Versioning structured object sets using text based Version Control Systems

B. Firmenich & C. Koch
CAD in der Bauinformatik, Bauhaus-Universität Weimar, Weimar, Germany

T. Richter & D. G. Beer
Informatik im Bauwesen, Bauhaus-Universität Weimar, Weimar, Germany

ABSTRACT: With the availability of an affordable and ubiquitous network environment the distributed co-operation of projects can be supported by computer software. Currently, the degree of support of a distributed cooperation is very different in the diverse classes of applications. While in the field of text-based applications the synchronous distributed cooperation is already state-of-the-art, the users of document-based applications can currently only cooperate asynchronously in terms of a workflow by exchanging documents. This contribution describes a solution approach for the re-use of existing document-oriented applications in net-distributed processes. The synchronous cooperation is realized by a novel procedure that stores the structured object sets of existing single user applications in version control systems, where the well proven tools of the software configuration process can be used in distributed construction planning processes as well.

1 INTRODUCTION

The construction planning process is characterized by the synchronous cooperation of a distributed team that develops a joint solution – the building instance. In practice, document management systems (DMS) are frequently used in this process to control the access and the workflow of the documents of single user applications.

Although the planning process has a lot in common with the software development process, DMSs are almost completely irrelevant in the software development process. Instead of that, cooperation in a software development team is controlled by version control systems (VCSs).

The kind of cooperation between a DMS and a VCS is completely different: While the former only supports the asynchronous cooperation the latter enables a true synchronous cooperation, either parallel or reciprocal (Fig. 1, Bretschneider 1998).

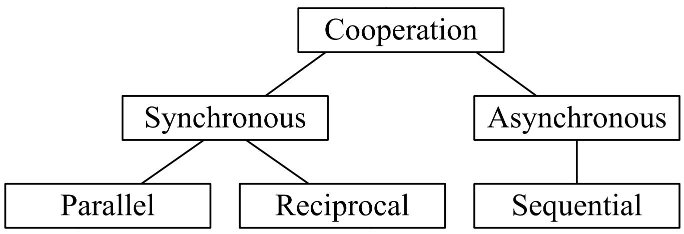


Figure 1. Classification of cooperation according to time

The goal of this contribution is to close the gap between these two worlds and to allow the tool-based VCSs to be used in conjunction with existing single user applications in the distributed planning process.

2 STATE-OF-THE-ART

It is distinguished between document based and text based applications. For these two application classes the local and the net-distributed processing are outlined according to the state-of-the-art.

2.1 *Document-based applications*

According to the state-of-the-art only applications that deal with structured object sets are considered (Fig. 2). The objects can only be accessed in the process that created them. Therefore, the object set has to be stored persistently in a document before terminating the process. Other processes can later reconstruct the transient object set via the contents of the document. Typical examples of document based applications are word processors, CAD and FEM programs.

Currently, a net-distributed cooperation of document based applications is enabled by a DMS (Fig. 3). A DMS manages documents of any format. Relationships between objects in different documents cannot be supported since the semantics of the document content are generally not known to the DMS. The goal of a DMS is the support of the workflow on the basis of a shared document pool. If a team member wants to edit a document a copy of that document is transferred to the user's local file system. When the editing process is completed a new copy of the document can be stored and published in the shared document pool.

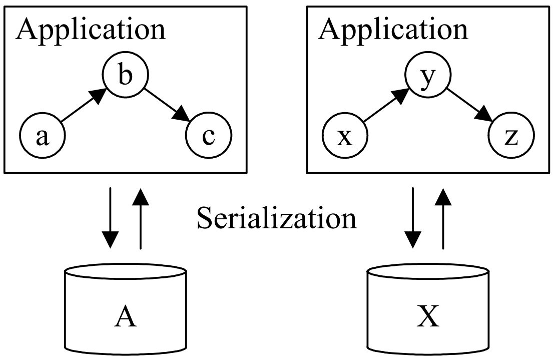


Figure 2. Document-based applications

Because existing applications tend to become increasingly complex the stored documents have a considerable file size as well. Differences between documents are not stored explicitly in a DMS. The users have to localize, visualize and merge differences by the help of very complex tools.

A DMS supports the asynchronous cooperation of a team where in general the tasks must be performed in sequential order. However, since no relationships exist between objects in different documents a restricted parallel cooperation is possible.

The DMS solution has the advantage that arbitrary document based applications can be integrated without any program adjustment.

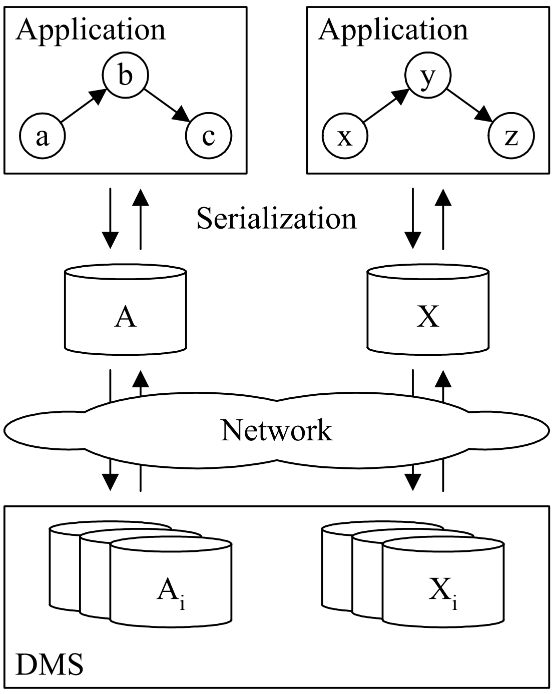


Figure 3. Document Management System (DMS)

2.2 *Text-based applications*

Text based applications are found in the software development process where the source code is stored as plain text in files. Figure 4 shows how the source code is managed by text editors.

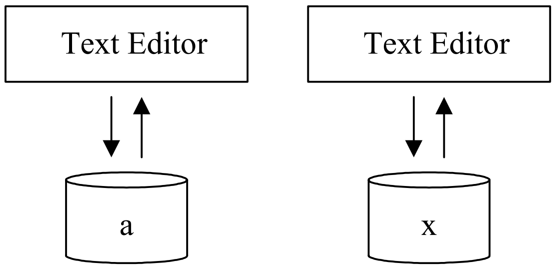


Figure 4. Text-based applications

In addition to the text editors many other tools for the processing of text editors exist. The availability of many tools is typical for a text based environment. The discipline of software configuration management (SCM) deals with the process of creating complex software systems. Figure 5 shows how the distributed software development process is supported by a VCS. Since the syntax of the file content – plain text – is known to the VCS many useful tools like *diff* and *merge* are available.

A VCS typically manages a multitude of different versions of text files that have a much smaller file size compared to documents stored in a DMS. Instead of storing the complete file contents, only the changes between file versions are very efficiently stored. Due to the requirements of the software development process the reciprocal synchronous cooperation is supported. Here, even tasks that synchronously refer to source code in the same line of the same file version must be allowed.

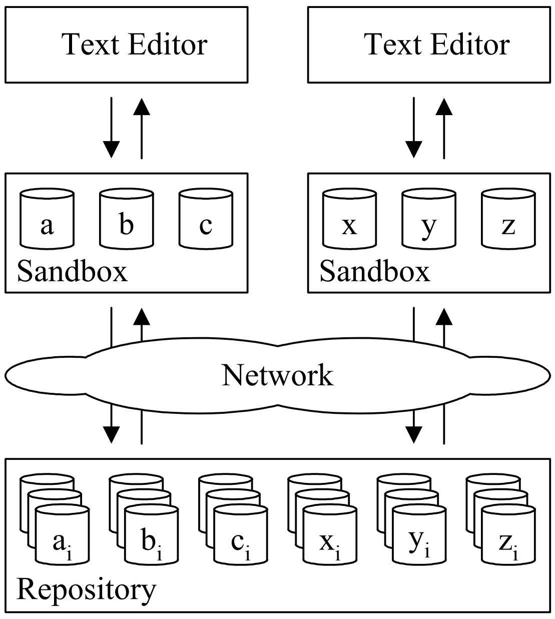


Figure 5. Version Control System

All file versions of the project are stored in a central Repository that is accessible to all team members. Each file version can be added to an arbitrary number of file version sets by the tagging mechanism. Equation 1 below defines the file version set T_i formally:

$$T_i := \{f \in V \mid f \text{ has tag } i\} \subseteq V \qquad (1)$$

where V is the set of all file versions and i is a specific tag. The tagging mechanism allows the VCS to manage configurations of file versions that belong together.

Files cannot be directly edited inside the Repository but must be previously transferred as a new copy to the user's Sandbox. The Sandbox is located in a directory of the local file system. Since only one version per file can be stored in the Sandbox existing (unversioned) single user applications can be integrated in the distributed process. The user's work is persistently stored as new file versions in the Repository. The net-distributed cooperation depends solely on the synchronization of the Sandboxes and the Repository by the VCS that provides a set of commands for this task. Conflicts are either solved automatically or by actively involving the participating users.

3 SOLUTION APPROACH

The solution approach is targeted at the reuse of existing document based applications during synchronous cooperation. This objective is achieved by storing the application's structured object set in a text based VCS. The procedure proposed is named *objectVCS*.

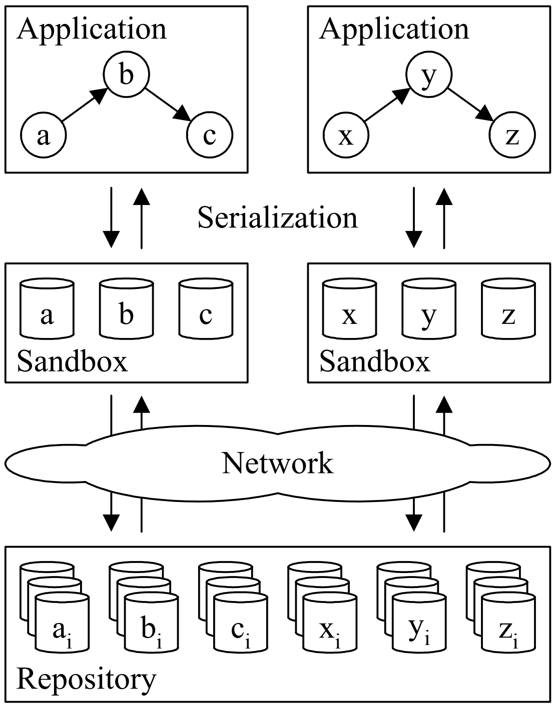


Figure 6. objectVCS: object Version Control System

Contrary to the state-of-the-art the unit of representation is not the document but the object with its attributes and relationships. Storing the objects in a VCS allows the tools of this VCS to be applied to the stored object set. This is particularly true for the tools that support the distributed synchronous cooperation and the versioned data management.

Figure 6 shows the system architecture of objectVCS.

3.1 *Serialization*

This mechanism establishes the interface between the document based applications and the text based VCS. The serialization is the method of representing a structured object set as a character stream, the deserialization in turn is the method of creating a structured object set from a character stream. Unlike the DMS approach objectVCS always requires a certain extension of the participating applications. While the former represents the object set in one single document (available functionality of a document based application) the latter requires that each single object is stored in a separate text file.

objectVCS knows the mapping between objects and files. A unique persistent name (*POID: persistent object identifier*) is assigned to each object and all its versions (Beer et al. 2004a,b). The object is stored in a file named after the POID. Objects are instances of different data types. It is distinguished between single valued and multi valued data types.

The serialization of an object of a single valued type consists of writing the data type followed by the object attributes as (*type*, *name*, *value*) tuples. There is a difference between a primitive and a reference attribute whose value is the POID of the referenced object.

Figure 7a shows the parameters of a circle as an example.

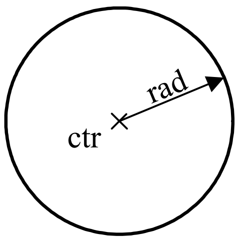


Figure 7a. Parameters of the circle

The related Java classes are listed in Figure 7b: The circle's center is represented as a Point object with two primitive attributes x and y. The Circle object has a primitive attribute for the radius and a reference attribute for the center point.

The serialized XML files are shown in Figure 7b. The POIDs are *PointInst* for the point and *CircleInst* for the circle.

```
class Circle {
  Point ctr;
  double rad;
  Circle(Point p, double r) {
    this.ctr = p;
    this.rad = r;
  }
}
```

```
class Point {
  double x;
  double y;
  Point(double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

```
class Application {
  static void main(String[] args) {
    Circle cir = new Circle(
      new Point(1.2, 1.8), 1.5);
  }
}
```

Figure 7b. Java classes to represent the circle

```
<?xml version="1.0"?>
<CircleInst>
  <Point name="ctr" value="PointInst"/>
  <double name="rad" value="1.5"/>
</CircleInst>
```

```
<?xml version="1.0"?>
<PointInst>
  <double name="x" value="1.2"/>
  <double name="y" value="1.8"/>
</PointInst>
```

Figure 7c. XML files to represent the circle

Objects of a multi valued data type are represented as a sequence of elements, where an element can either be a primitive value or a reference value as a POID. The serialization consists of writing the data type, a signature for the multi valued data type and the sequence of elements.

Sets represent a special case: Even though the order of elements is arbitrary, sets are always serialized in a fixed sequence: This algorithm has been convenient for the representation of sets in a text based VCSs.

The Java language has been selected for implementation. The serialization could be realized without any intervention of the application programmer exclusively on the base of the reflection package.

XML was selected as text format. The implementation was very efficient thanks to available XML tools like parsers. It should be noted that objectVCS is not limited to text files. Since the serialization interface is based on character streams, the object set can be stored in data bases as well.

3.2 *Repository*

The text file versions of the project are stored in the Repository. Between the file versions relationships exist. objectVCS ensures the referential integrity by adding object versions to the same version set if they belong together.

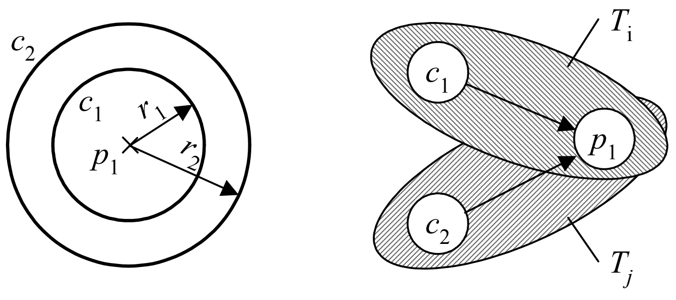


Figure 8. Building object version sets

Figure 8 shows an example for building version sets. The first version c_1 of the circle has a primitive radius attribute r_1 and a reference attribute for the center point p_1. The second version c_2 of the circle has a changed radius r_2 but still refers to the same center point object p_1. Equation 2 describes the definition of the two version sets by tagging:

$$\begin{aligned} T_i &:= \{f \in V \mid f \text{ has tag } i\} = \{c_1, p_1\} \\ T_j &:= \{f \in V \mid f \text{ has tag j}\} = \{c_2, p_1\} \end{aligned} \qquad (2)$$

where V is the set of all file versions, i and j are specific tags and T_i and T_j are the respective version sets.

Available applications store their documents as a single file in the file system. This is shown in Figure 9a where rectangles denote directories. This procedure is not appropriate for objectVCS that depends on storing each single object in a separate file. As a solution to this problem the document is not stored as a file but as a directory representing at most one single document. This procedure ensures that the known mode of operation can be retained by the users.

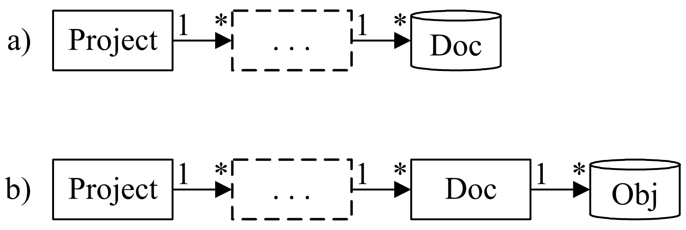


Figure 9. Storage of a document in the file system (a) and in the Sandbox/ Repository (b)

In addition to the well known document based approach a novel kind of processing across document boundaries is possible. Links between objects in different documents can be realized by adding the referencing and the referenced object to the same version set. For the theoretical foundations of the management of structured object version sets the user is referred to (Firmenich 2002a, b, Firmenich & Beucke 2002). An approach for defining consistent releases on a structured object version set has been proposed by Beer & Firmenich (2003).

3.3 *Sandbox*

The sandbox is located in a directory of the user's local file system. The structured object set of the application can be stored to and loaded from the Sandbox. The Sandbox must be synchronized with the Repository.

3.4 *Synchronization of Repository and Sandbox*

The synchronization of the Sandboxes and the Repository is the basis for the net-distributed cooperation. The VCS is responsible for this task. The process of storing the application's object set consists of two phases:

1 Each object is serialized into a separate file stored in the Sandbox. Storing can be done online and offline.
2 The changes become public by storing the files as a new versions in the Repository.

Likewise, loading the structured object set is also a two-phase process:

1 Selected files are transferred from the Repository to the Sandbox.
2 The applications' object set is deserialized from the files stored in the Sandbox. Loading can be done online and offline.

As a matter of principle, all tools of the underlying VCS are applicable in an objectVCS environment.

3.5 *Version Control System and API*

The selection of the free Concurrent Versions System *CVS* (Fogel & Bar 2002, Vesperman 2003, cvshome 2005) as the underlying VCS has proved of value. The available Java CVS Client (javacvs 2005) of Suns Netbeans IDE open source project was chosen for the prototypical implementation of objectVCS. The CVS client allows the establishment of a connection with the CVS Server and the execution of the subsequently described CVS commands:

– *add:* Adds a file or directory to the Repository.
– *checkout:* Creates a new Sandbox or actualizes an existing Sandbox.
– *commit:* Writes changed files from the Sandbox to the Repository.
– *diff:* Shows the differences between two file versions.
– *log:* Shows versioning and administration information about files located in the Sandbox.
– *remove:* Removes a file or directory from the Repository.
– *status:* Shows status about files.
– *tag:* Tags a file or a set of files with a unique name.
– *update:* Transfers changed files from the Repository to the Sandbox. This command allows the reciprocal synchronous cooperation.

4 VERIFICATION EXAMPLE

The solution approach has been verified as an extension of an available single user CAD system that is currently being developed at Bauhaus University for education and research (Firmenich & Beucke 2005).

4.1 *Implementation*

The CAD system has been extended by a set of commands for the synchronous cooperation of a team. The implementation of these commands is based upon our proprietary Java XML serialization package and the available Java CVS Client package.

During development of objectVCS it turned out that extended functionality concerning CVS was needed. An example is the synchronization of a single directory between Sandbox and Repository for performance reasons. The extended functionality has been realized as a wrapper class of the existing Java CVS Client.

The following CAD commands refer to directories and files of the Sandbox only. Online or offline execution is possible:

– *Project settings:* Defines a project as a CVS module in the Repository.
– *New:* Initializes a new drawing in the application.
– *Load:* Loads a document from the Sandbox into the application.
– *Store:* Stores the application's object set in the loaded document of the Sandbox.
– *StoreAs:* Stores the application's object set in a new document in the Sandbox.

The subsequently listed CAD commands refer to the Repository and can only be executed online.

– *UpdateNewDirectories:* Updates the Sandbox by directories stored in the Repository only.
– *UpdateOverride:* Replaces a document in the Sandbox by the head version of the Repository.
– *Update:* Issues the *update* command and performs a merge in case of conflicts.
– *Commit:* Synchronizes the Sandbox and the Repository by an *Update* command. Both new and changed files are stored in the Repository.
– *Server settings*: Sets the password, the local path of the Sandbox and a *diff* and *merge* tool.

4.2 *Scenario*

Figure 10 is the schedule of the verification example. The two Sandboxes *A* and *B* and the Repository are shown as vertical bars very similar to a UML sequence diagram (Rumbaugh et al. 2001).

The time coordinate proceeds downward. Labeled arrows describe the user action and the direction of data transfer. For instance, arrows starting at the Sandbox bar and ending at the Repository bar denote operations that transfer data from the Sandbox to the Repository. Arrows starting and ending at the Sandbox bar denote operations referring to the Sandbox only. Circles with text denote versions of Sandbox or Repository data. For instance, *A2* denotes the second version of data in Sandbox *A* and *R4* denotes the fourth version of data in the Repository.

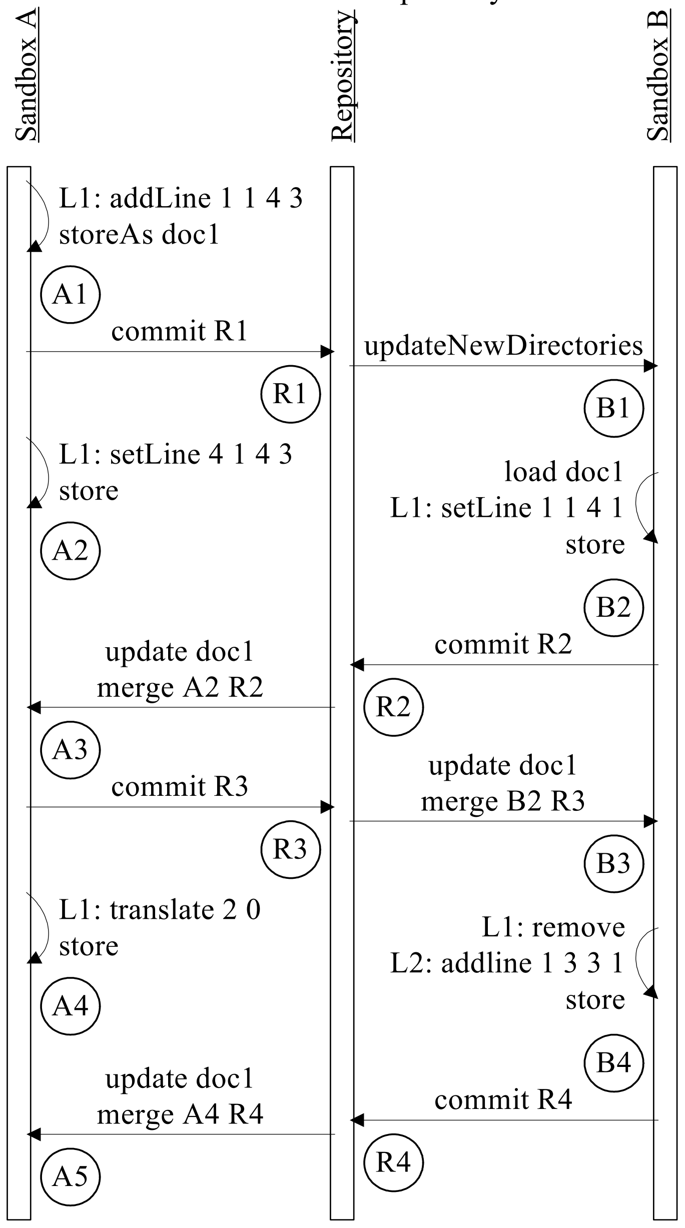


Figure 10. Schedule of the verification example.

Version A1
User *A* adds a line component to the database and stores version *A1* of document *doc1* in Sandbox *A*. The document contains a single line *L1* at (1, 1, 4, 3).

Version R1
User *A* commits the document as version *R1* from the local Sandbox to the Repository.

Version B1
User *B* updates his local Sandbox *B* by new directories created in the Repository. This operation stores version *R1* of the document in his Sandbox.

Version B2
User *B* loads the document version *B1* and changes the endpoint of line *L1* to (4, 1). The document is then stored locally as version *B2*. It contains a single line *L1* at (1, 1, 4, 1).

Version A2
After committing the document as version *R1* user *A* has continued the editing process. Thus, user *A* and user *B* are cooperating synchronously on different versions of the same document.

After changing the start point of the line to (4, 1) the document is stored as version *A2* containing a single line *L1* at (4, 1, 4, 3).

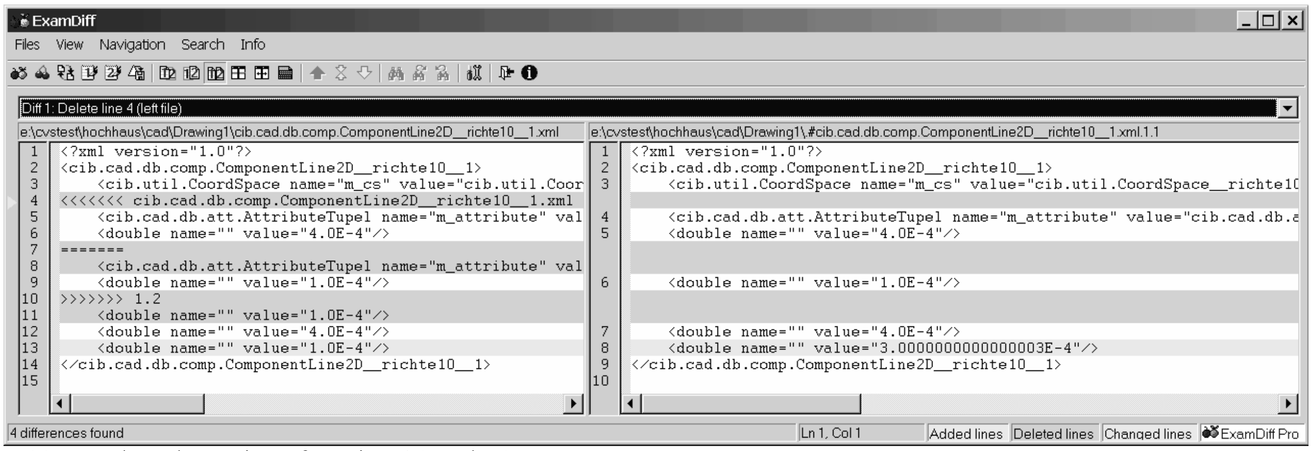
Version R2
User *B* commits the document version *B2* as version *R1* to the Repository.

Version A3
The synchronous cooperation of document versions *A2* and *R2* resulted in a conflict that has to be solved by user *A*. In the update operation the two versions have to be merged to a resulting version *A3*.

As a first implementation the authors have decided to generically merge the two object versions by the help of a simple text based *diff* tool referring to the serialized XML files. Figure 11 shows the *diff* tool in use. As was expected, the approach had some shortcomings since the semantics of private attributes are not clear to the users. In the example shown, instead of the familiar user coordinates internal world coordinates are presented to the user. The interpretation of the data shown requires a deep understanding of the underlying data structure.

The right panel in Figure 11 shows the textual representation of version *A2*: The coordinates (4, 1, 4, 3) of *L1* are shown as normalized world coordinates (scaled by a factor of 1 / 1000 in the specific case).

The left panel shows the differences between the two versions. The conflicting values of the start point's x-coordinate are shown in line 6 (original value '4' of version *A2*) and line 9 (selected value '1' of version *R2*). The conflicting values of the end point's y-coordinate are not marked: Since the value '3' has not been changed in version *A2* the merge tool automatically selects the value '1' of version *R2*.



ExamDiff
Files View Navigation Search Info
Diff 1: Delete line 4 (left file)

```
e:\cvstest\hochhaus\cad\Drawing1\cib.cad.db.comp.ComponentLine2D__richte10__1.xml
<?xml version="1.0"?>
<cib.cad.db.comp.ComponentLine2D__richte10__1>
    <cib.util.CoordSpace name="m_cs" value="cib.util.Coor
<<<<<<< cib.cad.db.comp.ComponentLine2D__richte10__1.xml
    <cib.cad.db.att.AttributeTupel name="m_attribute" val
    <double name="" value="4.0E-4"/>
=======
    <cib.cad.db.att.AttributeTupel name="m_attribute" val
    <double name="" value="1.0E-4"/>
>>>>>>> 1.2
    <double name="" value="1.0E-4"/>
    <double name="" value="4.0E-4"/>
    <double name="" value="1.0E-4"/>
</cib.cad.db.comp.ComponentLine2D__richte10__1>
```

```
e:\cvstest\hochhaus\cad\Drawing1\.#cib.cad.db.comp.ComponentLine2D__richte10__1.xml.1.1
<?xml version="1.0"?>
<cib.cad.db.comp.ComponentLine2D__richte10__1>
    <cib.util.CoordSpace name="m_cs" value="cib.util.CoordSpace__richte10

    <cib.cad.db.att.AttributeTupel name="m_attribute" value="cib.cad.db.a
    <double name="" value="4.0E-4"/>


    <double name="" value="1.0E-4"/>


    <double name="" value="4.0E-4"/>
    <double name="" value="3.0000000000000003E-4"/>
</cib.cad.db.comp.ComponentLine2D__richte10__1>
```

```
4 differences found          Ln 1, Col 1    Added lines  Deleted lines  Changed lines  ExamDiff Pro
```

Figure 11. Text based merging of version A2 and R2

In the merge procedure user *A* has decided to retain his version unchanged. Therefore, version *A3* of the document contains a single line *L1* at (4, 1, 4, 3).

Version R3
User *A* commits document version *A3* to the Repository as version *R3*.

Version B3
User *B* updates and merges versions *B2* and *R3*. No conflicts are detected because user B has not changed his document version in the meantime. The resulting version *B3* consists of line *L1* at (4, 1, 4, 3).

Version B4
User *B* removes *L1* form the database and adds the new line *L2* at (1, 3, 3, 1). The database is stored as version *B4* of the document.

Version A4
At the same time user *A* has transformed line *L1*.

```
<?xml version="1.0"?>
<java.util.HashSet—A-1
  <cib.db.cmp.ComponentLine2D
    name=""
    value="cib.cmp.ComponentLine2D-A-1"
  />
</java.util.HashSet-A-1>
```

Figure 12a. Serialized XML document with version A4 of the database.

```
<?xml version="1.0"?>
<java.util.HashSet-A-1>
  <cib.db.cmp.ComponentLine2D
    name=""
    value="cib.cmp.ComponentLine2D-B-1"
  />
</java.util.HashSet-A-1>
```

Figure 12b. Serialized XML document with version B4 of the database.

Version R4
User *B* stores version *B4* of the document as version *R4* in the Repository.

Version A5
User *A* now updates and merges. The two versions of the serialized database HashSets are shown in Figure 12. The diff tool does not detect a conflict. In the resulting version *A5* line *L1* has been automatically been removed and line *L2* has been automatically added. It should be noted that this is not the expected result from the user's view!

Conclusion
In the opinion of the authors, the example shows the general applicability of the proposed solution approach in the planning process. However, a lot of research topics remain open.

5 PERSPECTIVE AND CONCLUSIONS

With the proposed solution approach object-oriented applications of the planning process can benefit from tool-based version control systems. The potentials of this approach would exceed the possibilities of currently used DMSs. Since the versioning granularity is changed from documents to objects, subsets of objects can be flexibly composed and loaded. The explicit storage of object versions considerably simplifies the process of locating, visualizing and merging differences. Finally, available VCSs ensure a very effective storage of the object versions.

The verification example revealed that important aspects of the solution approach remain to be investigated. Future work will address the problems of visualizing the differences and merging object versions as well as the problems of selecting, loading and storing consistent subsets from the versioned model. Another research directive will be the consistent management of cross-document and cross-project relationships.

The proposed solution approach could be a step towards a true synchronous reciprocal cooperation in the planning process.

REFERENCES

Beer, D.G., Firmenich, B., Richter, T. & Beucke, K. 2004a. A Concept for CAD Systems with Persistent Versioned Data Models. *Proceedings of the Tenth International Conference (ICCCBE-X)*. Weimar: Bauhaus University.

Beer, D.G., Firmenich, B., Richter, T. & Beucke, K. 2004b. A Persistence Interface for Versioned Object Models. *Proceedings of the 5th European Conference on Product and Process Modelling in the Building and related Industries: eWork and eBusiness in Architecture, Engineering and Construction*. Leiden: Balkema

Beer, D.G. & Firmenich, B. 2003. Freigabestände von strukturierten Objektversionsmengen in Bauprojekten. *Digital proceedings of Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen (IKM)*. Weimar: Bauhaus University.

Bretschneider, D. 1998. Modellierung rechnerunterstützter, kooperativer Arbeit in der Tragwerksplanung. Düsseldorf: VDI Verlag.

cvshome 2005. https://www.cvshome.org/

Firmenich, B. 2002a. *CAD im Bauplanungsprozess: Verteilte Bearbeitung einer strukturierten Menge von Objektversionen.* Aachen: Shaker.

Firmenich, B. 2002b. Operations for the distributed synchronous cooperation of a shared versioned data model in the planning process. *Proceedings of the Ninth International Conference (ICCCBE-IX)*. Taipei: National Taiwan University

Firmenich, B. & Beucke, K. 2002. Consistency problems in a distributed CAD environment. *Proceedings of the IABSE Symposium Melbourne 2002*. Zürich: ETH Hönggerberg.

Firmenich, B. & Beucke, K. 2005. CAD in Computer Aided Civil Engineering: a Particular Approach for Research and Education (under preparation). *International Journal of IT in Architecture, Engineering and Construction*. Rotterdam: Millpress.

Fogel, K. & Bar, M. 2002. *Open Source-Projekte mit CVS*. Bonn: mitp.

javacvs 2005. http://javacvs.netbeans.org/

Rumbaugh, J., Jacobson, I. & Booch, G. 2001. *The Unified Modeling Language Reference Manual.* Boston: Addison-Wesley.

Vesperman, J. 2003. *CVS*. Beijing: O'Reilly.