МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра систем информатики

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

МИРОНОВ ВЛАДИМИР ВИКТОРОВИЧ

ВИЗУАЛИЗАЦИЯ И НАВИГАЦИЯ ПО ОНТОЛОГИИ И ГЕНЕРАЦИЯ SPARQL-ЗАПРОСОВ

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Руководитель

Апанович З.В.

к.ф.-м.н., с.н.с. ИСИ СО РАН

Автор

Миронов В.В.

ФИТ, 9201

Новосибирск, 2013 г.

Содержание

Введение....3
ГЛАВА 1 Цель работы и постановка задач....4
ГЛАВА 2 Обзор предметной области....5
2.1 Определение онтологии....5
2.2 Анализ имеющихся решений....6
ГЛАВА 3 Визуализация онтологий....9
3.1 Представление онтологии в виде графа....9
3.2 Методы визуализации деревьев....10
3.3 Интерактивные методы навигации и визуализации....13
3.4 Гиперболическое дерево....15
3.5 Метод иерархического стягивания ребер....18
3.6 Программная реализация....20
ГЛАВА 4 SPARQL-запросы....24
4.1 Язык SPARQL....24
4.2 Визуализация результатов SPARQL-запросов....25
4.3 Генерация SPARQL-запросов....26
4.4 Наполнение наборов данных....27
Заключение....29
Литература....30

Введение

В настоящее время все более популярным становится направление Semantic Web [1], основной целью которого является представление информации в виде, пригодном для машинной обработки. Благодаря этому широкое распространение получило понятие «онтология», под которым подразумевается формализация некоторой области знаний. В эту формализацию входят классы, принадлежащие данной области, а также связи между этими классами.

На данный момент в интернете в свободном доступе находится более 10 000 онтологий в формате OWL [2]. Эти онтологии содержат в себе колоссальный объем полезной, хорошо структурированной информации. Формализация данных делает информацию понятной не только для человека, но и для компьютера. За счет этого значительно уменьшается время и увеличивается качество поиска информации, при этом у компьютера появляется возможность давать ответы на сложные вопросы и создавать новую информацию на основе уже имеющейся.

С каждым днем количество онтологий растет, изменяется их структура, добавляются новые классы и новые данные. Все это затрудняет процесс изучения и анализа онтологий для человека. Появляется необходимость в программных инструментах, облегчающих работу с онтологиями. Одним из таких общепризнанных инструментов является визуализация информации с применением графовых моделей. Такие инструменты позволяют пользователю увидеть структуру и свойства области знаний, описанной с помощью онтологии.

На данный момент уже имеется инструменты [3, 4], предназначенные для визуализации онтологий, но все они рассчитаны на работу с онтологиями малых размеров и непригодны для больших онтологий. Кроме того, большинство из этих инструментов позволяет визуализировать только иерархическую составляющую онтологии, упуская при этом из внимания ассоциативные типы связей между классами. В ходе данной работы планируется реализовать программную систему, лишенную этих недостатков. В работе будут рассмотрены и проанализированы различные алгоритмы визуализации и их свойства, а наиболее подходящие из них будут реализованы в конечной программной системе.

Также планируется реализовать систему генерации SPARQL-запросов [5]. Это позволит пользователю более детально изучить структуру онтологии, а также уменьшит количество рутинной работы по набору запроса вручную. Визуализированная онтология будет своего рода подсказкой пользователю, а отдельные части SPARQL-запросов будут автоматически генерироваться на основе действий пользователя.

ГЛАВА 1 Цель работы и постановка задач

Основной **целью** данной работы является создание программной системы, которая бы облегчала процесс работы с онтологиями для человека. Основными функциональными возможностями этой системы являются:

- Визуализация онтологий с помощью графовых моделей.
- Навигация по информации, представленной в онтологии.
- Манипулирование данными онтологии при помощи SPARQL-запросов.

Ожидается, что созданная в результате программная система позволит пользователю работать с онтологиями больших размеров, а также будет более пригодна для использования, чем уже имеющиеся аналоги.

В рамках поставленной цели был сформулированы следующие **задачи:**

- Обзор предметной области, понятия онтологии и области его применения.
- Анализ требований к разрабатываемой системе.
- Изучение алгоритмов визуализации деревьев и графов.
- Изучение интерактивных методов визуализации и навигации**.**
- Выбор инструментов (язык программирования, среда разработки, наборы библиотек, система контроля версий и т.д.) для реализации.
- Проектирование архитектуры системы.
- Реализация системы с помощью выбранных инструментов.
- Проведение ряда экспериментов с онтологиями различных размеров.
- Анализ полученных результатов.

ГЛАВА 2 Обзор предметной области

2.1 Определение онтологии

Онтология – это попытка всеобъемлющей и детальной формализации некоторой области знаний с помощью концептуальной схемы [6]. Обычно такая схема состоит из структуры данных, содержащей все релевантные классы объектов, их связи и правила (теоремы, ограничения), принятые в этой области.

Современные онтологии строятся по большей части одинаково, независимо от языка описания. Обычно они состоят из экземпляров, понятий, атрибутов и отношений.

- **Экземпляры** или **объекты** – это основные компоненты онтологии. Экземпляры могут представлять собой как физические объекты (люди, дома, планеты), так и абстрактные (числа, слова). Строго говоря, онтология может обойтись и без конкретных объектов. Однако одной из главных целей онтологии является классификация таких объектов, поэтому они также включаются.
- **Понятия** или **классы** – абстрактные группы, коллекции или наборы объектов. Они могут включать в себя экземпляры, другие классы, либо же сочетания и того, и другого. Классы онтологии составляют **таксономию** – **иерархию** понятий по отношению вложения.
- Объекты в онтологии могут иметь **атрибуты**. Каждый атрибут имеет, по крайней мере, имя и значение и используется для хранения информации, которая специфична для объекта и привязана к нему. Значение атрибута может быть сложным типом данных.
- Важная роль атрибутов заключается в том, чтобы определять **отношения** (ассоциативные связи) между объектами онтологии. Обычно отношением является атрибут, значением которого является другой объект.

Более формальное определение онтологии может быть следующим: онтология – это набор множеств $O(C, S, isa)$ [3], где:

1. $C = \{c_1, c_2, \dots, c_m\}$ – множество классов, где каждый класс c_i представляет множество объектов из реального мира.
2. $S = \{s_1, s_2, \dots, s_n\}$ – множество слотов, где каждый слот s_i является либо значением простого типа, такого как Integer, String или Date, либо же показывает связи между классами.
3. $isa = \{isa_1, isa_2, \dots, isa_p\}$ – множество отношений наследования, определенных между классами. Эти отношения имеют семантику подмножеств и образуют частичный порядок на множестве классов, тем самым объединяя классы в одно или несколько деревьев.

Для того чтобы это определение совпадало с ранее данным неформальным определением, множество O может быть дополнено четвертым элементом $I = \{i_1, i_2, \dots, i_q\}$, где каждый i_w – объект некоторого класса $c_x \in C$. Каждый такой экземпляр имеет четко заданное значение для каждого своего слота.

2.2 Анализ имеющихся решений

Одним из наиболее популярных инструментов для работы с онтологиями является платформа Protégé [7], которая позволяет создавать онтологии, добавлять в них новые классы и редактировать связи между ними. В платформу может быть добавлена новая функциональность путем создания дополнений на языке Java. В настоящий момент также имеется ряд дополнений, позволяющих визуализировать онтологии. Об этих дополнениях и пойдет речь в данном разделе.

2.2.1. Браузер классов

Стандартным средством просмотра структуры онтологии в Protégé является браузер классов, который отображает классы с помощью древовидного списка (рисунок 2.2.1). Пользователь может выбрать интересующий его класс, просмотреть список его подклассов или скрыть ненужные классы. Также браузер классов позволяет просмотреть объекты, присутствующие в онтологии. Данный способ позволяет визуализировать только иерархию и не может быть использован для визуализации ассоциативных связей.

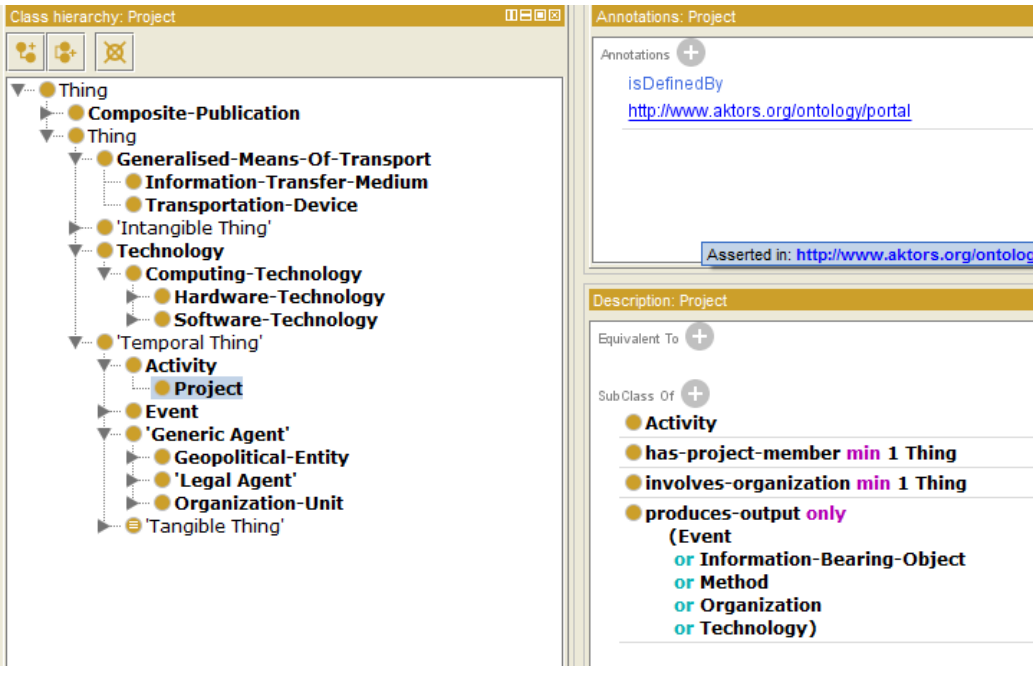

Рисунок 2.2.1. Браузер классов Protégé.

2.2.2. OntoViz

Дополнение OntoViz [8] позволяет визуализировать онтологии с помощью популярного средства визуализации информации Graphviz [9]. Имеется множество настроек визуализации, которые позволяют:

- Визуализировать только отдельное подмножество классов или объектов.
- Отображать ассоциативные связи.
- Изменять настройки цветов.

Данное дополнение хорошо подходит для визуализации небольших онтологий или их частей, но показывает плохие результаты на онтологиях средних и больших размеров.

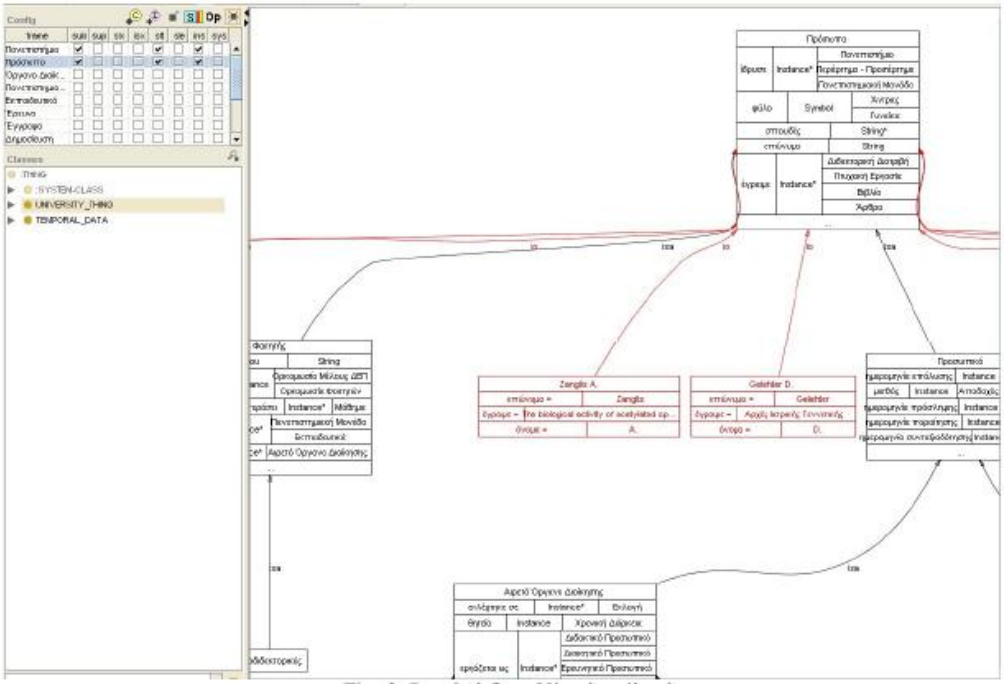

Рисунок 2.2.2 Визуализация с помощью OntoViz.

2.2.3. GlowVis

GlowVis [10] может быть использован для визуализации классов и объектов и предоставляет пользователю на выбор несколько алгоритмов визуализации. В дополнении реализована визуализация ассоциативных связей на основе метода иерархического стягивания ребер, который позволяет эффективно рисовать большое число таких связей. Однако в дополнении отсутствуют средства навигации по результату визуализации, что делает невозможным использование дополнения для работы с большими онтологиями.

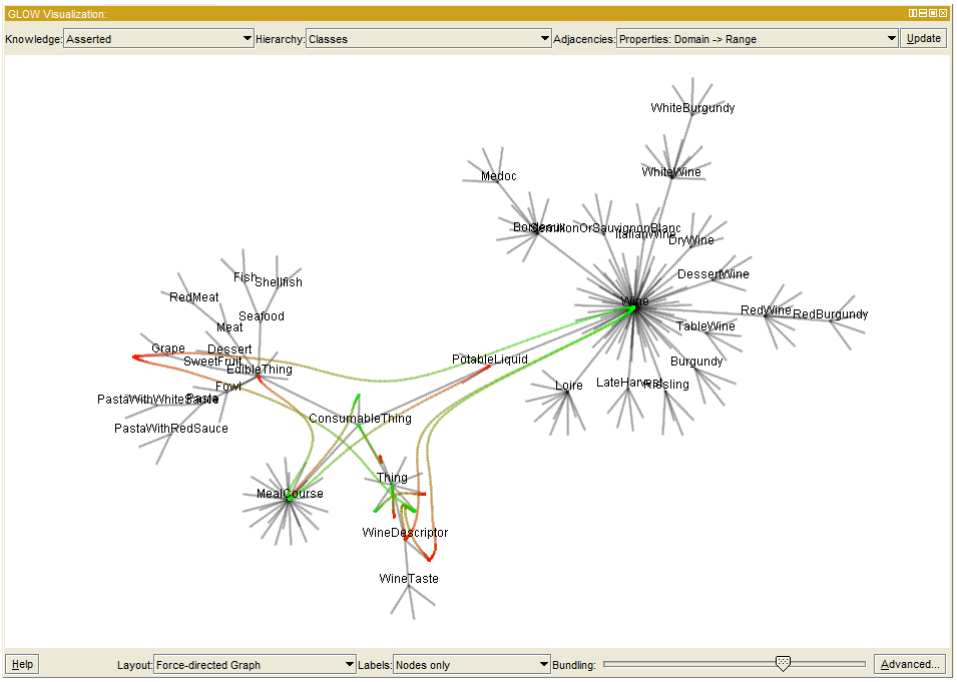

Рисунок 3.2.3. Визуализация с помощью GlowVis.

2.2.4. Другие средства

Перечисленные выше дополнения – далеко не весь набор дополнений, имеющихся для Protégé. Более того, имеется множество независимых приложений, позволяющих визуализировать онтологии как с помощью графовых моделей, так и с помощью других способов визуализации информации. Более подробные обзоры средств визуализации онтологий можно найти в статьях [3, 4]. Однако большинство из этих средств, как правило, имеют хотя бы один из перечисленных ниже недостатков, что значительно затрудняет их использование:

- Осложнена работа с онтологиями больших размеров.
- Нет возможности визуализировать ассоциативные связи.
- Нет возможности эффективно показывать связи *isa* и ассоциативные связи на одном изображении.

ГЛАВА 3 Визуализация онтологий

Визуализация онтологий является непростой задачей, так как помимо иерархии в онтологиях присутствуют ассоциативные связи между классами, которые тоже должны быть визуализированы. Более того, размеры онтологий неуклонно растут и уже сейчас встречаются онтологии, состоящие из нескольких тысяч классов. Каждый класс, как правило, включает в себя экземпляры, количество которых тоже может достигать нескольких тысяч. Далеко не всегда разумно и эффективно показывать классы и экземпляры на одном изображении. И наконец, существуют онтологии, в которых один и тот же класс может иметь нескольких родителей, что существенно усложняет процесс эффективной визуализации таксономии.

В рамках данной работы было решено сосредоточиться на визуализации только классов. При необходимости пользователь может посмотреть список экземпляров, присутствующих в онтологии, с помощью альтернативных методов. Например, в разработанной программной системе имеется специальная панель, на которой можно увидеть все экземпляры, принадлежащие онтологии, а также имеется возможность просмотреть все объекты определенного класса.

Одним из наиболее распространенных и общепринятых способов визуализации информации является визуализация с помощью графовых моделей [11]. В оставшейся части этой главы будет показано, как онтологии могут быть представлены в виде графов и какие алгоритмы и методы могут быть применены для их визуализации.

3.1 Представление онтологии в виде графа

Любая онтология может быть представлена в виде составного графа $G(V, E_I, E_A)$, состоящего из двух частей:

1. Первой частью является дерево $T(V, E_I)$, представляющее иерархическую составляющую онтологии, то есть отношения вида isa.
2. Второй частью является граф $G'(V, E_A)$, представляющий ассоциативные связи между классами онтологии.

Составной граф $G(V, E_I, E_A)$ может быть визуализирован несколькими способами. В самом простом случае наборы ребер E_I и E_A могут быть объединены в единое множество E, тогда граф G может быть визуализирован с помощью любого имеющегося алгоритма визуализации графов. Однако в таком случае типы ребер графа становятся неотличимы друг от друга, что затрудняет процесс понимания получившегося изображения.

Если имеется необходимость явно отделить два типа ребер друг от друга, то можно воспользоваться каким-либо алгоритмом визуализации деревьев и нарисовать

иерархическую составляющую онтологии. Ассоциативные ребра могут быть нарисованы поверх получившегося изображения простыми прямыми линиями. Однако если таких ребер будет слишком много, конечное изображение получится слишком перегруженным и непонятным для пользователя. Примеры таких изображений можно увидеть на рисунке 3.1.1.

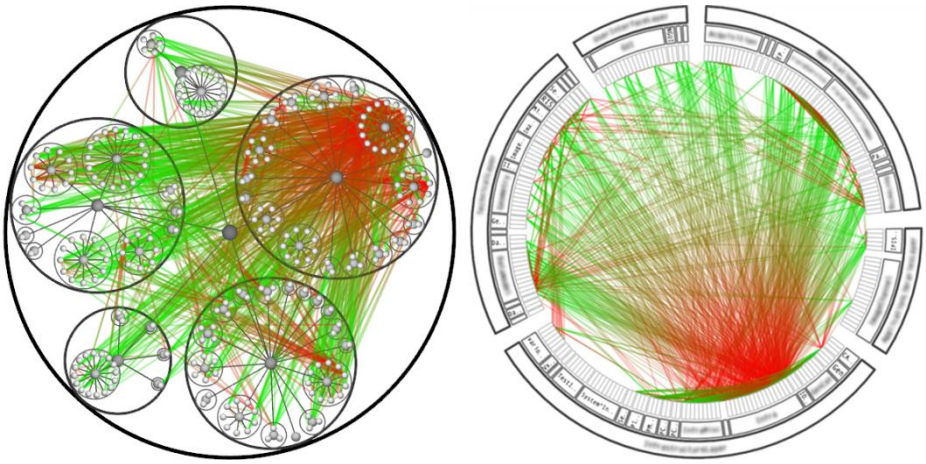

Рисунок 3.1.1. Примеры перегруженных изображений

Так как иерархия является одной из самых важных частей онтологии и предоставляет пользователю ключевую для понимания информацию, то в ходе данной работы было решено отказаться от визуализации онтологий с помощью алгоритмов визуализации графов, а вместо этого сделать явное разделение между иерархическими и неиерархическими ребрами. Для этого были изучены и проанализированы алгоритмы визуализации деревьев, а также были рассмотрены методы визуализации ассоциативных ребер, позволяющие получать понятное изображение, даже если количество этих ребер будет большим.

Следует отметить, что в случае, когда в онтологии присутствует множественное наследование, $T(V, E_I)$ перестает быть деревом и становится графом. Эта проблема может быть решена путем преобразования получившегося графа в дерево. При этом все классы, имеющие нескольких родителей, дублируются несколько раз под каждым из них. Аналогичным подходом пользуются и многие другие средства визуализации [3].

3.2 Методы визуализации деревьев

В данном разделе будут рассмотрены наиболее распространенные алгоритмы визуализации деревьев, их преимущества и недостатки, а так же будут представлены примеры изображений, полученных с помощью этих алгоритмов.

3.2.1 Поуровневый алгоритм

Поуровневое изображение деревьев [11, 12] является наиболее простым и понятным для пользователя и лучше других алгоритмов показывает иерархическую структуру. В данном алгоритме корень дерева располагается в верхней части изображения. Его дочерние вершины располагаются на уровень ниже, их дочерние вершины – на два уровня ниже, и т.д. Связь между дочерними и родительскими вершинами показывается с помощью прямой линии, соединяющей эти вершины. Возможна вариация алгоритма, когда вместо горизонтальных уровней используются вертикальные. Пример работы алгоритма можно увидеть на рисунке 3.2.1

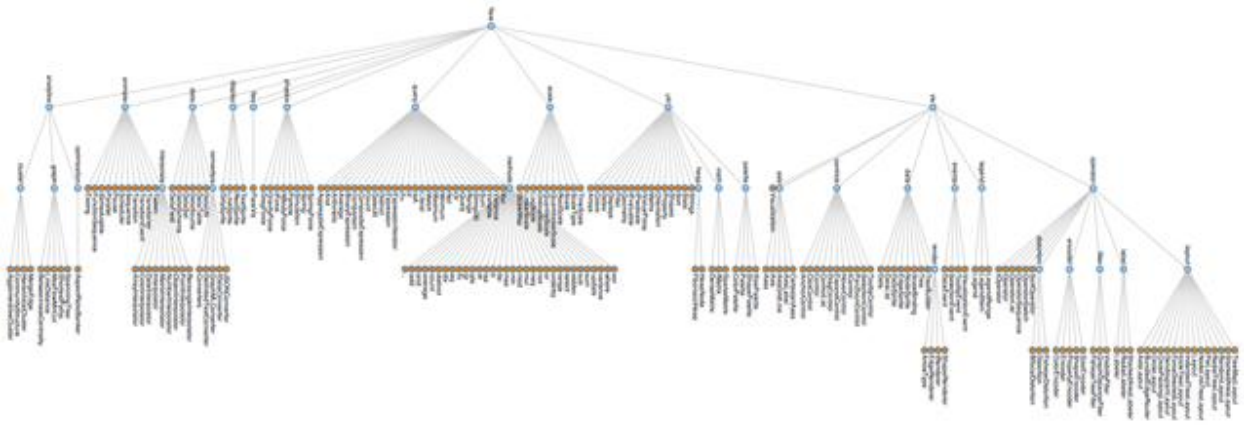

Рисунок 3.2.1. Поуровневое изображение

Основным недостатком поуровневого алгоритма является неоптимальное использование экранного пространства. На каждом новом уровне, как правило, находится большее количество вершин, чем на предыдущем, в результате чего часть экрана, на которой расположен корень дерева, практически пустует, оставшаяся же часть, наоборот, чрезмерно перегружена.

3.2.2 Радиальный алгоритм

Радиальный алгоритм [11, 12] является вариацией поуровневого, однако использует экранное пространство более оптимальным образом. В радиальном изображении все вершины так же разбиваются на уровни, но вместо вертикальных (горизонтальных) уровней используются концентрические окружности.

Простейший радиальный алгоритм работает в 2 прохода и имеет линейную сложность:

1. Для каждой вершины v вычисляется $l(v)$ – количество листьев в поддереве, корнем которого является вершина v.

2. Корень дерева размещается в центре изображения и для его размещения выбирается сектор, угол которого равен 360 градусам. Всем остальным вершинам v выделяется сектор, размер которого пропорционален количеству листьев $l(v)$ в поддереве с корнем в вершине v, по формуле:

$$r(v) = \frac{l(v) \cdot r(parent(v))}{l(parent(v))}$$

На рисунке 3.2.2 можно увидеть изображения, полученные с помощью данного алгоритма.

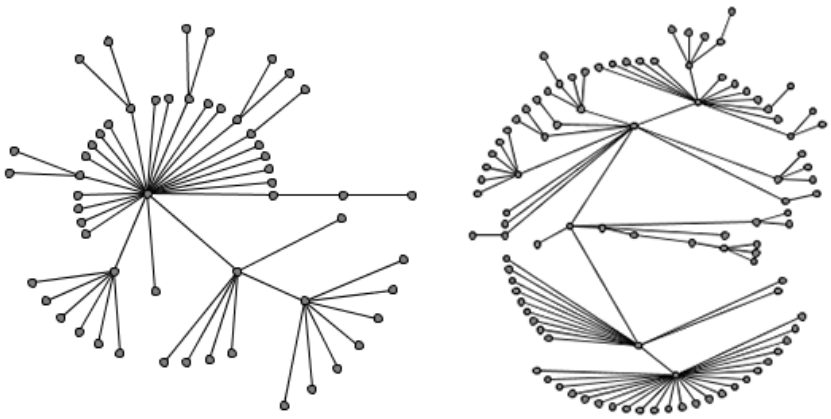

Рисунок 3.2.2. Радиальное изображение

3.2.3 Баллонный алгоритм

Баллонный алгоритм [11, 12], как и радиальный, принадлежит семейству круговых алгоритмов, использующих вложенные окружности для размещения вершин дерева. Данный алгоритм очень похож на радиальный, но имеет некоторые отличия:

- Радиальный алгоритм располагает вершины на окружности вокруг корня дерева, а баллонный алгоритм – вокруг родительской вершины.
- В радиальном алгоритме центры всех окружностей совпадают и находятся в корне дерева; в баллонном алгоритме у каждого поддерева свой центр.

В простейшем случае, когда каждому поддереву выделяются окружности одинакового радиуса, размер сектора может быть вычислен по формуле:

$$\alpha = \frac{2\pi}{num - of - sons}$$

Радиус дочерних окружностей r_{son} может быть выражен через размер сектора α и радиус родительской окружности R_{parent} с помощью формулы:

$$r_{son} = R_{parent} \frac{\sin \alpha}{1 + \sin \alpha}$$

Рисунок 3.2.3 показывает пример работы данного алгоритма.

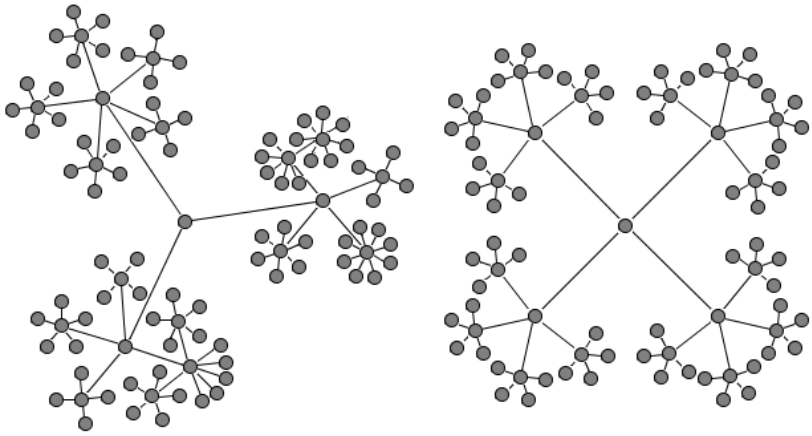

Рисунок 3.2.3. Баллонные изображения

3.2.4 Алгоритм Tree-Map

Алгоритм Tree-Map [11, 12] изображает дерево как набор вложенных прямоугольников. Каждая ветвь дерева представляется в виде прямоугольника, который, в свою очередь, заполнен другими прямоугольниками, представляющими дочерние ветви данной ветви. Площадь наиболее вложенных прямоугольников (т.е. тех, которые представляют листья дерева) пропорциональна некоторой количественной характеристике, ассоциированной с данной вершиной. Часто эти прямоугольники так же окрашиваются в различные цвета, чтобы разделить вершины по какой-либо другой характеристике.

Данный алгоритм является наиболее оптимальным с точки зрения использования экранного пространства. Пример работы алгоритма можно увидеть на рисунке 3.2.4.

Рисунок 3.2.4. Изображение Tree-Map

3.3 Интерактивные методы навигации и визуализации

Рассмотренные выше алгоритмы визуализации широко применяются во многих инструментах по визуализации информации. Несмотря на это, у них у всех имеется один существенный недостаток – в результате их работы получается статическое изображение и это становится проблемой даже для онтологий небольшого размера. На рисунке 3.3.1

можно увидеть пример онтологии, визуализированной с помощью радиального алгоритма:

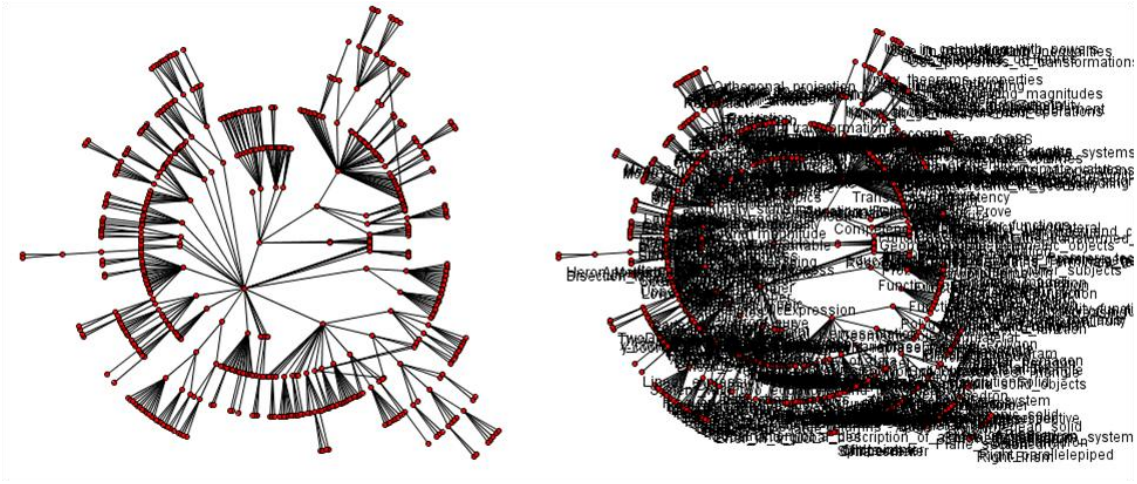

Рисунок 3.3.1. Недостатки статических методов визуализации

На данном рисунке изображена онтология, содержащая в себе 589 классов. На изображении слева присутствуют только вершины и ребра, на изображение справа добавлены текстовые метки для каждой из вершин. Как видно, оба изображения трудны для восприятия и не позволяют пользователю визуально понять интересующую его информацию. При этом на изображениях отсутствуют дополнительные ребра, и их добавление ухудшило бы понимаемость еще сильнее. Статичность изображения ограничивает действия пользователя и не дает возможности детально изучить отдельные части онтологии. Из-за этого появляется необходимость в интерактивных методах навигации и визуализации, которые бы позволяли управлять результатом визуализации в зависимости от нужд пользователя.

Интерактивная визуализация дает пользователю возможность изучать онтологии более простым и понятным способом. Далее будут кратко рассмотрены основные техники, которые могут быть использованы при интерактивной визуализации [13, 14].

3.3.1 Масштабируемая визуализация

Данная техника используется во многих системах визуализации и позволяет пользователю менять масштаб интересующей его области изображения. При этом различают два типа масштабирования – геометрическое и семантическое. Геометрическое масштабирование просто увеличивает (или уменьшает) размеры элементов визуализации, семантическое же добавляет (или убирает) на изображение новые элементы, которые не были видны ранее, на основе некоторой эвристики. Также масштабируемая визуализация позволяет менять положение изображения относительно окна просмотра.

3.3.2 Инкрементальная визуализация

Инкрементальные системы визуализации, вместо того чтобы рисовать большие графы целиком, изображают их в качестве небольших и понятных подграфов. Такие системы рисуют только видимые части большого изображения и позволяют пользователю перемещаться по графу, достраивая новые элементы на ходу. Так как в таких системах нет необходимости рисовать граф целиком, то они часто имеют меньшее время отклика на действия пользователя и меньшую вычислительную сложность, чем другие системы.

3.3.3 Фокус + Контекст

Системы визуализации, основанные на данной технике, создают изображение, в котором интересующая пользователя информация (фокус) изображена во всех деталях, а для оставшейся части информации (контекст) дается лишь поверхностный обзор. Пользователь может взаимодействовать с такими системами путем изменения фокуса, тем самым изменяя и само изображение. Методы группы «Фокус + Контекст» используют различные алгоритмы искажения пространства и их принято разделять на две категории: (1) искажение происходит после того, как получено изображение графа, и (2) искажение встроено в сам алгоритм визуализации. К первой категории можно отнести популярный алгоритм «Рыбий глаз» [15], ко второй – алгоритм «Гиперболическое дерево» [16], о котором и пойдет речь в следующем разделе.

3.4 Гиперболическое дерево

Данный метод специально разрабатывался для навигации по иерархиям больших размеров и позволяет эффективно располагать несколько тысяч элементов на экране небольшого размера. Изначально метод размещает корень дерева в центре и предоставляет пользователю возможность менять фокусную вершину, меняя все изображение в целом.

Вне зависимости от выбранной фокусной точки, каждой вершине предоставляется часть пространства, обратно пропорциональная расстоянию до центра изображения. За счет этого в контекст визуализации всегда входит несколько родительских, дочерних и сестринских вершин, что облегчает пользователю процесс изучения иерархии.

На рисунке 3.4.1 можно увидеть результаты работы алгоритма для одной и той же онтологии, но для разных фокусных точек. Следует отметить, что на этом рисунке изображена та же самая онтология, что и на рисунке 3.3.1.

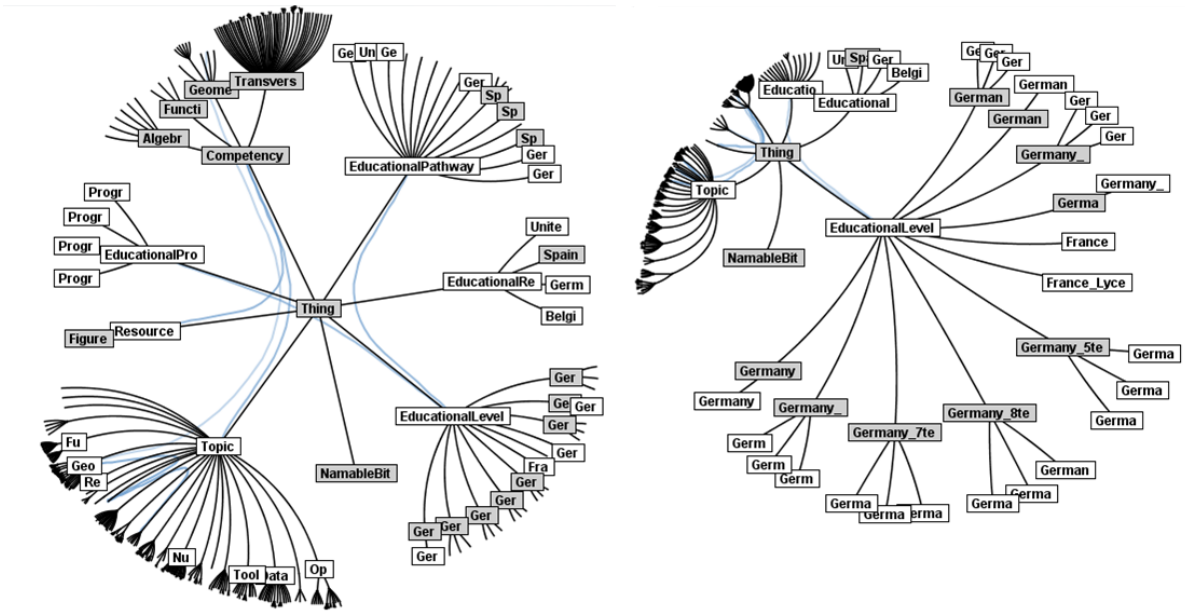

Рисунок 3.4.1. Пример работы алгоритма

3.4.1 Основы

Метод гиперболического дерева, в отличие от других алгоритмов визуализации, размещает вершины не в Евклидовом пространстве, а в гиперболическом. Как только все вершины размещены, они отображаются на единичный круг, который, в свою очередь, отображается на экран монитора. Изменение фокуса обрабатывается путем изменения отображения из гиперболического пространства в Евклидово, за счет этого нет необходимости каждый раз пересчитывать положение вершин на гиперболической плоскости. Процедура отображения вершины из одного в пространства в другое относительно недорогая, что позволяет рисовать на экране большое количество вершин.

3.4.2 Математическая модель

В данном методе каждая точка гиперболического пространства представляется как точка в модели единичного круга Пуанкаре [17]. Точки в единичном круге представляются в виде комплексных чисел, модуль которых меньше единицы.

Преобразование гиперболической плоскости – это функция комплексного переменного, которая отображает единичный круг в самого себя и может быть представлена следующей формулой:

$$z_t = \frac{\theta z + P}{1 + \bar{P}\theta z}$$

, где P и θ – комлексные числа, $|P| < 1$, $|\theta| = 1$ и $\bar{P}$ – комплексно-сопряженное числа P. Данное преобразование сначала поворачивает плоскость на θ вокруг начала координат, а затем переносит начало координат в точку P.

Для любого преобразования можно найти обратное преобразование по формуле:

$$\langle P, \theta \rangle' = \langle -\bar{\theta} P, \bar{\theta} \rangle$$

Суперпозиция двух преобразований $\langle P_1, \theta_1 \rangle$ и $\langle P_2, \theta_2 \rangle$ записывается формулой:

$$\langle \frac{\theta_2 P_1 + P_2}{\theta_2 P_1 \bar{P}_2 + 1}, \frac{\theta_1 \theta_2 + \theta_1 \bar{P}_1 P_2}{\theta_2 P_1 \bar{P}_2 + 1} \rangle$$

3.4.3 Размещение вершин

Компоновка вершин в гиперболическом пространстве происходит рекурсивно и начинается с корня дерева. На каждом шаге рекурсии с вершиной ассоциирована часть пространства в форме сектора бесконечного радиуса, которая задается тремя параметрами – угол α, позиция начала сектора p и точка m, задающая направление. Вершина дерева располагается в начале сектора, затем для всех ее дочерних вершин выделяется подсектор (в простейшем случае исходный сектор просто делится на n равных частей). Расстояние, на котором дочерняя вершина располагается относительно своего родителя, может быть вычислено по формуле:

$$d = \sqrt{\frac{(1 - s^2) \sin \alpha}{2s} + 1} - \frac{(1 - s^2) \sin \alpha}{2s}$$

, где α – угол подсектора, s - желаемое расстояние от вершины до границы подсектора (обычно берется значение равное 0.12).

На следующем шаге нужно посчитать параметры сектора для рекурсивного вызова. Они могут быть получены через параметры подсектора α, p, m и расстояния d по следующим формулам:

$$p' = Trans\,(dm, \langle p, 1 \rangle)$$

$$m' = Trans(Trans(m, \langle p, 1 \rangle), \langle -p', 1 \rangle)$$

$$\alpha' = im\left(\log\left(Trans\left(e^{i\alpha}, \langle -d, 1 \rangle\right)\right)\right)$$

, где $Trans$ – функция, принимающая точку и преобразование и возвращающая результат преобразования.

3.4.4 Визуализация

Процесс визуализации, так же как и процесс компоновки, происходит рекурсивно и использует текущее преобразование гиперболической плоскости на единичный круг. На каждом шаге рекурсии рисуются все ребра, исходящие из текущей вершины, после этого происходят рекурсивные вызовы для дочерних вершин, а затем рисуется непосредственно сама вершина.

Так как прямая гиперболического пространства имеет форму дуги в Евклидовом, ребра дерева изображаются в виде дуг. Координаты центра окружности, содержащей дугу $a - b$ (где a и b – комплексные числа), вычисляются по формулам:

$$d = re(a)im(b) - re(b)im(a)$$

$$c = \frac{i\big(a(1 + |b|^2) - b(1 + |a|^2)\big)}{2d}$$

3.4.5 Изменение фокуса

После компоновки положение вершин в гиперболическом пространстве остается неизменным, а изменение фокуса приводит к изменению текущего преобразования гиперболической плоскости. Изначально используется единичное преобразование $\langle 0, 1\rangle$. Когда пользователь выбирает новую позицию фокуса, вычисляется преобразование, отображающее выбранную точку в начало координат. Аналогичным образом, когда пользователь ведет мышью из одной точки в другую, вычисляется преобразование, которое отображает начальную точку движения в конечную. В обоих случаях новое преобразование может быть вычислено по следующим формулам:

$$a = Trans(s, \langle -p, 1\rangle)$$

$$b = \frac{re\big((e - a)(1 + \overline{ae})\big) + im\big((e - a)(1 - \overline{ae})\big)i}{1 - |ae|^2}$$

$$T = Compose(\langle -p, 1\rangle, \langle b, 1\rangle)$$

, где s – начальная точка, p – конечная точка, s – предыдущее положение фокуса.

3.5 Метод иерархического стягивания ребер

Рассмотренные ранее алгоритмы и методы могут быть применимы только для визуализации таксономии онтологии. Однако помимо иерархии в онтологиях присутствуют ассоциативные связи между классами, и их визуализация является не менее важной задачей. В данном разделе будет рассмотрен метод иерархического стягивания ребер [18], который позволяет эффективно рисовать ассоциативные связи.

Данный алгоритм хорошо адаптирован к типам данных, которые имеют как структурную составляющую (т.е. иерархию, отношение отец-сын), так и неструктурную, показывающую дополнительные типы связей между данными. Идея метода основана на том же принципе, что и кабельные сети. Как только проводов становится слишком много, их объединяют в жгуты, тем самым вся сеть в целом становится более простой и управляемой.

Основными преимуществами данного метода являются:

- Гибкость и возможность работы вместе с уже имеющимися методами визуализации деревьев. За счет этого метод может быть относительно легко интегрирован в уже имеющиеся системы, а у пользователей остается возможность пользоваться их любимым алгоритмом визуализации.
- Метод позволяет получать в результате понятное изображение, даже несмотря на большое количество ребер.
- Пользователь может изменять коэффициент стягивания, тем самым выбирая наиболее понятное изображение.

За основу метода берется изображение, полученное с помощью какого-либо алгоритма визуализации деревьев. Далее, вместо того чтобы рисовать дополнительные ребра простым соединением концов ребер прямой линией, в получившемся изображении дерева ищется путь, проходящий через концы ребра. Найденный путь используется в качестве полигона, на основе которого строится $\beta-$ слайн. Получившийся сплайн и есть изображение исходного ребра. Для построения сплайна используется следующая формула:

$$P_i' = \beta \cdot P_i + (1-\beta)\left(P_0 + \frac{i}{N-1}(P_{N-1} - P_0)\right)$$

, где β - коэффициент стягивания, P_i - набор контрольных точек, N – количество контрольных точек.

Пример построения сплайна можно увидеть на рисунке 3.5.1.

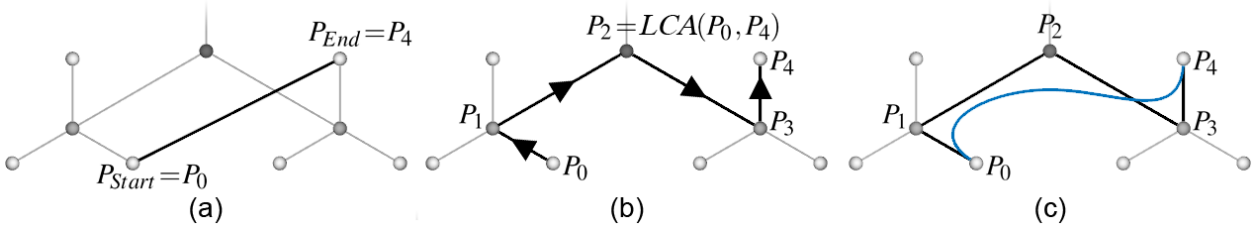

Рисунок 3.5.1. Построение сплайна

Следует отметить, что для различных сплайнов часть контрольных точек будет совпадать, за счет чего отдельные части сплайнов будут очень похожи. Благодаря этому и достигается эффект стягивания. Примеры работы алгоритма при разных коэффициентах стягивания можно увидеть на рисунке 3.5.2

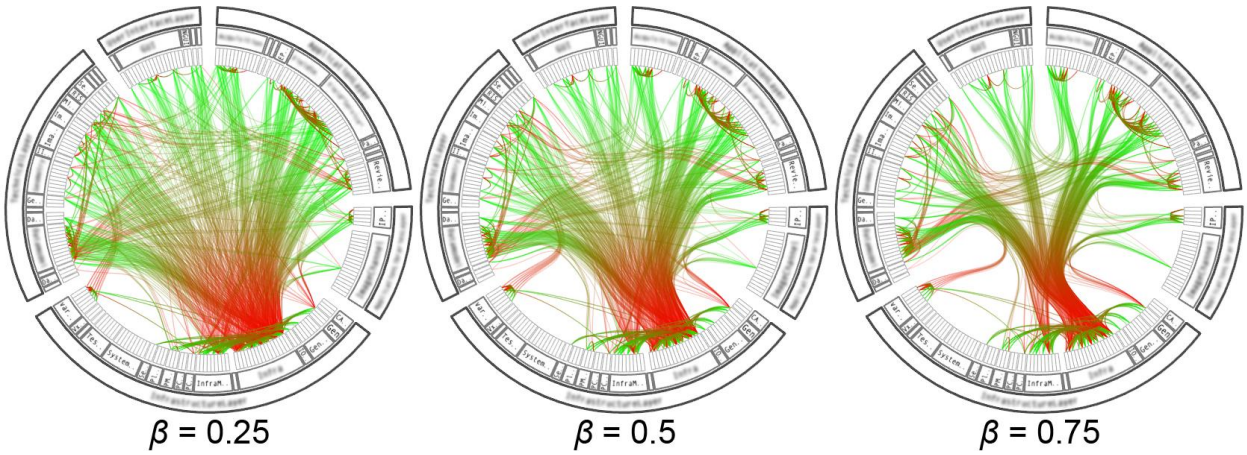

Рисунок 3.5.2. Примеры работы алгоритма при разных коэффициентах стягивания

При визуализации ассоциативных ребер очень важным является порядок, в котором они рисуются. Так как короткие ребра занимают немного экранного пространства, то они, скорее всего, будут перекрыты длинными ребрами. Чтобы решить эту проблему, следует рисовать короткие ребра поверх длинных. Более того, при визуализации короткие и длинные ребра могут быть нарисованы с использованием прозрачности, чтобы еще сильнее подчеркнуть разницу между ними. В таком случае длинные ребра будут более прозрачными, чем короткие. На рисунке 3.5.3 можно увидеть пример этого.

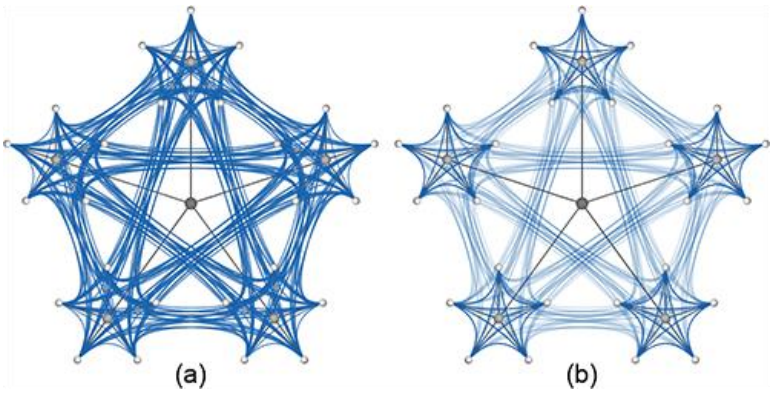

Рисунок 3.5.3. Использование прозрачности для длинных ребер

3.6 Программная реализация

После изучения описанных выше алгоритмов, в разрабатываемой программной системе были реализованы следующие алгоритмы: радиальный, баллонный, гиперболическое дерево и метод стягивания ребер. После этого был проведен ряд экспериментов с онтологиями различных размеров и в результате в системе были оставлены только два алгоритма – гиперболическое дерево и метод иерархического стягивания ребер, так как они, в отличие от других алгоритмов, показывали хорошие результаты на онтологиях больших (более 500 классов) размеров. Примеры изображений

онтологий, полученных с помощью разработанного приложения, представлены на рисунках 3.6.1, 3.6.2 и 3.6.3.

Рисунок 3.6.1. показывает, как метод иерархического стягивания ребер позволяет уменьшить визуальную перегруженность изображения путем объединения ассоциативных ребер в жгуты, тем самым делая визуализацию более эффективной. На изображении слева ассоциативные ребра (окрашены в голубой цвет) изображены простыми прямыми линиями, на изображении справа – объединены в жгуты.

На рисунках 3.6.2 и 3.6.3 можно увидеть результат визуализации онтологий **Akt Reference** [19] и **SUMO** [20]. Для каждой онтологии приведено по два изображения с различными фокусными вершинами.

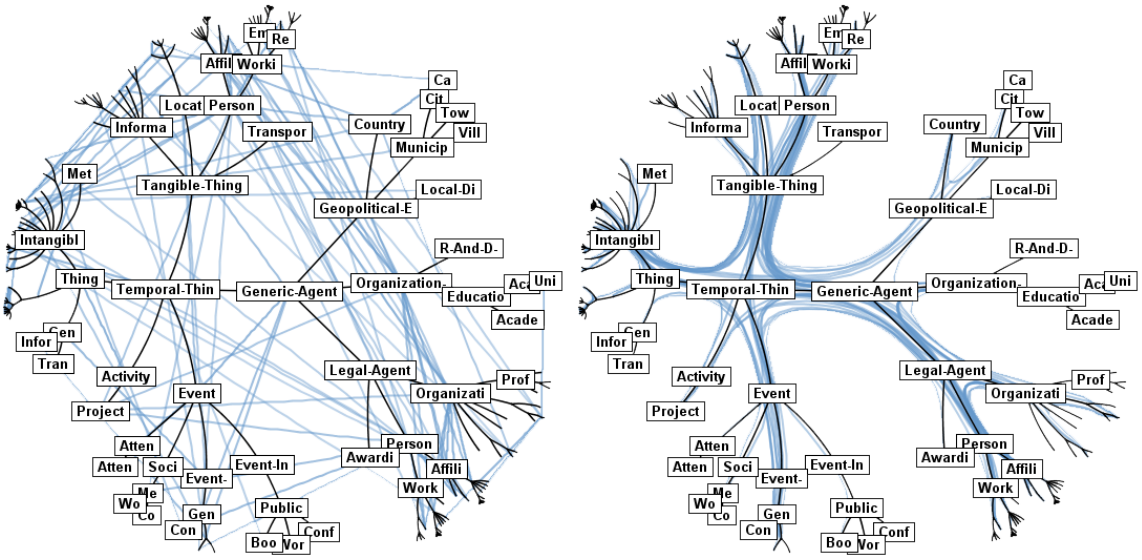

Рисунок 3.6.1. Устранение визуальной перегруженности с помощью метода стягивания ребер

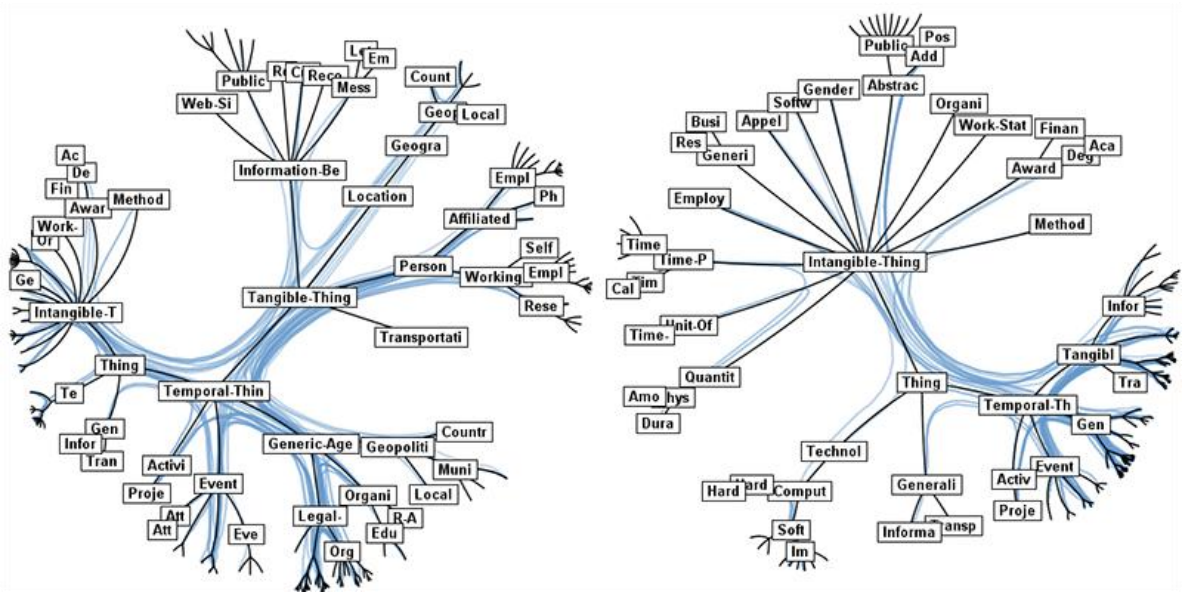

Рисунок 3.6.2. Визуализация онтологии Akt Reference. 250 вершин и 99 ассоциативных ребер

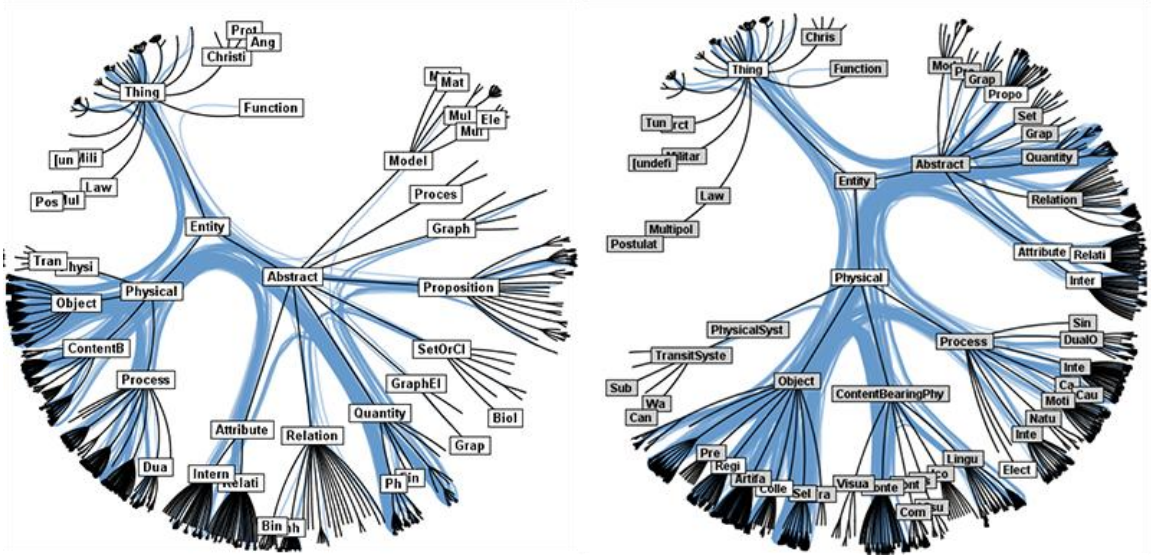

Рисунок 3.6.3. Визуализация SUMO онтологии. ~10000 вершин и ~700 ребер

На рисунке 3.6.4. можно увидеть общий вид окна визуализации, которое состоит из двух панелей – непосредственно панель визуализации справа и панель элементов онтологии слева, в которой можно найти интересующий класс, объект или отношение.

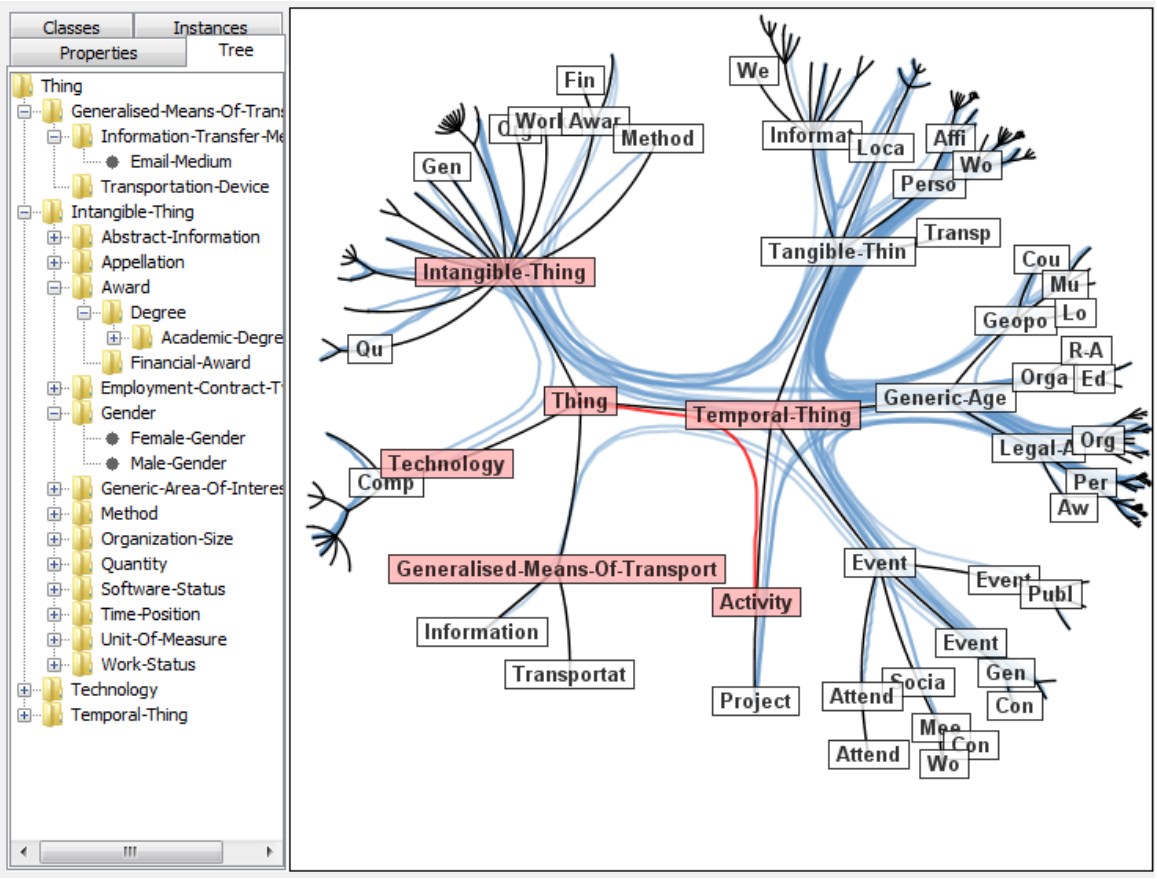

Рисунок 3.6.4. Окно визуализации.

Панель визуализации интерактивна и пользователь может управлять изображением в любое время с помощью мыши, просто изменяя фокусную точку, при этом все переходы изображения из одного состояния в другое происходят в реальном времени и сопровождаются анимацией, что делает эти переходы более наглядными и понятными.

Нажатие на любой элемент, принадлежащий какой-либо из вкладок, перемещает фокус в него. При этом нажатый элемент и все связанные с ним другие элементы отмечаются на визуализации как выделенные. Пример этого можно увидеть на рисунке 3.6.4. На вкладке **Tree** был выбран класс **Thing**, поэтому он и все его дочерние классы (**Intangible-Thing**, **Technology**, **Generalised-Means-Of-Transport**, **Temporal-Thing**) окрасились в розовый цвет. При этом класс **Activity**, который не является потомком **Thing**, тоже помечен розовым. Это происходит, потому что классы **Thing** и **Activity** связаны отношением **uses-resource** (которое помечено красным ребром). Это очень удобно и позволяет быстро найти и увидеть все классы и свойства, связанные с каким-либо классом.

ГЛАВА 4 SPARQL-запросы

Так как онтологии создавались для того, чтобы формально описывать различные данные, немаловажной задачей при их изучении является изучение самих данных. В зависимости от конкретной онтологии и конкретного набора данных, данные могут храниться либо в самой онтологии, либо в отдельном RDF [21] файле, либо в SPARQL точке доступа. Однако, несмотря на реальное местоположение данных, доступ к ним происходит унифицировано с помощью языка SPARQL. В данной главе будут кратко рассмотрены возможности этого языка, а также будет показано, как язык может быть использован при изучении онтологий.

4.1 Язык SPARQL

SPARQL — язык запросов к данным, представленным по модели RDF, а также протокол для передачи этих запросов и ответов на них. Язык довольно мощный и позволяет выполнять сложные операции над данными, среди которых извлечение и поиск информации, фильтрация информации на основе какого-либо критерия, преобразование информации из одного представления в другое.

При использовании языка SPARQL важно понимать, что любой набор данных – это не более чем неупорядоченное множество троек вида «Субъект Предикат Объект», и что все запросы направлены именно на это множество троек. Пример простого запроса, который возвращает имя и адрес электронной почты каждого человека, можно увидеть ниже:

```
select ?name ?email
where {
        ?person a foaf:Person .
        ?person <http://xmlns.com/foaf/0.1/name> ?name .
        ?person <http://xmlns.com/foaf/0.1/mbox> ?email .
}
```

На данный момент SPARQL поддерживает четыре типа запросов:

- **SELECT** – извлекает необработанные значения из точки доступа SPARQL и возвращает результаты в формате таблицы.
- **CONSTRUCT** – извлекает информацию из точки доступа SPARQL в формате RDF и преобразовывает результаты к определенной форме.
- **ASK** – формирует запрос типа Истина/Ложь.
- **DESCRIBE** – Получает описание RDF-ресурса.

4.2 Визуализация результатов SPARQL-запросов

Так как SPARQL-запросы – это мощный инструмент для изучения и анализа различных наборов данных, то в приложение была добавлена возможность ввода и получения результатов таких запросов. Для разных типов запросов результат может быть представлен в разном виде. Запросы типа **SELECT** отображаются в виде таблицы, а запросы типа **CONSTRUCT** – в виде визуализированного графа. На рисунке 4.2.1 показано окно ввода запроса.

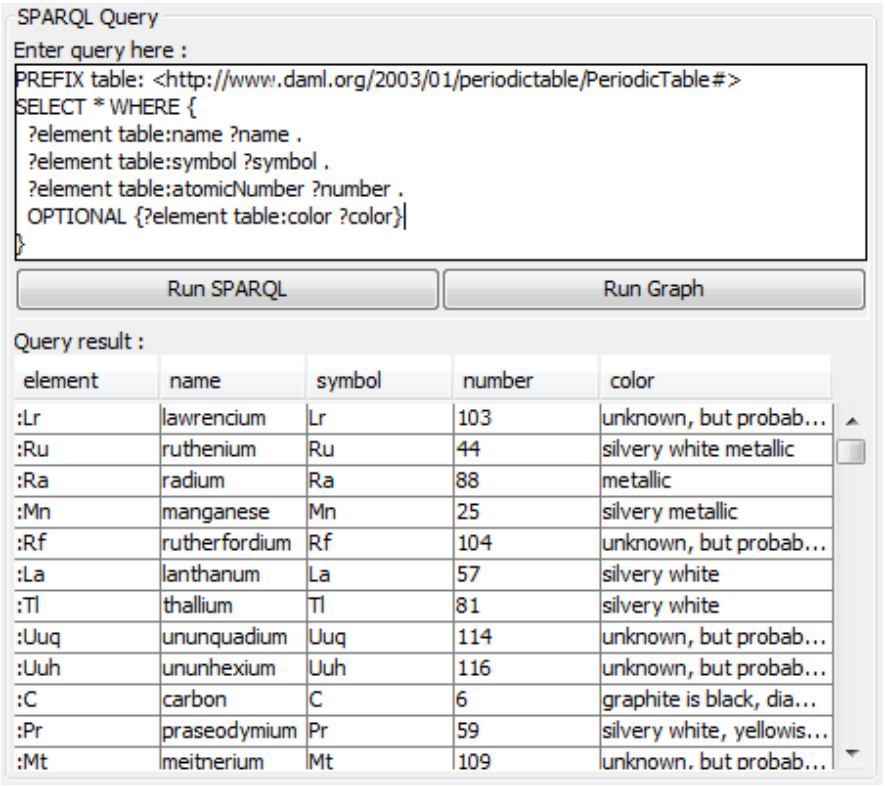

Рисунок 4.2.1. Окно ввода запроса.

На панели ввода в верхней части окна пользователь может ввести текст запроса, затем нажать кнопку **Run SPARQL** или **Run Graph** и увидеть результат. На рисунке можно увидеть простой запрос к онтологии, описывающей периодическую таблицу Менделеева, который возвращает список элементов, их имена, символ, порядковый номер и цвет. Так как используется запрос типа **SELECT**, то результат показывается в табличном представлении.

Результат запросов типа **CONSTRUCT** показывается в виде визуализированного графа, и на рисунке 4.2.2. можно увидеть изображение запроса, который получает первые 200 ссылающихся друг на друга публикаций:

```
prefix akt:  <http://www.aktors.org/ontology/portal#>
construct {
        ?reference akt:cites-publication-reference ?publication
} where {
```

?reference akt:cites-publication-reference ?publication

} limit 200

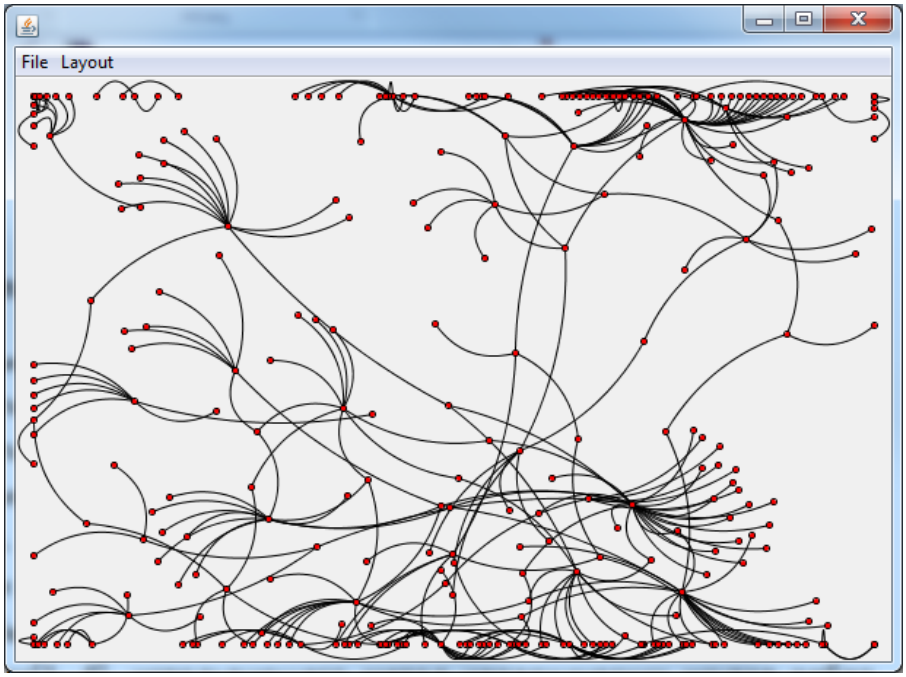

Рисунок 4.2.2. Визуализация SPARQL-запроса.

Для визуализации запросов используется сторонняя библиотека JUNG [22], которая предоставляет несколько уже реализованных алгоритмов визуализации графов, среди которых:

- Алгоритм Фрюхтерман-Рейнгольда [26].
- Алгоритм Камада-Кавая [27].
- Алгоритм Сугиямы [28].

При желании в меню **Layout** пользователь может выбрать любой из перечисленных выше алгоритмов. На рисунке 4.2.2 показан результат работа алгоритма Фрюхтерман-Рейнгольда.

4.3 Генерация SPARQL-запросов

Одним из недостатков языка SPARQL является громоздкость как самих запросов, так и имен классов и отношений, используемых в запросах. Практически в каждом из запросов пользователю нужно вводить по несколько длинных идентификаторов ресурсов (которые состоят из двух частей – пространства имен и непосредственно имени ресурса), что является довольной рутинной задачей. Ниже можно увидеть пример очень простого запроса, который получает информацию об авторах и их публикациях:

*select * where {*

?publication <http://www.aktors.org/ontology/portal#has-author> ?author.

?author <http://www.aktors.org/ontology/portal#full-name> ?name .

?author <http://www.aktors.org/ontology/portal#has-gender> ?gender .

?author <http://www.aktors.org/ontology/portal#has-academic-degree> ?degree .

?author <http://www.aktors.org/ontology/portal#has-appellation> ?appellation .

} limit 100

Уже в таком простом запросе пользователю необходимо пять раз вручную вводить полные названия ресурсов. Частично эта проблема может быть решена добавлением префикса в начало запроса (*prefix akt: <http://www.aktors.org/ontology/portal#>*) и дальнейшим его использование в теле запроса *(?publication akt:has-author ?author*). Однако даже это не спасает от длинных имен ресурсов (*has-academic-degree*).

Чтобы решить эту проблему в приложение была добавлена возможность генерировать отдельные части запроса автоматически. Пользователь может выбрать на окне визуализации интересующий его класс или отношение, нажать кнопку мыши, и название выбранного элемента будет подставлено в тело запроса. Более того, в приложении имеется система автодополнения, которая позволяет пользователю набрать только первые несколько символов названия класса и выбрать нужный ему класс из появившегося выпадающего списка (рисунок 4.3.1)

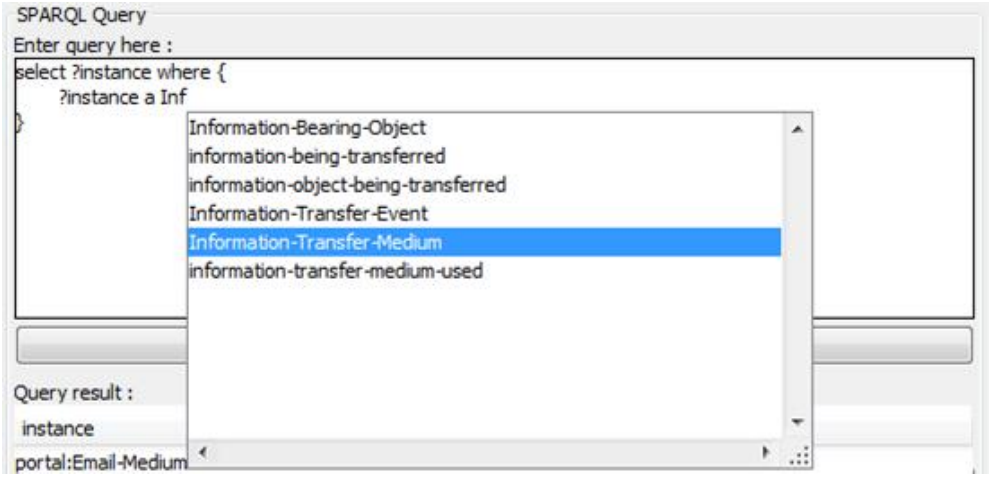

Рисунок 4.3.1. Система автодополнения.

4.4 Наполнение наборов данных

С помощью одной той же онтологии могут быть описаны различные наборы данных, причем далеко не все из этих наборов используют все классы и отношения, присутствующие в онтологии. Так, например, с помощью Akt Reference Ontology описаны следующие наборы данных:

- Digital Bibliographic Library Browser (**DBLP**) [29].
- Association for Computing Machinery (**ACM**) [30].
- France Telecom [31].

- И многие другие [32].

Понимание, какие классы используются в том или ином наборе, а какие – нет, может значительно упростить и ускорить процесс поиска нужной информации. По этой причине в приложении имеется возможность получить список используемых классов в конкретном наборе данных. Более того, можно сравнить два набора и увидеть какие классы используются в обоих наборах, какие – только в первом или только во втором, а какие – и вовсе не используются, при этом на панели визуализации различные классы окрашиваются в разные цвета. Пример сравнения наборов **ACM** и **DBLP** можно увидеть на рисунке 4.4.1 (изображения слева и справа отличаются только фокусной точкой).

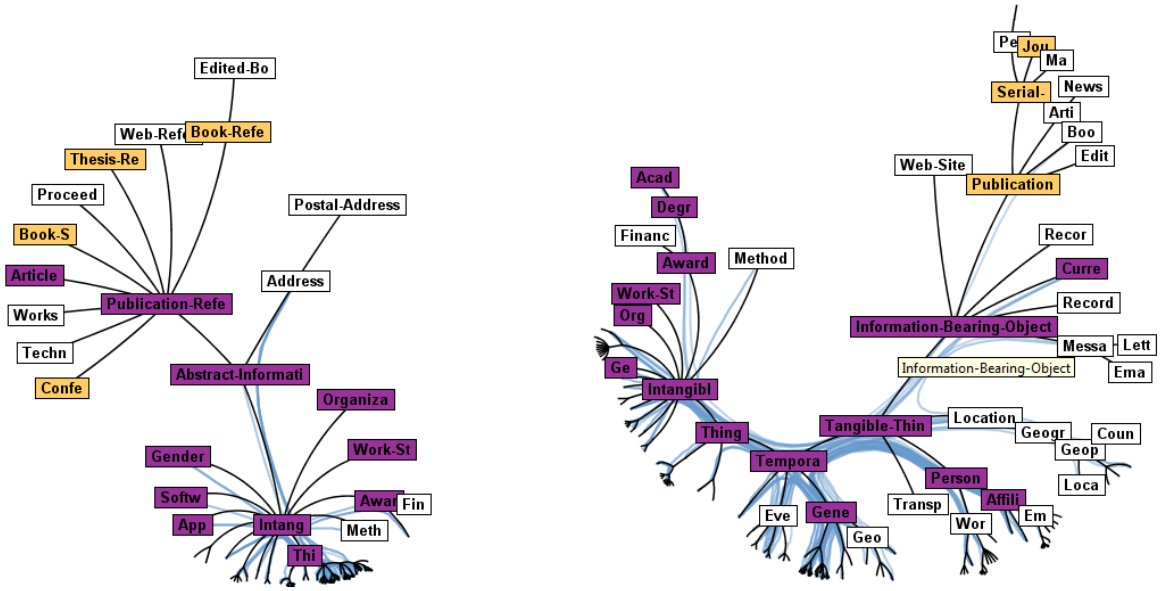

*Рисунок 4.4.1. Сравнение наполнения **ACM** и **DBPL** наборов*

На рисунке фиолетовым цветом помечены классы, используемые как в **ACM**, так и в **DBLP**, желтым – только в **DBLP**, а белым – неиспользуемые классы. Из всех 189 классов онтологии, в **DBLP** используется только 43 из них, в **ACM** – 35, причем все эти 35 классов входят в 43 класса **DBLP**.

В идеальном мире все используемые классы могут быть получены с помощью одного простого SPARQL-запроса *select distinct ?class where { ?instance a ?class }*. Однако в реальных условиях это не работает из-за большого количества данных, в результате чего запрос выполняется очень долго. Например, на SPARQL точке доступа [30] запрос выполняет около 3-х минут и в результате завершается ошибкой (по-видимому, имеется ограничение на максимальное время выполнения запроса). Альтернативным решением является выполнение следующего запроса для каждого из классов *ask {?object a <class-name-here>}*. В таком случае список используемых классов может быть получен за приемлемое время (около 1-ой минуты для AKT онтологии).

Заключение

В результате данной работы было разработано автономное приложение на языке Java, которое позволяет пользователю изучать и исследовать онтологии с помощью интерактивных способов навигации и визуализации информации, а также извлекать эту информацию с помощью SPARQL-запросов.

В рамках проектирования и программной реализации данного приложения были получены следующие результаты:

1. проведен анализ необходимых предметных областей;
2. проанализированы имеющиеся инструменты визуализации онтологий;
3. изучены и проанализированы алгоритмы визуализации графов и деревьев;
4. изучены методы интерактивной визуализации;
5. выбраны алгоритмы, наиболее подходящие для решения поставленных задач;
6. изучены программные библиотеки для работы с онтологиями и SPARQL-запросами;
7. реализован модуль визуализации онтологий с помощью алгоритма «Гиперболическое дерево» и метода иерархического стягивания ребер;
8. реализован модуль генерации SPARQL-запросов на основе результата визуализации;
9. проведен ряд экспериментов с онтологиями различных размеров;

Помимо данных результатов работа была представлена на МНСК-2013 в секции Информационные технологии, подсекция - Наукоемкое программное обеспечение [33].

Литература

1. Semantic Web [Электронный ресурс]. Режим доступа: http://semanticweb.org/wiki/Main_Pag. Дата обращения: 25.05.2013.
2. OWL Specification [Электронный ресурс]. Режим доступа: http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/. Дата обращения: 25.05.2013.
3. A. Katifori, C. Halatsis. Ontology visualization methods – a survey, ACM Computing Surveys (CSUR). Volume 39 Issue 4, 2007.
4. M. Lanzenberger, J. Sampson, M. Rester. Visualization in Ontology Tools, Journal of Universal Computer Science. Volume 16 Issue 7.
5. SPARQL Specification [Электронный ресурс]. Режим доступа: http://www.w3.org/TR/rdf-sparql-query/. Дата обращения: 25.04.2013.
6. Онтология (информатика) [Электронный ресурс]. Режим доступа: http://ru.wikipedia.org/wiki/Онтология_(информатика). Дата обращения: 25.05.2013.
7. Protégé, ontology editor and knowledge-base framework [Электронный ресурс]. Режим доступа: http://protege.stanford.edu/. Дата обращения: 25.05.2013.
8. OntoViz [Электронный ресурс]. Режим доступа: http://protegewiki.stanford.edu/wiki/OntoViz. Дата обращения: 25.05.2013.
9. Graphviz [Электронный ресурс]. Режим доступа: http://www.graphviz.org/. Дата обращения: 25.05.2013.
10. GLOW Visualization [Электронный ресурс]. Режим доступа: http://glowvis.org/Main_Page. Дата обращения: 25.05.2013.
11. Апанович З.В. «Методы и средства визуализации информации на основе графовых моделей» Новосибирск, НГУ, 2009 (Электронный учебник).
12. G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, 1999.
13. Апанович З.В. Методы навигации при визуализации графов// Вестник НГУ, Том 6, выпуск 3.— 2008.— С. 35-47.
14. I. Herman, G. Melancon, and M. S. Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. IEEE Transactions on Visualization and Computer Graphics, 6(1):24–43, 2000.
15. G.W. Furnas. Generalized Fisheye Views // Proceedings of CHI’86. New York: ACM Press, 1986. P. 16–23.

16. J. Lamping and R. Rao. The Hyperbolic Browser: A Focus + Context Technique for Visualizing Large Hierarchies. Journal of Visual Lan-guages and Computing, 7(1):33–55, 1996.
17. Poincare disk model [Электронный ресурс]. Режим доступа: http://en.wikipedia.org/wiki/Poincar%C3%A9_disk_model. Дата обращения: 25.05.2013.
18. D. Holten. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. IEEE Trans. Vis. & Comp. Graphics, 12(5):741-748, 2006.
19. AKT Reference Ontology [Электронный ресурс]. Режим доступа: http://www.aktors.org/ontology/support. Дата обращения: 25.05.2013.
20. The Suggested Upper Merged Ontology [Электронный ресурс]. Режим доступа: http://www.ontologyportal.org/. Дата обращения: 25.05.2013.
21. RDF Specification [Электронный ресурс]. Режим доступа: http://www.w3.org/TR/REC-rdf-syntax/. Дата обращения: 25.05.2013.
22. Java Universal Network/Graph Framework [Электронный ресурс]. Режим доступа: http://jung.sourceforge.net/ . Дата обращения: 25.05.2013
23. Prefuse visualization toolkit [Электронный ресурс]. Режим доступа: http://prefuse.org/. Дата обращения: 25.05.2013
24. Apache Jena [Электронный ресурс]. Режим доступа: http://jena.apache.org/. Дата обращения: 25.05.2013
25. The OWL API [Электронный ресурс]. Режим доступа: http://owlapi.sourceforge.net/. Дата обращения: 25.05.2013.
26. Fruchterman T. M. J., Reingold E. M. Graph Drawing by Force-Directed Placement, Software - Practice and Experience, 1991, Vol. 21, N11, P. 1129-1164.
27. Kamada, T., Kawai, S. An algorithm for drawing general undirected graphs, Information Processing Letters,Vol. 31, 1989, pp. 7-15.
28. Sugiyama K., Tagawa S., Toda M. Methods for visual understanding of hierarchical system structures// IEEE Transactions on Systems, Man, and Cybernetics. 1981. Vol. SMC- 11, N. 2. P. 109-125.
29. DBLP SPARQL endpoint [Электронный ресурс]. Режим доступа: http://dblp.rkbexplorer.com/sparql/. Дата обращения: 25.05.2013.
30. ACM SPARQL endpoint [Электронный ресурс]. Режим доступа: http://acm.rkbexplorer.com/sparql/. Дата обращения: 25.05.2013.
31. French Telecom SPARQL endpoint [Электронный ресурс]. Режим доступа: http://ft.rkbexplorer.com/sparql/. Дата обращения: 25.05.2013.

32. AKT ontology SPARQL endpoints list [Электронный ресурс]. Режим доступа: http://www.rkbexplorer.com/data/. Дата обращения: 25.05.2013.
33. Миронов В.В. Визуализация и навигация по онтологии и генерация SPARQL-запросов // Материалы 51-й международной научной студенческой конференции «Студент и научно-технический прогресс». Информационные технологии/. Новосибирск, 12-18 апреля 2013 г. - 141 с.