A SYSTEM ARCHITECTURE FOR NET-DISTRIBUTED APPLICATIONS IN CIVIL ENGINEERING

Daniel G. Beer[1], Berthold Firmenich[2], and Karl E. Beucke[3]

ABSTRACT

The planning process in civil engineering can be characterized by three iterative phases: the phase of distribution of tasks, the phase of parallel working with cooperation among the planners and the phase of merging the results. Available planning software does only support the second phase and the exchange of data via documents.

This paper presents a software design that supports the three phases and all types of cooperation (asynchronous, parallel and reciprocal) in principle and integrates existing engineering applications. The common planning material is abstracted as a set of object versions, element versions and their relationships. Elements extend objects with application independent properties, called features. Subsets on the base of features are calculated for the execution of tasks. Therefore, new versions of elements and objects are derived and copied into the planner's private workspace. Already stored versions remain unchanged and can be referred to. Modifications base on operations that ensure consistency of the versioned model.

The system architecture is formally described. Available information technology is analyzed and used for an implementation concept. A pilot that is based on the programming language Java, a version control system and a relational database proves the implementation concept.

KEY WORDS

system architecture, net-distributed applications, civil engineering, versioning

STATE OF THE ART AND RESEARCH

The state of the art and research in the field of cooperation in the planning process in civil engineering will be presented with the help of specific scenarios. There are two planners, A and B, with their local planning material in their sandboxes and the common planning material in the repository that is available via a network. In all scenarios, data is exchanged between the sandbox and the repository. The planning material is structured as documents, databases or object sets:

[1] Research Engineer, Informatik im Bauwesen, Bauhaus-Universität Weimar, Coudraystraße 7, D-99425 Weimar, Deutschland, Phone +49 3643/58-4221, FAX +49 3643/58-4216, daniel.beer@bauing.uni-weimar.de

[2] Professor, CAD in der Bauinformatik, Bauhaus-Universität Weimar, Coudraystraße 7, D-99425 Weimar, Deutschland, Phone +49 3643 58-4230, FAX +49 3643 58-4216, berthold.firmenich@bauing.uni-weimar.de

[3] Professor, Informatik im Bauwesen, Bauhaus-Universität Weimar, Coudraystraße 7, D-99425 Weimar, Deutschland, Phone +49 3643 58-4215, FAX +49 3643 58-4216, karl.beucke@informatik.uni-weimar.de

- Persistent overwriting: Both planners open the same document over the network and start editing. Planer A stores prior to planner B. As a result the changes of planner A are overwritten by the results of planner B and are therefore lost. Reciprocal work has to be disabled, for example by the help of locks.
- Locking: *Locks* can be placed on documents within the file system or on records within a database. They prevent a reciprocal cooperation and require a sequentialization of work. Only one planner can work on a document or database record at the same time. This may lead to idle time for other planners.
- Short Transactions: With the transaction concept of available databases only *short transactions* can be executed during interactive work. On commit, results are stored in the database and are visible to all planners that use this material. An application for engineering tasks has disadvantages because the iterative planning process has to be interrupted and intermediate results are published.
- Version History: Document management systems (Sutton 1996) support a *version history*. Documents stored do not overwrite existing documents but are stored in a linear sequence. This allows for reciprocal work but an examination of variants is not possible since there is no version graph that stores the parallel development of versions.
- Document Versioning: Version control systems (VCS) from the software configuration management (SCM, Hass 2003) support the *versioning of text documents* and hence a distributed processing. Software developers can work in parallel or even reciprocally. Space is preserved by storing the differences of subsequent document versions. However, in most systems the differences between binary files are stored inefficiently. This particularly applies for objects that are serialized into binary files.

The authors propose an object version approach (Firmenich 2001, Firmenich 2002, Beucke and Beer 2005) that is based upon the idea of VCS. If a VCS is used in an object version approach some extensions have to be done:

- Creation of subsets: Object version sets cannot be predefined as it is the case with documents. Instead, different engineering domains and many changes during the planning process require a flexible description. An object extension via application independent attributes – called features – is proposed. The selection of subsets is based on a feature algebra.
- Storing object graphs: Object graphs have to be serialized into documents to be versioned with VCS. Each object is stored in a separate file. The name of the file has to be unique and persistent to be identified as a version of an existing file.
- Merging variants: The version graph is stored with the help of features since a VCS is not able to store graphs but only trees that do not contain merges.

SYSTEM ARCHITECTURE

The *system architecture* proposed for an object versioning approach in civil engineering can be divided in layers (Figure 1).

From a vertical view there is a local layer (engineering application and workspace) and a central layer (project) that are connected on demand via a network. From a horizontal view it can be distinguished between application and persistent model. The interface between them is the selection component.

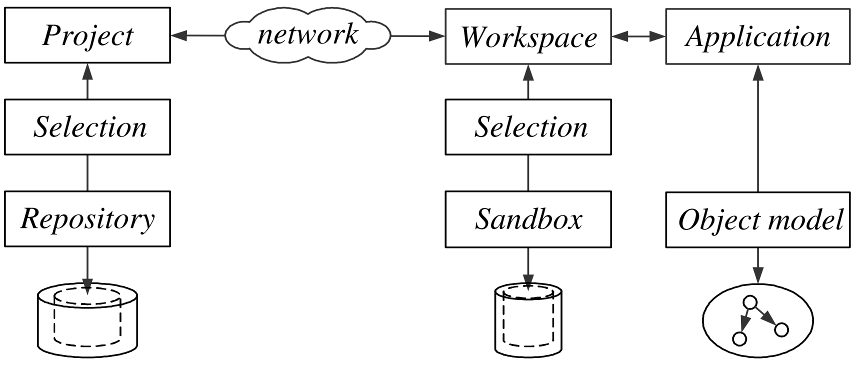


Figure 1: System Architecture (Beer 2006, Excerpt)

COMPONENTS

Engineering Application

The *engineering application* is used to solve engineering tasks. Subsets of the common planning material are processed. Intermediate results are stored locally. The application cannot process versioned models in general.

Workspace

The *workspace* enhances the model and the functionality of the engineering application for distributed processing.

An application independent element model that consists of elements with features enhances the object model. Application independent relations – called bindings – as well as the version graph are formulated with these elements. This approach enables a deferred model update.

The functionality needed to enhance the application is a generic serialization mechanism for object and element models generating a set of persistent objects and elements in the sandbox and a synchronization of the local sandbox with the remote repository (see operations below).

Project

The *project* manages the repository and communicates with the planners' workspaces to synchronize their sandboxes with the repository.

Object Model

The *object model* contains objects with attributes und references between objects.

Sandbox

The *sandbox* is the persistent representation of the object model and element model. It consists of a set of objects and elements. Additionally, bindings and version data are stored.

Repository

The *repository* is the persistent representation of all stored versions from the sandboxes of all participating planners. It additionally stores the binding graph and version graph.

OPERATIONS

Operations are applied to one specific model or serve to synchronize two models (Figure 2).

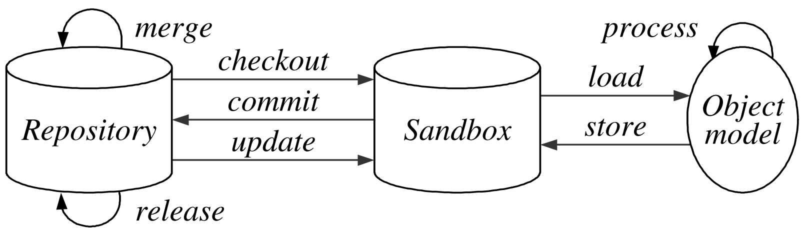


Figure 2: Operations, Beer 2006

Checkout and commit

Operation *checkout* transfers copies of selected object versions from the repository to the sandbox. Consistent subsets have to fulfill specific conditions (Beer 2006). Results are published in the repository via operation *commit*. New versions of objects and elements are created as a revision or a variant and the version graph is extended.

Load and store

Sandbox data is deserialized via *load* operation of the workspace. The result is an object model that can be serialized via *store* operation. Both operations consider features that are managed outside the application in the workspace.

Update

Long-term transactions may lead to obsolete planning material in the sandbox. Operation *update* determines obsolete objects and transfers current versions. This may need user interaction to define versions to be transferred.

Merge and release

Variants are *merged* by the homonymous operation. The feature algebra supports the user in defining the variants to be merged. Published results that can be used for further planning or that are legal states can be released via operation *release*. A valid release state has to fulfill some conditions (Firmenich and Beer 2003, Beer 2006).

FORMAL DESCRIPTION

MODELS

Object Model

The *object model* can be formally described with the following sets and relations (Table 1):

Table 1: Formal Description of the Object Model (Beer 2006, Excerpt)

Set/ Relation/ Mapping	Description
Ω	The object set contains the persistent identifiers of all objects.
α_a , $\alpha \subseteq \Omega \cup \alpha_a$	Set of atomic attribute values and all values including non atomic values (objects).
A , R	Set of attribute names and names of reference attributes whose value is an object.
$\boldsymbol{A} \subseteq \Omega \times \alpha$, $\boldsymbol{R} \subseteq \Omega \times (\alpha \setminus \alpha_a)$	Attribute relation between objects and values, reference attribute relation between objects and non atomic values.
$a : \boldsymbol{A} \rightarrow A$, $r : \boldsymbol{R} \rightarrow R$	Attribute set with the mapping between attribute relation and attribute names, respectively between reference relation and reference names.
$\mathfrak{O} := (\Omega, \boldsymbol{R}, r)$	Object model as named graph with attribute relation as edges, objects as nodes and attribute names as edge labels.

Sandbox

The *sandbox* is the persistent representation of the object model and its enhancement with features that form elements. The element model is described in Table 2:

Table 2: Formal Description of the Element Model (Beer 2006, Excerpt)

Set/ Relation/ Mapping	Description
μ	The element set contains the persistent identifiers of all elements.
$\varepsilon : \mu \rightarrow \Omega$	Assigment from elements to objects.
β	Set of feature values (atoms, collections, element identifiers).
F	Set of feature names.
$\boldsymbol{F} \subseteq \mu \times \beta$	Feature relation between elements and feature values.
$f : \boldsymbol{F} \rightarrow F$	Feature set, mapping between feature relation and feature names.
$\mathfrak{E} := (\mu, \boldsymbol{F}, f)$	Element model as named graph with feature relation as edges, elements as nodes and feature names as edge labels.
$\boldsymbol{B} \subseteq \mu \times \mu$	Binding relation as an application independent model structuring.

Repository

The *repository* stores versions of different sandboxes and their development in a version graph. The extension of the formal model of the sandbox is shown in Table 3:

Table 3: Extension of the Sandbox' Formal Model in the Repository (Beer 2006, Excerpt)

Set/ Relation/ Mapping	Description
O , M	Set of object/ element versions.
Δ , $M_\Delta = M \cup \Delta$	Set of virtual versions that are used in the version graph to mark versions as first versions or deleted versions.
$\boldsymbol{V} \subseteq M_\Delta \times M_\Delta \setminus \Delta \times \Delta$	Version graph.
$\boldsymbol{B} \subseteq M \times M$	Binding graph.
$\omega : O \to \Omega$	Mapping between object versions and objects.
$m : M \to \mu$	Mapping between element versions and elements.

Operations

Operations modify the sets and relations of the formal model described above. Beer 2006 defines them in detail.

Selection

For the *selection* of subsets an algebra of sets based on feature logic (Smolka 1992, Zeller 1997, Firmenich 2001) is proposed. An extension with operations that provide version functionality is shown in Table 4. The specific features *in, src* and *dst* are used for elements of a set and for the first/ second element of a relation. *Vn* is the transitive hull of the version relation. The language is called object version query language (OVQL, Beer 2006).

Table 4: Feature Logic and Object Version Query Language (Beer 2006, Excerpt)

Operation	Description
$\{S,T,\ldots\}$, $[S,T,\ldots]$	Union/ intersection of sets S, T, …
$f{:}S$	All elements whose feature f has a value from set S (selection).
$S.f$	All values for the feature f of elements from set S (extraction).
$prev(S) := [in{:}V,dst{:}S].src$	The first versions of all versions from S.
$anc(S) := [in{:}Vn,dst{:}S].src$	Ancestors of all versions from S.
$desc(S) := [in{:}Vn,src{:}S].dst$	Descendants of all versions from S.
$rev(S) := [in{:}V,src{:}S].dst$	Revisions of all versions from S.
$var(S) := rev(prev(S))\backslash S$	Variants of all versions from S.

IMPLEMENTATION CONCEPT

The *implementation concept* that bases upon the system architecture (Figure 1) is shown in Figure 3. The reuse of standard software for technical processing, storage and versioning is an advantageous implementation strategy.

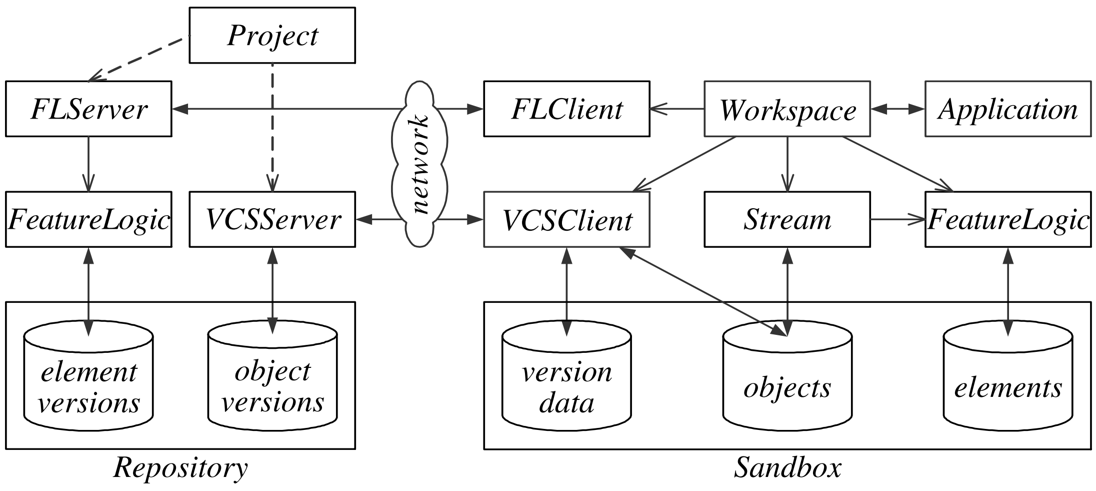


Figure 3: Implementation Concept (Beer 2006, Excerpt)

COMPONENTS

Engineering Application

All *engineering applications* that provide an object oriented programming interface are capable to be used with that approach in general. For a pilot implementation the open source engineering platform CADEMIA (www.cademia.org) is used.

Workspace

The *workspace* uses the streaming concept of the programming language Java (Sun 2003) to serialize the object and element graph into an object and element set (Firmenich et al. 2005). A VCS serves to version the object set. Unlike this approach, the element set is versioned by a data store independent FeatureLogic component. The FeatureLogic component provides a flexible and powerful mechanism to select subsets with the help of features via a language called Feature-Logic. With an available VCS, this would be possible only to a limited extent.

Thus, the workspace uses two clients to synchronize the sandbox with the repository: a version control client for the objects and a feature logic client for the elements. For the pilot implementation the concurrent versions system (CVS) and Subversion (SVN, Collins-Sussman 2004) were used.

Project

The *project* starts two servers that wait for incoming messages: a version control server and a feature logic server. Both servers are responsible for different parts of the repository: the object version model respectively the element version model.

Object Model

The Java *object model* is assumed. Therefore, a generic serializer was implemented (Firmenich et al. 2005)

Sandbox

The *sandbox* consists of three parts: the persistent object model, the persistent element model and additional version information of the VCS. Since the amount of data is relatively low, this information can be stored in the local file system. Objects and elements are stored in separate files with persistent names that are to be maintained during the whole life of this information.

Repository

The *repository* consists of two parts: the persistent versioned object model and the persistent versioned element model. The objects are stored incrementally with the VCS. The elements are stored by the FeatureLogic component in a relational database. This is appropriate since the element model consists of a small number of sets and relations.

OPERATIONS

Version Control System

Version control systems offer operations that are appropriate to be used for the approach proposed: checkout, commit and update. The tag command to mark subsets is not used since our investigations revealed that the repository structure is not suitable for a fast selection of subsets via tags.

Additional implementation

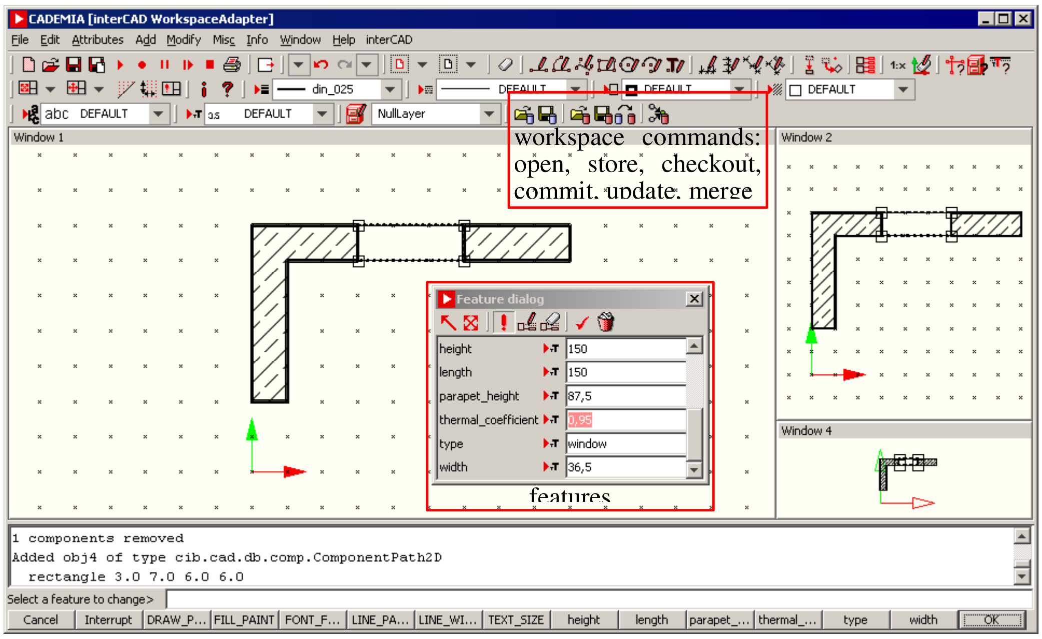
The operations on the element model have to be newly implemented. Operations that read from or write to the sandbox use a serialization concept and Java streams. The Feature-Logic structure of the persistent element model is created by the FeatureLogic component. The component provides operations for the version and binding graph.

Operations that read from or write to the repository use the same FeatureLogic component. This component is formulated independently of the data store. Different implementation types have been developed: one for a transient element model in the heap, one for a persistent element model in the file system and one for a persistent element model in a relational database.

Synchronization over the network is done by Java serialization and Java sockets. The data exchange is encapsulated by messages.

PROTOTYPE

Within a prototypical implementation of the concept the open source engineering platform CADEMIA (www.cademia.org, Figure 4) is used. This platform supports application features as well as user features. They can be used for storing version and binding information. Bindings are supported by this platform, too. The workspace functionality is implemented by separate commands that can be registered within the platform. Commands use generic workspace functionality and add system specific user interaction and visualization.



height	150
length	150
parapet_height	87,5
thermal_coefficient	0,95
type	window
width	36,5

```
1 components removed
Added obj4 of type cib.cad.db.comp.ComponentPath2D
  rectangle 3.0 7.0 6.0 6.0
```

Select a feature to change>

Figure 4: Pilot Implementation

CONCLUSIONS

The following aspects are in the focus of future research:

- Complexity: How can the complexity of this approach be handled by the engineers? Richter et al. 2006 proposes the development of flexible and powerful graphical user interfaces.
- Consistency: A transaction concept for the synchronization of object and element versions in the repository is needed to ensure model *consistency*.
- Integration of engineering applications: The concept has to be proven with different *engineering applications* using standard models like IFC (Nour et al. 2006). The IFC model can be used for the standardization of the feature names that is necessary for the integration of different applications.
- Operative modeling: Koch et al. 2006 investigates the standardization, exchange and versioning of operations that are performed to create a model.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the financial support by the German Research Foundation (Deutsche Forschungsgemeinschaft DFG) within the scope of the priority program 'Network-based Co-operative Planning Processes in Structural Engineering'.

REFERENCES

Beer, D. G., and Firmenich, B. (2003) "Freigabestände von strukturierten Objektversionsmengen in Bauprojekten". *Internationales Kolloquiums über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen (IKM)*, Weimar, Germany.

Beer, D. G. (2006*). Systementwurf für verteilte Applikationen und Modelle im Bauplanungsprozess*. PhD Diss, Civil Engineering, Bauhaus-Universität Weimar. Shaker, Aachen, Germany.

Beucke, K., and Beer, D. G. (2005) "Net Distributed Applications in Civil Engineering: Approach and Transition Concept for CAD Systems". *International Conference on Computing in Civil Engineering 2005*, ASCE, Cancun, Mexico.

Collins-Sussman, B., Fitzpatrick, B. W., and Pilato, C.M. (2004) *Version Control with Subversion*. Beijing, O'Reilly.

Firmenich, B. (2001) "CAD im Bauplanungsprozess: Verteilte Bearbeitung einer strukturierten Menge von Objektversionen", PhD thesis, Civil Engineering, Bauhaus-Universität Weimar. Shaker, Aachen, Germany.

Firmenich, B. (2002) "Operations for the distributed synchronous cooperation of a shared versioned data model in the planning process". *Computing in Civil and Building Engineering (ICCCBE-IX)*, Taipei, Taiwan.

Firmenich, B., Koch, C., Richter, T., and Beer, D. G. (2005) "Versioning structured object sets using text based Version Control Systems". *Conference on Information Technology in Construction*, CIB W78, Dresden, Germany, 105-112.

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

Koch, C., and Firmenich, B. (2006) "A Novel Diff and Merge Approach on the Basis of Operative Models". *ICCCBE 2006*. Montreal, Canada.

Nour, M., Firmenich B., and Beucke, K. (2006) "A versioned IFC Database for Multi-disciplinary Synchronous Cooperation". *ICCCBE 2006*. Montreal, Canada.

Richter, T., Firmenich, B., and Beucke, K. (2006) "Diff And Merge for Net-Distributed Applications in Civil Engineering". *ICCCBE 2006*. Montreal, Canada.

Smolka, G. (1992), "Feature Constraints Logics for Unification Grammars", The Journal of Logic Programming, New York.

SUN (2003), Java^{TM} 2 Platform, Standard Edition, v 1.5, API Specification, Copyright 2004 Sun Microsystems, Inc.

Sutton, M.J.D. (1996), *Document Management for the Enterprise: Principles, Techniques, and Applications*. Wiley, New York.

Zeller, A. (1997), *Configuration Management with Version Sets*, PhD thesis, Fachbereich Mathematik und Informatik der Technischen Universität Braunschweig, Germany.