Выявление конфликтов в спецификации требований на основе онтологической модели и системы продукционных правил

М.Ш. Муртазина[1], Т.В. Авдеенко[1]

[1]Новосибирский государственный технический университет, Карла Маркса 20, Новосибирск, Россия, 630073

Аннотация. В статье представлен подход к организации поиска конфликтов между требованиями на основе онтологической модели и системы продукционных правил. Для выявления конфликтов текстовые требования преобразуются в экземпляры классов OWL онтологии. В требованиях выделяется три базовых элемента «субъект», «действие» и «объект». Эти элементы становятся классами онтологии. В онтологии также определяются отношения между экземплярами классов. По системе продукционных правил, между парами требований выполняется поиск конфликтов. Для построения системы продукционных правил и ее применения разработан программный продукт на языке Python. Для работы с онтологией использована среда Protégé 5.2. Также в работе применены Python-библиотека PySwip и среда разработки SWI-Prolog 7.6.4.

1. Введение

Неотъемлемым условием успешного производства программных продуктов является разработка качественной спецификации требований. Эта задача решается в процессе инженерии требований, в рамках которого неточные и неполные представления о том, какие сервисы для решения проблемных задач потенциальных пользователей должен предоставлять программный продукт, преобразовываются в формальную спецификацию требований. При этом должны быть учтены ограничения, при которых программный продукт должен быть реализован и должен функционировать. Согласно стандарту ISO/IEC/IEEE 29148:2011 *Системная и программная инженерия. Процессы жизненного цикла. Разработка требований* набор требований должен удовлетворять четырем критериям качества: полнота (complete), консистентность (consistent), осуществимость (affordable) и ограниченность (bounded). Первый критерий означает, что набор требований не должен нуждаться в дальнейшем уточнении. Второй критерий предполагает, что набор требований не содержит конфликтных требований, требования не дублируются, а для одного и того же элемента во всех требованиях используется один и тот же термин. Третий критерий означает, что для набора требований может быть разработано решение, удовлетворяющее требованиям с учетом ограничений жизненного цикла. Четвертый критерий подразумевает, что для предполагаемого решения набор требований поддерживает идентифицированную область, не выходя за пределы необходимого для удовлетворения потребностей пользователей [1].

При гибком подходе к разработке программных продуктов спецификация требований обычно представляет собой не единый физический документ, а некоторую логическую структуру, наполняемую требованиями к программному продукту. Созданию такой спецификаций обычно предшествует разработка документа «Видение программного продукта и границы проекта», в котором дается общее представление о разрабатываемом программном продукте в терминах ключевых потребностей и ограничений, при которых проект будет реализовываться. Далее начинаются работы по формированию бэклога продукта, который содержит приоритезированный список требований. Основной особенностью гибкого подхода является постоянная готовность к изменению требований, поэтому детально прорабатываются только те элементы бэклога продукта, которые выбраны для реализации в рамках текущей итерации. Остальные требования будут ждать своей очереди в бэклоге продукта, где могут быть добавлены, удалены или изменены в любой момент времени. В условиях изменчивости требований их согласование становится одной из наиболее сложных задач инженерии требований. Согласование требований включает в себя: обнаружение конфликтов и их разрешение. В этой связи представляется, что для сбора элементов бэклога продукта из указанных четырех характеристик качества на первый план выходит обеспечение консистентности набора элементов бэклога продукта. В рамках данной работы будет рассмотрена основная составляющая данной характеристики качества – бесконфликтность набора требований.

Статья организована следующим образом. В разделе 1 обосновывается актуальность темы исследования. В разделе 2 представлен обзор публикаций по теме исследования. В разделе 3 предлагается подход к преобразованию текстовых требований в онтологию. В разделе 4 приводится описание разработанного программного инструментария для построения продукционной модели поиска конфликтов между требованиями, представленными экземплярами онтологии. В разделе 5 делаются выводы о перспективах применения предложенного подхода.

2. Обзор публикаций по теме исследования

В работе [2] описывается прототип системы проектирования «Oz», который обеспечивает автоматизированные методы обнаружения конфликтов, генерации вариантов разрешений конфликтов и объяснения выбора решения. Экспериментальный инструмент «Oz» предназначен для разработки согласованных требований. Требования от разных стейкхолдеров представляются в виде сценариев. Прототип «Oz» включает язык требований, язык спецификации, планировщик спецификации и инструменты согласования требований. В работе [3] тех же авторов излагается подход к выявлению и разрешению конфликтов между требованиями в рамках парадигмы конфликт-ориентированного анализа требований CORA (Conflict-Oriented Requirements Analysis). Для описания структуры требований используется онтология требований, которая обеспечивает выражение требований и их взаимодействий. Конфликты между требованиями представлены в онтологии требований как типы взаимодействия. Взаимодействия между требованиями определяются непосредственно из описаний требований. Разрешение конфликтов основано на генерации альтернативных вариантов требований, устраняющих конфликт.

Авторами работы [4] указывается, что конфликт-анализ требований, производимый вручную, требует значительных усилий и подвержен ошибкам. Ими предлагается использовать семантические технологии в качестве основы для автоматизации анализа конфликтов между требованиями и внедрение автоматизированного онтологического подхода OntRep. В работе [4] выделяется три типа конфликтов: конфликты между требованиями и ограничениями, конфликты между требованиями и руководящими указаниями (например, по формализации требований), конфликты между требованиями. Конфликты между требованиями разделены на две группы: простые (между двумя требованиями) и сложные (между несколькими требованиями). Для реализации подхода к конфликт-анализу все термины, используемые в требованиях, должны быть сформулированы как термины глоссария в онтологии OntRep, а также требования должны формулироваться с использованием определённой структуры.

Инструмент OntRep состоит из двух основных компонентов. Первый компонент – это сборщик экземпляров, который принимает входные данные. К примеру, это могут быть тикеты требований из средства управления проектами и отслеживания ошибок в программном обеспечении Trac. Под тикетом в данной системе подразумевается виртуальная карточка с описанием требования (задачи) или ошибки, которую требуется исправить. Сборщик экземпляров инструмента OntRep анализирует содержимое тикета и присваивает категории требований (классов), определенные в онтологии (например, требование безопасности). Второй компонент предназначен для формирования отчета о конфликтах между требованиями [5].

В работе [6] описывается автоматизированный инструмент для выявления конфликтов EA-Analyzer. Данный инструмент предназначен для выявления конфликтов в аспектно-ориентированных требованиях, указанных в тексте на естественном языке. Требования записываются с помощью языка RDL (Requirements Description Language), который использует XML теги для аннотирования спецификации естественного языка. Главными элементами языка являются теги <subject>, <object> и <relationship>. Тег <subject> грамматически соответствует субъекту предложения, тег <object> – объекту, на который распространяется действие, а тег <relationship> определяет действие, выполняемое субъектом над объектом. При этом объектов в одном предложении может быть несколько. Классификация глаголов в RDL основывается на семантических категориях, предложенных в работах Р.Диксона [7]. Также в RDL используются категории степеней (например, показывающие различия между модельными глаголами), отражающие важность требования. Контаминация (слияние разнородных факторов) состоит из трех подэлементов: ограничение, базовое требование и результат. В инструменте EA-Analyzer проблема обнаружения конфликтов формулируется как проблема классификации, которая является хорошо изученной проблемой машинного обучения. Инструмент EA-Analyzer использует аннотированные требования на языке RDL, чтобы решить, имеются ли противоречия в требованиях [6].

В работе [8] предлагается набор правил для конфликт-анализа диаграмм классов. Процесс конфликт-анализа представляет собой цикл, включающий четыре шага: моделирование априорных знаний, моделирование новых требований, выявление конфликтов и разрешение конфликтов. Правила обрабатывают четыре варианта конфликтов: несоответствие, избыточность, переопределение и недостающие части.

В работе [9] рассматриваются вопросы определения кооперации и конфликтов между требованиями. Требования конфликтуют, если содержат противоречивые заявления об общих программных атрибутах, и находятся в состоянии кооперации, если взаимно дополняют заявления об атрибутах. Предлагаемый в работе [9] подход включает преобразование требования вручную в программные атрибуты, автоматическое определение конфликтов и кооперации между требованиями на основе их атрибутов, автоматическую трассировку зависимостей между требованиями.

В работе [10] предлагается подход, позволяющий сопоставить точки зрения стейкхолдеров на подсистемы одной информационной системы. Требования стейкхолдеров, записанные на естественном английском языке, обрабатываются с помощью синтаксического парсера Stanford Parser и инструмента морфологической разметки текстов TreeTagger. На основе анализа полученных семантико-синтаксических отношений по набору правил производится построение онтологии функциональных требований. Концепты полученных онтологий сопоставляются между собой с использованием внешних ресурсов, таких как WordNet и методов сопоставления строк, таких как метод N-gram. В результате сопоставления между концептами двух текстовых описаний требований определяются семантические отношения: концепты эквиваленты, один концепт в отношении другого выражает более общее или меньше понятие, концепты входят в одно подмножество понятий или нет.

В работе [11] предлагается подход к автоматическому построению онтологии из набора пользовательских историй. Для обработки текста на естественном английском языке используется библиотека spaCy, позволяющая производить синтаксический анализ предложений на основе дерева зависимостей, поиск именованных групп, выделение именных сущностей, а также разрешение кореференции.

Проведенный анализ публикаций по теме исследования позволяет заключить, что одним из перспективных направлений поиска конфликтов между требованиями являются подходы, предполагающие представление текстовых требований в форме экземпляров онтологии. Это может быть сделано вручную и в полуавтоматическом режиме. Автоматическое построение онтологий из текстов требований в настоящее время является крайне актуальной и все еще недостаточно хорошо освещенной темой. Рассмотренные исследования по построению онтологий из текстов требований не затрагивают непосредственно задачу определения конфликтов, однако применяемые в них идеи могут быть использованы как основа для извлечения из результатов автоматической обработки текстов требований экземпляров онтологии, структурированной для задачи поиска конфликтов между требованиями.

3. Преобразование текстовых требований в онтологию

В предложении, которое выражает требование, вне зависимости от применяемой техники записи требований (например, требования могут быть записаны в стиле стандарта IEEE 830 или с использованием пользовательских историй) можно выделить три главные части: субъект, действие и объект, на который направлено действие субъекта. Каждое требование необходимо представить в онтологии в виде экземпляров этих трех классов и определить отношения между парами экземпляров каждого из классов. А также создать для каждого требования экземпляр класса «требование» (Requirement), который будет связан с соответствующими требованию экземплярами классов «субъект», «действие» и «объект». Также необходимо определить возможные отношения между экземплярами классов. Следует заметить, что класс, элементы которого грамматически соответствуют субъектам предложений с требованиями, в онтологии может именоваться по-разному, например, «актор» или «функциональная роль пользователя», если рассматриваются только пользовательские требования, или «субъект». В рассматриваемом далее примере, иллюстрирующем предлагаемый подход, для обозначения субъекта будет использован термин «актор».

Сначала в среде Protégé 5.2 была реализована структура классов и заданы свойства объектов между ними. На рисунке 1 перечислены свойства объектов онтологии и их домены и диапазоны.

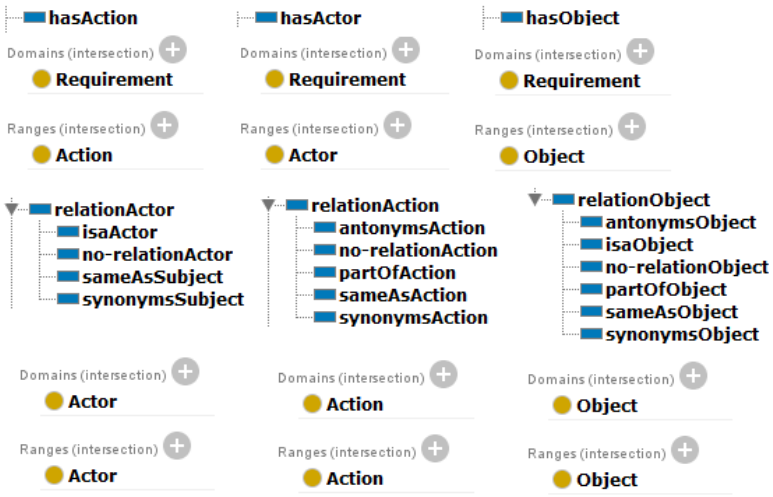

Рисунок 1. Свойства объектов онтологии, их домены и диапазоны.

Свойства объектов, соответствующие отношениям между классами «актор», «действие» и «объект», сгруппированы в классы свойств объектов relationActor, relationAction, relationObject. Свойства объектов sameAsActor, sameAsAction, sameAsObject устанавливаются между экземплярами соответствующих классов, если используется одно наименование для элементов двух требований или полное наименование в одном и сокращенное обозначение во втором. Например, экземпляры класса Actor со значениями «главный бухгалтер» и «глав.бух» будут

соединены свойством объектов sameAsActor. Свойства объектов isaActor и isaObject используются для установления иерархических отношений. Например, главный бухгалтер является бухгалтером. Свойства объектов antonymsAction и antonymsObject устанавливаются между экземплярами соответствующих классов, если те имеют противоположное значение. Например, «свой комментарий» и «чужой комментарий». Свойства объектов partOfAction и partOfObject устанавливаются между экземплярами соответствующих классов, если один является частью другого. Например, «комментарий на стене пользователя» является частью «стены пользователя». Свойства объектов synonymsActor, synonymsAction и synonymsObject устанавливаются, если значения соответствующих элементов требований имеют синонимичное значение. Во всех остальных случаях считается, что соответствующие элементы требований связаны свойствами объектов no-relationActor, no-relationAction и no-relation Object. Иными словами связи между ними не выявлены.

На рисунке 2 приведен пример с преобразованными в экземпляры онтологии элементами требований.

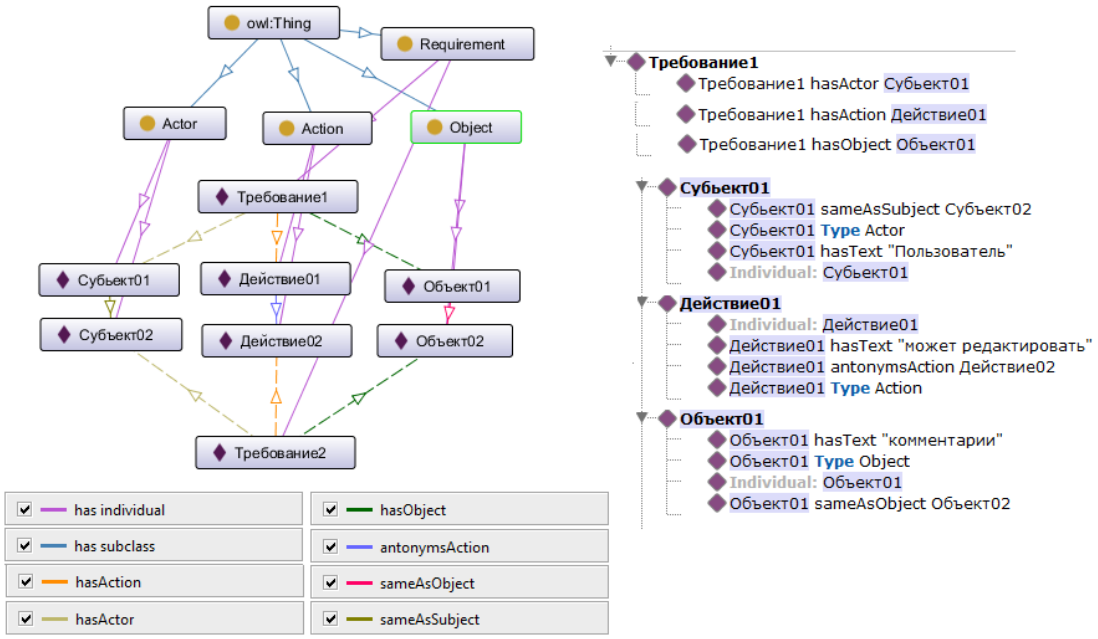

Рисунок 2. Фрагмент онтологии в редакторе Protégé.

Следует отметить, что преобразование текстовых требований в экземпляры онтологии можно проводить в полуавтоматическом режиме с применением инструментов автоматической обработки текстов. Для обработки текстов требований на русском языке может быть применен парсер UDPipe с моделью русского языка russian-ud-2.0-170801. Данный парсер выполняет разбор предложения и предоставляет результаты в формате CoNLL. Из результатов разбора предложения можно извлечь субъект, действие и объект при условии, что предложение с требованием сформулировано достаточно просто, т.е. не содержит множества речевых оборотов. Для обработки результатов разбора предложения могут быть применены продукционные правила. Например: Если X – одушевленное существительное, то это – субъект. Далее необходимо проверить, есть ли у данного существительного определения, чтобы извлечь наименование субъекта целиком.

Установление свойств объектов между экземплярами элементов требований также может быть произведено в полуавтоматическом режиме. Например, при помощи регулярных выражений может быть выполнен поиск модальных глаголов и отрицаний, а при помощи таких лингвистических онтологий, как WordNet, может быть выполнен поиск синонимов. Следует сказать, что поиск антонимов при помощи лингвистических онтологий для глаголов может привести к нежелательным результатам, поскольку, например, действия «добавить» и

«удалить» можно рассматривать, как антонимы, однако, с точки зрения решаемой задачи антонимичное значение будут иметь глагольные фразы «может удалить» и «не может удалить».

4. Подход к выявлению конфликтов между требованиями

С точки зрения решаемой в работе задачи, требования можно разделить на две группы. Первая группа – это требования, которые пересекаются по элементам «актор», «действие» и/или «объект». Вторая – требования, которые не пересекаются ни по одному из них. При этом пересекающиеся требования могут содержать конфликты. Исходя из этого, между парой требований может быть установлено одно из трех отношений: конфликтуют, пересекаются, не пересекаются. Для определения типа отношения между требованиями может быть составлена система продукционных правил вида:

Если Тип_отношения_между_экземлярами_класса (Актор1, Актор2)
И Тип_отношения_между_экземлярами_класса (Действие1, Действие2)
И Тип_отношения_между_экземлярами_класса (Объект1, Объект2)
Тогда Тип_отношения_между_требованиями (Требование1, Требование 2)

Для реализации предлагаемого подхода был разработано приложение с графическим интерфейсом на языке Python. Приложение обеспечивает выполнение следующих функций:

- построение системы продукционных правил для обработки отношений между экземплярами классов онтологии и сохранение их в виде модели в файл в формате XML;
- редактирование системы продукционных правил, ранее сохраненных в виде модели;
- применение правил модели к требованиям, представленным в виде экземпляров классов OWL онтологии.

Для реализации первой функции создан конструктор, который помогает пользователю в несколько «кликов» построить систему правил для анализа отношений между элементами требований. Конструктор позволяет выбрать файл с OWL онтологией, для которой строится модель. На следующих шагах работы с конструктором можно определить, какие классы онтологии соответствуют элементам «актор», «действие» и «объект», а также выбрать свойства объектов, которые будут использованы для анализа. Это позволяет конечному пользователю самостоятельно определять правила обработки имеющихся у него данных. На последнем шаге работы приложения выдаются сгенерированные комбинации антецедентов для продукционных правил. Пользователю необходимо определить консеквенты правил, выбрав тип отношения между требованиями для каждого правила. Полуавтоматическая генерация правил не позволяет пользователю пропустить какое-либо правило, поскольку покрывает все возможные сочетания для антецедентов правил. На рисунке 3 представлен последний шаг построения модели.

Далее рассмотрим два примера для правил, полученных по описанной выше онтологии.

Пример 1:

Правило № 62:

IF sameAsActor(Actor1, Actor2) and antonymsAction(Action1, Action2) and isaObject(Object1, Object2) Then conflict(Requirement1, Requirement2)

Требования:

R1: Пользователь не может редактировать чужие комментарии.

R2: Пользователь может редактировать комментарии.

Подстановка данных в правило:

IF sameAsActor(Пользователь, Пользователь) and antonymsAction(не может редактировать, может редактировать) and isaObject(чужие комментарии, комментарии) Then conflict(Requirement1, Requirement2)

Пример 2:

Правило № 65:

IF sameAsActor(Actor1, Actor2) and antonymsAction(Action1, Action2) and sameAsObject(Object1, Object2) Then conflict(Requirement1, Requirement2)

Требования:

R1: Пользователь может удалять комментарии.

R2: Пользователь может только просматривать комментарии.

Подстановка данных в правило:

IF sameAsActor(Пользователь, Пользователь) and antonymsAction(может удалять, может только просматривать) and sameAsObject(комментарии, комментарии) Then conflict(Requirement1, Requirement2)

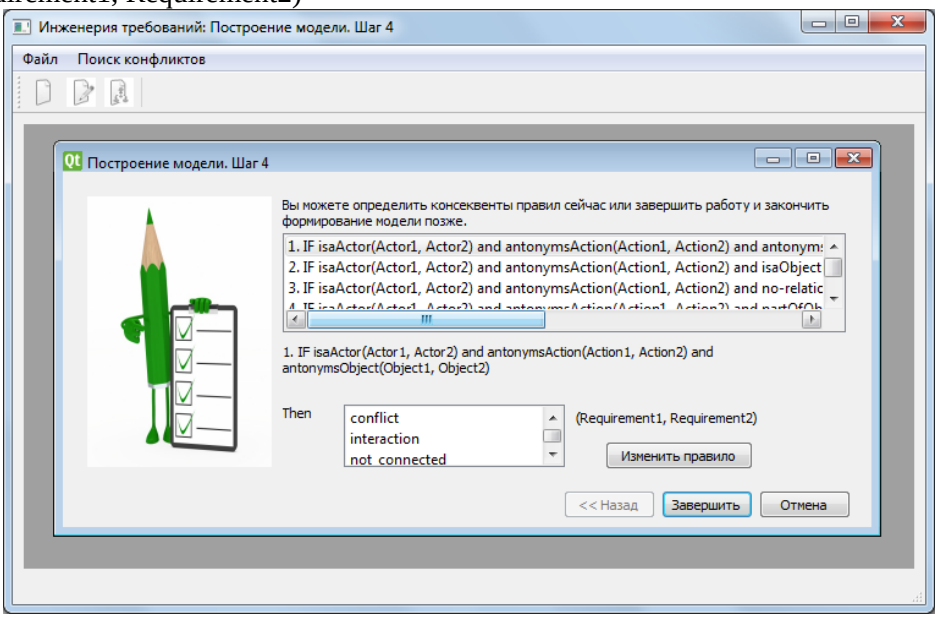

Рисунок 3. Экранная форма конструктора.

В первом примере пользователь с одной стороны может редактировать комментарии, а с другой стороны у него есть запрет на редактирование чужих комментариев. Поскольку чужие комментарии являются частью комментариев, получается противоречие. Чтобы устранить полученное противоречие, требование R2 нужно переформулировать: «Пользователь может редактировать свои комментарии». Во втором примере мы наблюдаем противоречие в действиях пользователя над одним объектом «комментарии», которое также требует уточнения и исправления, поскольку непонятно, о каких комментариях идет речь: только о комментариях, созданных самим пользователем, или обо всех комментариях к объектам, созданным пользователем в системе. И на первый взгляд «понятные недочеты», могут привести к реализации функциональности, не соответствующей изначальной идее.

В ходе анализа требований данные об экземплярах классов и отношениях между ними импортируются в базу фактов на языке Prolog, правила модели – в правила логического вывода на языке Prolog и далее выполняется поиск решения. Результаты анализа требований сохраняются в текстовый файл. Для реализации указанной функциональности использованы Python-библиотека PySwip и среда разработки SWI-Prolog 7.6.4.

5. Заключение

В работе был предложен подход к выявлению конфликтов в спецификации требований на основе онтологической модели и системы продукционных правил. Для организации обработки текстов требований было предложено извлекать из текстовых требований субъект, действие и объект, на который направлено действие, и сохранять их в форме экземпляров онтологии. А также устанавливать между экземплярами классов, принадлежащими паре требований, отношения, определенные в онтологии в виде свойств объектов. На основании этих свойств объектов строить систему продукционных правил для определения типа отношения между парой требований. Предлагаемый подход и разработанный инструментарий позволит облегчить процесс выявления противоречивых требований, поскольку выполняет сопоставление требований в полуавтоматическом режиме, что способствует обнаружению незначительных на первый взгляд неточностей в формулировке требований. Следует отметить, что применение

данного подхода требует записи текстовых требований в виде несложных предложений, которые могут быть частично обработаны автоматически, поскольку полностью ручное преобразование текстовых требований в экземпляры онтологии достаточно трудозатратно.

6. Литература

[1] ISO/IEC/IEEE 29148:2011(E) / Systems and software engineering – life cycle processes – requirements engineering //Version 1.0, 2011-12-01.
[2] Robinson, W.N. Supporting the Negotiation Life Cycle / W.N. Robinson, V. Volkov // Communications of the ACM. – 1998. – Vol. 41. – P. 95-102.
[3] Robinson, W.N. Requirement Conflict Restructuring / W.N. Robinson, V. Volkov // GSU CIS Working. Georgia State University, Atlanta, 1999.
[4] Heindl, M. Automating the detection of complex semantic conflicts between software requirements with OntRep. An empirical study on requirements conflict analysis with semantic technology / M. Heindl, T. Moser, D. Winkler, S. Biffl // Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE), 2011. – P. 729-735.
[5] Moser, T. Requirements Management with Semantic Technology: An Empirical Study on Automated Requirements Categorization and Conflict Analysis / T. Moser, D. Winkler, M. Heindl, S. Biffl // 23rd International Conference on Advanced Information Systems Engineering (CAiSE), 2011. – P. 3-17.
[6] Sardinha, A. Conflict identification with EA-Analyzer / A. Sardinha, R. Chitchyan, J. Araújo, A. Moreira, A. Rashid // Aspect-oriented requirements engineering. – Berlin: Springer, 2013. – P. 209-224.
[7] Chitchyan, R. Semantics-Based Composition for Aspect-Oriented Requirements Engineering / R. Chitchyan, A. Rashid, P. Rayson, R. Waters // Proceedings of the 6th international conference on Aspect-oriented software development (ACM), 2007. – P. 36-48.
[8] Liu C.-L. Ontology-Based Requirement Conflicts Analysis in Class Diagrams / C.-L. Liu, H.-H. Huang // Proceedings of the World Congress on Engineering. – London, U.K, 2015. – Vol I.
[9] Egyed, A. Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help / A. Egyed, P. Grunbacher // IEEE Software. – 2004. – Vol. 21(6). – P. 50-58.
[10] Assawamekin, N. Ontology-based multiperspective requirements traceability framework / N. Assawamekin, T. Sunetnanta, C. Pluempitiwiriyawej // Knowledge and Information Systems. – 2010. – Vol. 25(3). – P. 493-522.
[11] Robeer, M. Automated Extraction of Conceptual Models from User Stories via NLP / M. Robeer, G. Lucassen, J.M.E.M. van Der Werf, F. Dalpiaz, S. Brinkkemper // 24th International Requirements Engineering (RE) Conference, 2016. – P. 196-205.

Благодарности

Работа поддержана грантом Министерства образования и науки РФ в рамках проектной части государственного задания, проект № 2.2327.2017/4.6 «Интеграция моделей представления знаний на основе интеллектуального анализа больших данных для поддержки принятия решений в области программной инженерии».

The detection of conflicts in the requirements specification based on an ontological model and a production rule system

M.Sh. Murtazina[1], T.V. Avdeenko[1]

[1]Novosibirsk State Technical University, Karl Marx Avenue 20, Novosibirsk, Russia, 630073

Abstract. The paper presents an approach to organizing the detection of conflicts between requirements based on an ontological model and a system of production rules. Textual requirements are converted to instances of OWL ontologies to identify conflicts. There are three basic elements “subject”, “action” and “object” in the requirements. These elements become classes of ontology. The ontology also defines relations between instances of classes. A production rules system is used to detect conflicts between pairs of requirements. A software product in the Python language has been developed for building the production rules system and its use. Protégé 5.2 is used to work with ontology. Also Python library PySwip and development environment SWI-Prolog 7.6.4 are used in the work.