A NOVEL DIFF AND MERGE APPROACH ON THE BASIS OF OPERATIVE MODELS

Christian Koch[1] and Berthold Firmenich[2]

ABSTRACT

According to the State-of-the-Art building model instances are described by objects and attributes. Due to the iterative nature of the planning process many versions of a building instance are created and distributed between the actors involved.

In object oriented environments like planning of buildings existing tools to manage versions have considerable shortcomings, which are attributed to the fact that private object attributes have to be compared and merged by tools which are not aware of the semantics of the differences between the versions.

The solution approach presented is based on a completely different modeling approach called operative modeling. Unlike the traditional approach, an operative model instance is described by the applied operations that lead to the respective building instance state. The applied operations describe the differences between two versions of a model instance. Appropriate diff and merge tools can be developed on the basis of operative models because the semantics of differences are explicitly stored.

KEY WORDS

operative modeling, distributed work, versioning, civil engineering.

INTRODUCTION

In the computer-supported planning process building information is described by models that are instantiated. Due to the iterative and distributed nature of the planning process several versions m_i of the building instance are created and exchanged between the actors involved.

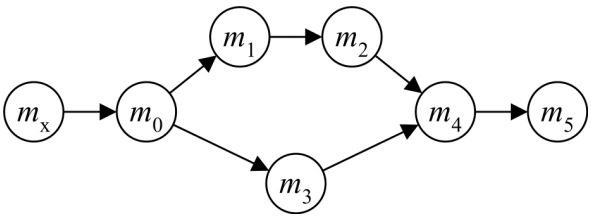

Figure 1: Versions of a building instance with m_x as the empty version

[1] Research assistant, CAD in der Bauinformatik, Coudraystr. 7, Bauhaus-University Weimar, 99423 Weimar, Germany, Phone +49 3643/58-4231, FAX +49 3643/58-4216, christian.koch@bauing.uni-weimar.de

[2] Professor, CAD in der Bauinformatik, Coudraystr. 7, Bauhaus-University Weimar, 99423 Weimar, Germany, Phone +49 3643/58-4230, FAX +49 3643/58-4216, berthold.firmenich@bauing.uni-weimar.de

In order to be able to manage versions it is indispensable to compare and merge revisions (Fig.1: m_1, m_2) and variants (Fig.1: m_2, m_3) of the building instance during the cooperative planning process.

Available tools for comparing (diff) and merging (merge) versions have considerable shortcomings, which are to some extent attributed to how building information is usually modeled today.

The objective of this contribution is to introduce the operative modeling approach and to show how new tools for comparing and merging building instance versions can be advantageously designed and developed on the basis of operative models.

STATE-OF-THE-ART

According to the State-of-the-Art software applications used in the building planning process create and modify structured object sets called application specific models.

STANDARDIZED OBJECT MODELS

The standardization of object models that describe building information is one attempt to support the distributed planning process of buildings. An example for this approach is the introduction of the Industry Foundation Classes (IFC) and the physical exchange format STEP. In practice, however, the cooperation on the basis of standardized object models comes along with many problems. Firstly, a common model that covers all the disciplines tends to be too complex. If it existed, an application would have either to implement the standardized object model and consequently would have to be newly developed or the standardized object models and the application specific models would have to be transformed into one another. The latter is characterized by a cumulative information loss because standardized object models and application specific models cannot be mapped completely onto one another [Firmenich 2004].

Assuming that for each state of the building instance the standardized object model is serialized in a document, a version of the building instance can only be described by a version of a document.

DOCUMENT MANAGEMENT SYSTEMS

Document management systems (DMSs) are used to manage document versions. For that reason DMSs are applied in the planning process for managing and distributing versions of building instances stored in documents.

In DMSs the history of a document and therefore only revisions but not variants of a building instance can be stored (Fig. 2) [Beer et al. 2006].

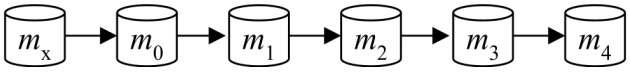

Figure 2: Versions of a building instance on the basis of document history in DMSs

Furthermore, DMSs cannot support the comparing of versions of a building instance since the semantics of the document are not known to the DMS in general [Firmenich et al. 2005].

For the addressed reasons standardized object models in conjunction with DMSs have considerable shortcomings in the iterative and distributed planning process of buildings.

OBJECT VERSION CONTROL SYSTEMS

In the software development process version control systems (VCS) are in use for managing versions of text-based documents. In contrast, object version control systems allow for managing object versions. The development of an object version control system called objectVCS is in the focus of research at Bauhaus University Weimar [Firmenich et al. 2005].

Object version control systems can support the planning process of a building as described below. While existing software applications have to be enhanced by load and store functionality, their specific unversioned models remain unchanged. After loading a version of a building instance as a structured set of object versions the application can reconstruct its native unversioned object model. Afterwards the unversioned application model has to be stored in the object version control system that creates a new version of the processed building instance.

By means of an object version control system in conjunction with existing planning applications a version of a building instance can be described by a structured set of object versions (Fig. 3).

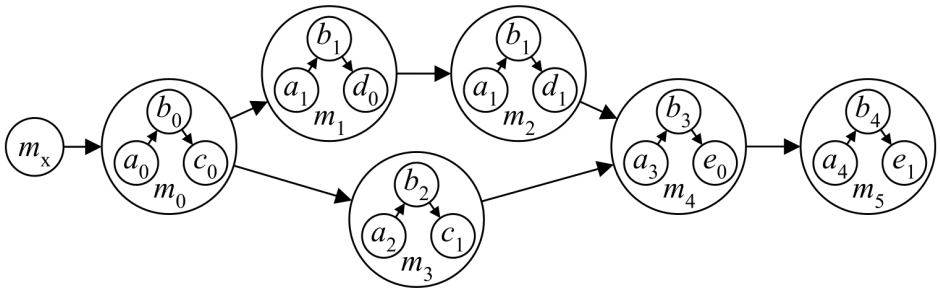

Figure 3: Versions of a building instance on the basis of object version control systems

Comparing and merging versions of a building instance tend to be very complicated. The reasons for this assumption are as follows: Firstly, the stored structured set of object versions might be very complex due to deeply nested model structures and can be hardly compared. For decision making reasons the user should be involved in the merging process. Unfortunately, the user is not aware of the semantics of private attributes that have to be synchronized.

Comparing and merging two versions of the building instance always causes the same effort, independently of considering revisions or variants.

The mentioned problems show that a structured set of object versions is inappropriate for comparing and merging versions of building instances.

SOLUTION APPROACH

OPERATIVE MODELING

In the traditional approach a model instance is represented by objects and attributes. In contrast, an operative model instance is described by the operations applied in the design process. These operations finally lead to the respective building state. While an object model is evaluated, an operative model is unevaluated [Firmenich 2004].

In order to exemplify the concept of operative building modeling a comparison with solid modeling can be drawn. While a BRep model instance describes the topology and geometry of a solid boundary in an evaluated form, a CSG model instance stores an unevaluated description of a solid. The building shown in Fig. 4 shall be abstracted as a solid model instance.

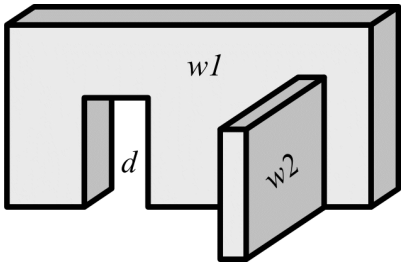

Figure 4: An abstracted building model instance

Using the *Scheme* language of the ACIS solid modeler [Corney et al. 2001] the operations for defining an unevaluated CSG model instance can be formulated in just four lines of code:

1: (define w1(solid:block 0 0 0 10 0.5 5))
2: (define w2(solid:block 8 3 0 9 0 3.5))
3: (define d(solid:block 2 0 0 4 0.5 3))
4: (solid:subtract(solid:unite w1 w2)d)

In contrast, using the BRep modeling approach the evaluated description of the building's solid results in a *SAT* file (Standard ACIS Text) containing almost 300 lines of code:

1: 700 0 1 0
2: 22 ACIS/Scheme AIDE - 7.0 11 ACIS 7.0 NT 24 Thu May 27 09:50:57 2004
3: 1 9.9999999999999995e -007 1e –010
4: body $-1 -1 $-1 $1 $-1 $2 #
...
285: straight-curve $-1 -1 $-1 4 -1.75 -2.5 0 -1 0 I I #
286: point $-1 -1 $-1 4 -0.25 -2.5 #
287: point $-1 -1 $-1 4 -3.25 -2.5 #
288: End-of-ACIS-data

This example demonstrates that an operative descriptions (1) is compact and (2) can be interpreted by users – not only by applications.

DISTRIBUTED WORKFLOW

A prerequisite for the use of operative models in the planning process is the definition of standard operations. Once defined, the standard operations can be used to exchange the building information.

Additionally, existing applications need to be slightly adapted. While the evaluated application building model remains unchanged the application itself has to be extended by journaling functionality. A journaling mechanism is responsible for recognizing changes in the model instance and describing these changes by the standardized operations. The operations are serialized in a journal file as a δ change.

A sequence of journal files represents a version of a building instance. Consequently, building information is exchanged by journal files. Contrary to the traditional data exchange of evaluated models the proposed approach has the advantage of a non-accumulating information loss since the exchanged changes δ remain unchanged [Firmenich 2004].

Besides the journaling mechanism an application has to be extended by an interpreter that applies the operations of the journal file on the application specific building model instance.

Figure 5 illustrates the workflow between planner *A* and planner *B* on the basis of operative building models. Planer *A* starts designing and creates his native application model instance M_{0A}. The journaling mechanism records the applied operations in the journal file Δ_{x0} that represents the first version of the operative building instance. Planner *B* receives the journal file Δ_{x0} from planner *A* and generates the native building instance M_{0B} by applying these changes. While processing the building instance M_{1B} the changes are recorded in the journal file Δ_{01}. The next version of the operative building instance can now be described by the sequence of changes stored in the files Δ_{x0} and Δ_{01}.

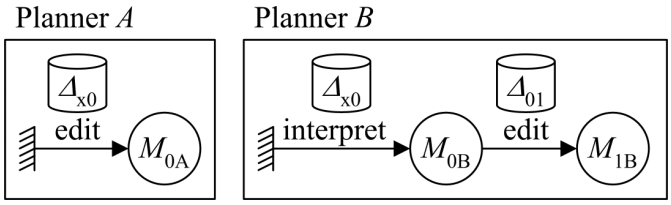

Figure 5: Workflow on the basis of exchanging journal files

MATHEMATICAL DESCRIPTION

Formally, the operative building instance is described by a sequence of applied operations that are contained in one of the sets *A*, *S*, *C* or *R* with

$$A := \{a_i \mid a_i \text{ is an operation that adds an object}\}$$
$$S := \{s_i \mid s_i \text{ is an operation that selects or unselects objects}\}$$
$$C := \{c_i \mid c_i \text{ is an operation that modifies selected objects}\}$$
$$R := \{r_i \mid r_i \text{ is an operation that removes selected objects}\}.$$

The set *O* of all operations applied is defined as

$$O := A \cup S \cup C \cup R.$$

It is assumed that a change δ_{ij} is a sequence of operations $o \in O$

$$\delta_{ij} := \langle o_0, o_1, o_2, \cdots, o_{n-1} \rangle$$

and can be represented as an edge (m_i, m_j) in the version graph. The version graph is a rooted tree with m_x as its root node (Fig. 6). It should be noted that in this approach the tree structure is preserved, even in the case of merges.

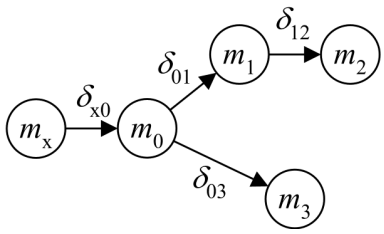

Figure 6: Version graph as a rooted tree

A model instance version m_j is a node in the version graph and can be formulated as a root path starting at the root node m_x. A path is denoted either by a sequence of n edges or a sequence of $n+1$ nodes.

$$rootpath(m_j) := \langle \delta_{x0}, \delta_{01}, \cdots, \delta_{ij} \rangle = \langle m_x, m_0, m_1, \cdots, m_i, m_j \rangle$$

For example, the version m_2 of the model instance in figure 6 is described by the root path $rootpath(m_2) = \langle \delta_{x0}, \delta_{01}, \delta_{12} \rangle = \langle m_x, m_0, m_1, m_2 \rangle$.

COMPARING AND MERGING

The diff and merge approach is based on a general version tree as shown in figure 6. Comparing and joining different versions results in comparing and joining root paths to the respective version nodes. The presented procedure operates on the version tree and is based on graph theory and relational algebra [Pahl and Damrath 2000]. The two versions m_i and m_j are to be compared and merged:

a) The first task is to find the common parent version node m_{common} that leads to the common sub-path

$$rootpath(m_{common}) := \langle m_x, \cdots, m_{common} \rangle.$$

The common sub-path does not have to be considered in the subsequent procedure, because we are interested in the difference and not in the commonness of versions.

b) The next step is to find the differing sub-paths *diffpath*(m_i) and *diffpath*(m_j) that start at the common version node m_{common} and end at the respective version nodes.

$$diffpath(m_i) := \langle m_{common}, \cdots, m_i \rangle, \; diffpath(m_j) := \langle m_{common}, \cdots, m_j \rangle.$$

The differing sub-paths describe the differences between the considered versions.

c) At this stage, the differing sub-paths *diffpath*(m_i) and *diffpath*(m_j) have to be compared. This leads to a semantic comparison of two sequences of operations.

d) Merging the differing sub-paths *diffpath*(m_i) and *diffpath*(m_j) results in a new instance version m_{new}. Therefore, a new edge δ_{new} with

$$\delta_{new} := (m_{common}, m_{new})$$

has to be created. The change δ_{new} describes a sequence of operations that are based on the operations stored in the differing sub-paths. The resulting version m_{new} of the model instance can be described as

$$rootpath(m_{new}) := \langle m_x, \cdots, m_{common}, m_{new} \rangle .$$

Revision example

It is assumed that the version m_0 and its revision m_2 (Fig. 8) have to be compared and merged. The proposed procedure yields the following results:

a) Considering the root paths to the version nodes m_0 and m_2 the common parent node and the common sub-path are

$$m_{common} = m_0 \ , \ rootpath(m_0) = \langle m_x, m_0 \rangle = \langle \delta_{x0} \rangle .$$

b) The differing sub-paths *diffpath*(m_0) and *diffpath*(m_2) correspondingly yield

$$diffpath(m_0) = \langle \ \rangle , \ diffpath(m_2) = \langle m_0, m_1, m_2 \rangle = \langle \delta_{01}, \delta_{12} \rangle .$$

c) Comparing the differing sub-paths results in the fact, that the difference between the empty path *diffpath*(m_0) and the sub-path *diffpath*(m_2) is explicitly described in the sequence of changes stored in the sub-path *diffpath*(m_2). Considering the operations that are described in the sub-path's changes it can be found out which objects have been added, modified or removed. Even the modifications can be described by the applied operations.

d) Merging the differing sub-paths results in joining the empty path *diffpath*(m_0) and the sub-path *diffpath*(m_2). That means the change δ_{new} to be created can only be based on the sub-path *diffpath*(m_2). It is assumed that the new instance version is m_4, its root path can be described as

$$rootpath(m_4) = \langle m_x, m_0, m_4 \rangle = \langle \delta_{x0}, \delta_{04}^2 \rangle \ \text{ with } \ \delta_{04}^2 = \delta_{new} = (m_0, m_4) .$$

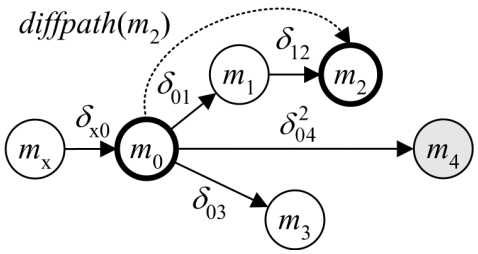

Figure 8: Merging revisions

Variant example

It is assumed that the variants m_2 and m_3 (Fig. 9) have to be compared and merged. The proposed procedure yields:

a) Considering the root paths to the version nodes m_2 and m_3 the common parent node and the common sub-path are

$$m_{common} = m_0 \,,\ rootpath(m_0) = \langle m_x, m_0 \rangle = \langle \delta_{x0} \rangle \,.$$

b) The differing sub-paths *diffpath*(m_2) and *diffpath*(m_3) correspondingly yield

$$diffpath(m_2) = \langle m_0, m_1, m_2 \rangle = \langle \delta_{01}, \delta_{12} \rangle$$

$$diffpath(m_3) = \langle m_0, m_3 \rangle = \langle \delta_{03} \rangle .$$

c) Comparing the differing sub-paths results in a semantic comparison of the changes stored in the respective sub-paths. Therefore, the difference between the variants is the difference between the sequences of operations stored in the changes. These operations reveal the objects added to, modified in or removed from specific versions.

d) Merging the differing sub-paths results in joining the paths *diffpath*(m_2) and *diffpath*(m_3). Thus the change δ_{new} to be created is based on the sequence of changes stored in both sub-paths. If the new instance version is denoted as m_5 then its respective root path can be described as

$$rootpath(m_5) = \langle m_x, m_0, m_5 \rangle = \langle \delta_{x0}, \delta_{05}^{2,3} \rangle \ \text{ with } \ \delta_{05}^{2,3} = \delta_{new} = (m_0, m_5) .$$

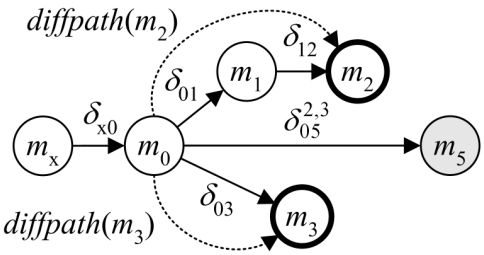

Figure 9: Merging variants

The user has to be involved in the process of merging two model instance versions. The versions are described by operations. Therefore, the user has to indicate the operations to be applied for achieving the new version. At the user's option operations of differing sub-paths can be used or skipped. Also completely new operations can be executed. A tool that supports the merging process has to take care of several specific situations. If operations in a sub-path are skipped the merge procedure has to be aware of the fact that the related subsequent operations might be invalid or even not executable. For example, skipping an operation that adds an object results in not being able to select or modify the considered object afterwards.

EXAMPLE FROM 2D-CAD

Considering a 2D-CAD application a simplified set O of standardized operations looks like the following:

$$O = \{addline, select, unselect, transform, remove\}.$$

The scenario described below tries to clarify the diff and the merge approach on the basis of operative modeling. Figure 10 illustrates the versions of the model instance containing primitive lines only.

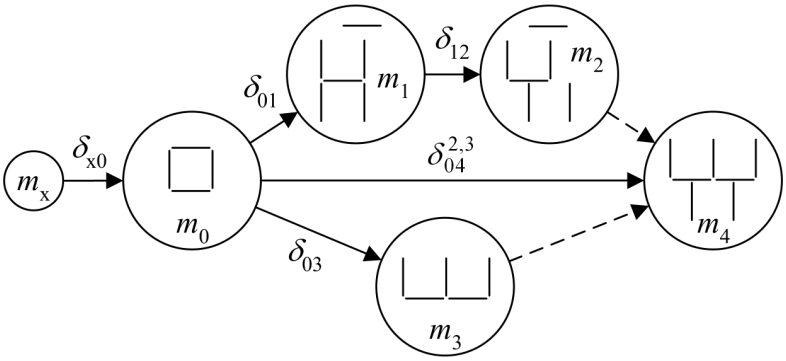

Figure 10: Versions of a 2D-CAD model instance

The changes δ_{ij} used to describe the versions of the operative model instance are assumed as follows:

$$\delta_{x0} = \langle addline\,0,0,1,0,obj0; addline\,0,0,0,1,obj1; addline\,0,1,1,1,obj2; addline\,1,0,1,1,obj3\rangle$$

$$\delta_{01} = \left\langle \begin{matrix} select[obj2]; transform[1,0,0,1,0.5,0.5]; unselect[obj2]; \\ addline\,0,0,0,-1,obj4; addline\,1,0,1,-1,obj5 \end{matrix} \right\rangle$$

$$\delta_{12} = \langle select[obj4,obj5]; transform[1,0,0,1,0.5,0]; unselect[obj4,obj5]\rangle$$

$$\delta_{03} = \langle select[obj2]; remove; addline\,1,0,2,0,obj6; addline\,2,0,2,1,obj7\rangle.$$

The proposed procedure for comparing and merging the versions m_2 and m_3 results in:

a) An investigating of the root paths of the version nodes m_2 and m_3 yields the common parent node and the common sub-path:
$$m_{common} = m_0,\ rootpath(m_0) = \langle m_x, m_0\rangle = \langle \delta_{x0}\rangle.$$

b) The differing sub-paths *diffpath*(m_2) and *diffpath*(m_3) correspondingly yield
$$diffpath(m_2) = \langle m_0, m_1, m_2\rangle = \langle \delta_{01}, \delta_{12}\rangle,\ diffpath(m_3) = \langle m_0, m_3\rangle = \langle \delta_{03}\rangle.$$

c) A semantic comparison of the changes stored in the sub-paths *diffpath*(m_2) and *diffpath*(m_3) yields:
- version m_2: *obj*2 modified; *obj*4 and *obj*5 added and modified
- version m_3: *obj2* removed; *obj6* and *obj7* added

d) Joining the sub-paths *diffpath*(m_2) and *diffpath*(m_3) results in creating the change $\delta_{04}^{2,3}$ on the basis of the sequences of operations stored in the sub-paths. The user's decision is described as

- use(δ_{03}); use(δ_{03}); skip(δ_{01}); …; use(δ_{03}); use(δ_{03}); unselect [obj4,obj5];

Thus, the resulting change $\delta_{04}^{2,3}$ is the sequence of operations

$$\delta_{04}^{2,3} = \left\langle \begin{array}{l} select[obj2]; remove; addline\,0,0,0,-1, obj4; addline\,1,0,1,-1, obj5; \\ select[obj4, obj5]; transform[1,0,0,1,0.5,0]; addline\,1,0,2,0, obj6; \\ addline\,2,0,2,1, obj7; unselect[obj4, obj5] \end{array} \right\rangle$$

CONCLUSIONS

Existing building information modeling approaches have considerable shortcomings in the context of comparing and merging versions of the building instance. As a contribution to these problems our paper describes a novel diff and merge procedure on the basis of unevaluated operative models. However, a lot of research topics remain open – for instance a practicable user interface for the comparing and merging procedure.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of this project by the German Research Foundation (DFG).

REFERENCES

Beer, D. G., Firmenich, B., Beucke, K. (2006). "A System Architecture for Net-distributed Applications in Civil Engineering". *Proceedings of the Joint International Conference on Computing and Decision Making in Civil and Building Engineering*, Montreal, Canada – accepted paper

Corney, J., Lim, T. (2001). *3D modeling with ACIS*. Saxe-Coburg. Stirling

Firmenich, B., Koch, C., Richter, T. and Beer, D. G. (2005). "Versioning structured object sets using text based Version Control Systems". *Proceedings of the 22nd CIB-W78*. Institute of Construction Informatics, Dresden, 105 pp.

Firmenich, B. (2004). "A Novel Modelling Approach for the Exchange of CAD Information in Civil Engineering". *Proceedings of the 5th ECPPM*. A.A. Balkema, Leiden, London, New York, 77 pp.

Pahl, P. J., Damrath, R. (2000). *Mathematische Grundlagen der Ingenieurinformatik*. Springer, Berlin, Heidelberg, New York.