УДК 004.942

А.М. Наместников, Г.Ю. Гуськов

СИСТЕМА УПРАВЛЕНИЯ ПРОГРАММНЫМИ ПРОЕКТАМИ НА ОСНОВЕ ОНТОЛОГИЧЕСКОГО ПОДХОДА[1]

Наместников Алексей Михайлович, *кандидат технических наук, доцент, окончил радиотехнический факультет Ульяновского государственного технического университета. Доцент кафедры «Информационные системы» УлГТУ. Имеет более 80 работ в области автоматизированного проектирования и интеллектуальных систем. [e-mail:nam@ulstu.ru].*

Гуськов Глеб Юрьевич, *аспирант кафедры «Информационные системы», окончил факультет информационных систем и технологий УлГТУ. Ассистент кафедры «Информационные системы» УлГТУ. Имеет работы в области онтологического моделирования и интеллектуального анализа временных рядов. [e-mail: g.guskov@ulstu.ru].*

Аннотация

В процессе проектирования информационных систем основным артефактом проектной деятельности являются UML-диаграммы. Для обеспечения преемственности разработки и организации взаимодействия аналитиков и программистов необходимо адекватное представление понятий предметной области. Хранение и обработку данных и знаний о предметной области удобно производить в формате онтологии OWL. В статье описан инструмент, позволяющий автоматизированно интегрировать UML-диаграммы и онтологию OWL. UML-диаграммы проекта переводятся в формат XMI и по ним строится онтология, которая впоследствии доступна для дополнения со стороны специалиста по онтологическому проектированию. Онтология может быть расширена введением правил, ограничивающих возможности по построению UML-диаграмм и, как следствие, программного продукта в соответствии с предметной областью.

Основным преимуществом онтологического подхода при управлении программными проектами является проверка на непротиворечивость полученной онтологии и связанных с ней UML-диаграмм. Подобная проверка должна производиться при изменении UML-диаграмм или онтологии предметной области, что обеспечивает поддержание системы в непротиворечивом состоянии на протяжении всего жизненного цикла программного продукта.

На базе полученной онтологии могут быть построены новые программные продукты, которые будут совместимы с уже имеющимися. Общая онтологическая база программных продуктов позволит минимизировать издержки при интеграции как проектов, так и отдельных модулей.

В статье приведены результаты вычислительного эксперимента по трансляции UML-диаграмм из открытых источников (github) в онтологию OWL, а также анализ частоты использования элементов диаграммы классов из транслированных диаграмм. Также в статье приводятся результаты проверки на согласованность сгенерированных онтологий и предлагаются дальнейшие пути развития данного направления.

Статья рассчитана на ИТ-специалистов и специалистов по онтологическому моделированию.

1 Исследование поддержано Российским фондом фундаментальных исследований в рамках выполнения проекта № 16-47-732033 «Разработка моделей и средств онтологического анализа проектных диаграмм на основе методов машинного обучения», Министерством образования и науки Российской Федерации в рамках выполнения проекта № 1167 «Разработка нового подхода к интеллектуальному анализу слабоструктурированных информационных ресурсов» по государственному заданию.

Ключевые слова: онтология предметной области, UML-диаграмма, система логического вывода (reasoner).

THE SOFTWARE PROJECT MANAGEMENT SYSTEM BASED ON THE ONTOLOGY APPROACH

Aleksei Mikhailovich Namestnikov, *Candidate of Engineering, Associate Professor, Associate Professor at the Department of Information Systems at Ulyanovsk State Technical University; graduated from the Faculty of Radioengineering of Ulyanovsk State Technical University; an author of more than 80 papers in the field of CAD and intelligent systems. e-mail: nam@ulstu.ru.*

Gleb Iurevich Guskov, *Postgraduate Student at the Department of Information Technologies of Ulyanovsk State Technical University; graduated from the Faculty of Information Systems and Technologies of UlSTU; Assistant at the Department of Information Systems of UlSTU; an author of papers in the field of ontology modeling and intellectual time series analysis. e-mail: g.guskov@ulstu.ru.*

Abstract

In information system designing process, one of the most important artefacts of project activities is a set of UML-diagrams. In order to ensure development continuity and organization of interaction between the analysts and programmers, the structure of the domain concepts should be provided. Storage and processing of domain data and knowledge can be presented optimally in the OWL ontology format. The article describes a tool that allows automated integration of the UML-diagrams and OWL-ontologies. Project UML-diagrams are translated into a XMI-format. After that, the ontology is constructed on the basis of the diagrams, which is available for extension by the specialist in ontological engineering. The ontology can be extended by the introduction of rules limiting the opportunities for building UML-diagrams and, as a consequence, for a software product in accordance with the domain.

The main advantage of the ontological approach in the management of software projects is checking the consistency of ontology and receiving the associated UML-diagrams as a result. Such checks should be carried out when changing the UML-diagrams or the domain ontology that provides consistent state of a system during the software product life cycle.

On the basis of the obtained ontology, new software compatible with the already existing ones can be designed. General ontological base of the software can help to minimize costs of integration of the projects as well as individual modules.

The article contains an experiment of translating UML-diagrams from open source projects (github) to the OWL-ontology as well as the analysis of the frequency of the use of class diagrams elements. Moreover, the article presents the results of checking the consistency of the ontologies generated and offers the further ways of this direction development.

This article is intended for IT professionals and experts in ontological modeling.

Key words: domain ontology, UML-diagram, reasoner.

Введение

Одним из результатов проектирования сложных информационных систем (ИС) является набор UML-диаграмм [1, 2]. Часто на последующих итерациях разработки проекта не все изменения функционирования ИС отражаются в UML-диаграммах. Часть изменений ИС концептуальны и должны быть отражены в требованиях к проекту, что приводит к обновлению технического задания и архитектуры проекта [3]. Лучшей практикой является корректировка технического задания или доработка приложения. Изменение или дополнение технического задания является весьма трудоемким процессом, но ещё более трудоемким является процесс актуализации знаний о техническом задании у субъектов проектирования. Временные рамки проекта зачастую не позволяют вносить изменения в проектные диаграммы, вместо этого изменения вносятся на уровне исходного кода.

Решение проблемы может быть получено путем адаптации технологии OWL [4, 5] для промышленной разработки ИС. Построение проектных диаграмм на основе единой в рамках реализуемых проектов непротиворечивой онтологии предметной области приведет к существенной оптимизации временных затрат на разработку. Оптимизация будет получена за счет гарантированной совместимости проектов между собой и семантической проверки проектных диаграмм. Совместимость проектов обеспечивается семантически схожей структурой классов, объектов, интерфейсов и процессов на уровне предметной области. Семантическая проверка проектных диаграмм позволяет исключить два вида ошибок, допускаемых при разработке: ошибки в требованиях к проекту и концептуальные ошибки предметной области в архитектуре проекта.

Для интеграции онтологий в промышленную разработку ИС UML-диаграммы выбраны как наиболее подходящие артефакты процесса разработки, так как они содержат информацию о предметной области, частично абстрагированную от реализации в формализованном виде.

В настоящее время язык представления онтологий OWL не имеет развитой инфраструктуры для интеграции с основными инструментами разработки. Известны способы сопоставления OWL и элементов UML-диаграмм, которые не в полной мере решают вышеописанные задачи [6, 7].

В основном язык OWL используется обособленно от разработки программного обеспечения для хранения формализованных знаний по предметной области. Также OWL часто используют для унификации знаний о предметной области между специалистами в разных странах. В частности, большое распространение получила медицинская онтология, собравшая в себе знания о болезнях, симптомах, диагнозах, формате ведения истории пациента [8].

1 Архитектура, возможности и ограничения модуля трансляции

Среди UML-диаграмм абсолютным лидером по применению является диаграмма классов. Диаграммы классов распространяются прямо пропорционально развитию объектно-ориентированного программирования в целом, так как диаграмма классов неявно присутствует в каждом объектно-ориентированном проекте (ООП). Даже если UML-диаграмма классов не была построена, её всегда можно автоматически построить с помощью обратного проектирования.

За счет распространения технологии и культуры программирования проекты, написанные с учетом универсальных требований к стилю программирования, используют длинные мнемонические имена и общепринятые шаблоны проектирования. Данная унификация позволяет использовать UML-диаграмму классов автоматизированно.

UML-диаграмма классов модуля системы, отвечающего за трансляцию UML-диаграммы классов в онтологию, приведена на рисунке 1.

Модуль разработан для использования XMI (XML Metadata Interchange) формата. Формат файла XMI был предложен консорциумом OMG (Object Management Group) и является единственным общепринятым форматом хранения UML-диаграмм. К сожалению, не все программные редакторы UML-диаграмм поддерживают сохранение UML-диаграммы в формате XMI.

На данный момент модуль поддерживает версии XMI от 2.1 и выше. В таблице 1 приведены результаты тестирования по трансляции диаграмм классов, разработанных в разных редакторах.

Как видно из приведенной таблицы, далеко не все редакторы поддерживают XMI-экспорт. Тем не менее, XMI является единственным унифицированным форматом.

2 Принципы и процедура трансляции UML-диаграмм

Типы данных в UML и OWL совпадают не полностью. Базовые типы данных как в UML, так и в OWL были заимствованы в XSD (XML *schema definition*). Общий формат экспорта UML-диаграмм в XMI полностью основан на XSD-типах данных. Пользовательские типы данных UML и OWL основаны на одном и том же наборе базовых типов, что предполагает их взаимную конвертацию.

Понятие класса существует как в UML, так и в OWL. Класс в OWL является множеством экземпляров, а в UML класс определяется как структурная единица диаграммы. Для класса в UML-диаграмме правомерно использовать определение из ООП. Классы предметной области связа-

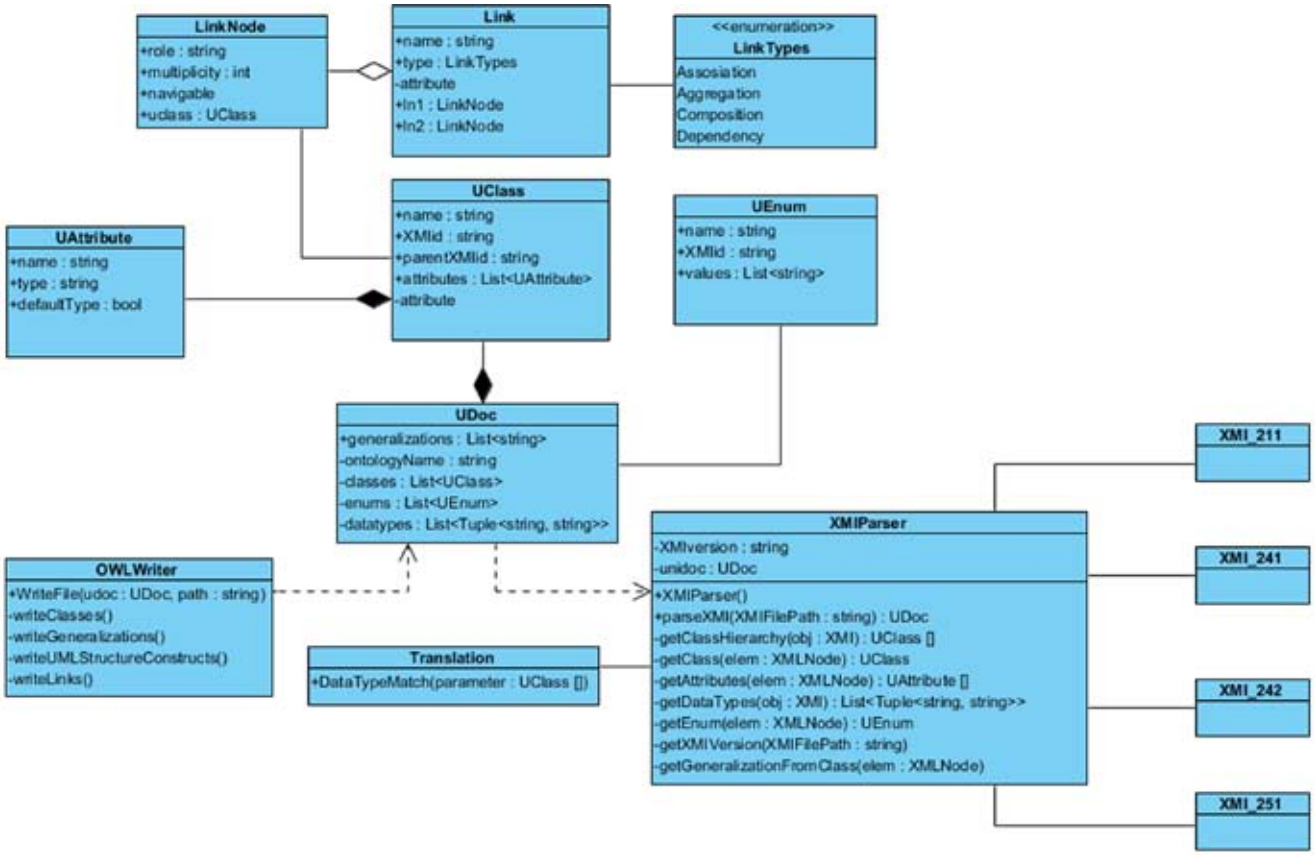

Рис. 1. Диаграмма классов модуля трансляции UML-диаграмм в онтологию

Таблица1

Результаты тестирования экспорта диаграмм в XMI-формат

Редактор	Версия XMI	Результат транслирования	Комментарий
ArgoUML	1.2	Не транслировано	Слишком старая версия XMI
Astah	1.1	Не транслировано	Слишком старая версия XMI
Altova UModel	2.1, 2.4.1	Транслировано	Редактор может быть использован для экспорта диаграмм в XMI-формат
Enterprise Architect	1.1, 2.1	Транслировано	Редактор может быть использован для экспорта диаграмм в XMI-формат
MagicDraw	Есть	Не транслировано	Экспорт в XMI не доступен в бесплатной версии
Innovator Enterprise	Нет	Не транслировано	Экспорт в XMI не доступен в бесплатной версии
Modelio	Нет	Не транслировано	Экспорт в XMI не доступен в бесплатной версии
StarUML	Нет	Не транслировано	Экспорт в XMI не доступен в бесплатной версии
VisualParadigm	2.1	Транслировано	Редактор используется как основное средство при разработке трансляции UML-диаграмм

ны отношениями с помощью объектных свойств и литеральными значениями с помощью свойств типов данных.

Трансляция интерфейсов предполагает создание в онтологии класса, на который наложен запрет на создание индивидуалов. При этом наследники класса, реализующие данный интерфейс, по логике ООП могут иметь индивидуалы. Предложено ввести свойства объекта (*ObjectProperty*) – *IsInterface* и *RealiseInterface*. Свойство *IsInterface* определяет, что класс онтологии, обладающий этим свойством, является интерфейсом. Свойство *RealiseInterface* утверждает, что класс онтологии, находящийся в области определения *Domain*, – это наследный класс, а класс онтологии в интервале значений Range является интерфейсом, и для него должно быть определено свойство *IsInterface*.

UML-диаграмма классов рассматривает класс как совокупность внутреннего устройства и поведения объектов. Правила нотации UML предоставляют аналитику широкие возможности для выражения диаграммы классов, которые тем или иным способом реализуются в конкретном инструменте. Концепты класса отличаются семантически, но возможна трансляция на уровне иерархии классов. Внутреннее устройство классов UML-диаграммы можно транслировать в OWL-классы с использованием объектных свойств и свойств типов данных.

В процессе проведенного исследования была решена задача построения процедуры трансляции иерархии классов UML в иерархию классов OWL. Из определения наследования как в UML, так и в OWL следует, что каждый экземпляр (индивид) класса наследника является экземпляром базового класса. Ввиду совпадения семантики данного понятия в ходе реализации инструмента трансляции было принято решение транслировать наследование без учета дополнительной логики.

Большинство объектно-ориентированных языков программирования не поддерживают множественное наследование в связи с проблемой неоднозначности наследования отдельных членов класса. Тем не менее, при построении предметной области подобная конструкция допустима. Множественное наследование влечет за собой проблему неоднозначного наследования одинаковых свойств и методов родительских классов. Данная проблема решается либо путем явного указания родителя при наследовании общих элементов классов, либо введением интерфейсов. Наибольшее распространение получило использование интерфейсов, поэтому стоит ввести правило ограничения наследования классов и добавить интерфейсы в процесс трансляции.

Атрибуты класса по типам данных можно разделить на примитивные типы (*XSD shema*) и пользовательские (атрибут содержит объект класса, атрибут типа перечисление и т. д.).

Атрибуты примитивного типа данных транслируются в онтологию как свойство *DataTypeProperty* с областью определения *Domain*, равной классу, которому принадлежит атрибут, и рангом значений *Range*, равным примитивному типу. Перечисление транслируется в заданный пользователем тип данных, для которого определены возможные значения и свойство *DataTypeProperty*. Атрибут типа класса транслируется в свойство *ObjectProperty* с областью *Domain*, равной классу, которому принадлежит атрибут, и рангом *Range*, равным классу типа атрибута.

Практически все связи, применяемые при проектировании UML-диаграммы классов, не имеют прямого аналога в онтологическом представлении предметной области. Однако отношения между классами в OWL можно модифицировать, добавляя аксиомы, ограничения или свойства. Для трансляции UML-диаграмм в онтологию изначально создается набор отношений, характеризующих различные связи из диаграмм.

Исключением из правила является связь обобщения (*generalization*), которая имеет аналогичную по смыслу связь в онтологическом представлении (*subClassOf*).

Ассоциативная связь в UML не имеет прямого аналога

в OWL. Данная связь означает, что объект одного класса связан с объектом другого класса. Агрегация и композиция по спецификации представляют собой разновидности ассоциации, реализующие отношения «часть-целое».

Ассоциации предлагается представлять в явном виде, как три отношения: два *ObjectProperty* для каждого атрибута связанных классов и *ObjectProperty* с названием, соответствующим наименованию ассоциации. Идея подобного способа выражения ассоциаций состоит в том, что такого рода связи выражают интерпретацию предметной области в виде программного продукта, но не являются частью этой предметной области.

Зависимость (*dependency*) – связь, которая означает, что при изменении класса зависимый от него класс так же необходимо изменить с точки зрения нотации UML. Зависимость действует не на уровне объектов, а на уровне программной реализации. Именно результат анализа связи зависимости послужил основой для разделения в онтологии знаний о самой предметной области и о программном продукте, построенном на ее основе. Результат трансляции связи зависимости представляет собой одно свойство *ObjectProperty*, у которого ранг *Range* хранит зависимый класс, а *Domain* – класс, от которого существует зависимость.

Обратная трансляция при подобном способе трансляции осуществима за счет однозначной трактовки результата трансляции и уникальных значений элементов, полученных из XMI.

3 Результат работы модуля трансляции

На рисунке 2 приведен фрагмент онтологии в качестве примера работы транслятора, построенный на основе UML-диаграммы (рис. 1).

Разработанный модуль трансляции был протестирован на 30 проектах открытого репозитария исходного кода *github.com*. Частотность использования элементов диаграмм в проектах указана на рисунке 3.

В результате трансляции имели место некоторые исключительные ситуации, но все элементы диаграмм, описанные в статье, были успешно транслированы в онтологию. Стоит отметить, что разработчики редко пользовались широким спектром элементов. Гистограмма, приведенная на рисунке 3, свидетельствует, что выбор элементов в представленном исследовании был сделан верно.

4 Синхронизация UML-диаграмм и онтологии

Ключевым моментом в использовании разработанной системы является цикличное проектирование. Предполагается, что разработчики перед внесением любого концептуального изменения в проект на уровне исходного кода обновляют проектную часть – UML-диаграммы. Возможно внесение изменений в проектную часть после изменения исходного кода. Подобный подход к разработке не может считаться верным, так как вносить изменения в проект без анализа изменения программного продукта в целом не профессионально и допустимо лишь в сравни-

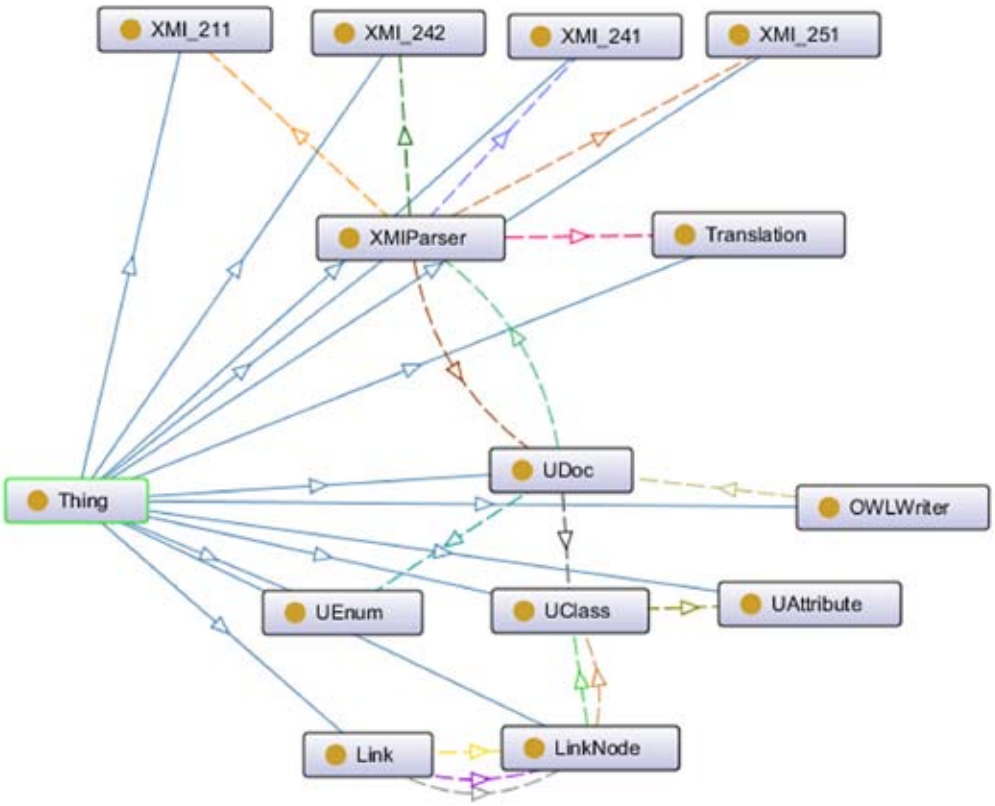

Рис. 2. Результат трансляции UML-диаграммы инструмента

тельно небольших проектах с малым временем разработки и поддержки.

Необходимо стремиться к уменьшению трудозатрат на поддержание проектной части программного продукта в актуальном состоянии. Образцом могут служить системы контроля версий, вошедшие в практику разработки программного обеспечения.

Разработанная программа трансляции позволяет синхронизировать изменения между онтологией, проектными диаграммами и исходным кодом проекта. Ее задача состоит не только в поддержании истории изменений всех фрагментов проекта, но и в сохранении связей между элементами, например, Класс А в онтологии связан с Классом А на диаграмме классов и диаграмме активностей и Классом А в исходном коде программного продукта.

5 Верификация и анализ онтологии проекта

Все основные преимущества вербализации и представления понятийной структуры проблемной области при разработке программного продукта проявляются при использовании онтологии, которая позволяет проверять проект, как совокупность содержательных диаграмм, на согласованность и получать ответы на сложные формализованные запросы.

Для проверки согласованности онтологии и исполнения запросов необходима система логического вывода. В данном проекте была использована система pellet [9], так как у нее есть система поддержки *SWRL*-правил (*Semantic Web Rule Language*).

Валидация онтологии включает в себя следующие проверки:

- проверку целостности;
- проверку классов;
- проверку отношений;
- проверку класса для каждого индивидуала.

Проверка целостности гарантирует, что онтология не содержит противоречивые факты. Проверка классов выполняется на возможное наличие индивидуалов. Проверка отношений – это вычисление отношений новых и проверка текущих отношений иерархичности по заданной онтологии. Проверка класса для каждого индивидуала вычисляет класс индивидуала в онтологии с учетом изменений онтологии.

В случае, если онтология успешно прошла валидацию, система логического вывода вернет истинное значение результата валидации. Но, если в результате валидации были получены ошибки, система вернет список отчётов с ошибками. Отчет содержит:

- текстовое описание ошибки;
- дополнительную информацию об ошибке (зависит от того, какая именно система логического вывода используется: *pellet, fact++*, и т. д.);
- статус ошибки (ошибка или предупреждение);
- тип ошибки.

SWRL-правила представляют из себя еще один аспект анализа состояния онтологий. SWRL-правила расширяют выразительную силу OWL-онтологии, позволяя задавать унарные, бинарные и n-арные предикаты.

SPARQL-запросы позволяют строить запросы и получать ответы на нетривиальные вопросы по предметной области, отчасти заменяя эксперта. Результатом выполнения такого запроса является логическое выражение или часть онтологического графа.

На текущий момент все 30 фрагментов онтологий (рис. 3), полученные при тестировании трансляции, успешно прошли проверку целостности.

Рис. 3. Гистограмма частотности использования элементов в проектах

Заключение

Наиболее значимым результатом описанного этапа исследования является успешная интеграция трансляции и анализа онтологий. Результаты тестирования показали, что система успешно работает, но для получения дальнейших результатов требуется создать более сложную систему для трансляции и анализа полного перечня конструкций. Решение следующих задач предполагает:

- добавление поддержки SWRL-правил модулем трансляции;
- реализацию модуля учета истории артефактов программного продукта;
- реализацию более сложного графического интерфейса для проведения детальных исследований над уже реализованной системой, основанного на SPARQL-запросах.

СПИСОК ЛИТЕРАТУРЫ

1. OWL 2 Web Ontology Language Document Overview. – URL: https://www.w3.org/TR/owl2-overview/ (дата обращения 19.08.16).

2. Наместников А.М. Метауровень информационного обеспечения САПР: от теории к практике. – Ульяновск : УлГТУ, 2015.

3. Ярушкина Н.Г. Гибридные системы, основанные на мягких вычислениях: определение, архитектура, возможности // Программные продукты и системы. – 2002. – № 3. – С. 19–22.

4. Almeida Ferreira D., Silva A., UML to OWL Mapping Overview An analysis of the translation process and supporting tools. Proceedings of the 13th International Conference on Information Systems for Crisis Response and Management. – ISCRAM, Delft. – the Netherlands : 2007.

5. Фаулер М. UML Основы: 3-е издание : пер. с англ. – СПб. : СимволПлюс, 2004. – 192 с.

6. OMG UML formal/2015-03-01 – URL: http://www.omg.org/spec/UML/2.5 (дата обращения 19.08.16).

7. Zedlitz J, Jorke J, and Luttenberger N. From UML to OWL2 // Knowledge Technology Volume 295. – Springer Berlin Heidelberg : 2012.

8. The OBO Foundry. – URL: http://obofoundry.org/ (дата обращения 19.08.16).

9. Pellet. – URL: https://www.w3.org/2001/sw/wiki/Pellet (дата обращения 19.08.16).

REFERENCES

1. *OWL 2 Web Ontology Language Document Overview*. Available at: https://www.w3.org/TR/owl2-overview/ (accessed 19.08.16).

2. Namestnikov A.M. *Metauroven informatsionnogo obespecheniia SAPR: ot teorii k praktike* [Meta-level of CAD Information Support: from Theory to Practice]. Ulyanovsk, ULSTU Publ., 2015.

3. Yarushkina N.G. Gibridnye sistemy, osnovannye na miagkikh vychisleniiakh: opredelenie, arkhitektura, vozmozhnosti [Soft Computing Based Hybrid System: Definition, Architecture, Opportunities]. *Programmnye produkty i sistemy* [Software Products and Systems], 2002, no. 3, pp. 19–22.

4. Almeida Ferreira D., Silva A. UML to OWL Mapping Overview An Analysis of the Translation Process and Supporting Tools. *Proc. of the 13th Int. Conf. on Information Systems for Crisis Response and Management*. ISCRAM, Delft, the Netherlands, 2007.

5. Fowler M. *UML. Foundations. 3-d Edition*. Transl. from Engl. St. Petersburg, SymbolPlus Publ., 2004.

6. *OMG UML formal/2015-03-01*. Available at: http://www.omg.org/spec/UML/2.5 (accessed 19.08.16).

7. Zedlitz J, Jorke J, and Luttenberger N. From UML to OWL2. *Knowledge Technology,* Vol. 295, Springer Publ., Berlin, Heidelberg, 2012.

8. *The OBO Foundry*. Available at: http://obofoundry.org/ (accessed 19.08.16).

9. *Pellet*. Avaialable at: https://www.w3.org/2001/sw/wiki/Pellet (accessed 19.08.16).