

Towards the Implementation of Temporal-Based Software Version Management at Universiti Darul Iman Malaysia

M Nordin A Rahman, Azrul Amri Jamal and W Dagang W Ali

Faculty of Informatics

Universiti Darul Iman Malaysia, KUSZA Campus

21300 K Terengganu, Malaysia

mohdnabd@udm.edu.my, azrulamri@udm.edu.my, wan@udm.edu.my

ABSTRACT

Integrated software is very important for the university to manage day-to-day operations. This integrated software is going through evolution process when changes are requested by the users and finally the new versions are created. Software version management is the process of identifying and keeping track of different versions of software. Complexity level of this process would become complicated should software was distributed in many places. This paper presents a temporal-based software version management model. The model is purposely implemented for managing software versions in Information Technology Centre, Universiti Darul Iman Malaysia. 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 monitoring activities with more efficient.

Keywords: version management, temporal database, valid time, transaction time.

1. 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 software versions may be retrieved when required and are not accidentally changed. Controlling the development of different software versions 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 infrastructure such as data models, query languages and index structures has been developed for the management of data involving time [11]. Nowadays, a number of software has adopted the concepts of temporal database management such as artificial intelligence software, geographic information systems and robotics. Temporal management aspects of any objects 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.

Universiti Darul Iman Malaysia (UDM) is the first full university at East Coast of Malaysia located at the state of Terengganu. It was setup on 1st January 2006. UDM has two campus named as KUSZA Campus and City Campus. Another new campus known as Besut Campus will be operated soon. To date, KUSZA Campus has six faculties and City Campus has three faculties. The university also has an Information Technology Centre (ITC-UDM) that purposely for developing and maintaining the university information systems and information technology infrastructure.

In this paper, we concentrate on the modelling of a temporal-based software version management. Based on the model, a simple web-based web application has been developed and suggested to be used by ITC-UDM. The rest of the paper is organized as follows: next section reviews the concept of temporal data management. Section 3 discusses on the current techniques in software version management. Current issues in software version management at ITC-UDM are discussed in Section 4. The specifications of the proposed temporal-based software version management model are explained in Section 5. Conclusion is placed in Section 6.

2. Temporal Data Concept

To date, *transaction time* and *valid time* are the two well-known of time that are usually considered in the literature of temporal database management [2, 4, 6, 9, 10, 11, 12]. The valid time of a database fact is the time when the fact is

true in the *miniworld* [2, 6, 9, 10]. 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 [9]. Meanwhile, the transaction time of a database fact is the time when the fact is current in the database and may be retrieved [2, 6, 9, 10]. 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. [9] proposed 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 [9]. Most of the research in temporal databases concentrated on temporal models with absolute temporal information. To extend the scope of temporal dimension, [12] presented a model which allows relative temporal information e.g., “event A happened before event B and after January 01, 2003”. [12] suggests several temporal operators that could be used for 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* [6, 9, 11]. 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:

$$\text{time_element}, t = \begin{cases} \text{interval}, & \text{if } d(t) > 0 \\ \text{point}, & \text{otherwise} \end{cases}$$

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.

3. Related Tools in Software 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 [5]. 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, [3] 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 are described as follows:

- *Software Release Manager (SRM)* – SRM is a free software and

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

- *Revision Control System (RCS)* – RCS uses 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.
- *Change and Configuration Control (CCC)* – CCC is one of the complete tools 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.
- *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.

4. The Software Version Management Issues in ITC-UDM

There are three divisions have been formed at ITC-UDM. These divisions and their function are as follows:

- Infrastructure and Application Systems (AIS) – to develop and maintain the university software; maintain the university computer networking;
- Technical and Services (TS) – to support the maintenance of information technology hardware, training, multimedia services and help desk.
- Administration and Procurement (AP) - to manage the daily operation of ITC-UDM such as administration, procurement etc.

Each division is headed by a division leader and supported by several information technology officers, assistant information technology officers and technicians. All the university software modules are developed and maintained by AIS Division. Figure 1 depicts the main software modules managed by the ITC-UDM. There are over thousands source code files are produced by the division. Therefore, it is not easy for the division to manage all those artefacts.

From study done by the authors, two main weaknesses have been found in the current approach for ITC-UDM in managing all versions of source codes produced:

- Non systematic procedure used for managing software versions and it is difficult to recognize the valid time for each version.

- The current approach does not consider the aspect of relative temporal in representing the valid time for each version.
- The current approach maintains 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.

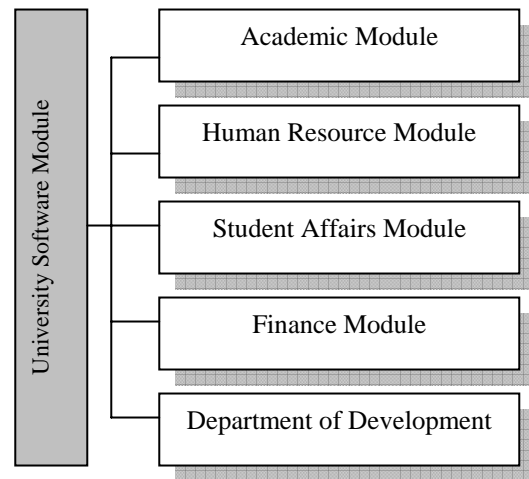


Figure 1: University Software Modules

Based on the mentioned problems, we strongly believe that the development of temporal-based software version management tool for ITC-UDM could gain the following benefits:

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

software version management process.

5. The Model

Version control is one of the main tasks in software configuration management. For any software version would have its own valid time. The collection of software versions should be organized into systematic way for the purpose of retrieval efficiency and to recognize valid time of those versions. Besides the used of unique sign for the associate version, the method of time-stamping is also needed to be embedded into the version management database.

5.1 The Temporal-Based Version Management Specifications

Temporal elements involved in the 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 represents a transaction when a new valid time for a version is recorded into the application 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, $OPERATORS = \{op_1, op_2, op_3, \dots, op_n\}$ and one or more defined event(s), signed as $EVENTs = \{event_1, event_2, event_3, \dots, event_n\}$. This model, considered only five temporal operators, hence will be denoted as $OPERATORS = \{equal, before, after, meets, met_by\}$. Table 1 illustrates the general definitions of temporal operators based on time interval and time points. Figure 2 shows the organization of temporal elements that involved in software

version management. If we have a software with a set of version signed as, $V = \{v_1, v_2, v_3, \dots, v_n\}$ then the model is:

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

where,

$$avt = [avt\text{-}from, avt\text{-}until],$$

$$rvt = [rvt\text{-}from, rvt\text{-}until],$$

$$rvt\text{-}from = \{ \{op_i \in OPERATORS\} \cap \{event_i \in EVENTS\} \} \text{ and,}$$

$$rvt\text{-}until = \{ \{op_i \in OPERATORS\} \cap \{event_i \in EVENTS\} \}.$$

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

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

where, A_i = attribute name of a version, $tt \in P$ and, $avt\text{-}from, avt\text{-}until, rvt\text{-}from$ and $rvt\text{-}until \in T$.

Table 2 exhibits the temporal-based version-record management for representing KEWNET's software version history. For example, KEWNET Ver. 1.1 has been updated three times. For the first time, the version has been recorded on *tt3* with absolute valid time is from *avf2* to *avu3* and relative valid time is from *rvf2* to *rvu3*. For the second updated, on *tt4*, absolute valid time is from *avf2* to *avu4* and relative valid time is from *rvf2* to *rvu4*. The version has another change request and therefore the version would have a new absolute valid time from *avf2* to *avu5* and relative valid time from *rvf2* to *rvu5*. This transaction is recorded on *tt5*.

Table 1: The definitions of temporal operator base on time point and time interval

Temporal Operator	Time Point	Time Interval
equal	$t = \{(t = t_i) \in T\}$	$\tau = \{(\tau = \tau_i) \in T\}$
before	$\tau = \{(\tau < t_i) \in T\}$	$\tau = \{(\tau < \tau_i) \in T\}$
after	$\tau = \{(\tau > t_i) \in T\}$	$\tau = \{(\tau > \tau_i) \in T\}$
meets	$\tau = \{(\tau \leq t_i) \in T\}$	$\tau = \{(\tau \leq \tau_i) \in T\}$
met_by	$\tau = \{(\tau \geq t_i) \in T\}$	$\tau = \{(\tau \geq \tau_i) \in T\}$

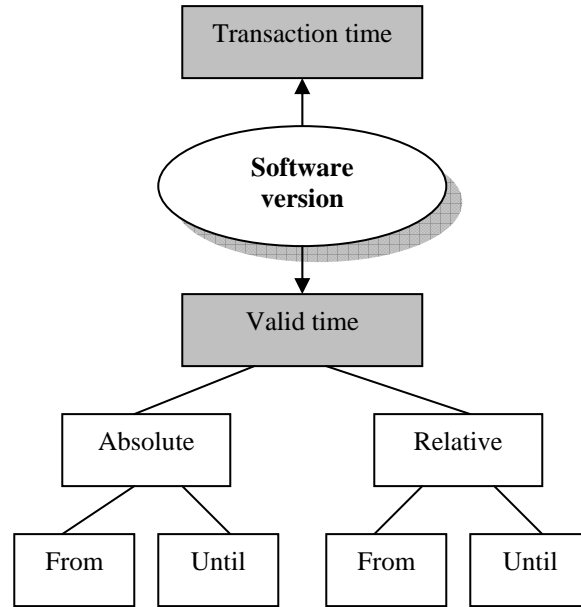


Figure 2: Temporal elements in software version management

Table 2. Version-Record for KEWNET software

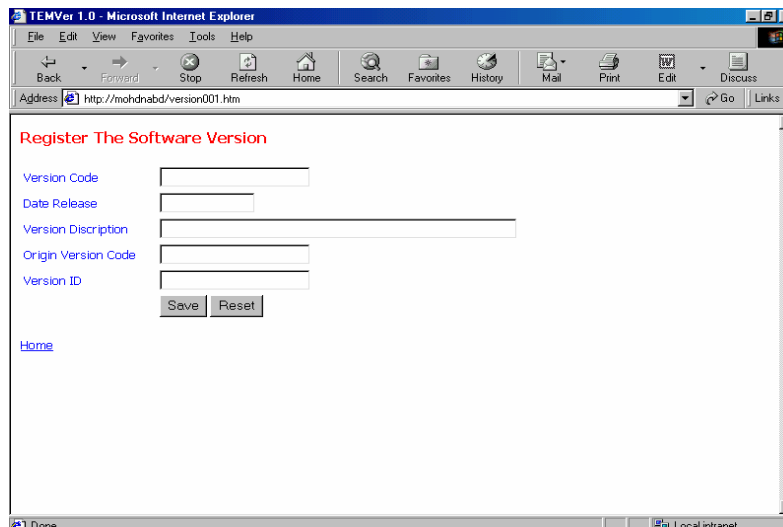
Ver #	tt	avt-from	avt-until	rvt-from	rvt-until
1.0	tt1	avf1	avu1	rvf1	rvu1
1.0	tt2	avf1	avu2	rvf1	rvu2
1.1	tt3	avf2	avu3	rvf2	rvu3
1.1	tt4	avf2	avu4	rvf2	rvu4
1.1	tt5	avf2	avu5	rvf2	rvu5
1.2	tt6	avf3	avu6	rvf3	rvu6
1.2	tt7	avf3	avu7	rvf3	rvu7
2.0	tt8	avf4	avu8	rvf4	rvu8
2.0	tt9	avf4	avu9	rvf4	rvu9
2.1	tt10	avf5	avu10	rvf5	rvu10

5.2 The Temporal-Based Version Management Functionality

To carry out experiments validating the model proposed, a client-server prototype has been developed. The prototype has three main modules: *register version*, *update the version valid time* and *queries*.

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.



The screenshot shows a web browser window titled "TEMVer 1.0 - Microsoft Internet Explorer". The address bar displays "http://mohdnabd/version001.htm". The main content area is titled "Register The Software Version" in red. It contains a form with the following fields: "Version Code", "Date Release", "Version Description", "Origin Version Code", and "Version ID". Each field has a corresponding text input box. Below the input boxes are "Save" and "Reset" buttons. At the bottom left of the form area is a "Home" link. The browser's status bar at the bottom shows "Done" and "1 normal intranet".

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. The tool also allows the user to make a query to the database. The users can browse the version valid time and status for any registered software as shown in Figure 5. Meanwhile, Figure 6 shows the output form of query for all histories of valid time and status for a software version.

Updating The Software Version Valid Time

Version Code:

Transaction Date:

Description:

Absolute Valid Time

Date Start:

Date End:

Relative Valid Time

Time Start:

Time End:

Update by:

Position:

[Home](#)

Figure 4: Update the software version valid time

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: The software version valid time report

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: The transaction records of a version

6. Conclusion

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 have been stamped on each software version so that the monitoring and conflict management processes can be easily made.

Based on the proposed model, a prototype has been developed. The prototype will be experimented in ITC-UDM. It will be used to monitor and keep track the evolution of the software version, systems module and software documents in university's software. 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.

References:

- [1] Bertino, E., Bettini, C., Ferrari, E. and Samarati, P. "A Temporal Access Control Mechanism for Database Systems", IEEE Trans. On Knowledge and Data Engineering, 8, 1996, 67 – 79.
- [2] C. E. Dyreson, W. S. Evans, H. Lin and R. T. Snodgrass, "Efficiently Supporting Temporal Granularities", IEEE Trans. On Knowledge and Data Engineering, Vol. 12 (4), 2000, 568 – 587.
- [3] G. M. Clemm. "Replacing Version Control With Job Control", ACM – Proc. 2nd Intl. Workshop On Software Configuration Management, 1989, 162 – 169.
- [4] D. Gao, C. S. Jensen, R. T. Snodgrass and M. D. Soo, "Join Operations in Temporal Databases", The Very Large Database Journal, Vol. 14, 2005, 2 – 29.
- [5] A. Dix, T. Rodden, and I. Sommerville. "Modelling Versions in Collaborative Work", IEE – Proc. Software Engineering, 1997, 195 – 206.
- [6] H. Gregerson, and C. S. Jensen, "Temporal Entity-Relationship Models – A Survey", IEEE Trans. On Knowledge and Data Engineering, 11, 1999, 464 – 497.
- [7] A. Gustavsson. "Maintaining the Evaluation of Software Objects in an Integrated Environment", ACM – Proc. 2nd Intl. Workshop On Software Configuration Management, 1989, 114 – 117.
- [8] A. Havewala. "The Version Control Process: How and Why it can save your project", Dr. Dobb's Journal. 24, 1999, 100 – 111.
- [9] C. S. Jensen and R. T. Snodgrass. "Temporal Data Management", IEEE Trans. on Knowledge and Data Engineering. 11, 1999, 36 – 44.
- [10] K. Torp, C. S. Jensen and R. T. Snodgrass, "Effective Timestamping in Database", The Very Large Database Journal, Vol. 8, 1999, 267 – 288.
- [11] B. Knight, and J. Ma. "A General Temporal Theory", The Computer Journal, 37, 1994, 114 – 123.

- [12] B. Knight and J. Ma. "A Temporal Database Model Supporting Relative and Absolute Time", *The Computer Journal*. 37, 1994, 588 – 597.
- [13] A. Lie. "Change Oriented Versioning in a Software Engineering Database", *ACM – Proc. 2nd Intl. Workshop on Software Configuration Management*. 1989, 56 – 65.
- [14] H. Mary. "Beyond Version Control", *Software Magazine*. 16, 1996, 45 – 47.