Міністерство освіти України
Запорізький державний університет

ДО ЗАХИСТУ ДОПУЩЕНО
Зав.кафедрой ММтаІТ, доцент

(підпис)
С.Ю. Борю
(ім'я, по батькові, прізвище)

(дата)

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Методіка та інструментарій для колективної розробки онтологій на Web

Виконав Петручек Валентин Вікторович

ст. групи 8228-2 (шифр) (підпис і дата) В.В. Петручек (ім'я, по батькові, прізвище)

Керівник доц., к. ф.-м. н. (посада) (підпис і дата) В.А. Єрмолаєв (ім'я, по батькові, прізвище)

Нормоконтролер (підпис) Н.Б. Александрова (ім'я, по батькові, прізвище)

Запоріжжя,
2002

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ

РЕФЕРАТ

Дипломная работа: 39 страниц, 3 рис., 3 приложения, 6 источников.

Целью дипломной работы является разработка методики и инструмента для коллективного создания онтологий.

При выполнении проекта были поставлены и решены следующие задачи:

- разработать модель хранения онтологии, оптимизированную для выполнения операций коллективной работы с онтологией;
- разработать формальный и алгоритмический аппарат трансляции онтологии на входном языке (OIL) в язык модели хранения;
- адаптировать и формализовать социальную модель голосования, применив ее к коллективной разработке онтологии;
- реализовать интерфейсы механизма взаимодействия членов сообщества посредством web-технологий.

ОНТОЛОГИЯ, OIL, КОЛЛЕКТИВНАЯ РАЗРАБОТКА, РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ, МЕТОДИКА РАЗРАБОТКИ, СООБЩЕСТВО РАЗРАБОТЧИКОВ.

Содержание

ВВЕДЕНИЕ

Термин "онтология" имеет философское происхождение. В философии онтология – учение о бытии вообще, бытии как таковом, независимо от его частных видов.

Одно из первых определений онтологии в информационных технологиях было дано Neches[1]. Согласно Neches, онтология определяет базовые понятия предметной области и отношения между ними, а также правила для составления новых понятий и отношений для расширения словаря.

Это определение позволяет нам сформулировать алгоритм для построения онтологии:

1) определить базовые понятия и отношения между ними;
2) определить правила для их сочетаний;
3) предоставить определения базовых понятий и отношений.

Очень важным в данном определении является то, что онтология состоит не только из уже определенных понятий и отношений между ними, но и из понятий, которые могут быть получены из них применением определенных правил.

Наиболее популярным определением онтологии является следующее определение, данное Gruber[2]: онтология - формальная спецификация концептуализации. Оба эти определения были объяснены Studer[3] следующим образом: "концептуализация обращается к абстрактной модели некоторого явления реального мира, определяя релевантные понятия этого явления. Типы используемых понятий и ограничений явно определены. Онтология должна быть формальной, т.к. она предназначена для чтения машинами. Онтология представляет собой всеобщее знание, т.е. принимаемое некоторой группой, а не индивидом".

Зачем нужны онтологии вообще? Для компьютерного анализа информации необходимо, чтобы программное обеспечение обладало информацией о смысле, т.е. о понятиях и взаимоотношениях между ними той предметной области, в которой эти программы оперируют. Концептуально важным здесь является возможность определения программами общего смысла одного и того же предмета.

Онтологии используются в Collaborative Distributive Problem Solving – в системах с распределенными автономными решателями задач, например в управлении производством, электронной коммерции, экспертных системах, электронных библиотеках и дистанционном обучении. Онтологии нужны для того, чтобы автономные программы могли взаимодействовать друг с другом на смысловом уровне, т.е. оперировать понятиями и отношениями реального мира.

С другой стороны, нельзя автоматизировать процесс концептуализации знаний о явлениях и процессах реального мира. Это весьма и весьма трудоемкий процесс. Так, например, онтология ACM публикаций насчитывает 2500 концептов, и при этом не относится к классу больших онтологий.

Онтологии, используемые при решении практических задач (Upper Cyc® онтология, WordNet онтология), состоят из нескольких тысяч концептов.

Одному человеку для составления подобной онтологии потребуется значительный промежуток времени. При этом составленная одним человеком онтология будет субъективной, что противоречит одному из требований, предъявляемым к онтологиям.

Решить обе эти проблемы позволил бы инструмент, позволяющий сообществу людей совместно создавать онтологию, добавляя в нее концепты и отношения между ними, утверждать или отвергать уже существующие элементы онтологии.

То небольшое количество систем разработки онтологий, которые существуют сегодня, в большой степени ориентированы на определенный круг задач.

Актуальность работы заключается в разработке способа представления онтологий произвольных предметных областей с помощью реляционной модели данных а также внедрение механизмов совместной работы над онтологией.

Использование веб-технологий обеспечивает широкую применимость разработанного инструментария, а применение социальной модели голосования в отношении к сообществу разработчиков онтологии повышает уровень объективности разрабатываемой онтологии.

Таким образом, создание методики для коллективного построения и разработку инструмента, реализующую эту методику в среде Web, можно считать оправданной.

1 ОБЗОР МЕТОДИК КОЛЛЕКТИВНОЙ РАЗРАБОТКИ ОНТОЛОГИЙ

1.1 КЛАССИФИКАЦИЯ ОНТОЛОГИЙ. ОСНОВНЫЕ СВОЙСТВА

Выделяют такие основные типы онтологий:

a) онтологии представления знаний;

b) онтологии верхнего уровня;

c) лингвистические онтологии;

d) общие (независимые от предметной области) онтологии;

e) онтологии предметной области.

При разработке онтологии необходимо следовать следующим правилам:

- ясность и объективность;
- полнота;
- согласованность;
- максимальная монотонная расширяемость;
- минимальное число соглашений;
- принцип онтологической обособленности;
- многообразие иерархий;
- модульность;
- минимизация семантического расстояния между родственными концептами;
- стандартизация имен.

1.2 МЕТОДИКИ ПОСТРОЕНИЯ И КОЛЛЕКТИВНАЯ РАЗРАБОТКА ОНТОЛОГИЙ

Рассматривают 5 основных методик построения онтологий:

a) методики построения онтологий «с нуля»;

b) методики для переработки онтологий;

c) методики для коллективного построения онтологий;

d) методики построения онтологий путем изучения других онтологий;

e) методы оценки (проверки) онтологий.

Рассмотрим методики для коллективной разработки методологий подробнее.

Онтология является распределенной и общей понятийной базой некоторой предметной области. Для коллективной разработки онтологий необходимо достижение соглашения о содержимом онтологии, в том смысле, что группа людей соглашается с формальной спецификацией концептов, отношений между ними, атрибутов и аксиом онтологии. Проблема коллективной (т.е. проводимой группой людей) и распределенной (т.е. проводимой удаленными друг от друга людьми) разработки онтологий до сих пор не решена.

Выделяют следующие проблемы, возникающие при коллективной распределенной разработке онтологий:

1) обеспечение взаимодействия и общения между людьми;
2) контроль над доступом к данным;
3) обнаружение и устранение ошибок;
4) контроль за параллельным изменением данных.

На основании этих проблем было выдвинуто несколько предложений о том, как строить онтологии коллективно. На данный момент наиболее известными являются инициативы CO4 и $(KA)^2$.

CO4 представляет из себя протокол для достижения консенсуса между несколькими базами знаний. Его цель – предоставить разработчикам возможность обсуждать и представлять знания, хранящиеся в базах знаний.

Эти базы знаний являются распределенными, они хранят согласованные знания и поэтому могут считаться онтологиями.

К достоинствам CO4 можно отнести:

1) представление онтологий в виде баз знаний;
2) совместное хранение баз знаний;
3) разделение концептов онтологии на общие и частные.

К недостаткам CO4 относятся:

1) отсутствие возможности анализа подобия конкурирующих концептов частных отнологий;
2) отсутствие механизма развития онтологий.

Методика $(KA)^2$ направлена на моделирование сообщества по разработке онтологии группой распределенных людей на основе общих шаблонов. К ее достоинствам можно отнести разработанную модель представления онтологии в виде базы данных, а также организованный протокол обмена сообщениями между группами, разрабатывающими онтологию. Недостатком данной методики является привязка к одной предметной области – сообщества исследователей, занимающихся инженерией знаний.

Данная работа является актуальной, поскольку в ней реализованы:

- представление онтологий произвольных предметных областей с помощью реляционной модели данных;
- механизмы совместной работы над онтологией в т.ч.:
  1) анализ подобия конкурирующих концептов и принятие решения об их отношениях;
  2) механизм поддержки принятие решений о структуре онтологий;

В то же время, данная работа отличается от аналогичных разработок:

- использование веб-технологий обеспечивает широкую применимость разработанного инструментария;
- использование социальной модели голосования в применении к сообществу разработчиков онтологии.

Целью дипломной работы является разработка методики и инструмента для коллективного создания онтологий.

При выполнении проекта были поставлены и решены следующие задачи:

- разработать модель хранения онтологии, оптимизированную для выполнения операций коллективной работы с онтологией;
- разработать формальный и алгоритмический аппарат трансляции онтологии на входном языке (OIL) в язык модели хранения;
- адаптировать и формализовать социальную модель голосования, применив ее к коллективной разработке онтологии;
- реализовать интерфейсы механизма взаимодействия членов сообщества посредством web-технологий.

2 МОДЕЛЬ ПРЕДСТАВЛЕНИЯ ОНТОЛОГИИ

2.1 ФОРМАЛЬНАЯ СПЕЦИФИКАЦИЯ ЯЗЫКА ПРЕДСТАВЛЕНИЯ ОНТОЛОГИЙ OIL

Для представления онтологий используются различные языки. В первую очередь это языки представления знаний – KIF (www.csee.umbc.edu/kse/kif/), Ontolingua (ontolingua.stanford.edu), OKBC (http://www.ksl.stanford.edu/software/OKBC), SHOE (www.cs.umd.edu/projects/plus/SHOE).

Также разработаны языки для представления непосредственно онтологий: OIL (www.ontoknowledge.org/oil/), RDF и RDFS (www.w3.org/TR/rdf-schema), OML (www.ontologos.org/OML/OML 0.3.htm), DAML+OIL (www.daml.org/language/). В данной работе был использован язык OIL в качестве языка для представления онтологий поскольку он:

a) обладает богатым набором семантических средств для представления концептов онтологии;

b) ориентирован на фреймовую модель данных, наиболее распространенную на данный момент;

c) близок по своей выразительности (средствам языка) к языкам представления знаний;

d) снабжен быстрой машиной вывода как и языки представления знаний.

Синтаксис языка OIL представлен в виде нормальной формы Бэкуса в Приложении А.

Любой формальный язык представляет собой множество цепочек в некотором конечном алфавите. В лингвистике вместо термина «алфавит» употребляется термин «словарь», так как элементы, из которых он составлен, представляют собой слова, или, точнее, словоформы. В то же время цепочка

над словарем рассматривается как словосочетание или предложение. К формальным языкам относятся, в частности, искусственные языки для общения человека с машиной — языки программирования. Для задания описания формального языка необходимо, во-первых, указать алфавит, т. е. совокупность объектов, называемых символами (или буквами), каждый из которых можно воспроизводить в неограниченном количестве экземпляров (подобно обычным печатным буквам или цифрам), и, во-вторых, задать формальную грамматику языка, т. е. перечислить правила, по которым из символов строятся их последовательности, принадлежащие определяемому языку,— правильные цепочки.

Заметим, что каждый символ алфавита рассматривается как неделимый в том смысле, что при построении цепочек никогда не используются его графические элементы (части символа) и всякая последовательность символов однозначно представляет некоторую цепочку. Практически это требование достигается, например, путем установления пробела (промежутка стандартной длины) между символами, который превышает длину любого из промежутков, встречающихся внутри символов алфавита.

Правила формальной грамматики можно рассматривать как продукции (правила вывода) — элементарные операции, которые, будучи применены в определенной последовательности к исходной цепочке (аксиоме), порождают лишь правильные цепочки. Сама последовательность правил, использованных в процессе порождения некоторой цепочки, является ее выводом. Определенный таким образом язык представляет собой формальную систему. Известными примерами формальных систем служат логические исчисления (исчисление высказываний, исчисление предикатов), которые подробно изучаются в соответствующих разделах математической логики.

По способу задания правильных цепочек формальные грамматики разделяются на порождающие и распознающие. К порождающим относятся грамматики, по которым можно построить любую правильную цепочку с

указанием ее структуры и нельзя построить ни одной неправильной цепочки. Распознающая грамматика — это грамматика, позволяющая установить, правильна ли произвольно выбранная цепочка и, если она правильна, выяснить ее строение.

Формальные грамматики широко применяются в лингвистике и программировании в связи с изучением естественных языков и языков программирования.

Определение. Грамматика $G = (\mathrm{A}, V_n, \sigma, P)$ - порождающая грамматика, где $\mathrm{A} = \{a_1, a_2, ..., a_m\}$ - основной (терминальный) алфавит, V_n - конечный вспомогательный (нетерминальный) алфавит, символы которого обозначаются малыми греческими буквами, $\sigma \in V_n$ - начальный (нетерминальны) символ – аксиома, $P = \{\varnothing_i \to u_i\}, i = 1,2,..., k$ - конечная система подстановок (схема грамматики), левые и правые части которых есть цепочки $u_i, v_i \in F(v)$, где $F(v)$ - свободная полугруппа над объединенным алфавитом $V = V_n \cup \mathrm{A}$, символ $\to$ является внешним и не принадлежит объединенному алфавиту.

Определение. Грамматика $G = (\mathrm{A}, V_n, \sigma, P)$ называется контекстно-свободной, если каждое правило схемы P имеет вид $\psi \to z$, где $\psi \in V_n$, а z - непустая цепочка над объединенным алфавитом $V = V_n \cup \mathrm{A}$.

Определение *Многоосновная алгебра* – система $< \mathrm{M}, \Omega >$, состоящая из семейства множеств $\mathrm{M} = \{A_\alpha : \alpha \in I\}$, которые называются основными, и сигнатуры операций Ω, определенных на M следующим образом: каждой n-местной операции однозначно сопоставлен кортеж $<\alpha_1, ..., \alpha_n; \alpha_r>$ - схема данной операции, так что $F_{<\alpha_1, ..., \alpha_n; \alpha_r>}$ является отображением декартова произведения $A_{\alpha_1} \times A_{\alpha_2} \times ... \times A_{\alpha_n}$ во множество A_{α_r}, где $\alpha_i, \alpha_r \in I (i = 1, ..., n)$.

Таким образом, операции $F_{<\alpha_1, ..., \alpha_n; \alpha_r>} \in \Omega$ являются n-местными функциями $F(x_1, ..., x_n)$, аргументы которых $x_q (q = 1, ..., n)$ определены на множествах $A_{\alpha_q} \in \mathrm{M}$,

а сама функция F принимает значения на множестве. Каждая 0-местная операция $F_{<\alpha_r>} \in \Omega$ фиксирует в $A_{\alpha_r} \in \mathrm{M}$ некоторый элемент (константу) $a \in A_{\alpha_r}$.

Для любой кс - грамматики ([4]) $G = (\mathrm{A}, V_n, \sigma, P)$ может быть построена многоосновная алгебра $\mathrm{M}(G) = \{A_\psi : \psi \in V_n\}$ с выделенной компонентой A_σ, $L(G) = A_\sigma$ (т.е. цепочки L - элементы множества A_σ).

Алгебра строится следующим образом: рассматривается основное множество $\mathrm{M}(G) = \{A_\psi : \psi \in V_n\}$, A_ψ - множество всех терминальных цепочек, выводимых из нетерминала ψ в грамматике G.

Каждой кс-продукции $p : \psi_i \to u_{i_1}\psi_1^i u_{i_2}\psi_2^i ... u_{i_{r_i}}\psi_{r_i}^i \bullet u_{i_{r_i+1}} \in P$ поставим в соответствие конкатенарную операцию $\omega_p : A_{\psi_1^i} \times A_{\psi_2^i} \times ... \times A_{\psi_{r_i}^i} \to A_{\psi_i}$, определенную на множествах $A_{\psi_1^i}, ..., A_{\psi_{r_i}^i}$ и такую, что при подстановке вместо соответствующих аргументов произвольных цепочек $x_s \in A_{\psi_s^i} (1 \le s \le r_i)$ значением операции ω_p является цепочка

$$x = u_{i_1} x_1 u_{i_2} x_2 ... u_{i_{r_i}} x_{r_i} u_{i_{r_i+1}} \in A_{\psi_i}$$

Конкатенарные операции ω_p, ассоциированные со схемой P кс-грамматики G, составляют сигнатуру Ω. Построенная таким образом многоосновная алгебра $< \mathrm{M}(G), \Omega >$ с выделенной основной компонентой A_σ, соответствующей аксиоме σ грамматики G, называется кс-многоосновной алгеброй, ассоциированной с грамматикой G.

Определение порождающее алгебры для заданной грамматики языка позволяет использовать аппарат общей алгебры для анализа данного языка, а именно выделение обязательных конструкций языка, выявление эквивалентных конструкций, позволяет анализировать и оптимизировать тексты, написанные на данном языке.

Построим грамматику для языка Standard OIL.

Формально грамматика языка Standard-OIL представима следующим образом

$$G_{S-OIL} = (\mathrm{A}, V_n, \sigma, P)$$

где A={begin-ontology, end-ontology, ontology-container, title, creator, subject, description, description.release, publisher, contributor, date, type, format, identifier, source, language, relation, rights, ontology-definitions, import, class-def, documentation, primitive, defined, subclass-of, top, thing, and, or, not, one-of, (,), slot-constraint, has-filler, has-value, value-type, max-cardinality, min-cardinality, cardinality, integer, string, min, max, greater-than, less-than, equal, slot-def, subslot-of, domain, range, inverse, properties, transitive, symmetric, functional, disjoint, covered, disjoint-covered, equivalent}

V_n={<ontology>, <container>, <definitions>, <import-exp>, <class-definition>, <type>, <subclass>, <class-exp>, <set-exp>, <slot-constraint>, <constraint-exp>, <filler-exp>, <filler>, <has-value-exp>, <value-type-exp>, <max-cardinality-exp>, <min-cardinality-exp>, <cardinality-exp>, <expression>, <concrete-type-exp>, <int-exp>, <int-pred>, <string-exp>, <string-pred>, <slot-definition>, <subslot-exp>, <slot-def-exp>, <domain-exp>, <range-exp>, <inverse-exp>, <property-list>, <property>, <axiom>, <disjoint-exp>, <cover-exp>, <disjoint-cover-exp>, <equiv-exp>, <class-name>, <slot-name>, <ind-name>, <URI>, <symbol> }

σ = <ontology>

P - список правил грамматики, представленный в Приложении А.

Построим теперь порождающую алгебру по данной кс-грамматике G_{S-OIL}. Алгебра $<\mathrm{M}(G), \Omega>$ будет контекстно-свободной, т.к. в левых частях правил из P содержатся только нетерминальные символы, а во-вторых, многоосновной.

В качестве основных множеств возьмем семейство множеств $\mathrm{M}(G) = \{A_\psi : \psi \in V_n\}$

Определим множество операций Ω алгебры таким образом:

$$\varpi_{ontology} : A_{container} \times A_{definitions} \to A_{ontology}$$

$$\varpi_{definitions} : A_{import-exp} \times A_{definition} \to A_{definitions}$$

$$\varpi_{import-exp} : A_{uri} \to A_{import-exp}$$

$$\varpi_{definition} : A_{class-definition} \times A_{slot-definition} \times A_{axiom} \to A_{definition}$$

$$\varpi_{class-definition} : A_{type} \times A_{class-name} \times A_{subclass} \times A_{slot-constraint} \to A_{class-definition}$$

$$\varpi_{type} : \{primitive, defined\} \to A_{type}$$

$$\varpi_{subclass} : A_{class-exp} \to A_{subclass}$$

$$\varpi_{class-exp}^{class-name} : A_{class-name} \to A_{class-exp};\ \varpi_{class-exp}^{set-exp} : A_{set-exp} \to A_{class-exp};\ \varpi_{class-exp}^{slot-constraint} : A_{slot-constraint} \to A_{class-exp}$$

$$\varpi_{class-exp}^{\{and,or\}} : A_{class-exp} \times A_{class-exp} \to A_{class-exp};\ \varpi_{class-exp}^{\{not\}} : A_{class-exp} \to A_{class-exp}$$

$$\varpi_{set-exp} : A_{ind-name} \to A_{set-exp}$$

$$\varpi_{slot-constraint} : A_{slot-name} \times A_{constraint-exp} \to A_{slot-constraint}$$

$$\varpi_{constraint-exp}^{has-value-exp} : A_{has-value-exp} \to A_{constraint-exp};\ \varpi_{constraint-exp}^{value-type-exp} : A_{value-type-exp} \to A_{constraint-exp};$$

$$\varpi_{constraint-exp}^{\max-cardinality-exp} : A_{\max-cardinality-exp} \to A_{constraint-exp};\ \varpi_{constraint-exp}^{\min-cardinality-exp} : A_{\min-cardinality-exp} \to A_{constraint-exp};$$

Грамматика состоит из 45 нетерминальных символов, с выделенным символом <ontology>, и 63 терминальных символов.

2.2 ВЫБОР МОДЕЛИ ХРАНЕНИЯ ДАННЫХ

В пункте 2.1 было определено, что предметной областью, с которой нам предстоит работать, являются тексты, написанные на определенном языке – языке Standard OIL.

Модель для хранения данных должна отвечать следующим критериям:

- наличие удобных инструментов для хранения и доступа к данным;
- универсальность и широкое использование модели;
- наличие формальной математической основы для хранения и доступа к данным;
- простота концепции модели;
- удобство работы с моделью из веб-приложения.

Наиболее полно данным критериям удовлетворяет реляционная модель данных, основные положения которой описаны в разделе 2.3.

2.3 ОСНОВНЫЕ ПОЛОЖЕНИЯ РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ

Реляционная модель данных была предложена Коддом [5] в 1970г. Он показал, что набор “таблиц” (relations) может быть использован для моделирования взаимосвязей между объектами реального мира и для хранения данных об этих объектах. Такой подход к представлению данных в виде таблиц приобрел огромную популярность благодаря простоте основных идей.

В 1976 г. Чен [6] предложил улучшение реляционной модели данных, которую он назвал “Entity - Relationship Model” (модель “Сущность” -

“Отношение”). В этой работе предлагалось представлять предметную область задачи двумя типами отношений.

Первый тип описывает некую сущность (entity) - объектное отношение.

Второй тип описывает связи между сущностями предметной области (relationships) - отношения связей.

Дадим неформальное определение терминов “сущность” и “связь”:

- сущность представляет собой любой отличимый объект с той степенью абстракции, какая требуется в данной задаче.
- связь представляет собой ассоциирование двух или более сущностей.

В общем случае связи бывают двух типов: “многие ко многим” и “многие к одной”. Связи типа “многие к одной “ называются характеристическими, связи типа “многие ко многим” называются ассоциативными.

Реляционная модель данных (РМД) - особый способ видения данных, т.е. предписание, определяющее, каким образом следует представлять данные и как ими манипулировать. Рассмотрим основные пункты, отличающие эту модель от других.

2.3.1 РЕЛЯЦИОННАЯ СТРУКТУРА ДАННЫХ

Наименьшая единица данных в РМД - отдельное значение данных. Такое значение рассматривается как атомарное, т.е. неразложимое на другие, когда дело касается данной задачи.

Домен - множество значений данных одного типа. Значения одного домена могут сравниваться между собой, значения из разных доменов - в общем случае - нет. Атрибут - именованный домен, представляющий семантически значимый объект.

Имя атрибута - символическое имя данных, характеризующих отдельное свойства объекта из предметной области. В дальнейшем будет употребляется слово атрибут с тем же смыслом.

Кортеж - множество значений , взятых по одному для каждого значения атрибута из схемы отношения.

t - кортеж - $< A_i{:}a_i >$, $i = \overline{1,n}$

Пользуясь терминологией Мейера [22], сформулируем следующие определения:

Схема отношения *R* - конечное множество имен атрибутов $\left\{A_1,...,A_n\right\}$

Каждому имени атрибута A_i ставится в соответствие множество D_i - конечное или бесконечное множество допустимых значений атрибута - домен: $dom(A_i) = D_i = \varnothing$, $i = \overline{1,n}$

Отношение *r* со схемой *R, r(R)* - конечное множество отображений, $\left\{t_1...t_p\right\}$, $t_j : R \to D$, где $D = D_1 \bigcup ... \bigcup D_n$ - домен схемы *R*, t_j - кортеж, $t_j \in r \Rightarrow t_j(A_i) \in D_i, i = \overline{1,n}, j = \overline{1,p}$;

p - кардинальное число отношения *r*, *n* - степень отношения *r*.

Т.о., отношение определяется математически как множество кортежей, и это множество является подмножеством декартова произведения фиксированного числа доменов. В терминах концептуального моделирования отношение- некоторый тип сущностей. Для того, чтобы использовать в качестве типов сущностей математические отношения, необходимо выполнение двух требований:

Требование 1. В каждом кортеже должно быть одно и тоже число атрибутов, значения каждого атрибута выбираются только из соответствующего атрибуту домена.

Требование 2. Поскольку множество не может иметь совпадающих элементов, а кортежи можно различать только по значениям их компонент, то никакие два различных кортежа не могут иметь полностью совпадающих значений атрибутов.

2.3.2 ОБЕСПЕЧЕНИЕ ЦЕЛОСТНОСТИ ДАННЫХ

Каждая сущность предметной области представляет собой описание целого класса объектов одного типа. Очевидно, что должны существовать

правила, позволяющие отличить объекты одного типа один от другого, поскольку эта отличимость - одно из основных требований при описании предметной области задачи.

В реляционной модели для этого используется *механизм ключей.*

Пусть $R[A_1,...,A_n]$ - схема отношения r.

Говорят, что ключ отношения r со схемой R - подмножество K:

$$K = \{B_1, B_2, ..., B_m\} \subseteq R: \ \forall t_1, t_2 \in r: \ t_1(K) \neq t_2(K)$$

и ни одно собственное подмножество $K' \subset K$ не обладает этим свойством.

Сформулируем 2 свойства ключа.

Первое свойство ключа - *уникальность*: “Не существует двух кортежей, имеющих одно и то же значение ключа на всех атрибутах из K”

Второе свойство - *минимальность:* “Ни один из атрибутов $B_1,...,B_m$ не может быть исключен из K без нарушения условия уникальности”

Поскольку любое отношение в реляционной модели данных должно удовлетворять требованию - не иметь одинаковых кортежей, то любое отношение обладает по крайней мере одним ключом - комбинацией всех его атрибутов.

Один произвольно выбранный ключ для данного отношения принимается за его первичный ключ. Остальные возможные ключи называются альтернативными.

Таким образом каждое отношение имеет свой первичный ключ. Если отношение описывает сущность, то этот ключ служит для идентификации конкретного объекта данной сущности. Отношение связи - связывает ключи двух и более сущностей.

Внешний ключ - это атрибут или комбинация атрибутов одного отношения, значение которого обязательно должно совпадать со значением первичного ключа некоторого другого отношения. Т.е. любое отношение связи содержит один или более внешних ключей некоторых сущностей.

Для того, чтобы набор сущностей и связей адекватно описывал предметную область задачи, необходимо, чтобы данные в разных отношениях модели данных были согласованы между собой.

Поскольку база данных должна описывать изменяющийся мир, то состояние отношений и связей между объектами в каждый момент времени может изменяться. Значит, эти изменения должны осуществляться так, чтобы в каждый момент времени данные были непротиворечивы. Требование непротиворечивости должна обеспечивать целостность данных.

Сформулируем правила целостности для реляционной модели данных:

1) целостность по сущностям.

Не допускается, чтобы какой-либо атрибут, участвующий в первичном ключе объектного отношения, принимал неопределенные значения. Если бы первичный ключ принимал неопределенные значения, это говорило бы о том, что есть сущность, которая не обладает индивидуальностью.

2) целостность по ссылкам.

Если объектное отношение r_2 включает некоторый внешний ключ FK , соответствующий первичному ключу PK какого-либо объектного отношения r_1, то каждое значение FK в r_2 должно либо:

а) быть равным значению PK в некотором кортеже r_1,

либо

б) быть полностью неопределенным, т.е. каждое значение атрибута, участвующего в этом значении FK, должно быть неопределенным.

(При этом r_1, r_2 - не обязательно различные объектные отношения)

Таким образом, основными компонентами реляционной модели данных являются:

а) структура данных

Домены, n-арные отношения (атрибуты , кортежи)

Ключи (возможные, первичные, альтернативные, внешние)

б) целостность данных

1) значения первичных ключей не должны быть неопределенными.

2) значения внешних ключей должны соответствовать значениям первичных ключей (или быть неопределенными)

в) манипулирование данными

г) реляционная алгебра B и реляционное присваивание.

2.4 ПОСТРОЕНИЕ РМД ДЛЯ ХРАНЕНИЯ ОНТОЛОГИЙ

Для хранения онтологии в реляционной модели данных была построена диаграмма, представленная на рис. 2.1.

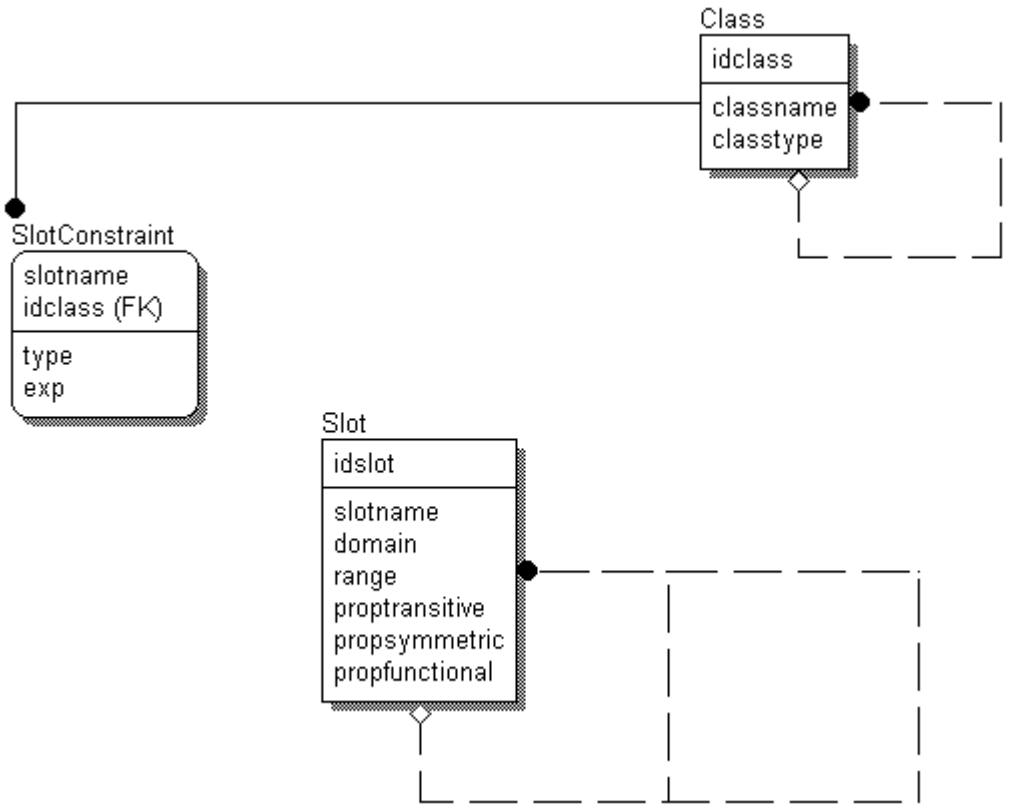

Рисунок 2.1 - Диаграмма РМД для хранения онтологий

В качестве СУБД для хранения онтологий была выбрана СУБД MySQL (www.mysql.com) поскольку она:

a) поддерживает стандартный диалект языка Transact SQL;

b) распространяется бесплатно;

c) широко распространена среди пользователей;

d) поддерживает быстрое взаимодействие с веб-приложениями.

В среде MySQL разработанная реляционная модель данных представляется с помощью 6 таблиц, структура которых представлена в Приложении Б.

3 СОЦИАЛЬНАЯ МОДЕЛЬ ГОЛОСОВАНИЯ

Рассмотрим следующее сообщество разработчиков онтологии. Пусть $R_1, R_2, \ldots, R_n$ - разработчики (каждый из них идентифицируется своим номером). $w_1, w_2, \ldots, w_n$ - веса их голосов.

$C_1, C_2, \ldots, C_k$ - элементы онтологии, за которые отдают свои голоса разработчики. Начальные оценки элементов равны: $d_k = 50$.

Начальные значения весов равны у всех разработчиков: $w_i = 100$.

Пусть m_{ij} – оценка, которую поставил разработчик i элементу j.

Тогда с учетом веса разработчика эта оценка равна $m_{ij} * w_i$.

Пусть элементу k_o поставили свои оценки разработчики $i_1, i_2, \ldots, i_p$.

Тогда его оценка (уровень стабильности) может быть получена следующим образом:

$$g_{k_0} = \begin{cases} \frac{1}{2}\left(\frac{\sum_{j=1}^{p} m_{i_j k_0} * w_{i_j}}{p} + d_k \right), p \neq 0 \\ 0, p = 0 \end{cases}$$

Эта оценка учитывает предыдущую оценку элемента.

Относительная ошибка при оценке элемента k_0 разработчиком i равна:

$$p_{ik_0} = \frac{\left| m_{ik_0} - g_{k_0} \right|}{m_{ik_0}}$$

За рассматриваемый период суммарная ошибка разработчика i равна:

$$v_i = \sum_k p_{ik}$$

Суммирование производится по тем элементам *k,* которые разработчик оценивал. Средняя ошибка за период (по всем разработчикам) равна:

$$s = \sum_{i=1}^{n} v_i$$

Теперь мы можем вычислить ошибку разработчика *i* в сравнении с остальными разработчиками:

$$z_i = s - v_i$$

Сравнительная ошибка разработчика может быть положительной, если он ошибался меньше, чем «средний» разработчик, и отрицательной в противном случае. В первом случае разработчик заслуживает поощрения, во втором – наказания. Средством поощрения/наказания является изменение веса голоса разработчика. Для перехода к следующему этапу необходимо изменить оценки элементов и веса разработчиков:

$$d_k = g_k,$$
$$w_i^{/} = w_i(1 + z_i)$$

Данная система оценки элементов позволяет динамически отслеживать изменение «статуса» онтологии и ее частей, отслеживать стабильные и нестабильные ее элементы. При этом часть онтологии (подграф) может считаться стабильной, если все ее элементы (вершины) стабильны.

Также эта система стимулирует разработчиков принимать участие в голосовании, поскольку в случае пассивного участия разработчик будет терять свой вес за счет отрицательной оценки в сравнении с остальными разработчиками.

4 ИНТЕРФЕЙСЫ МЕХАНИЗМА ВЗАИМОДЕЙСТВИЯ РАЗРАБОТЧИКОВ

Система предоставляет такие интерфейсы для разработчиков:

- импорт онтологий на языке Standard Oil;
- визуализация всей онтологии;
- визуализация части онтологии;
- обсуждение элемента онтологии или обсуждение произвольного вопроса с помощью форума;
- голосование за элемент онтологии;
- добавление нового элемента онтологии и связь его с уже существующими;
- экспорт онтологии (части онтологии) на языке Standard Oil;
- добавление сообщения в гостевую книгу.

Система разработана на языке программирования PHP (www.php.net) версии 4 с использованием графической библиотеки GD версии 2.0.

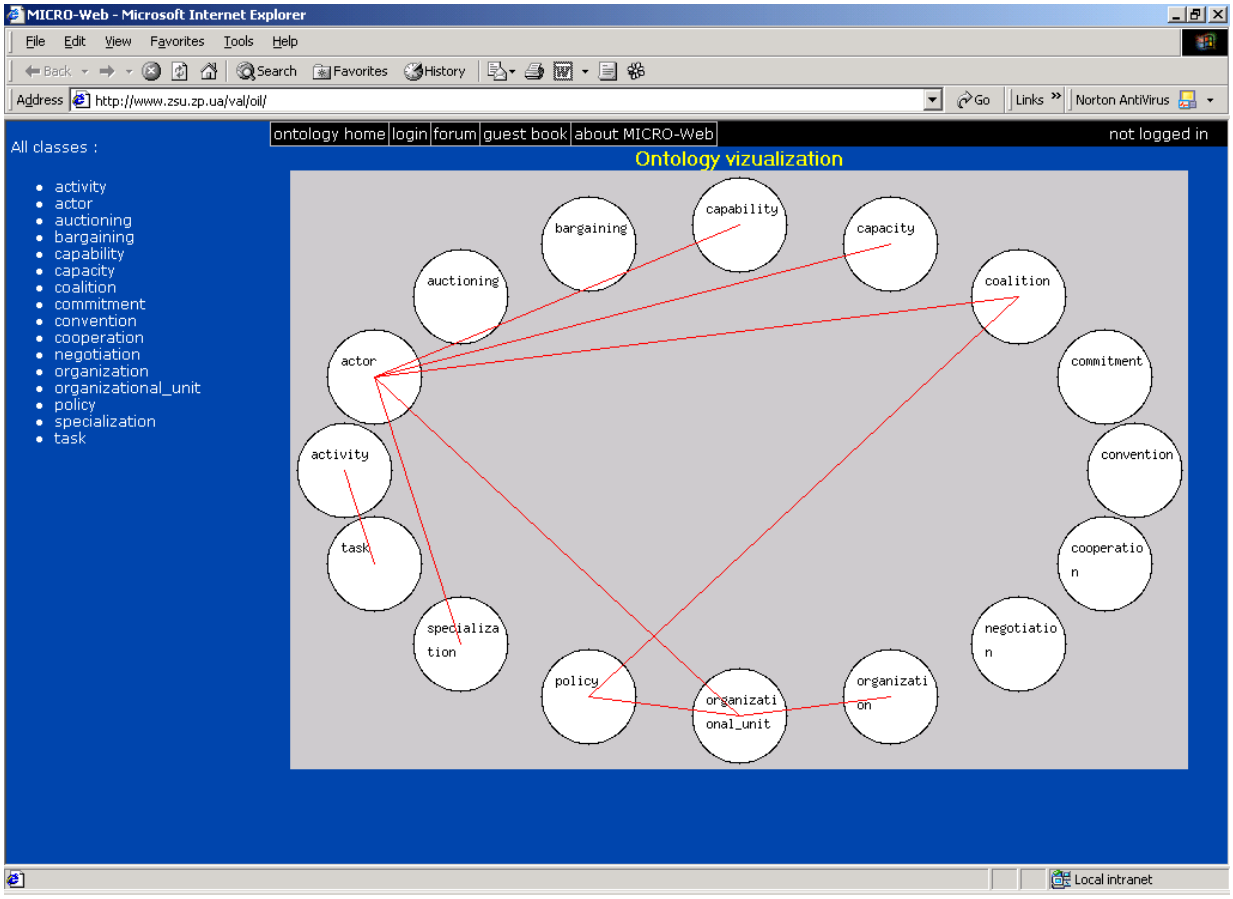

Рисунок 4.1 Визуализация всей онтологии

На рис. 4.1 представлена онтология, хранящаяся в базе данных, в графическом виде. Тексты исходных программ представлены в Приложении В.

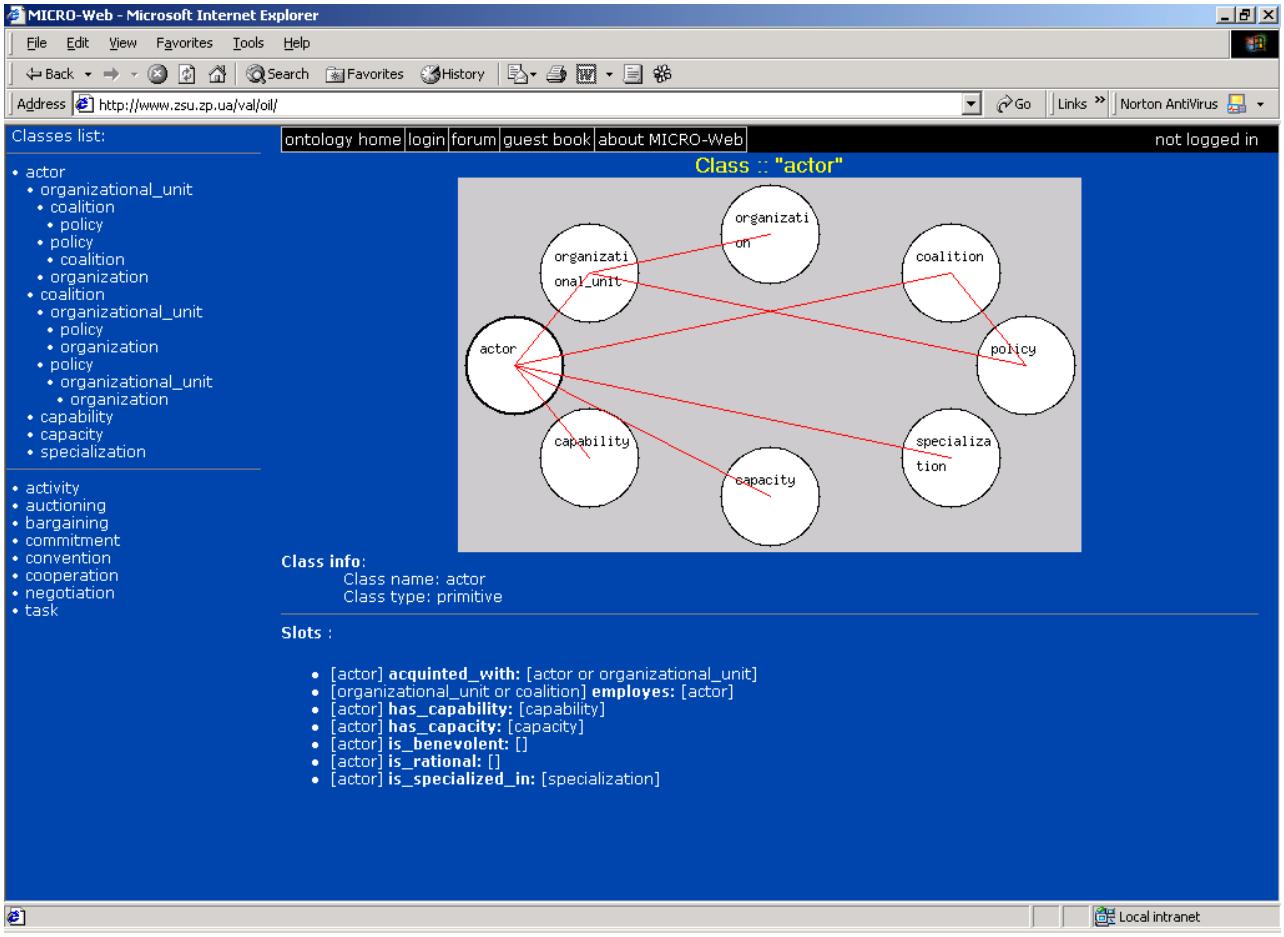

Рисунок 4.2 Визуализация части онтологии

При выборе класса онтологии пользователь переходит на страницу этого класса, на которой представлена взаимосвязь этого класса с другими классами как в графическом, так и в текстовом виде. Также на странице класса представлены его свойства, список слотов (со ссылками) в которых задействован этот класс и древовидная структура всех классов.

ВЫВОДЫ

В работе реализованы стандартные интерфейсы для обеспечения процесса коллективной разработки онтологий сообществом разработчиков.

Для обеспечения более удобной работы необходимо провести подробное тестирование системы, опросить ее пользователей на предмет удачных и неудачных решений.

Данная система будет использоваться

Пути по которым система может быть расширена:

- улучшение РМД для хранения онтологий Instance Oil
- расширение методики в направлении услуг для разработчиков
- поддержка версий онтологий
- импорт+експорт в языки

ПЕРЕЧЕНЬ ССЫЛОК

1. Neches R., Fikes R.E., FininT., Gruber T.R., Senator T., Swartout W.R. Enabling Technology for Knowledge Sharing // AI Magazine. – 1991. – No. 12(3) - pp. 36-56
2. Gruber T.R. A Translation Approach to Portable Ontologies // Knowledge Acquisition. – 1993. – No. 5(2) – pp. 199-220
3. Studer R., Benjamins V.R., Fensel D. Knowledge engineering, principles and methods // Data and Knowledge Engineering. – 1998 – No. 25 (1-2) – pp. 161-197
4. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. Киев, Наукова думка, 1989.-376 с.
5. Codd E.F. A Relational Model of Data for Large Shared Data Banks. // Comm.ACM, -Vol.13,- No.6.-1970. - pp.377-387.
6. Chen P.S. The ER-model: Towards a Unified View of Data. // ACM TODS,- Vol.1, 3 – 1976. – pp.9-36.

ПРИЛОЖЕНИЕ А.
СИНТАКСИС ЯЗЫКА OIL В ВИДЕ НОРМАЛЬНОЙ ФОРМЫ БЭКУСА

Нормальная форма Бэкуса – представление грамматики языка Standard-OIL (предоставлено Frank van Harmelen, Ian Horrocks, Michel Klein www.ontoknowledge.org/oil/downl/otk.del02.pdf).

```
<ontology> ::= "begin-ontology"
            <container>
         <definitions>
         "end-ontology"
```

/* Notes on the container syntax:
- it overspecifies the order of the elements in a signature, but never mind.
- see ISO8601 for format of dates
- MIME/types are suggested to use for format
- relations can also be qualified: relation.requires, relation.isRequiredBy, etc.
*/

```
<container> ::= "ontology-container"
               ("title" <%STRING>)+
             ("creator" <%STRING>)+
             {"subject" <%STRING>}
             "description" <%STRING>
             "description.release" <%STRING>
             {"publisher" <%STRING>}
             {"contributor" <%STRING>}
             {"date" <%STRING>}
             "type" "ontology"
             {"type" <%STRING>}
             {"format" <%STRING>}
             ("identifier" <URI>)+
             {"source" <URI>}
             ("language" <%STRING>)+
             {"relation" <URI>}
             {"rights" <%STRING>}

<definitions> ::= "ontology-definitions"
                  [<import-exp>]
           {<definition>}

<import-exp> ::= "import" <URI>+

<definition> ::= <class-definition> | <slot-definition> | <axiom>

<class-definition> ::= "class-def"  [<type>] <class-name>
                        ["documentation" <%STRING>]
                   [<subclass>]
                   {<slot-constraint>}

<type> ::= "primitive" | "defined"

<subclass> ::= "subclass-of" <class-exp> +
```

```
<class-exp> ::= <class-name> |
            "top" | "thing" | "bottom" | <set-exp> |
            "(" <slot-constraint> ")" |
          "(" <class-exp> ("and" <class-exp>)+ ")" |
          "(" <class-exp> ("or"  <class-exp>)+ ")" |
          "(" "not" <class-exp> ")"

<set-exp> ::= "(" "one-of" <ind-name> + ")"

<slot-constraint> ::= "slot-constraint" <slot-name> <constraint-
exp> +

<constraint-exp> ::= <has-value-exp> |
                     <value-type-exp> |
                     <max-cardinality-exp> |
              <min-cardinality-exp> |
              <cardinality-exp> |
                     <filler-exp>
```

/* filler-exp is syntactic sugar for "has-value (one-of ind-name) +" or
"has-value concrete-type +" */

```
<filler-exp> ::= "has-filler" <filler> +
<filler> ::= <ind-name> | <%STRING> | <%INTEGER>
<has-value-exp>  ::= "has-value"  (<class-exp>+ | <concrete-type-
exp>+)
<value-type-exp> ::= "value-type" (<class-exp>+ | <concrete-type-
exp>+)
<max-cardinality-exp> ::= "max-cardinality" <%INTEGER>
[<expression>]
<min-cardinality-exp> ::= "min-cardinality" <%INTEGER>
[<expression>]
<cardinality-exp>     ::= "cardinality"     <%INTEGER>
[<expression>]

<expression> ::= (<class-exp> | <concrete-type-exp>)

<concrete-type-exp> ::= <int-exp> | <string-exp>
<int-exp> ::= "integer" |
          "(" <int-pred> <%INTEGER> ")" |
          "(" "range" <%INTEGER> <%INTEGER> ")" |
          "(" <int-exp> ("and" <int-exp>)+ ")" |
          "(" <int-exp> ("or"  <int-exp>)+ ")" |
          "(" "not" <int-exp> ")"
<int-pred> ::= "min" | "max" | "greater-than" | "less-than" |
"equal"
<string-exp> ::= "string" |
          "(" <string-pred> <%STRING> ")" |
          "(" "range" <%STRING> <%STRING> ")" |
          "(" <string-exp> ("and" <string-exp>)+ ")" |
          "(" <string-exp> ("or"  <string-exp>)+ ")" |
          "(" "not" <string-exp> ")"
<string-pred> ::= "min" | "max" | "greater-than" | "less-than" |
"equal"

<slot-definition> ::= "slot-def" <slot-name>
                      ["documentation" <%STRING>]
               [<subslot-exp>]
                      {<slot-def-exp>}
```

```
<subslot-exp>   ::= "subslot-of" <slot-name> +

<slot-def-exp> ::= <domain-exp> |
            <range-exp> |
            <inverse-exp> |
            <property-list>

<domain-exp>    ::= "domain" <class-exp> +
<range-exp>     ::= "range" (<class-exp>+ | <concrete-type-exp>+)
/* NOTE - it is good practice to restrict the range to a concrete type when a slot is to be used in that
context, e.g.,
  "slot-def age range integer" */
<inverse-exp>   ::= "inverse" <slot-name> +
<property-list> ::= "properties" <property> +
<property>      ::= "transitive" | "symmetric" | "functional"
/* NOTE - transitive and functional are disjoint */

<axiom> ::= <disjoint-exp> |
        <cover-exp> |
        <disjoint-cover-exp> |
        <equiv-exp>
<disjoint-exp>       ::= "disjoint" <class-exp> <class-exp> +
<cover-exp>          ::= "covered" <class-exp> "by" <class-exp> +
<disjoint-cover-exp> ::= "disjoint-covered" <class-exp> "by"
<class-exp> +
<equiv-exp>          ::= "equivalent" <class-exp> <class-exp>+

/* symbols are a sequence of letters, digits and hyphens, starting with a letter
*/
<class-name> ::= <symbol>
<slot-name>  ::= <symbol>
<ind-name>   ::= <symbol>

/* <symbol> ::= [A-Za-z][-A-Za-z0-9]* */

/* raw approximations of types */

<URI>        ::= <%STRING>
<symbol>     ::= <%STRING>
```

ПРИЛОЖЕНИЕ Б.
СТРУКТУРА ДАННЫХ ДЛЯ ХРАНЕНИЯ ОНТОЛОГИИ В СРЕДЕ MYSQL

```
CREATE TABLE classes (
  idclass int(11) NOT NULL auto_increment,
  classname varchar(255) NOT NULL default '',
  classtype enum('defined','primitive') NOT NULL default
'primitive',
  PRIMARY KEY (idclass)
) TYPE=MyISAM COMMENT='все явно определенные классы';
# --------------------------------------------------------

#
# Структура таблицы для таблицы `inverses`
#

CREATE TABLE inverses (
  slot1 int(11) NOT NULL default '0',
  slot2 int(11) NOT NULL default '0',
  PRIMARY KEY (slot1,slot2)
) TYPE=MyISAM COMMENT='';
# --------------------------------------------------------

#
# Структура таблицы для таблицы `slotconstraints`
#

CREATE TABLE slotconstraints (
  idclass int(11) NOT NULL default '0',
  slotname varchar(255) NOT NULL default '',
  type enum('max-card','min-card','card','has-value','value-
type','has-filler','one-of') default NULL,
  exp varchar(255) NOT NULL default '',
  PRIMARY KEY (idclass,slotname)
) TYPE=MyISAM COMMENT='';
# --------------------------------------------------------

#
# Структура таблицы для таблицы `slots`
#

CREATE TABLE slots (
  idslot int(11) NOT NULL auto_increment,
  slotname varchar(255) NOT NULL default '',
  domain varchar(255) default NULL,
  range varchar(255) default NULL,
  proptransitive tinyint(1) NOT NULL default '0',
  propsymmetric tinyint(1) NOT NULL default '0',
  propfunctional tinyint(1) NOT NULL default '0',
  PRIMARY KEY (idslot)
) TYPE=MyISAM COMMENT='';
```

```
# --------------------------------------------------------

#
# Структура таблицы для таблицы `subclasses`
#

CREATE TABLE subclasses (
  idclass int(11) NOT NULL default '0',
  subclassof varchar(255) NOT NULL default ''
) TYPE=MyISAM COMMENT='';
# --------------------------------------------------------

#
# Структура таблицы для таблицы `subslots`
#

CREATE TABLE subslots (
  idslot int(11) NOT NULL default '0',
  subslotof int(11) NOT NULL default '0',
  PRIMARY KEY (idslot,subslotof)
) TYPE=MyISAM COMMENT='';
```

ПРИЛОЖЕНИЕ В. ПРОГРАММА ВИЗУАЛИЗАЦИИ ОНТОЛОГИИ, ХРАНЯЩЕЙСЯ В БАЗЕ ДАННЫХ

```
<?php
$w = 500; $h = 300; $r = 80;


function selectclasses($i)
     {
     global $selected, $slots, $classes, $selectedslots;

     $selected[] = $i;
     foreach ($slots as $sid=>$slot)
          {
          if ((in_array($i,$slot[parsed]-
>domain)||in_array($i,$slot[parsed]-
>range))&&(!in_array($sid,$selectedslots)))
               {
               $selectedslots[] = $sid;
               foreach ($slot[parsed]->domain as $cur)
                    if ($classes[$cur]) selectclasses($cur);
               foreach ($slot[parsed]->range as $cur)
                    if ($classes[$cur]) selectclasses($cur);
               }
          }
     }

$c = $classes[$class];

if (!$c)
     mydie ("Invalid class specified");

$selected = array();
$selectedslots = array();
selectclasses ($class);

$old = $classes;
$classes = array();

foreach ($selected as $cl)
     $classes[$cl] = $old[$cl];

?>
//-------------------------------------------------------------
<?php
include ('mysql.class.php');
$sql = new MySQL;
include ('function.inc.php');

header ("Content-type: image/png");
$im = @ImageCreate ($w, $h)
    or die ("Cannot Initialize new GD image stream");
```

```
//color defing group
$background_color = ImageColorAllocate ($im, 200, 200, 200);
$white = ImageColorAllocate ($im, 0xff, 0xff, 0xff);
$black = ImageColorAllocate ($im, 0x00, 0x00, 0x00);
$red = ImageColorAllocate ($im, 0xff, 0x00, 0x00);


$i = 0; $Rx = $w/2 - $r/2 - 5; $Ry = $h/2 - $r/2 - 5; $phi =
2*M_PI/count($classes);
foreach ($classes as $id=>$class)
     {
     $x = round ($w/2-$Rx*cos($phi*$i));
     $y = round ($h/2-$Ry*sin($phi*$i));
     $classes[$id][coords]->x = $x;
     $classes[$id][coords]->y = $y;
     imagefilledellipse ($im,$x,$y,$r,$r,$white);
     imagearc ($im,$x,$y,$r,$r,0,360,$black);
     imagestring ($im,2,$x-$r/3,$y-
$r/4,substr($class['classname'],0,10),$black);
     imagestring ($im,2,$x-
$r/3,$y,substr($class['classname'],10),$black);
     $i++;
     }

foreach ($slots as $id=>$slot)
     {
     $domain = $slot['domain'];
     $range = $slot['range'];
     if ($domain && $range)
          {
          $_d = explode (' ',$domain);

          $ds = array();
          foreach ($_d as $d)
               {
               $d = check ($d,0,0);
               if ($d) $ds[] = $d;
               }
          $_r = explode (' ',$range);
          $rs = array();
          foreach ($_r as $r)
               {
               $r = check ($r,0,0);
               if ($r) $rs[] = $r;
               }
          slotconnect ($ds, $rs);
          }
     }

ImagePNG ($im);
$sql->close();

?>
```

```
//-------------------------------------------------------
<?php include ('top.inc.php'); ?>
<?php
     include ('mysql.class.php');
     $sql = new MySQL;
     include ('function.inc.php');

     function cmp ($a,$b)
     {
     global $classes;
     return
strcmp($classes[$a][classname],$classes[$b][classname]);
     }

     $c = $QUERY_STRING;

if (!$classes[$c])
     {
     uksort ($classes,"cmp");
     echo "<br> <a href=all.php target=info>All
classes</a>\n:<ul>\n";
     foreach ($classes as $id=>$class)
          {
          echo "<li><a href=class.php?class=$id
target=info>{$class[classname]}</a></li>\n";
          }
     echo "</ul>\n</body>\n</html>";
     }
else{
     $usedinrec[$c] = true;
     $usedinbranch[$c] = true;

     function connected ($a)
          {
          global $slots, $classes;
          $result = array ();

          foreach ($slots as $slot)
               {
               foreach ($slot[parsed]->domain as $d)
                    if ($classes[$d] &&
(in_array($a,$slot[parsed]->domain)||in_array($a,$slot[parsed]-
>range)) )
                         {
                         if (!in_array($d,$result))$result[] = $d;
                         }
               foreach ($slot[parsed]->range as $d)
                    if ($classes[$d] &&
(in_array($a,$slot[parsed]->domain)||in_array($a,$slot[parsed]-
>range)))
                         {
                         if (!in_array($d,$result))$result[] = $d;
                         }
```

```
                }
           return ($result);
           }

     function showbranch ($a)
           {
           global $level, $classes, $usedinrec, $usedinbranch;
           $brothers = connected ($a);
           $level++;
                {
                foreach ($brothers as $b)
                     {
                     if (!$usedinbranch[$b])
                          {
                          echo
str_repeat(' ',$level*2)."&#149; <a href=class.php?class=$b
target=info>{$classes[$b][classname]}</a><br>\n";
                          $usedinbranch[$b] = true;
                          showbranch ($b);
                          $usedinrec[$b] = true;
                          $usedinbranch[$b] = false;
                          }
                     }
                }
           $level--;
           }

     echo " Classes list:<hr noshade> &#149; <a
href=class.php?class=$c
target=info>{$classes[$c][classname]}</a><br>\n";
     $level = 1;
     showbranch ($c);
     echo "<hr noshade>\n";
     foreach ($classes as $i=>$class)
           {
           if (!$usedinrec[$i])
              echo " &#149; <a href=class.php?class=$i
target=info>{$classes[$i][classname]}</a><br>\n";
           }
     }
?>
</body></html>
```