Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ

_____К.А. Домбровский
«___»_____2017 г.

Представление предметной области в виде онтологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ ЮУрГУ-09.03.01.2017.223. ПЗ ВКР

Руководитель работы
Доцент, к.т.н.
_____ И.Л. Кафтанников
«___»_____2017 г.

Автор работы
студент группы КЭ – 445
_____М.В. Петропавловский
«___»_____2017 г.

Нормоконтролер
Старший преподаватель, доцент
_____В.В. Лурье
«___»_____2017 г.

Челябинск 2017

АННОТАЦИЯ

Петропавловский М.В. Представление предметной области в виде онтологий. – Челябинск: ЮУрГУ, ВШЭиКН; 2017, 63 с., 10 ил., библиогр. список – 10 наим.

Ключевые слова: предметная область, онтология, объектная модель, программные системы, объектно-ориентированный подход.

Цель работы – Создание представления структуры онтологии семантической сети на основе объектно-ориентированного подхода.

В связи с быстрым развитием IT-технологий перед программистами стоит задача написания качественного программного обеспечения. Для этого необходимо четко понимать предметную область, на которую будет опираться тот или иной программный продукт. Для написания такого продукта необходимо создание объектной модели. Также очень важен перевод этой модели в исходный код программы. В работе описывается процесс создания объектной модели и ее представление в рамках объектно-ориентированного языка программирования на примере онтологий семантической сети.

Результаты исследования – составлена и описана на объектно-ориентированном языке программирования объектная модель онтологии, которая затем может применяться в программных системах для хранения описания предметной области.

Работа может представлять интерес для программистов и специалистов, работающих с онтологиями.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Развитие наукоемких областей человеческой деятельности в современном обществе сопровождается возрастанием роли компьютерных технологий. Сейчас значительно увеличивается поток информации, появилась необходимость поиска новых способов ее хранения, представления, формализации и систематизации, а также автоматической обработки. Таким образом, растет интерес к всеобъемлющим базам знаний, которые возможно использовать для различных практических целей. Огромный интерес вызывают системы, способные без участия человека извлечь какие-либо сведения из текста. Как результат, на фоне вновь возникающих потребностей развиваются новые технологии, призванные решить заявленные проблемы. В развитии World Wide Web появляется его расширение, Semantic Web, в котором гипертекстовые страницы снабжаются дополнительной разметкой, несущей сведения о семантике включаемых в страницы элементов. Неотъемлемым компонентом Semantic Web является понятие онтологии, описывающее смысл семантической разметки.

В общих чертах под онтологией понимается система понятий некоторой предметной области, которая представляется как набор сущностей, соединенных различными отношениями. Онтологии используются для формальной спецификации понятий и отношений, которые характеризуют определенную область знаний. Преимуществом онтологий в качестве способа представления знаний является их формальная структура, которая упрощает их компьютерную обработку.

Неявное применение онтологий состоит в использовании их в таких науках, как биология, медицина, геология и других естественных науках, где они являются фундаментом для построения теорий.

Явное применение онтологий состоит в использовании онтологий, как источника данных для многих компьютерных приложений. То есть они позволяют осуществлять информационный поиск, анализ текстов, извлекать знания, а также используются в других информационных технологиях, что

представляет возможность эффективно обрабатывать сложную и разнообразную информацию. Такой способ представления знаний позволяет приложениям распознавать те семантические отличия, которые являются само собой разумеющимися для людей, но не известны компьютеру. В связи с этим возникла необходимость создания стандартизованных способов представления онтологий. Началось развитие языков, которые могли бы применяться во всех системах. Самыми известными языками являются RDF и OWL. Также появились редакторы для создания, пополнения и изменения онтологий. Каждое из них в основном направлено на работу с определенным форматом данных и обладает своими особенностями.

Любое проектирование программной системы начинается с анализа предметной области заказчика [2, с.68-70]. Дело в том, что предметная область сильно влияет на все аспекты проекта: требования к системе, взаимодействие с пользователем, модель хранения данных, реализацию и т.д. Под анализом понимается деятельность, направленная на выявление реальных потребностей заказчика, а также на выяснения смысла высказанных требований. Это первый шаг этапа системного анализа, с которого начинается разработка программной системы. На данном этапе разработчики должны научиться: понимать язык, на котором говорят заказчики; выявить цели их деятельности; определить набор решаемых ими задач; определить набор сущностей, с которыми приходится иметь дело при решении этих задач.

Зачастую рассматриваемая предметная область бывает нетривиальна, содержит множество различных классов объектов, с широким множеством связей между ними. Количество объектов и связей может достигать нескольких сотен даже для небольшой рассматриваемой предметной области [4].

Для упорядоченного описания предметной области в том числе используются онтологии, составляемые аналитиками предметных областей. Но затем встает вопрос, как же перевести онтологию предметной области в объектную модель

предметной области, которая уже может быть использована в любой объектно-ориентированной системе.

Цель работы – создание представления структуры онтологии семантической сети на основе объектно-ориентированного подхода.

Работа содержит три главы. Объем работы 2 листа, библиографический список содержит источников.

В первой главе описываются значения языков предметных областей, методы формирования языков, общие сведения и составные части онтологий.

Во второй главе описывается организация процедуры составления объектной модели, позволяющей представить онтологии в объектно-ориентированном виде.

Третья глава включает описание применения данной модели.

1 ПОНЯТИЕ ОНТОЛОГИИ. ПРИМЕНЕНИЕ ОНТОЛОГИИ

1.1 Языки предметных областей

Вслушиваясь в разговор экспертов в какой-либо предметной области, будь то шахматисты, работники детского сада или страховые агенты, можно заметить, что они используют слова, отличающиеся от обычных, используемых повседневно. Это то, что называется «язык предметной области» – набор слов и выражений, описывающий вещи, характерные для данной предметной области.

В мире программного обеспечения под ЯПО подразумевается набор исполняемых выражений на языке, специфичном для данной области, с ограниченным словарем и грамматикой, которые при этом читабельные, понятные и используемые экспертами предметной области. ЯПО предназначен для разработчиков или научных работников, находящихся в этой предметной области долгое время. Например, язык конфигурационных файлов Unix или языки, созданные при помощи LISP-макросов.

Таким образом, ЯПО предназначены для понимания экспертами той предметной области, в которой они работают.

ЯПО принято разделять на внутренние и внешние.

Внутренние ЯПО создаются при помощи языков программирования, чей синтаксис близок к обычному языку людей. Сделать такое проще на языках, предлагающих больше «синтаксического сахара», как например Ruby или Scala, и сложнее на тех, которые такого не предоставляют (например, Java). Большинство внутренних ЯПО представляют собой обертки для существующих API, библиотек, предоставляя менее головоломный вариант использования существующей функциональности. Их можно исполнять напрямую. В зависимости от реализации и домена, ЯПО могут создавать структуры данных, определять зависимости, запускать процессы и задачи, связываться с другими системами или проверять пользовательский ввод. Синтаксис внутренних ЯПО ограничивается языком программирования, на котором они реализованы.

Существует много шаблонов, например, построители выражений, цепочечные методы, аннотации, которые могут помочь вам «привести» ваш язык программирования к вашему ЯПО. Если используемый язык программирования не требует перекомпиляции, то внутренний ЯПО может быть весьма быстро разработан при непосредственном участии эксперта предметной области.

Внешние ЯПО – это текстовые или графические выражения языка, причем текстовые, как правило, используются чаще. Текстовые выражения могут обрабатываться инструментарием, включающим лексический анализатор, грамматический разбор, модификатор модели, генераторы и другие виды постобработки. Внешние ЯПО часто преобразуются во внутреннюю модель, представляющую базис для дальнейшей обработки. Хорошо помогает задать грамматику (например, расширенную форму Бэкуса-Наура). Грамматика предоставляет начальную точку для всего остального (редактор, визуализатор, грамматический разборщик). Для простых ЯПО ручной разборщик грамматики может быть достаточным (например, можно использовать регулярные выражения). В более сложных случаях лучше посмотреть в сторону специально разработанных для работы с грамматиками инструментов, таких как openArchitectureWare, ANTlr, SableCC, AndroMDA. Определение внешнего ЯПО как диалекта XML тоже часто встречается, хотя читабельность такого языка часто является проблемой, особенно для тех, кто не знаком с XML.

1.2 Понятие онтологии

Понятие онтологии имеет 2 значения:

1. философская дисциплина изучает наиболее общие характеристики бытия и сущностей;

2. онтология – артефакт, структура, описывающая значения элементов некоторой системы.

Неформально, онтология представляет собой некоторое описание взгляда на мир применительно к конкретной области интересов [1]. Это описание состоит из терминов и правил использования этих терминов, ограничивающих их значения в рамках конкретной области. На формальном уровне, онтология – это система, состоящая из набора понятий и набора утверждений об этих понятиях, на основе которых можно строить классы, объекты, отношения, функции и теории.

Основные компоненты [4]:

* классы или понятия;

* отношения;

* функции;

* аксиомы;

* примеры.

Классы являются общими категориями, которые могут быть упорядочены иерархически. Каждый класс в свою очередь описывает группу индивидуальных сущностей, которые объединены на основании наличия общих свойств. Классы могут связываются друг с другом с помощью отношений:

1) таксономическое отношение;

2) отношение IS-A;

3) класс-подкласс;

4) гипоним-гиперним;

5) родовидовое отношение;

6) отношение a-kind-of.

Аксиомы задают условия соотнесения категорий и отношений, они выражают очевидные утверждения, связывающие классы и отношения.

Экземпляры – отдельные представители класса сущностей или явлений, то есть конкретные элементы какой-либо категории.

Онтологией могут быть:

* глоссарий

* простая таксономия

* тезаурус

* понятийная структура с произвольным набором отношений

* структура с аксиоматикой

Все элементы структуры онтологии подчиняются иерархии. На нижнем уровне экземпляры, выше классы, затем отношения между классами, а самой связующей является ступень аксиом (Рисунок 1.2.1).

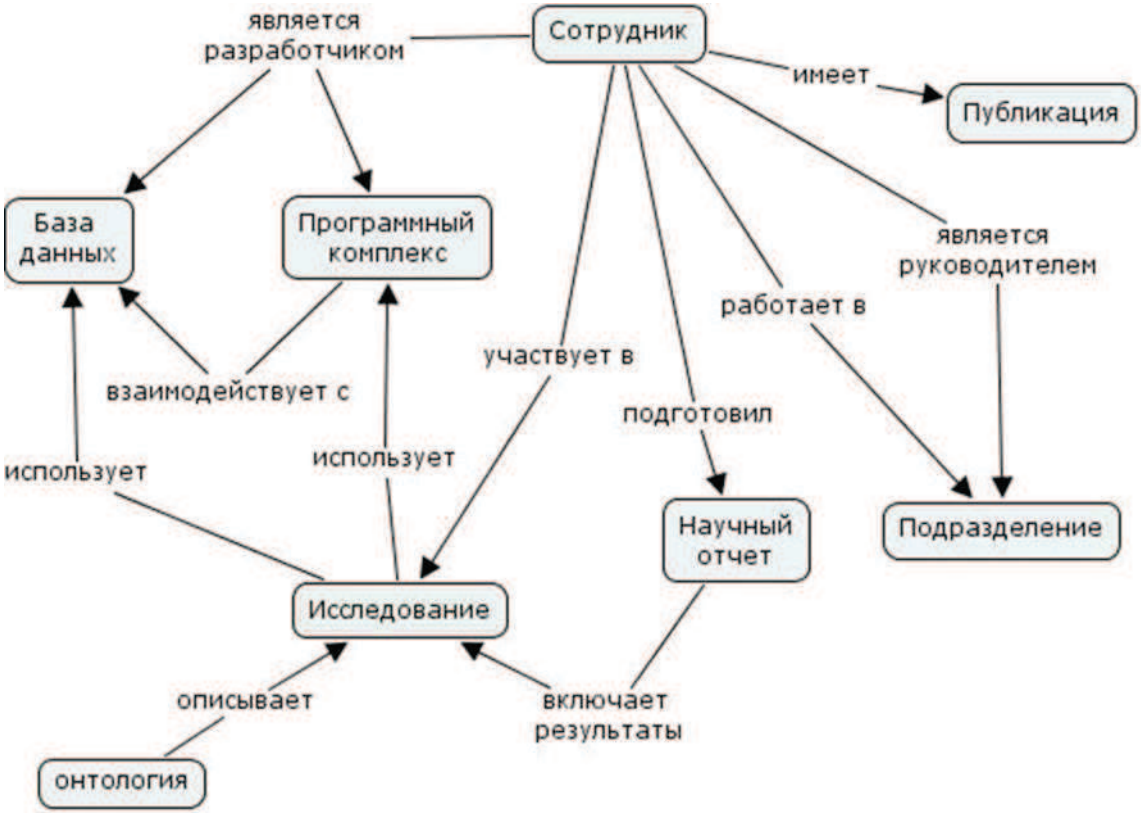

Рисунок 1.2.1 - Пример онтологии

Таким образом, онтологии применяются для общего понимания структуры информации при использовании людьми и программными агентами, для возможности повторного использования знаний в предметной области, чтобы сделать допущения в предметной области явными, для отделения знаний в предметной области от оперативных знаний, для анализа знаний в предметной области. Соответственно, онтология является базовым ресурсом для построения предметных областей.

1.3 Принципы онтологий

Онтологические системы строятся на основе следующих принципов:

• формализации, то есть описания объективных элементов действительности в единых, строго определённых образцах (терминах, моделях);

• использования ограниченного количества базовых терминов (сущностей), на основе которых конструируются все остальные понятия;

• внутренней полноты и логической непротиворечивости.

В отличие от обычного словаря для онтологической системы характерно внутреннее единство, логическая взаимосвязь и непротиворечивость используемых понятий.

1.4 Классификация онтологических систем

Модели онтологии классифицируются следующим образом:

- простые (имеют лишь концепты);
- на основе фреймов (имеют лишь концепты и свойства);
- на основе логик.

По языку представления онтологических знаний онтологии классифицируются следующим образом:

- RDF. Язык разработан в рамках проекта семантик-веб (Semantic Web). Основное предназначение языка - описание метаданных документов, размещаемых в Интернет. RDF использует базовую модель представления данных "объект - атрибут - значение", и способен сыграть роль универсального языка описания семантики ресурсов и связей между ними.
- DAML+OIL – семантический язык разметки Web-ресурсов, который расширяет стандарты RDF i RDF Schema за счет более полных примитивов моделирования. В последнюю версию DAML+OIL включен набор

дополнительных конструкций для создания онтологий и разметки информации в легко интерпретируемом машиной виде.

- OWL (Web Ontology Language) – язык представления онтологий следующего поколения после

DAML+OIL. Обладает более богатым набором возможностей чем XML, RDF, RDF Schema i DAML+OIL. Проект предполагает создание мощного механизма семантического анализа. Планируется, что в нем будут устранены ограничения конструкций DAML+OIL. Онтология OWL – это последовательность аксиом, фактов и ссылок на другие онтологии.

- KIF (Knowledge Interchange Format или формат обмена знаниями) - основан на S-выражениях синтаксис для логики. KIF - специальный язык, предназначенный для обмена знаниями между разными компьютерными системами. Разрабатывался для описания общего формата представления знаний независимого от конкретных систем.
- CycL (язык описания онтологии Cyc) – это гибридный язык, в котором объединены свойства фреймов и логики предикатов. Синтаксис языка CycL схожий с синтаксисом языка Lisp. CycL различает такие сущности, как экземпляры, классы, предикаты и функции. Словарь CycL состоит из термов. Множество термов можно разделить на константы, неатомарные термы и переменные. Термы используются при составлении значащих выражений CycL, из которых формируются суждения. Из суждений состоит база знаний.
- OCML (Operational Conceptual Modeling Language) язык поддерживает построение нескольких типов конструкций представления знаний. Он позволяет задавать спецификацию и операционализацию функций, связей, классов, экземпляров и правил. Он также включает механизмы для описания онтологий и методов решения задач - основные технологии, разработанные в области представления знаний. Около десятка проектов в

KMi (Knowledge Media Institute) в настоящее время используют OCML для разработки моделей в таких областях как управление знаниями, разработка онтологии, электронная торговля и системы обработки знаний.

- LOOM и PowerLoom® - языки представления знаний, разработанные исследователями из группы Artificial Intelligence Research Group Университета Южной Калифорнии (University of Southern California's Information Sciences Institute). Цель проекта Loom – разработка и внедрение продвинутых средств для представления знаний и рассуждений в области искусственного интеллекта. Loom и PowerLoom распространяются по открытой лицензии (open source licenses), но являются интеллектуальной собственностью Университета Южной Калифорнии и не являются общедоступными.
- Loom это и язык и среда для построения интеллектуальных приложений. Центром языка является система представления знаний, которая используется для построения дедуктивных выводов на основе декларативных знаний. Декларативные знания состоят из определений, правил, фактов и правил по умолчанию. Дедуктивный движок использует прямые цепочки логического вывода, семантическую унификацию и объектно-ориентированные технологии поддержания достоверности.
- Ontolingua предоставляет распределенную среду для совместного просмотра, создания, редактирования, изменения и использования онтологий. Сервер поддерживает более 150 активных пользователей, с некоторыми проектами которых можно ознакомиться по адресу http://ontolingua.stanford.edu/doc/ontology-server-projects.html. Ontolingua состоит из KIF парсера, инструментов для анализа онтологии, и набора трансляторов для преобразования исходных данных Ontolingua в форму, приемлемую для внедрения в системы представления знаний.
- F-Logic – онтологический язык, который базируется на логиках первого порядка, однако классы и свойства в нем представлены как термины, а не

как предикаты. Язык создавался для осуществления взаимодействия между онтологиями, построенными на основе предикатов, и онтологиями, построенными на основе F-Logic. Создатели определили интуитивные трансляторы для преобразования знаний из предикатных онтологий в F-Logic онтологии и показали, что такой перевод сохраняет логические связи (preserves entailment) для большого количества онтологических языков, в том числе и для многих OWL DL. Также, язык может применяться для мета-моделирования расширений Description Logics (v-semantics).

1.5 Разработка онтологии

Разработка онтологии состоит из:

- определения классов;
- расположения классов в таксономическую иерархию (подкласс – надкласс);
- определения слотов и описания допускаемых значений этих слотов;
- заполнения значений слотов-экземпляров.

Для описания онтологий и работы с ними применяется визуальный подход, позволяющий представить онтологии в виде графа (тип графа – в виде дерева), что помогает наглядно сформулировать и объяснить природу и структуру явлений. Любой программный графический пакет можно использовать как первичный инструмент описания онтологий.

1.6 Понятие и принципы семантической сети

Семантическая сеть — информационная модель предметной области, имеющая вид ориентированного графа, вершины которого соответствуют объектам предметной области, а дуги (рёбра) задают отношения между ними.

Объектами могут быть понятия, события, свойства, процессы. Таким образом, семантическая сеть является одним из способов представления знаний. В названии соединены термины из двух наук: семантика в языкознании изучает смысл единиц языка, а сеть в математике представляет собой разновидность графа — набора вершин, соединённых дугами (рёбрами). В семантической сети роль вершин выполняют понятия базы знаний, а дуги (причем направленные) задают отношения между ними. Таким образом, семантическая сеть отражает семантику предметной области в виде понятий и отношений.

Семантическая сеть - структура для представления знаний в виде узлов, соединенных дугами. Самые первые семантические сети были разработаны в качестве языка-посредника для систем машинного перевода, а многие современные версии до сих пор сходны по своим характеристикам с естественным языком. Однако последние версии семантических сетей стали более мощными и гибкими и составляют конкуренцию фреймовым системам, логическому программированию и другим языкам представления знаний.

Начиная с конца 50-ых годов прошлого века были созданы и применены на практике десятки вариантов семантических сетей. Несмотря на то, что терминология и их структура различаются, существуют сходства, присущие практически всем семантическим сетям:

1. Узлы семантических сетей представляют собой концепты предметов, событий, состояний.

2. Различные узлы одного концепта относятся к различным значениям, если для них не помечено, что они относятся к одному концепту.

3. Дуги семантических сетей создают отношения между узлами-концептами (пометки над дугами указывают на тип отношения).

4. Некоторые отношения между концептами представляют собой лингвистические падежи, такие как агент, объект, реципиент и инструмент (другие означают временные, пространственные, логические отношения и отношения между отдельными предложениями).

5. Концепты организованы по уровням в соответствии со степенью обобщенности. Как пример, сущность, живое существо, животное, плотоядное.

Однако существуют и различия: понятие значения с точки зрения философии; методы представления кванторов общности и существования и логических операторов; способы манипулирования сетями и правила вывода, терминология. Все это варьируется от автора к автору. Несмотря не некоторые различия, сети удобны для чтения и обработки компьютером, а также достаточно мощны, чтобы представить семантику естественного языка. Самые простые сети, которые используются в системах искусственного интеллекта, - это реляционные графы. Они состоят из узлов, соединенных дугами. Каждый узел представляет собой понятие, а каждая дуга - отношения между различными понятиями. Ниже представлено предложение "Собака жадно гложет кость". Четыре прямоугольника представляют понятия самой собаки, процесса глодания кости, самой кости и такой характеристики как жадность. Надписи рядом с дугами означают, что собака является агентом процесса, кость является объектом этого процесса, а жадность - это характеристика выполнения процесса.

Терминология, использующаяся в этой области, различна. Чтобы добиться некоторой однородности, узлы, соединенные дугами, принято называть графами, а структуру, где имеется целое гнездо из узлов или где существуют отношения различного порядка между графами, называть сетью. Помимо терминологии, использующейся для пояснения, также различаются способы изображения. Некоторые используют кружки вместо прямоугольников; некоторые пишут типы отношений над или под дугами, заключая или же не заключая их в овалы; некоторые используют аббревиатуры вида О или А для обозначения агента или объекта; некоторые используют различные типы стрелок.

1.7 Представление семантических сетей

Математика позволяет описать большинство явлений в окружающем мире в виде логических высказываний. Семантические сети возникли как попытка

визуализации математических формул. Основным представлением для семантической сети является граф. Однако не стоит забывать, что за графическим изображением непременно стоит строгая математическая запись, и что обе эти формы являются не конкурирующими, а взаимодополняющими.

Существуют следующие виды представлений семантических сетей:

1.7.1 Графическое представление

Основной формой представления семантической сети является граф. Понятия семантической сети записываются в овалах или прямоугольниках и соединяются стрелками с подписями — дугами. Это наиболее удобно воспринимаемая человеком форма. Её недостатки проявляются, когда мы начинаем строить более сложные сети или пытаемся учесть особенности естественного языка.

Пример графического представления семантической сети представлен на рисунке 1.7.1.

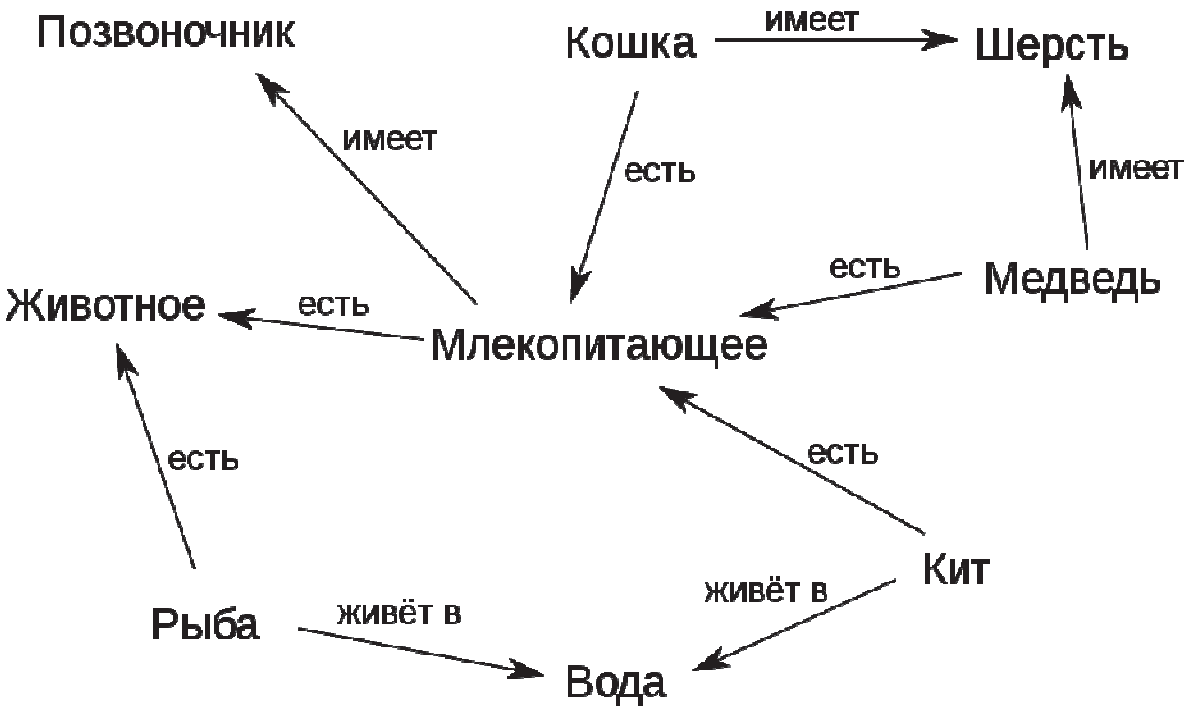

Рисунок 1.7.1 – Графическое представление семантической сети

1.7.2 Математическая запись

В математике граф представляется множеством вершин V и множеством отношений между ними E. Используя аппарат математической логики, приходим к выводу, что каждая вершина соответствует элементу предметного множества, а дуга — предикату.

Пример математической записи семантической сети представлен на рисунке 1.7.2.

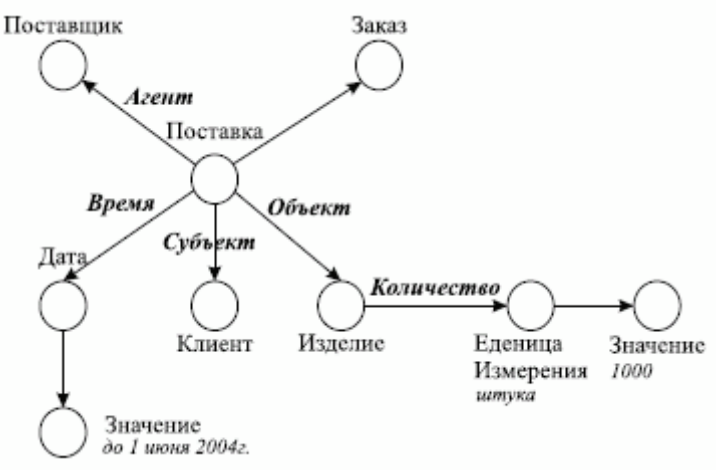

Рисунок 1.7.2 – Математическая запись семантической сети

1.7.3 Лингвистическая запись

В лингвистике отношения фиксируются в словарях и в тезаурусах. В словарях в определениях через род и видовое отличие родовое понятие занимает определённое место. В тезаурусах в статье каждого термина могут быть указаны все возможные его связи с другими родственными по теме терминами. От таких тезаурусов необходимо отличать тезаурусы информационно-поисковые с перечнями ключевых слов в статьях, которые предназначены для работы дескрипторных поисковых систем.

1.8 Классификация семантических сетей

Для всех семантических сетей справедливо разделение по арности и количеству типов отношений.

- **По количеству типов отношений, сети могут быть однородными и неоднородными.**
  - Однородные сети обладают только одним типом отношений (стрелок), например, таковой является классификация биологических видов (с единственным отношением AKO (A Kind Of – отношение между надмножеством и подмножеством)).
  - В неоднородных сетях количество типов отношений два и более двух. Классические иллюстрации данной модели представления знаний представляют именно такие сети. Неоднородные сети представляют больший интерес для практических целей, но и большую сложность для исследования. Неоднородные сети можно представлять, как переплетение древовидных многослойных структур. Примером такой сети может быть Семантическая сеть Википедии.
- **По арности:**
  - типичными являются сети с **бинарными** отношениями (связывающими ровно два понятия). Бинарные отношения очень просты и удобно изображаются на графе в виде стрелки между двух концептов. Кроме того, они играют исключительную роль в математике.
  - На практике, однако, могут понадобиться отношения, связывающие более двух объектов — **N-арные**. При этом возникает сложность — как изобразить подобную связь на графе, чтобы не запутаться. Концептуальные графы снимают это затруднение, представляя каждое отношение в виде отдельного узла.

Пример концептуального графа приведен на рисунке 1.8.1

Рисунок 1.8.1 – Концептуальный граф для высказывания «Сети представляются в виде графа

- **По размеру:**
  - Для решения конкретных задач, например, тех, которые решают системы искусственного интеллекта.
  - Семантическая сеть отраслевого масштаба должна служить базой для создания конкретных систем, не претендуя на всеобщее значение.
  - Глобальная семантическая сеть. Теоретически такая сеть должна существовать, поскольку всё в мире взаимосвязано. Возможно когда-нибудь такой сетью станет Всемирная паутина.

Помимо концептуальных графов существуют и другие модификации семантических сетей, это является ещё одной основой для классификации (по реализации).

1.9 Семантические отношения

Количество типов отношений в семантической сети определяется её создателем, исходя из конкретных целей. В реальном мире их число стремится к бесконечности. Каждое отношение является, по сути, предикатом, простым или составным. Скорость работы с базой знаний зависит от того, насколько эффективно реализованы программы обработки нужных отношений.

1.9.1 Иерархические

Наиболее часто возникает потребность в описании отношений между элементами, множествами и частями объектов. Отношение между объектом и множеством, обозначающим, что объект принадлежит этому множеству,

называется отношением классификации (ISA). Говорят, что множество (класс) классифицирует свои экземпляры. (пример: «Шарик является собакой» ="Шарик является объектом типа собака"). Иногда это отношение именуют также MemberOf, InstanceOf или подобным образом. Связь ISA предполагает, что свойства объекта наследуются от множества. Обратное к ISA отношение используется для обозначения примеров, поэтому так и называется — «Example», или по-русски Пример. Иерархические отношения образуют Древовидную структуру.

· Отношение между надмножеством и подмножеством называется **AKO*** — «A Kind Of» («разновидность») (пример: "собака является животным"=" тип с именем «собака» является подтипом типа «животные»"). Элемент подмножества называется гипонимом (собака), а надмножества — гиперонимом (животное), а само отношение называется **отношением гипонимии**. Альтернативные названия — «SubsetOf» и «Подмножество». Это отношение определяет, что каждый элемент первого множества входит и во второе (выполняется ISA для каждого элемента), а также логическую связь между самими подмножествами: что первое не больше второго и свойства первого множества наследуются вторым.

· Объект, как правило, состоит из нескольких частей, или элементов. Например, компьютер состоит из системного блока, монитора, клавиатуры, мыши и т. д. Важным отношением является **HasPart**, описывающее части/целые объекты (**отношение меронимии**). Мероним — это объект, являющийся частью для другого. Двигатель — это мероним для автомобиля. Холоним — это объект, который включает в себя другое. Например, у дома есть крыша. Дом — холоним для крыши. Компьютер — холоним для монитора. Мероним и холоним — противоположные понятия. В этом случае свойства первого множества не наследуются вторым.

Часто в семантических сетях требуется определить отношения синонимии и антонимии. Эти связи либо дублируются явно в самой сети, либо в алгоритмической составляющей.

1.9.2 Вспомогательные

В семантических сетях часто используются также следующие отношения:

- функциональные связи

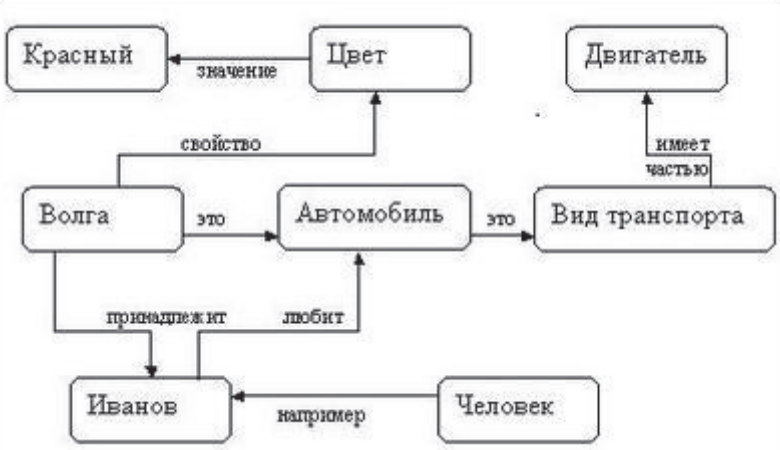

Рисунок 1.8.2 – Семантическая сеть

(определяемые обычно глаголами «производит», «влияет»…);

- количественные (больше меньше, равно…);
- пространственные (далеко от, близко от, за, под, над…);
- временные (раньше, позже, в течение…);
- атрибутивные (иметь свойство, иметь значение);
- логические (И, ИЛИ, НЕ);
- лингвистические.

Этот список может сколь угодно продолжаться: в реальном мире количество отношений огромно. Например, между понятиями может использоваться отношение «совершенно разные вещи» или подобное: Не_имеют_отношения_друг_к_другу(Солнце, Кухонный_чайник).

Пример. На Рисунке 1.8.2 изображена семантическая сеть. В качестве вершин понятия: Человек, Иванов, Волга. Автомобиль, Вид транспорта. Двигатель.

* - Отношение между надмножеством и подмножеством называется **AKO** — «A Kind Of» («разновидность») (пример: "собака является животным"=" тип с именем «собака» является подтипом типа «животные»").

1.10 Стандартные способы описания связей между объектами данных: онтология, определяемая с помощью онтологического языка Web

Синтаксическое взаимодействие сетей - необходимое условие для того, чтобы множественные приложения могли по-настоящему "понимать" данные и работать с ними как с информацией. Это также необходимое условие для корректной проверки данных. Синтаксическое взаимодействие сетей требует преобразования ("отображения") между терминами, для чего, в свою очередь, необходим контент-анализ.

Такой контент-анализ требует формальных и подробных спецификаций моделей доменов, которые определяют используемые термины и их связи. Подобные формальные модели доменов иногда называются *онтологиями*. Они определяют модели данных в терминах классов, подклассов и свойств.

Онтологический язык Web (Web Ontology Language), рекомендуемый консорциумом W3C, помогает в выражении онтологий. Рабочий онтологический язык (Ontology Working Language, сокр. OWL) добавляет больше словарных возможностей для описания свойств и классов, чем RDF или схема RDF. В частности, он позволяет описывать связи между классами (например, неперекрываемость), мощность множества (например, "ровно один"), равенство, более богатую типологию свойств и их характеристики (например, симметрия).

Онтологический язык Web на основе OWL разработан для использования приложениями, которые должны работать с содержанием информации, а не просто предоставлять ее пользователю. OWL улучшает возможности автоматической интерпретации содержимого интернета по сравнению с теми, что могут обеспечить XML, RDF и схема RDF. Это происходит благодаря тому, что OWL предоставляет дополнительные словарные возможности наряду с

формальной семантикой. OWL включает три подъязыка: полный OWL (OWL Full), OWL DL и облегченный OWL (OWL Lite) (перечислены в порядке убывания их выразительных возможностей).

Полная версия онтологического языка Web на основе OWL называется **OWL Full**. Этот язык использует все базисные элементы языка OWL и позволяет комбинировать их случайным образом с RDF и схемой RDF. Полный OWL совместим "снизу вверх" с RDF, как синтаксически, так и семантически: любой разрешенный документ RDF является также разрешенным документом OWL Full. Маловероятно, что какие-либо интеллектуальные программные средства способны поддерживать все возможности OWL Full, поскольку этот язык предлагает максимум выразительных средств и синтаксической свободы RDF при отсутствии вычислительных гарантий.

OWL DL предназначен для тех пользователей, кому необходим максимум выразительных средств без потери вычислительных возможностей. OWL DL - это подъязык конструкций языка OWL Full с некоторыми ограничениями, такими как *разделение типов* (type separation) (например, класс не может быть одновременно индивидуальным элементом или свойством, а свойство не может одновременно быть индивидуальным элементом или классом).

OWL Lite предназначен для пользователей, которым необходима классификационная иерархия и простые ограничительные возможности. Преимуществом этого языка являются большая легкость его понимания и внедрения по сравнению с двумя другими. Но в то же время его выразительные возможности гораздо ниже. Например, хотя OWL Lite и поддерживает ограничения мощности множества, единственными допустимыми значениями этого параметра являются 0 или 1.

1.11 Компоненты онтологического языка Web на основе OWL

Основные компоненты OWL включают классы, свойства и индивидуальные элементы. Рассмотрим каждый компонент подробнее.

1.11.1 Классы

Классы - это основные блоки онтологии OWL. Класс - это концепция в домене. Классы обычно образуют таксономическую иерархию (т.е. систему подкласс-надкласс).

Классы определяются с помощью элемента owl:Class. В языке OWL существует два заранее определенных класса: owl:Thing и owl:Nothing. Первый из них является наиболее общим и включает все, второй - это пустой класс. Любой класс, определяемый пользователем, является подклассом класса owl:Thing и надклассом класса owl:Nothing. Примеры классов в области банковского дела могут включать классы Account или Customer.

В листинге 1.11.1 представлен пример класса OWL.

Листинг 1.11.1 – Пример класса OWL

```
<owl:Class rdf:ID="SavingsAccount">
 <rdfs:subclassOf rdf:resource="#Account"/>
</owl:Class>
```

Таким образом код в листинге 1 указывает, что элемент SavingAccount - это класс, являющийся подклассом класса Account.

Также стоит отметить, что OWL поддерживает шесть основных способов описания классов. Самый простой - это класс с именем (named). Другие типы - это классы пересечений (intersection), объединений (union), дополнений (complement), ограничений (restrictions) и классы перечислений (enumerated). В листинге 1 представлены два из этих способов описания классов: класс ограничений определяет SavingAccount как подкласс класса с именем Account.

1.11.2 Свойства

Свойства включают две основные категории:

- свойства объекта (Object properties), которые связывают индивидуальные элементы между собой;
- свойства типов данных (Datatype properties), которые связывают индивидуальные элементы со значениями типов данных, такими как целые числа, числа с плавающей запятой и строки. Для определения типов данных OWL использует схему XML.

Свойство может включать домен и некоторую область, связанную с ним. Любое свойство попадает в одну из следующих категорий:

- функциональная**:** для любого объекта свойство может принимать только одно значение (например, возраст, рост или вес человека);
- обратно-функциональная**:** два различных индивидуальных элемента не могут иметь одно и то же значение. Например, у каждого человека свой уникальный номер банковского счета bankNumber или так называемый SSN (social security number);
- симметричная: если свойство связывает элемент A с элементом B, то из этого можно сделать вывод, что оно также связывает элемент B с элементом A. Примеры симметричных свойств включают выражения типа "является братом (сестрой)" или "такой же, как";
- транзитивная: если свойство связывает элемент A с элементом B, а элемент B с элементом C, то можно предположить, что оно также связывает элемент A с элементом C. Например, если A выше B, а B выше C, то A выше C.

К классам и свойствам могут применяться различные ограничения. Например, ограничения мощности множества указывают на число связей, в которых может участвовать класс или индивидуальный элемент.

1.11.3 Индивидуальные элементы

Индивидуальные элементы - это элементы классов; свойства могут связывать их друг с другом. Например, индивидуальный элемент Smith может быть описан как элемент, принадлежащий классу Person (индивидуум).

Свойство hasEmployer (имеет работодателя) может связывать его с другим индивидуальным элементом - Webify Solutions, указывая, таким образом, что Smith работает в компании Webify Solutions.

1.12 Значение онтологии для бизнеса

IT-системы организуют значения с помощью реляционных моделей данных, линейных файлов, объектно-ориентированных моделей или специально разработанных моделей данных. Время от времени, в связи с изменениями бизнес-требований, возникает необходимость добавления новых элементов и связей в реляционные модели данных или объектно-ориентированные модели.

Более того, если организация использует множественные приложения от различных поставщиков, то придется копировать одни и те же модели во все базы данных приложений. Например, банк предлагает набор различных продуктов для обслуживания разнообразных категорий клиентов. Корпоративному клиенту может потребоваться услуга по обнаружению мошенничества, а обычному потребителю окажется достаточно функциональных возможностей интерактивного осуществления банковских операций с помощью интернета. Обычно банк приобретает приложения у нескольких поставщиков, но каждое из них повторяет одну и ту же общую информацию - номера счетов, имена клиентов и т. д. - в своей базе данных. По мере того как организация добавляет новые продукты для удовлетворения растущих запросов бизнеса, одна и та же избыточная информация распространяется по всей корпорации.

Целый ряд услуг является общим для всех приложений, например, просмотр банковских транзакций и электронных переводов. Каждая из этих услуг также дублируется в формате, присущем тому или иному приложению, что ведет к необходимости осуществления точечной интеграции.

Если же банк принимает подход, основанный на онтологии, то он может собирать и представлять общую информацию о продуктах в нейтральной по отношению к языку форме и сохранять эту информацию в центральном

репозитории. С помощью такой общей адаптированной онтологии организация может обеспечивать единое стандартизированное представление данных для всех приложений. Такое стандартизированное представление позволяет точно извлекать необходимую информацию и без проблем осуществлять корпоративную интеграцию, поскольку бизнес-процессы и различные источники данных могут быть связаны ("отображены") друг с другом с помощью общей мета-модели. Таким образом, общая онтология исключает необходимость в точечной интеграции и упрощает интеграцию приложений, сокращая избыточность данных и обеспечивая одно и то же семантическое значение для всех приложений, что облегчает поддержание функционирования банка и его обновление.

1.13 Преимущества семантических сетей для интернета

Интернет - это крупнейший из когда-либо существовавших информационных репозиториев, причем его содержание все время растет и представлено на самых разнообразных языках и практически во всех областях знаний. Но в конечном счете становится все труднее находить смысл во всем этом содержимом. Поисковые системы способны находить информацию, содержащую определенные слова, но эта информация не всегда оказывается именно той, что требуется. Какой-то элемент всегда оказывается упущенным. Поиск основан на содержании страниц, но не на семантическом значении этого содержания или информации о странице.

Как только будет создан семантический интернет, он даст возможность разметки всего содержания интернета, описания каждого элемента информации и обеспечения семантического значения этих элементов. Таким образом, поисковые системы становятся более эффективными, чем сейчас, а пользователи могут находить именно ту информацию, которая им необходима. Организации, оказывающие различные услуги, способны индексировать их с особым значением. А пользователи будут в состоянии оперативно находить эти услуги,

используя программные средства на основе интернета, и использовать их для своей пользы или в сочетании с другими услугами.

1.14 Роль и значение семантических технологий для сервис-ориентированной архитектуры

Для того чтобы соответствующим образом моделировать и управлять СОА (сервис-ориентированной архитектурой), корпоративные архитекторы должны поддерживать активное представление услуг, доступных для корпорации. В частности, для выявления и организации своих услуг, архитекторы должны использовать передовой опыт в моделировании и объединении услуг с использованием метаданных, преобразовывать бизнес-логику в метаданные для динамического объединения и осуществлять управление с помощью метаданных. Онтология обеспечивает очень мощный и гибкий способ для агрегирования, визуализации и нормализации этого слоя услуг с помощью метаданных.

Онтология - это сеть концепций, связей и ограничений, которые обеспечивают контекст для данных и информации, а также для процессов. Онтология способствует улучшению обнаружения услуг, моделирования, объединения, посредничества и семантического взаимодействия сетей. Она усовершенствует для пользователей способы поиска, изучения и взаимодействия со сложными информационными пространствами метаданных. Бизнес-онтология - это формальная спецификация бизнес-концепций и их взаимосвязей, которая улучшает машинные причинно-следственные связи и взаимодействия. Бизнес-онтология связывает системы, используя метаданные, во многом аналогично тому, как база данных объединяет разрозненные данные. Такая абстракция обеспечивает гибкость и подвижность, поскольку позволяет легко менять интерфейсы, а также добавлять новые ресурсы и пользователей, причем даже во время работы системы.

Семантика - это будущее сервис-ориентированной интеграции. Семантические технологии обеспечивают существование определенного уровня абстракции над

существующими IT-технологиями. Этот уровень позволяет осуществлять связь данных, содержания и процессов между различными видами бизнеса и изолированными IT-структурами. Наконец, с точки зрения взаимодействия людей, семантические технологии добавляют новый уровень семантических порталов, которые обеспечивают гораздо более аналитические, соответствующие теме и контексту взаимодействия, чем те, которые доступны с помощью традиционных точечных подходов к интеграции, использующихся в информационных порталах.

2 РАЗРАБОТКА ОБЪЕКТНОЙ МОДЕЛИ И ЕЕ ПРЕДСТАВЛЕНИЕ В РАМКАХ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ НА ПРИМЕРЕ ОНТОЛОГИЙ СЕМАНТИЧЕСКОЙ СЕТИ

Как уже было сказано ранее, для написания качественного кода программистам необходимо наличие объектной модели той предметной области, на которую будет опираться тот или иной программный продукт. Чтобы представить предметную область с помощью онтологий используются специальные программы для их редактирования, которые называются редакторами онтологий. В данной главе рассмотрим процесс создания такого редактора.

2.1 Анализ существующих редакторов онтологий

Как известно, разработка любого программного продукта начинается с анализа. Таким образом, проанализируем уже имеющиеся редакторы онтологий. Сравнительные характеристики представлены в таблице 2.1.

Таблица 2.1 – Сравнение имеющихся редакторов онтологий

Наименование редактора	Исходный код	ОС	Стоимость	Поддержка OWL
Protege	Открыт, java	Windows, Linux, MacOSX, Salaris, Unix	Бесплатно	Да
Fluent Editor	Закрыт	Windows	Бесплатно для некоммерческого использования	Да
TopBraid Composer	Закрыт, java	Windows	Платная, бесплатная версия имеет ограничения	Да

Как видно из таблицы 2.1, самым универсальным и распространенным из представленных редакторов является Protégé. Данная платформа поддерживает

два основных способа моделирования онтологий посредством редакторов Protégé-Frames и Protégé-OWL. Нас будет интересовать Protégé-OWL, который позволяет пользователям строить онтологии для семантической паутины, в частности на OWL. OWL-онтология может включать описания классов, свойств и их экземпляров. Давая такую онтологию, формальная семантика OWL определяет, как получать логические следствия, т.е. факты, которые не присутствуют непосредственно в онтологии, но могут быть выведены из существующих посредством семантики. Эти выводы могут быть основаны на одном документе или на множестве распределенных документов, которые были объединены с использованием определенных механизмов OWL.

OWL – язык описания онтологий для семантической паутины, позволяющий описывать классы и отношения между ними, присущие веб-документам и приложениям. Основой языка является представление действительности в модели данных «объект-свойство». OWL пригоден для описания не только веб-страниц, но и любых объектов действительности. Каждому элементу описания в этом языке (в том числе свойствам, связывающим объекты) ставится в соответствие URI.

На данный момент актуальной считается вторая версия языка OWL, в которой определяются следующие разновидности:

- **OWL 2 DL** предназначен для пользователей, которым нужна максимальная выразительность при сохранении полноты вычислений (все логические заключения, подразумеваемые той или иной онтологией, будут гарантированно вычислимыми) и разрешаемости (все вычисления завершатся за определенное время). OWL DL включает все языковые конструкции OWL, но они могут использоваться только согласно определенным ограничениям (например, класс может быть подклассом многих классов, но не может сам быть представителем другого класса). OWL DL так назван из-за его соответствия дескрипционной логике — дисциплине, в которой

разработаны логики, составляющие формальную основу OWL. Существует три подмножества OWL DL, называемые "профилями":

- **OWL EL**, предназначенный для использования в приложениях с большим количеством свойств и классов. На EL-онтологиях основные алгоритмы логического вывода гарантированно завершаются за полиномиальное время.
- **OWL QL**, особенно полезный для онтологий, содержащих множество индивидов. В этом профиле основной акцент приходится на обеспечение высокой скорости запросов к данным - они отрабатывают за логарифмическое время.
- **OWL RL** предназначен для запуска алгоритмов, основанных на языках правил. Включает отличный от EL набор средств и позиционируется как язык, позволяющий повысить выразительность существующих RDFS-онтологий.

- **OWL 2 Full** предназначен для пользователей, которым нужна максимальная выразительность и синтаксическая свобода RDF без гарантий вычисления. Например, в OWL Full класс может рассматриваться одновременно как собрание индивидов и как один индивид в своём собственном значении. OWL Full позволяет строить такие онтологии, которые расширяют состав предопределённого (RDF или OWL) словаря. Маловероятно, что какое-либо программное обеспечение будет в состоянии осуществлять полную поддержку каждой особенности OWL Full.
- В первой версии языка также присутствовало подмножество **OWL Lite,** призванное ограничить выразительность языка и повысить скорость алгоритмов. В новой версии стандарта OWL Lite отсутствует.

У Protege существует много недостатков:

1. Данный программный продукт написан на языке Java, что требует наличия на запускающем вычислительном устройстве виртуальной машины Java.

2. Вторым недостатком является сложный интерфейс программы, что делает ее достаточно сложной для пользования.

2.2 Определение функционала редактора и создание диаграммы использования на языке UML

После анализа существующих редакторов онтологий, уточнения всех требований у заказчика, составим диаграмму использования разрабатываемого программного продукта на языке UML, которая представлена на рисунке 2.2.

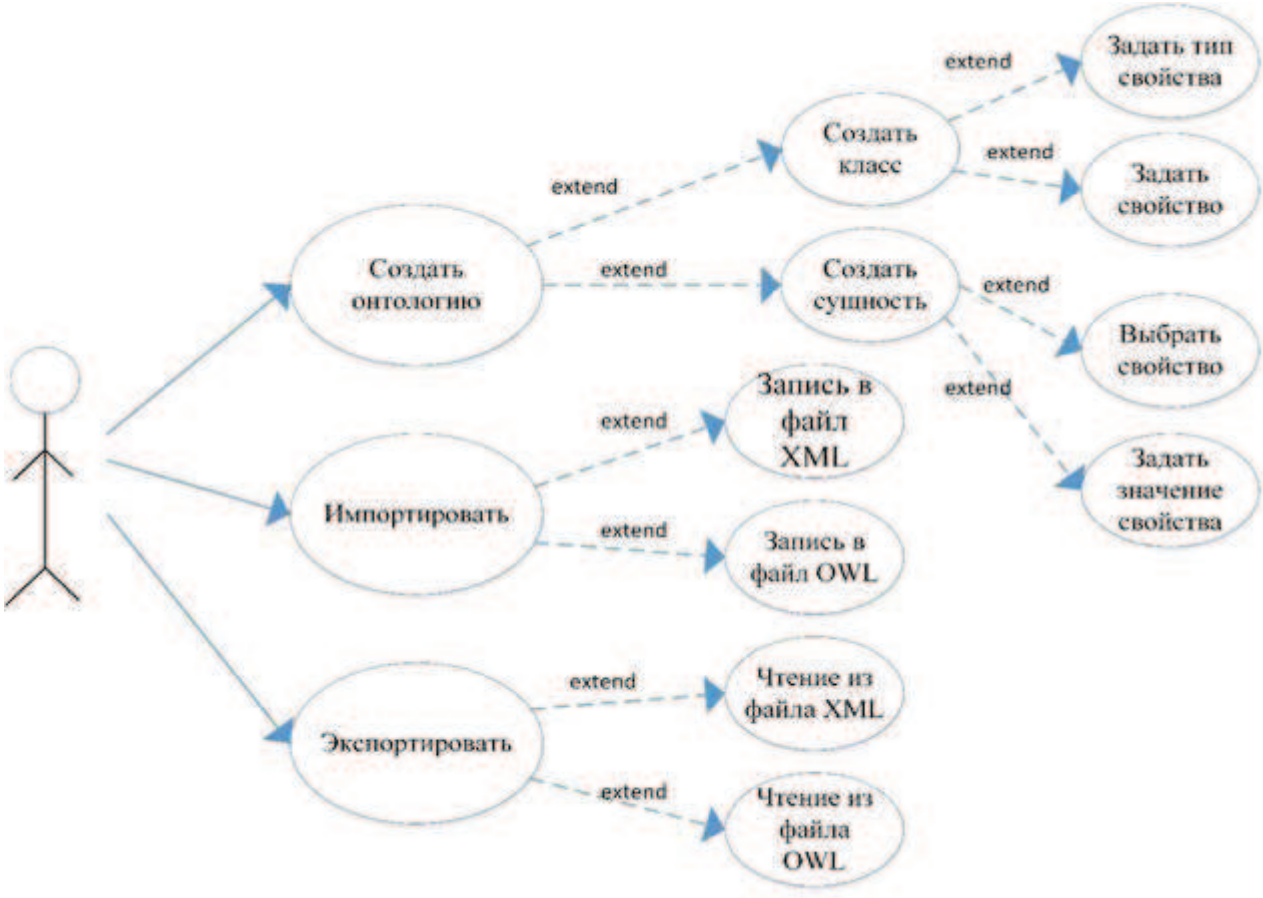

Рисунок 2.2 – Диаграмма использования UML разрабатываемого программного продукта

2.3 Разработка объектной модели

Исходя из диаграммы использования создается объектная модель системы. Для этого необходимо представить онтологии в программных системах. Онтологии применяются при процессе программирования, как форма представления знаний о реальном мире или его части. Чаще всего они применяются при моделировании бизнес-процессов, в семантической сети, в сфере искусственного интеллекта [3, с.303-306].

Понятие класс в онтологии и класс в объектно-ориентированном подходе схожи: это понятие означает некую группу объектов, описанных с необходимым уровнем абстракции, имеющие одинаковые свойства и связи с другими объектами [2, с.306-308]. В терминах программирования класс и есть абстракция или, еще это называют, типом данных.

Экземпляры онтологии в терминах программирования называются объектами, т.е. конкретными экземплярами того или иного типа данных (класса).

Свойства экземпляров у онтологий (например, в языке описания онтологий OWL) выделяют двух типов: объектные свойства (отношения) и свойства-значения. Первые связывают между собой экземпляры, а вторые связывают экземпляры со значениями данных [0].

В языках программирования объектные свойства – это методы класса, через которые класс может взаимодействовать с другими классами. Свойства-значения же подходят под описание полей класса, в которых класс хранит те или иные данные.

Таким образом, можно сделать отображение структуры онтологии на объектную модель объектно-ориентированного подхода.

Проведя анализ всего вышесказанного, создается объектная модель программного продукта, представленная на рисунке 2.3.1.

Структуру самой онтологии будет хранить пространство имен Ontology.Data, в котором будут храниться классы, объекты, свойства объектов и свойства данных, которые будут реализованы с помощью классов.

Важным является отделить хранение метаданных (данные о данных) онтологии, т.е. классов, отношений между классами и свойств, от данных, т.е. конкретных экземпляров классов.

Для хранения метаданных объектов предметной области будет использоваться тип данных OntologyClass, он будет хранить свойства классов онтологии и связи между данными классами. За хранение непосредственно данных, т.е. информации о конкретных экземплярах того или иного класса будет отвечать OntologyInstance.

Свойства классов также имеют метаданные (имя и тип той информации, которая хранится в свойстве) и собственно данные свойства – то значение, которым обладает это свойство. За хранение метаданных свойств будет отвечать тип данных OntologyProperty, а за данные OntologyPropertyValue. Логично, что метаданные свойств хранит OntologyClass, а данные OntologyInstance, т.е. тип данных олицетворяющий экземпляр онтологии.

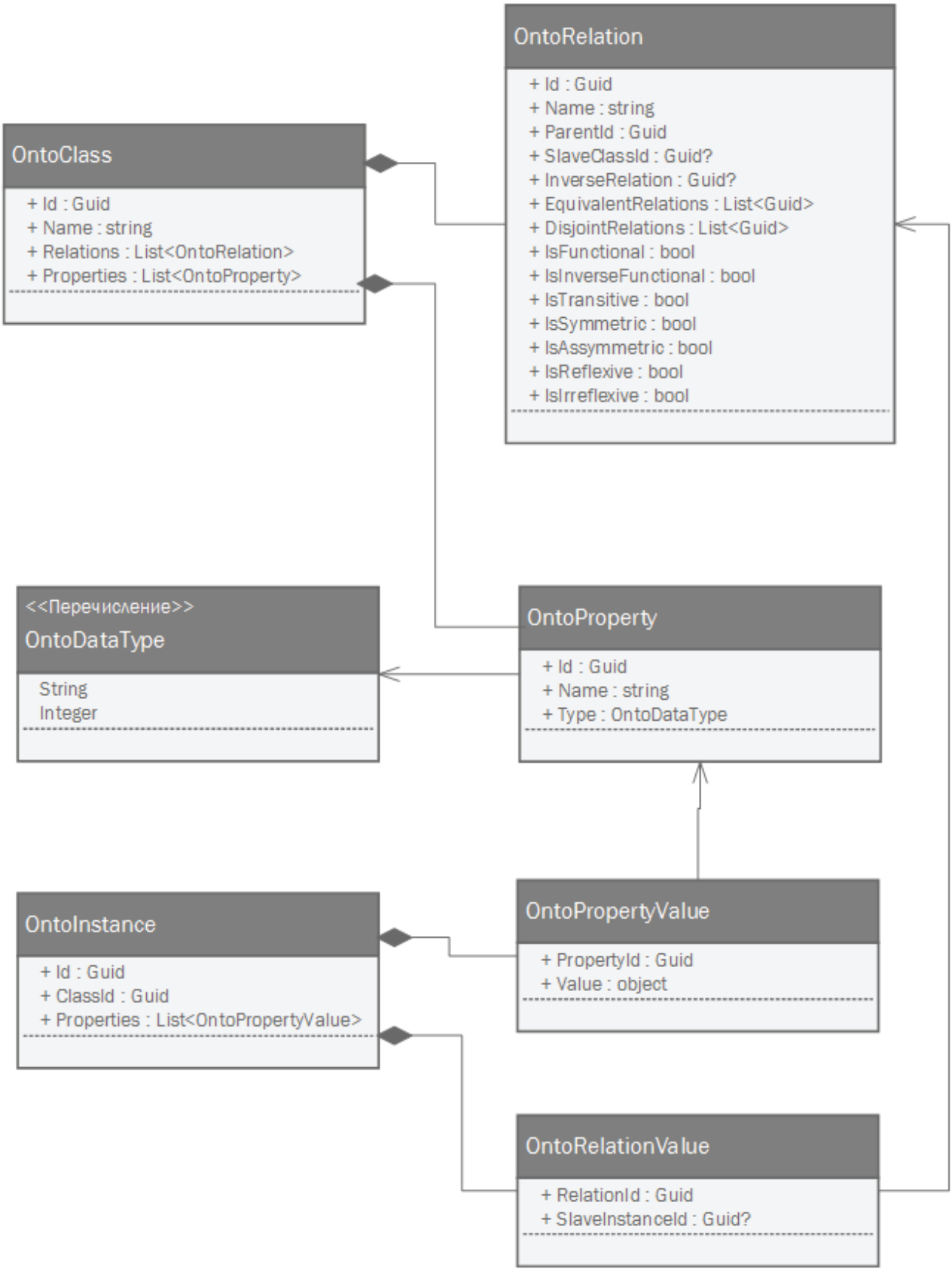

Рисунок 2.3.1 – Объектная модель разрабатываемого редактора онтологий

Также в онтологии важно и необходимо установить связи между объектами, которые будут задаваться в отдельном классе OntologyRelation, который будет хранить имя связи и ссылку на класс онтологии, с которым установлена связь при помощи этого отношения. Однако, ссылка на такой класс не даст однозначного ответа с каким же конкретным экземпляром установлена связь, т.е. это лишь метаданные о том, с кем происходит отношение. Чтобы хранить данные о конкретном объекте, с которым должен взаимодействовать «владелец» связи, будет использоваться тип данных OntologyRelationValue, в котором хранится ссылка на экземпляр онтологии, с которым будет взаимодействовать «владелец» связи.

Для работы с онтологиями существует язык описания онтологий OWL [3], который нашел свое широкое применение. В данном языке описания присутствуют такие понятия как домен и диапазон. Домен определяет множество объектов, которые могут является «родителем» связи между объектами, а диапазон соответственно определяет множество «подчиняющихся» объектов для отношения. Но эти понятия не имеют своего отдельного отражения в объектно-ориентированном подходе.

В ООП понятия домен и диапазон уже заложены в иерархии классов и объектов, которая однозначно определяет связи между объектами, т.к. те уже заданы в классах, поэтому уточнять домен и диапазон для классов не требуется и эти понятия в объектной модели можно опустить.

Получив объектную модель системы, можно перейти непосредственно к реализации программного продукта. Объектная модель представляется на языке программирования C#. Исходные коды создания объектной модели на языке программирования C# представлены в листинге 2.3.1.

Листинг 2.3.1 – Исходные коды создания объектной модели на языке программирования C#.

```
public class OntoClass
    {
```

```
    public Guid Id { get; set; }
    /// <summary>
    /// Имя класса
    /// </summary>
    public string Name { get; set; }
    /// <summary>
    /// Ссылка на OntoClass
    /// </summary>
    public Guid? ParentId { get; set; }
    /// <summary>
    /// Список связей между классами
    /// </summary>
    public List<OntoRelation> Relations { get; set; }
    /// <summary>
    /// Список свойств класса
    /// </summary>
    public List<OntoProperty> Properties { get; set; }


}

public class OntoRelation
{
    public Guid Id { get; set; }
    /// <summary>
    /// Имя связи
    /// </summary>
    public string Name { get; set; }
    /// <summary>
    /// Ссылка на OntoRelation
    /// </summary>
    public Guid? ParentId { get; set; }
    /// <summary>
    /// Ссылка на OntoClass. Класс, с которым устанавливается связь
    /// </summary>
    public Guid? SlaveClassId { get; set; }
    /// <summary>
    /// Ссылка на OntoRelation. Отношение обратное данному
    /// </summary>
    public Guid? InverseRelation { get; set; }
    public bool IsFunctional { get; set; }
    public bool IsInverseFunctional { get; set; }
    public bool IsTransitive { get; set; }
    public bool IsSymmetric { get; set; }
```

```
        public bool IsAsymmetric { get; set; }
        public bool IsReflexive { get; set; }
        public bool IsIrreflexive { get; set; }
    }

    public class OntoProperty
    {
        public Guid Id { get; set; }
        /// <summary>
        /// Имя свойства
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Тип данных свойства
        /// </summary>
        public OntoDataType Type { get; set; }
    }

    public enum OntoDataType
    {
        String,
        Int,
        Float,
        List,
        Id,
        Class
    }
    public static class OntoDataTypeFactory
    {
        public static object GetDataTypeDefaultValue(OntoDataType dataType)
        {
            switch (dataType)
            {
                case OntoDataType.Int:
                case OntoDataType.Float:
                    return 0;
                case OntoDataType.String:
                    return "";
            }

            return null;
        }
    }
public class OntoInstance
```

```
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public Guid ClassId { get; set; }
    /// <summary>
    /// Список ссылок на OntoRelationValue
    /// </summary>
    public List<OntoRelationValue> Relations { get; set; }
    /// <summary>
    /// Список ссылок на OntoPropertyValue
    /// </summary>
    public List<OntoPropertyValue> Properties { get; set; }

}

public class OntoRelationValue
{
    /// <summary>
    /// Ссылка на OntoRelation
    /// </summary>
    public Guid RelationId { get; set; }
    /// <summary>
    /// Ссылка на OntoInstance
    /// </summary>
    public Guid? SlaveInstanceId { get; set; }
}

public class OntoPropertyValue
{
    /// <summary>
    /// Ссылка на OntoProperty
    /// </summary>
    public Guid PropertyId { get; set; }
    public object Value { get; set; }
}
```

Так как самым распространенным редактором онтологий является Protégé, то практически все онтологии (во всяком случае большинство онтологий) имеют формат OWL. Поэтому требуется возможность конвертации онтологии из формата OWL в формат нашей объектной модели.

Далее рассмотрим реализацию конвертера из формата OWL в формат нашей объектной модели.

2.4 Проектирование функции конвертации из формата OWL в объектную модель

Задачей конвертации является сопоставление всех элементов онтологии в формате OWL нашей объектной модели. Для этого написана функция, которая парсит OWL файл, вытаскивает значения каждого элемента онтологии и присваивается к такому же типу элемента объектной модели. В качестве аргумента у данной функции является файл формата OWL, в котором описана онтология на этом языке. Блок-схема алгоритма конвертации представлена на рисунке 2.4.1. Рассмотрим процесс конвертации подробнее.

Для начала считываем содержимое файла с помощью метода Load класса XDocument. Для этого создадим объект этого класса и в него запишем значение, которое возвращает сама функция XDocument.Load(). В листинге 2.4.1 показан исходный код считывания содержимого файла.

Рисунок 2.4.1 – Блок-схема алгоритма конвертации из файла OWL в объектную модель

Листинг 2.4.1 – Считывание содержимого OWL файла

```
XDocument doc = XDocument.Load(file);
```

В листинге 2.4.1 под именем file подразумевается путь до OWL файла в виде строки.

После того, как считано содержимое, анализируем каждую строку OWL файла, то есть построчно проверяем наличие совпадений. Для этого используем конструкцию, представленную в листинге 2.4.2.

Листинг 2.4.2 – Отбор элементов для построчного анализа

```
foreach (var element in doc.Root.Descendants())
```

В листинге 2.4.2 используется цикл foreach, который служит для циклического обращения к элементам коллекции, представляющей собой группу объектов. Коллекцию элементов получаем с помощью метода XDocument.Root.Descendants().

Далее анализируем имена каждого объекта. Берем поле Name переменной element, представляем его, как строку, используя метод ToString(), который переводит значение класса в строку. Затем используем метод String.Contains(), который возвращает значение, указывающее, содержит ли указанная строка значение подстроки переданной в качестве параметра. А в качестве параметра передаем те значения, значения которых мы будем сопоставлять с объектной моделью. То есть в качестве аргументов для этого метода будут, например, следующие: ObjectPropertyDomain, Declaration, ClassAssertion, SubObjectPropertyOf, SubClassOf, Class, ObjectProperty, DataProperty, NamedIndividual. Используя все вышесказанное, производим проверку на наличие в строке данных выражений. И если выражение содержится, то выставляем флаг в значение true.

В листинге 2.4.3 представлен исходный код проверки содержания в строке выражения.

Листинг 2.4.3 – Исходный код проверки и установления флагов

```
if (element.Name.ToString().Contains("ObjectPropertyDomain"))
            {
               isLinkingObjectPropertyDomain = true;
               continue;
            }

if (element.Name.ToString().Contains("Declaration"))
            {
               isLinkingDeclaration = true;
               continue;
            }

if (element.Name.ToString().Contains("ClassAssertion"))
            {
               isLinkingClassAssertion = true;
               continue;
            }

if (element.Name.ToString().Contains("SubObjectPropertyOf"))
            {
               isLinkingSubObjectProperty = true;
               continue;
            }

if (element.Name.ToString().Contains("SubClassOf"))
            {
               isLinkingSubClassOf = true;
               continue;
            }
```

После того, как мы зашли в узел, необходимо присвоить значения, которые содержатся в этом узле, объектной модели. Опишем процедуру конвертации на примере, когда мы зашли в узел, в котором содержится наименование класса онтологии. Для этого проверим условие на то, что в строке, которую мы анализируем содержится строка “Class”. Но данного условия недостаточно, так как наличие данной строки на сто процентов не гарантирует того, что мы находимся внутри узла, в котором содержится наименование класса. Для этого в первую очередь проверяем установлен ли флаг того, что мы зашли в узел онтологии. Если флаг, отвечающий за это имеет значение true (в нашем случае это

флаг isLinkingDeclaration), то выполняем следующее. В первую очередь сбрасываем данный флаг в значение false. Далее создаем объект созданного нами класса OntoClass, выделяем под него место в памяти командой “new OntoClass()”. Затем полю Name созданного объекта присваиваем значение, полученное в результате парсинга. Определяем это значение следующим образом. С помощью функции FirstAttribute возвращаем первый атрибут элемента, а с помощью метода Value возвращаем значение этого атрибута, тем самым получаем необходимое наименование элемента онтологии (в данном случае получаем имя класса онтологии). Также, так как связи между классами устанавливаются по идентификатору, то необходимо сгенерировать его. Для этих целей используем метод Guid.NewGuid (), который инициализирует новый экземпляр структуры глобального уникального идентификатора. Затем с помощью метода Add добавляем полученный класс в список всех классов онтологии.

Исходный код на языке C# всего вышеизложенного представлен в листинге 2.4.4.

Листинг 2.4.4 – Процесс конвертации класса онтологии из файла формата OWL в объектную модель

```
if (element.Name.ToString().Contains("Class"))
            {
               if (isLinkingDeclaration)
               {
                  isLinkingDeclaration = false;
                  OntoClass class1 = new OntoClass();
                  class1.Name = element.FirstAttribute.Value;
                  class1.Id = Guid.NewGuid();
                  classes.Add(class1);
               }
            }
```

Рассмотрим процесс конвертации связей между объектами в класс, отвечающий за это. Как уже было сказано ранее, установление связи между объектами задаются в отдельном классе OntologyRelation, который хранит имя

связи и ссылку на класс онтологии, с которым установлена связь при помощи этого отношения. Поэтому необходимо сконвертировать данные поля в объектную модель. Алгоритм такой же, как и применяется для классов. Для начала проверяем, что в строке, которую мы анализируем содержится строка “ObjectProperty”. Но это также не гарантирует, что мы находимся внутри узла, который отвечает за это. Для этого проверим установлен ли флаг вхождения в узел в состояние true (isLinkingDeclaration). Если условие выполняется выполняем следующие действия. В первую очередь сбрасываем флаг. Создаем новый объект класса OntoRelation и выделяем под него место в памяти с помощью “new OntoRelation()”. Далее присваиваем полю Name объекта класса OntoRelation значение первого атрибута элемента с помощью уже известных методов element.FirstAttribute.Value. Конвертация уникального глобального идентификатора аналогична предыдущему случаю.

Исходный код продемонстрирован в листинге 2.4.5.

Листинг 2.4.5 – Исходный код конвертации

```
if (element.Name.ToString().Contains("ObjectProperty"))
            {
               if (isLinkingDeclaration)
               {
                  isLinkingDeclaration = false;
                  OntoRelation relation = new OntoRelation();
                  relation.Name = element.FirstAttribute.Value;
                  relation.Id = Guid.NewGuid();
                  relations.Add(relation);
               }
            }
if (element.Name.ToString().Contains("DataProperty"))
            {
               if (isLinkingDeclaration)
               {
                  isLinkingDeclaration = false;
                  OntoProperty property = new OntoProperty();
                  property.Name = element.FirstAttribute.Value;
                  property.Id = Guid.NewGuid();
```

```
                properties.Add(property);
            }
        }


if (element.Name.ToString().Contains("NamedIndividual"))
        {
            if (isLinkingDeclaration)
            {
                isLinkingDeclaration = false;
                OntoInstance instance = new OntoInstance();
                instance.Name = element.FirstAttribute.Value;
                instance.Id = Guid.NewGuid();
                instances.Add(instance);
            }
        }
```

Рассмотрим конвертацию дочерних классов в объектную модель. В первую очередь проверяем, что в анализируемой строке присутствует наименование “Class”. Но это также не гарантирует, что мы находимся внутри узла, который отвечает за это. Для этого проверим установлен ли флаг вхождения в узел в состояние true (в данном случае isLinkingSubClassOf). Если условие выполняется выполняем следующие действия. Увеличиваем значение переменной i, которая служит счетчиком вхождений в данное условие, так как в данное условие нужно заходить 2 раза. Для этого проводим еще проверку на количество вхождений. Если переменная i=2, то сбрасываем флаг вхождения в данный узел в значение false и значение счетчика вхождений сбрасываем в ноль. Далее заносим информацию о классе в переменную subclass. Затем, соответственно, полю идентификатора родителя объекта класса присваиваем значение идентификатора дочернего класса. Таким образом, указываем какой класс является родителем определенного класса по его идентификатору.

Исходный код конвертации дочерних классов представлен в листинге 2.4.6.

Листинг 2.4.6 – Конвертация дочерних классов в объектную модель

```
if (element.Name.ToString().Contains("Class"))
```

```
            {
                if (isLinkingSubClassOf)
                {
                    i = i + 1;
                    if (i == 2)
                    {
                        isLinkingSubClassOf = false;
                        i = 0;
                    }
                    if (tempSubClassOf == null)
                    {
                        tempSubClassOf = classes.FirstOrDefault(x => x.Name ==
element.FirstAttribute.Value);
                    }
                    else
                    {
                        var subclass = classes.FirstOrDefault(x => x.Name ==
element.FirstAttribute.Value);
                        if (tempSubClassOf != null)
                        {

                            tempSubClassOf.ParentId = subclass.Id;
                            tempSubClassOf = null;
                        }

                    }
                }
            }
```

Таким образом, получили конвертер, позволяющий переводить онтологии, созданные в Protégé в объектную модель нашей системы, что, в свою очередь, позволяет открывать ранее созданные онтологии в другом продукте, переводить их в объектную модель и, как следствие, изменять ее, дополнять какими-либо элементами. На данный момент эта функция не позволяет переводить абсолютно все онтологии в объектную модель, так как рамки выпускной квалификационной работы не позволяют перебрать все возможные варианты структуры файла OWL, но со временем данная функция может быть дописана и преобразована, что позволит конвертировать абсолютно любую онтологию в формате OWL в объектную модель, которая будет применяться при проектировании программных

систем на объектно-ориентированных языках программирования. Также рамки выпускной квалификационной работы не позволяют создать графический пользовательский интерфейс, который в дальнейшем может создан.

3 ПРИМЕНЕНИЕ ОБЪЕКТНОЙ МОДЕЛИ НА ПРИМЕРЕ ПРЕДМЕТНЫХ ОБЛАСТЕЙ

В данной главе рассмотрим использование редактора онтологий на примере двух онтологий. Одну онтологию будем создавать методами нашего программного продукта, а вторую онтологию, созданную в Protégé, сконвертируем в нашу объектную модель.

Первой онтологией является всем, занимающимся в данной области, специалистам известная онтология пиццы. Попробуем ее создать в нашем редакторе онтологий, используя методы нашей объектной модели.

3.1 Создание онтологии пиццы в разрабатываемом программном продукте

Суть данной онтологии состоит в том, что существует несколько видов пицц. Какая-либо пицца может быть основой для другой. В каждую из пицц добавляются ингредиенты. Под ингредиентами понимаются пищевые продукты. Также в пиццу могут быть добавлены добавки, которые придают своеобразный индивидуальный вкус. Добавки также являются ингредиентами. Помимо всего вышесказанного в пиццу могут быть добавлены специи. Виды пиццы имеют разные страны происхождения. Каждый вид пиццы определяется по ее составу, а, конкретно, по ингредиентам, из которых состоит данный вид пиццы, по ограничениям на добавку того или иного ингредиента, а также специям и стране происхождения.

После того, как мы выяснили суть онтологии, которую нам необходимо построить, начнем ее создавать. Как известно, основой для любой онтологии являются классы. Поэтому начнем создание онтологии с создания этих классов. В Protégé пустая онтология уже имеет пустой класс Thing, от которого все остальные классы наследуются, но в нашем программном продукте такого класса нет. Таким образом, классы сразу добавляются в объектную модель и хранятся в ней.

Создадим класс пиццы. Для этого определяем объект класса OntoClass и выделяем под него место в памяти. Учитывая то, как связываются объекты в нашей модели сгенерируем глобальный уникальный идентификатор GUID для этого класса. Пропишем наименование класса, то есть полю Name присвоим значение “Pizza”. Так как данный класс ни от какого класса не наследуется, то есть он не является подклассом какого-либо класса, то полю ParentId присваиваем значение null. Следующим полем класса OntoClass является поле отношений, которое представляется списком всех отношений для этого класса. В данном случае создадим в этом поле новый список отношений, также выделив место в памяти под него. Внутри данного списка создадим объект класса OntoRelation и заполним следующие поля. Первым полем является опять таки глобальный идентификатор, который мы генерируем для каждого объекта класса отношений. Вторым полем является поле ParentId, которое ссылается на идентификатор родительской связи. Третьим полем является наименование связи. В данном примере заполним это поле “HasTopping”. Следующим полем, которое необходимо заполнить является идентификатор подчиненного класса, то есть ссылка на класс онтологии, с которым установлена связь при помощи этого отношения. Ниже будет рассмотрено создание класса, к которому производится данная ссылка. Также в классе отношений присутствуют еще поля, но в конкретном примере они нам не потребуются.

Исходный код создания объекта pizzaClass, относящегося к классу OntoClass представлен в листинге 3.1.1.

Листинг 3.1.1 – Создание класса pizzaClass

```
OntoClass pizzaClass = new OntoClass
                {
                        Id = Guid.NewGuid(),
                        Name = "Pizza",
                        ParentId = null,
                        Relations = new List<OntoRelation>
                        {
                                new OntoRelation
```

```
                {
                    Id = Guid.NewGuid(),
                    ParentId = null,
                    Name = "HasTopping",
                    SlaveClassId = toppingClass.Id
                }
            }
        };
```

После того, как создали класс пиццы, создадим класс, с которым связан предыдущий класс при помощи связи HasTopping. Для этого создадим также новый объект класса OntoClass, выделим место в памяти для него и заполним следующие поля этого объекта. Первым полем является индентификатор класса, по которому мы будем идентифицировать данный класс и к которому ранее в предыдущем классе мы ссылались как к подчиненному классу , с которым установлена связь при помощи отношения HasTopping. Вторым полем является наименование класса. Присвоим имя классу “Topping”. Так как данный класс не является дочерним классом любого другого класса, поэтому поле ParentId оставляем со значением null.

Таким образом, получили новый класс. Теперь попробуем прописать свойства этого класса. Для этого создадим новый объект класса OntoProperty, который, как уже упоминалось, хранит метаданные каждого свойства, а именно имя и тип той информации, которая хранится в свойстве. И соответственно заполним данные поля. Первым поле также является идентификатор свойства, и мы генерируем новый идентификатор для этого свойства с помощью функции Guid.NewGuid(). Вторым полем является наименование свойства OntoProperty.Name, а именно “Quantity”, то такое свойство, как количество пицц. Как известно, количество прописывается натуральным числом, то не имеет дробной части, поэтому для данного свойства типом будет являться Int, следовательно, полю OntoProperty.Type присвоим значение Int из класса OntoDataType, который, в свою очередь, хранит список всех возможных типов, применяемых в свойствах.

Процесс создания класса toppingClass приведен в листинге 3.1.2.

Листинг 3.1.2 – Создание класса toppingClass

```
OntoClass toppingClass = new OntoClass
                {
                        Id = Guid.NewGuid(),
                        Name = "Topping",
                        ParentId = null,
                        Properties = new List<OntoProperty>
                        {
                                new OntoProperty
                                {
                                        Id = Guid.NewGuid(),
                                        Name = "Quantity",
                                        Type = OntoDataType.Int
                                }
                        }
                };
```

Теперь попробуем создать дочерний класс для класса toppingClass. Как понятно из всего вышеизложенного, данный класс отвечает за пиццы с добавками. Но ведь добавки могут быть разными, соответственно, и пиццы могут быть разными в зависимости от того, какая добавка применена к пицце. Рассмотрим как это будет выглядеть на нашей объектной модели.

Ранее мы создали класс toppingClass, который представляет собой класс пицц с добавками, то есть он содержит в себе некое множество всех возможных видов пицц в зависимости от того, какую добавку имеет та или иная пицца. Таким образом, все эти виды являются дочерними классами класса toppingClass.

Перейдем непосредственно к появлению такого дочернего класса в онтологии. Создадим новый класс meatToppingClass, не забудем выделить под него место в памяти и заполним поля данного класса. В первое поле, которое отвечает за идентификатор класса генерируем Guid с помощью функции Guid.NewGuid(). Во второе поле прописываем наименование класса “NameTopping”. А вот в третьем поле ParentId до этого момента мы прописывали значение null, но сейчас, так как мы создаем дочерний класс, необходимо указать идентификатор родительского

класса, а мы знаем, что родителем является класс toppingClass, поэтому значением этого поля будет toppingClass.Id. Таким образом, получили дочерний класс.

Процесс создания дочернего класса приведен в листинге 3.1.3.

Листинг 3.1.3 – Создание дочернего класса

```
OntoClass meatToppingClass = new OntoClass
                {
                        Id = Guid.NewGuid(),
                        Name = "MeatTopping",
                        ParentId = toppingClass.Id
                };
```

После того, как созданы классы, перейдем к созданию экземпляров этих классов. Как мы знаем за хранение непосредственно данных, т.е. информации о конкретных экземплярах того или иного класса у нас отвечает класс OntoInstance. Поэтому создадим новый объект этого класса и не забудем выделить под него место в памяти. Также, как и на предыдущих этапах создания онтологии, сгенерируем этому экземпляру глобальный идентификатор с помощью функции Guid.NewGuid(). Пропишем наименование этого экземпляра “Ham”. Третьим полем экземпляра является поле значения его свойств. Значение каждого свойства хранится в списке объектов класса OntoPropertyValue. Поэтому создадим новый элемент этого класса и заполним поля следующим образом. Полю PropertyId необходимо присвоить значение идентификатора, которое хранится в поле Properties класса toppingClass. Так как мы создали там только одно свойство с наименованием “Количество”, и оно было добавлено самым первым в список свойств, следовательно, оно находится на нулевой позиции данного списка свойств. Поэтому значение PropertyId будет toppingClass.Properties[0].Id. Ну и вторым полем класса OntoPropertyValue является само значение Value данного свойства. В нашем примере, как было сказано ранее, данное поле имеет тип Int. Поэтому присвоим ему значение 30.

Процесс создания экземпляра приведен в листинге 3.1.4.

Листинг 3.1.4 – Создание экземпляра “Ветчина”

```
OntoInstance meatToppingInstance = new OntoInstance
                {
                        Id = Guid.NewGuid(),
                        Name = "Ham",
                        Properties = new List<OntoPropertyValue>
                        {
                                new OntoPropertyValue
                                {
                                        PropertyId = toppingClass.Properties[0].Id,
                                        Value = 30
                                }
                        }
                };
```

Разберем добавление еще одного экземпляра. Для этого создадим новый объект класса OntoInstance, не забудем выделить место в памяти. Сгенерируем экземпляру идентификатор с помощью функции Guid.NewGuid(). Присвоим полю Name наименование экземпляра “Mazarella”. Следующему полю ClassId присваиваем идентификатор того класса, которому принадлежит этот экземпляр. В нашем случае это Id класса пиццы pizzaClass. Далее заносим в поле relations список значений типа данных OntologyRelationValue, в котором хранится ссылка на экземпляр онтологии, с которым будет взаимодействовать «владелец» связи, чтобы хранить данные о конкретном объекте, с которым должен взаимодействовать «владелец» связи.

Процесс создания экземпляра для класса pizzaClass приведен в листинге 3.1.5.

Листинг 3.1.5 – Создание экземпляра для класса pizzaClass

```
OntoInstance pizza = new OntoInstance
                {
                        Id = Guid.NewGuid(),
                        Name = "Mazarella",
                        ClassId = pizzaClass.Id,
                        Relations = new List<OntoRelationValue>
                        {
                                new OntoRelationValue
```

```
                {
                    RelationId = pizzaClass.Relations[0].Id,
                    SlaveInstanceId = meatToppingInstance.Id
                }
            }
        };
```

После того как мы создали объектную модель онтологии для предметной области "Пицца", рассмотрим пример работы с объектами онтологии. Для этого создадим новый объект класса Ontology, который представляет собой структуры онтологии, то есть хранит в себе классы, объекты, свойства объектов и свойства данных. Состав полей класса Ontology приведен в листинге 3.1.6. Как было замечено ранее, элементы структуры онтологии, реализованы с помощью классов.

Листинг 3.1.6 – Поля класса Ontology

```
public List<OntoClass> Classes { get; set; }
public List<OntoInstance> Instances { get; set; }
public List<OntoRelation> Relation { get; set; }
```

После того, как создан объект Ontology, начнем добавлять все элементы онтологии в структуру. В первую очередь добавим классы онтологии в соответствующий список Classes с помощью метода Add. Данный метод добавляет объект в конец коллекции. Таким образом мы добавили классы в нашу онтологию. Пример добавления приведен в листинге 3.1.7.

Листинг 3.1.7 – Добавление классов в список Classes структуры онтологии

```
Ontology ontology = new Ontology();
ontology.Classes.Add(pizzaClass);
ontology.Classes.Add(toppingClass);
ontology.Classes.Add(meatToppingClass);
```

После добавления классов в онтологию, добавим экземпляры онтологии с помощью метода CreateInstance. В качестве аргументов данный метод принимает объект класса OntoClass и строку с наименованием экземпляра. В теле метода

производятся проверки на наличие аргументов, передаваемых методы. Также метод работает с дочерними классами. После выполнения всех проверок заполняем ссылку на подневольный экземпляр значением по умолчанию. Далее заполняем поле экземпляра значением по умолчанию. Затем проводим следующую проверку. Если у класса есть родительский класс, то в потомке учитываем все связи и свойства родителя. И в завершении мы создаем новый экземпляр со всеми необходимыми полями.

Исходный код процедуры CreateInstance представлен в листинге 3.1.8.

Листинг 3.1.8 – Процедура CreateInstance

```
public OntoInstance CreateInstance(OntoClass ontoClass, string instanceName)
        {
            if (ontoClass == null)
                throw new ArgumentNullException();

            var temp = Classes.FirstOrDefault(x => x == ontoClass);

            if (temp == null)
                throw new OntoClassNotFound(String.Format("ClassId {0} was not found in current ontology. Instance cannot be created.", ontoClass));

            OntoInstance parent = null;
            if (ontoClass.ParentId != null)
            {
                OntoClass parentClass = Classes.FirstOrDefault(x => x.Id == ontoClass.ParentId);
                parent = CreateInstance(parentClass, "parent");
            }

            List<OntoRelationValue> relations = ontoClass.Relations?.Select(relation => new OntoRelationValue
            {
                RelationId = relation.Id,
                SlaveInstanceId = null
            }).ToList();

            List<OntoPropertyValue> properties = ontoClass.Properties?.Select(property => new OntoPropertyValue
            {
```

```
                PropertyId = property.Id,
                Value = OntoDataTypeFactory.GetDataTypeDefaultValue(property.Type)
            }).ToList();


            if (parent != null)
            {
                if (parent.Relations != null)
                {
                    if (relations == null)
                        relations = new List<OntoRelationValue>();

                    relations.AddRange(parent.Relations);
                }

                if (parent.Properties != null)
                {
                    if (properties == null)
                        properties = new List<OntoPropertyValue>();

                    properties.AddRange(parent.Properties);
                }
            }

            var instance = new OntoInstance
            {
                Id = Guid.NewGuid(),
                Name = instanceName,
                ClassId = ontoClass.Id,
                Relations = relations,
                Properties = properties
            };
            return instance;
        }

    public OntoInstance CreateInstance(string ontoClassName, string instanceName)
        {
            var ontoClass = Classes.FirstOrDefault(x => x.Name == ontoClassName);

            if (ontoClass == null)
                throw new OntoClassNotFound(String.Format("ClassId {0} was not found
in current ontology. Instance cannot be created.", ontoClassName));

            return CreateInstance(ontoClass, instanceName);
```

```
}
```

Способ применения метода CreateInstance показан в листинге 3.1.9.

Листинг 3.1.9 – Применение метода CreateInstance

```
var hamInstance = ontology.CreateInstance("MeatTopping", "Ham");
```

Теперь добавим свойства классов. Для этого создадим переменную property для хранения результата работы функции ontology.GetInstanceProperty, которая возвращает значение свойства экземпляра. Исходный код данной функции представлен в листинге 3.1.10.

Листинг 3.1.10 – Тело процедуры ontology.GetInstanceProperty

```
public OntoPropertyValue GetInstanceProperty(OntoInstance instance, string propertyName)
        {
           if (instance.Properties == null)
              return null;

           foreach (var property in instance.Properties)
           {
              var classProperty = GetClassProperty(instance.ClassId, propertyName);
              if (classProperty != null)
                 if (classProperty.Name == propertyName) return property;
           }

           return null;
        }
```

Процесс добавления отношения похож на процесс добавления экземпляров и классов. Поэтому подробно останавливаться на этом не будем. Исходные коды данных действий приведены в листинге 3.1.11.

Листинг 3.1.11 – Добавление свойств и отношений в онтологию

```
var property = ontology.GetInstanceProperty(hamInstance, "Quantity");
        if (property != null)
```

```
                property.Value = 30;

        var pizzaInstance = ontology.CreateInstance("Pizza", "Mazarella");
                var relation = ontology.GetInstanceRelation(pizzaInstance,
"HasTopping");
                if (relation != null)
                        relation.SlaveInstanceId = hamInstance.Id;

                ontology.Instances.Add(hamInstance);
                ontology.Instances.Add(pizzaInstance);

                Console.WriteLine(ontology.ToString());
                Console.ReadLine();
```

Как видно из листинга 3.1.11, мы преобразуем полученную онтологию в строку, чтобы ее вывести на консоль. Также процедура перевода в строку представлена в листинге 3.1.12.

Листинг 3.1.12 – Процедура перевода онтологии в строку для вывода содержимого онтологии.

```
public override string ToString()
        {
          StringBuilder sb = new StringBuilder();

          sb.AppendLine("|==============================|");
          sb.AppendLine("|===========Classes============|");
          sb.AppendLine("|==============================|");
          sb.AppendLine();
          foreach (var ontoClass in Classes)
          {
            sb.AppendLine("CLASS {0}".FormatWith(ontoClass.Name));
            if (ontoClass.ParentId != null)
              sb.AppendLine("\t is subclass of {0}"
                .FormatWith(GetClass(ontoClass.ParentId.Value).Name));

            if (ontoClass.Relations != null)
              foreach (var relation in ontoClass.Relations)
                if (relation.SlaveClassId != null)
                  sb.AppendLine("\t {0} with class {1}"
                    .FormatWith(relation.Name,
GetClass(relation.SlaveClassId.Value).Name));
```

```
            if (ontoClass.Properties != null)
                foreach (var property in ontoClass.Properties)
                    sb.AppendLine("\t has property {0} of type {1}"
                        .FormatWith(property.Name, property.Type.ToString("g")));

        }

        sb.AppendLine();
        sb.AppendLine("|==============================|");
        sb.AppendLine("|==========Instances==========|");
        sb.AppendLine("|==============================|");
        sb.AppendLine();
        foreach (var instance in Instances)
        {
            sb.AppendLine("INSTANCE {0} of CLASS
{1}".FormatWith(instance.Name, GetClass(instance.ClassId).Name));

            if (instance.Relations != null)
                foreach (var relation in instance.Relations)
                    sb.AppendLine("\t {0} with instance {1}".FormatWith(
                        GetClassRelation(instance.ClassId, relation.RelationId).Name,
                        relation.SlaveInstanceId != null ?
                            Instances.First(x => x.Id == relation.SlaveInstanceId).Name :
                            "UNKNOWN"));

            if (instance.Properties != null)
                foreach (var property in instance.Properties)
                    sb.AppendLine("\t has property {0} of type {1} and value
{2}".FormatWith(
                        GetClassProperty(instance.ClassId, property.PropertyId).Name,
                        GetClassProperty(instance.ClassId,
property.PropertyId).Type.ToString("g"),
                        property.Value));
        }

        return sb.ToString();
    }
```

Таким образом, с помощью данного программного продукта можно создавать любые онтологии. И созданные здесь онтологии представляются в виде объектной модели, то есть, получив описание некоторой предметной области в виде объектной модели, как это мы сейчас сделали для предметной области

“Пиццы”, программист может представить эту предметную область на любом объектно-ориентированном языке программирования.

На рисунке 3.1.1 представлен результат работы программного продукта после создания онтологии предметной области «Пицца».

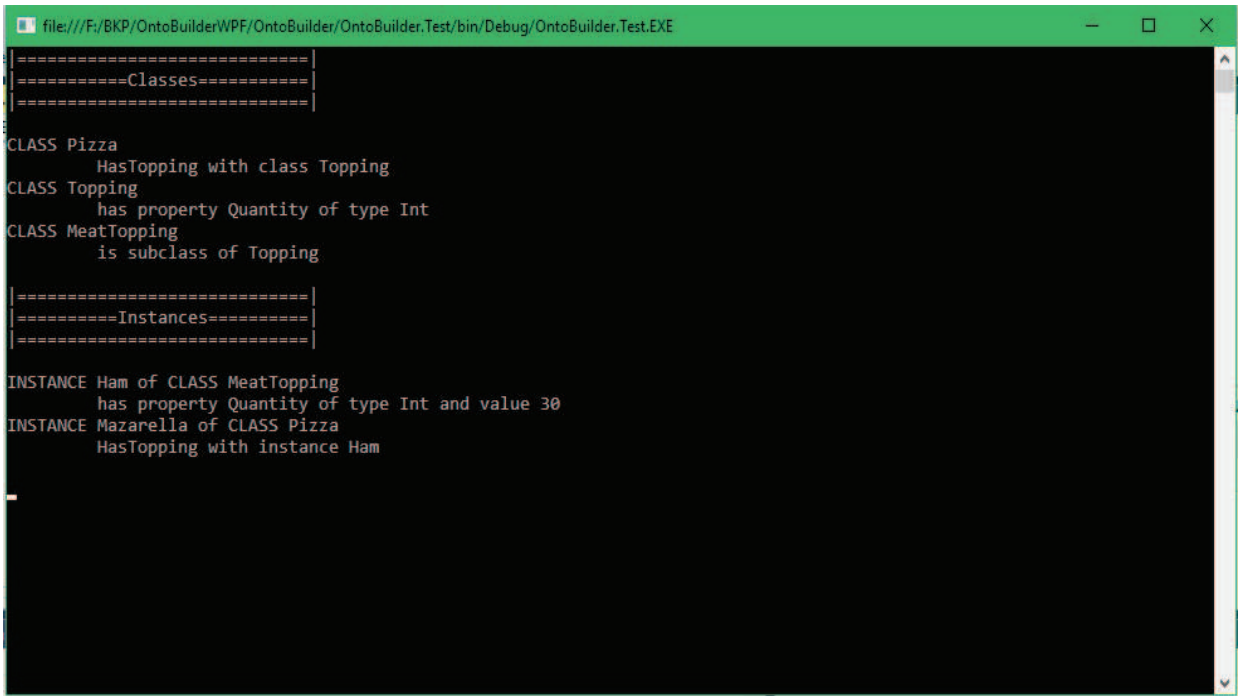

Рисунок 3.1.1 – Результат создания онтологии «Пицца»

Как видно из Рисунка 1, создалась онтология, описывающая предметную область онтологии пиццы. Программный продукт выводит отдельно два блока.

В первом блоке выводится список созданных классов. Также для каждого из них показываются все связи в которых участвует данный класс, указываются свойства этого класса и для дочерних классов показывается, что данный класс является таковым для конкретного класса.

Во втором блоке выводятся экземпляры. В данном блоке указывается наименование экземпляра, к какому классу он принадлежит, какие имеет свойства, какого типа свойства и значение этих свойств.

Также разработанный программный продукт позволяет открывать онтологии, созданные в Protégé в формате OWL с помощью функции конвертации из файлов формата OWL в объектную модель.

3.2 Конвертация онтологии из формата OWL в объектную модель

Для конвертации онтологии из формата OWL в нашу объектную модель воспользуемся онтологией, которая представляет предметную область “Институт”. Данная предметная область представляет собой структуру любого института и описывает взаимодействие групп учащихся, преподавателей, студентов. Описывает в каком университете учится тот или иной студент, на каком факультете, какой кафедре, в какой группе или же в каком филиале института. Также предметная область определяет какое образование получает студент (бакалавриат, аспирантура или магистратура). Выяснив все подробности предметной области, перейдем к составлению онтологии, которая будет описывать эту предметную область в программном продукте Protege. Структура содержит в себе такие классы как группа, преподаватель, студент, учебный план и конкретный университет. В свою очередь класс студентов имеет дочерние классы ученых степеней студента, такие, как аспирант, бакалавр и магистр. Класс университет тоже имеет дочерний класс подразделение университета, в котором присутствуют такие подразделения, как кафедра, факультет и филиал. Полученная иерархия классов, созданная в Protégé представлена на рисунке 3.2.1.

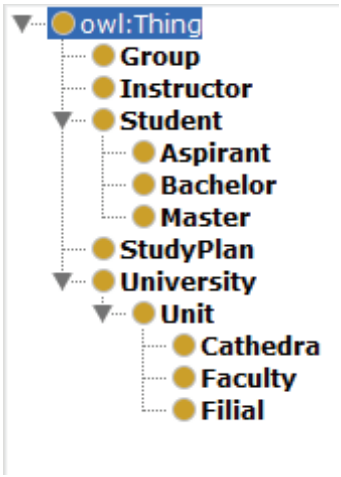

Рисунок 3.2.1 – Иерархия классов предметной области «Институт»

После создания иерархии классов, перейдем к созданию отношений между этими классами.

Первое отношение будет hasStudent, а дочерними отношения данного будут hasAspirant, hasBachelor, hasMaster, которые определяют принадлежность экземпляров из класса студентов к ученым степеням аспиранта, бакалавра и магистра.

Второе отношение hasStudyPlan.

Третье отношение hasUnit определяет принадлежность подразделения.

Четвертое отношение isStudent включает еще три отношения isStudentOfCathedra (является студентом кафедры), isStudentOfFaculty (является студентом факультета), isStudentOfUniversity (является студентом университета), которые в свою очередь подразделяются на ученые степени студента. То есть, например, является аспирантом кафедры, бакалавром факультета, магистром университета и т.д.

Иерархия отношений онтологии представлена на рисунке 3.2.2.

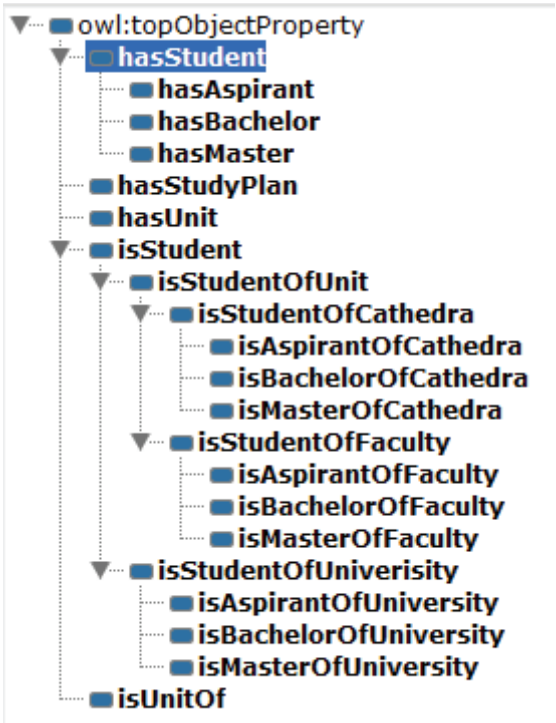

Рисунок 3.2.2 – Иерархия отношений онтологии «Институт»

Предположим, что данная онтология была создана до написания программного продукта, но нам нужно именно по этой предметной области написать программный продукт. Для этого воспользуемся нашим программным продуктом для перевода этой онтологии в объектную модель. Для этого сначала создадим пустую онтологию, а затем в нее запишем результат выполнения ранее описанной функции ParseOWL, в качестве аргумента которой передадим путь до файла формата OWL. Таким образом получим объектную модель онтологии университета. Результат работы конвертера приведен в Приложении А.

На Рисунке 3.2.3 и Рисунке 3.2.4 представлен результат работы конвертации файла формата OWL в разработанную объектную модель.

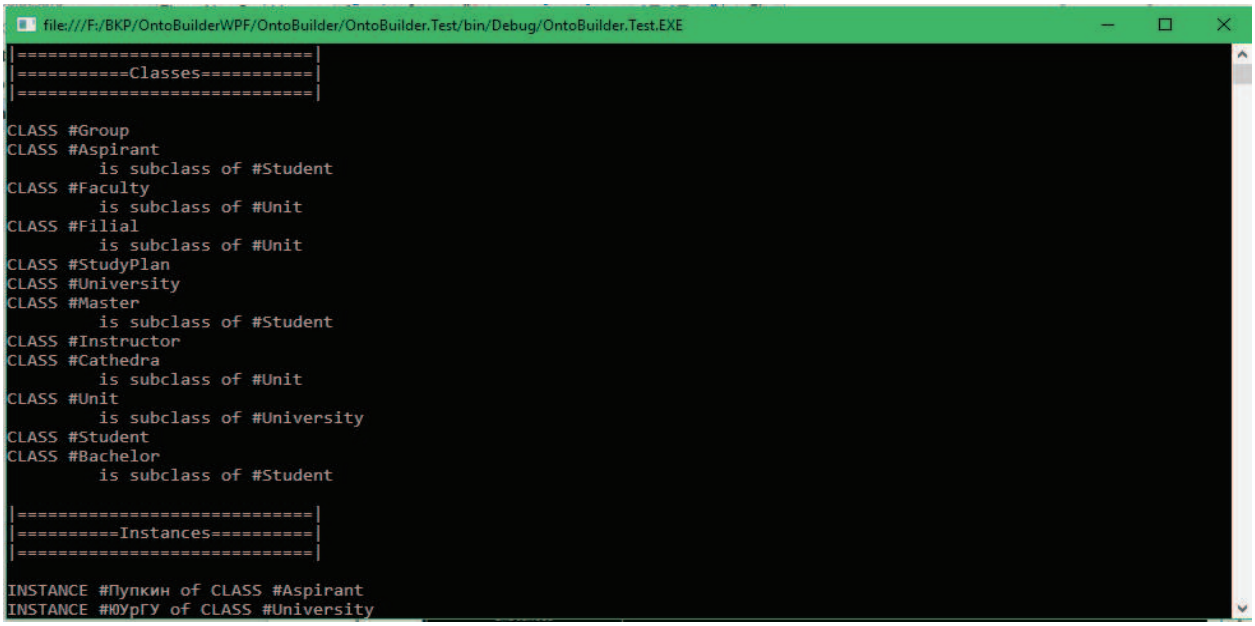

Рисунок 3.2.3 – Результат работы функции конвертации файла OWL в объектную модель программной системы (вывод классов)

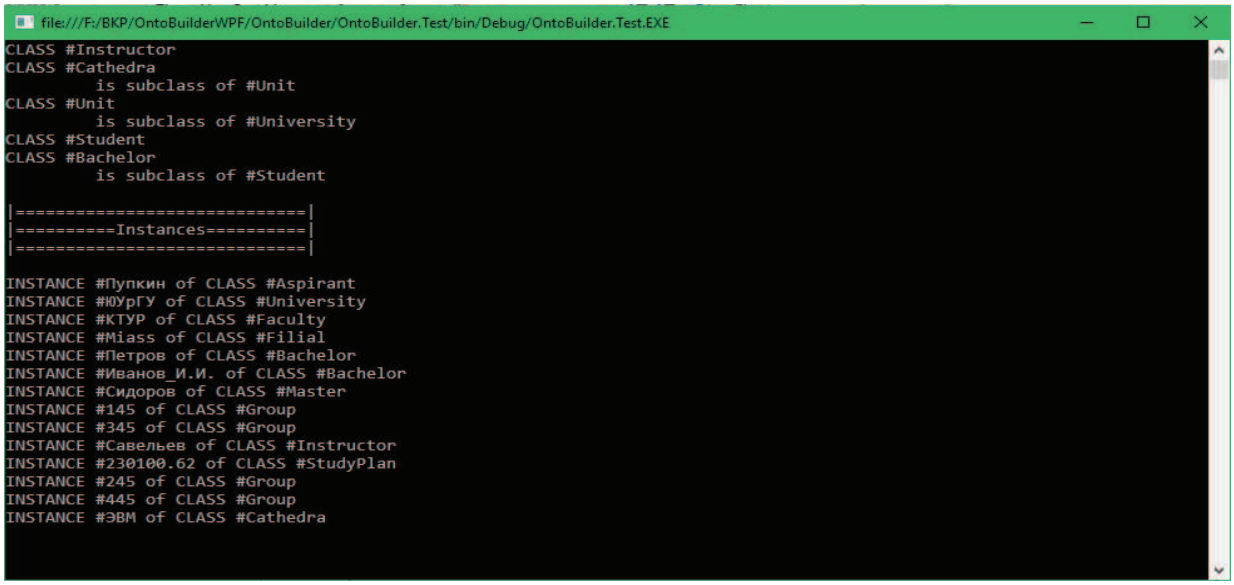

Рисунок 3.2.4 – Результат работы функции конвертации файла OWL в объектную модель программной системы (вывод экземпляров)

Таким образом, функция конвертации работает корректно и программный продукт позволяет выводить результат и конвертации, и созданной онтологии на консоль двумя блоками, отделяя классы и экземпляры этих классов, что улучшает чтение содержимого онтологии.

ЗАКЛЮЧЕНИЕ

Основные результаты выпускной квалификационной работы заключаются в следующем:

1 После получения задачи и уточнения требований к программному продукту, составлена диаграмма использования на языке UML.

2 Проведен анализ существующих аналогов, выявление в них отрицательных качеств, установление актуальности данной работы.

3 Произведен анализ предметной области Онтологий, а именно понятия онтологии, его составных понятий и особенностей описания.

4 На основе анализа предметной области разработана объектная модель онтологии, которая применяется для хранения описания предметной области.

5 Разработана функция конвертации онтологий из формата OWL в объектную модель для повышения универсальности данного программного продукта.

Полученный в итоге программный продукт позволяет четко понимать предметную область, на которую будет опираться тот или иной программный продукт, является удобным инструментом для специалистов, работающих с онтологиями и позволяет программистам создавать качественное программное обеспечение. Таким образом с помощью разработанного программного продукта можно создать объектную модель любой предметной области и представить ее в рамках объектно-ориентированного языка программирования. Помимо этого, продукт позволяет переводить ранее созданные онтологии в разработанную объектную модель.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Добров, Б.В., Иванов, В.В., Лукашевич, Н.В., Соловьев, В.Д. Онтологии и тезаурусы: модели, инструменты, приложения / Б.В. Добров, В.В. Иванов, Н.В. Лукашевич, В.Д. Соловьев – М.: Интернет - Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2009. – 173 с.
2. Бессмертный, И.А. Искусственный интеллект / И.А. Бессмертный – СПб: СПбГУ ИТМО, 2010. – 132 с.
3. Девятков, В.В. Системы искусственного интеллекта: Учебное пособие для вузов / В.В. Девятков – М.: Издательство МГТУ им. Н.Э. Баумана, 2001. – 352 с.
4. Башмаков, А.И., Башмаков, И.А. Интеллектуальные информационные технологии: Учебное пособие / А.И. Башмаков, И.А. Башмаков – М.: Издательство МГТУ им. Н.Э. Баумана, 2005. – 304 с.
5. Гаврилова, Т.А., Хорошевский, В.Ф. Базы знаний интеллектуальных систем / Т.А. Гаврилова, В.Ф. Хорошевский – СПб: Питер, 2000 – 384 с.
6. Загоруйко, Н.Г. и др. Система "Ontogrid" для построения онтологий /Н.Г. Загоруйко //Компьютерная лингвистика и интеллектуальные технологии. Тр. междунар. конференции – Диалог,2005. М., 2005. – С. 146 – 152.
7. Найханова, Л. В. Анализ научного текста и формирование категорийно-понятийного аппарата в виде терминосистемы / Л. В. Найханова // Теоретические и прикладные вопросы современных информационных технологий : материалы Всероссийской научно-технической конференции. – Улан-Удэ : Изд-во ВСГТУ, 2005. – С. 130–139.
8. Найханова, Л. В. Основные аспекты построения онтологий верхнего уровня и предметной области / Л. В. Найханова // Интернет-порталы: содержание и технологии : сборник научных статей. Вып. 3 / [редкол.: А. Н. Тихонов (пред.) и др.] ; ФГУ ГНИИ ИТТ «Информика». – М. : Просвещение, 2005. – С. 452–479.

9. Поспелов, Д. А. Прикладная семиотика и искусственный интеллект / Д. А. Поспелов // Программные продукты и системы. – 1996. – № 3. – С. 10–13.
10. Бабкин, Э.А. Принципы и алгоритмы искусственного интеллекта: Монография / Э.А. Бабкин, О.Р. Козырев, И.В. Куркина. – Н. Новгород: Нижегород. гос. техн. ун-т, 2006. – 132 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

СОДЕРЖАНИЕ ФАЙЛА ФОРМАТА OWL ПРЕДМЕТНОЙ ОБЛАСТИ «УНИВЕРСИТЕТ»

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="EducationDomainOnthology"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     ontologyIRI="EducationDomainOnthology">
    <Prefix name="" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Declaration>
        <Class IRI="#Aspirant"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Bachelor"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Cathedra"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Faculty"/>
    </Declaration>
    <Declaration>
```

```
    <Class IRI="#Filial"/>
</Declaration>
<Declaration>
    <Class IRI="#Group"/>
</Declaration>
<Declaration>
    <Class IRI="#Instructor"/>
</Declaration>
<Declaration>
    <Class IRI="#Master"/>
</Declaration>
<Declaration>
    <Class IRI="#Student"/>
</Declaration>
<Declaration>
    <Class IRI="#StudyPlan"/>
</Declaration>
<Declaration>
    <Class IRI="#Unit"/>
</Declaration>
<Declaration>
    <Class IRI="#University"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasAspirant"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasBachelor"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasMaster"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasStudent"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasStudyPlan"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasUnit"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#isAspirantOfCathedra"/>
</Declaration>
```

```
<Declaration>
   <ObjectProperty IRI="#isAspirantOfFaculty"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isAspirantOfUniversity"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isBachelorOfCathedra"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isBachelorOfFaculty"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isBachelorOfUniversity"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isMasterOfCathedra"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isMasterOfFaculty"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isMasterOfUniversity"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isStudent"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isStudentOfCathedra"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isStudentOfFaculty"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isStudentOfUnit"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isStudentOfUniverisity"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isUnitOf"/>
</Declaration>
<Declaration>
   <DataProperty IRI="#Name"/>
```

```
</Declaration>
<Declaration>
    <NamedIndividual IRI="#P□PIP°PSPsPI_P□.P□."/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#PљPўPJP "/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#PP’Pњ"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#P®PJCЂP“PJ"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#345345345_345"/>
</Declaration>
<SubClassOf>
    <Class IRI="#Aspirant"/>
    <Class IRI="#Student"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Bachelor"/>
    <Class IRI="#Student"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Cathedra"/>
    <Class IRI="#Unit"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Faculty"/>
    <Class IRI="#Unit"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Filial"/>
    <Class IRI="#Unit"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Master"/>
    <Class IRI="#Student"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Unit"/>
    <Class IRI="#University"/>
</SubClassOf>
```

```
<ClassAssertion>
    <Class IRI="#Bachelor"/>
    <NamedIndividual IRI="#P□PIP°PSPsPI_P□.P□."/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Faculty"/>
    <NamedIndividual IRI="#PљPўPJP "/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Cathedra"/>
    <NamedIndividual IRI="#PP’Pњ"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#University"/>
    <NamedIndividual IRI="#P®PJCЂP‘PJ"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Group"/>
    <NamedIndividual IRI="#345345345_345"/>
</ClassAssertion>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#hasAspirant"/>
    <ObjectProperty IRI="#hasStudent"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#hasBachelor"/>
    <ObjectProperty IRI="#hasStudent"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#hasMaster"/>
    <ObjectProperty IRI="#hasStudent"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#isAspirantOfCathedra"/>
    <ObjectProperty IRI="#isStudentOfCathedra"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#isAspirantOfFaculty"/>
    <ObjectProperty IRI="#isStudentOfFaculty"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#isAspirantOfUniversity"/>
    <ObjectProperty IRI="#isStudentOfUniverisity"/>
</SubObjectPropertyOf>
```

```
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isBachelorOfCathedra"/>
   <ObjectProperty IRI="#isStudentOfCathedra"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isBachelorOfFaculty"/>
   <ObjectProperty IRI="#isStudentOfFaculty"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isBachelorOfUniversity"/>
   <ObjectProperty IRI="#isStudentOfUniverisity"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isMasterOfCathedra"/>
   <ObjectProperty IRI="#isStudentOfCathedra"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isMasterOfFaculty"/>
   <ObjectProperty IRI="#isStudentOfFaculty"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isMasterOfUniversity"/>
   <ObjectProperty IRI="#isStudentOfUniverisity"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isStudentOfCathedra"/>
   <ObjectProperty IRI="#isStudentOfUnit"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isStudentOfFaculty"/>
   <ObjectProperty IRI="#isStudentOfUnit"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isStudentOfUnit"/>
   <ObjectProperty IRI="#isStudent"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
   <ObjectProperty IRI="#isStudentOfUniverisity"/>
   <ObjectProperty IRI="#isStudent"/>
</SubObjectPropertyOf>
<InverseObjectProperties>
   <ObjectProperty IRI="#hasStudent"/>
   <ObjectProperty IRI="#isStudent"/>
</InverseObjectProperties>
```

```
<InverseObjectProperties>
   <ObjectProperty IRI="#isUnitOf"/>
   <ObjectProperty IRI="#hasUnit"/>
</InverseObjectProperties>
<TransitiveObjectProperty>
   <ObjectProperty IRI="#isUnitOf"/>
</TransitiveObjectProperty>
<ObjectPropertyDomain>
   <ObjectProperty IRI="#isAspirantOfFaculty"/>
   <Class IRI="#Aspirant"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
   <ObjectProperty IRI="#isAspirantOfUniversity"/>
   <Class IRI="#Aspirant"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
   <ObjectProperty IRI="#isBachelorOfUniversity"/>
   <Class IRI="#Bachelor"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
   <ObjectProperty IRI="#isMasterOfUniversity"/>
   <Class IRI="#Master"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
   <ObjectProperty IRI="#isStudent"/>
   <Class IRI="#Student"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
   <ObjectProperty IRI="#isStudentOfCathedra"/>
   <Class IRI="#Student"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
   <ObjectProperty IRI="#isStudentOfUnit"/>
   <Class IRI="#Student"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
   <ObjectProperty IRI="#isUnitOf"/>
   <Class IRI="#Unit"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
   <ObjectProperty IRI="#hasAspirant"/>
   <Class IRI="#Aspirant"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
```

```
        <ObjectProperty IRI="#hasBachelor"/>
        <Class IRI="#Bachelor"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasMaster"/>
        <Class IRI="#Master"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasStudent"/>
        <Class IRI="#Student"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasUnit"/>
        <Class IRI="#Unit"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#isAspirantOfFaculty"/>
        <Class IRI="#Faculty"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#isStudent"/>
        <Class IRI="#University"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#isStudentOfCathedra"/>
        <Class IRI="#Cathedra"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#isStudentOfUnit"/>
        <Class IRI="#Unit"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#isUnitOf"/>
        <Class IRI="#University"/>
    </ObjectPropertyRange>
</Ontology>


<!-- Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net -->
```