Versioning in Adaptive Hypermedia

Evgeny Knutov, Paul De Bra, Mykola Pechenizkiy
Eindhoven University of Technology, Department of Computer Science
PO Box 513, NL 5600 MB Eindhoven, The Netherlands
debra@win.tue.nl, {e.knutov, m.pechenizkiy}@tue.nl

Abstract. This paper presents an approach that uses the terms, operations and methods of versioning applied to the Adaptive Hypermedia (AH) field. We show a number of examples of such a use which helps to facilitate authoring, managing, storing, maintenance, logging and analysis of AHS behaviour, providing extensive flexibility and maintainability of the systems.

Keywords: Adaptive Hypermedia, Versioning Systems, Revision Control, Authoring, User Model Updates, Context Changes.

1 Introduction

Versioning (also known as revision control, source control or code management) is the management of multiple revisions of the same unit of information. It is commonly used in engineering and software development to manage the development of digital documentation/code that are usually produced and maintained by a team of software developers. Changes to these pieces of information are usually identified by incrementing an associated version and binding it historically with the person, group or system making the change. On a large scale it is becoming the only way to keep track of all the changes, perform roll back operations if needed, merge modifications between different systems, individual users and groups and facilitate provenance analysis.

It is becoming obvious that tracking all the changes and operations over evolving structures makes reasonable and feasible to apply versioning methodologies in adaptive hypermedia systems. In this paper we will consider basic principles and operations of revision control applications in Adaptive Hypermedia Systems (AHS). We will not only start investigating the use of versioning for designing the structure and content of AHS applications but also take a look at the user model evolution in a versioning context.

2 Background

Engineering version control systems emerged from a formalized processes of tracking document versions. The main idea was to give a possibility to roll back to an early state of the document. Moreover, in software engineering, version control was introduced to track and provide access and control over changes to source code. Nowadays it is widely used to keep track of documentation, source files, configuration settings, etc.

As a system is designed, developed and deployed, it is quite common for multiple versions of the same system to be deployed with minor or major differences within different environments, furthermore system configuration/settings, state or/and context evolve which results in multiple co-existing system versions as well.

If we look at the revision control system as a system used to deliver a certain version of source code, documents, tables, or any file type or folder to a particular user at a given time or on request it may resemble delivering adapted content to a user in an Adaptive Hypermedia System. Of course version control systems are straightforward and don't take into account any user model or context changes, don't have any semantic or proof layers or any conceptual knowledge representation, but on the other hand provide very flexible techniques to deliver selected content in a certain context to a designated user, and facilitate methods to save and track system changes in a very efficient way.

At the same time AH systems store and process a lot of information, which is highly sensible to a context data and vice versa - context data itself, is highly dynamic and influences the adaptation process. Thus tracking AHS alterations and environment changes is a tightly connected task which requires storage and computational resources.

3 Revision Control in AHS

Hereafter we consider basic aspects of versioning in application to AHS, provide analysis of core terms and operations in parallel with AH. We take up major types of changes in a context of AH systems, models and process flow, thus coming up with a taxonomy of AH versioning techniques, types of changes and applications areas. As a result we come up with two examples typical for the AH field: one is an e-learning application providing an adaptive course and the other is a recommender system. We encapsulate these examples in a versioning context.

3.1 Types of Changes and Versioning Flow

In a layered structure of a generic AHS [7, 8] we presume that each layer is responsible for "answering" a single adaptation question [3], like Why?, What?, To What?, Where?, When and How?. Each layer may have its own data structure and thus also have its own way of dealing with evolutionary and versioning concepts of the system. This versioning structure in its turn can be devised and maintained using different technologies varying from a source control application to a historical Data Base. We don't investigate in the paper which technology is best suited for each of these layers (or questions), however we will speculate on this question, suggesting solutions. At the same time we take up basic terms and concepts of versioning and look at them through the eyes of Adaptive Hypermedia. In figure 1 we sketch the core components of a layered AH architecture and provide annotations with use-cases. We will first discuss versioning terms and operations in section 3.2 and then consider most of these use-cases in detail in section 3.3 through examples.

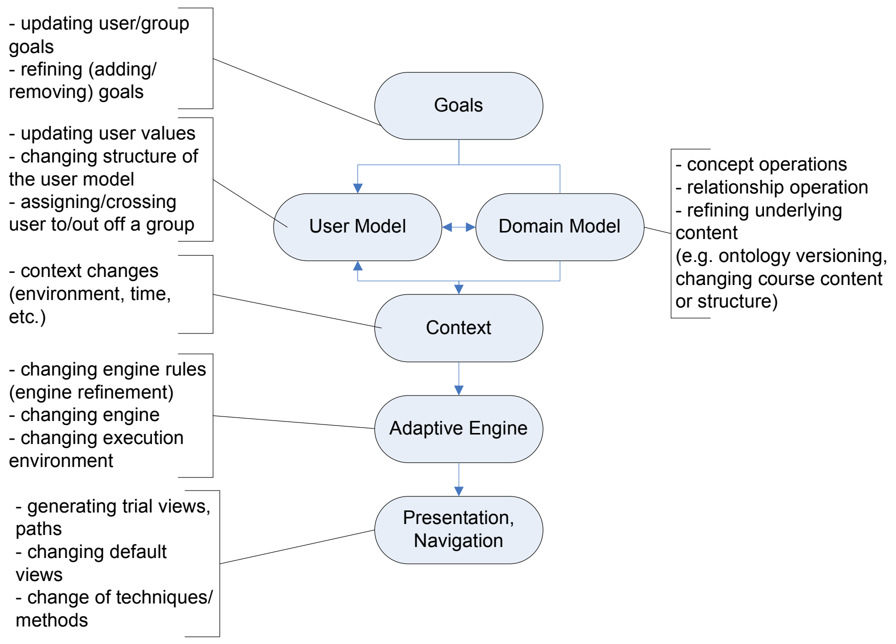
- updating user/group goals
- refining (adding/ removing) goals
Goals
- updating user values
- changing structure of the user model
- assigning/crossing user to/out off a group
User Model
Domain Model
- concept operations
- relationship operation
- refining underlying content
(e.g. ontology versioning, changing course content or structure)
- context changes (environment, time, etc.)
Context
- changing engine rules (engine refinement)
- changing engine
- changing execution environment
Adaptive Engine
- generating trial views, paths
- changing default views
- change of techniques/ methods
Presentation, Navigation

Fig. 1 Versioning process highlights

In figure 2 we outline a versioning taxonomy distinguishing types of changes, applications of versioning methodologies and we anticipate the set of potentially suitable versioning technologies. There are two major types of changes: structural and value changes. Both may take place in every layer/sub-system of AHS. For instance, refining the Domain Model structure and changing the underlying content or updating User Model values versus adding new properties to the model. Throughout the adaptation process, from authoring to presentation generation we anticipate the following versioning application areas:

Authoring – changes made by authors, to create, refine, update the structure and value parts of the adaptive application (usually these are the Domain Model and the Adaptation Model).

UM operations/updates – these refer to the user behaviour in the adaptive environment, tracking changes of the user knowledge, goals, preferences, achievements throughout system usage. Our UM versioning approach corresponds to ideas discussed in [4] and can serve as a back-end solution for ubiquitous user modeling.

Context changes – keeping track of context changes of the whole system as well as the context of a particular model, value or operation.

System deployment and analysis – changes done to an AHS to deploy the system in different environments, usually performed by the domain experts, whereas taking up the specifics of the domain to set up system logging, and analysis facilities.

We are not discussing technological aspects in depth, but can speculate about versioning (and related) applications to be used, and can foresee a number of well-known methodologies such as conventional source control systems and historical data bases for storing and querying for changes. OLAP technologies would suit to perform analysis. In addition a variety of custom-build systems (such as ontology versioning available in OntoView [2]) could also be considered.

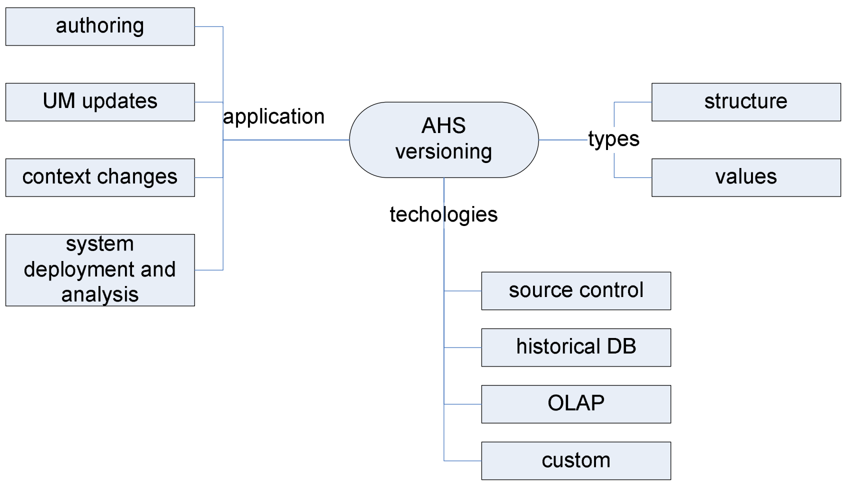

Fig. 2 AHS versioning taxonomy

3.2 Concepts and Operations of Revision Control

Here we would like to discuss the basic concepts of revision control and how they refer to the concept of evolution in AH systems. We will try to give a clear idea of how a particular concept or operation can be applied and will draw parallels with evolution and changes in AH through examples.

3.2.1 Basic versioning terms

Baseline – an approved version, usually the most stable, from which all the consequent versions are derived. In the AH field this is usually the core application, consequently used in different environment settings, within different user groups and thus being tuned up for those particular settings, which results in evolution and therefore new versions of the application. If we talk about **UM** – then the default version with which the user started can be considered as the baseline. In case of further analysis baseline is always the point of comparison of the application functionality and stability.

Branch – an instance is considered to be branched at a point in time, when from that time and on, two concurrent but independent versions of the same instance co-exist. As we have mentioned above, any AHS may result in a new set of settings or being used by a completely different group of people, thus a tune-up of the system will be required to meet the new settings or requirements. In these terms every AH system or model emerges in a branch of the initial baseline, representing a new authored version of the application (model/sub-system). Hence these systems may co-exist.

Branch can be also used as a prior concept of the User Model update, presentation generation or any other functionality which preview or try-out is desirable to analyze system behaviour. Even though the branch itself may not be used in the forthcoming version or update, a preview of a presentation or a try-out version of the application or engine will help to identify, observe and analyze the "***what-if***" case, and with the successful outcome commit branched changes, label as a new baseline or a head revision or merge them. This can be applied in terms of UM as well, considering the case when committing user properties to the latest application state is not desirable, however would be advantageous to see what can happen with the user in the new application environment. Though this UM case deviates from the original concept of the branch in a source control, it still uses the same principle of a concurrent instance, and in our case it is mapped on a context of UM in AH.

Head – the most recent commit, in other words the most latest version of the application/model existing on a single or multiple branches, with the latest functionality aspects prescribed to this branch(es). For UM the head revision is always the latest state of the model with all the corresponding values being updated to the latest state in the application.

Commit – applying changes to a central repository (branched or baselined) from one or multiple working sources. Usually the authoring and set-up processes require a number of experts, correspondingly a number of working instances which should emerge in a baseline version which in turn will require to commit all the authors' changes. In User Modeling we consider committing the changes to the user model values. At the same time not all the latest user changes should be committed to the user model in order to provide system stability (for example as it was done in AHA! system [5]).

Conflict – conflict occurs when the system can't reconcile changes made to the same instance of the structure. E.g. when the same concept properties are changed by two different authors, trying to commit them in the same course may result in a conflict which will require reconsideration of a concept structure and manual interference. Though conflict resolution may not guarantee such a property of the AH system as confluence, we can anticipate that versioned and annotated structures of the Domain or Adaptation model will be helpful in confluence analysis.

Change (difference, delta) – represents a specific modification under the version control. Though the granularity of change may vary from system to system, change always represents the basic structure of versioning concept. Changes evolve from the baseline, are branched, become new baselines, are committed, then resolved and merged. Depending on AH system, different instances can be changed (concepts, relationships, properties, content, context, rules or even a complete model or sub-system).

Moreover considering changes – is the best way to compare system functionality and settings, which is very important for analysis of inherited and evolved (branched – in terms of version systems) functionality of AHS.

Merge – in conventional source control systems – merge usually refers to a reconciliation of multiple changes made to different copies of the same file. In AH merge can be considered as part of an authoring or set-up process, where all the development and changes from different authors or domain experts come together. In general we can perform the merge operation over concepts, relationships, properties, individual user properties and user groups, ontologies, rule sets, etc. As a result merge provides a clear picture how (from which sources) the concerned concept or a property has emerged, facilitating application playback and analysis of changes. In applications where merge regulates UM values this operation can be granted to the user as well.

Resolve – is a user intervention in conflict resolution of 2 or more conflicting versions. We refer this to the authoring stage and let the experts decide on the outcome, or facilitate end-user resolution when dealing with UM.

Tag, Label – refer to an important snapshot of the system or its part in time implementing a certain functionality which has to be marked/tagged to settle a new state of the application.. This may be a stable Domain Model structure or a new user group with common interests (for instance).

At the same time an explicitly labeled structure of the model may be used in the context of OLAP (On-Line Analytical Processing). We will elaborate on this case below.

As we conclude, all aforementioned concepts of versioning and source control as they are applied purely in software engineering and document control can be represented in terms of Adaptive Hypermedia Systems, facilitating system flexibility and extensibility

3.2.2 Versioning operations

Having considered the basic concepts of versioning in the context of AHS, we can come up with the following classes of operations which will reflect typological and structural types of changes. The following taxonomy of operations was proposed in [2] in application to ontology evolution, however here we're trying to extend and elaborate it in terms of Adaptive Hypermedia. Hereafter we describe the properties of versioning operation.

Transformation – is usually a set of actual changes taking place over the evolving structure. These may be: changing properties of concept, classes (in case of ontologies), adding or removing concept structures.

Conceptual changes (may include concept and concept relationship changes) will refer to conceptual and abstract changes of the representation, structure and relationships within the AH system. This type of changes includes changes of types, relations, conceptual representation of a knowledge. It may also include relations between constructs of two versions of the model. Conceptual changes should be always supervised and carried out manually.

Descriptive changes – usually deal with metadata describing the reason, intentions, author's credentials, etc., regarding the new version of the model, ontology, relation or a concept. Descriptive changes don't contribute to the actual change, however may help to reason over different versions of the same instance.

Context changes - will describe the environment in which the current update was made and in which it is going to be valid. At the same time all the environment settings for a particular change can be considered as a context change. E.g. changing a concept in the Domain Model we can consider the state of all the concept relationships as well as some other Domain specific information as a local context of this change. At the same the complete system environment is a context of embedding a new adaptive engine or re-organizing user groups. As we can see from these examples – context changes can be and usually are the most space and effort consuming, moreover domain experts usually analyze all the modifications in terms of contextual changes to capture the complete picture of a particular change.

4 Use-Cases

In order to demonstrate our conceptual view of versioning in AH, we present two examples that are typical for the AH field.

4.1 E-Learning application use-case

Let's consider the following: the same discipline is taught in two different universities, where adaptive courses Course 1 and Course 2 on this subject are provided to the students (see fig. 3). After one year, a curriculum plan in the first university (University 1) changes and one of the teachers is asked to change (refine) the course navigation structure and some of the content which results in a new version of the course (Course 1 ver.2). At the same time one of the students moves from the second university (University 2) to the first one (University 1), having received the credits for a first part of the course (Course 2). The aforementioned teacher is creating a new version of the course, reusing the complete structure and introducing only designated changes (in navigation structure, updating adaptive engine with the associated changes in rules and changing content). Thus the initial version of the navigational structure is preserved to trace back, compare and analyze how the user behaviour changes (paths, clickstreams). At the same time the student's User Model (which has been created in another university) (UM ver.1) emerges into a new version (UM ver.2), providing the student and his supervisor the possibility to compare basic concepts of Course 2, perform reconciliation and merge the corresponding UM values (of the results that the student has achieved taking the similar course in another university), and at the same time keep the history of his studies. On the other hand the automatic concordance is also possible if only the changes made in the course are corroborated with the additional descriptive information (is accompanied by the descriptive change (see section 3.2.2)) which contains extra explanations how to reason and update values (in this case UM values) due to the evolved course structure and/or content.

The complete picture of courses updates and user (student) development becomes transparent and the course instructor may trace back all the changes done to the course, re-use, update, merge any particular part without creating anything from the scratch. He can analyze differences in and between courses.

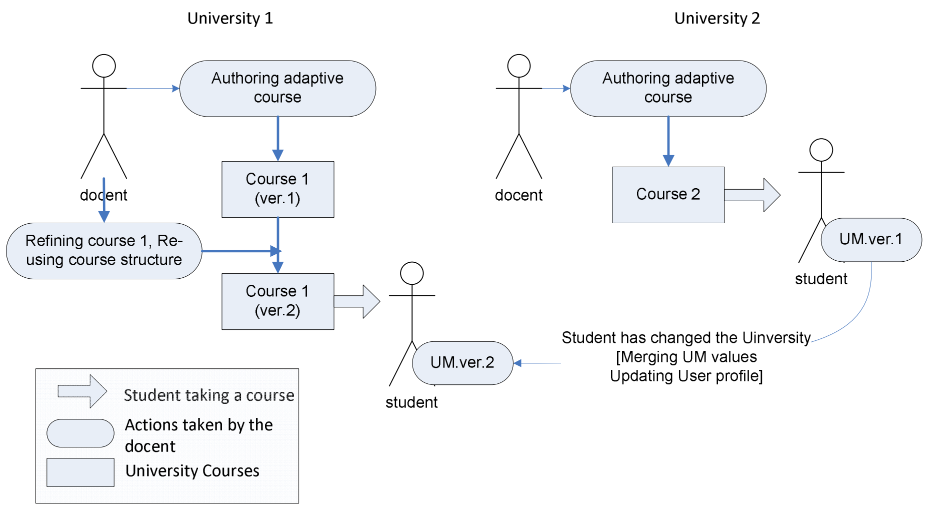
University 1
University 2
Authoring adaptive course
Authoring adaptive course
docent
docent
Course 1 (ver.1)
Course 2
Refining course 1, Re-using course structure
Course 1 (ver.2)
student
UM.ver.1
student
UM.ver.2
Student has changed the Uinversity [Merging UM values Updating User profile]
Student taking a course
Actions taken by the docent
University Courses

Fig. 3 Versioning adaptive courses

4.2 Recommender system use-case

In our second use-case we would like to outline the advantages of applying revision control in recommender systems. Typically, comparing a UM state to some reference characteristics, and predicting the user's choice, the system evolves from the initial user profile. Each recommendation step (viewing an item, ranking it and getting recommendations) corresponds to a change in the User Model, which in a conventional system is committed to the latest state not taking into account user interactions and updates. Considering a versioning structure, we can store the difference in UM between two commits, thus logging user behaviour, and annotating it with the action (ranking) done by the user at each step. Later such a pattern of successful behaviour can serve as a recommendation to the other users providing the provenance of each recommendation step (how was this recommended and when, in what context, what was the user model state at that point in time and what has influenced the recommendation). The same happens with the recommended items, which evolve, can be modified and in general can have their own history of ratings.

If the user moves from one system to another, he will face a problem of the user profile alignment (like in the previous use-case sec. 4.1), where the history of ratings that he made may help to resolve conflicts. On the other hand labeling and tagging of user and system behaviour may become a basis for OLAP analysis. Building an OLAP cube where numeric facts (or measures), categorized by dimensions can have a set of labels or associated metadata (which is essentially corresponding to the labels used in the versioning system) will allow to organize everything in a form of multidimensional conceptual view and apply 'slice and dice' operation as the most essential analysis tool in this respect, instead of simple system or user logging and then applying complicated and time-consuming extraction algorithms. This will facilitate quick extraction of rating information of a particular date or timeframe, or for example the set of users who rated a certain product a year ago. The same approach can be undertaken to version rated items in order to trace back the changes and associated ratings (as well as corresponding UM values) if the product or its description or properties change. In figure 4 we briefly summarize versioning aspects of the recommendation use-case.

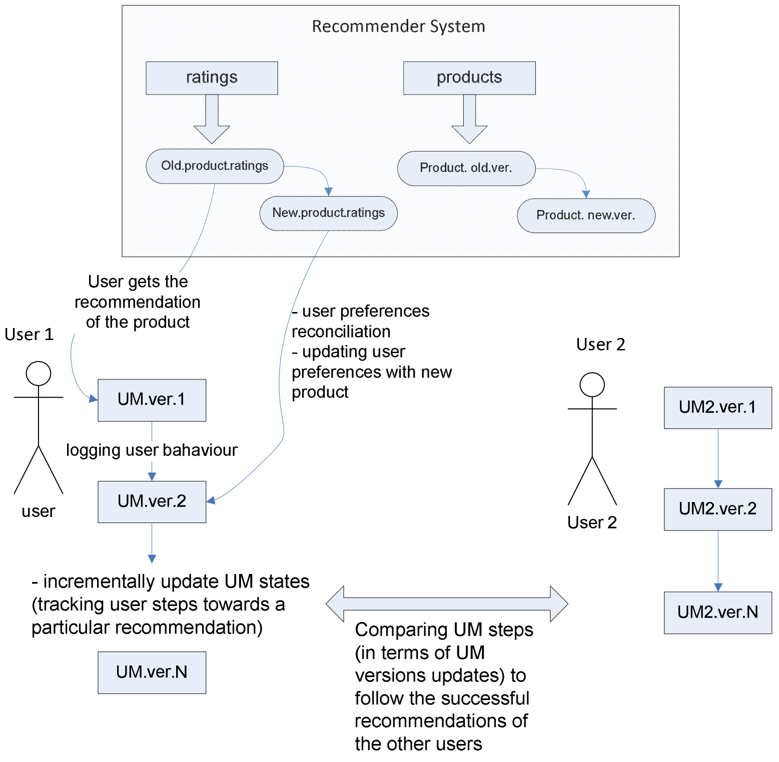
Recommender System
ratings
products
Old.product.ratings
New.product.ratings
Product. old.ver.
Product. new.ver.
User gets the recommendation of the product
- user preferences reconciliation
- updating user preferences with new product
User 1
user
UM.ver.1
logging user bahaviour
UM.ver.2
- incrementally update UM states (tracking user steps towards a particular recommendation)
UM.ver.N
Comparing UM steps (in terms of UM versions updates) to follow the successful recommendations of the other users
User 2
User 2
UM2.ver.1
UM2.ver.2
UM2.ver.N

Fig. 4 Versioning in Recommender system

To conclude this use-case we would like to outline the ideas and principles used in a context of Recommender System. A new modification of product in the stock can be designated by a new version, it inherits corresponding properties and ratings. Versioning makes possible for products (product concurrent versions) to co-exist. It also helps to store product updates and a context of these changes efficiently. Thus we can create concurrent instances of products and ratings within different applications, inherit ratings, properties and merge these changes. The User Model versioning allows to analyze user step-by-step behavior (incl. user trend, etc.), compare changes with other users in order to provide similar recommendation patterns in a case of successful outcome.

4.3 Use-cases conclusions:
For the aforementioned use-cases we would like to conclude with a summary of approaches we suggested in the introduction and outline the advantages of versioning:
Authoring – versioning helps to create, maintain and re-use concurrent versions of an application, model or a particular property and value (e.g. course and a corresponding content), saving authoring effort.
Storing – versioning offers an efficient way to store changes, annotate them, label, present in a hierarchical structure. This saves space for a large scale system and keep track of the system history.
Maintenance – structured changes and a set of basic concepts and operations (merge, resolve, commit, tag/label, head and branch) are sufficient to maintain and reconcile application conflicts, create concurrent versions or comprise functionality implemented in different systems in one application (merging changes from different branches).
Logging – logging incremental changes (of the application, user model or a context) provides playback possibilities and serves as a ground of system analysis. Logging in terms of UM updates (or steps taken) will provide a basis for comparison of behaviour patterns and advance provided recommendations.
Analysis – versioning facilitates analysis of the step-by-step system and user behaviour to identify trends; tagging and labeling can facilitate more complex analytical approaches (such as OLAP) providing the required structure and description of the data; hierarchical incremental logging results in a clear and transparent structure of system changes and an overall evolution picture.

5 Conclusions and Further Work

In this article we tried to map a conventional versioning approach onto the field of Adaptive Hypermedia, providing clear parallels in terms of versioning concepts and operations. We tried to make first steps in this direction in order to show advantages of this approach and outline the perspective of applying these techniques and methodologies in order to facilitate the transparency of AHS evolution through versioning.

Considering further work directions we can think of describing layers of a generic adaptation framework in a versioning context (or essentially looking at the basics of adaptation through versioning), investigate technologies (e.g. source control, historical data bases, etc.) that meet the requirements of the framework and come up with the detailed structure and operations to devise these versioning facilities.

Acknowledgements. This research is supported by NWO through the “GAF: Generic Adaptation Framework” project.

References

1. Altmanninger, K., Gerti Kappel G., Kusel, A., Retschitzegger, W., Schwinger W., Seidl, M., Wimmer, M., AMOR – Towards Adaptable Models Versioning. In 1st International Workshop on Model Co-Evolution and Consistency Management in conjunction with Models'08, 2008.
2. Klein M., Fensel D., Kiryakov A., Ognyanov D., Ontology versioning and change detection on the Web. In Proceedings of In 13th International Conference on Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web. Berlin, Heidelberg: Springer, pp. 247-259, 2002.
3. Brusilovsky, P., Methods and techniques of adaptive hypermedia, User Modeling and User Adapted Interaction 6(2-3), pp. 87-129, 1996.
4. Heckmann, D., Ubiquitous User Modeling, PhD Thesis. Universitat des Saarlandes, 2006.
5. De Bra, P., Calvi, L., AHA! An open Adaptive Hypermedia Architecture. *Hypermedia*, 4(1), pp. 115-139, 1998.
6. Herlocker, J., Konstan, J., Terveen, L., Riedl., J., Evaluating collaborative filtering recommender systems. ACM Transaction on Information Systems, vol.22, No.1, pp.5-23, 2004.
7. De Bra, P., Houben, G.J., Wu, H., AHAM: A Dexter-based Reference Model for Adaptive Hypermedia, Proceedings of the ACM Conference on Hypertext and Hypermedia, pp. 147-156. 1999.
8. Cristea, A., De Mooij, A., LAOS: Layered WWW AHS Authoring Model and their corresponding Algebraic Operators, The Twelfth International World Wide Web Conference, Alternate Track on Education, 2003.