World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:7, No:6, 2013

Proposition for a New Approach of Version Control System Based On ECA Active Rules

S. Benhamed, S. Hocine, and D. Benhamamouch

***Abstract*—**We try to give a solution of version control for documents in web service, that's why we propose a new approach used specially for the XML documents. The new approach is applied in a centralized repository, this repository coexist with other repositories in a decentralized system. To achieve the activities of this approach in a standard model we use the ECA active rules. We also show how the Event-Condition-Action rules (ECA rules) have been incorporated as a mechanism for the version control of documents. The need to integrate ECA rules is that it provides a clear declarative semantics and induces an immediate operational realization in the system without the need for human intervention.

***Keywords*—**ECA Rule, Web service, version control system, propagation.

I. Introduction

THE version control is essential in the environments where several users handle the same base of resource where the data is shared and in continuous evolution. As the Web services do not take into account the history of the modifications of the documents exchanged between the services and their customers [1], [2], we propose to consider this aspect in order to reach the history of the modifications of XML documents [3], [4] and to have for each document the traceability of all the modifications made along of its life cycle. For this reason, we propose to incorporate a new model of version control system in a centralized work group containing a set of users and their work copies connected to a central repository and it also connect with other central repositories of works groups forming the global decentralized system [5].

The approach suggested supports the «Lock-Modify-Unlock» and «Copy-Modify-Merge» strategies [6]. Our effort is also focused on integrating the ECA active rules, for the management of ECA number of versions, the control of shared data management and propagation of changes as well as changes in the content of XML documents.

The remainder of this paper is organized as follows. Section II presents, the definition of active system and ECA rules, and Section III discusses the models of version control system. We specify, in Section IV, our model. In Section V, we present the architecture of the proposed model with a definition of its components in Section VI. Finally, we conclude in the Section VII.

S. Benhamed is with LITIO Laboratory in computer Sciences department, University of Oran, Algeria, (e-mail: benhamed2007@yahoo.fr).

S. Hocine is with LITIO Laboratory in computer Sciences department, University of Oran, Algeria, (e-mail: S_hocine@yahoo.fr).

D. Benhamamouch is with LITIO Laboratory in computer Sciences department, University of Oran, Algeria, (e-mail: d_benhamamouch@yahoo.fr).

II. Active Systems and ECA Rules

The active systems are characterized by an ability to automatically react to certain situations by the execution of predefined operations [7]-[9].This capacity of reaction consists in enriching a "*traditional*" system of data bases by active rules. The active rules describe the operations which must be performed in response to a significant and not at the explicit request of an application or a user. The active rules generalize the notion of Triggers in the context of databases relational data [10]. These triggers was used to guarantee the integrity and the coherence of the data bases [11], [12] where the action generally consisted in rejecting the operation of update in the event of violation of constraints.

The ECA active rules express the active functionalities through a formalism of rule "*Event-Condition-Action*". An event indicates a point in time when the system has be reacting, and a condition relates to a system state; it has to be evaluated when the occurrence of corresponding event is signaled. If the condition is satisfied, the associated action has to be executed. The events are subdivided into two categories: Primitives which correspond to elementary occurrences, and composite events which are composed out of other composite or primitive events and built by means of event constructors (conjunction, disjunction, sequence, negation…) [12], [13], and [9].

III. Version Control System (VCS)

The Version Control System (VCS) is software which contains a set of tools to store and manipulate a file as well as any revisions it has undergone since its creation [14]. The utility of VCS is seen in the competing access by several developers, the follow-up of the history, and the visualization of the differences between the various versions and the return to old versions [15] and [16]. Some versions of VCS are centralized, and other decentralized. We cannot use these systems on web service because they are limited by their platform.

In a centralized version control system, there's a single repository that track the changes to all files of the project and any version control operations (check out, commit, merges…) must go through the central repository [17]. The decentralized version control systems, as the name implies, can have any

number of repository. Each user has own repository and changes can be swapped back and forth between repositories arbitrarily [18].

The centralized system used «Lock-Modify-Unlock» and «Copy-Modify-Merge » model, on the way to avoid users the sharing of information and remove of changes passed. In «Lock-Modify-Unlock» model, only one user is authorized to modify a document at a given moment. This technique is managed by the Lock. If a user locks a document, no change could be established by another user. The access to the locked document by another user will be possible once the document is unlocked.

In «Copy-Modify-Merge» model, the users contact the repository of the project, and create a personal working copy, which is a local replication of the structure and files of the repository. The users can work in parallel, by modifying directly and only their private copy. Then, the private copies are merging together into a new final version.

Although, «Lock-Modify-Unlock» strategy guarantees the user to work on the latest version of a document; the problem with this strategy is that it's restrictive and can become read block for users locking may cause administrative problems unnecessary serialization and create a false sense of security [19]. However, the «Copy-Modify-Merge» model allows avoiding the need for waiting, to have an absence of conflict in the case where there are modifications on the same document, and to solve a conflict is often faster than to await the unlocking of the first model [20].

The locking imposed in «Lock-Modify-Unlock» model cause a delay and a waste of time when the user ignores the Unlock of the document or when he manipulates a document part independent of other concurrent users. But the merger, in the «Copy-Modify-Merge» model, it can cause a loss of time when the users try to resolve conflicts manually.

The centralized control systems support CVS [21] and Subversion [22] as well decentralized control systems [18], [23], and [24] support Git, Bazaar, Darcs, Mercurial and Monotone [25].

A. Centralized Control Version System

A centralized control version system uses a single central repository distant access made to share information moreover modification impose an access in writing on this repository [17]. The centralized control version Systems are secure but in the other hand, the exchanges between the repository and the local copies are impossible, the work without connection is impossible, the time of update is long for the big projects and if the server breaks down the management is stopped.

B. Decentralized Control Version System

In decentralized control version system several repository coexist requiring synchronization. The objective of the decentralized control version Systems is to solve the problems of the systems centralized, like to be able to use the manager of version without connection and to give a possibility to exchange files with part of the developers, since each developer has its own repository.

It is also possible to create its own repository from another, to establish out its modifications locally, after that to recover the later changes since the source, to propagate its own modifications if one has an access in writing.

IV. The Proposed Model

To avoid the maximum of problems of «Lock-Modify-Unlock» and «Copy-Modify-Merge» with keeping the advantage provided by these models, we propose a new approach of version control intended for the information exchanged in the Web services between providers and their customers. This approach is not intended to manage the various versions of the services of the Web services, thing which under development but currently not introduced yet into the operation of the Web service, it aims at managing different versions of the documents, which are used in the Web services. Because the nature of the exchanged data, we are interested to manage and control of the documents which are in XML form.

The mechanism for version control of document in our model is inspired by «Lock-Modify-Unlock» and «Copy-Modify-Merge» models, in a centralized system. Because we benefit, of advantages of the «Copy-Modify-Merge» model and it must to recognize that, sometimes, the locking is essential in the version control systems. The model proposes to keep a history of the various versions of the documents which the system has, the return to an unspecified former version and to keep a history of document's changes, their date, and their user. To make this management active, we incorporate ECA active rules in our model to execute necessary functionalities for the reason that it provides a flexible architecture that can be adapted for different application scenarios, as well as providing an active service as composition of other elementary services.

The global figure below (Fig. 1) shows the interlacing of the central repository in the global distributed system.

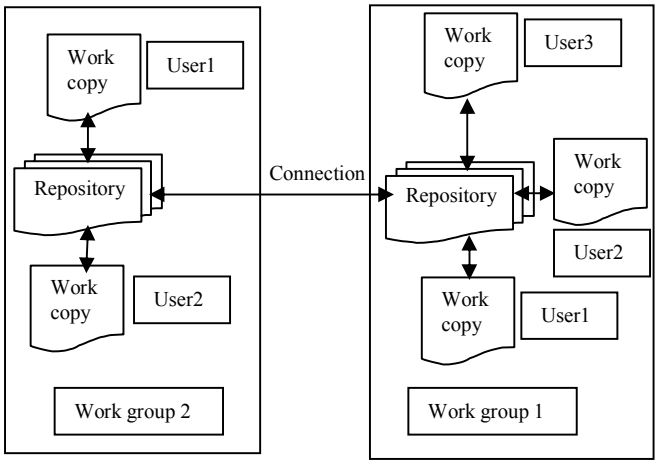

Fig. 1 The Global Diagram system

V. The Architecture of Proposed Model

To answer the objectives of proposed model of versions

control for documents, we present in the figure below (Fig. 2) the architecture of reference of our model.

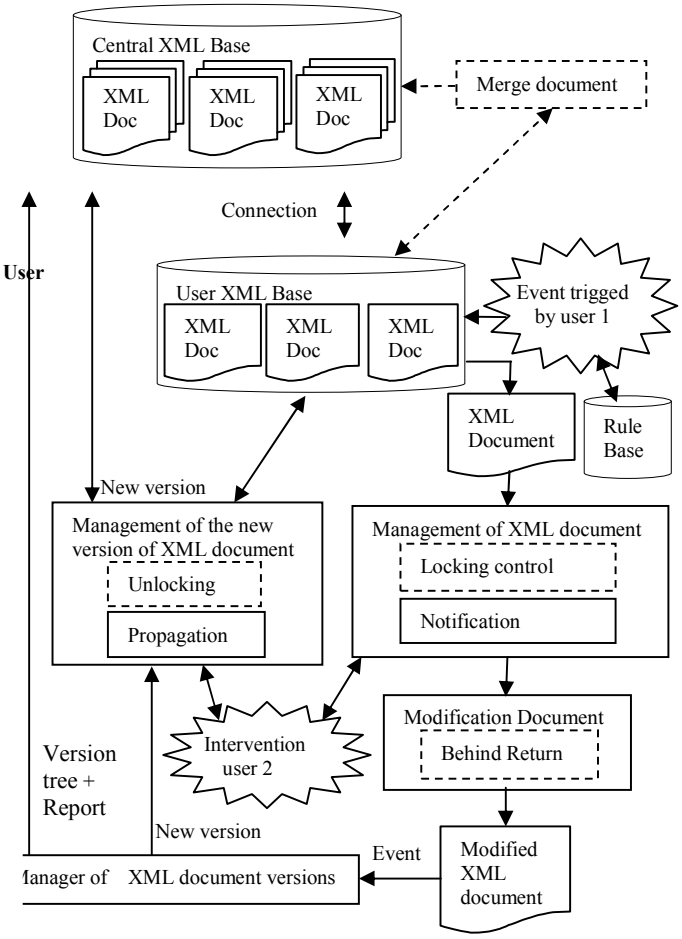

Fig. 2 The architecture of versions control based on ECA active systems

In the web services environment, the central server is the service provider and the users are the customers.

VI. Components of the Model

A. Central XML Base / User XML Base

The central XML base (repository) contains all the documents with their various versions in repertories form. Concerning the user XML base, it represents an environment of work for each user (creation of a personal working copy as in the «Copy-Modify-Merge» model). It contains only the last version of each document which the user needs in his work. The connection between central XML base and that of the user, is established when the user wants to consult the contents of a former version of a document, or if the user wishes to establish out modifications.

B. Management of XML Document

The module of management XML document is able to manage the selected document; it composes two principal components which intervene at the same time: The Notification component emits a warning with the user informing it, of change document specifying his name.

Five situations are possible and are expressed by the following ECA rules:

```
ECA rule 1: Event: Accept the modification
by user 2
            Condition: If opinion=accept
            Action: do the modification
ECA rule 2: Event: Stopping by user 2;
acceptance for the stop of user 1
            Condition: true
            Action: Cancel the modification
                    and stop the execution
                    of the system
ECA rule 3: Event: Blocking by user 2;
               acceptance for the
               blocking of user 1
            Condition: true
            Action:      Trigger      the
                    modification of user 2,
                    after   trigger   the
                    modification of user1
ECA rule 4: Event: Blocking by user 2;
refuse blocking of user 2 by user 1
             Condition: true
             Action:      Trigger      the
                    modification of user 1,
                    after  triggering  the
                    modification of user 2
ECA rule 5: Event: Stopping by user 2;
refuse stopping of user 2 by user 1
              Condition: true
              Action:     Trigger     the
                    modification of user 1
```

NOTE: The ";" denote the event constructor of sequence.

The diagram of sequence (Fig. 3) illustrates the different exchanges between both users.

NOTE: Before beginning the locking and the notification we check if the second user is connected, if it is not, we pass directly at the following step (modification of the document) without the locking and the notification modules.

The aim of locking control is to lock the access to the document which made modifications by the two users and when the second user ignores the message of Notification component. When the latter wants to reach the document and triggers modifications though it is not the last version, the locking of the document is necessary; the technique of adopted locking consists in adding information to the action like an attribute noted «density». The density informs us about the impact of the changes established in the documents: if the change is regarded as being partial; locking is not necessary on the other hand, the lock is essential if the change is total.

The density can have the true value or false according to the type of the action.

If density = true then:

- The actions of ECA rule apply total changes to the document.
- It is necessary to activate the locking of document

for the other users.

-

If density = false then:

- The actions of ECA rule apply partial changes to the document XML (insertion or suppression of a part of the document).
- It is not necessary to activate the locking of the document for the other users (authorize the users to make their modifications). The type of the actions established by both rules trigged by the users must be in the same type.

These situations are expressed by the fooling ECA rules:

```
ECA Rule 6: Event: selected document;
            modification rule started
        Condition: If density=true
        Action: Activate the locking
ECA Rule 7: Event: selected document;
            modification rule started
        Condition: If density=false
        Action: Don't activate the
        locking
```

NOTE: The actions in the same type mean that if the first user has established an insertion (respectively the suppression), the second will establish also an insertion (respectively the suppression) in the same document.

When user 1 informs the user 2 that he wishes to establish a modification on a document, then user 2 must select one of three choices. User 2 can accepts the modification and the modification is then establishes, or stops the modification and in this case the modification trigged by user 1 is stopped; else he can block the modification of user 1 and at this moment he must triggers a modification on the document while user 1 is blocked, and when he finishes, user 1 can establish a modification on the document.

When the both users want to modify the same document in the same time and the locking is not activate, the system call the module Merge document of the server to merge the results of the users modifications.

A. Modification Document

The module of Modification Document is conceived like a component making it possible to evaluate and trigger the ECA rule which was started by event from Rule Base. As it also makes it possible to recover a former version without activate ECA rule with the component of behind Return. The product of this module is a modified XML document.

B. Manager of XML Document Versions

The module of Manager of XML document versions treats and manages the various versions of the document in addition to the new version. Thus this module is responsible for document revision, and the creation of the tree of versions (realized automatically by the mechanism of ECA rules).

To visualize the maximum of information starting from the version numbering, the Manager of XML document versions adopt «1.X.Y» numbering where 1 represents root version, X is user number who caused the creation of this new version, and Y is number version, which is incremented after each creation of a new version by each users (1.0.0 represents the number of the first version of a document).

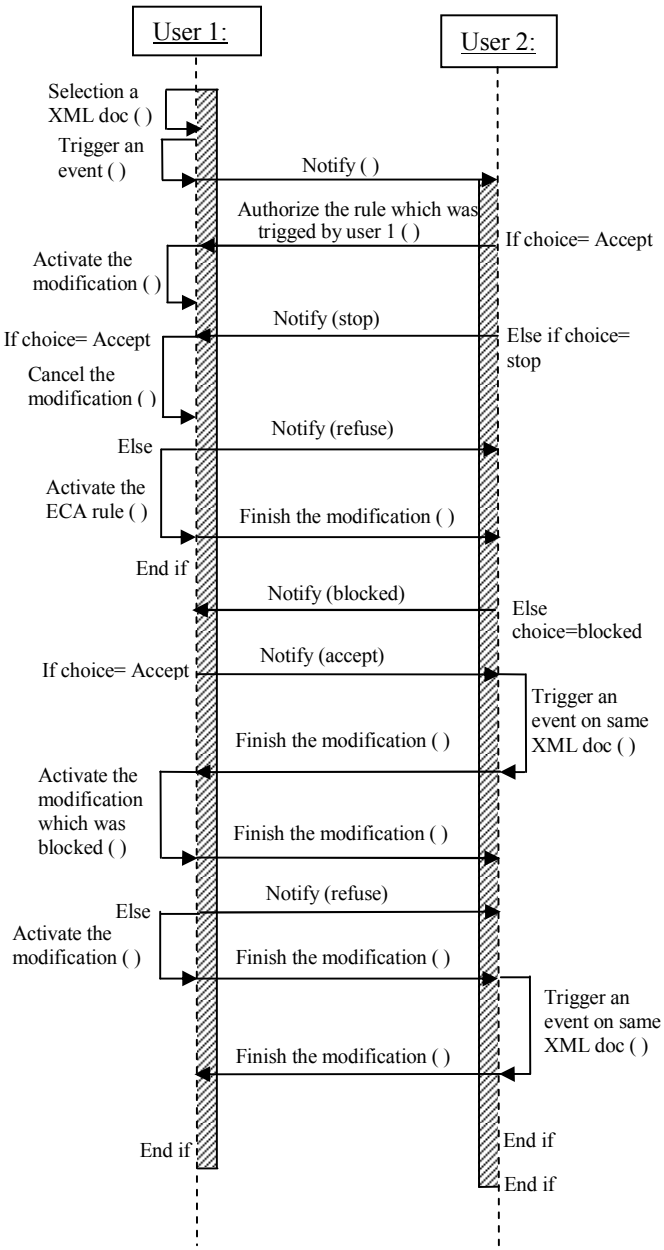

Fig. 3 Diagram of sequence of Notification component

Once the numbering of the document finished this module is responsible to create and manage the tree of versions.

However the ECA Rules introduced into this module are defined below:

```
ECA Rule 8: Event: Original document;
presence of a modified document.
          Condition: If the version tree
          exists
          Action: 1. Traverse version tree
                  2. Affect a number for the
                  new version
```

```
                    3. Add a node to version
                    tree
ECA Rule 9: Event: No original document;
                  presence of a modified
                  document
             Condition: If the version tree
                        does not exist
           Action: 1. Create version tree
                   2. Affect a number for
                    the new version
                   3. Add a node to the
                   version tree
```

The version tree is a graphic representation of central XML base. It illustrates which is the document that has the modifications to obtain the last version. It makes possible to keep the history of all established changes in the system. The version tree is stored in central XML Base.

The creation of a new version implies the creation of a report whose name and number are those of the document version, it is necessary to store the operations of these modifications. The report is inserted automatically in central XML base.

C. Management of the New Version of XML Document

Once the new version created, the management of the new version module calls two components Propagation and unlocking which each one integrates a system containing ECA rule. In order to give to the users' access to the document, the Unlocking component releases the lock (case where the document was locked). The propagation module ensures that users have the new version; for that it propagates the new version of the document modified to central XML base and removes the old version of user XML base.

VII. Conclusion

In this paper, we covered the versions control of XML documents, which is important for the management of documents. We presented a model of versions control for XML documents. This model combines the techniques of «Lock-Modify-Unlock» and «Copy-Modify-Merge» models. One aspects of our approach is the use of ECA rules especially in the document's modification, the management of version numbering and the propagation of versions of documents.

References

[1] W3C. Web Services Description Language (WSDL) 1.1, note 15, 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[2] C.Devaux, L. Bourceret, B. Gory and L. Bernard "Urbanisation & Architecture Oriontée Service (SOA), Quelques bonnes pratiques pour leur mise en œuvre," 2008.

[3] D. Hunter, C. Cagle, N. Ozu, J. Pinnock and P. Spencer. "*Initiation à XML*," Eyrolles Edition," 2001.

[4] A. Brillant, "*XML cours et exercices*," Eyrolles Edition. 2007.

[5] Sparx systems. Version Control Best Practices for Enterprise Architect, 2010. www.sparxsystems.com.

[6] B. Collins-Sussman, W. Brian, C. Fitzpatrick and M. Pilato "Version Control with Subversion For Subversion 1.6," 2009.*Compiled from Revision3734*.

[7] U. Dayal, "Active Database Management Systems", *ACM Sigmod Record*, 18(3) :150–169. 1989

[8] T. Coupaye, C. Collet, "Modèles de comportement des SGBD actifs: caractérisation et comparaison," 1998. *Technique et Science Informatiques (TSI)*, 17(3) :299-328.

[9] T. Coupaye, C. Collet, "Primitive and composite event in NAOS," Cassis France. 1996.

[10] J. Bailey, G. Dong, RAMAMOHANARAO, K., "On the Decidability of the Termination Problem of Active Database Systems," 2004. *Theoretical Computer Science*, p. 389-43.

[11] U. Dayal, E. HANSON, J. WIDOM, "Active Database Systems. W.Kim editor, Modern Database Systems," 1995. pp. 434-456. *ACM Press*.

[12] C. Collet, "Bases de données Actives: des systèmes relationnels aux systèmes à objets," *Laboratoire IMAG*. 1996. Report RR965-ILSR4.

[13] S. Gatzia and K. R.Dittrich, "Detecting Composite Events in Active Database Systems Using Petri Nets," *in Proc. of the 4th Intl. workshop on Research Issues in Data Engineering (RIDE'94)*, Houston, Texas. 1994.

[14] A. F. Bresson, Gestion de contenu Web. http://www.axidea.org/form_info.htm e 17 january 2006.

[15] A. Cadiou, "Introduction à CVS : un système de gestion de version," *LMFA UMR CNRS 5509, FLCHP*, Central School of Lyon. 2004.

[16] A. Alvarez Escobedo, " SAGED-XML : Serveur actif pour la gestion de la cohérence de documents," Appliqued Scientific national Institut of Lyon. 2003.

[17] M. Guesdon and G. Rousse, "Gestion de configuration avec CVS et Subversion," 2011.

[18] S. Chacon, " *Pro Git (Expert's Voice in Software Development)* ," Apress. 2010.

[19] B. Collins-Sussman, W. Brian, C. Fitzpatrick, and M. Pilato, "*Version Control with Subversion for Subversion*," O'Reilly Media edition. 2004.

[20] F. Melot, " SVN un gestionnaire de versions," *SARI Seminar*, LPSC, IN2P3, CNRS Grenoble. 2008.

[21] P. Cederqvist et al. "Version Management with CVS," Free Software Foundation, Inc. 2008.

[22] D. Donsez, "SubVersion (SVN)," University of Joseph Fourier Grenoble I. 2010.

[23] B. Lynn, "*Git Magique Historique des versions*," BL. 2007.

[24] T. Swicegood, "*Pragmatic Guide to Git*," The Pragmatic Bookshelf, Texas. 2010.

[25] H. Graydon, S. Nathaniel, D. Scherger, D. Carosone, J. Pellegrini, A. Queiroz, W. Uther, T. Keller, and S. Leake, "Monotone A distributed version control system," version 1.0. 2011.