DIFF AND MERGE FOR NET-DISTRIBUTED APPLICATIONS IN CIVIL ENGINEERING

Torsten Richter[1], and Karl Beucke[2]

ABSTRACT

During the planning process in civil engineering collaborative work can be mainly characterized by the exchange of documents. With the new approach of versioning structured object sets on a central server many planers can work together in parallel without any locking mechanism. Thus, different variants can be developed. Before a commit of a drawing from a client to the server it has to be merged with the last version on the server repository. Therefore, the problem of the comparison (diff) and the merge has to be solved.

This paper presents different solution approaches for the diff and merge. Three kinds of mergers will be explained to demonstrate the development chronology. The third one is integrated in an application and aimed at a mature usability by hiding the internal object-oriented structure of the data model from the user. The concepts are implemented and verified for a CAD application.

KEY WORDS

collaborative work, versioning, diff and merge, user interface.

INTRODUCTION

Through the spreading and availability of the Internet the distributive work aided by computer software has been enabled but is not used adequately. Especially the planning processing in civil engineering would benefit from a parallel and consistent collaboration. But in our days the cooperation is either sequential or parallel with a laborious merge of documents. One fact for that is the missing storing of object references and bindings as well as the missing compare (diff) and merge functionality.

Normally, the members of a project exchange their data by sending an email containing a document as attachment or by using a document management system. A comparison of document versions is often not supported inside engineering applications. Moreover, a merge of documents from different planers is not provided at all.

In the field of software development the software configuration management (SCM) based on versions and relations has been established as a standard. Combining SCM with the versioning of objects by serializing arbitrary object models to XML files leads to a

[1] Research Assistant, Informatik im Bauwesen, Bauhaus-Universität Weimar, Coudraystraße 7, D-99425 Weimar, Germany, Phone +49 (03643) 58-4221, FAX +49 (03643) 58-4216, torsten.richter@bauing.uni-weimar.de

[2] Professor, Informatik im Bauwesen, Bauhaus-Universität Weimar, Coudraystraße 7, D-99425 Weimar, Deutschland, Phone +49 (03643) 58-4215, FAX +49 (03643) 58-4216, karl.beucke@informatik.uni-weimar.de

synchronous planning without locking mechanisms. This approach was presented by [Firmenich et al. 2005].

The objective of this contribution is to explain different merge concepts, preconditions for them and the integration into existing applications.

STATE OF THE ART

DIFF AND MERGE OF TEXT FILES

In the field of software development a comparison of two file versions is an often-used feature whether during an update or just to recapitulate the own changes since the last update.

The aim of a diff of text files is to find the largest areas with no changes or the smallest areas with changes. The diff algorithm solves the "longest common subsequence" problem [Hunt 1976, Myers 1986]. Three cases of changing can occur:

- New line(s) was (were) added sign: A
- Line(s) was (were) deleted sign: D
- Line(s) was (were) changed sign: C

External, specialized tools like ExamDiff are often used for this purpose. Figure 1 shows this program in action during a comparison of two text files. Changed lines are highlighted by different colors depending on the three cases above.

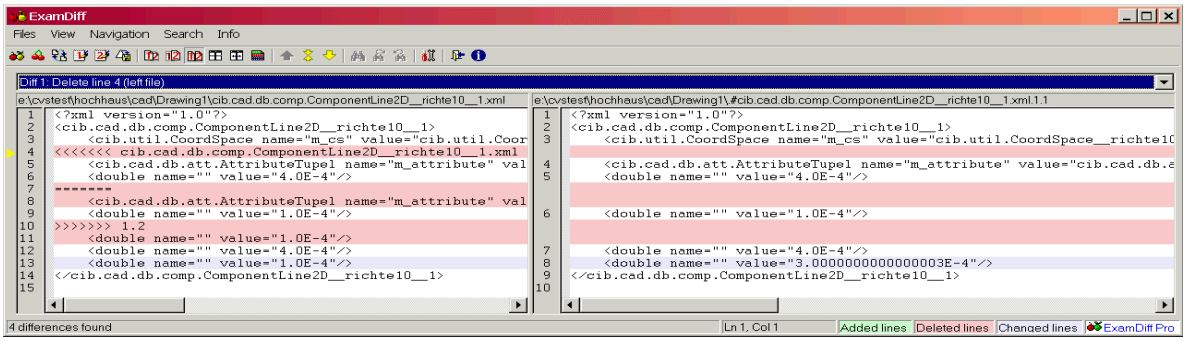

Figure 1: Text-based diff

Most of the textual version control systems (VCS) like the concurrent versioning system (CVS) offer a tool that can apply an automatic merge. But for some cases, shown in Table 1, the merger detects a not solvable conflict. The programmer has to resolve the conflict by understanding the semantic of the source code. A commit requires the resolving of all conflicts. A compiler can prove the correctness of the merge result.

Table 1: Synchronization cases of a text-based VCS

	File was not changed on the server	**File was changed on the server**
File was not changed locally.	–	Automatic merge.
File was changed locally.	No merge needed. Commit is possible.	Different lines: Automatic merge. Same lines: Conflict. Manual merge.

DIFF OF STRUCTURED OBJECT SETS

Model dependent diff

Since few standard applications support the comparison of its own documents external tools were developed to support this functionality. Comparing documents of one application offers the possibility to access and interpret the data object model directly. Otherwise the diff and merge is only as powerful as the features provided in the file format.

In the field of CAD the tool compareDWG (Furix 2006) was examined, that is specialized on DWG files. DWG can be characterized as a standard in the building industry. CompareDWG is integrated into AutoCAD and creates a temporary drawings containing whether unchanged, changed, new or deleted objects. The object handles, the geometry, element attributes and drawing tables are evaluated for the comparison.

The following shortcomings were determined:

- AutoCAD supports handles for the objects, which fulfills one strong precondition of the diff and merge, but unfortunately they are not unchangeable during the lifetime of the drawing.
- Handles are only unique inside one drawing. Therefore, different objects can have the same identifier and will be marked as changed.
- Some changes are differently interpreted. A linear translation of a line was marked as changed but not that of a volume shape. Altering the style of a component was marked as changed but the altering of the general style was not denoted.
- Drawing objects are aggregated to one block on layer zero during the comparison, which destroys all information of the original drawing.
- Altering the scale of the drawing leads to complete change of all components.
- The comparison is only applicable for the model area inside AutoCAD but not for the paper area containing the print view.

Because of the faults with the consistency and the global object identification a merge of compared drawings is not supported. An essential point is the need of persistent object identifiers that are valid in the global identifier space. This is to find corresponding object versions that have to be compared and merged.

Model independent diff

Another approach uses the model independent diff by generating more general and standardized output formats like PDF or plot files. Hence, the diff is not restricted on one application. Otherwise the structure of the model is lost. The diff can only refer to components and their features stored in the output file. Furthermore, a merge is impossible.

The tool PlanDiffViewer (WeltWeitBau 2004) as a representative of this concept was examined, which is able to compare two plot files of the HP-GL/2 format. HP-GL/2 is also used as exchange format for vector data because of its simplicity. The following conclusions can be drawn.

- Due to the open plot file format HP-GL/2 drawings of all supporting applications can be compared.
- The PlanDiffViewer is a good choice for the fast overview of changes between two plans.
- Different virtual plotters are using different instruction sets and interpret HP-GL/2 variedly. Slight divergences may be interpreted as changes although they do not exist.
- HP-GL/2 only knows basic geometric shapes like lines or circles. Hence, the information of the original CAD components is lost. Furthermore, and this a main disadvantage, the missing of object identifiers prevent a merge of two drawings.
- The reconversion of HP-GL/2 files into the original drawing format is not possible.

Short resume of existing diff tools

As shown before the comparison of binary documents is partially solved either by transforming them to standardized output format or by writing specialized tools for proprietary applications. Both solutions failed with the merge because of a missing or inconsistent object identification and other shortcomings.

DIFF AND MERGE OF STRUCTURED OBJECT SETS

The examination of existing diff tools arrives at the conclusion that a merge is not supported. This attributes to shortcomings in the missing or insufficient object identifications provided by the application. This chapter presents a novel architecture that remedies this deficiencies and the development of a new merge method enabled by the new architecture.

OBJECTVCS WITH TEXTUAL DIFF AND MERGE

[Firmenich et al. 2005, Beer 2006] have presented a novel approach to version structured object sets according to text-based SCM. The name objectVCS stands for object version control system. The first prototype of objectVCS was developed on [CADEMIA 2006] to show the benefits of net-distributed work at the example of computer-aided design. CADEMIA is an engineering platform for geometry-based applications.

Beside the common way of storing, existing software applications will be enhanced by the functionality of objectVCS. The system architecture is shown in Figure 2. Each client has a sandbox where a directory represents one document. The directory contains one version of each object stored as textual XML file. The transfer from the transient object model to the persistent object model in the file system is done by the Java serialization mechanism. Every object is assigned a unique POID (persistent object identifier) consisting of the data type, user name and a consecutive number that is also used as file name. Each attribute of an object is stored whether it is a simple or a complex data type. Therefore, all object references are kept. The loading uses the inverted way of Java serialization called deserialization.

The synchronization between server and sandbox is similar to the SCM with commits and updates. New versions of changed or added objects are created in the repository. Afterwards the tagging mechanism preserves the consistence of different object models by marking the appropriate object versions. Finally, this new release becomes public for all connected users.

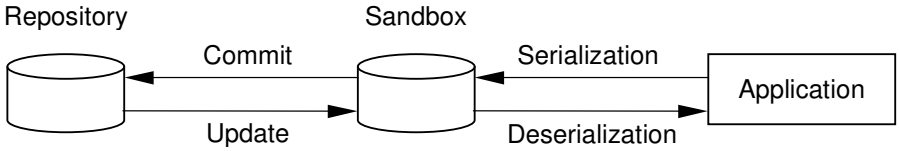

Figure 2: System architecture of objectVCS

In the first implementation of objectVCS no semantic merger was integrated. Hence, the user has to know the internal structure of the object model used by the application. Table 2 shows all possible variants of object attributes.

Table 2: Different kind of object attributes

	Single valued type	**Multi valued type**
Simple data type	int, double, String, ...	set, map, ... of simple data types
Complex data type	object references	set, map, ... of object references

The example below shows an extract from a partly merged file of an object version containing a conflict between line 4 and 10. Line 5 and 6 represent two attributes of the line component in the sandbox whereas ‘m_attribute’ is an object reference and the second one stores a coordinate. Line 8 and 9 represent the attributes of the same line, but with the changed values from the current version in the repository. During the textual merge the user has to edit this file and to leave the appropriate lines.

```
 4: <<<<<<cib.cad.db.comp.ComponentLine2D__richter__1.xml
 5:    <cib.cad.db.att.AttributeTuple name="m_attribute"
          value="cib.cad.db.att.AttributeTuple__richter__1"/>
 6:    <double name="" value="-4.0E-4"/>
 7: =======
 8:    <cib.cad.db.att.AttributeTuple name="m_attribute"
          value="cib.cad.db.att.AttributeTuple__richter__3"/>
 9:    <double name="" value="-1.0E-4"/>
10: >>>>>>> 1.2
```

As shown in this small example, this kind of merge is unacceptable for an application user. The following disadvantages can be located:

- The user has to know the class structure of the object model.
- The object representation differs from the representation in the application, because the object model will be deserialized only after the merge has finished.
- Consistency cannot be assured by editing the serialized object text files. The changes must not infringe the syntax and the semantics of the object model.
- A characteristic of a VCS system is the sequential merge. During this procedure some files are updated and some are not. This makes this process more difficult.

ATTRIBUTE VIEWER FOR DIFF AND MERGE

The first solution approach resulted in an object merger that operates on instantiated object models. The merge mechanism of the VCS has been therefore ignored. As precondition two serialized object models in two different directories are needed. After the deserialization both are transient in the memory and can be traversed by the help of the Java reflection package. Beginning from the root object every other object can be reached via references stored in the attributes. As a result of the traversing an object graph exists.

Corresponding versions of an object can be identified on the basis of the immutable POID. The comparison and merge of two object versions can be supported within a dialog box. The kind of merging depends on the attribute type (Table 2). Figure 3 shows an example for merging a map. The upper window displays the way in the object graph to this map and the lower three windows the members of this map in the first, second and resulting object model. The user can decide which members, in this case they are lines, will be added by selecting it with the mouse.

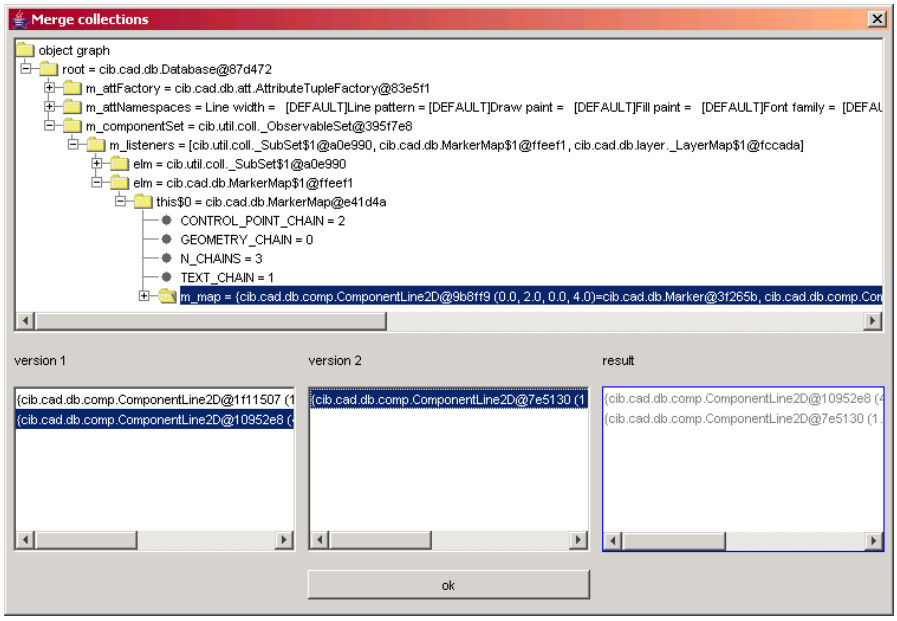

Figure 3: Merge of a map within the attribute viewer

Although this approach allows a more comfortable way of performing a diff and merge it has still some incising disadvantages. The consistency problem is not solved yet, for instance objects could be inscribed into more than one container or the disregard of data encapsulation may lead to wrong attribute values inside an object. Another question leads to decision what objects and object attributes are important for the user. He surely does not want to deal with the settings of internal objects.

NOVEL METHOD FOR DIFF AND MERGE

The next evolution step of the previous approach is the integration of the attribute viewer concept into an existing application. The aim is to hide the internal structure of the data model and to show the components in the usual representation. Hence, the merger has to know the structure and the organization of the object-oriented data model. The procedure of merging shall be explained at a simple scenario for an easier understanding. It consists of 6 steps shown in Figure 4.

I. User A draws a little house h with a door d on the left side.
II. Afterwards he commits the plan to the repository on the server.
III. A second user B checks it out and works in parallel with user A on the same plan.
IV. User B moves the door to the right side and adds a window w_2 on the left side. Meanwhile, user A adds window w_1 at a different position.
V. User B commits his changes back to the repository.
VI. User A wants to commit his plan the application forbids this action because of the new version on the server. Consequently, both have to be merged after an update.

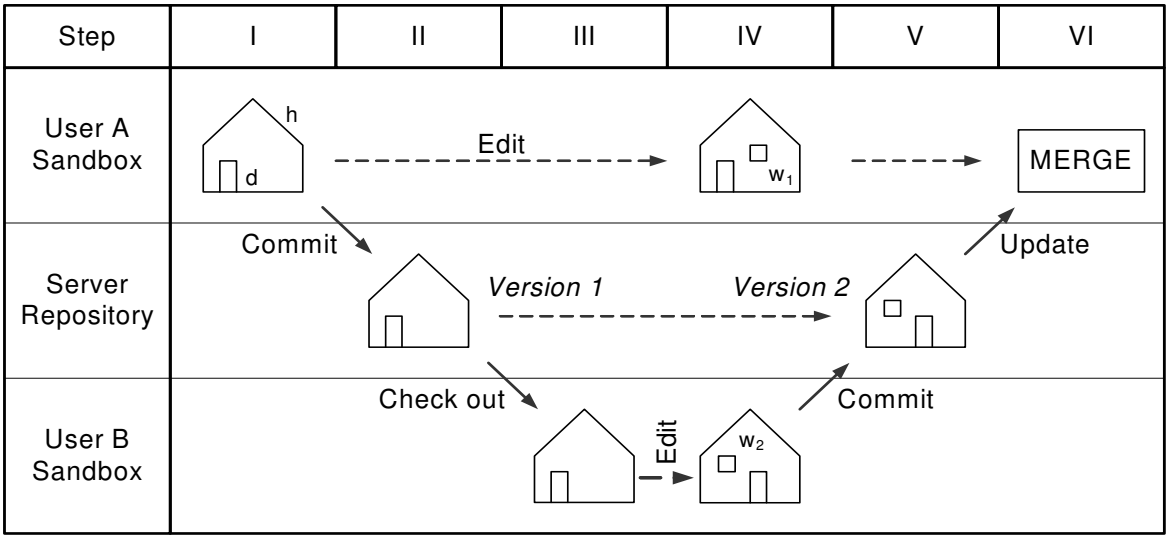

Figure 4: Planning scenario with two users

It is assumed that the local data object model of user A will be merged with a second one from the server, which remains unaltered during this process. A new object model will exist in the sandbox of user A after the merge, which can still be edited and committed afterwards.

A novel merger was developed that can operate on the object model and uses CADEMIA functionality like the undo/redo mechanism. In this implementation the importance was attached to an easy handling by the user. According to phase VI this merger will be started after the update task. The graphical user interface can be seen in Figure 5. The window is divided into three parts. The left part contains a list with all CAD components identified by its POID. The filters above can reduce the components in the list regarding to unchanged, changed, added and deleted state. The two right parts show the visualized object models whereas the lower one is version 2 from the server and the upper one is the current version from the sandbox of user A.

The differences between both drawings are identified by the change states of the components. Each change state is visualized by an own color. Coming back to the scenario user A has to take the following steps.

- House h is marked as unchanged. Therefore, this component remains unaltered and cannot be changed by the user.
- Door d is marked as changed. User A may leave his version, replace his version by that from the server or set new coordinates for this door with an appropriate feature dialog.
- Window w_1 is marked as added. User A may leave w_1 or select and discard it.
- Window w_2 is marked as deleted. In this case this window was created on the server, but it could also mean that this window was removed locally. User A may ignore window w_2 or drag and drop it with the mouse into his drawing.

- Finally, user A has to store his drawing by overwriting the existing one in the sandbox. Afterwards he can commit this plan as version 3 to the repository.

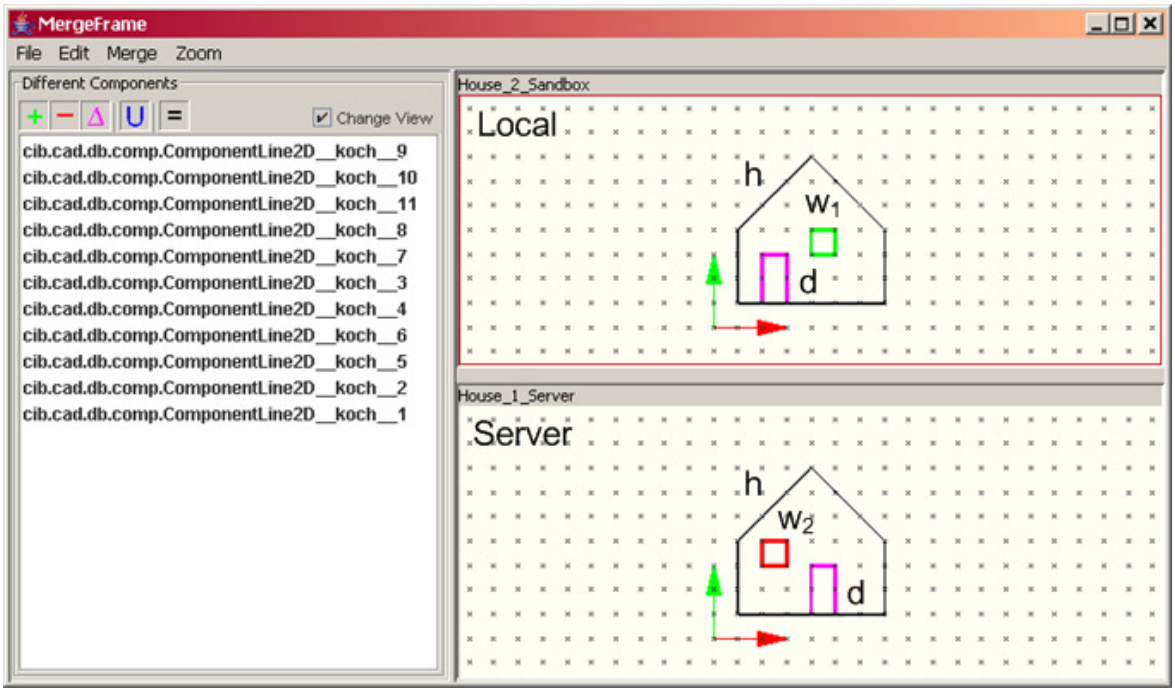

Figure 5: Object merger integrated in a CAD application

The main problem to be solved for a diff and merge is to retrieve those objects or components that are affected and references that have to be considered to preserve the consistency of the data object model. For each of the two object model instances exists a map storing relations between the POID and the transient Java handle. That allows to parse the object graph for versions of the same object and to recognize differences between the versions. Six merge cases for an object have to be distinguished:

- The object was not changed. No merge is needed and the POID does not change.
- The object was changed either on the server or locally. The user has to select one object version. The POID does not change.
- The object was added on the server. Either the user takes over this object or not. The POID from the server will be added for the first case.
- The object was added locally. Either the user discards this object or it will be finally added. A new POID will be assigned.
- The object was deleted on the server. Either the user accepts the erasure or keeps the object. The POID will be removed from the model for the first case.
- The object was deleted locally. The user can withdraw the erasure of the object or delete it. The POID will be removed from the model for the second case.

Depending on the case we have to establish, remain or delete a mapping between the handle and the POID in the local model. Furthermore, references have to be considered.

CONCLUSIONS

The diff and merge of documents are essential for the collaborative work. With the concept of versioned object models a persistent identifier (POID) is assigned to each object. The unequivocal object identification is a precondition for a consistent diff and merge. Differences between two object models can be obtained by traversing the object graphs and comparing the attributes of objects with the same POID. Encapsulating this process by a mature graphical user interface eases the use of the diff and merge functionality.

For the future research the merge of bounded object versions has to be investigated. The question to solve is to identify kind of bindings and to clarify the appropriate merge case.

ACKNOWLEDGMENTS

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

REFERENCES

Beer, D. G. (2006): A System Architecture For Net-Distributed Applications in Civil Engineering. *ICCCBE 2006*. Montreal, Canada.

CADEMIA (2006) – URL: http://www.cademia.org/

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

Furix (2006) – URL: http://www.furix.com/products/compare/

Hass, A. M. J. (2003). "Configuration Management Principles and Practice". The Agile software development series. Boston, Addison-Wesley.

Hunt, J. W., McIlroy, M. D., Douglas, M. (1976). "An Algorithm for Differential File comparison". 41. Computing Science Technical Report, Bell Laboratories.

Myers, E. W. (1986). "An O(ND) Difference Algorithm and Its Variations," Algorithmica 1, 251-266. – URL: http://citeseer.ist.psu.edu/myers86ond.html

Sutton, M. J. D. (1996). "Document Management for the Enterprise Principles, Techniques and Applications". New York, John Wiley & Sons.

WeltWeitBau (2004) – URL: http://www.weltweitbau.de/de/PlanNet/PlanDiffViewer/PlanDiffViewer.html