Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 150 (2019) 628–635

Procedia Computer Science

www.elsevier.com/locate/procedia

13th International Symposium "Intelligent Systems" (INTELS'18)

An Ontology-based Approach to Support for Requirements Traceability in Agile Development

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

Novosibirsk State Technical University, 20 Karla Marksa ave., Novosibirsk 630073, Russia

Abstract

The paper proposes an ontology-based approach to support for requirements traceability in Agile development. The task of supporting the requirements traceability in a software development project is considered as a part of requirements engineering process. A brief overview of the benefits of requirements traceability usage is given. The expediency of using the ontology-based approach to support for requirements engineering, requirements traceability in particular, is shown through the analysis of scientific publications. OWL ontology is chosen to support the requirements engineering process. The ontology is implemented in the Protégé environment. The developed ontology takes into account particular qualities of working with the requirements in Agile projects, accumulates knowledge about the requirements types and requirements artefacts, enables tracing the relations between them.

Keywords: ontology; requirements traceability; agile development.

1. Introduction

In order to ensure qualitative management of the requirements in the software engineering projects, the set of requirements must satisfy the traceability property. Traceability is understood as the ability to trace the life cycle of a requirement, both in the direct and in the opposite direction. The support for requirements traceability enables saving time, reducing the cost of work and improving the quality of the result. This is confirmed by research

[*] Corresponding author.
E-mail address: murtazina@corp.nstu.ru

conducted in recent years. In particular, 71 participants took part in one such study. Participants performed real tasks (bug fixing, adding new features, etc.) for two software development projects. One half of the tasks was performed with the support of requirements traceability, the other half – without it. The experiment showed that in cases where support for requirements traceability is implemented participants coped with tasks 24% faster and create 50% more correct software solutions on average [1].

One of the main features in the agile development of the software products is the willingness to constantly change the requirements. Each change request affects the current configuration of the software requirements, so the support for requirements traceability in agile development becomes particularly important.

The requirements are traced through the establishment of the relations between the requirements and the artefacts of the development project, primarily, the requirements artefacts. By the requirements artefacts we mean all the information about the requirements that is created, changed, and used in the process of implementing the requirements. In agile development, the requirements artefacts are business goals, sources of requirements (for example, legislative norms), the acceptance criteria of user stories, behavior scenarios, test scenarios, the criteria for the next version of the software product, the domain model, etc.

The main purpose of the requirements tracing is to get knowledge of how the requirements are related with each other and with the artefacts of the development project, to understand the complexity of logical relationships between the requirements and the artefacts, and to be able to trace previously adopted decisions and to make new decisions about the revision of the requirements set when one of the requirements changes. In other words, the requirements tracing is a mechanism allowing us to answer the questions: who, what, where, when, why and how changes, or should change, the requirements.

Of particular interest in the requirements management are the answers to the following questions: is it necessary to revise non-functional requirements in connection with changes in functional ones, which project artefacts the change affects, which test scenarios do not currently meet the actual requirements. The use of the ontology to describe the different types of relations between the requirements artefacts seems promising for solving the task of requirements tracing compared with the relational databases commonly used in the requirements management systems and project management systems. Depending on the artefact type, the relationships types such as “is a part”, “includes”, “is associated with”, “is tested with” can be distinguished in the ontology. Using a variety of predefined types of relationships increases the flexibility of the queries that can be made while analyzing which aspects of the software product affect the change in the requirement. This in turn makes it possible to improve the accuracy of assessing the impact of changes in the requirements of the software product and to reduce the risks of incorrect assessment of changes in the requirements.

Thus, in present paper we propose an ontology-based approach to support for software requirements traceability. The paper is organized as follows. Section 1 justifies the relevance of the research topic. Section 2 provides a review of the research topic and introduces the terminology used. In section 3 we describe ontology-based approach to organizing support for the process of requirements tracing in agile environment. In section 4 we draw conclusions.

2. Theoretical background

2.1. Requirements traceability

Methods for checking traceability of the requirements while the development of software products began to be used in the early 1970s [2]. In the 1980s, the concept of traceability of the software product began to be applied in national and international standards. One of the first major studies concerning the requirements traceability was the work of O. Gotel and C. Finkelstein [3]. They identified the problem of understanding the boundaries of requirements traceability and separated two different types of traceability: Pre-RS traceability (covers a part of the requirement life cycle of the requirement before being included into the requirements specification) and Post-RS traceability (covers a part of the life cycle of the requirement after being included into the requirements specification). Pre-RS traceability is necessary to trace, for example, the source of the requirement, and Post-RS traceability – to trace the requirement deployment results. Unfortunately, the above work does not provide definitions of the requirements and requirements specification, but it can be assumed that the conceptual apparatus corresponding to the ideology of the IEEE 830 standard was implied.

In the software development, the main purpose of requirements traceability is used to ensure that the developed software product meets the expectations of the customers and the users. According to the paper [4] the objectives of the requirements tracing stage are to demonstrate for the customer that changes in the requirements have been taken into account and the software product complies with the requirements, checking that the test cases meet the requirements, and also checking that unnecessary features were not created for which there were no requests for implementation. At the same time, motivation to provide the traceability of requirements is to ensure completeness, to search for elements that are subject to changes, to promote understanding of the requirements for new project participants and to manage the information produced during the development of the software product.

In paper [5] among the main problems of supporting traceability process are as follows: the use of informal traceability methods, unproductive cooperation between the persons responsible for coordinating traceability, the presence of obstacles to obtaining necessary information.

The challenges in requirement traceability are discussed in the paper [6]. The first significant challenge is cost of time and effort required to enter traceability data. The second challenge is the difficulty of maintaining traceability through changes. The third challenge is the existence of different viewpoints on the practical benefits of traceability held by various project stakeholders. This leads to the intent to spend the shortest possible time in maintaining of traceability because the benefits from it are not sufficiently understood by project participants. The next challenge is the organizational problems that lead to careless maintenance of requirements traceability. The cause of such challenge is the fact that “many organizations do not train their employees regarding the importance of traceability and traceability is not emphasized in undergraduate education”. The fifth challenge is “Poor Tool Support”. Manual traceability methods are not suitable for the software engineering industry because the number of traceability relations grows exponentially in this area. On the other hand, COTS (commercial-off-the-shelf) traceability tools are also hardly suitable for the needs of the software development industry, because COTS tools are typically marketed as complete requirements management packages.

The most popular requirements traceability technique is the requirements traceability matrix (RTM). Requirements traceability matrix is a two-dimensional table that shows the relations between the requirements and the requirements artefacts. Most often, the requirements traceability matrix visualizes the relations between the requirements and the test cases. In this case, the column headings are requirements identifiers and / or requirements short names and the row headers are test cases identifiers and / or test cases short names. If the requirement from the current column is checked by the test case from the current row then a label is placed on the intersection of the row and the column. The matrix can be either implemented in the form of digital document (for example, MS Excel), manually created or automatically generated. The manual traceability is extremely time-consuming and can be used for very small projects only. This matrix is automatically generated in some of the requirements management systems (for example, Rational RequisitePro) and project management systems (for example, JIRA).

2.2. Agile requirements

We define a requirement as an assertion that, firstly, describes a certain characteristic of the software product and, secondly, is recorded using a certain technique during requirements engineering. The requirements can be divided into two main categories: functional and non-functional. Functional requirements define behavior of the software product. Non-functional requirements determine software quality characteristics that are divided into two types: external and internal. External quality characteristics are those from the user’s point of view, for example, system performance. Internal quality characteristics are considered from the developer’s point of view, for example, code readability.

The most common form to record the requirements when applying the agile methods is user story. User story is “simple narrative illustrating the user goals that a software function will satisfy” [7].

According to ISO / IEC / IEEE 26515-2011 user story has to include [7]:

- role of the user;
- goal that the user achieves;
- value for the customer;
- acceptance criteria that allow to determine that the user story is implemented.

The following recording scheme is usually used for the requirements in the form of the user story:
As a <type of user X >
I want <some goal Y>
So that <some reason (benefit) Z>

User history is a high-level requirement. The text of the user story usually serves as the basis for a conversation between the development team and the Product Owner (or customer) in order to clarify the details of the implementation. Requirements engineering in the Agile environment is an iterative process in which the requirements evolve in every sprint. The sprint is the time interval during which the development team creates a potentially releasable software product (product increment). For each user story implemented in the sprint, acceptance criteria must be specified. Acceptance criteria are a set of statements that specify both functional and non-functional requirements for current iteration of the software development. Acceptance criteria have a clear pass or fail result. Acceptance criteria can be written in a variety of formats. There are two main approaches to the recording of acceptance criteria: rule-oriented or scenario-oriented ones. A rule-oriented approach involves the use of business rules written in structured natural language (for example, The RuleSpeak® Business Rule Notation). Gherkin notation can be used with a scenario-oriented approach.

User stories and Gherkin scenarios are most often used to describe functional requirements. However, some development teams also practice describing non-functional requirements in user story format to maintain the uniform style of all requirements.

The definition of done is a list of requirements that must be fulfilled by the team in one sprint to create a product increment that is potentially ready for delivery. Definition of done is usually applied to all elements of the sprint backlog. Acceptance criteria is applied to a given user story. The user story is not considered completed until all the acceptance criteria and the definition of done criteria are fulfilled for it.

2.3. Application of ontologies to the requirements traceability

By their nature, software requirements are complex knowledge. At present, ontologies have established themselves as one of the possible ways of presenting complex knowledge and reasoning about them. The possibilities of extracting new knowledge from given facts and knowledge that makes the technologies for constructing a semantic network, in particular OWL-ontologies, attractive for presenting knowledge about the requirements and artefacts of the project.

Methodology OntoREM (Ontology-driven Requirements Engineering Methodology) developed by Kossmann M. et al. was one of the first scientific results to fully demonstrate possibilities of ontologies for requirements engineering. OntoREM is a semi-automated methodology that includes processes, methods and tools designed to create requirements specifications. Methodology OntoREM was applied by its authors in the company Airbus to develop aircraft operability requirements [8]. The toolkit developed within the framework of this project enables creating a Traceability Mindmap [9,10].

Siegemund K. proposed an ontology-oriented approach to the requirements engineering for software products. The system developed "enables the documentation of structured, reusable, unambiguous, traceable, complete and consistent requirements as demanded by the IEEE specification for Software Requirement Specifications" [11].

In paper [12] it is proposed to use a semantic network as a model for representing knowledge of traceability. The network includes a set of objects created during the software development and a set of relations between the objects (for example, relations "whole-part", "describes", "specifies", etc.). A set of objects includes two subsets: a set of requirements and a set of requirements artefacts. The semantic network also includes a set of inference procedures which enables determining the existence of relations between the elements of the project. The validity coefficient of the relation between the two elements of the project is calculated on the basis of information about the length of the chain of logical inference, the types and the validity coefficients of the relations.

The metamodel used for reasoning about the requirements is proposed in paper [13]. The model includes requirements, requirement artefacts and stakeholders. The requirements are linked by four types of relations: Refines, Requires, Conflicts, and Contains. These relations allow building rules to reason about requirements traceability, consistency and completeness.

Some problems of requirements traceability are discussed in paper [14]. This research is devoted to the questions of frame ontology development. The latter enables building a harmonized model of requirements for a particular software development project.

Ontological approach to the requirements engineering in the agile projects is presented in paper [15]. Thamrongchote C. et al. proposed to identify relations between user stories based on user role matching. For example, an authorized user can take all actions that a user “Guest” can perform. The class “Guest” is considered as a subclass of the class “Authorized user” in the domain ontology. This approach can be used to trace the fact when the feature provided to the user with the role “Guest” changes, the same feature should be available to the user with the role “Authorized User”.

Thus, the analysis of scientific publications shows the perspective of using ontologies to support for processes of the requirements engineering in general, and support for processes of requirements traceability in particular. Two main directions of research can be distinguished: (1) the development of domain ontologies describing the relations between different user roles, the user's actions in the system, the elements of the software product and (2) the development of ontologies describing the possible relations between the requirements and project artefacts in accordance with some standard or a set of standards and methodologies.

3. Ontology-based approach to the requirements traceability in Agile development

3.1. Ontology for requirements engineering process in Agile development

The right ontological language should be chosen before an ontology construction. Ontology specification languages are divided into two classes: traditional ontology specification languages and web-based ontology specification languages. The former languages are based on the some paradigm such as first-order logic, description logic, frame-based languages, etc. Traditional ontology languages were developed in the early 1990s. The second group of languages is based on Web standards (RDF, XML and etc.). Ontology languages can be assigned to both classes [16,17].

A language that supports descriptive logic and semantic markup standards is needed to implement the task in the work, so Web Ontology Language (OWL) was chosen. The OWL is a Semantic Web language designed to represent knowledge about things, groups of things and relations between things. The OWL ontology could be represented by the following tuple:

$$Z = \langle C, R, Ao, I \rangle \tag{1}$$

where $C = \{c_i \mid i = \overline{1,n}\}$ is a finite non-empty set of classes (concepts) describing the basic notions of the application domain;

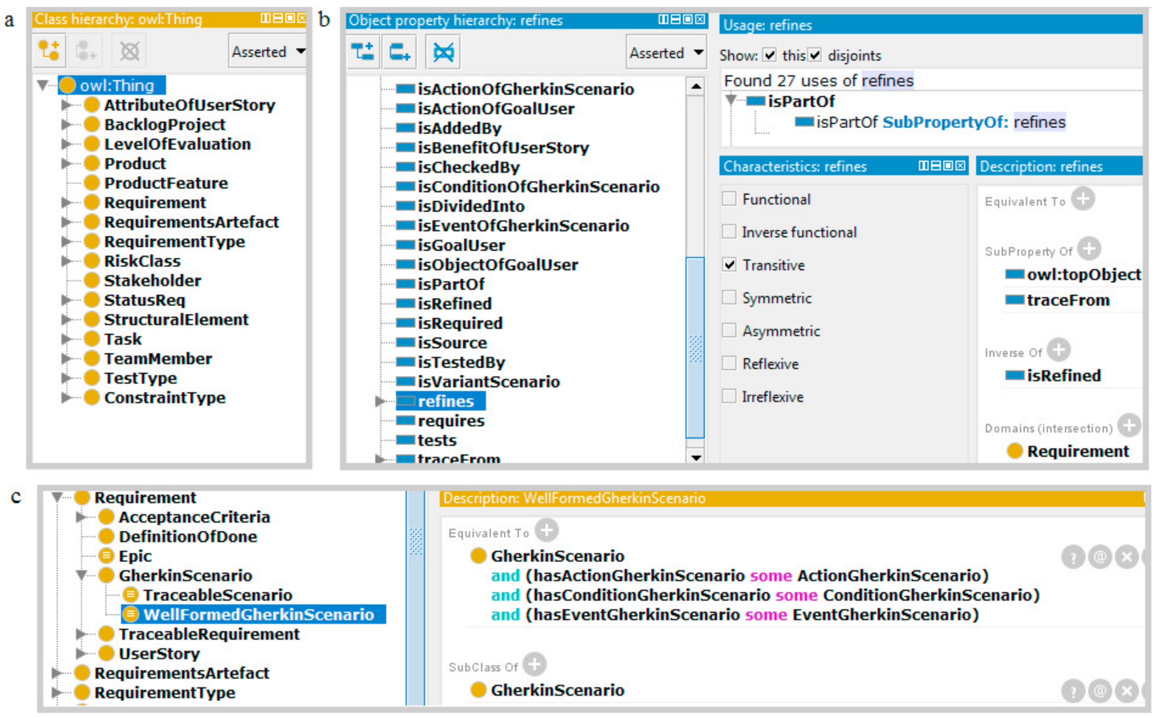
$R = \{r_i \mid i = \overline{1,m}\}$ is a finite set of binary relations between the classes, $R \subseteq C \times C$, including an antisymmetric, transitive, non-reflexive hierarchy relation R_{ISA};

Ao is a finite set of assertional axioms;

I is a set of instances of the concepts.

Construction of the software requirements ontology is based on understanding of the term “requirement” presented in Section 2.2. In this paper, requirements recorded as user stories and behavior scenarios will be considered as subclasses of the class “Requirement”. “Acceptance criteria” and “Definition of done” are also subclasses of this class. User stories will be considered as the top-level requirement. The “behavior scenarios”, “acceptance criteria” and “definition of done” will be also considered as elements of the class “RequirementsArtefact”, because they represent information about the requirements being created, changed and used in the process of implementing the requirements. Axioms that verify pattern matching of user stories and behavior scenarios are also defined in the ontology. Object properties and data properties form a set of the requirements and the requirements artefacts. Fig.1 shows taxonomy of top-level classes, part of object properties and an example of axioms.

We used Protégé 5.1 editor to create the ontology model. Protégé 5.1 editor comes with a HermiT reasoner. The HermiT is the OWL reasoner based on hypertableau reasoning algorithm. This reasoner supports all of OWL 2 DL.



Class hierarchy: owl:Thing

Asserted

owl:Thing
- AttributeOfUserStory
- BacklogProject
- LevelOfEvaluation
- Product
- ProductFeature
- Requirement
- RequirementsArtefact
- RequirementType
- RiskClass
- Stakeholder
- StatusReq
- StructuralElement
- Task
- TeamMember
- TestType
- ConstraintType

Object property hierarchy: refines

Asserted

- isActionOfGherkinScenario
- isActionOfGoalUser
- isAddedBy
- isBenefitOfUserStory
- isCheckedBy
- isConditionOfGherkinScenario
- isDividedInto
- isEventOfGherkinScenario
- isGoalUser
- isObjectOfGoalUser
- isPartOf
- isRefined
- isRequired
- isSource
- isTestedBy
- isVariantScenario
- refines
- requires
- tests
- traceFrom

Usage: refines

Show: ✓ this ✓ disjoints

Found 27 uses of refines

- isPartOf
  - isPartOf SubPropertyOf: refines

Characteristics: refines

- [ ] Functional
- [ ] Inverse functional
- [✓] Transitive
- [ ] Symmetric
- [ ] Asymmetric
- [ ] Reflexive
- [ ] Irreflexive

Description: refines

Equivalent To

SubProperty Of
- owl:topObject
- traceFrom

Inverse Of
- isRefined

Domains (intersection)
- Requirement

- Requirement
  - AcceptanceCriteria
  - DefinitionOfDone
  - Epic
  - GherkinScenario
    - TraceableScenario
    - WellFormedGherkinScenario
  - TraceableRequirement
  - UserStory
- RequirementsArtefact
- RequirementType

Description: WellFormedGherkinScenario

Equivalent To

GherkinScenario
and (hasActionGherkinScenario some ActionGherkinScenario)
and (hasConditionGherkinScenario some ConditionGherkinScenario)
and (hasEventGherkinScenario some EventGherkinScenario)

SubClass Of

GherkinScenario

Fig. 1. (a) taxonomy of the upper level concepts; (b) object properties; (c) axiom “Gherkin Scenario is well formalized”.

The assignment description of the upper-level classes is given in Table 1.

Table 1. The description of the upper-level classes.

Class name	Description
AttributeOfUserStory	describes the attributes of user stories such as priority, risk level, etc.
BacklogProject	includes subclasses describing the project's backlogs: maintenance backlog, product backlog, sprint backlog, task backlog
LevelOfEvaluation	includes subclasses such as qualitative scales used in assessing priorities, risks, etc.
Product	contains subclasses corresponding to the concept “software product”
ProductFeature	instances of this class are the features being developed
Requirement	contains subclasses corresponding to the concept “requirement”
RequirementsArtefact	contains subclasses corresponding to the concept “requirements artefact”
RequirementType	includes subclasses describing the classification of requirement types
RiskClass	includes subclasses describing the classification of risks
Stakeholder	contains subclasses corresponding to the concept “stakeholder”
StatusReq	includes subclasses describing the classification of statuses that can be assigned to the requirements
StructuralElement	includes subclasses describing the structural elements of which the requirement template is composed
Task	includes instances which are tasks a development team works on
TeamMember	includes subclasses describing roles in a team
TestType	includes subclasses describing the classification of test types
ConstraintType	includes subclasses describing the types of constraints for non-functional requirements

3.2. Traceability of requirements knowledge in the requirements ontology

Ontology stores information about requirements, requirements artifacts, requirements sources (including stakeholders), and team members who add requirements as instance knowledge into the ontology. The relations established between the instances can be used to identify directly or indirectly affected requirements and artifacts when making changes into the software requirements.

Ontology includes two types of relations implemented through object properties: “traceFrom” and “traceTo”. The first relation includes subrelations which enables bottom-up tracing, the second – top-down tracing. Subrelations of the relation “traceFrom” are: (1) “isTestedBy”; (2) “checks”; (3) “refines”; (4) “hasSource”. The relation “isTestedBy” enables finding a set of tests that checks a functional part of a certain software product. The relation “checks” shows which acceptance criteria are checked by the test. The relation “refines” means that the requirement is derived from another requirement and adds more details. The relation “refines” includes subclasses: (1) “isVariantScenario”; (2) “isKeyScenarioOfUserStory”; (3) “isRule”; (4) “isPartOf”. The relation “isVariantScenario” shows a link to the key scenario which is detailed by variant scenario. The relation “isKeyScenarioOfUserStory” shows a link to the user story which is detailed by the acceptance criterion formulated using a scenario-based approach. The relation “isRule” shows a link to the user story which is detailed by the acceptance criterion formulated using a rule-based approach. The relation “isPartOf” shows a link to the user story which is detailed by another user story. For the relation “traceFrom” and its subrelations, inverse properties are given through the relation “Inverse Of”. This enables to trace top-down using the relation “traceTo”. Fig. 2 shows the hierarchy of properties for the relation “traceFrom” and types of domains and ranges for each of subrelation of this relation.

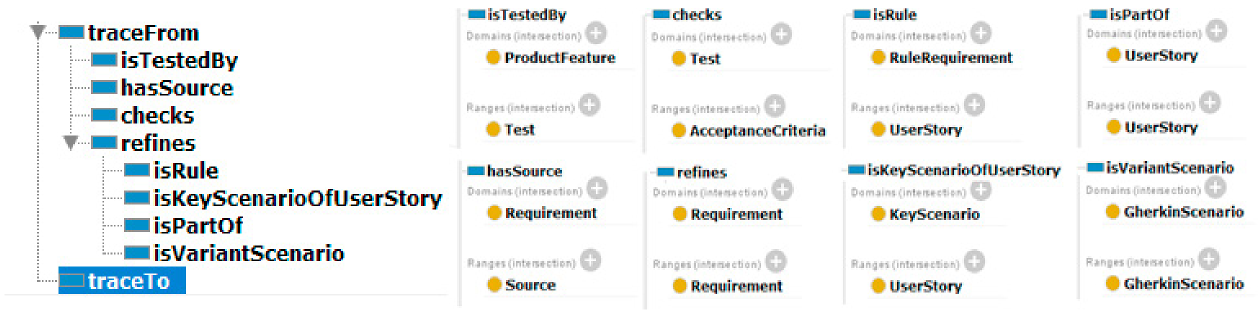
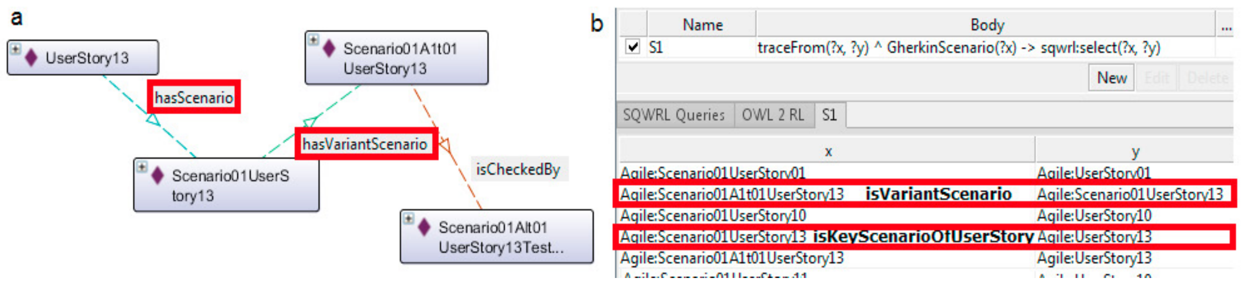


Fig. 2. Object property “traceFrom” and its subproperty.

When the relations between instances of classes are being described, the relations “traceFrom” and “traceTo” can be concluded by means of reasoning and can be used, for example, in SQWRL requests for bottom-up or top-down tracing. Fig. 3 shows an example of SQWRL request for bottom-up tracing behavior scenarios.



	Name	Body
✓	S1	traceFrom(?x, ?y) ^ GherkinScenario(?x) -> sqwrl:select(?x, ?y)

x	y
Agile:Scenario01UserStory01	Agile:UserStory01
Agile:Scenario01A1t01UserStory13 **isVariantScenario**	Agile:Scenario01UserStory13
Agile:Scenario01UserStory10	Agile:UserStory10
Agile:Scenario01UserStory13 **isKeyScenarioOfUserStory**	Agile:UserStory13
Agile:Scenario01A1t01UserStory13	Agile:UserStory13

Fig. 3. (a) an example of relations between instances; (b) an example of bottom-up tracing.

As a result of the query, all instances of the class “GherkinScenario” are selected in the variable *x*. The variable *y* contains instances of the requirements that are linked through the relation “traceFrom” with the variable *x*. They are derived through the reasoning mechanism including in particular the relation “isVariantScenario” being the inverse to the relation “hasVariantScenario” obtained as a result of determining the relation “Inverse Of”.

4. Conclusion

Thus, we proposed an ontology-based approach to support for requirements traceability. The developed ontology contains theoretical knowledge for the requirements engineering in Agile environment. Support for traceability requirements makes it possible for a development team to operatively manage the evolution of the requirements for the software product. It accelerates decision making on the need to revise the project artifacts associated with the changes and enables more precise determination of the scope of these changes.

Acknowledgements

The reported study was funded by Russian Ministry of Education and Science, according to the research project No. 2.2327.2017/4.6.

References

[1] Mäder P, Egyed A. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering* 2015; **20**(2):413–441.

[2] Azmi A, Ibrahim S Formulating a Software Traceability Model for Integrated Test Documentation: a Case Study. *International Journal of Information and Electronics Engineering* 2011; **1**(2):178-184.

[3] Gotel O, Finkelstein C. An analysis of the requirements traceability problem. *Proc. of the First International Conference on Requirements Engineering* 1994; p. 94 –101.

[4] Grave A. Testing and Traceability Aspects in the Context of the Model Driven Architecture (MDA). *Scientific Journal of Riga Technical University. Computer Sciences* 2010; **41**(1):52-59.

[5] Salem AM. A Model for Enhancing Requirements Traceability and Analysis. *International Journal of Advanced Computer Science and Applications* 2010; **1**(5):14-21.

[6] Kannenberg A, Saiedian H. Why software requirements traceability remains a challenge. *The Journal of Defense Software Engineering* 2009; **22**(7):14–19.

[7] ISO/IEC/IEEE 26515:2011(E), Systems and software engineering. Developing user documentation in an Agile environment.

[8] Kossmann M, Wong R, Odeh M, Gillies A. Ontology-driven Requirements Engineering: Building the OntoREM Meta Model. *Proc. of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications* 2008; p. 1–6.

[9] Antonini K, Odeh M, Kossmann M, Lange C. Evaluating the Effectiveness of Mindmapping in Generating Domain Ontologies using OntoREM: The MASCOT Case Study. *Proc. of the 15rd International Arab Conference on Information Technology* 2014; p. 247-255.

[10] Zayed R, Kossmann M, Odeh M. The OntoREM-Mind Mapper Software for Visualising OWL Ontologies. *Proc. of the 16rd International Arab Conference on Information Technology* 2015.

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

[12] Rogalchuk VV. Collection of Data on Traceability by Using the Module of Integrated Development Environment (in Russian). *Proc. of Petersburg Transport University* 2009; **1**(18):127–134.

[13] Goknil A, Kurtev I, van den Berg K. A Metamodeling Approach for Reasoning about Requirements. *Proc. of the 4th European Conference on Model Driven Architecture - Foundations and Applications* 2008; p. 310-325

[14] Avdeenko TV, Pustovalova NV. The ontology-based approach to support the completeness and consistency of the requirements specification. *Proc. of the 11th International Siberian Conference on Control and Communications* 2015; p.1-4.

[15] Thamrongchote C, Vatanawood W. Business process ontology for defining user story. *Proc. of the 15th International Conference on Computer and Information Science* 2016.

[16] Kalibatiene D, Vasilecas O. Survey on ontology languages. *Proc. of the 10th International Conference International Conference on Business Informatics Research* 2011; p. 124-141.

[17] Rebstock M, Janina F, Paulheim H. *Ontologies-Based Business Integration*. 1st ed. Berlin: Springer-Verlag Berlin Heidelberg; 2008.