Version Management Based on the Association of Generalized Domains

Li Qingfeng¹, Guan Bo¹, Bailin Yang*²
¹College of Electron & Information Engineering, Ningbo University of Technology, Ningbo, China
²Department of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, 310018, China
*corresponding author, e-mail:ybl@mail.zjgsu.edu.cn

Abstract
Multiple versions will be produced in the process of complex products developing. There are mass data of product components in different types and formats and relationships between components. Traditional researches of version management model concentrate on the information of single object rather than that of multi-objects and relationships between them, so it is difficult to manage the versions of complex product. A version management model based on the association of generalized domains was presented according to the polychromatic graph theory in the paper, and dynamic variations, features and rules of the association in version evolution were studied, and a version management system for controller software was developed. Examples show that the association model of generalized version based on the poly-chromatic graph realizes the effective management of the complex product developing.

Keywords: Version management, polychromatic graph, generalized domain, dynamic association

1. Introduction
Complex multispecies product development is a tentative process of gradual refinement, with recurrent, tentative, interactive and developmental characteristics. Product development process is often carried out in several stages. Because of various methods of performance description and differences in research programs, different versions will form object to the same product in various stages of development. Most of these new versions are the results of improving the original versions in part. Different results produced at different stages, different environmental conditions of the same object can be seen as different versions produced in various stages of an objective object, which required a higher requirement of version management technology during the researching process.

Version management records and maintains the evolution history of the object (including documents, drawings, software programs, etc.), including the definition, generation, deletion, freeze, thaw, etc. As the complexity of the object, version management should not only sustain the changes, selection and configuration of the design scheme, but also support multiple copy of the objects, configuration of multiple sets of components, definition, expression, establishment, management and their relations of a variety of options during the various stages of product development, to form the integrated product development environment which contains versions, version links and version environment. Version management should not only describe the history of the object’s generation, export, and modification, but also to reflect the composition of the structure between objects to some extent.

Traditional version management model includes the tree model, graph model, etc.[1-4]. Version of the tree model includes temporary version and working version, the former can generate a new temporary version via operations such as modification and deletion, at the same time, can also be transformed into a working version. The later can only be queried and deleted, not updated. The tree version management model features more perfect, but there are issues such as confusing semantics and complex implementation, and its model version and data version use different units, thus make the system difficult to use in a unified, single storage of version also increases the data redundancy to the system, so it is difficult to ensure the consistency of the data version; acyclic digraph version model only supports data version, the data version uses an object as unit, all the object can generate version. The new version can be
generated implicitly or explicitly. The operations on the new version have no restriction, but the
old version can only implemented the query and delete operations, directed acyclic graph
solved the problem effectively of version combining, but it's support for complex object version
is weak, and it does not support schema versions, same to the tree version model, there are
also defect of large amount of data redundancy.

To make up these shortcomings, the modeling methods of version management based
on polychromatic graph is proposed[5-10]. we use acyclic digraph to create a version model,
chromatic edges in the graph using the principle of polychromatic graph, describe the
relationship between the version with the color of the edge, each side can be colored a variety
of colors, making the model more open, node in the model show the information of the version,
including version level information, identification information, status information, version’s
conditions of validity information, version index information , storage information, etc. The
overall color of the node represent status information, version identification information can be
represented by ordered pairs, for their information owned by the version themselves, we can
color each node with several colors to represent different attributes.

Based on the work of the study proposed as above, this paper put forward a
generalized version inter-domain correlation model based on the polychromatic graph and
management method of version object status information, the evolution of the complex version,
inter-domain version relationship and the storage of version object, by analyzing the
characteristics of association between the generalized version of the domain, we summarized
inter-domain association’s changing rules in the process of version’s evolution; applying the
generalized version of the polychromatic graph-based inter-domain correlation model to the
controller version of the management system, we established the association between products
and the software version, achieve the effective management of the controller software version.

2.Research Method
2.1. The Association Model of Generalized Domains
The composition of a conventional set A is as follows:

\[ A = \{a_1, \ldots, a_i, \ldots, a_n\} \]  \hspace{1cm} (1)

For elements \( a_i \), the difference resides in their names even though these elements
could be different. In polychromatic sets, the whole set and all its internal elements, can be
pigmented with different colors to represent the research object as well as the properties of its
elements. e.g.: the color set \( F(a_i) \) corresponds to every element \( a_i \in A \) and the color set
 corresponds to the entirety of A,

\[ F(A) = \{F_1, \ldots, F_j, \ldots, F_n\} \]  \hspace{1cm} (2)

\( F(a_i) \subseteq F(A) \) , \( F_j(A) \) are called unified color. The color constituents of its elements
\( F_j(a_i) \) are called pigmentation.

When an object of simulation is represented in terms of polychromatic sets, its color
\( F_j(A) \) and \( F_j(a_i) \), the color of the elements ,correspond to the jth character of the object or the
element. All the elements of polychromatic sets can be represented using the following Boolean
matrix:

\[
\begin{bmatrix}
F_1 & \ldots & F_2 & \ldots & F_n \\
\begin{bmatrix} a_{i(1)} & \ldots & a_{i(j)} & \ldots & a_{i(n)} \end{bmatrix} & a_i \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\begin{bmatrix} a_{j(1)} & \ldots & a_{j(j)} & \ldots & a_{j(n)} \end{bmatrix} & a_j \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\begin{bmatrix} a_{n(1)} & \ldots & a_{n(j)} & \ldots & a_{n(n)} \end{bmatrix} & a_n
\end{bmatrix}
\]
\hspace{1cm} (3)
Where \( c_{(i,j)} = 1 \) if \( F_j \in F(a) \), and \( F(a) = \bigcup_{i=1}^{n} F(a_i) \), color \( F_j \) is represented as a logic variable:

\[
F = \begin{cases} 
1 & \text{if } F_j \in F(a) \\
0 & \text{if } F_j \notin F(a) 
\end{cases}
\]

The existence of individual colors of the element that constitutes a polychromatic set is the key factor that determines the availability of unified color. As an example, the existence of unified color \( F_j(A) \) is determined by the existence of individual colors with the same names of element \( a_i \in A \) with same names, but also by any other colors. Therefore, the composition of unified color and the following Boolean matrix:

\[
\| y_{(i)} \|_{\Omega_{(a), F(a)}} = [F(a) \times F(A)]
\]

Mathematical representation of polychromatic sets:

\[
PS = \{A, F(a), F(A), [A \times F(a)], [A \times F(A)], [A \times A(F)]\}.
\]

### 2.2. Polychromatic Graph

The basic idea of polychromatic graph is generated during the pigmentation. In general, a node is pigmented in a chromatic graph with one kind of color. On this basis, polychromatic graph extends as it pigments not only a node but also an edge with multicolor. The polychromatic graph corresponds to sets:

\[
G = (A, C)
\]

Where:

\[
A = (a_1, \cdots, a_n); [A \times A] = C = [c_{1,2}, c_{1,3}, \cdots, c_{(n)(n)}, \cdots, c_{(m)(m)}];
\]

Node set:

\[
PS_A = (A, A \times A, F(a), F(A), [A \times F(a)], [A \times F(A)]);
\]

Edge set:

\[
PS_E = (G, F(C), [C \times F(C)], [C \times C(F)]);
\]

Polychromatic graph set:

\[
PS_G = (F(G), PS_A, PS_E).
\]

### A. Basic concepts

**Definition 1:** The generalized version

It refers to various data collection in a state (version) during the product development process. Generalized version has three significant differences from general version in concept: (1) generalized version has a broader extension, it concludes gradual version in the process of research and development, and all the medium versions; (2) the generalized version has a richer content, it concludes a research and development status of components, and various analysis and test data corresponding to the research and development status; (3) the generalized version is more complex to manage. The management of generalized version should organize, manage and operate versions’ data effectively, and organize the relationship between versions correctly, and reflect the development process of the version clearly.

**Definition 2:** Domain of the generalized version

Domain (abbreviated as D) is divided according to the object described by the generalized version, the elements in the domain are generalized Version (abbreviated as V), the generalized version in the same domain is a collection of all versions of the same object. There are \( D = \{V_1, V_2, \ldots, V_n\} \).

**Definition 3:** Correlation of the inter-domain of the generalized version
Generalized version of the inter-domain are disjoint, that is, \( D_1 \cap D_2 = \Phi \), but the version elements of the domain can have a link with elements of other domains, called the inter-domain correlation. This inter-domain association can be access to the relationship, such as \( V_i \) in the version of the \( D_1 \) domain has access to \( V_j \) in the version of \( D_2 \) Domain; timing relationships can be, for example, in version \( D_1 \), operations to \( V_i \) lag behind operations to \( V_j \) in the version of \( D_2 \) domain, etc.

**B. Model Construction**

According to the definition of generalized version of the domain, we can abstract a group of related versions of the domain into a set of parallel planes in three-dimensional space. Figure 1 is generalized version of the domain model based on the theory of multi-color atlas, in which two generalized versions of the domain \( U \) and \( V \), for example, each domain represents a collection of all versions of an object, that is there are two parallel planes, the plane include extendable multi-color atlas directed acyclic graph. The core of the association between \( U \) and \( V \) the association between versions (object versions \( U_2 \) and \( V_1 \) as shown in Figure 1) in the two domains, so we link the nodes of two versions with a line in space, color the line depending on the association of segments, such as access to relationships, timing relationships. For complex situations, different versions within one domain and multiple nodes of different domains have association, for example: \( U_3 \) has two associated edges with \( V_0 \) and \( V_3 \), edges can be colored according to different needs.

Figure 1. Association model of generalized version

**C. Model features**

1. **Encapsulation**

   Association model of generalized version of the inter-domain could package all versions of the same product in a domain, ensure the independence of the product version of the object during evolution, avoid interference between multiple versions and objects, and reduce the complexity of version management effectively.

2. **Relevance**

   All the generalized version of the domain are not isolated, there are multiple types of contact between them, they constitute the generalized version of the domain space together, constitute a complex and versatile form of architecture. Generalized version of the domain associated model can describe multiple relationships between versions of the domains effectively, so that objects described in each version domain achieve an organic union, exchange information, share resources, eliminate the phenomenon of information silos, on the premise of ensuring inter-domain version’s access security, it greatly improve the reusability of inter-domain version.
(3) Diversity
Within diversity: the use of multi-color atlas principle within the version of domain manage all the evolutionary version of object, effectively distinguish the replacement, amendment and other relations between versions, record the development process of the version within domain in detail. Inter-domain diversity: using edge coloring method, the relationship between versions of the inter-domain describes a variety of logic relations of version in domain, such as dependencies, timing relationships, etc. and an object of the version can issue a number of inter-domain correlation, and according to the difference of the relationship, the domain association be colored with different colors.

2.3. Association Variations in version Evolution
A. Inter-domain correlation characteristics
Inter-domain correlation is generalized version of the domain D1, D2, respectively expressed by two colored directed acyclic graph, and $D1 \cap D2 = NULL$, which version of the $V1 \in D1$, $V2 \in D2$, if there is one kind of association relation $R$ between $V1$ and $V2$, with a side link $V1$ and $V2$, then this edge is inter-domain correlation. This inter-domain association can be formed association, access association, rules of association and so on. According to these differences of association, the edge was colored with different colors (can be defined as needed).

Inter-domain correlation includes eight features:
(1) Duality
Inter-domain association occurs between two versions in different domain, cannot occur in the same domain, that is, two versions of the same object have no inter-domain relationship.
(2) Direction
Correlation of the domain could have direction, the two object versions which participate in the association are described by $V1$, $V2$, then the $V1 \rightarrow V2$ explain that the association only exist on one direction, called the one-way association (Unidirectional Association), $V1$ is the issue of the association, $V2$ is the receiver; $V1 \leftrightarrow V2$ explain that the association exist in both directions, known as the bidirectional associations (Bidirectional Association).
(3) Implication
The association between the two associated object versions has a certain connotation, and this connotation is defined by a set of rule conditions, only the two objects all meet the conditions set, can they have the association.
(4) Dynamic
One object version $V1$ results in a subsequent version $V2$ through evolution, set up an association $V1 \leftarrow V3$ in the association relationship, if the $V3$ can transfer the association to $V2$, that is hid association $V1 \leftarrow V3$ and the establish association $V2 \leftarrow V3$, then this association is a dynamic association, otherwise known as static correlation.
(5) Backtrace
Version $V1$ produces revised version $V2$, dynamic association $V1 \leftarrow V3$ redirected into $V2 \leftarrow V3$, when $V2$ is removed, the association can be placed back into $V1 \leftarrow V3$.
(6) Transitivity
For the association $V1\rightarrow V3$, version $V1$ produces $V2$ after revising, if $V2$ can obtain the association issued by the precursor version $V1$, generate $V2\rightarrow V3$, then the association $V1\rightarrow V3$ is transitive, or the association can not pass.
(7) Effectiveness
Effectiveness is that association is effective to both sides of the association. In the evolution of version, affected by the dynamic, transitivity and retrospective of the association, when retrospective change occur to some version's association, if you cannot find versions that meet the changing rules set, then the association will become invalid.
(8) Color retention
In the version evolution process, both sides of the association vary complicated, but the meaning of association is unchanged, which means the color of the association remains consistent.
Dynamic and transitivity of the association determine the ability that associated object change from the current version to a subsequent version, retrospective determines the ability that the association changes from subsequent version to the current version.
B. Changing rules for association

Inter-domain association was expressed by a 6-tuple:

\[
\text{Relation} = \langle \text{Source, Destination, Condition, Dynamic, Pass, Validation, Color} \rangle
\]

Source and Destination reflect the two associated object versions, Source is the sender of association, and Destination is the receiver of association.

Condition is a set to meet the associated conditions, for example, the properties of Source and Destination in partial order should satisfy the partial order; the property data of Source and Destination in subset relationship should be within a range.
- Dynamic indicates the dynamic property.
- Pass indicates the pass property.
- Validation indicates the validation property.

Color indicates the color, because of the color retention of association; the color property doesn’t change in the process of evolution, so we don’t do too much discussion.

During the evolution of version, the association changes according to changing rules, change rules include related conditions rules, effective rules, revised rules, dynamic rules, transfer changing rules, changing back rules.

\[
<\text{ChangeRules}> ::= <\text{Condition}> <\text{Validation}> <\text{Update}> <\text{DynamicRule}> <\text{PassRule}> <\text{RollbackRule}>
\]
\[
<\text{Condition}> ::= <R's \text{ Condition}>
\]
\[
<\text{Validation}> ::= <R's \text{ Validation}>
\]
\[
<\text{Update}> ::= <\text{version V is the revision of version U, the association received by the version of the U will change, otherwise changes will not release}>
\]
\[
<\text{DynamicRule}> ::= <R's \text{ Dynamic}> <\text{Dynamic_Rule}>
\]
\[
<\text{PassRule}> ::= <R's \text{ Pass}> <\text{Pass_Rule}>
\]

C. Dynamic Rules

There are two objects: D and S. D has a version D1 and S has a version S1. There is a relationship between S1 and D1: S1->D1.

Rule 1 : If D1 develops a revision D2 (consistent with redaction rule), S1 hides the relationship between S1 and D1, and develops the relationship between S1 and D2: S1->D2.

Rule 2 : When S1 develops the relationship between S1 and D2, D1 develops another revision D3. Then S1 develops the hidden relationship between S1 and S3. That is, the relationship which comes form the hidden relationship’s dynamic change is hidden too. This rule guarantees the number of the visible relationships which come form the initial relationship. And S1 will maintain stack architecture to save the hidden relationships (Select heap or stack according to sequential relationship). Then we can backdate the hidden relationships from newest to oldest when needed.

Rule 3 : Users can change the status of the relationship (visible or hide).

Rule 4 : If D1’s successive versions (D2 and D3) merge and develop a new version D4, S1 will hide the visible relationship (S1->D2, according to sequential relationship) and develop the relationship S1->D4.

Rule 5 : After Rule 4, S1 will search D4’s direct pre-versions from visible relationships according to sequential relationship, and operate by rule 1; then search D4’s direct pre-versions from hidden relationships according to sequential relationship, and operate by rule 2. Rule 5 will produce redundancy relationship, so it should not be executed by default.

Rule 6 : Users can choose whether to execute Rule 5.

Rule 7 : Users can change the sequential relationship (“from newest to oldest” or “from oldest newest to newest”).

D. Transfer Rules

There are two objects: D and S. D has a version D1 and S has a version S1. There is a relationship between S1 and D1: S1->D1.

Rule 1 : If S1 develops a revision S2, S2 develops the relationship between S1 and D2: S2->D1.

Rule 2 : S4 is the mergence of versions (S2 and S3). S2 and S3 have an identical relationship which comes from the relationship (S1->D1)’s transfer. Then we can choose one of
them to develop the relationship S4→D1 according to sequential relationship. (This rule will not produce redundancy relationship.)

Rule 3: Users can choose to produce redundancy relationship according to sequential relationship and rule 1.

Rule 4: Users can change the sequential relationship.

E. Backtrace Rules

There are two objects: D and S. D has a version D1 and S has a version S1. D2 and D3 are D1’s successive versions. D4 is the merge of versions (D2 and D3).

Rule 1: Relationship’s backdate can come back to the origination and can’t come back to the origination’s precursor. For example, the relationship which comes from “S1→D2” can come back to “S1→D2”.

Rule 2: If the visible relationship “S1→D4” comes from “S1→D2”, when we delete D4, S1 will delete the relationship “S1→D4”. If “S1→D2” is hidden, S1 will change it to visible relationship.

Rule 3: If the hidden relationship “S1→D4” comes from “S1→D2”, when we delete D4, S1 will delete the relationship “S1→D4”.

Rule 4: If a version is illegally or not available during the backdating, skip the version and backdate the version to its precursor on the premise of being consistent with rule 1.

2.4. Version Management of Controller Software

Managing controller software should not only to effectively organize, manage and operate versions’ data, but also to correctly organize the relationship between versions, clearly reflect the development process of the version. Association model of generalized version based on the polychromatic graph was established to manage controller software version. As a generalized version, the controller software divided the version of domain according to the software name. The evolution of the software version domain was indicated with a directed acyclic graph in domain. We colored the node to indicate the state of the software version, the edge to indicate the relationship between software versions.

As shown in Figure 2, the original version 7HT165GY of the controller software 7HTA13.8AT is modified to revision 7HT165GY_1_20080430_update. Figure 3 shows the evolution process of versions and relationship between versions in version domain of controller software 7HTA13.8AT.

Version 7HT165GY_1_20080430_update is the effective working version, using the P2 and P5 as marks; and the version has revised of the relationship with the original version, so the connection between them is colored yellow.

Modifying the original version 7HT165GY produce an alternative version 7HT165GY_3_20080430_instead, using P2 and P5 as marks, linked with the red directed edge (on behalf of substitutes).

7HT165GY_6_20080430_merge was merged with two valid working versions: 7HT165GY_4_20080430_update and 7HT165GY_5_20080430_update, and this version is the, using P1, P2, P5 as marks.

The host uses a variety of controller software versions and one software version is used by more than one host, that is to say there is a many-to-many correspondence between the host and the software version. The host was looked as a version object, according to the type of host, we divide the domain; controller software was looked on as the version object, according to the software name, we divide the domain, then the correspondence between the host and the controller software has the corresponding relationship with the version inter-domain association, when the software version evolves, this correspondence changes according to the rules of related changing between generalized version of the inter-domain, with the process of version evolution.

In Figure 3, 7HT165GY_1_20080430_update is a revised version of software version 7HT165GY, the system will find the association collection of the precursor version, and update the elements in the collection. Specific process is as follows:

Step 1: get the associated collection RelationReceiveSet and RelationSendSet of the precursor software version 7HT13.8AT_7HT165GY, in which RelationReceiveSet is an associated relationship collection received by version object, RelationSendSet is an associated relationship collection send by version object. Elements in the collection
are described in the form of 6-tuple Relation <Source, Destination, Condition, Dynamic, Pass, Validation, Color>.

Figure 2 Editing of controller software 7HTA13.8AT_7HT165GY

Figure 3 Version management of controller software 7HTA13.8AT version domain

Step 2: For RelationReceiveSet, we look for a valid dynamic association (i.e. property of Dynamic and Validation are all true), record the sender identification of the association (Source), obtain sender identification collection SourceSet (related to the precursor version of the host).

Step 3: The system has established an index for host and controller software version, according to the indexed version of the object retrieval algorithm, it searched the host object collection ObjSourceSet represented by SourceSet.

Step 4: The updated association of host satisfying the association changing conditions in ObjSourceSet points to 7HT165GY_1_20080430_update.

Step 5: For RelationSendSet, find the valid transitive association within it (i.e. property of Pass and Validation are all true), and record the recipient identification of the association.
(Destination), obtain the identification collection DestSet of the receiver (precursor version associated host).

Step 6: Find the host objects collection ObjDestSet represented by DestSet according to searching algorithm.

Step 7: For host in ObjDestSet satisfying association changing conditions, establish the association from 7HT165GY_1_20080430_update to the host.

As shown in Figure 4, there is more than one host have association with version 7HT165GY of controller software 7HTA13.8AT. If the revised version 7HT165GY_1_20080430_update generate, the host associated with the precursor version automatically associate to the revised version 7HT165GY_1_20080430_update, as shown in Figure 5.

![Figure 4 Version of 7HT165GY-7HTA13.8AT](image1)

![Figure 5 Version of 7HT165GY_1_20080430_update](image2)
3. Conclusion

The paper studied the version management in product developing, and a version management model based on the association of generalized domains was presented, the main researches are as follows:

(1) An association model of generalized domains based on the polychromatic graph was proposed to manage the complex versions of the object, and describe the association between different versions of the same object and versions of different objects, and achieve a correlation method of the multi-object version.

(2) The rules collection of associated version's dynamic changes was summed up during the evolution of version, and the dynamic response from the association of the inter-domain to evolution of version in domain was realized.

(3) A version management system for controller software was developed and realized in application.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (Grant No. 61170214, 61170098), Zhejiang Province Natural Science Fund for Distinguished Young Scientists (Grant No. LR12F02001), and Zhejiang Gongshang University for Distinguished Young Scientists (QZ11-9).

References