Adaptation of Event-Based Traceability Method for Environment with Hierarchal Composed Workspaces

Vl. Jotov

*Veliko Tarnovo University “St. Cyril and St. Methodius”,
Veliko Tarnovo, 5000, Dimitar Najdenov str., no.34A, tel. +359888004067, e-mail: vjotov@acm.org*

Abstract: The task of tracing change causes and managing the change requests has major impact especially for middle-large projects. This paper mission is to introduce possibilities for version traceability within environment with hierarchical workspace composition. Adapting event-based traceability method is focused to provide fundamental for new tools that will be able to support change analysis. All this is combined with extension of previously presented data model for supporting versioning in environment with hierarchical workspace composition

Key words: data event-based traceability, hierarchical workspace composition, data model, cause-effect link tracing

1. INTRODUCTION

The bad practice of delivering an immutable Requirements Specification Document could cause enormous increase of project budget. This type of document cannot cover all aspects that the end product has to support. The requirements and project scope change often without been reflex from the Requirements Specification Document. Therefore requirements document has to be live document that evaluate in parallel with the software product. The authors of [1] recommend requirement traceability technique as a good practice for facilitation of change management.
Good change management requires good traceability of all artifacts that are part of the system – from requirements, system architecture design up to quality assurance tests cases, source code and documentation. Using traceability among artifacts leads to the following benefits:

- Improved impact analysis that help to determine how the system should be changed and preliminary estimation for budget and resources;
- Lower maintenance costs as the system is supported with updated requirements,
- Better assessment of product quality that is tested against the last approved requirements,
- Reduce of development rework and other [2, 4, 10].

In this paper is presented an approach for tracing a version among artefacts in an environment with hierarchical workspace composition adapting event-based traceability method. Current section examines change traceability methods. Here are introduced principles of versioned object visibility among hierarchically composed workspaces. Section 2 presents the cause-effect traceability links among objects. In section 3 we present new transactions that allow applying of event-based traceability method. Section 4 extends our previous data model with elements for supporting version traceability. In last section 5 are presented conclusions and direction of further researches.

1.1 TRACEABILITY METHODS

“Traceability is the ability to chronologically interrelate uniquely identifiable entities in a way that is verifiable” [11]. In [3] the authors make an attempt to summarize existing tracing methods for tracing non-functional requirements. The same methods are applicable also for functional requirements. The advantages and disadvantages of the methods generalized below:
Matrices – a commonly used technique, that not scalable for big amount of links among objects. The method is difficult for maintaining due to high amount of manual operations.
Keywords and Ontology – It supports traceability between requirements and system architecture. The advantage of this approach is that it doesn’t require maintenance of centralized matrix. The issue here is the construction set of reserved keywords that are semantically used through the whole product lifecycle.
Aspect Weaving – This method is applicable only for aspect oriented development approach. In it links are dynamically created.
Information Retrieval Methods – This is one of the most frequent used methods. Even that it requires the user to verify request for traceability it has high level of automation. Using this method in [9] authors present an impact analysis method for time and budget estimation of requests for change.
Event-based Traceability – This method uses publish-subscription architecture for describing traceability links and it provides high level of automation. This method is used for supporting the proposed in current paper trace methodology.
Tracing also has a direction and a direction classification is presented in [4]:

- Forward from requirements – this traceability describes the allocation of implemented objects.
- Backward to requirements – the idea of this traceability is to provide assurance that the implemented artefacts meet the requirements i.e. their derivation.
- Forward to requirements – technology could set some boundaries to the required system and in such cases “Forward to requirements” traceability is used.
- Backward from requirements – this supports the socio-political context of requirements appearance.

1.2. ENVIRONMENT WITH HIERARCHICAL COMPOSITED WORKSPACES

In [6] we have presented a model of environment with hierarchical composition of workspaces. The presented model

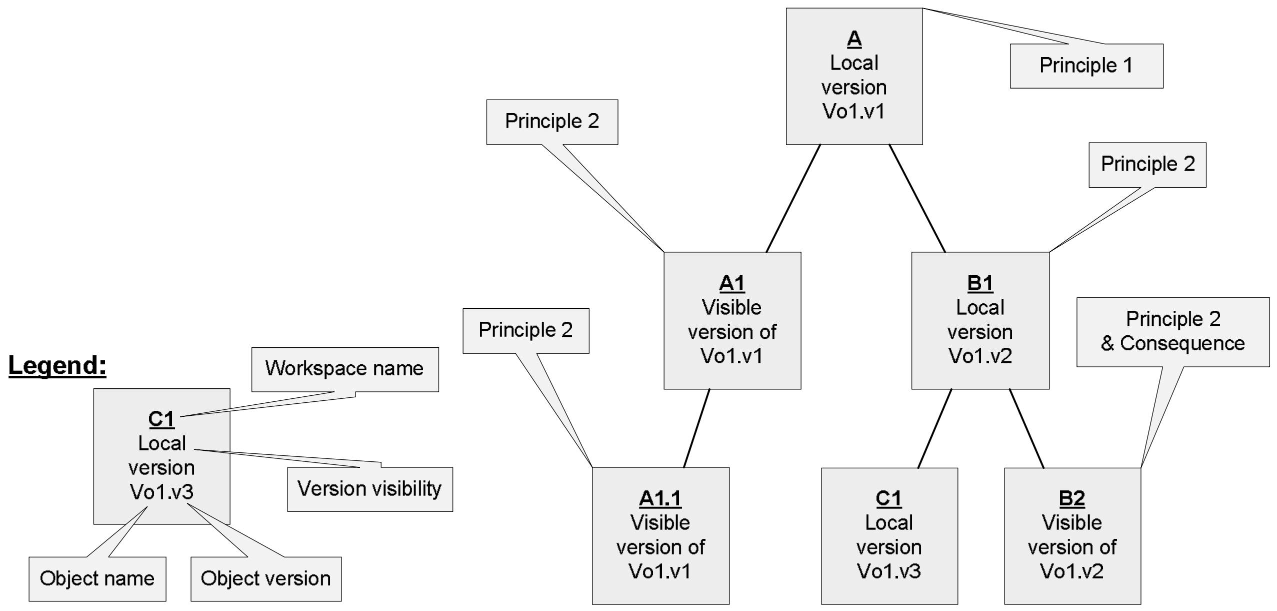

Figure 1 Example of visibility principles and the consequence

stands on the following two principles for object's version visibility:

Principle 1: For each workspace all local versions of versioned objects are visible within the workspace in despite of existence of any other versions in parental workspaces.

Principle 2: One version of versioned object from certain workspace is visible in all children workspaces unless there are no local versions (e.g. main principle).

From the both principles we can obtain the following.

Consequence 1: Let object Vo1 does not have local version in workspace B2. Then object Vo1 is visible in workspace B2 with its version situated in the closed parental workspace where it has a local version (figure 1).

The above model principles are incorporated with the following two transactions over versioned object: version propagation to parental workspace; version put-back from parental workspace. The version propagation is similar to check-in operation in commonly used version control systems [8]. Nevertheless of the similarity the main difference is that using check-in transaction developer uploads an object version to a common repository. In our model the term workspace is closer to the term of sub-repository and the presented transactions are defined as interaction between two sub-repositories. The essence of propagation transaction is to distribute a version of an object from one workspace to its parent workspace where in combination with Principle 2 we achieve automatic object's version distribution among sibling workspaces. The opposite transaction of put-back could be described as a decline of an object's version and its automatic substitution (Consequence 1) with its visible version in the parent workspace.

2. INITIATORS AND EFFECTORS

The system Molhado [7] supports traceability link networks expanded with versioning for each of them. The authors regard the trace link as versioned object and this idea will have major place in the approach elaboration as the requirements are changed during the project. Other examples of change-causes link could be marked the change requests and reported malfunctions. Each trace link has two ends, begin or initiator for the change and the end is the effected item or effector.

The initiator represents the cause for change but in software-development process there is a special term – work item representing the amount of work that has to be done. perspective the initiator We have to have in account that certain software unit implements more requirements i.e. one effector implementation can be caused of more than one initiator. The opposite situation is also valid – one initiator to cause change into more then one object.

Another characteristic of traceability links is that one object's version in one case can be effector for one link but in another case the same version could play the role of change initiator for another object's version. Good example for this is the system architecture design and UML diagrams that reflect the requirements. They are initiators for creation/modification of the source code of the program and its documentation.

In his paper [4] Helming presents classification of workitem types: action items, issues, bug report and work package - a composition of sub-workitems. Workitems of type work package are used for division, distribution and management of work among teams and their members. When the work is performed the cause for that is the workitem therefore the object of workitem is the initiator in cause-effect relation link.

3. TRANSACTION OVER VERSIONED OBJECTS FOR SUPPORTING EVENT-BASED TRACEABILITY METHOD

As it was mentioned above the event-based traceability method is highly automated method and it is based of catching events and traceability links generation. In our case events for the method are the changes over versioned objects. There several manual actions that has to be made for setup the method. In this section we will specify three new transactions that support the method.

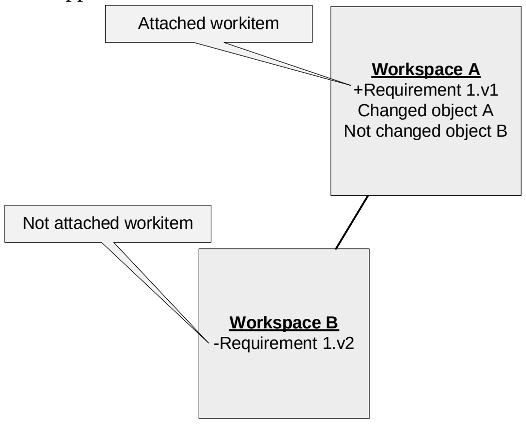

Figure 2 Risk of conflict on requirement publication to workspace A

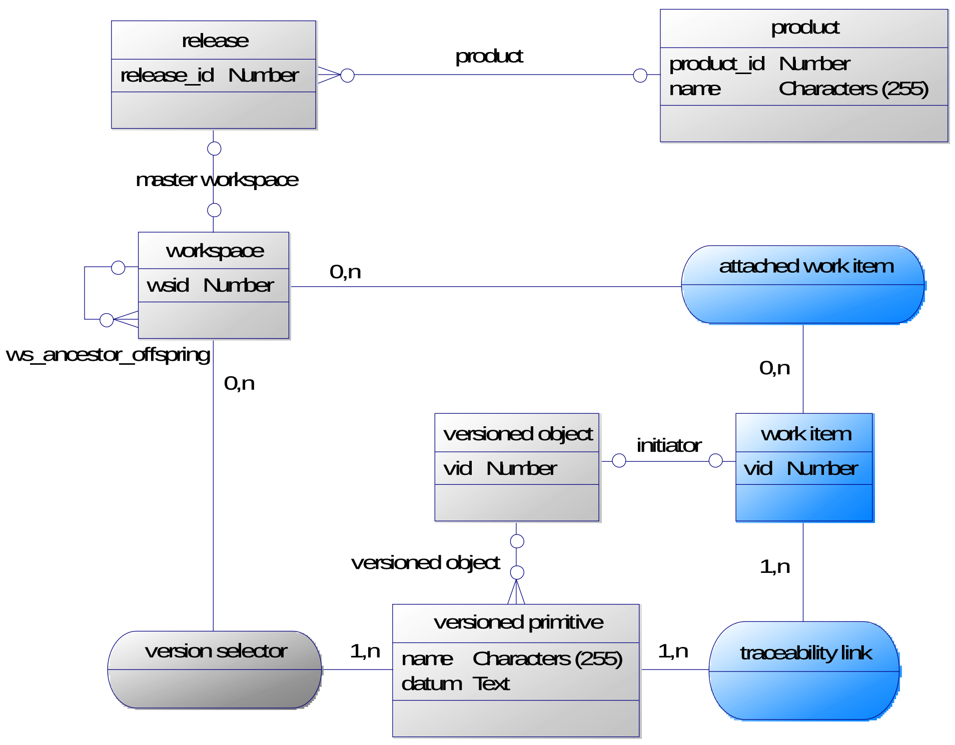

Figure 3 ER data model supporting event-based traceability

As we see in the beginning of the paper requirement artefact is changeable therefore it is a good example to be examined as a versioned object. On other hand each specific version of a requirement is a potential initiator for change or for creation of new artefacts. Thus we are able to define the first transaction over versioned objects – mark-up the object as workitem.
When a developer starts making changes over the software product he/she makes them according to selected sub-set of requirements, architecture, etc. For the environment with workspaces this approach leads us to the following transactions – attaching of workitems to a workspace and detaching of workitems from a workspace. After attaching one or more workitems to a workspace the system will be able automatically generate traceability links on events of change of versioned objects that are visible and modifiable in the workspace.
In order to gain a completeness of the presented model we have to introduce the following rules on workitem usage:
Rule 1: A workitem should not be changeable in all workspaces where it is attached. Otherwise we will get a recursive traceability links.
Rule 2: A versioned object that is specified to be a workitem cannot be published to a workspace where it is attached. In case of avoiding this rule it is possible that one developer working in workspace B will be able to change requirements according to which second developer is implementing the system in workspace A. On Figure 2 we present an example of this situation - On publishing Requirement 1.v2 developer using workspace A will implement Object B according to version 2 of the requirement, but Object A will remain implementation of first version.

4. DATA MODEL FOR SUPPORTING VERSION TRACEABILITY

In [5] we have presented an ER model of versioning system based on environment with hierarchical composition of workspaces. The entities defined in the model (figure 3 gray elements) are as follow: product, product release, workspaces, versioned object and its versioned primitives. Each product has at least one revision, in particular an empty or zero release. Each revision has one master-workspace that represents revision compilation. Master–workspace and all other workspaces in the release hierarchy are connected with the recourse relation ancestor-offspring. Using the relation of version selector defines locally situated versions of objects – versioned primitives that represent all versions of objects (versioned_object).
In this paper we will extend data model so that it will support event-based traceability method described in current paper. Main goals of that extension are to keep all current system capabilities and to guard further system extension feature of system data model. Therefore only one entity and two relationships are added– work item; attached work item; traceability link. On figure 3 we present new model for versioning that supports event-based traceability. New elements of the model are presented with blue colour.
Transaction of specifying an object as a workitem is equal to creation of new record in workitem entity. Transactions of attaching and detaching of workitem to/from a workspace are implemented in the model as creation and deletion of attached workitem relationship between the entity of workspace and workitem. Traceability method is supported as follow: on creation of new version of an object (versioned primitive) and using relationships of vector selector and attached workitem we are able automatically to create relationships of traceability link.

5. CONCLUSION AND FUTURE WORK

Current research allows us using of event-based traceability method. This method facilitates gaining high level of automation in development process in environment with hierarchical composition of workspaces. Moreover we have opportunity to introduce good practices of audit and statistic in software development process without burdening all process participants with manual operations.
Presented extensions in transaction and data models do not change the existing infrastructure. Our further researches would be in the area of implementation and introduction of the models into the practice.

REFERENCES

1. A Guide to the Business Analysis Body of Knowledge, International Institute of Analysis, Ontario, 2009.
2. Asun ion, H. U., Towards practical software traceability, ICSE Companion '08: Companion of the 30th international conference on Software engineering, Leipzig, Germany, 2008, pp 1023–1026.
3. Cleland-Huang, J., Toward improved traceability of non-functional requirements, TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering, Long Beach, California, 2005, pp 14–19.
4. Helming, J., Koegel, M., and Naughton, H., Towards traceability from project management to system models TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, Washington, DC, USA, 2009, pp 11–15.
5. Jotov, Vl., Data model in a version control system based on hierarchical workspaces, Conference on 25th Anniversary of the Faculty of Education, 2009, Veliko Tarnovo, (under printing).
6. Jotov, Vl., Transaction over Versioned Objects in Hierarchical Workspace Environment, 3rd International Conference on Electronics, Computers and Artificial Intelligence, 2009, Pitesti, Romania.
7. Nguyen, T. N., Munson, E. V., and Boyland, J. T. Object-oriented, structural software configuration management. OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, Vancouver, BC, Canada, 2004, pp 35–36.
8. Collins-Sussman B., Brian W. Fitzpatrick, C. Michael Pilato, Version Control with Subversion, 2004, http://svnbook.red-bean.com/en/1.0/index.html (accessed 19 July 2010)
9. Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. How Long Will It Take to Fix This Bug?, MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories, Minneapolis, USA, 2007.
10. Wiegers, K. E., Software Requirements, Second Edition, Microsoft Press, Redmond, USA, 2003.
11. Wikipedia contributors, "Traceability," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Traceability&oldid=361263804 (accessed May 23, 2010).