

#9014: NET-DISTRIBUTED APPLICATIONS IN CIVIL ENGINEERING: APPROACH AND TRANSITION CONCEPT FOR CAD SYSTEMS

Karl Beucke¹, and Daniel G. Beer²

ABSTRACT

CAD applications have been developed for civil engineering with the intent to support the design process of engineers. So far, general acceptance of these solutions is limited to the support of technical documents. Requirements for extending the scope of CAD solutions in civil engineering are discussed and a solution is proposed for ensuring consistency of an engineering information model. Furthermore, virtually all solutions available today were originally designed as stand-alone solutions. A versioning approach for object models is proposed for supporting net-distributed applications in civil engineering. Finally, a system architecture is developed for engineering applications that is intended to support the workflow in civil engineering projects based upon a versioning concept for engineering object models. One important aspect of the system architecture proposed is a transition concept from current solutions to newly designed solutions. There exists a wealth of knowledge and data in engineering companies based upon commercially available CAD systems. Therefore, the authors believe it to be important to build upon this knowledge and information in order to show a path of growth from existing solutions to a next generation of CAD solutions in civil engineering.

KEY WORDS

Civil engineering, net-distribution, CAD, versioning

CONCEPT FOR CAD APPLICATIONS IN CIVIL ENGINEERING ENVIRONMENTS

Traditionally, technical drawings are used in civil engineering in order to *document* the geometry, the construction and the visual representation of structures. Technical drawings can either be 2-dimensional projections that are drawn to scale or schematic drawings. Both have their use and justification and form the basis of communication between civil engineers cooperating in a project.

CAD concepts were originally introduced into civil engineering applications with the intent to support the *design process* of structures independent of the limitations that are

¹ Professor, Informatik im Bauwesen, Coudraystraße 7, Bauhaus University Weimar, 99423 Weimar, Germany, Phone +49 3643/58-4215, FAX 3643/58-4216, karl.beucke@informatik.uni-weimar.de

² Research Assistant, Informatik im Bauwesen, Coudraystraße 7, Bauhaus University Weimar, 99423 Weimar, Germany, Phone +49 3643/58-4221, FAX 3643/58-4216, daniel.beer@bauing.uni-weimar.de

inherent with 2-dimensional technical drawings. In practical reality today, this has by far not developed to a degree that was originally intended and expected. By far most CAD applications used in civil engineering world wide are still being used for producing and handling technical documents. This is called the document-oriented approach versus the model-oriented approach.

In a contribution to IKM 2000 (Pahl and Beucke 2000) it was postulated that an effective use of digital models in civil engineering applications requires some essential prerequisites for a successful introduction of such models into the civil engineering workflow. Vital prerequisites were defined and solutions proposed. Modern CAD systems are based upon object technology. In civil engineering applications these objects are the information objects of engineering structures with their specific semantic meanings. One essential aspect of the corresponding object model is formed by a multitude of relations between objects. These relations need to be observed in order to ensure consistency of the engineering model. Relations between objects must thus be described and handled and modifications to a structured set of objects must be treated in a consistent manner. For this purpose, the concept of a *binding relation* was introduced in (Pahl and Beucke 2000) in order to ensure consistency of an object model.

Current CAD applications in civil engineering mostly allow for modifying the object basis of a model without strictly observing dependencies that consist with respect to other objects, e.g. an object is deleted even though another object still contains a reference to that object thus resulting in an inconsistent state of the object model.

NET-ENABLED VS. DESIGNED FOR DISTRIBUTED COOPERATION

Most application software for civil engineering purposes that is commercially available today was originally designed as a stand-alone application. It was designed at a time when neither the necessary requirements for computer and network technology nor for physically available wide-area network connections were fulfilled.

A typical example for this situation is the installed basis of most CAD-systems that are being used in engineering offices and construction companies around the world today. Virtually all of these are designed according to the principles of a single-user environment. A single user is supported in generating drawings and even models but in order for another user to get access to the information generated, it is necessary to either export the information required to that other user or to close the application and allow another user to access the information.

With the availability of an affordable and ubiquitous network environment and with the enormous increase in storage capacity and processing power, the network developed into a vital part of engineering work. Every engineer today utilizes network functionality for communication purposes, however comparatively few engineers use it effectively for cooperation purposes. The concepts and software needed for these purposes are simply not in a state that is widely acceptable to the civil engineering user community. Initial ideas and concepts for the utilization of central product models as a basis for net-distributed cooperation and collaboration have not yet proven to be widely acceptable by civil engineering users. The authors believe that an important reason for this fact is the nature of the engineering design process. Such a design process is regarded as a *long transaction* with different engineering

teams working on developing and refining solutions independent of other related engineering teams in the project, thus accepting temporarily inconsistent states in the complete information model of the project. Such temporarily inconsistent, locally independent states need to be resolved at defined milestones in the project by merging separate information sets.

Of course, commercial software packages today are usable in a network environment and they are enhanced with many features for supporting cooperation via the network. In CAD, for instance, special formats were defined for effective use via the world-wide-web, special viewers are available, various exchange formats were developed and workflow and redlining technologies were added. All of these features, however, were added subsequently to software that was never designed for purposes of net-distributed cooperation. Such software solutions are often referred to as *net-enabled* today. Net-enabled software solutions in civil engineering are an important step forward in utilizing modern networks, however, they will always be limited to the restrictions of the original design of the application. An application that will truly support the civil engineering workflow while exploiting the full potential of the network needs to be defined in accordance with the needs of the users and a system architecture must be developed that reflects the workflow of civil engineering problem solving. This task cannot be expected to be solved by computer science. It is primarily a civil engineering task.

A VERSIONING CONCEPT FOR CIVIL ENGINEERING APPLICATIONS

Based upon these preconditions and needs a versioning concept was developed by Firmenich (Firmenich 2002a) that addresses the needs and requirements of object models in civil engineering applications and of the engineering workflow in civil engineering projects.

The engineering workflow can be abstracted into three general phases. In a first phase, one or more engineers select a subset from a shared information basis called the *project*. This is loaded into a private, local information basis called the *workspace*. In the second phase, the engineers work independently from the *project* in their local *workspace* with a specific engineering application and operate on the information that was loaded. In the third phase, an engineer can discard the results or store them back to the *project*. Because of the long duration of the second phase (days, even weeks) it is required that intermediate results can be stored in the project even though temporarily accepted inconsistencies have not been resolved yet, i.e. without merging the information sets. This is necessary in order to minimize the risk of losing information. The complete process can be repeated recursively (Figure 1).

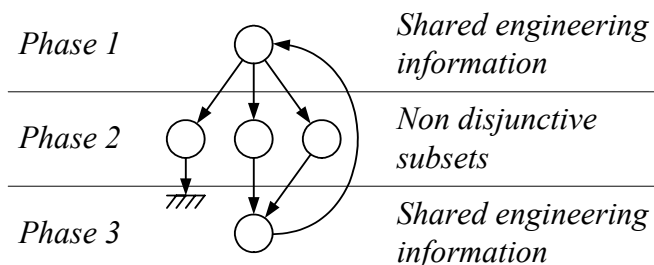


Figure 1: Phases of the Engineering Workflow

The general solution concept is based upon set and graph theory. All relevant states of all objects of the engineering process – called object versions – are maintained in an object version set M . The object version set is continuously expanded. Once an object version is entered into the set it will never be changed or even deleted. In case modifications to an object will be required, a *new* object version of the object is created while still preserving the *old* object version. Thus, any reference to any object version will still be valid and the object model will never assume an inconsistent state (Figure 2).

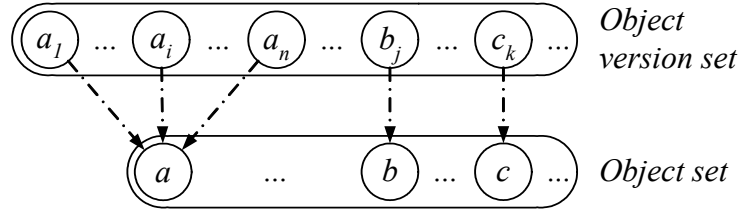


Figure 2: Objects and Object Versions

Relations between different object versions are handled via a version relation $V \subseteq M \times M$. A version history may include branches ($a_i \rightarrow a_k, a_l$), merges ($a_k, a_l \rightarrow a_n$), variants (a_k, a_l), revisions ($a_n \rightarrow a_o$) or deletions (δ_2). The relation V contains pairs of object versions and succeeding object versions whereby a next object version is a modification of a preceding one. The predecessor of a first object version and the successor of a last (deleted) object version is called *virtual object version* as shown in Figure 3 for δ_1 and δ_2 . This approach was adopted in order to avoid special consideration of these cases in relation V .

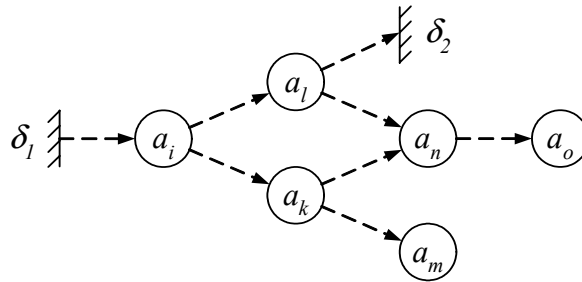


Figure 3: Version Relation

Firmenich proposed in (Firmenich 2002a, Firmenich 2002b) the use of a binding relation $B \subseteq M \times M$ that includes pairs of object versions. The purpose is to preserve consistency of the object model. The first element is the binding object version, the second element is the bound object version. This is an n to m relation. No object version can be bound to more than one object version of another object (Figure 4: b_m can either be bound to a_k or a_l). Otherwise, the state of the bound object version would be dependent upon several, possibly conflicting versions and may be inconsistent. The problem of consistent updates and handling of cycles in models caused by modifications to objects was addressed by Hanff (Hanff 2003). Operations for the distributed and consistent processing of the object version set were defined and described by Firmenich in (Firmenich 2002b).

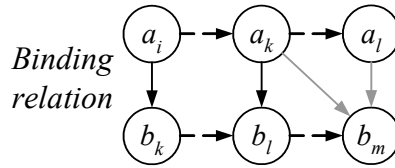


Figure 4: Binding Relation

Finally, the engineering project workflow requires specific release states of the current state of a project to be released for use by other participants in the project at specific milestones. A solution for the definition of release states based upon the mathematical background described above was proposed by Beer and Firmenich in (Beer and Firmenich 2003). A release state includes object versions – at maximum one per object – and the bindings between them (Figure 5a). It is required that all binding object versions are included to determine the state of every object version in the release state (Figure 5b). Furthermore, an engineer should be informed if obsolete object versions – bound to *old* object versions where newer versions exist – are to be added to the release state. Therefore, it is not trivial to build a valid release state that is in accordance with the requirements formulated above.

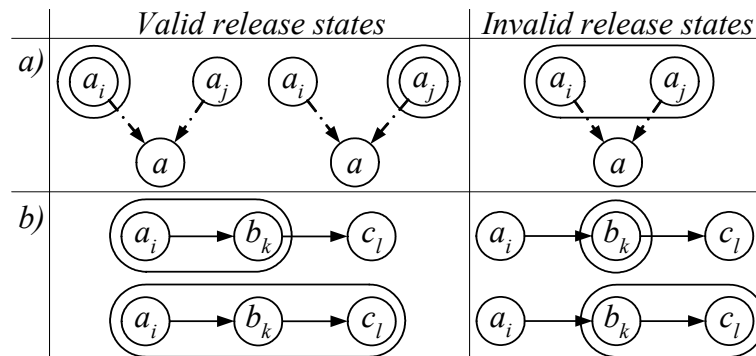


Figure 5: Release State

REQUIREMENTS FOR THE IMPLEMENTATION OF THE CONCEPT

APPLICATION VIEW

The concept consists of three different application types: the project, the workspace and the engineering application to be utilized by the user (Beer et al. 2004b, Figure 6).

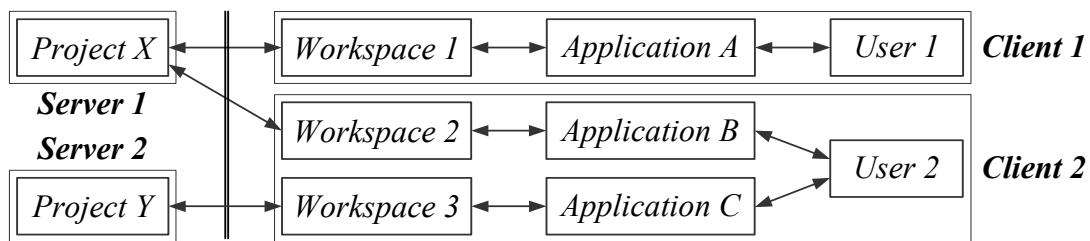


Figure 6: Application View of the Implementation Concept

Project

The shared project is a server that waits for messages from workspaces to load or store engineering information of a specific project. The *project* manages the information of all project participants and thus communicates with *several workspaces*.

Workspace

The workspace is a “wrapper” for an application to provide functionality for distributed co-operation. It communicates with *exactly one project* that is accessible via a network (e.g. Internet) and *exactly one application*. Object identifiers are unique within a project respectively within an application. Thus, the processing of different projects or the use of several applications at the same time with a single workspace is not supported.

Engineering application

The application is an engineering software system – e.g. CAD – used in an engineering project by a project participant. It is important to the authors that existing engineering applications may still be usable in the new environment. A transition concept from the utilization of existing solutions to specifically designed solutions is of vital importance for acceptance by the engineering users. Therefore, the software system may be a newly designed solution or a currently available software solution which is subject to specific requirements and restrictions. The application is used within *exactly one workspace*. Thus, a corresponding number of workspaces – specific for every application type – is used to work with more than one application or project simultaneously.

MODEL VIEW

Every application mentioned above has its own model(s): the persistent model of the project, the transient and persistent models of the workspace and the application. Transient models are stored in the memory of the computer, persistent models are stored on permanent storage devices as files or databases (Beer et al. 2004a, Figure 7).

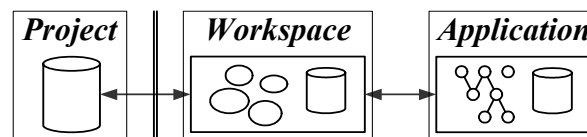


Figure 7: Model View of the Implementation Concept

Project model

Every project manages *exactly one persistent model* that stores the complete information of a project including all relevant states stored as versions of objects. The model is designed for a flexible access with the help of a query language called feature logic (Firmenich 2002a). Feature logic is a set algebra to calculate subsets based on object attributes (features).

Every object version stored has a unique Persistent Object Identifier (POID) that cannot be changed anymore once it is created. All POIDs are stored in the domain set D_P . The attribute names (features) are stored in set F_P , the primitive values (strings and numbers) are

stored in set V_P . The binary relation $A_P \subseteq D_P \times V_P$ assigns Persistent Value Identifiers (PVID) to all values. The PVIDs are elements of domain set D_P . Objects are decomposed into (POID, feature, PVID)-tuples that are stored in relation $R_P \subseteq D_P \times F_P \times D_P$ (Figure 8).

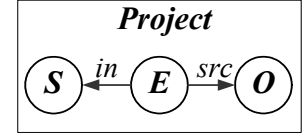
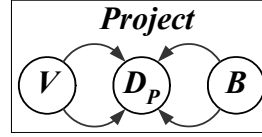
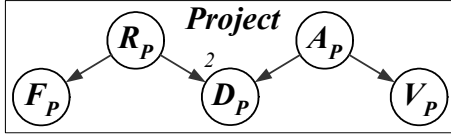


Figure 8: Persistent Project Model Figure 9: Versioning Model Figure 10: Set Model

The version relation V and the binding relation B consist of (POID, POID)-pairs (Figure 9). Set elements are represented by specific objects $e \in E$. The attribute “in” stores the Persistent Set Identifier (PSID) of set S , “src” (source) stores the POID (Figure 10). Binary relation elements (pairs) are specific objects with three attributes: “in”, “src” and “dst” (destination).

Workspace models

The workspace has to ensure that the version relation can be created from the workspace model. Thus, the workspace model consists of four disjunctive sets that include the POIDs of unmodified (U_W), modified (M_W), newly created (N_W) and deleted (D_W) object versions. Set $W = U_W \cup M_W \cup N_W \cup D_W$ of workspace objects is the union of the four sets. A binary relation $R_W = \{(POID, OH) \in W \times A\}$ connects the object handle (OH) of every application object with the POID of the corresponding parent object version of the project. A Temporary Object Identifier (TOID) that is distinguishable from any POID is created by the workspace for newly created objects (Figure 11).

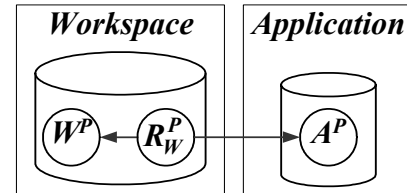
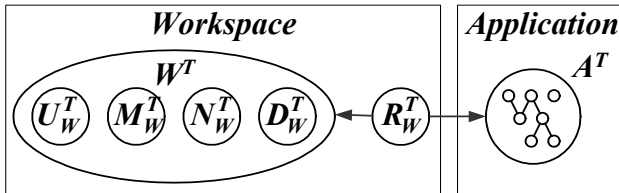


Figure 11: Transient Workspace Model Figure 12: Persistent Workspace Model

The persistent model stores the same information as the transient model. Only the relation R_W has to change the second component of its elements from Transient Object Handle (TOH) to Persistent Object Handle (POH) created by the application at store time (Figure 12).

Application models

The transient and persistent model is application specific. It is assumed that the transient model is object oriented. All objects are elements of object set O_A . The attribute names of all objects are elements of attribute set A_A . The primitive values (strings or numbers) and the TOHs of non-primitive values are stored in value set V_A (Figure 13).

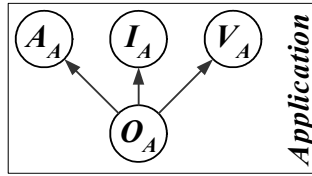


Figure 13: Transient Application Model

Every class of the application model has to implement the `Storable` interface in order to gain access to the attributes. This is acceptable for new classes. For existing, unchangeable classes a `storer` – that implements the `Storable` interface – has to be provided. A `storer` can handle several classes and create class instances by class name (Figure 14).

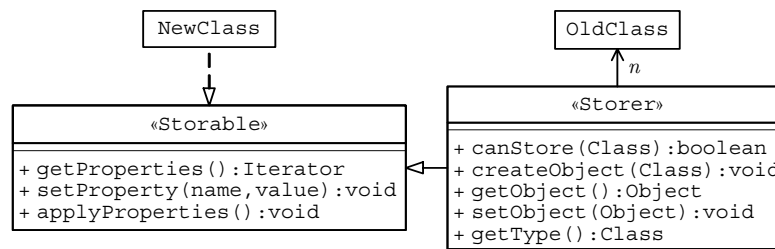


Figure 14: Attribute Access

The persistent model may be stored in a database, a file etc. This aspect is not a part of the concept. The application performs the transformations between the transient and persistent model.

A TYPICAL WORKFLOW

The model transformations can be visualized using a typical workflow approach.

Step 1: Loading from the project

First, a user starts loading data from the project into a workspace. He formulates a query in feature logic language. The result are (POID, object)-pairs that are stored in relation R_W . The object creation is described in (Beer et al. 2004a). POIDs are stored in set U_W because objects are unmodified yet. Objects are stored in the transient model A of an application (Figure 15).

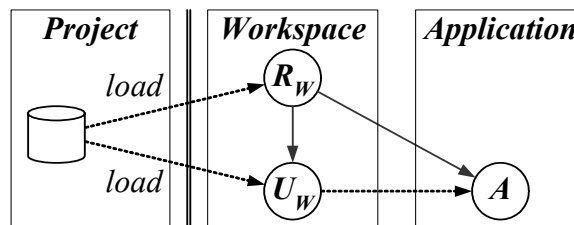


Figure 15: Loading from the Project

Step 2: Local modifications

After loading an object set, operations are performed on the objects (create, modify or delete). The operations are performed with the functionality of an engineering application. The workspace observes these changes and places the POIDs of the changed objects into corresponding sets (Figure 16):

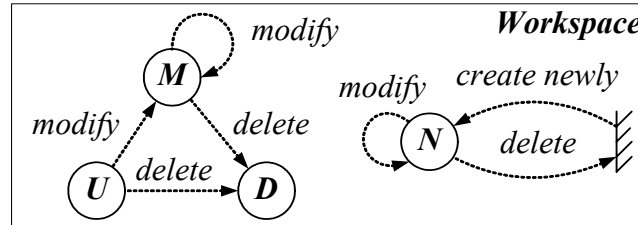


Figure 16: Local Modifications

- Newly created objects are stored in set N . They stay in set N in case of modification because there is no parent version stored in the project. On deletion the object is removed from set N .
- Object versions loaded from a project are unmodified at this time and stored in set U . After a modification (deletion) they are modified (deleted) and moved to set M (D).
- Modified object versions are stored in set M . They are moved to set D if they are deleted. In case of modifications the keep the status “modified”.
- Deleted versions are stored in set D . They cannot be modified further.

Step 3: Local storing and loading

Not every intermediate state has to be stored to the project. However, if the engineer wants to interrupt his work, intermediate results have to be stored locally without losing version information.

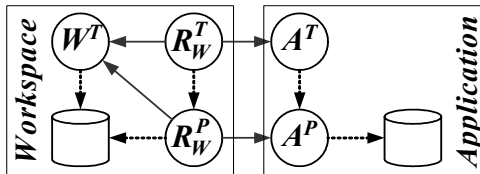


Figure 17: Local Storing

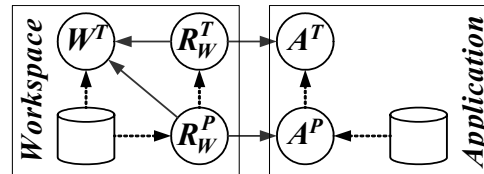


Figure 18: Local Loading

The transient model A^T of the application is transformed to the persistent model A^P of the application and new POHs are assigned to the objects at storage time. The storage depends on the application type (file or database). The transient object handles (TOH) used within relation R^T are replaced by the newly created persistent object handles (POH). This leads to the persistent relation R^P . The workspace set W with its persistent object identifiers (POID) need not be changed. Set W and relation R^P are stored into a workspace file (Figure 17).

For continuation of work on the workspace, information stored locally is loaded back from the persistent application model A^P and the persistent workspace model W^P . Both are transformed to the corresponding transient models. The application transforms the application models and assigns new TOHs to the objects. The workspace transforms the workspace models and replaces the POHs used within relation R^P with the newly created TOHs (Figure 18).

Step 4: Storing workspace information back into the project

After completion of local modifications corresponding objects need to be stored as new versions into the project. Unmodified objects (set U_W) are discarded. For all other objects the project creates new persistent object identifiers (POID) and stores it in the domain set D_P . Object attributes are stored in feature set F_P , atomic values are stored into value set V_P . Non-atomic values are objects again that are decomposed as described.

The transient object handles (TOH) are replaced by the old POIDs, the information is retrieved from the workspace relation R_W . The relation between the old and new POIDs is stored to version relation V : The predecessors of newly created object versions from set N_W as well as the successors of deleted object versions from set D_W are virtual object versions as shown in Figure 19.

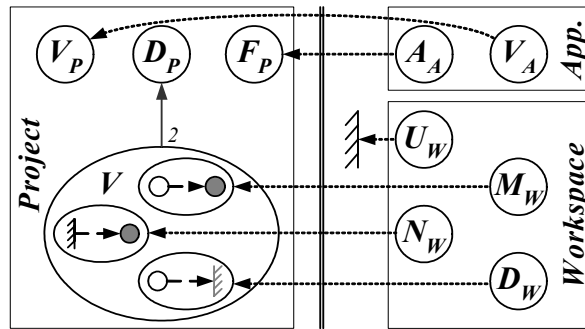


Figure 19: Storing back into the Project

TRANSITION CONCEPT FOR EXISTING APPLICATIONS

Existing applications cannot simply be replaced by new concepts. In CAD for instance there exists a tremendous value in knowledge and data in the civil engineering industry and in addition there is a great depth of functionality implemented in existing CAD applications. Any new concept cannot be introduced independently and isolated from these considerations. A transition concept will be required which will have to define how existing applications can still be used in the new environment and how these applications can benefit from the new concepts. Eventually, these will have to be phased out when new applications will have grown to a level that will at least be comparable to that of currently existing applications still restricted to single user concepts.

The main requirement for existing applications to be usable in the new environment is that these need to be implemented according to object-oriented principles and that an object-oriented Application Programming Interface (API) is available. On the basis of these re-

quirements additional functionality may be implemented that will not only allow these application to be usable in the new environment but even let them benefit from the new concepts. Basically, the standard *load* and *store* functionality of the applications will have to be replaced by new functionality taking into account the new concepts. Furthermore, modifications to the application model need to be monitored and handled.

The workspace functionality is application independent and can be provided by a workspace adapter class. The connection to the application (observation of changes in the application model and access to the model) is application dependent and must be provided by a specific workspace implementation that inherits from the workspace adapter. The cooperation functionality of the workspace can be implemented in new applications or existing applications that may be extended accordingly.

Most commercially successful CAD systems will support these requirements and can thus be extended with very limited efforts to work in the new environment.

OPEN PROBLEMS AND FURTHER PERSPECTIVES

MINIMAL SPACE NEEDS VS. MAXIMAL ACCESSIBILITY

The `Storable` implementation decides, what attributes of an object have to be made persistent. Attributes can be renamed to fulfill existing naming conventions, can be stored with more than one name (e.g. `width = depth = thickness = z`) and new attributes can be created (e.g. calculated from existing one) in order to increase the flexibility of the selection of subsets of the project model at load time (Step 1).

Minimal persistence means that an object version must be initialized from the data stored in the project. Maximal accessibility means that the user has sufficient functionalities to select subsets with feature logic based upon attribute names (features) stored. This increases the storage requirements.

An open question is the best compromise between minimal storage requirements and maximal accessibility. Therefore, engineering workflow scenarios have to be investigated and analyzed.

SHARED DATA MODEL

Attributes to be stored in the project are made available by the `Storable` functionality. This leads to the persistent project model. This model must be a common denominator of understanding of different applications, if the model should be processed with different applications. Therefore, a shared data model and a flexible type concept are required and will be developed. The shared data model is not in the focus of this paper, it can be a standard data model or a proprietary data model that is used within a specific project.

CLASS EVOLUTION

New application releases may change the classes of the application model. Thus, the new class definition may not match with the attributes stored in the project. The `setProperty()` method respectively the `applyProperties()` method of the `Storable` implementation has to handle this flexibility. The corresponding implementation will become very complex.

FURTHER PERSPECTIVES

The result (Beer 2005) will be an interface definition for distributed applications and models used by civil engineers that has to be implemented by future applications. The investigation of usable software patterns, the complexity of the applications and the storage requirements of the models as well as a pilot implementation will serve to verify the general concept.

CONCLUSIONS

This contribution presents a concept for net-distributed applications in civil engineering that is based upon a versioning concept that is mathematically formulated. The result of the research work is an interface definition for distributed applications and models that can either be implemented within existing applications or in totally new designed application solutions. A CAD pilot implementation is provided as a proof of concept.

ACKNOWLEDGMENTS

This research is funded by the German Research Foundation (DFG) within the scope of the priority program ‘Network-based Co-operative Planning Processes in Structural Engineering’ (SPP 1103). The theoretical background of the versioning concept is part of the dissertation of our colleague assistant professor Berthold Firmenich from Bauhaus University Weimar.

REFERENCES

- Beer, D. G., Firmenich, B., Richter, T., and Beucke, K. (2004a). “A Persistence Interface for Versioned Object Models” *Proceedings ECPPM*, Leiden, Balkema, p. 49-57.
- Beer, D. G., Firmenich, B., Richter, T., and Beucke, K. (2004b). “A Concept for CAD Systems with Persistent Versioned Data Models” *Digital proceedings ICCCB*, Weimar, Bauhaus University Weimar.
- Beer, D. G., and Firmenich, B. (2003). “Freigabestände von strukturierten Objektversionismengen” *Digital proceedings IKM*, Weimar, Bauhaus University Weimar.
- Beer, D. G. (2005). *Systementwurf für die verteilte Bearbeitung von versionierten Objektmodellen*. Ph.D Diss, Bauhaus University Weimar, indented publication.
- Firmenich, B. (2002a) *CAD im Bauplanungsprozess: Verteilte Bearbeitung einer strukturierten Menge von Objektversionen*. Ph.D Diss, Bauhaus University Weimar, Aachen, Shaker.
- Firmenich, B. (2002b) “Operations for the distributed synchronous cooperation of a shared versioned data model in the planning process” *Proceedings ICCCB*, Taipei, Taiwan, p. 1081-1086.
- Hanff, J. (2003) “Abhängigkeiten zwischen Objekten in ingenieurwissenschaftlichen Anwendungen.” *Digital proceedings IKM*, Weimar, Bauhaus University Weimar.
- Pahl, P. J., and Beucke, K. (2000) “Neuere Konzepte des CAD im Bauwesen, Stand und Entwicklungen” *Digital proceedings IKM*, Weimar, Bauhaus University Weimar.