Онтологический подход к интеллектуальной поддержке инженерии требований при гибкой разработке программных продуктов

М.Ш. Муртазина[1], Т.В. Авдеенко[1]

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

Аннотация. В статье представлен онтологический подход к интеллектуальной поддержке инженерии требований при гибкой разработке программных продуктов. Предлагаемая вниманию онтологическая модель объединяет онтологию информационной поддержки инженерии требований и онтологию предметной области разрабатываемого программного продукта. В рамках предлагаемого подхода требования анализируются как логические утверждения о предметной области разрабатываемого программного продукта. Также предлагается метод извлечения и анализа требований из формулировок на естественном языке. Дается описание разработанного прототипа системы поддержки принятия решений. Представленный подход предполагает совместное использование нескольких OWL-файлов, которые позволяют представить знания о проекте, предметной области проекта и о семантических отношениях между ключевыми элементами предложений с требованиями (акторах, действиях и объектах). Для построения и тестирования онтологий использована среда Protégé 5.2. Система поддержки принятия решений написана на языке Python.

1. Введение

Отличительной особенностью гибкого подхода к разработке программных продуктов является ориентированность на быструю адаптацию требований к изменчивым условиям бизнес-среды. При этом командой проекта учитывается фактор неопределенности, ее работа детально планируется на небольшие отрезки времени (спринты), исходя из известных и ясных на настоящий момент требований. При гибком подходе не составляются детальные долгосрочные и обычно не реализуемые планы проекта и спецификации требований. Спецификация требований к программному продукту при гибком подходе представляет собой некоторую логическую структуру, наполняемую требованиями на протяжении всего процесса разработки программного продукта. Гибкий подход к разработке программных продуктов поощряет создание минимального количества документации, поэтому для документирования требований применяются специфичные и отличные от традиционного подхода артефакты требований. Сначала разрабатывается документ «Видение программного продукта и границы проекта», который определяет представление стейкхолдеров о разрабатываемом программном продукте и ограничения проекта. В данный документ может включаться дерево функциональных возможностей программного продукта. Под функциональной возможностью продукта

понимается единица функциональности программного продукта, которая удовлетворяет требованиям.

На сегодняшний день при применении гибкого подхода наиболее часто для организации работы участников проекта используется процессный фреймворк Scrum, согласно которому все требования по проекту записываются в бэклог продукта, который представляет собой приоритизированный список имеющихся на данный момент требований (задач). Бэклог продукта никогда не бывает полным. Перед тем как та или иная функциональная возможность программного продукта будет запланирована в спринт, для нее должны быть написаны небольшие пользовательские истории, разработан базовый сценарий поведения, оценены усилия на реализацию, определены зависимости от других историй, выявлены риски и определен приоритет. Данная работа выполняется владельцем продукта совместно с командой разработки в рамках груминга бэклога продукта.

По своей значимости, трудозатратности и сложности задача извлечения и анализа требований сравнима с задачей извлечения знаний экспертов для создания интеллектуальных систем. Поэтому в последние годы для решения данной задачи активно применяются методы инженерии знаний. В частности, используются онтологические модели представления знаний, как сами по себе, так и в сочетании с другими моделями, например, продукционными и логическими. В данной работе представляется онтологический подход к интеллектуальной поддержке инженерии требований при гибкой разработке программных продуктов. Возможности онтологической модели представления знаний дополняются возможностями рассуждения на основе продукционно-логических моделей.

Статья организована следующим образом. В разделе 1 обосновывается актуальность темы исследования. В разделе 2 дается обзор публикаций по теме исследования. В разделе 3 предлагается система онтологий для целей инженерии требований при гибком подходе к управлению проектом. В разделе 4 представляется подход к анализу требований. В разделе 5 приводится описание разработанного программного инструментария. В разделе 6 делаются выводы по проделанной работе.

2. Обзор публикаций по теме исследования

В работе [1] показано, как, используя DL-аксиомы, можно описать правила проверки полноты и консистентности требований, детально описан подход к решению задачи инженерии требований к программному обеспечению с указанием правил проверки, представлен подход к автоматизации процесса валидации и измерению знаний о требованиях.

В работе [2] анализируются возможности онтологий для поддержки процесса разработки программного обеспечения, выделяется два типа онтологий, которые можно использовать для представления знаний о предметной области разрабатываемого программного обеспечения: онтология предметной области программного продукта и онтология функциональных возможностей программного продукта. В работе [3] исследуется подход к созданию онтологий предметной области для сбора требований. Подход к разработке требований к программному обеспечению, основанный на онтологии, предложен в [4]. Извлечение онтологий из спецификации требований и проверка согласованности и/или завершения спецификации требований на базе методов обработки естественного языка дается в работе [5].

Основываясь на работах [6, 7, 8] можно заключить, что, несмотря на большое количество работ, посвященных вопросам применения онтологий в области управления проектами по разработке программного обеспечения, а также в области инженерии требований, вопрос интеграции таких онтологий остается недостаточно проработанным. В этой связи представляет научный интерес разработка механизма принятия решений на основе онтологических моделей в области инженерии требований в контексте управления проектами, а также механизма анализа требований на основе онтологических моделей, лингвистических ресурсов и методов автоматической обработки русскоязычного текста.

3. Система онтологий в области инженерии требований

Целью представления знаний является организация необходимой для решения предметной задачи информации в форме, применимой для принятия решений, планирования, анализа и вывода суждений, осуществляемых системами поддержки принятия решений. В логических моделях знания представляются как совокупность фактов и утверждений, выраженных в виде формул в некоторой логике. Популярным формализмом для описания онтологий являются подмножества дескриптивной логики [9]. Для описания онтология в настоящее время чаще всего применяется онтологический язык OWL (Web Ontology Language), поддерживающий стандарты семантической разметки и возможность написания аксиом дескрипционной логики (DL). Логические машины вывода для OWL-онтологий обеспечивают проверку консистентности модели, выполняют классификацию экземпляров классов в соответствии с DL-аксиомами, а также устанавливают неявных отношений между экземплярами классов по заданным характеристикам свойств-отношений.

Построение онтологии для информационной поддержки процесса инженерии требований при гибком управлении проектом будем производить, исходя из следующего понимания термина «требование»: это утверждение о некотором свойстве программного продукта. Когда требование задокументировано, возникает артефакт требования. Знания о процессе управления гибким проектом будут основываться на подходе, применяемом во фреймворке Scrum. Знания о характеристиках качества требований – на основе модели качества программного продукта ГОСТ Р ИСО /МЭК 25010-2015. В разрабатываемой онтологии учитывается техника написания пользовательских историй и сценарии на языке Gherkin. Иерархическая структура основных классов онтологии для информационной поддержки процесса инженерии требований при гибком управлении проектом $Onto^{RE}$ и онтологии предметной области программного продукта $Onto^{Dom}$ приведена на рисунках 1 и 2 соответственно.

Описание терминологии предметной области программного продукта включает в себя понятия предметной области и слова, из которых состоят понятия предметной области. В рамках данной работы терминологию предметной области предлагается представить с учетом некоторых свойств-отношений и классов, характерных для лексических онтологий. Это позволит в дальнейшем при анализе данных сопоставлять данные онтологии предметной области с данными внешних лексических онтологий. Онтология $Onto^{Dom}$ включает в себя терминологию предметной области, модель ролей пользователей и модель дерева функциональных возможностей программного продукта. Знания онтологии $Onto^{Dom}$ могут быть использованы при написании текстов требований, в частности, пользовательских историй. В классе «Концепт предметной области программного продукта» (DomainConcept) выделяются два основных подкласса, нужных для описания и проверки корректности требований: сущность (Entity) и отношение-действие (RelationAction). Сущности в свою очередь подразделяются на акторов (Actor) и объекты (Object). Актор – это сущность, обладающая некоторым поведением. В роли актора может выступать пользователь или программная система (подсистема). Объект – это сущность, на которую направлены действия, совершаемые актором. Актор и объект связываются между собой отношениями-действиями. Отношение-действие в свою очередь должно включать смысловой глагол, а также может включать вспомогательный глагол и частицу «не». Например: «не может удалять». Глагол «удалять» в данной фразе является смысловым. Смысловые глаголы предметной области будут выделяться в отдельный класс «Операция-действие» (OperationAction).

К классам верхнего уровня онтологии $Onto^{RE}$ отнесены следующие концепты предметной области: Критерий готовности (DefinitionOfDone), Критерий подготовности (DefinitionOfReady), Груминг-встреча (Grooming), Вид характеристики качества требований (QualityCharacteristic), Группа характеристик качества (QualityCharacteristicUserNeed), Значения порядковых шкал (Rank), Артефакт требований (RequirementsArtefact), Риск (Risk), Класс рисков (RiskType), Элемент фреймворка Scrum (ScrumFrameworkElement), Группа стейкхолдера (StakeholderType), Статус задачи (Status), Структурный элемент (StructuralElement), Задача по элементу бэклога продукта (Task), Цель тестирования (TestObjective), Вид теста (TestType).

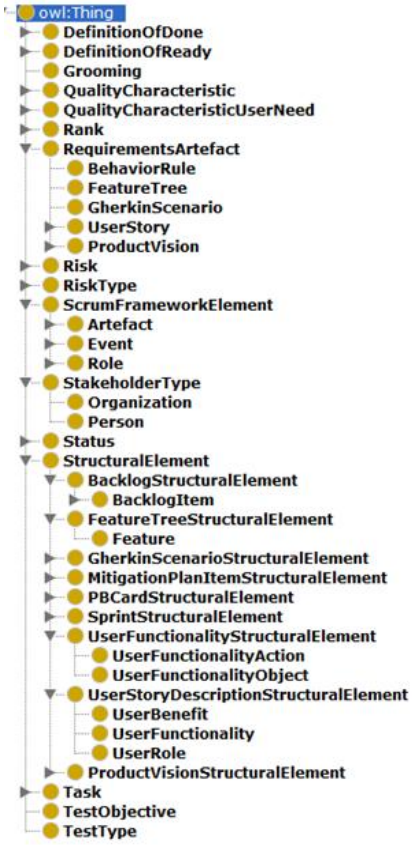

Рисунок 1. Структура классов *Onto*^{RE}.

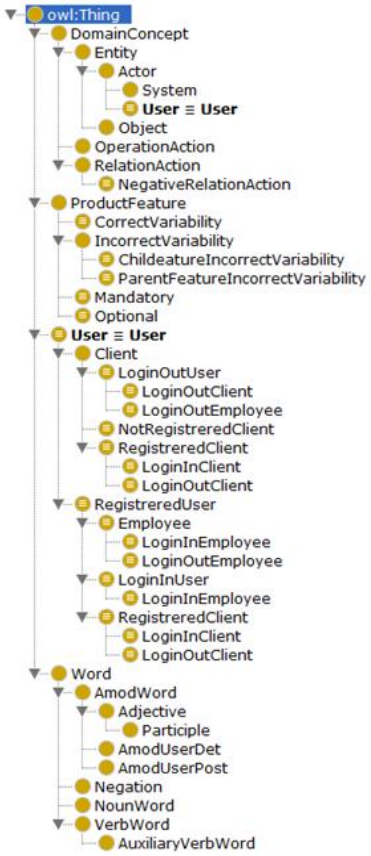

Рисунок 2. Структура классов *Onto*^{Dom}.

Класс «Артефакт требований» включает артефакты документирования требований к программному продукту. К ним относятся видение продукта, дерево функциональных возможностей, пользовательские истории, сценарии поведения (в нотации Gherkin), правила поведения. Класс «Структурный элемент» включает семь подклассов: структурный элемент видения продукта, структурный элемент дерева функциональных возможностей, структурный элемент пользовательской истории, структурный элемент функциональности пользовательской истории, структурный элемент сценария поведения, структурный элемент бэклога продукта, структурный элемент бэклога спринта.

Модель онтологии для информационной поддержки процесса инженерии требований при гибком управлении проектом реализована в виде набора owl-файлов: OntoRE.owl, agile.owl, user_model.owl, feature_tree.owl. Первый файл позволяет объединить остальные и манипулировать ими как единым файлом. Второй предназначен для представления знаний об артефактах требований и процессах работы с ними в рамках проекта. Третий – о модели ролей пред-метной области, а четвертый – для построения дерева функциональных возможностей для предметной области программного продукта. Третий и четвертый файл также включены во вторую модель. Модель онтологии предметной области реализована в виде следующих owl-файлов: OntoDom.owl, domain_concept.owl, user_model.owl, fea-ture_tree.owl. Первый файл позволяет объединить остальные и манипулировать ими как единым файлом. Второй предназначен для представления знаний о концептах предметной области.

При проверке артефактов требований, представленных в онтологии Onto^{RE} , применяется онтология Onto^{Dom}. В свою очередь онтология Onto^{Dom} пополняется новыми концептами из артефактов требований онтологии Onto^{RE}. Артефакт требований «Дерево функциональных возможностей» состоит из элементов класса *ProductFeature*. Данный класс эквивалентен классу *Feature* из онтологии Onto^{RE}. Такое представление данных позволяет синхронизировать процесс управления знаниями предметной области с процессом управления требованиями.

Артефакт требований «Пользовательская история» включает в структуре три элемента:

```
Как <роль пользователя X > ,
Я хочу <некая функциональность Y>,
Чтобы <получаемая выгода Z>.
```

Данным элементам соответствуют классы *UserRole*, *UserFunctionality* и *User-Benefit*. Функциональность, заданная в пользовательской истории, представлена действием (класс *UserFunctionalityAction*) и объектом, над которым совершается действие (класс *UserFunctionalityObject*). Экземпляры класса *UserRole* выбираются из модели ролей, которая является частью модели предметной области программного продукта. Элементы классов *UserFunctionalityAction* и *UserFunctionalityObject* онтологии Onto^{RE} соотносятся с элементами классов *RelationAction* и *Object* онтологии Onto^{Dom}. Данный подход позволяет контролировать использование терминологии предметной области и при необходимости пополнять ее новыми концептами. Концепты предметной области и слова, из которых состоят концепты, накапливаются в классах *DomainConcept* и *Word*.

Логические аксиомы онтологии Onto^{RE} предназначены для установления отношений между концептами предметной области. Они позволяют выявлять неполноту описания артефактов требований и находить ошибки некорректного описания артефактов требований. Рассмотрим принципы определения аксиом. Например, пользовательская история не может быть включена в план спринта, если она не соответствует критериям подготовленности. Это означает, что она должна быть рассмотрена во время груминга и не иметь неразрешенных вопросов, для нее должны быть определены данные для оценки приоритета и критерии приемки. Указанное утверждение без учета ограничения на понятность пользовательской истории команде разработки задано аксиомой эквивалентности, приведенной на рисунке 3.

```
(hasAcceptanceCriteria some FunctionalAcceptanceCriteria) and (hasPBIRisk some ((hasRiskValueI
some xsd:integer[>= 0 , <= 10]) and (hasRiskValueP some xsd:integer[>= 0 , <= 10]))) and
(isAanalyzedAtGrooming some Grooming) and (hasBusinessValue some xsd:integer[>= 0 , <= 100])
and (hasEstimationValue some xsd:integer[>= 0 , <= 20])
```

Рисунок 3. Аксиома эквивалентности для отнесения истории к классу DORUserStory.

Будем считать, что пользовательская история понятна команде, если при ее рассмотрении во время груминга не осталось неразрешенных вопросов. Указанное утверждение записано в онтологии аксиомой эквивалентности, приведенной на рисунке 4.

```
hasPBIProblem some (isResolution value false)
```

Рисунок 4. Аксиома эквивалентности для отнесения истории к классу ProblemUserStory.

Ввиду того, что логические машины вывода для OWL-онтологий основываются на концепции OWR (Open World Reasoning), а для вышеуказанного определения необходимо применение отрицания в рамках концепции NAF (Negation As Failure), для решения задачи в при создании системы поддержки принятия решений использован подход, включающий применение аксиом эквивалентности на уровне OWL DL и обработку полученных по ним результатов средствами SWI-Prolog с использованием встроенного предиката not, реализующего концепцию отрицания через неуспех.

4. Анализ требований на основе онтологической и продукционной моделей

Для анализа текстовые требования преобразуются в экземпляры классов OWL-онтологии *Onto^{ReqRelation}*, которые затем сопоставляются с уже проверенными знаниями из онтологии Onto^{Dom} и внешних лингвистических ресурсов. Онтология Onto^{ReqRelation} содержит четыре класса верхнего уровня, соответствующие концептам: Требование, Актор, Действие-отношение, Объект. Структура онтологии и ее свойства-отношения подробно описаны в работе [10].

Для извлечения экземпляров классов онтологии требований используется инструмент UDPipe, применяющий при разметке текста аннотации «Универсальные зависимости». Список частей речей для всех языковых моделей является фиксированным и состоит из 17 тегов. Лексические и грамматические признаки слов задаются при помощи тегов морфологических признаков. Для аннотаций синтаксических зависимостей между словами используется набор отношений для кросс-языкового синтаксического разбора, который ограничен 37 тегами. Результаты обработки текста записываются в виде текстовых файлов в формате CoNLL-U.

Множество разработанных продукционных правил, применяемых для обработки результатов разбора предложения, можно разделить по назначению на два подмножества. Первое – правила поиска главного слова в концепте предметной области. Второе – правила выявления всех слов, входящих в концепт. Рассмотрим некоторые из разработанных правил. Для определения главного слова в наименовании сущности используется следующее правило:

Если *X* – это существительное **И** X связан с глаголом Z отношением зависимости $Relation_{XZ} \in$ { nsubj, obj, advcl, parataxis} И Z – это корень дерева зависимостей **Тогда** *X* – это главное слово в наименовании сущности.

Правило для определения смыслового глагола:

Если *Y* – это глагол **И** *Y* имеет морфологический признак VerbForm ∈ {Inf, Fin} **И** *Y* не является служебным глаголом **И** (*Y* связан с корнем дерева зависимостей **ИЛИ** Y – это корень дерева зависимостей) **Тогда** *Y* – это операция-действие.

После выделения главных слов в наименованиях сущностей осуществляется определение связанных с ними слов, которые могут составлять наименование сущности. Например:

Если X – это главное слово в наименовании сущности **И** X связан с прилагательным Z отношением зависимости $Relation_{XZ}$ **Тогда** Z || X – это наименование сущности.

База правил представляет собой правила перебирающие варианты отношений между словами, которые составляют концепты предметной области. Предлагаемый метод извлечения концептов может быть использован при извлечении акторов, объектов и отношений-действий из текстовых формулировок пользовательских историй и критериев приемки в форме простых утвердительных предложений. В общем виде метод извлечения требований выглядит следующим образом. На первом этапе простой текстовый файл, содержащий набор требований, передается для обработки анализатору Udpipe. На втором этапе из полученных деревьев зависимостей в файле формата CoNLL-U в результате применения разработанных правил извлечения выделяются актора, отношения-действия и объекты. Далее производится запись результатов, извлеченных концептов в онтологию.

Анализ консистентности набора требований выполняется в два этапа. Первый – это установление отношений между сущностями и действиями-отношениями в онтологии требований. Второй – применение правил для анализа отношений между требованиями. Концепты онтологии требований могут быть получены путем извлечения из текстовых требований. Далее устанавливаются свойства-отношения между экземплярами. Для этого используются онтология предметной области и внешние русскоязычные WordNet-подобные ресурсы (например, YARN). Далее к данным онтологии требований применяются продукционно-логические правила. Общая схема для генерации правил:

```
Если Тип_отношения_между_акторами(Актор1, Актор2)
И Тип_отношения_между_действиями(Действие1, Действие2)
И Тип_отношения_между_объектами(Объект1, Объект2)
Тогда Тип_отношения_между_требованиями(Требование1, Требование2)
```

Для исключения пропуска правил разработан механизм полуавтоматического их формирования. По свойствам-отношениям между классами «актор», «отношение-действие» и «объект» автоматически генерируются все возможные комбинации антецедентов правил. Консеквенты правил определяются вручную. В качестве консеквента правила выбирается одно из четырех отношений: требования конфликтуют (свойство-отношение «isConflict»), требования дублируются (свойство-отношение «isDuplicate»), требования взаимосвязаны (свойство-отношение «isInterconnect»), связь между требованиями не установлена (свойство-отношение «isNotInterconnect»).

В процессе анализа требований данные об экземплярах классов, отношениях между ними, правила модели преобразуются во множество аксиом логики предикатов и импортируются в систему SWI-Prolog. Далее выполняется поиск решения с использованием механизма обратного логического вывода, встроенного в Prolog.

5. Система поддержки принятия решений

Прототип системы поддержки принятия решений (СППР) обеспечивает сбор и анализ данных в соответствии с принятыми при гибком подходе к разработке программных продуктов техниками работы с требованиями. СППР включает в себя следующие функциональные модули: «Справочники», «Видение продукта», «Дерево функциональных возможностей», «Бэклог продукта», «Бэклог спринта», «Анализ требований», «Обработка текстовых требований». Проект по разработке программного продукта начинается с создания документа «Видение продукта». Для обеспечения разработки данного документа используется функциональный модуль «Видение продукта». Следующим шагом работ является создание такого артефакта требований как «Дерево функциональных возможностей». Далее начинаются работы по формированию бэклога продукта. Его основным элементом являются карточки пользовательских историй. При заполнении карточки пользовательской истории вносятся: формулировка истории, функциональные и нефункциональные критерии приемки истории, зависимости от других элементов бэклога продукта, а также экспертные оценки бизнес-ценности, уровня риска, оценки усилий, на основании которой определяется уровень приоритета. Архитектура СППР приведена на рисунке 6.

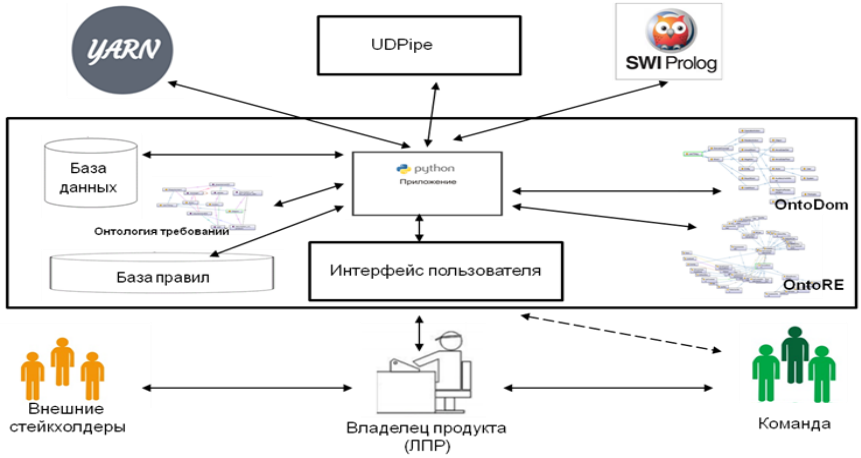

Рисунок 5. Архитектура СППР.

Для реализации логического вывода по аксиомам онтологии используется машина логического вывода HermiT, которая входит в Python-библиотеки Owlready2. Данная Python-библиотека позволяет загружать OWL-онтологии как объекты Python, изменять их, сохранять и выполнять логический вывод. Смежными системами, с которыми может взаимодействовать СППР, являются: среда SWI-Prolog и анализатор UDPipe.

6. Заключение

В работе был предложен подход к интеллектуальной поддержке инженерии требований при гибкой разработке программных продуктов. Представлена система онтологических моделей, интегрирующая онтологию информационной поддержки процесса инженерии требований при гибком управлении проектом и онтологию предметной области разрабатываемого программного продукта. Данная интеграция позволяет проверять требования на соответствие формальным критериям модели качества и согласованность с концептуальной моделью предметной области. Представлен подход к анализу требований на основе онтологических моделей, лингвистических ресурсов и методов автоматической обработки русскоязычного текста.

7. Литература

[1] Siegemund, K. Contributions To Ontology-Driven Requirements Engineering : dissertation to obtain the academic degree Doctoral engineer (Dr.-Ing.) / K. Siegemund – Dresden: Technischen Universität Dresden, 2014. – 236 p.

[2] Bhatia, M.P.S. Ontologies for Software Engineering: Past, Present and Future / M.P.S. Bhatia, A. Kumar, R. Beniwal // Indian Journal of Science and Technology. – 2016. – Vol. 9(9).
[3] Omoronyia, I. A Domain Ontology Building Process for Guiding Requirements Elicitation / I. Omoronyia, G. Sindre, T. Stålhane, S. Biffl, T. Moser, W. Sunindyo // Requirements Engineering: Foundation for Software Quality: 16th International Working Conference, REFSQ, 2010.– P. 188-202.
[4] Makrickienė, N. Ontology and Enterprise Modelling Driven Software Requirements Development Approach / N. Makrickienė, S. Gudas, A. Lopata // Baltic Journal of Modern Computing. – 2019. – № 7(2). – P. 190-210.
[5] Arellano, A. Natural Language Processing of Textual Requirements / A. Arellano, E. Carney, M.A. Austin // ICONS: The Tenth International Conference on Systems – Barcelona, Spain, 2015. – P. 93-97.
[6] Fitsilis, P. Ontologies for Software Project Management: A Review / P. Fitsilis, V. Gerogiannis, L. Anthopoulos // Journal of Software Engineering and Applications. – 2014. – №7. – P. 1096-1110.
[7] Lin, Y. Scrum Conceptualization Using K-CRIO Ontology / Y. Lin, V. Hilaire, N. Gaud, A. Koukam // Data Driven Process Discovery and Analysis – Springer Berlin Heidelberg, Berlin, 2012. – P. 189-211.
[8] Werewka, J. Integration of classical and agile project management methodologies based on ontological models / J. Werewka, P.S zwed, G. Rogus // Production engineering in making. AGH University of Science and Technology. – Press, Kraków, 2010.
[9] Загорулько, Ю.А. Онтологии и их практическое применение в системах, основанных на знаниях / Ю.А. Загорулько, Г.Б. Загорулько // Материалы Всероссийской конференции с международным участием «Знания – Онтологии – Теории» – Новосибирск: Институт математики им. С.Л. Соболева СО РАН. – 2011. –Т. 1. – С. 132-141.
[10] Муртазина, М.Ш. Выявление конфликтов в спецификации требований на основе онтологической модели и системы продукционных правил / М.Ш. Муртазина, Т.В. Авдеенко // Информационные технологии и нанотехнологии (ИТНТ): V междунар. конф. и молодеж. шк. – Самара : Изд-во Новая техника, 2019. – С. 592-600.

The ontology-driven approach to intelligent support of requirements engineering in agile software development

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

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

Abstract. The work presents the ontological approach to the intelligent support of requirements engineering with the agile software development. An ontological model is proposed that combines ontology for information support of requirements engineering and ontology of the application domain. The requirements are analyzed as logical statements about the subject area of the software product being developed. A method is also proposed for extracting and analyzing requirements from natural language formulations. The description of the developed prototype of the decision support system is given. The presented approach involves the joint use of several OWL files that enables presenting knowledge about the project, the application domain and the semantic relations between the key elements of the proposals with the requirements (actors, actions and objects). Protégé 5.2 is used to work with ontology. The decision support system is written in Python.