Tool For Collaborative Temporal-Based Software Version Management

Mohd Nordin Abdul Rahman[a], **Mohd Isa Awang**[a], **Aziz Deraman**[b], **Amir Ngah**[c]

[a]*Information Technology Center*
Kolej Ugama Sultan Zainal Abidin, 21300 Kuala Terengganu, MALAYSIA
Tel : 609-6653214, Fax : 609-6662566, E-mail : mohdnabd@kusza.edu.my

[b]*Computer Science Department*
Universiti Kebangsaan Malaysia, 43600 UKM Bangi, MALAYSIA
E-mail: ad@pknet.cc.ukm.my

[c]*Computer Science Department*
Kolej Universiti Sains dan Teknologi Malaysia, 21030 Kuala Terengganu, MALAYSIA
Tel: 609-6683359, Fax : 609-6694660, Email: amirnma@kustem.edu.my

ABSTRACT

Software version management is the processes of identifying and keeping track of different versions of a software. Complexity level of this process would become complicated should software was distributed in many places. This paper present a new dimension in software version management which based on temporal elements. Temporal elements such as valid time and transaction time are the main attributes considered, to be inserted into the software version management database. By having these two attributes, it would help the people involved in software process to organize data and perform activity monitoring with more efficient. For a practical application of the model, therefore an automate tool has been developed that could be applied under collaborative software process called TEMVer.

Keywords

Software version management, temporal database, valid time, transaction time

1.0 INTRODUCTION

Software evolution is concerned with modifying software once it is delivered to a customer. Software managers must devise a systematic procedure to ensure that different versions of a software may be retrieved when required and are not accidentally changed. Controlling the development of different versions of a software can be a complex task, even for a single author to handle. This task is likely to become more complex as the number of software authors increases, and more complex still if those software authors are distributed geographically with only limited means of communication, such as electronic mail, to connect them.

Temporal based data management has been a hot topic in the database research community since the last couple of decades. Due to this effort, a large insfrastructure such as data models, query languages and index structures has been developed for the management of data involving time (Knight, 1994a). Presently, many information systems has adopted the concepts of temporal database management such as geographic information systems and artificial intelligence systems. Temporal management aspects of any objects transaction data could include:

- The capability to detect change such as the amount of change in a specific project or object over a certain period of time.
- The use of data to conduct analysis of past events e.g., the change of valid time for the project or version due to any event.
- To keep track of all the transactions status on the project or object life cycle.

In this paper, we will concentrate on the technique to improve a version management and monitoring model by using temporal elements such as valid time, transaction time and temporal operators. Finally, we see how a web-based software version management (TEMVer) fits within the framework of our model.

1.1 Temporal Data Review

To date, two well-known of time that are usually considered in the literature of temporal database are *transaction time* and *valid time* (Knight and Ma, 1994a; Jensen and Snodgrass, 1999; Ling and Bell, 1992; Gregerson and Jensen, 1999). The valid time of a database fact is the time when the fact is true in the *miniworld* (Jensen and Snodgrass, 1999; Gregerson and Jensen, 1999). In other words, valid time concerns the evaluation of data with respect to the application reality that data describe. Valid time can be represented with single chronon identifiers (e.g., event time-stamps), with intervals (e.g., as interval time-stamps), or as valid time elements, which are finite sets of intervals (Jensen and Snodgrass, 1999). Meanwhile, the transaction time of a database fact is the time when the fact is current in the database and may be retrieved (Jensen and Snodgrass, 1999; Gregerson and Jensen, 1999). This means, that the transaction time is the evaluation time of data with respect to the system where data are stored. Supporting transaction time is necessary when one would like to *roll back* the state of the database to a previous point in the time. (Jensen and Snodgrass, 1999) proposes four implicit times could be taken out from valid time and transaction time:

- valid time – valid-from and valid-to
- transaction time – transaction-start and transaction-stop

Temporal information can be classified into two divisions; absolute temporal and relative temporal (Jensen and Snodgrass, 1999). Most of the research in temporal databases concentrated on temporal models with absolute temporal information (Koubarakis, 1993). To extend the scope of temporal dimension, (Knight and Ma, 1994b; Kaobarakis, 1993) have present a model which allows relative temporal information e.g., "event A happened before event B and after January 01, 2003". (Knight and Ma, 1994b; Kaobarakis, 1993) suggests several temporal operators that could be used in describing the relative temporal information: {equal, before, after, meets, overlaps, starts, during, finishes, finished-by, contains, started-by, overlapped-by, met-by and after}.

In various temporal research papers the theory of time-element can be divided into two categories: *intervals* and *points* (Knight and Ma, 1994a; Jensen and Snodgrass, 1999; Ling and Bell, 1992; Gregorson and Jensen, 1999). If T is denoted a nonempty set of time-elements and d is denoted a function from T to $R+$, the set of nonnegative real numbers then:

$$\text{time-element, t,} = \begin{cases} \textit{interval}\text{, if } d(\text{t}) > 0 \\ \text{point, otherwise} \end{cases}$$

According to this classification, the set of time-elements, T, may be expressed as $T = I \cup P$, where I is the set of intervals and P is the set of points.

1.2 Previous Work in Version Management

In distributed software process, a good version management combines systematic procedures and automate tools to manage different versions in many locations. Most of the methods of version naming use a numeric structure (Dix et. al., 1997). Identifying versions of the system appears to be straightforward. The first version and release of a system is simply called 1.0, subsequent versions are 1.1, 1.2 and so on. Meanwhile, (Clemm, 1989) suggests that every new version produced should be placed in a different directory or location from the old version. Therefore, the version accessing process would be easier and effective. Besides that, should this method be implemented using a suitable database management system, the concept of *lock access* could be used to prevent the occurrence of overlapping process. Present, there are many software evolution management tools available in market. Selected tools will be discribed follows.

1.2.1 Software Release Manager (SRM)

SRM is a free software and supported on most UNIX and LINUX platforms. It supports the version of a systems for distributed organizations. In particular, SRM tracks dependency information to automate and optimize the retrieval of systems components as well as versions.

1.2.2 Revision Control System (RCS)

RCS using the concepts of tree structures. Each *branch* in the tree represents a variant of the version. These branches will be numbered by an entering sequence into a system database. RCS records details of any transaction made such as the author, date and reason for the updating.

1.2.3 Change and Configuration Control (CCC)

CCC is one of the complete tool for software configuration management. It provides a good platform for an identification, change control and status accounting. CCC allows a simultaneously working for a same version via virtual copies. This can be merged and changes can be applied across configurations.

1.2.4 V-Web

V-Web, developed by (Dix et al., 1997) is a web-based software version management tool. Each version to be recorded will be identified based on certain characteristics defined. This tool is a very simple and most suitable to apply in a cooperatives software development.

1.2.5 Software Management System (SMS)

SMS allows all the aspects in software configuration management such as version control, workspace management, system modelling, derived object management, change detection in the repository etc. SMS possesses the desired characteristics, providing resources of version control of systems and having a good user interface.

1.2.6 Adele

This software configuration management tool has been developed in University of Grenoble. It has a good basic features in software configuration management such as version hierarchical modelling, worksapce control and change request management. The Adele database is an Entity Relationship one that provides for the defining of configuration items.

1.3 Problem and Issues in Version Management

From study done by the authors, several weaknesses has been found in the several current version management model. These weaknesses are as follows:

- Non systematic in the procedure of version management and it is difficult to recognize the valid time for each version.
- Many models do not consider the aspect of relative temporal in representing the valid time for each version.
- Most of the existing models maintain only the concept of *current view* version of which an existing version will be overwritten by a new incoming version during the process of an update.

The World Wide Web provides a useful infrastructure for collaborative software process, the main benefits being its prevalence, platform independence and familiarity. Besides, the savings in time and paper work can be significant, when using the web as an version management platform. Therefore, by means of these benefits we are strongly believe that the development of temporal-based software version management tool which is can be applied under Internet environment could gain the following benefits:

- To increase the effectiveness and efficiency of the collaborative software version management process
- To support project and software managers in planning, managing and evaluating version management.
- Assigning timestamps (absolute and relative) to each transactions will provide transaction-time database functionality, meaning to retain all previously current database state and making them available for time-based queries.

2.0 TEMVer MODEL

Great advances have been achieved in collaborative software engineering process during the last decade. From the application point of view, there is a growing demand for web-based systems that can manage and coordinate the dynamic information of software artifacts and its hierarchical of change.

2.1 The Architecture

The tool architecture employs the concept of client-server model with a central server and the user stations connected by a network. It is an Internet application, open user interface and can be accessed simply by a web browser such as Microsoft Internet Explorer 5 (IE5). All attributes from version management processes are reported using a simple dialogue, and after submission they are stored in a centralized database, thus allowing their distribution to all software development stakeholders as well as the software users.

Regarding to the effective implementation of the applications accessed via Internet, the tool architecture is formed upon the two-tier distributed application concept which is built in two levels: (1) database management systems and (2) active server page (ASP) components and interface. The general concept of the two-tier application is presented in Figure 1. By using of this concept, hopefully this collaborative temporal based software version management tool led to the following features:

- Encapsulation of data processing
- Minimal software requirements for client workstations
- Multi-access
- Effective communication in distributed project team or software users
- Open user interface

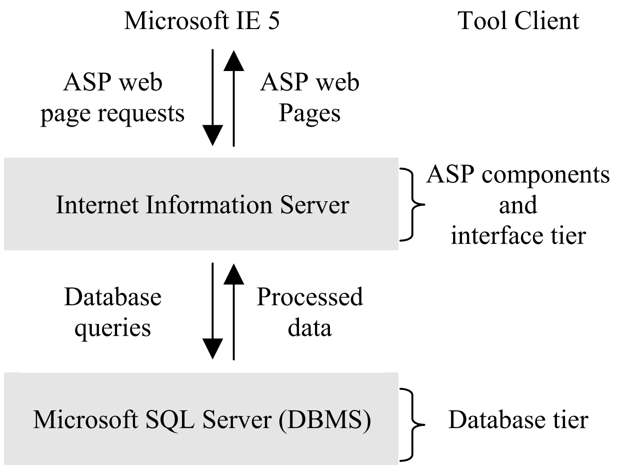


Figure 1: Two-tier software architecture

2.2 Temporal Elements In Software Version Management

Temporal elements involved in TEMVer model are transaction time (tt), absolute valid time (avt) and relative valid time (rvt) which can be denoted as, TE = {tt, avt, rvt}. Transaction time is a date-stamping and it represent a transaction when a new valid time for a version is recorded into the TEMVer database. Absolute valid time is represent by two different attributes known as valid-from and valid-until and it also using an approach of date-stamping. Meanwhile, relative valid time which involves a time interval, will be represented by a combination of temporal operators, OPERATORs = $\{op_1, op_2, op_3, \ldots, op_n\}$ and one or more defined event(s), signed as EVENTs = $\{event_1, event_2, event_3, \ldots, event_n\}$. In this model, only five temporal operators will be considered, hence will be denoted as OPERATORs = {*equal, before, after, meets, met_by*}. Figure 2 shows in general the insertion point of valid time and transaction time into software version management.

Therefore, if we have a software with a set of version signed as, $V = \{v_1, v_2, v_3, \ldots, v_n\}$ then the model is:

$$\text{TEMPORAL}(v_i \in V) \subseteq (tt \cap avt \cap rvt)$$

where, avt = [avt-from, avt-until],
rvt = [rvt-from, rvt-until],
rvt-from = $\{\{op_i \in \text{OPERATORs}\} \cap \{event_i \in \text{EVENTs}\}\}$ and

rvt-until = $\{\{op_i \in OPERATORs\} \cap \{event_i \in EVENTs\}\}$.

Thus, if a software which has a set of feature attributes A_i then a complete scheme for a temporal based in software version management can be signed as:

$S = \{A_1, A_2, A_3, \ldots, A_n$, tt, avt-from, avt-until, rvt-from, rvt-until$\}$

where, A_i = attribute name,
tt $\in$ P and
avt-from, avt-until, rvt-from and rvt-until $\in$ T.

For the process of queries to that scheme, we can use the following convention to retrieve for any temporal attribute in temporal database (Kaubarakis, 1993):

- Attribut name only $\Rightarrow$ current time
- Attribute name followed by ALL $\Rightarrow$ all history
- Attribute name followed by a time point or interval $\Rightarrow$ specific time period
- Attribute name followed by RESTRICT $\Rightarrow$ specific time period designated by the condition

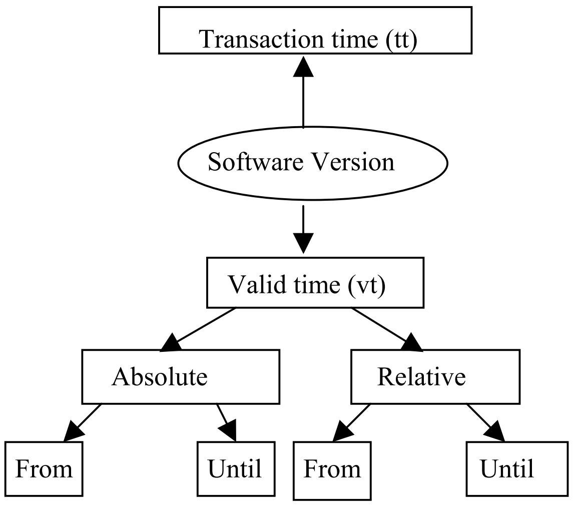


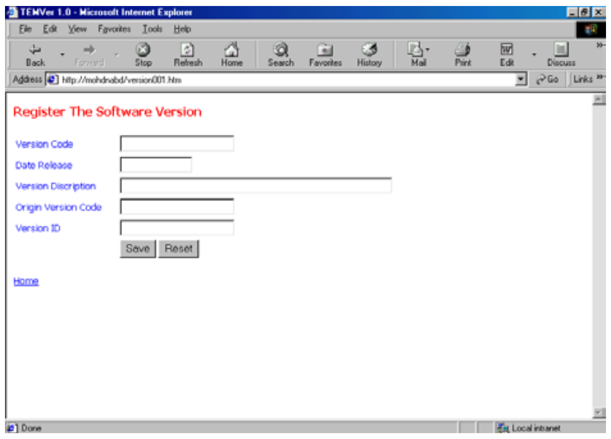
Figure 2: Temporal elements in software version management

3.0 TEMVer IMPLEMENTATION

To carry out experiments validating the concepts proposed earlier, we have designed and implemented a client-server software tool called TEMVer. There are 3 main modules supported in TEMVer and could be denoted as $\text{TEMVer}_{\text{modules}}$ = {*register version, update the version valid time, queries*}. Meanwhile, the main roles recognized as users of TEMVer could be signed as $\text{TEMVer}_{\text{users}}$ = {*software manager, software engineer, programmer, end-users*}. In this section, we describe the details' functionallity of these modules and several management issues to be considered in order to make sure TEMVer will be optimum utilized.

3.1 TEMVer Functionality

During the *register version* process the software manager needs to record the foundations information of the software version. Attributes that needed to be key-in by software manager can be signed as, A_v = {*version code, date release, version description, origin version code, version id*}. Figure 3 illustrates the screen sample used to register the basic information of the software version.



TEMVer 1.0 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://mohdnabd/version001.htm
Go Links
Done
Local intranet

Register The Software Version

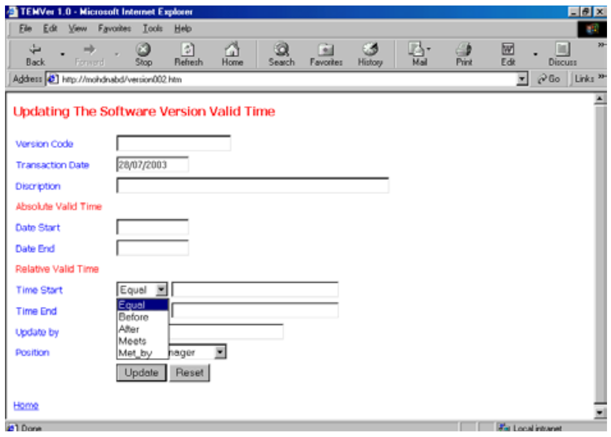
Version Code	
Date Release	
Version Discription	
Origin Version Code	
Version ID	

Save Reset

Home

Figure 3: Register the software version

On completion of new software version registration, then the software manager needs to update its valid time and this can be done by using the module update the version valid time, illustrated in Figure 4. The attributes for this module formed as A_T = {*version code, transaction date, description, date start, date end, time start, time end, update by, position*}. Attribute transaction date is the current date and will be auto-generated by the server. Any changes of a software version valid time, software manager needs to update by using this form. TEMVer also allows the user to make a query to the database. The roles of the TEMVer can browse the version valid time and status for any registered software (Figure 5). Meanwhile, Figure 6 shows the output form of query for all histories of valid time and status for a software version.



TEMVer 1.0 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://mohdnabd/version002.htm
Go Links
Equal
Before
After
Meets
Met_by
Done
Local intranet

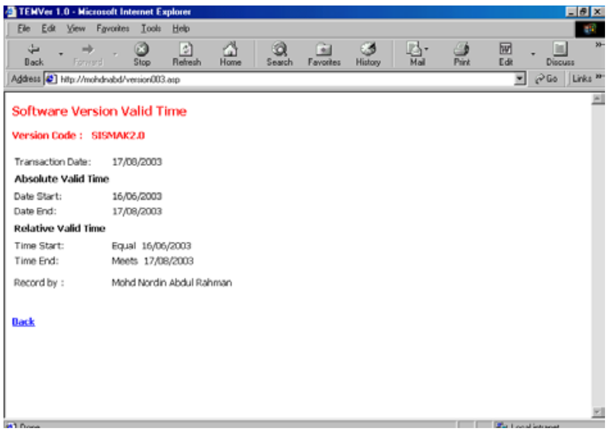
Updating The Software Version Valid Time

Version Code	
Transaction Date	28/07/2003
Discription	
Absolute Valid Time	
Date Start	
Date End	
Relative Valid Time	
Time Start	Equal
Time End	Equal
Update by	
Position	nager

Update Reset

Home

Figure 4: Update the software version valid time


TEMVer 1.0 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://mohdnabd/version003.asp

Software Version Valid Time

Version Code : SISMAK2.0

Transaction Date: 17/08/2003

Absolute Valid Time

Date Start: 16/06/2003

Date End: 17/08/2003

Relative Valid Time

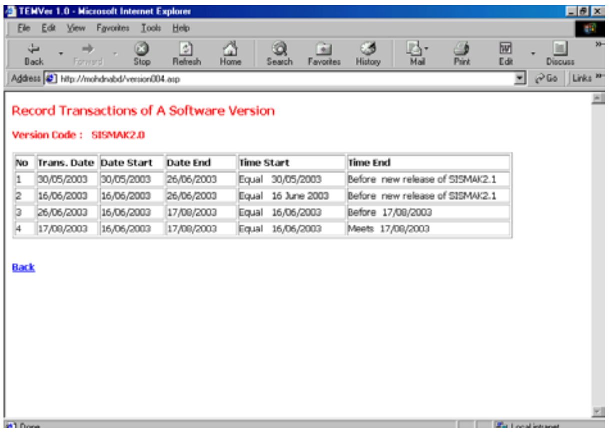
Time Start: Equal 16/06/2003

Time End: Meets 17/08/2003

Record by : Mohd Nordin Abdul Rahman

Back

Figure 5: Software version valid time report


TEMVer 1.0 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://mohdnabd/version004.asp

Record Transactions of A Software Version

Version Code : SISMAK2.0

No	Trans. Date	Date Start	Date End	Time Start	Time End
1	30/05/2003	30/05/2003	26/06/2003	Equal 30/05/2003	Before new release of SISMAK2.1
2	16/06/2003	16/06/2003	26/06/2003	Equal 16 June 2003	Before new release of SISMAK2.1
3	26/06/2003	16/06/2003	17/08/2003	Equal 16/06/2003	Before 17/08/2003
4	17/08/2003	16/06/2003	17/08/2003	Equal 16/06/2003	Meets 17/08/2003

Back

Figure 6: Transaction records of a software version

3.2 TEMVer Execution Management Issues

Distributed software evolution management is a new approach in getting a quality software process. Its does not necessarily happen automatically besides it required co-ordination, especially a large-scale project. Participants on a large-scale project found it helpful to have a strong, highly organised, proactive co-ordinator to solicit suggestions for improvements, receive opinions, draw conclusions and present them for discussion. Therefore, it is good to have at least one person from information steering committee to overseeing all processes and activities in a software evolution management in order to resolve conflicts, work flow as well as to enforce the using of the tool. An additional role in distributed software evolution management that should to be considered is Trainee. No one seems to have a clear idea how to create embedded developers. One technique is to include new folks (only one or two) into the software process management. The Trainee(s) can help the software manager to updating any software version transaction into TEMVer database.

4.0 CONCLUSIONS

In practical software version management, it is frequently important to retain a perfect record of past and current valid time for a version states. We cannot replace or overwritten the record of old valid time of a software version during the updating process. Hence, this paper introduces a new model in software version management based on temporal elements. Here, an important issue discussed is temporal aspects such as valid time and transaction time which have been stamped on each software version so that the monitoring and conflict management processes can be easily made. We hope that the material, model and web-based tool (TEMVer) presented in this paper will be useful to support future work on.

To validate of our model, TEMVer is being experimented in Kolej Ugama Sultan Zainal Abidin (KUSZA). It is used to keep track the evolution of the software version, systems module and software documents in KUSZA Information Systems (SISMAK). For further improvements, currently, we are investigating related issues including combining the model with change request management, considering more temporal operators and developing a standard temporal model for all configuration items in software configuration managements.

5.0 REFERENCES

Bertino, E., Bettini, C., Ferrari, E. and Samarati, P. (1996). *A Temporal Access Control Mechanism for database Systems*, IEEE Trans. On Knowledge And Data Eng. 8: 67-79.

Clemm, G. M. (1989). *Replacing Version Control With Job Control*, ACM – Proc. 2nd Intl. Workshop On Software Configuration Management. 162-169.

Dix, A., Rodden, T., and Sommerville, I. (1997). *Modelling Versions in Collaborative* Work. IEE – Proc. Software Engineering. 195 – 206.

Gregerson, H. and Jensen, C. S. (1999). *Temporal Entity-Relationship Models – A Survey*, IEEE Trans. On Knowledge And Data Eng. 11: 464-497.

Gustavsson, A. (1989). *Maintaining the Evaluation of Software Objects in an Integrated Environment*, ACM – Proc. 2nd Intl. Workshop On Software Configuration Management. 114-117.

Havewala, A. (1999). *The Version Control Process: How and Why it can save your project*, Dr. Dobb's Journal. 24: 100-111.

Jensen, C. S. and Snodgrass, R. T. (1999). *Temporal Data Management*, IEEE Trans. On Knowledge And Data Eng. 11: 36-44.

Knight, B. and Ma, J. (1994). *A General Temporal Theory*, The Computer Journal. 37: 114-123.

Knight, B. and Ma, J. (1994). *A Temporal Database Model Supporting Relative and Absolute Time*, The Computer Journal. 37: 588-597.

Koubarakis, M. (1993). *Representation and Querying in Temporal Databases: the Power of Temporal Constraints*, IEEE. 1063-6383/93. 327-334.

Lie, A. (1989). *Change Oriented Versioning in a Software Engineering Database*, ACM – Proc. 2nd Intl. Workshop on Software Configuration Management. 56-65.

Ling, D.H.O. and Bell, D. A. (1992). *Modelling and Managing Time in Database Systems*, The Computer Journal. 35: 332-342.

Mary, H. (1996). *Beyond Version Control*, Software Magazine. 16: 45-47.

Rahman, M. N. A., Awang, M. K., Rose, A. N. M. and Deraman, A. (2003). *A Framework for Temporal Based Version Management*, Proc. Engineering and Technology Conference.

Sarda, N. L. (1990). *Algebra and Query Language for A Historical Data Model*, The Computer Journal. 33: 11-19.