Advanced Engineering Forum *Online: 2011-12-22*
ISSN: 2234-991X, Vols. 2-3, pp 629-634
doi:10.4028/www.scientific.net/AEF.2-3.629
© 2012 The Author(s). Published by Trans Tech Publications Ltd, Switzerland.

A New Dynamic Version Control Method Oriented the Complex Product

Mingxu Ma[1, a], Xin Mao[1] and Qingchun Zhang[1]

[1] School of Mechanical Engineering and Automation, Northeastern University, Shenyang 110189, China

[a]mxma@mail.neu.edu.cn

Keywords: Complex product, Version control

Abstract. There are a lot of dynamic changing problems for design version in the complex product design process. We have study a dynamic version control technology for result it. Firstly, we have researched the three level version control model, including the complex product structure layer, the node version tree layer and the version structure layer. Secondly, we have researched the version control model operations, including the version structure operations and the version operations, and analyzed the implementation of the version generation function and the version mergence function. Finally, the method has been applied and verified in the development of an aircraft flight loads integrated design platform.

Introduction

Complex product engineering design, endowed with repeafigure, tentative, interactive and developmental characteristics, is an incremental refinement process. The continuous modification and optimization of design objects engender different design stages. Thus, one design object has different design results, in other words, different versions. Version control, a method of processing version objects systematically, not only manages different versions of one design object, but also administrates relationships between them. The version management in a product design process not only should establish a rational version model, record the emergence and transformation process history of design object versions and classify and preserve the design object, but also be prone to cast back and quickly search the variation of design objects and form a significative product structure.

The current common model for version control is VOM (Version Oriented Model)[1] and COM (Change Oriented Model)[2], both of which cannot effectively manage and coordinate quantities of engineering design data because of their own deficiency like easily producing combinatorial explosion. On the foundation of COM model and taking federal data management system as the carrier, Schonhoff etc. researched global version model from perspectives of design changes broadcast notification[3]. And on the foundation of VOM model, he discussed the federal data management problem[4]. Additionally, researches on version control are implemented from perspectives of product data management[5] and collaborative design[6].
Researches mentioned above solved some problems in version management to some degree. But to complex product version control, there are many deficiencies in those researches. Complex product version information possesses the following characteristics: Firstly, data structure is complicated. For it contains a great many structural data, such as control parameters, design period and staffs etc. as well as many unstructured data existing in file forms, such as geometric modeling and finite element analysis models etc. Secondly, design changes frequently. The amount of intermediate data is huge over the whole design process. Thirdly, changes of version structure are intricate. Directing at these deficient characteristics, this article presents a dynamic version control technology of complex products for the purpose of dealing with such issues.

Structure of Version Control Model

The section headings are in boldface capital and lowercase letters. Second level headings are typed as part of the succeeding paragraph (like the subsection heading of this paragraph). With so many parts, structure of complex products is complicated. Information of one version of the entire complex product includes version structures of the version's various parts. In other words, version information of a product should comprise structure information of the product and the version information of each structure. Product structure information has version feature too, and its version information is also known as configuration information. Particularly, those determinate complex products often have many kinds of configurations. Components corresponding to each configuration possess identical version information. Structure versions of complex products and version information of every structure joint constitute an intricate multidimensional spatial version combination. A specific CPBI is a snapshot or a view, according to which this article proposes a complex product version control model, as shown in Figure 1.

Hierarchy of the Complex Product Version Control Model. The left side of the model appears the configuration congregation of complex product structure tree. When design is finalized, a complex product often has many configurations in order to meet the demand of different type. These different configurations may be applied to parts of different version. Therefore, when a product version is designated, structure configurations of the product and components information of corresponding configuration version must be contained in the product version.

The middle of the model shows the version tree congregation of each node in the product structure tree. And the version tree here refers to tree collections of various versions of nodes. Version tree structure of node P3 shown in middle of Figure 1 is a good instance for above. In this article, the root node of the version tree represents basic information describing product structure nodes; nodes of the first layer of the version tree delegate conditions of great change, such as changes of version structure and changes of design and developing standard etc.

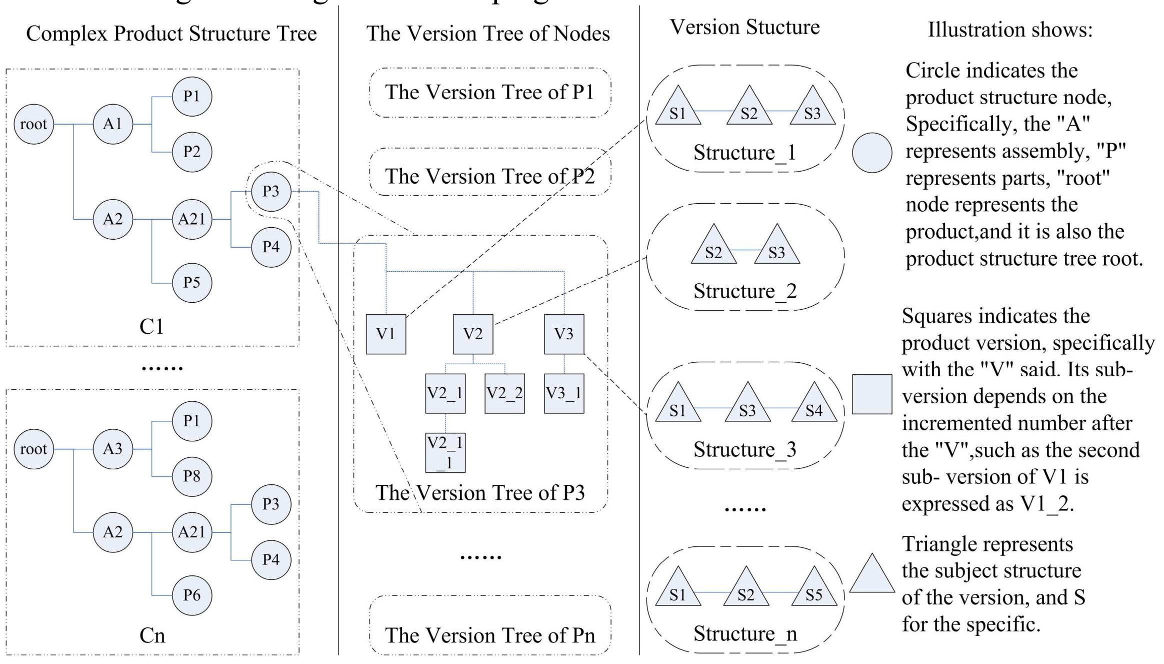
Complex Product Structure Tree
The Version Tree of Nodes
Version Stucture
Illustration shows:
root
A1
P1
P2
A2
A21
P3
P4
P5
C1
......
root
A3
P1
P8
A2
A21
P3
P4
P6
Cn
The Version Tree of P1
The Version Tree of P2
V1
V2
V3
V2_1
V2_2
V3_1
V2_1_1
The Version Tree of P3
......
The Version Tree of Pn
S1
S2
S3
Structure_1
S2
S3
Structure_2
S1
S3
S4
Structure_3
......
S1
S2
S5
Structure_n
Circle indicates the product structure node, Specifically, the "A" represents assembly, "P" represents parts, "root" node represents the product,and it is also the product structure tree root.
Squares indicates the product version, specifically with the "V" said. Its sub-version depends on the incremented number after the "V",such as the second sub- version of V1 is expressed as V1_2.
Triangle represents the subject structure of the version, and S for the specific.

Fig. 1. The complex product version control hierarchy model

The right side of the model reveals the so called version structure that refers to various 'information block' constituting version logos. That 'information block' means academic disciplines, such as structure subject, electromagnetic subject and flow-field subject etc, here makes it convenient to manage the design and developing of complex products that needs a great deal of design analysis and calculations. Software used to do design analysis and calculations are different. Even if the same type of analysis software, conditions of version change and input and output parameters format change etc. may occur. For that reason, this article takes disciplines as version management information block for the purpose of handling such problems. Version change could be change of design parameters as well as change of design software (including change of software type and change of software itself). As Figure 1 shows, great changes in product design process, like version structure changes caused by differences of design and analysis subjects or major adjustments of design parameters etc, are recorded in V_1, which is a node of the first layer in P_3 version tree. Let Structure$_{\_1}$ expresses version structure of version V_1 and Structure$_{\_2}$, version V_2 so and so. And all of such version structures may have different subject composition. Assemble all of the version structures applied in a product design process, and they will become a version structure database which can be gradually completed with the design and developing going on. Before the design of product, the first thing must do is to determine which kind of discipline software to be employed in product design process, then, to establish a homologous version structure tree according to which version management works.

Memory Type of Complex Product Version Control Model. Version item information and version relationship information are two main categories of complex product version information. Version item information is a kind of database that records version information of creator, date created, modifier, date modified, figure of father version, version status, relevant file path and name etc. These information, which can be restored in relationship database, act to manage node version information and provide convenience to carry out version operation of searching and modifying etc. Version relationship information is various typed files, including CAD files, CAE files and CAPP files and so on, which directly expresses design information of node version. System links node version and its design and analysis files through building fields of those typed files paths and names etc. Due to possibly involving inputs and outputs of many files in discipline analysis process, homologous file information are expressed in type of path file figure. As revealed in Figure 2, Aerodynamic field of a product structure node version V$_{\_1}$ includes Aerodynamic path figure 1 and Aerodynamic path figure 2, both of which separately point to specific path and file in file library.

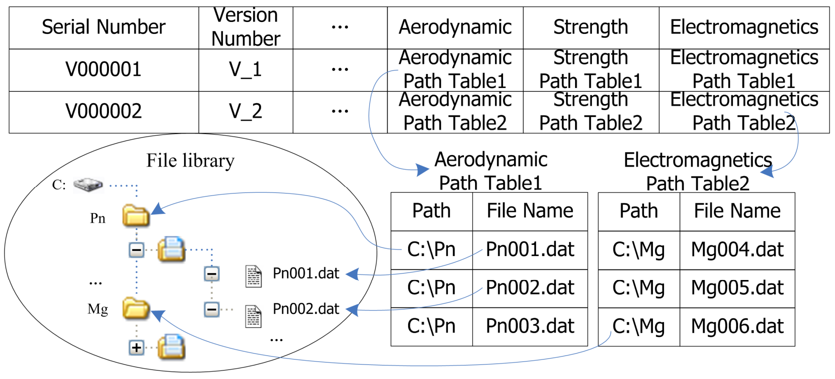

Fig. 2. The storage methods of the product node version information

Operation of Version Control Model

From the constitution of version model we can seen that a version of a complex product must be composed by a structure configuration of the version and all the specific version information of the structure nodes. Therefore, operations of the version control model can be divided into version structure operation, version operation and product node operation. The first two operations are the keystones of this article, and the last one can be found in literature [7].

Version Structure Operation. Version structure is the ‘skeleton’ of version information by expressing information constitution pattern of a version. Version structure of complex products indicates the discipline analysis condition of complex product structure nodes. For instance, if a version of a node has two discipline analysis data of fluid dynamics and intensity, version structure of the node must have information blocks of these two disciplines. Complex product design and analysis is a iterative process. In addition, the input and output parameters of each analysis, the applied analysis software and their version or the analyzed disciplines are diverse from each other. As a result, version structure is required to have adequate flexibility to flexibly add, delete and modify discipline blocks. Four version structure operations: creating version structures, adding disciplines, deleting disciplines and modifying version disciplines are presented, through which version structures can be customized flexibly. And this lays a foundation for dynamic version control.

For ease of exposition, this article defines D as a discipline collection and d_j the jth subject element of D, namely $d_j \in D$. S is defined as version structure collection and s_i the ith version structure element of S; that is $s_i \in S$. i, j, m and n are natural numbers and m and n differs.

Creating Version Structures. It is used to create a new version structure which could contain subjects from number 0 to more numbers. The created version structure includes two parts. One is version structure element information recording the version structure information of date created, creator, and sum of subjects etc; the other one is version structure input and output information recording the version structure information of definite subject name, brief introduction of subjects and subject analysis etc. The formalized symbol of creating version structure can be $\odot$, and $\odot s_i$ refers to creating version structure s_j.

Adding Version Disciplines. It is applied to add a specified subject and other attached information to an existing version subject. It is formalized as $\oplus$, and $s_i \oplus d_m$ means adding subject dm to version structure s_j. However, only one subject can be added at a time in order to make the logic of version structure operation clear.

Deleting Version Disciplines. It implies to delete a specified subject and other attached information from an existing version subject. It is formalized as$\ominus$, and $s_i \ominus d_m$ means deleting subject dm from version structure s_j.

Modifying Version Disciplines. It implicates to modify attached information of a specified subject in the existing version disciplines. It is formalized as ®, and s_i®d_m means modifying subject dm in version structure s_j.

Version Operation. On the condition of version structure created, advanced version operations, such as version creating, combining, searching, comparing, backup and recovering etc. can be accomplished. These functions lean on corresponding product structure nodes. Version operations are achieved automatically and interactively according design process of system. Version operations of complex product version control, such as searching, comparing, backup and recovering, are similar to traditional product version operations, and no more words about them are needed. This article will focus on the realization of the other two operations: version creating and combing.

Version Status and Its Operations. A version has two states of releasing and working. When a version is in releasing state (means official version), it is evinced that no child nodes are being created in the version, and checkout operation can be implemented to change the version's state into working state. When the version is under working state (refers to working version), it is shown that child nodes are being engendered, and checkout operation to the version is forbidden.

To avoid ambiguity, definitions of checkout version operation and checking-in version operation are given. ①Version checkout means that an official version in public service district becomes a working version and is locked, if being copied to local personal workspace. The locked or checkout version cannot be checked out again in order to avoid conflict. But operations, like searching and copying, that do not influent version number can be carried out. ②Version checking-in refers to submitting a working version in personal workspace to public service district to form an official version and unlocking the official version.

Version Generating. As mentioned earlier, any node in a complex product structure has a Version item attribute which is the root node of version tree. Version item of each node at least has one node version that simultaneously possesses father nodes, child nodes and brother nodes. As shown in Figure 3, V1 has father node: Version item, child nodes: V_{1_1} and V_{1_2} and brother nodes: V_2 and V_3. A version item can possess many first level versions, and each first level node is also allowed to have many second level versions, so and so, every version tree is comprised of multi level version nodes. Version structures of the first level can be different or not. Nevertheless, version structures of every first level node and all their child versions must be the same; if not, the version structures must belong to different first level versions. The former and later version nodes indicate brother relationships, e.g. the relationships of V_1, V_2 and V_3 in Figure 3. The above and below version nodes share a parent-child relationship, e.g. the relationship of V_2 and V_{2_1} in Figure.3.

This article takes $V_{2\_1\_2}$ in Figure 3 as an instance to explicate generating process of a new version that includes: Checking out version V_{2_1} from public service district to form a wording version V_{2_1}'. Examining the current maximum child version number i of V_{2_1}', here i=2_1_1. Generating a new child version $V_{2\_1\_2}'$ numbered i+1. At last, submitting child version $V_{2\_1\_2}'$ to public service district and forming an official version $V_{2\_1\_2}$.

In the evolutionary process of a node version, all node versions are accomplished by above-mentioned method except the Version item. The method can merely generate child version nodes and does not directly dispose the generation of brother versions. Brother versions can be engendered indirectly by searching its father versions and generating child versions under the father versions.

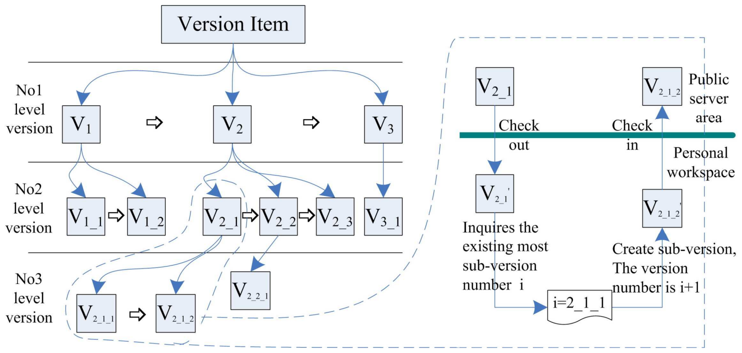

Fig. 3. The node version tree generation and its evolution process

Version Merging. Version merging in this article refers to acquiring a new version, which possesses version characteristics of the later mentioned original versions, based on two or more versions of product nodes,. Generally speaking, the new version has two characteristics both similar version structure and selectively inheriting diverse product information possessed by the original versions.

Version merging is complicated and occurs commonly in complex product design process. Some kinds of Version merging type are exposed in left side of Figure 4.

Same Level Combining Under the Same Child Version Tree. V_{2_1} and V_{2_2} in Figure 4 are combined into version V_{2_3}, which is very similar to the generation of brother versions mentioned earlier. Unit information of V_{2_3} records information that it is combined from V_{2_1} and V_{2_2}, and what unit information of V_{2_3} combined in brother version type records is brother version information. This shows the two are different.

Nodes of Different Level Combining Under the Same Child Version Trees. For instance, V_{1_1} and $V_{1\_2\_1}$ in Figure 4 are combined into version V_{1_3}.

Version Merging Between Different Child Version Trees. E.g. $V_{1\_2\_2}$ and V_{2_3} in Figure 4 are combined into version V_3. One thing to be mentioned is that, in this case, the combined version will become the first level version just below Version item, whatever the combining versions are from the same level or not. In actual complex product development, Version merging usually is a complicated combination of the three combining type. However, no matter what situation appears, Version merging operation can be realized in the light of combining method mentioned below.

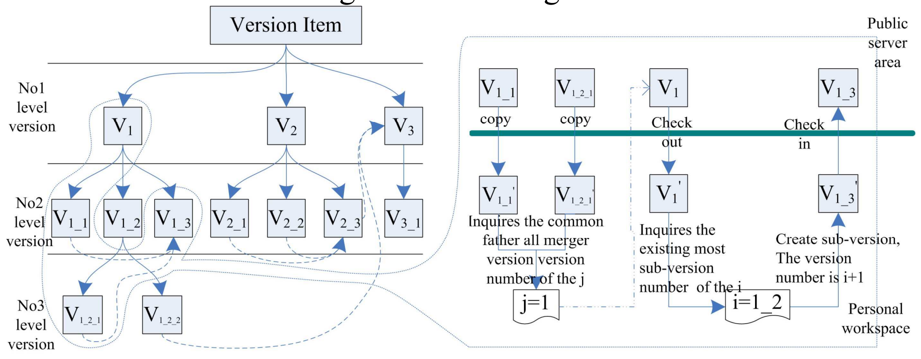

Fig. 4. The node versions merging types and their generative process

The principle of version combination is to search a mutual father version of the combining versions and create a new child version under the father version. The first level versions can have different version structures. Therefore, Version merging has great flexibilities and can avoid version conflict. Taking version V_{1_3} combined from V_{1_1} and $V_{1\_2\_1}$ for example, a Version merging process is explained, as shown in right side of Figure 4 surrounded by broken line. Six steps are need to complete the combination:

a. Copying version V1_1 and V1_2_1 to be combined from public service district to personal workspace.
b. Obtaining the minimum number j of mutual father version of V1_1 and V1_2_1, here j=1.
c. Checking out V1 to workspace.
d. Inquiring the maximum child version number i, here i=1_2.
e. Creating child version V1_3' numbered i+1.
f. Submitting version V1_3' into public service district.

Version structures may change when combination between different child version trees occurs. This can be handled by version structure operation mentioned earlier.

Summary

We have studied a dynamic version control method for the lot of the design version dynamics problems in the complex product design process.

Firstly, we have achieved the hierarchy and the storage mode of the version control model including complex product structure layer, the node version tree layer, and release structure layer.

Secondly, we have completed the various operations including the version structure operation and the version operation, and obtained the realization methods including the version generation and the versions merge.

Thirdly, this new method is applied in the design for an aircraft flight loads integrated development platform.

Acknowledge

This research was supported by Liaoning province Dr start-up fund (20071022).

References

[1] M.Wein, W.M.Cowan, W.M.Gentleman. Visual support for version management. Proceeding of the 1992 ACM/SIGAPP symposium on applied computing (SAC), Kansas, USA, (1992)1217-1233.

[2] I Oshri, S Newell, Component sharing in complex products and systems: challenges, solutions, and practical implications. IEEE Transactions on Engineering Management, 52 (2005) 509-521.

[3] M.Schonhoff, M.Strassler. Global version management for a federated turbine design environment, LNCS, 1649 (1999) 203-220.

[4] M.Schonhoff, M.Strassler, K.R.Dittrich, Version propagation in federated database system. 2001 International Symposium on Database Engineering & Applications, Grenobble, France, (2001)189 -198.

[5] F.Meziane, Y.Rezgui, A Document Management Methodology Based On Similarity Contents. Information Sciences, 158 (2004) 15-36.

[6] B.Lee, K.Chang, H.Y.Nara, An integrated approach to distributed version management and role-based access control in computer supported collaborative writing[J]. The Journal of Systems and Software. 59 (2001)119-134.

[7] M.X.Ma, Y.S.Fan, C.W.Yin, Research on product lifecycle based on product structure, Chinese Journal of Mechanical Engineering. 42 (2006) 83-90.