приложения, които представят на потребителите много данни с богат графичен интерфейс. Основната идея е да се отделят данните (модела) от тяхното представяне (изгледа) и обработката на заявките(контролер).
- Model-2 MVC – разновидност на MVC дизайна, специализиран за уеб приложения със спецификите на HTTP протокола.
- Unified Modeling Language(UML) – език за моделиране на софтуерни системи, който има стандартизирано графично представяне за различните аспекти на системите.
- JUnit - рамка за тестване на части на Java приложения.
- LoadRunner – продукт за performance и load тестване за изследване на поведението на системите по време на симулирано натоварване. Създаден е от компанията Mercury Interactive. През ноември 2006 тази компания беше придобита от Hewlett-Packard.
- Sun Java Studio Creator – среда за разработка на Java приложения, създадена от Sun Microsysems.
- SAP NetWeaver Developer Studio – Eclipse-базирана среда за разработка на SAP приложения, създадена от SAP.
- Asynchronous JavaScript and XML (AJAX) – техника за разработване на уеб приложения, която повишава тяхната интерактивност и бързодействие.

Използвана литература

1. Semantic Web: http://www.w3.org/2001/sw/
2. RDF Vocabulary Description Language 1.0: RDF Schema: http://www.w3.org/TR/rdf-schema/
3. RDF Primer: http://www.w3.org/TR/rdf-primer/
4. OWL Web Ontology Language Overview: http://www.w3.org/TR/owl-features/
5. OWL Web Ontology Language Guide: http://www.w3.org/TR/owl-guide/
6. Nigel Shadbolt, Tim Berners-Lee, Wendy Hall, "The Semantic Web Revisited": http://www.w3.org/TR/owl-guide/
7. Tim Berners-Lee, James Hendler, Ora Lassila, Scientific American, May 2001, "The Semantic Web": http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2
8. Ivan Herman, "Introduction to the Semantic Web": http://www.w3.org/2006/Talks/0524-Edinburgh-IH/#(1)
9. Eric Miller, Norwegian Semantic Days 2006, "The Semantic Web: Overview":http://www.w3.org/2006/Talks/0426-semweb-em/
10. Tim Berners-Lee, "Putting the Web back to Semantic Web": http://www.w3.org/2005/Talks/1110-iswc-tbl/#(1)
11. Rules Interchange Format Working Group: http://www.w3.org/2005/rules/wg
12. Semantic Web Best Practices and Deployment Working Group: http://www.w3.org/2001/sw/BestPractices/
13. Aaron Swartz, "The Semantic Web in Breadth": http://logicerror.com/semanticWeb-long
14. Java Platform Enterprise Edition (Java EE) Specification, v5: http://java.sun.com/javaee/technologies/javaee5.jsp
15. Rima Patel Sriganesh, Gerald Brose, Micah Silverman, "Mastering Enterprise Java Beans 3.0"

16. Kito D. Mann, "JavaServer Faces in Action"
17. Bill Dudney, Jonathan Lehr, Bill Willis, LeRoy Mattingly, "Mastering JavaServer Faces"
18. Enterprise Java Technologies Tech Tips: http://java.sun.com/developer/EJTechTips/
19. Getting Started with Protégé OWL: http://protege.stanford.edu/doc/owl/getting-started.html
20. Wikipedia: http://www.wikipedia.org/