|  |  |  |
| --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  РОССИЙСКОЙ ФЕДЕРАЦИИ | | |
| Федеральное государственное бюджетное образовательное  учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Кафедра математического обеспечения  вычислительных систем | |
| **Разработка системы поддержки интервьюирования эксперта для создания и поддержания онтологий**  *Курсовая работа*  *по дисциплине «Математическое и программное обеспечение*  *информационных и интеллектуальных систем»* | | |
|  | | Работу выполнила студентка группы ПМИ–1,2–2020 3 курса механико–математического факультета  Калинина Маргарита Олеговна  «\_\_\_» \_\_\_\_\_\_\_\_2023 г. |
| Научный руководитель:  Ланин Вячеслав Владимирович  «\_\_\_» \_\_\_\_\_\_\_\_2023 г. |
| Пермь 2023 | | |

Оглавление

[ВВЕДЕНИЕ 3](#_Toc133256410)

[ГЛАВА 1. АНАЛИЗ СУЩЕСТВУЮЩИХ МЕТОДОВ И ИНСТРУМЕНТОВ 4](#_Toc133256411)

[1.1. Методы построения онтологий на основе интервьюирования 4](#_Toc133256412)

[1.1.1. Метод MASK 4](#_Toc133256413)

[1.1.2. Методология NeOn 8](#_Toc133256414)

[1.1.3. Методология Tove 9](#_Toc133256415)

[1.2. Обзор существующих программных решений 10](#_Toc133256416)

[1.2.1. Программное средство Protégé 11](#_Toc133256417)

[1.2.2. Программное средство TopBraid Composer 12](#_Toc133256418)

[1.2.3. Программное средство Fluent Editor 13](#_Toc133256419)

[1.3. Требования к разрабатываемому ПО 14](#_Toc133256420)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ ПОДДЕРЖКИ ИНТЕРВЬЮИРОВАНИЯ ЭКСПЕРТА ДЛЯ СОЗДАНИЯ И ПОДДЕРЖАНИЯ ОНТОЛОГИЙ 17](#_Toc133256421)

[2.2. Проектирование взаимодействия компонентов системы поддержки интервьюирования эксперта 17](#_Toc133256422)

[ГЛАВА 3. РЕАЛИЗАЦИЯ ПРОГРАММНОГО РЕШЕНИЯ 24](#_Toc133256423)

[3.1. Реализация форм интервьюирования 24](#_Toc133256424)

[3.3. Реализация дополнительных функций 26](#_Toc133256425)

[3.4. Тестирование программы 27](#_Toc133256426)

[ЗАКЛЮЧЕНИЕ 31](#_Toc133256427)

[Список литературы 32](#_Toc133256428)

ВВЕДЕНИЕ

В современном мире накопление знаний происходит настолько быстро, что существующие системы управления знаниями не всегда могут эффективно справиться с организацией и хранением информации. Одним из подходов к решению этой проблемы является использование онтологий – формализованных представлений знаний о предметной области. Онтологии являются основой для многих современных систем и приложений, таких как системы управления знаниями, интеллектуальные агенты и робототехника.

Создание и поддержка онтологий требуют активного участия экспертов, что делает этот процесс крайне трудоемким и затратным. В связи с этим возникает необходимость в разработке системы поддержки интервьюирования эксперта, которая бы могла автоматизировать и ускорить процесс заполнения онтологии и поддержания её в актуальном виде.

Цель данной курсовой работы – разработка системы поддержки для процесса интервьюирования экспертов.

Задачи работы:

1. Анализ существующих методов и инструментов, используемых для поддержки интервьюирования экспертов и построения онтологий;
2. Анализ требований к системе поддержки интервьюирования экспертов;
3. Проектирование системы;
4. Реализация системы поддержки интервьюирования эксперта;
5. Тестирование разработанной системы поддержки интервьюирования эксперта.

В ходе решения поставленных задач предполагается использовать следующие методы исследования: анализ существующих решений, составление требований к разрабатываемой системе, объектно–ориентированное программирование.

ГЛАВА 1. АНАЛИЗ СУЩЕСТВУЮЩИХ МЕТОДОВ И ИНСТРУМЕНТОВ

В этой главе будут рассмотрены несколько популярных методов построения онтологий на основе интервьюирования. Также будет произведён анализ существующих программных решений.

* 1. Методы построения онтологий на основе интервьюирования

Существует многообразие методов построения онтологий, оно объясняется тем что разные методы соответствуют разным целям и задачам. Методы построения онтологий на основе интервьюирования эксперта позволяют получить качественные и точные данные от экспертов, которые имеют высокую квалификацию в предметной области, для которой создается онтология. Эксперты могут предоставить ценную информацию о терминах, понятиях, связях и принципах, которые используются в данной области.

* + 1. Метод MASK

MASK [5] [7] (Method of Analysis and Structuring Knowledge) – эволюция метода MKSM. Метод MASK предлагает гибкую среду, которая позволяет успешно реализовывать проекты по капитализации знаний. Он был применен в большом количестве областей (безопасность, бизнес–процессы, механическое проектирование), которые позволили ему эволюционировать.

MASK рекомендует совместное построение модели. Инженер знаний представляет модель и приглашает эксперта заполнить ее. Эксперт может предложить изменить некоторые элементы модели и так далее. Также в MASK не рекомендуется последовательный порядок построения модели. Каждая область приложения имеет свой метод формализации.

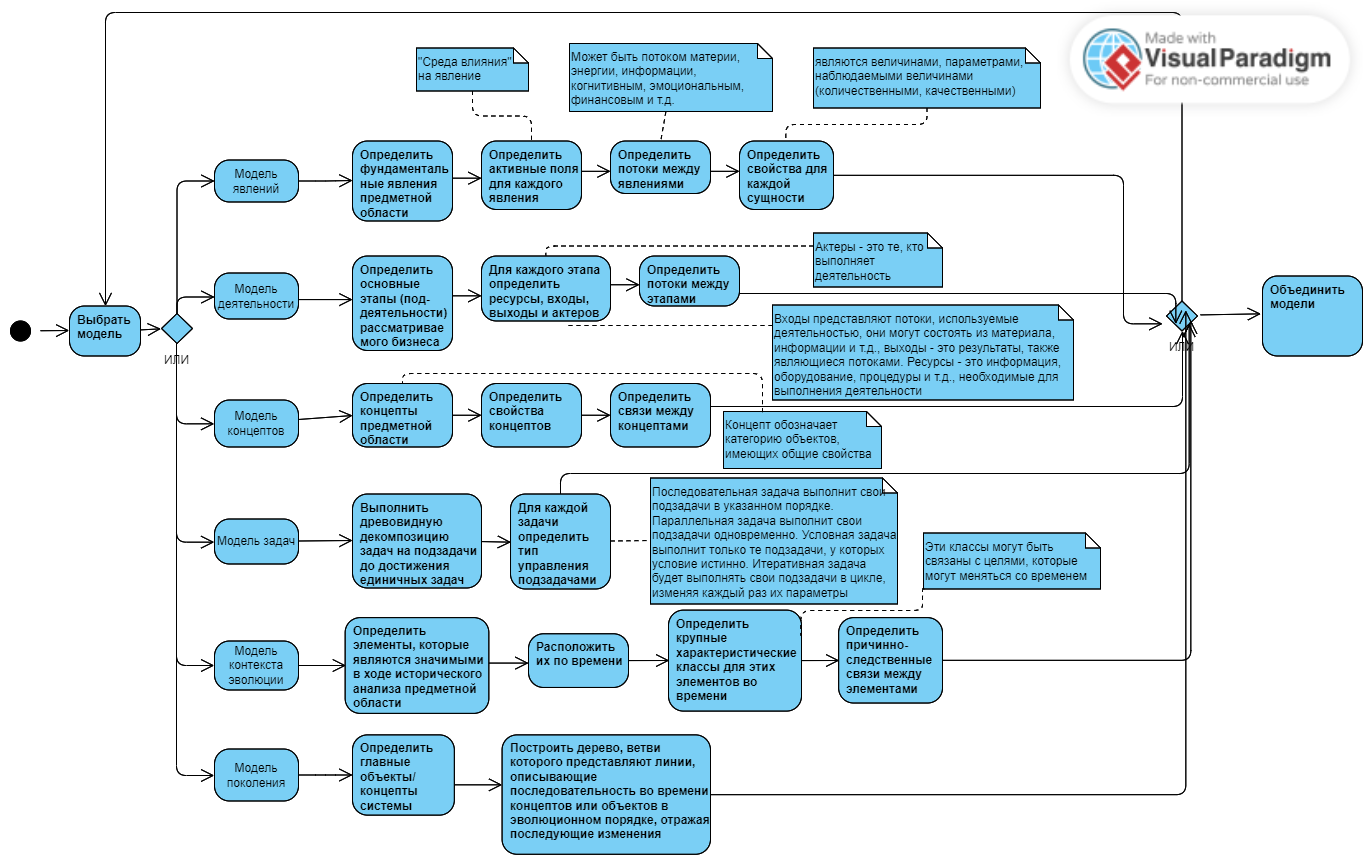
Инструменты моделирования:

* Системный анализ. Он отвечает на вопрос “О чём это знание?”. В него входит две модели – модель явлений и модель активности. Модель явлений – это концептуальная модель, которая описывает фундаментальные явления, происходящие в предметной области. Модель явлений используется для анализа и объяснения различных явлений в предметной области, а также для принятия решений на основе этого анализа. В модели явлений используются две основные двойственности: поток/поле и источник/цель. Поток – это то, что происходит в активном процессе, а поле – это “среда влияния” на процесс. Источник и цель – это две подсистемы, которые связывает поток и которые также характеризуют процесс.

Вторая модель – модель деятельности. Модель деятельности – это декомпозиция основных этапов (под–деятельностей) рассматриваемой системы, связанных потоками. Целью этой модели является не детальное описание "как это делается", а идентификация актеров, входов, выходов и ресурсов процесса. Пределом разбиения деятельности будет "преобразование входного и выходного материала". Это разложение должно показывать от трех до шести элементов максимум.

* «Эргокогнитивный» (терминология введена разработчиками метода MASK) анализ. Вопросы, на которые хочет ответить анализ, таковы: "Какого рода задачи мы должны решать?", "Как обычно следует решать такого рода задачи?", "Как мы должны использовать эти статические знания для решения нашей проблемы?» Метод MASK, используемый для создания модели задач, дает описание расписания задач с использованием иерархической рекурсивной декомпозиции высокоуровневой задачи на более низкоуровневые подзадачи. Каждая неконечная задача имеет тип, который описывает способ управления ее подзадачами. Последовательная задача выполнит свои подзадачи в указанном порядке. Параллельная задача выполнит свои подзадачи одновременно. Условная задача выполнит только те подзадачи, у которых условие истинно. Итеративная задача будет выполнять свои подзадачи в цикле, изменяя каждый раз их параметры. Дерево, которое создает эта модель, называется контрольным потоком. Этот контрольный поток характеризует стратегию решения проблем и может быть графически представлен.
* «Психокогнитивный» (терминология введена разработчиками метода MASK) анализ. Данный анализ решает проблему “статического аспекта знаний”, то есть связывает знание с контекстом и ситуацией использования. Модель концептов в методе MASK представляет собой иерархию понятий, описывающих объекты или явления в предметной области. Каждый концепт описывает некоторый абстрактный объект и включает в себя такие атрибуты, как название, определение, свойства и отношения с другими концептами.
* Эволюционный анализ. Эволюционный анализ MASK онтологии заключается в описании и анализе эволюции объектов и технологий в рамках конкретной предметной области с течением времени и опыта. Для этого применяются две модели: модель контекста эволюции, которая описывает эволюцию объектов и технологий в историческом контексте, и модель поколения, которая описывает последовательность концепций и объектов, соответствующих прогрессу техник и их усовершенствованию. Таким образом, эволюционный анализ MASK онтологии позволяет понимать, как изменялись и развивались объекты и технологии в предметной области, и как эти изменения могут быть отражены в структуре онтологии, чтобы она была более точной и актуальной.

По методу MASK была разработана диаграмма активности в нотации UML (см. рис. 1). Из преимуществ данного метода можно отметить гибкость, так как MASK позволяет моделировать онтологии для широкого спектра приложений и дисциплин, таких как биология, медицина, инженерия и т.д. и масштабируемость, так как MASK может использоваться для моделирования онтологий любой сложности, от простых схем до больших онтологических моделей.



*Рисунок 1. Диаграмма метода MASK*

* + 1. Методология NeOn

Особенностью методологии NeOn [8] является её набор сценариев, которые предназначены для построения онтологий и их сетей. Этот набор состоит из девяти элементов, которые описывают различные подходы к созданию онтологий, начиная с создания новой онтологии и заканчивая локализацией уже существующей. Каждый сценарий предлагает определенный подход к созданию онтологий, включая повторное использование уже существующих ресурсов, реинжиниринг, объединение, слияние и другие техники. Эти сценарии могут быть комбинированы по–разному в зависимости от целей разработки онтологии.

* + *Сценарий 1: От спецификации к реализации*. В данном сценарии онтология разрабатывается с нуля. Сначала разработчики указывают требования к онтологии, затем проводят поиск ресурсов с потенциалом для повторного использования и в конце составляют план, которому они должны следовать.
  + *Сценарий 2: Повторное использование и реинжиниринг неонтологических ресурсов (NOR)*. Сначала разработчики повторно используют NOR, решая какие из них можно повторно использовать для онтологии, основываясь на её требованиях, а затем преобразовывают выбранные ресурсы в онтологии.
  + *Сценарий 3: Повторное использование онтологических ресурсов.* В данном сценарии при разработке онтологии используются онтологические ресурсы: целые онтологии, их модули и/или утверждения.
  + *Сценарий 4: Повторное использование и реинжиниринг онтологических ресурсов.* В разработке повторно используются и, при необходимости, реконструируются онтологические ресурсы.
  + *Сценарий 5: Повторное использование и объединение онтологических ресурсов*. В рамках этого сценария онтология создаётся путём объединения нескольких онтологических ресурсов в одном домене.
  + *Сценарий 6. Повторное использование, слияние и реинжиниринг онтологических ресурсов*. Данный сценарий аналогичен предыдущему за тем исключением, что набор объединённых ресурсов проектируется повторно.
  + *Сценарий 7. Повторное использование шаблонов проектирования онтологий (ODP)*. При создании онтологии разработчики обращаются к репозиториям с ODP для их повторного использования.
  + *Сценарий 8: Реструктуризация онтологических ресурсов*. При этом сценарии разработчики модулируют, сокращают, расширяют и специализируют онтологические ресурсы с целью их дальнейшей интеграции в разрабатываемую онтологию.
  + *Сценарий 9: Локализация онтологических ресурсов.* В рамках этого сценария происходит адаптация онтологии к другим языкам и культурным сообществам в целях получения многоязычной онтологии.

Основным преимуществом NeOn является её понятность и прозрачность для пользователей, благодаря чёткому определению целей, входных и выходных данных, а также набору техник и инструментов для каждого процесса или действия.

* + 1. Методология Tove

Методология TOVE [3] – это базовый набор принципов для создания онтологий, включающий шесть этапов: фиксацию мотивационных сценариев, формулировку критериев проверки компетенции, формальную спецификацию онтологии, формулировку критериев оценки компетенции, спецификацию функций интерпретации для терминов онтологии и задание условий наполненности онтологии.

* *Фиксация мотивационных сценариев*. Под сценариями в данном случаем подразумеваются принципы конкретной предметной области, специфицирующие множество возможных решений для задачи онтологии.
* *Формулировка критериев проверки компетенции*. Данные критерии основываются на описанных в предыдущем пункте сценариях и выступают в качестве требований к выразительности и связности онтологий.
* *Формальная спецификация онтологии*. Данный этап состоит из проектирования неформальной терминологии – выделения множества базисных терминов для создания онтологии из множества сформулированных вопросов проверки компетенции и спецификации спроектированной терминологии на каком–либо формальном языке (KIF, LOOM и т. д.).
* *Формулировка критериев оценки компетенции с использованием специфицированных терминов*. Данный этап посвящён спецификации запросов на том же формальном языке.
* *Спецификация функций интерпретации для терминов онтологии*. На данном этапе определяется семантика внутренней терминологии онтологии, а также ограничения на их интерпретацию.
* *Задание условий наполненности онтологии*. Последний этап методологии, посвящённый определению условий, обеспечивающих полноту решения задач онтологии.

Основными преимуществами методологии TOVE являются ее простота и минимальный набор принципов, необходимых для построения онтологий, недостатком является отсутствие указаний на повторное использование знаний.

* 1. Обзор существующих программных решений

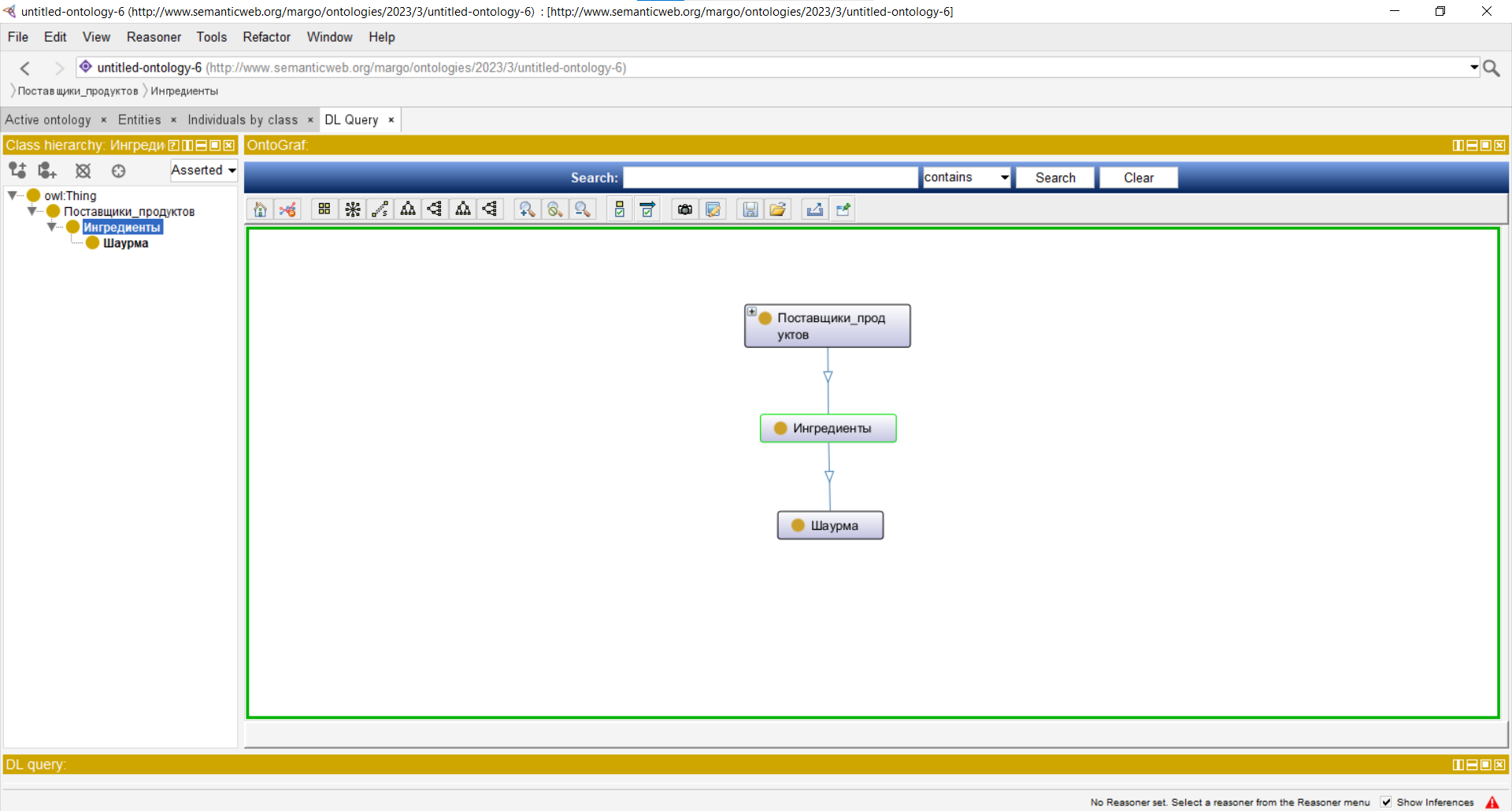
В области разработки систем для создания онтологий было сделано много работы. Одни из первых систем были Ontolingua и Prompt, которые разработаны в 1990–х годах. Затем появилось больше систем, таких как Protege, которая предоставляет широкие возможности для разработки и редактирования онтологий, а также плагины для интервьюирования экспертов и автоматической генерации онтологий. Существуют также другие системы, например, такие как WebODE, OntoMat, OntoBuilder, OntoCreate, TopBraid Composer.

Однако, несмотря на все достижения в этой области, все ещё существуют некоторые недостатки и ограничения. Например, большинство существующих систем склонны к сложности использования для экспертов, не обладающих достаточными знаниями в области онтологического инжиниринга. Кроме того, многие из этих систем требуют длительного времени для обучения и настройки, а также значительных усилий для создания и сопровождения онтологии. Также существует проблема ограниченной поддержки различных языков и отсутствия единого стандарта для описания онтологий. Рассмотрим наиболее популярные программные решения подробнее.

* + 1. Программное средство Protégé

Программное средство Protégé [2] [6], разработанное Стэндфордским центром исследований в области биомедицинской информатики на языке программирования Java, является наиболее известным инструментом для работы с онтологиями. Его главной задачей является создание понятных моделей предметной области и их использование в программном коде. Программа обладает широким функционалом, который позволяет не только создавать, но и объединять несколько онтологий в одну структуру и поддерживает новейшие спецификации OWL и RDF. Существует множество плагинов для Protégé, которые позволяют создавать полноценные приложения на основе разработанных или импортированных онтологий.

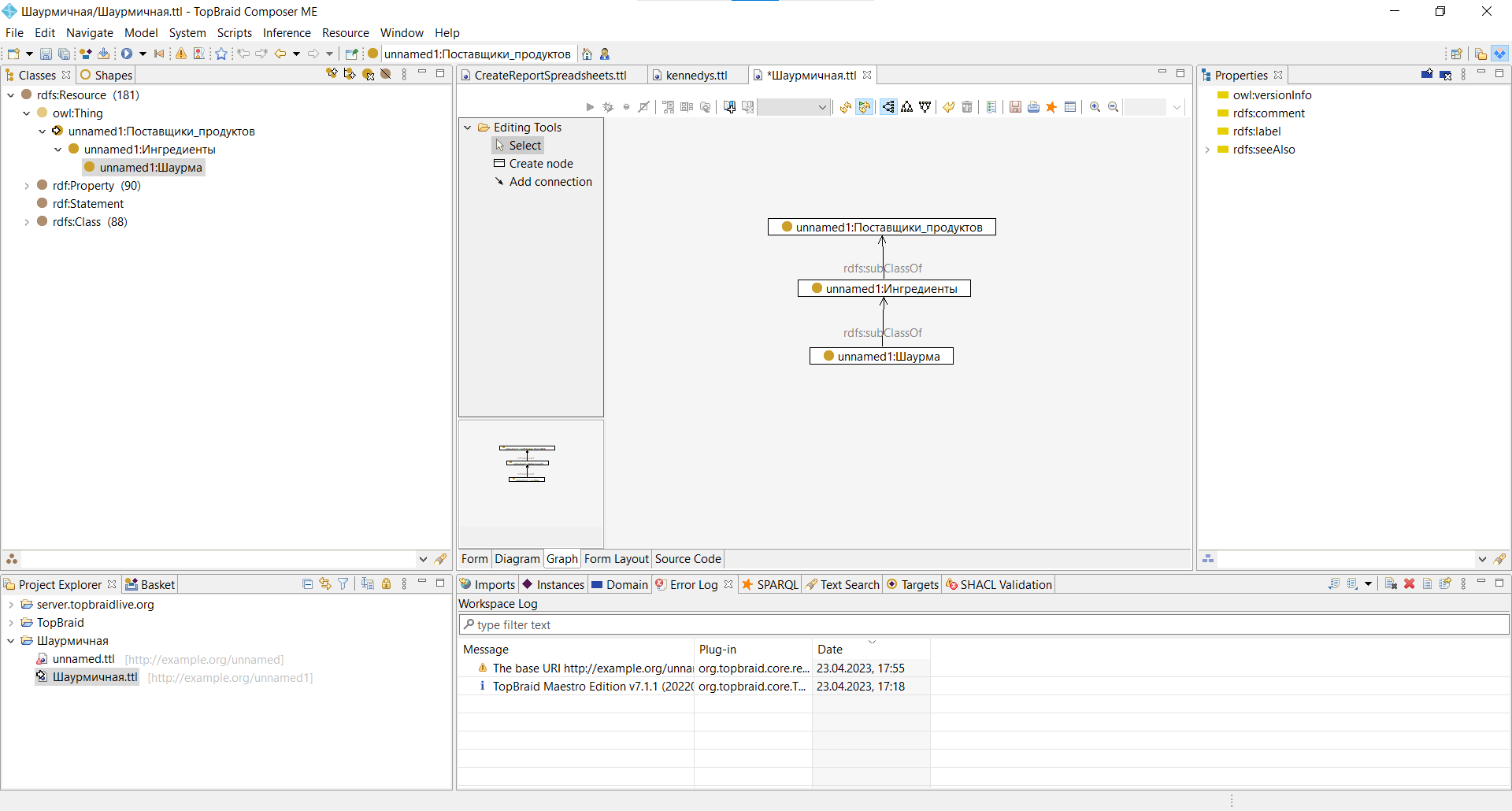
Одним из недостатков этой программы является перегруженный интерфейс различными функциями, которые не будут применяться обычным пользователем при создании простой онтологии. Кроме того, в интерфейсе программы использованы понятия из области онтологического инжиниринга: инстанции, метрики, аксиомы и т. д., что может привести к недопониманию у пользователей, не имеющих знаний в области инженерии знаний (см. рис. 2).



*Рисунок 2. Окно интерфейса Protégé для создания онтологии*

* + 1. Программное средство TopBraid Composer

TopBraid Composer [9] – это инструмент для создания и редактирования онтологий, который предоставляет пользователю мощный набор функций для разработки семантических моделей данных. TopBraid Composer основан на языке RDF и использует спецификации OWL и SPARQL для описания и запросов семантических данных. Одним из преимуществ TopBraid Composer является его удобный интерфейс (см. рис. 3). Также возможность интеграции с различными базами данных и информационными системами, многообразие различных инструментов для управления онтологиями, поддержка работы с ресурсами и классами. Функционал интервьюирования экспертов в TopBraid Composer более продвинут и гибок, чем в Protege. В TopBraid Composer можно настроить более сложные и детализированные вопросы, а также логику их построения на основе ранее заданных свойств и правил.



*Рисунок 3. Окно интерфейса TopBraid Composer для создания онтологии*

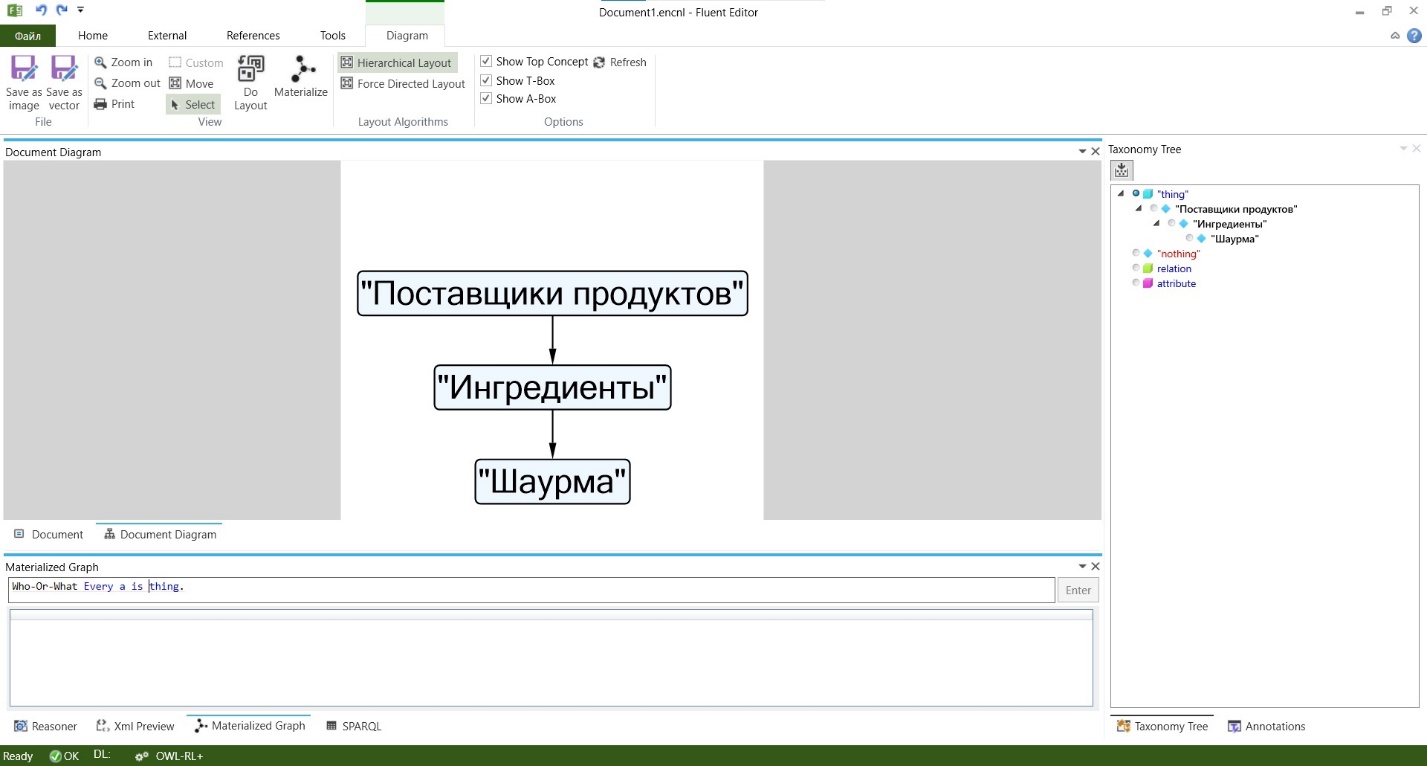
Из недостатков можно отметить сложность использования для новичков в области семантических технологий. Также TopBraid Composer может быть требовательным к ресурсам компьютера, особенно при работе с большими онтологиями.

* + 1. Программное средство Fluent Editor

Fluent Editor [1] [10] – онтологический редактор польской компании Cognitum для редактирования сложных онтологий. Основным преимуществом Fluent Editor является возможность создания OWL–онтологий людьми, не обладающими знаниями о синтаксисе OWL. Вместо этого, он позволяет создавать онтологии на естественном языке Controlled English, который является обычным английским языком, к которому применены определенные правила и ограничения. Необходимость знаний о концепциях моделирования информационных структур остается, но создание онтологий становится более доступным для людей без специализированного образования в этой области.

Fluent Editor имеет еще одну интересную особенность – использование активных правил (Active Rules). Активные правила – это механизм, который позволяет выполнять пользовательский императивный код на языке C#, при соблюдении определенных критериев, заданных как правила SWRL. С помощью активных правил можно добавлять знания в базу знаний, удалять их, а также выводить сообщения в окно вывода. Таким образом, активные правила позволяют дополнительно автоматизировать процесс работы с онтологиями, что может быть полезно при больших объемах данных и сложных запросах.

Также редактор имеет современный, удобный и приятный глазу интерфейс (см. рис. 5), который предоставляет множество удобных функций, такие как автодополнение, валидация синтаксиса, подсветка синтаксиса и другие.



*Рисунок 4. Окно интерфейса Fluent Editor для создания онтологии*

* 1. Требования к разрабатываемому ПО

После анализа сильных и слабых сторон аналогов были сформулированы следующие требования к разрабатываемому ПО.

Функциональные требования:

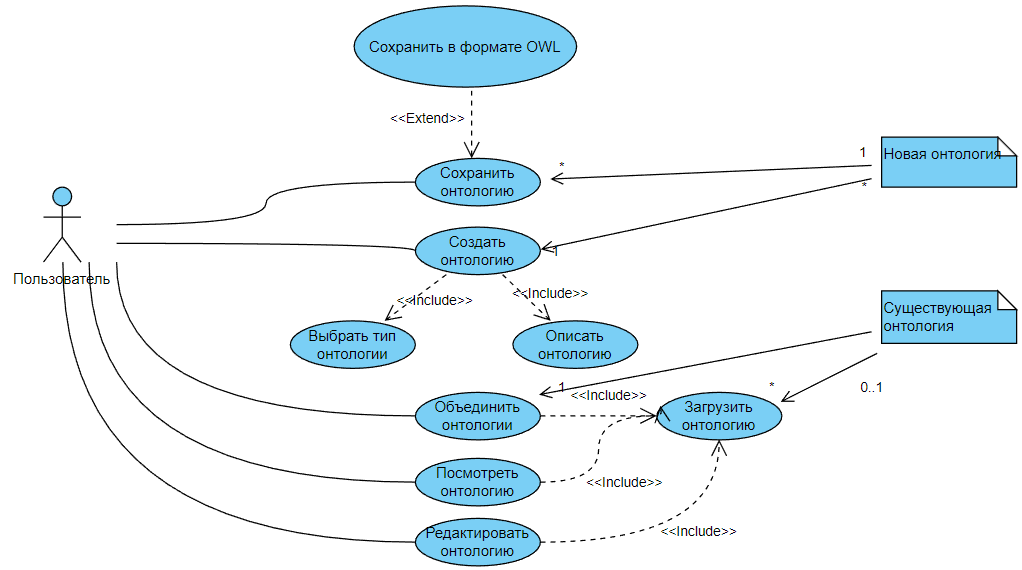
* Генерация автоматических инструкций для эксперта на основе текущего состояния онтологии.
* Отображение онтологии в графическом виде с возможностью редактирования и просмотра связей между сущностями.
* Поддержка всех инструментов моделирования метода MASK, а именно модели явлений, концептов, деятельности, задач, контекста эволюции и поколения.
* Возможность сохранять и загружать онтологии в формате OWL и XML.
* Возможность объединения онтологий.

На основе этих требований была создана диаграмма прецедентов в нотации UML (см. рис. 5), где основным актором является эксперт–пользователь системы, который имеет доступ ко всем функциям, таким как создание и редактирование онтологий, визуализация, экспорт в формат XML.

Кроме того, в диаграмме присутствуют два артефакта – "Новая онтология" и "Существующая онтология", которые представляют собой файлы, используемые пользователем внутри программы.

Нефункциональные требования:

* Настольное приложение для операционной системы Windows;
* Языком программирования для реализации программного средства был выбран ЯП C–Sharp (C#). C# является языком программирования, выбор которого обусловлен тем, что он относится к группе объектно–ориентированных языков. Это позволяет более эффективно работать с онтологиями, которые также состоят из объектов, таких как классы, атрибуты и связи. Отличительной особенностью C# является наличие множества NuGet пакетов (библиотек), которые способствуют упрощению процесса создания, хранения и визуализации онтологий в рамках .NET Framework. В отличие от других ЯП данной группы, таких как Python, C++, Ruby, Java, C# обеспечивает большую гибкость и производительность при работе с онтологиями;
* Для реализации пользовательского интерфейса был выбран Windows Forms (WF) — интерфейс программирования приложений, который совместим с ЯП C# в рамках одной платформы .NET Framework. Этот выбор обусловлен не только простотой использования и понятностью итоговых форм для всех пользователей, но и широким использованием WF в разработке стандартных приложений Microsoft Windows;
* В качестве среды разработки используется Microsoft Visual Studio 2022, за счёт своего огромного функционала, а также поддержки .NET Framework, включая C# и WF;



*Рисунок 5. Функциональные требования для системы поддержки интервьюирования эксперта для создания и поддержания онтологий*

ГЛАВА 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ ПОДДЕРЖКИ ИНТЕРВЬЮИРОВАНИЯ ЭКСПЕРТА ДЛЯ СОЗДАНИЯ И ПОДДЕРЖАНИЯ ОНТОЛОГИЙ

В данной главе будет происходить проектирование непосредственно внутренней архитектуры системы путём создания диаграмм классов и компонентов.

2.2. Проектирование взаимодействия компонентов системы поддержки интервьюирования эксперта

Далее был создан класс Ontology для представления онтологий, который содержит необходимые поля для хранения информации об онтологии, такие как идентификатор (целое число), название (строка), описание (строка), предметная область (объект типа SubjectArea), список концептов для модели концептов (список объектов типа Concept), список активных полей для модели явлений (список объектов типа Field), список явлений (список объектов типа Phenomenon), список активностей (список объектов типа Activity), список задач (список объектов типа Task), список отношений (список объектов типа IRelation) и свойств (список объектов типа Property). Класс SubjectArea используется для представления информации о предметной области онтологии и содержит следующие информационные поля: наименование (строка), цели использования онтологии (строка) и целевая аудитория онтологии (строка).

Для управления всеми созданными или загруженными в программу онтологиями был создан класс OntologyManager, который представляет собой менеджер онтологий. Он содержит единственное поле – список онтологий (список объектов типа Ontology) и два метода – добавление онтологий в этот список и извлечение определенной онтологии из списка по ее ключу.

Схематичное изображение связей между классами представлено на рисунке 7.

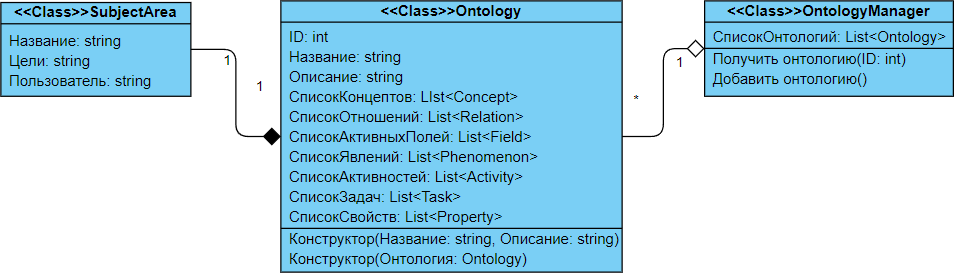


Рисунок 7. Классы, связанные с классом онтологии

Для представления концептов онтологии в программе был создан интерфейс IConcept и класс Concept, который реализует данный интерфейс. Интерфейс IConcept содержит два поля – идентификатор (число) и название (строка), а класс Concept, помимо этих данных, также содержит поля для описания (строка), идентификатор родителя (число), идентификатор онтологии (число), список свойств (список объектов типа Property) и список дочерних концептов (список объектов типа Concept), также функции добавления и удаления дочернего концепта. На диаграмме классов ниже показана связь между этими сущностями (см. рис. 8).

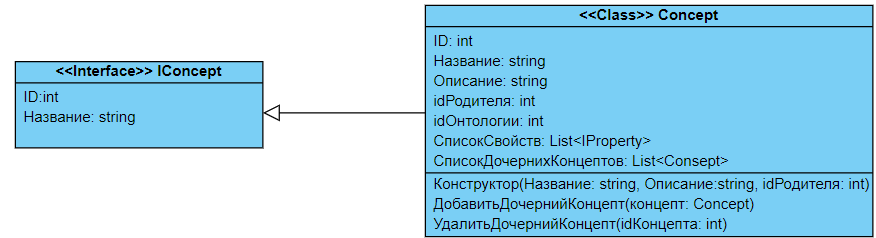
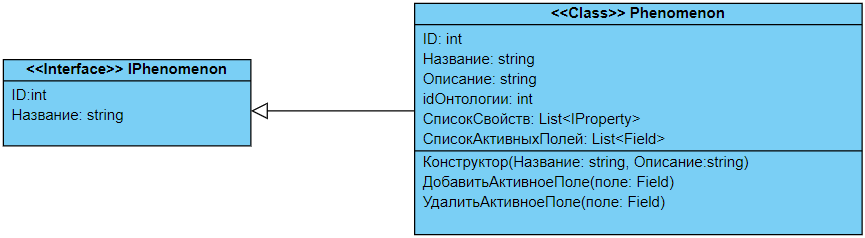


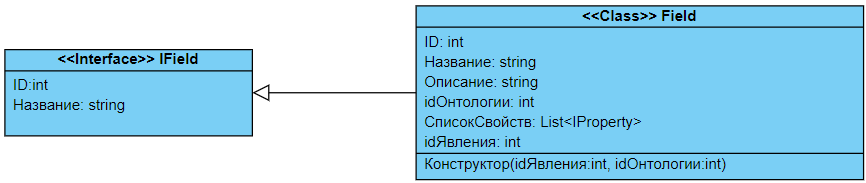
Рисунок 8. Программное представление концептов онтологии

Для представления явлений онтологии в программе был создан интерфейс IPhenomenon и класс Phenomenon, который реализует данный интерфейс. Интерфейс IPhenomenon содержит два поля – идентификатор (число) и название (строка), а класс Phenomenon, помимо этих данных, также содержит поля для описания (строка), идентификатор онтологии (число), список свойств (список объектов типа Property) и список активных полей (список объектов типа Field), также функции добавления и удаления активного поля. На диаграмме классов ниже показана связь между этими сущностями (см. рис. 9).



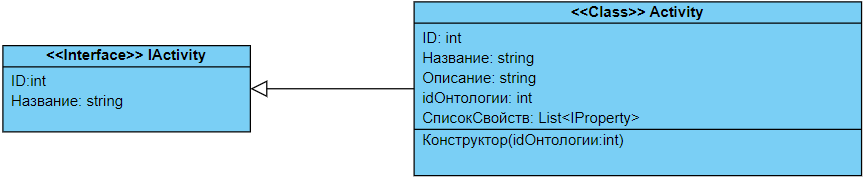
*Рисунок 9. Программное представление явлений онтологии*

Для представления активных полей онтологии в программе был создан интерфейс IField и класс Field, который реализует данный интерфейс. Интерфейс IField содержит два поля – идентификатор (число) и название (строка), а класс Field, помимо этих данных, также содержит поля для описания (строка), идентификатор онтологии (число), список свойств (список объектов типа Property) и идентификатор явления, к которому относится поле (число). На диаграмме классов ниже показана связь между этими сущностями (см. рис. 10).

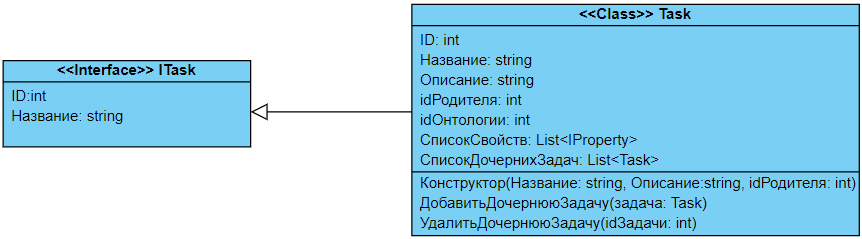
**

*Рисунок 10. Программное представление активных полей онтологии*

Для представления активностей онтологии в программе был создан интерфейс IActivity и класс Activity, который реализует данный интерфейс. Интерфейс IActivity содержит два поля – идентификатор (число) и название (строка), а класс Activity, помимо этих данных, также содержит поля для описания (строка), идентификатор онтологии (число) и список свойств (список объектов типа Property). На диаграмме классов ниже показана связь между этими сущностями (см. рис. 11).

*Рисунок 11. Программное представление активностей онтологии*

Для представления задач онтологии в программе был создан интерфейс ITask и класс Task, который реализует данный интерфейс. Интерфейс ITask содержит два поля – идентификатор (число) и название (строка), а класс Task, помимо этих данных, также содержит поля для описания (строка), идентификатор родителя (число), идентификатор онтологии (число), список свойств (список объектов типа Property) и список дочерних концептов (список объектов типа Task), также функции добавления и удаления дочернего концепта. На диаграмме классов ниже показана связь между этими сущностями (см. рис. 12).

**

*Рисунок 12. Программное представление задач онтологии*

Для представления свойств онтологии в программе были созданы интерфейс IProperty и класс Property, который реализует данный интерфейс. Интерфейс IProperty содержит два поля – идентификатор (число) и название (строка), а класс Property, помимо этих данных, также содержит поля для описания (строка), значения (объект), типа значения (перечисляемый тип) и идентификатора сущности в зависимости от типа модели, к которой относится данное свойство (число), а также метод конструктора. На диаграмме классов ниже показана связь между этими сущностями (см. рис. 13).

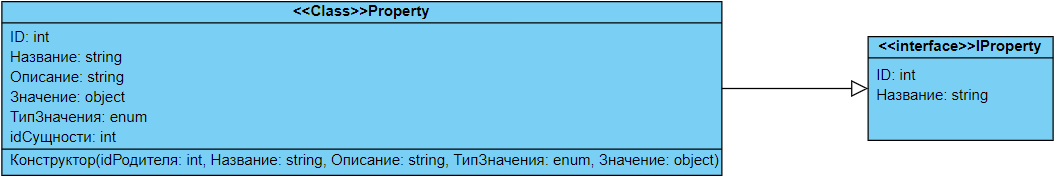


Рисунок 13. Программное представление свойств сущностей онтологии

Аналогично свойствам, для представления отношений в программе были созданы две программные сущности: интерфейс IRelation и класс Relation, который его реализует. Интерфейс IRelation содержит поля для идентификатора (число) и названия (строка), а класс Relation, помимо этих полей, также имеет поля для описания (строка), типа отношений (строка) и двух связываемых сущностей в зависимости от типа модели, а также идентификатора онтологии (число) и метод конструктора для создания объектов этого класса. На диаграмме классов (см. рис. 14) можно увидеть связь между этими программными сущностями.

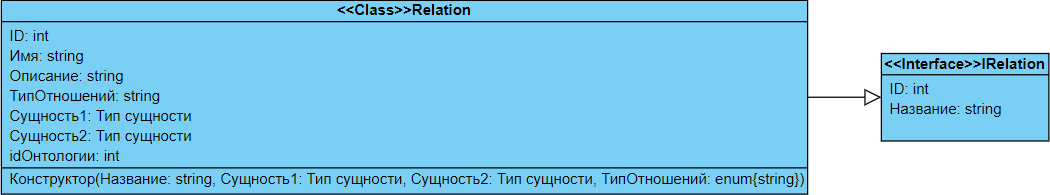
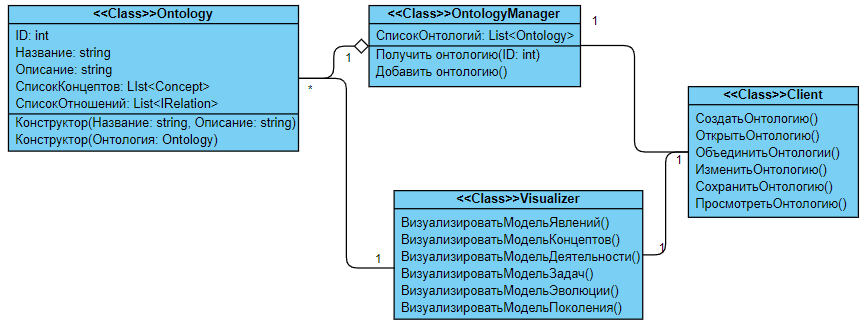


Рисунок 14. Программное представление отношений онтологии

Всё взаимодействие пользователя с системой осуществляется через класс Client, который содержит методы для создания, открытия, редактирования и просмотра онтологий, а также их объединения. Для просмотра онтологий Client использует класс Visualizer, который создан для визуализации онтологий в различных формах. Visualizer содержит методы для построения диаграммы сущностей. Для других действий с онтологиями используется класс OntologyManager, который был описан выше. Связь между этими классами представлена на диаграмме классов (см. рис. 15).

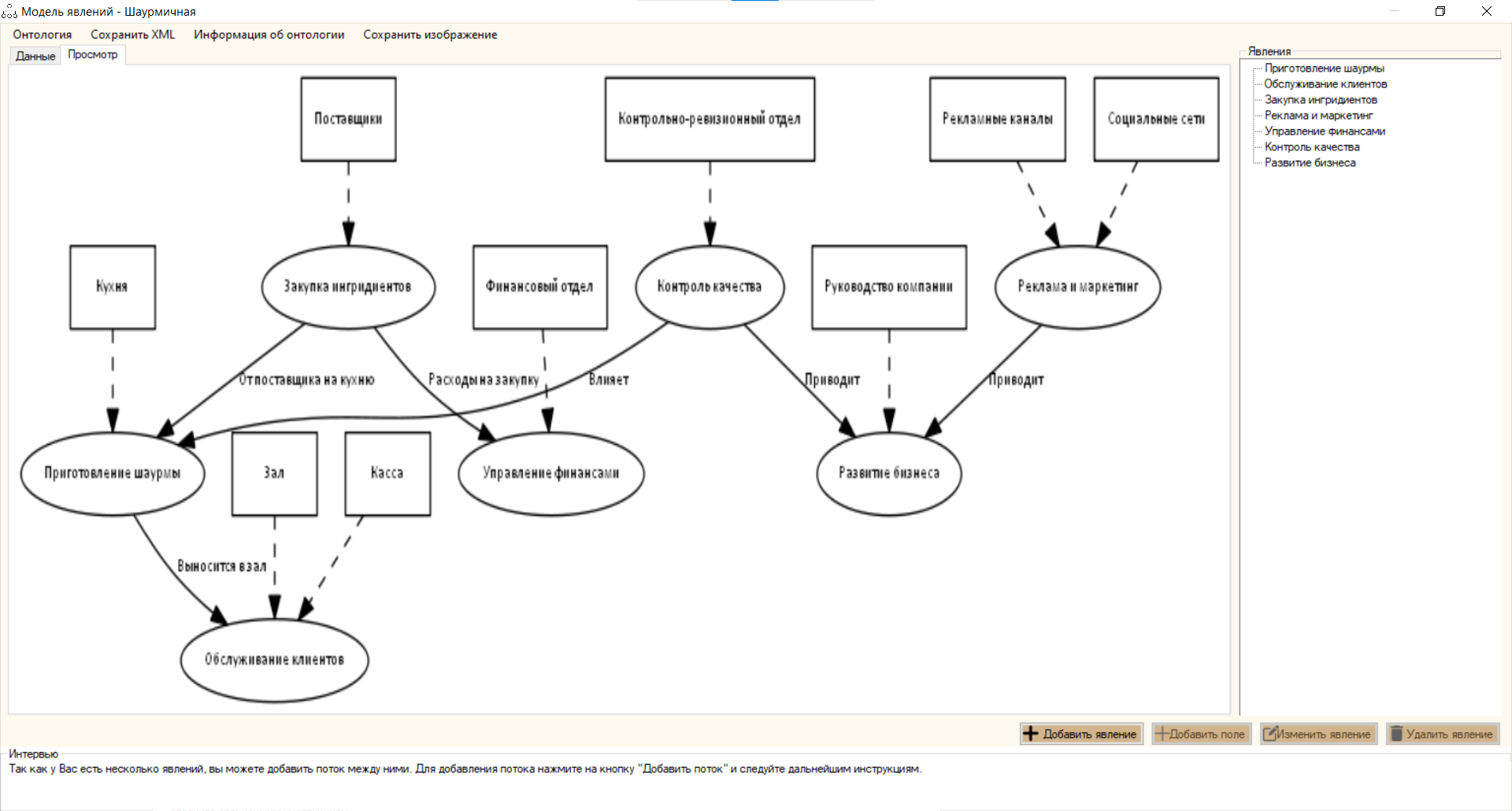
Рисунок 15. Классы для обработки онтологий

ГЛАВА 3. РЕАЛИЗАЦИЯ ПРОГРАММНОГО РЕШЕНИЯ

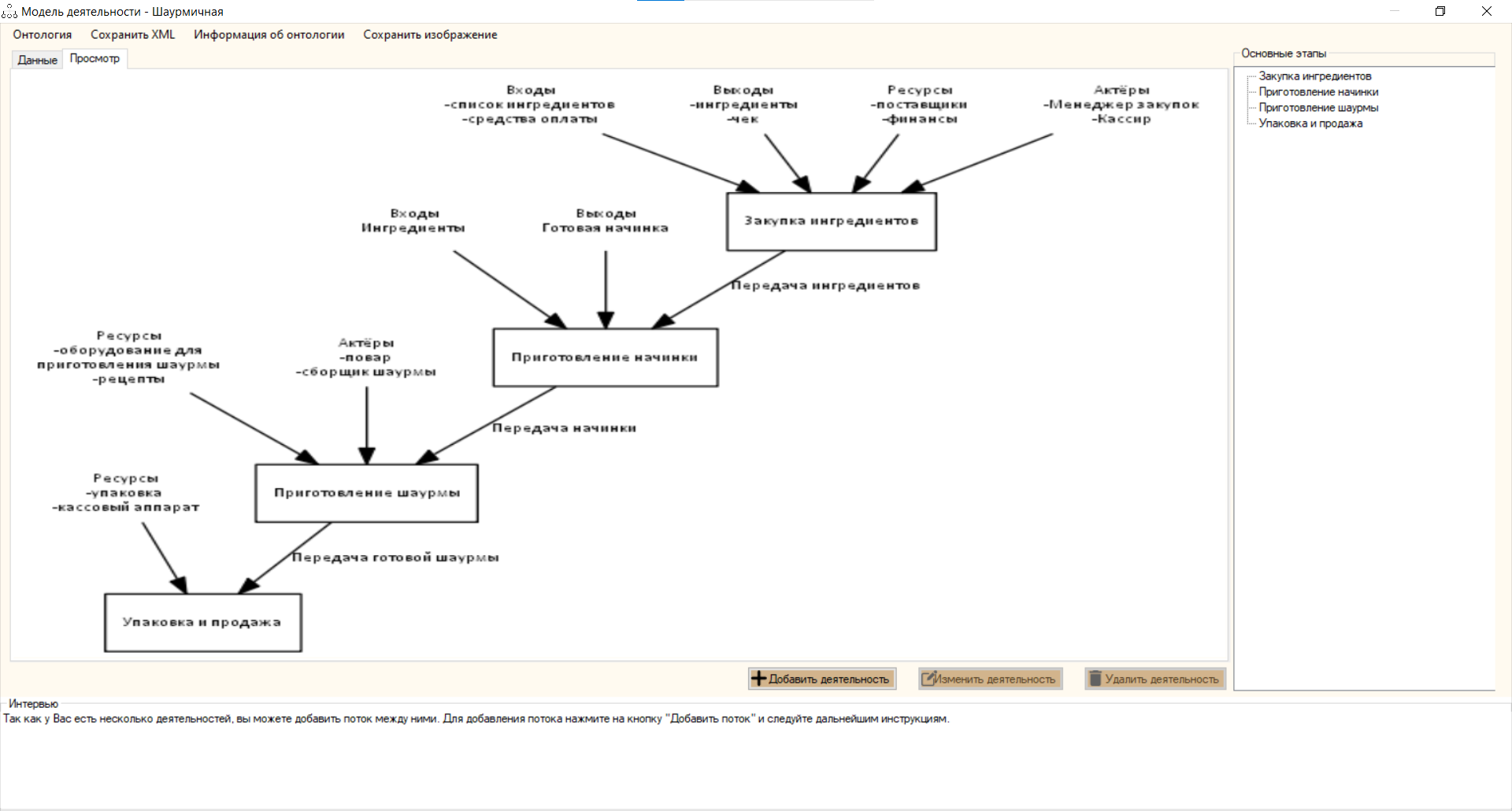
В данной главе рассмотрена реализация программного решения, разработанного для поддержки интервьюирования эксперта и построения онтологий с помощью метода MASK. Для реализации использовались современные технологии программирования на языке C# и библиотека Graphviz [4].

3.1. Реализация форм интервьюирования

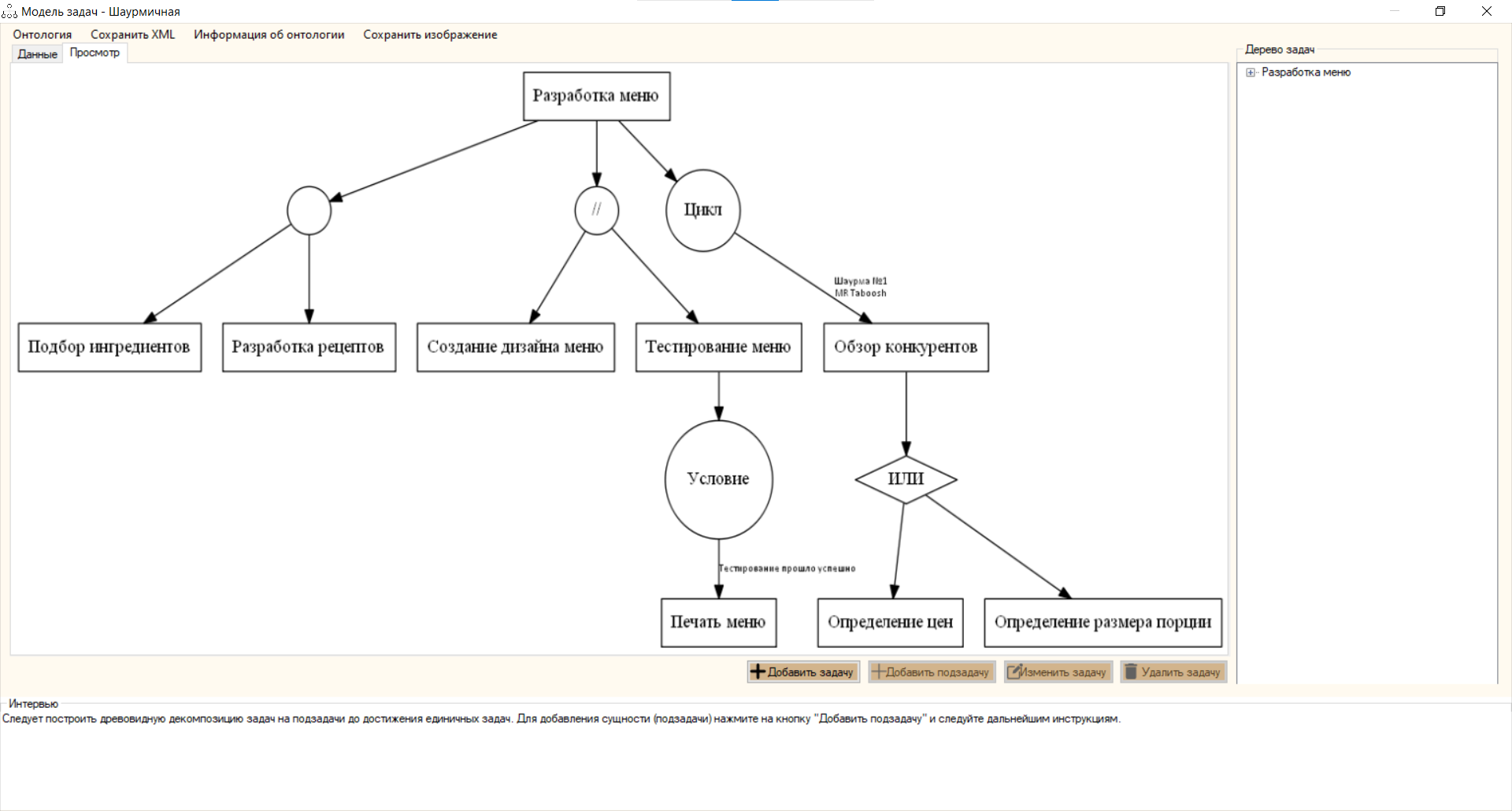
После реализации всех компонентов, представленных в прошлой главе, для каждой из моделей метода MASK была разработана отдельная форма интервьюирования (см. рис. 16–19). Каждая форма содержит ряд инструкций и вопросов, связанных с моделью, и позволяет эксперту отвечать на эти вопросы и создавать соответствующую часть онтологии.



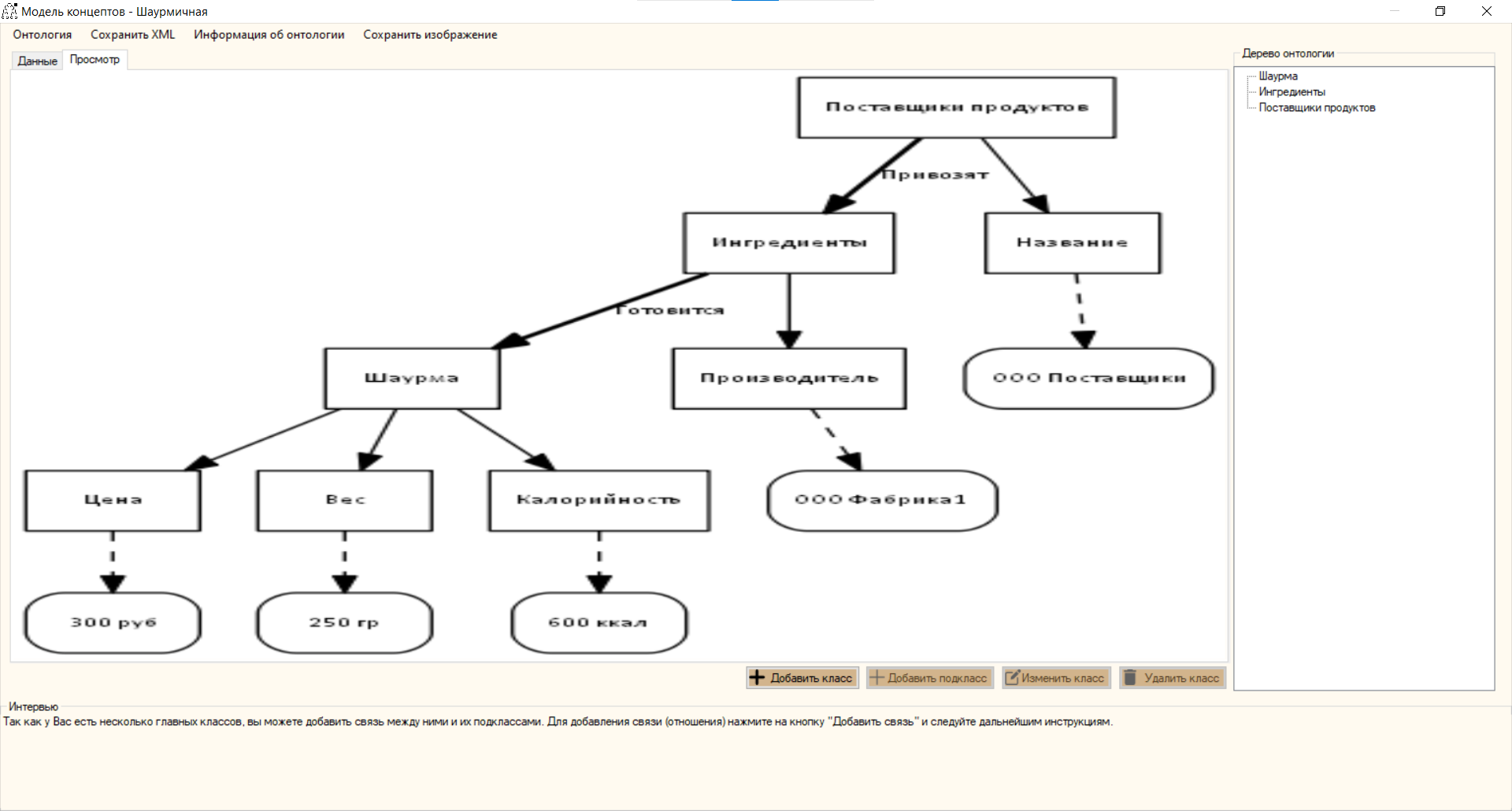
*Рисунок 16. Форма для модели явлений*

**

*Рисунок 17. Форма для модели деятельности*

**

*Рисунок 18. Форма для модели задач*

**

*Рисунок 19. Форма для модели концептов*

В C# есть несколько популярных библиотек для работы с графами ­– это NodeXL, GraphSharp и Graphviz. Так как NodeXL ориентирована на работу с графами в Excel, a GraphSharp требует знания WPF, выбор пал на библиотеку Graphviz. Она позволяет визуализировать онтологию в виде графа и сохранять граф в виде картинки. При заполнении формы интервьюирования эксперт выбирает соответствующие сущности, которые отображаются на графе.

Также Graphviz работает на основе языка DOT (Graph Description Language), который позволяет описывать структуру графов и других структур данных. DOT–язык состоит из набора команд, которые определяют узлы, ребра, атрибуты и другие свойства графа. После того, как DOT–файл сгенерирован, Graphviz обрабатывает его и создает изображение графа или диаграммы. Он может создавать изображения в различных форматах, таких как PNG, SVG, PDF, PostScript и других.

3.3. Реализация дополнительных функций

Для сохранения и загрузки онтологии в формате XML использовался DataContractSerializer. DataContractSerializer – это класс в .NET Framework, который предоставляет сериализацию и десериализацию объектов в формат XML. Он позволяет сериализовать объекты с помощью атрибутов DataContract и DataMember, которые определяют, какие поля и свойства объекта будут включены в сериализованный XML–документ.

3.4. Тестирование программы

Были составлены тесты для каждой модели разработанной системы (см. таб. 1).

Таблица 1. Тестирование системы

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Тестируемая функция | Ожидаемый результат | Фактический результат |
| 1 | Создание новой онтологии  (Создать новую онтологию) | Открытие ввода метаданных | Открытие ввода метаданных |
| 2 | Открытие готовой онтологии  (Открыть готовую онтологию) | Открытие определенной модели | Открытие определенной модели |
| 3 | Просмотр информации о программе (Информация) | Вывод информации о программе | Вывод информации о программе |
| 4 | Выход из программы (Выход) | Закрытие формы | Закрытие формы |
| 5 | Ввод метаданных | После ввода названия онтологии и выбора модели возможность перейти на след. форму ввода предметной области | После ввода названия онтологии и выбора модели возможность перейти на след. форму ввода предметной области |
| 6 | Ввод предметной области | После ввода предметной области возможность перейти на основную форму модели | После ввода предметной области возможность перейти на основную форму модели |
| 7 | Загрузка готовой онтологии из текущей (Онтология → Открыть) | Если типы онтологий совпадают, то успешная загрузка онтологии, если нет, то вывод инф. об ошибке | Если типы онтологий совпадают, то успешная загрузка онтологии, если нет, то вывод инф. об ошибке |
| 8 | Сохранение в формате XML (Сохранить XML) | Успешное сохранение | Успешное сохранение |
| 9 | Переключение на другую онтологию (Онтология → Открытые онтологии) | Обновление информации на форме | Обновление информации на форме |
| 10 | Создание новой онтологии из текущей (Онтология → Создать новую) | Открытие стартовой формы | Открытие стартовой формы |
| 11 | Просмотр информации об онтологии (Информация об онтологии) | Вывод информации об онтологии | Вывод информации об онтологии |
| 12 | Просмотр предметной области (Информация об онтологии → Подробно) | Вывод предметной области онтологии | Вывод предметной области онтологии |
| 13 | Редактирование информации об онтологии (Информация об онтологии → Редактировать) | Возможность редактировать информацию об онтологии и её сохранение | Возможность редактировать информацию об онтологии и её сохранение |
| 14 | Редактирование предметной области (Информация об онтологии → Подробно → Редактировать) | Возможность редактирования информации о предметной области онтологии и её сохранение | Возможность редактирования информации о предметной области онтологии и её сохранение |
| 15 | Просмотр онтологии в виде графика (Просмотр) | Отображение корректного графика на форме | Отображение корректного графика на форме |
| 16 | Сохранить график в виде изображения (Сохранить изображение) | Успешное сохранение графика в виде изображения | Успешное сохранение графика в виде изображения |
| 17 | Добавление сущности | Появление сущности на форме | Появление сущности на форме |
| 18 | Просмотр сущности | Вывод информации о сущности | Вывод информации о сущности |
| 19 | Изменение сущности | Возможность редактировать информацию о сущности и её сохранение | Возможность редактировать информацию о сущности и её сохранение |
| 20 | Удаление сущности | Удаление сущности с формы | Удаление сущности с формы |
| 21 | Закрытие формы | Предупреждение, если да – то закрытие формы | Предупреждение, если да – то закрытие формы |
| 22 | Развёртывание формы | Корректное развёртывание формы | Корректное развёртывание формы |

В таблице 2 приведена проверка тестов таблицы 1 по критериям «чёрного ящика».

Таблица 2. Проверка тестов по критериям чёрного ящика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Критерии | | | |
| Тест | Модель явлений | Модель деятельности | Модель задач | Модель концептов |
| 1 | + | + | + | + |
| 2 | + | + | + | + |
| 3 | + | + | + | + |
| 4 | + | + | + | + |
| 5 | + | + | + | + |
| 6 | + | + | + | + |
| 7 | + | + | + | + |
| 8 | + | + | + | + |
| 9 | + | + | + | + |
| 10 | + | + | + | + |
| 11 | + | + | + | + |
| 12 | + | + | + | + |
| 13 | + | + | + | + |
| 14 | + | + | + | + |
| 15 | + | + | + | + |
| 16 | + | + | + | + |
| 17 | + | + | + | + |
| 18 | + | + | + | + |
| 19 | + | + | + | + |
| 20 | + | + | + | + |
| 21 | + | + | + | + |
| 22 | + | + | + | + |

В результате тестирования не было выявлено ошибок в работе программы. Приложение можно считать находящимся в релизной стадии.

ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы была разработана программа, которая поддерживает процесс интервьюирования эксперта для создания и поддержания онтологий, используя язык программирования C#, язык разметки XML, а также средства Windows Forms и Graphviz. Для этого был проведён аналитический обзор методов построения онтологий на основе интервьюирования, в результате которого для реализации системы был выбран метод MASK, и анализ существующих программных средств, предоставляющих функционал по построению онтологий. На основе результатов анализа был сформирован список из функциональных и нефункциональных требований к программе. В процессе проектирования программы была разработана диаграмма классов UML. На этапе реализации был создан пользовательский интерфейс в Windows Forms, было реализовано 4 модели из 7, а именно 14 форм, 12 классов, 7 интерфейсов, написано 19168 строк кода и проведено тестирование программы с помощью набора тестов «чёрного ящика», который был составлен в соответствии с программой и методикой испытаний. В дальнейшем программа может быть улучшена путём добавления сохранения онтологий в формате OWL, написания функции объединения онтологий и добавлением двух моделей – контекста эволюции и поколения. Также в дипломной работе планируется реализовать данную систему в виде веб-приложения.

Список литературы

1. Fluent Editor. [Электронный ресурс]. Доступен по ссылке: <https://www.cognitum.eu/semantics/fluenteditor/>
2. *Gennari J. H.* The evolution of Protégé: an environment for knowledge-based systems development //International Journal of Human-computer studies. – 2003. – Т. 58. – №. 1. – P. 89-123.
3. *Grabusts, P.* Information structures in the framework of information warfare - ontology approach //Открытые семантические технологии проектирования интеллектуальных систем. – 2019. – No 3. – P. 145-150.
4. Graphviz. [Электронный ресурс]. Доступен по ссылке: <https://graphviz.org>
5. *Nada Matta, Jean-Louis Ermine, Gérard Aubertin, Jean-Yves Trivin.* Knowledge Capitalization with a knowledge engineering approach: the MASK method. – 2002. – P. 7-13.
6. Protégé. [Электронный ресурс]. Доступен по ссылке: <https://protege.stanford.edu>
7. *Serge Aries, Benoît Le Blanc, Jean-Louis Ermine.* MASK: une méthode d’ingénierie des connaissances pour l’analyse et la structuration des connaissances. Management et ingénierie des connaissances: modèles et méthodes, Hermes Science Publications-Lavoisier, 2008. – P. 8-22.
8. *Suárez-Figueroa M. C.* The NeOn methodology for ontology engineering / Suárez-Figueroa M. C., A. Gómez-Pérez, M. Fernández-López //Ontology engineering in a networked world. – Springer, Berlin, Heidelberg, 2012. – С. 9-34.
9. TopBraid Composer. [Электронный ресурс]. Доступен по ссылке: <https://franz.com/agraph/tbc/>
10. *Weichbroth P.* Fluent editor and controlled natural language in ontology development //International Journal on Artificial Intelligence Tools. – 2019. – Т. 28. – №. 04. – P. 19-40.