ADAPTABLE VIEWS SUPPORTING LONG TRANSACTIONS IN CONCURRENT ENGINEERING

Raimar J. Scherer[1] **, Matthias Weise** [2] **and Peter Katranuschkov** [3]

ABSTRACT

Product data models such as IFC and CIS/2 integrate different design domains thereby enabling coordination of building design. However, the involved discipline practitioners, such as architects and civil engineers, usually need only a subset of the product model information to fulfill their tasks. In this paper we present an approach for *adaptable views* enabling definition and use of such individual model subsets. Technically, a model subset is specified by using our *Generalized Model Subset Definition* schema (GMSD) which can be mapped to a data query language. It provides a technical basis to combine implementation agreements with the actual use of model subsets in the design process. Using GMSD generic views can be predefined which can then be dynamically adapted to specific needs with only a few user interactions. We show how by means of adaptable views effective use of *long transactions* and *concurrent work* can be supported. The reported research was performed in the frames of the German DFG priority program SPP 1103.

KEY WORDS

Product Data Management, Long Transactions, Model Views, IFC.

INTRODUCTION

In building design multiple discipline specialists (architects, structural engineers, building services engineers etc.) work concurrently on the same project, the building that has to be designed. Each of them has a different view on the planned building and the problems to be solved. Communication between them and their design applications often fails because of a slightly different understanding of the same terms and concepts. This problem, known as a semantic interoperability problem, is tackled by the use of standardized product data models such as IFC (Liebich et al. 2006) or the STEP family of standards (ISO 10303) trying to harmonize and integrate the various discipline views into a single common data schema. With the help of standardized product data models and a shared database it is a lot easier to manage design data and to control consistency.

Working with a shared database is widely accepted in the IT world and the concept of using views is well known. In database theory a *view* is generally defined as a result set of a data query containing a subset of the database. Moreover, data can be shown in different ways, thereby hiding the complexity of the underlying database or creating a more suitable presentation structure. However, pure database solutions are not applicable for product data

[1] Professor, [2] Research Assistant, [3] Senior Research Assistant, Institute for Construction Informatics, Technical University Dresden, D-01062 Dresden, Germany, Phone +49/351-46332966, FAX +49/351-46333975, email: {raimar.j.scherer, matthias.weise, peter.katranuschkov}@cib.bau.tu-dresden.de

management in the AEC domain for a number of reasons (Amor & Faraj 2001). Two of them which are most relevant are as follows:

- Database views provide no answer how to deal with parallel work using long transactions, i.e. to handle concurrent modifications without data locks.
- Transactions, i.e. design activities, and herein needed design data are not or only little formalized in the AEC world. Rather, they will be done ad hoc with no clear awareness about needed design data and planned changes.

In this paper we address a simplified scenario of a general *view approach* as shown in figure 1. In this scenario data mappings remain at design applications that are able to import and export product model data and consequently are black boxes for the database. The scenario supports the use of long transactions which subdivides each design step in a sequence of (1) *check-out* of the specifically needed design data into a private workspace, (2) *making design changes* within the private workspace, whereas the same original data remain public and can thereby be read and changed by others in the collaborative design group, and (3) *check-in* - after days or even weeks - of the changed sub-model into the shared workspace to make the changes visible for the other designers and update the common model state.

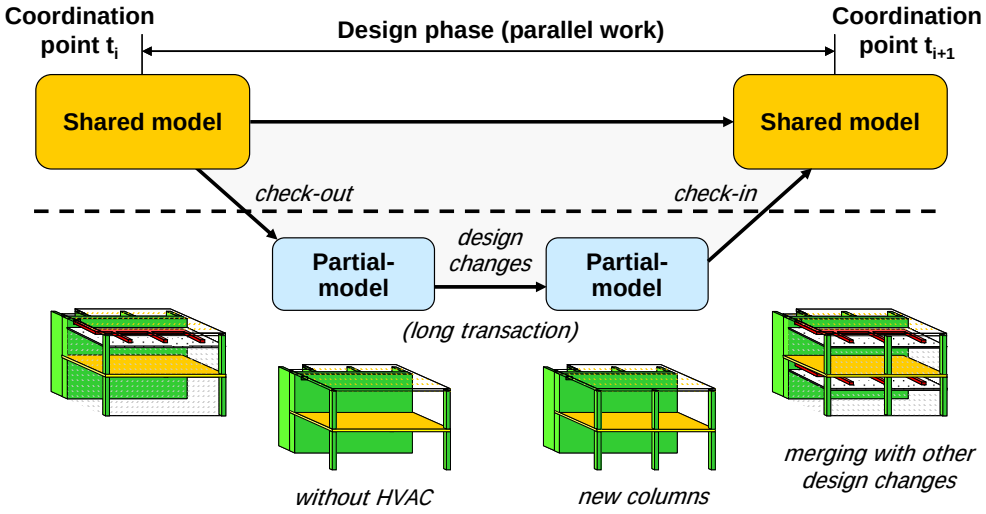

Figure 1: Cooperation model for using partial model data on the basis of a standardized product data model and long transactions

The shown scenario is state of the art in product data sharing, especially for cross-domain product models addressing a comprehensive range of multidiscipline design information. However, in AEC we miss an efficient procedure how to formalize and apply views. In the following discussion a view means to filter out all data of interest from a model but does not include any data mappings. The focus of the research is on EXPRESS-based AEC product data models, and more specifically on IFC. For terms related to product modeling the definitions from Björk (1995) are used, according to which a *product model* describes a particular building and a *product data model* describes the underlying conceptual schema.

BACKGROUND AND RELATED RESEARCH

When using cross-domain product models such as IFC we have to focus on several different aspects to discuss view approaches. Conceptually, there are two levels for defining a model subset: (1) class level, restricting product model data to a *generic view* that is interesting for a certain type of work tasks, e.g. any tasks related to space layout, and (2) instance level, restricting a generic view to product model data needed for a specific work task, e.g. for the space layout of the ground floor in building section A. These two complementary levels are shown schematically in figure 2. They are addressed either by view definition approaches or by data query languages. Using only one of these aspects has some drawbacks with regard to the application in design practice. This will be outlined in a later section where definition and maintenance of model subsets as well as their use for specific work tasks will be discussed.

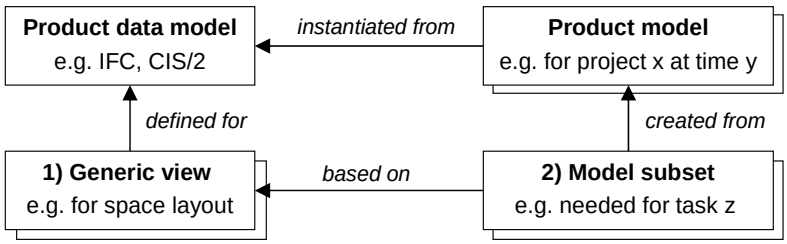

Figure 2: Two levels of view definition and their relationship to product (data) models

DEFINITION AND MAINTENANCE OF GENERIC VIEWS

Generic views and product data models have a lot of similarities. The creation of product data models normally starts with a requirement analysis of all processes which have to be supported. In the STEP family as well as in the IFC modelling guidelines a more or less obligatory procedure is described to create an EXPRESS-based schema definition that fulfils all requirements for its use (Fowler 1995). However, product modelling is focused on finding a good solution for the product data model as a whole and not on defining generic views. The latter is in fact not really possible in early model development phases where no data schema is yet available. Due to that, in earlier modelling efforts requirement analysis has been rarely used to define relevant data exchange scenarios. This has changed in the last years since product data models have reached a level of complexity and an amount of data which are not any more possible to be implemented and supported by specialized design application.

The roots of view definitions can be traced back to the object-oriented modeling approach which introduced the idea of a modular and reusable model design. In more complex product data models such as IFC there can be found a layer structure, normally based on inheritance, and partitioning into a set of schemas. However, this fixed modularization has not been appropriately adapted for working with partial model data. Instead, for the implementation of the IFC model additional view definitions are introduced to address work tasks, e.g. coordination of design activities using the so called *coordination view* (Steinmann 2006). Such view definitions are the basis for certifying IFC applications that all quality criteria are fulfilled. Currently they are implemented manually in a long tiresome process and have to be human readable and maintainable to allow future refinements.

The ProIT project investigates many application scenarios providing the basis for IFC view definitions (ProIT 2004), and the IDM project is developing a principal methodology for the creation of partial model views and their binding to process through so called functional parts (Wix 2005). The current definition of IFC views follows a pragmatic approach using available software tools such as Microsoft Excel, Word or Visio to capture implementation agreements. In such agreements the use of objects and their attributes is defined by choosing a predefined type, such as *not supported*, *read*, *write* or *round trip*, and by giving additional annotations. So far only the *coordination view* is fully available for implementation and used to certify applications. Other view definitions as e.g. for HVAC design or structural engineering are in work, with deployment announced for the near future. These efforts show that the concept of views is already being accepted in practice, but it is still little formalized and because of yet missing tool support very time consuming. This can be changed by switching to a more structured way of defining views, as suggested in the BLIS project (Hietanen 2002). In the BLIS approach the idea of concepts is introduced allowing to start with high level classes, such as *wall*, *beam* and other things in the mind of engineers, which are then detailed step by step to reach low level classes, such as *point, location* and other entities needed to complete the high level classes. Accordingly, a view will be defined in two ways: (1) generally, by using an engineering ontology representing the content of the targeted product data model, and (2) specifically, by mapping the general view to an existing release of the product data model, e.g. IFC2x2. For view definition a set of diagrams and form sheets are used to improve comprehensibility and maintenance. However, similar to the approaches from the IAI these definitions are not directly applicable as data queries and thus need to be implemented and maintained by hand.

ADAPTATION OF A GENERIC VIEW TO THE SPECIFIC NEEDS OF A WORK TASK

As outlined above generic views are defined on class level. To apply them in design activities it is necessary to additionally concretize and differentiate them on instance level, e.g. to select a specific floor for space layout. This further differentiation would not only allow to focus on the really needed design information but also reduces the amount of exchanged data and helps to manage parallel transactions. Consequently, an implementation of a generic view should be adaptable to specific needs and thus requires additional user interactions. Furthermore, to be used in a database environment, an application programming interface (API) is needed to support creation of model subsets. This can be done in different ways:

- Using a low level API to access and manipulate individual data instances and attributes. Related to EXPRESS-based product data models, the STEP Standard Data Access Interface (SDAI) falls into this group. It is rather applicable for local use or only for very small model subsets as all needed instances have to be requested in numerous small steps.
- Using a descriptive data query language enabling to specify a model subset in a single or only a few requests. In that respect the EXPRESS-X specification (Denno 1999) and the *Partial Model Query Language* (PMQL) proposed by Adachi (2002) have to be mentioned. Like SQL for relational databases, they create a result set containing the requested product model data. Their limitations are also similar to the limitations of SQL.

- Using a high level API providing a function for each view thereby being adaptable for specific needs by modifying the parameters of these functions. A mentionable effort to standardize such functions is currently coming from the BLIS-SABLE initiative, trying to harmonize data access interfaces which have to be provided from product model servers.

From these observations we can conclude that to apply generic views in design practice means (1) to add the feature of adaptability, and (2) to provide them as API or data query. However, generic views are related to human understanding and do not directly deal with adaptability. For them, an automatically generated data query is neither possible nor necessary. A data query is a technical specification, quite similar to a programming language. It is related to the concepts of the underlying database and hence more difficult to understand and maintain. In the following we present an approach how this difficulty can be overcome.

THE ADAPTABLE VIEWS APPROACH

The suggested new adaptable views approach combines the definition of generic views with its use for data queries. It allows to shorten the time needed to define and implement views. Consequently, it is easier to customize model subsets, i.e. to identify precisely what are the really needed design data. This is important to improve the quality and richness of the managed data and to support concurrent work. In this section we first outline the overall concept and then present the developed tool support. The next section discusses the benefits of the approach for the management of long transactions.

OVERALL CONCEPT

The goal of the developed approach is to formalize model subsets in the spirit of already used view definitions and to make them adaptable to specific needs. To enable database processing, the *Generalized Model Subset Definition* schema (GMSD) was created. It is capable of specifying generic views as well as specifically needed model subsets. Essentially, it is a neutral definition format with possible mappings for various practical data exchange and client/server realizations. The schema itself is comprised of two almost independent subparts providing functions for (1) object selection and (2) generic view definition. It follows the idea of selecting the needed high level object instances which are then processed by a predefined generic view, picking up all needed data related to these object instances. Thus, it is similar to the BLIS approach, which allows to subdivide views and to realize adaptability. The first part of GMSD is purely focused on the selection of object instances using set theory as baseline. Within this part a large set of functions is available to select object instances by value, class type and references. In contrast, the second part is intended for post-processing of the selected data in accordance with a predefined generic view. It can be used to:

- sort out object instances by their class type or reduce their data content on attribute level,
- sever references to unneeded objects to create a consistent data set without external, probably not managed references,
- generalise objects to supertypes, to lower the complexity of the resultant model subset,
- extend or reduce object selections by including/excluding referenced objects in accordance with some specified criteria.

The definition of generic views follows the concept to choose a default handling for all class types and attributes which can then be changed by adding exclusion criteria. This allows to reduce time and effort to formalize views, makes it more user friendly and results in rather compact data queries. More important is the concept of *subviews* allowing to specify handling of referenced object instances. It not only helps to structure and manage view definitions but also allows to differentiate the handling of object types by considering their use, i.e. the references to high level instances.

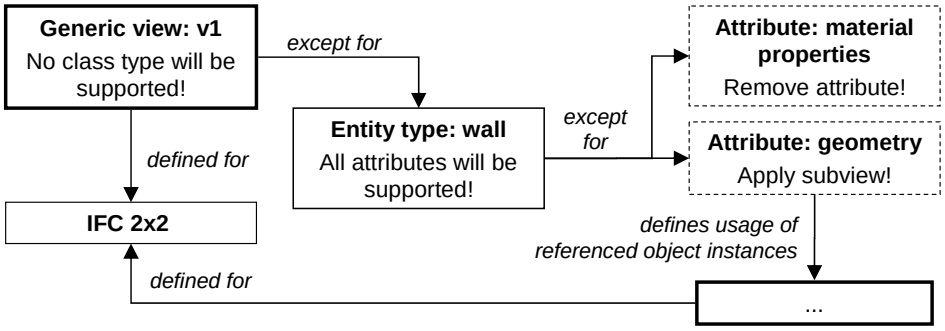

Figure 3: Example exüplaining the principle for defining generic views with GMSD

An example for a view definition is shown in figure 3. The generic view v1 by default supports no class type. An exception are *walls* which are defined as supported with all attributes. Again, an exception of this rule is defined for *material properties*, which shall be omitted, and *geometry*, which in this case is using a subview to pick up all relevant geometric object types such as *location*, *point* etc. More details about the GMSD specification can be found in Weise et al. (2003), discussing the underlying schema with its main entity types and the provided functionality for object selection and further object processing.

TOOL SUPPORT FOR THE DEFINITION OF GENERIC VIEWS

Since view definitions can be predefined and are the part being agreed on, tool support can significantly help to formalize specific needs, as for example to set up views dealing with capabilities of specific CAD applications or advanced work task types. Taking that into account, for populating the GMSD schema we have developed an editor allowing to interactively define generic views. A screenshot of the tool called *ViewEdit* is shown in figure 4 below. The GUI is divided in three panels containing (1) a tree view of the EXPRESS-based product data model with all entity types (classes) and attributes (features), which are provided in an inheritance graph and in alphabetical order, (2) a tree view of the view definition starting with high level classes being detailed by attribute handling, which itself can contain (sub-)views for handling of referenced object instances thereby deepening the view definition tree, and (3) a settings panel allowing to set various properties for the item selected in the view definition panel. As it can be seen in the screenshot, the settings panel contains a set of check and select boxes allowing to easily formalize generic views. Additionally, there is a text field for further annotations which is used to improve understandability and provide supplementary remarks.

The populated GMSD schema can be written according to the STEP physical file format. Thus, the formalized view can used by other EXPRESS tools and, if the semantics of GMSD is supported, it can be directly processed or mapped to a data query.

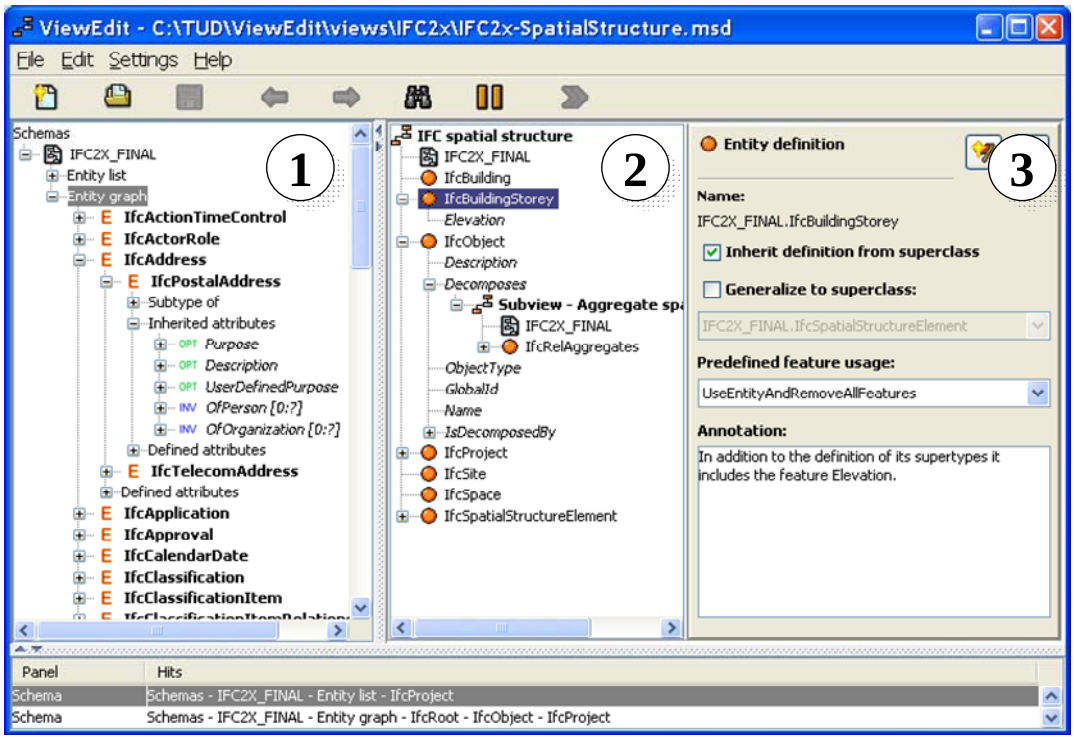

Figure 4: Screenshot of ViewEdit supporting the population of the GMSD schema

REQUESTS FOR MODEL SUBSETS

As mentioned above, generic views created with the help of *ViewEdit* can be used for data queries. For that a generic view has to be completed by selecting object instances. Basically there are two levels of selecting object instances: (1) by choosing a product model version, e.g. the design state created by Mark Miller on the 1st of February, and (2) by selecting individual object instances of the product model version, e.g. the first floor of the building being the object instance with identifier #4123 of class type *IfcBuildingStorey*. For example, if we use a generic view which starts at the level of *IfcBuildingStorey*, only handles object instances of this type and uses subviews for handling referenced object instances, the first level of object selection will create a model subset with all building storeys of the chosen product model version whereas the second level will narrow the model subset to only one of the building storeys.

Since the selection of object instances requires adapted user interfaces depending on the semantics of the underlying product data model, highly specialized clients are needed to apply generic views on the more precise level of individual object instances. In contrast, if user interactions can be limited to the selection of a product model version, a generic client can be

used instead. Figure 5 shows a developed simple generic IFC client allowing to dynamically narrow requests on the level of the spatial structure of a building. As it can be seen in the tree view of the client, the selection of object instances itself requires a view on all available building storeys. Thus, after choosing a product model version the generic view is used to visualize the full building structure which is then available for further user selections. Such clients can greatly simplify cooperative work processes but they require that model versioning is appropriately supported (cf. Weise et al. 2004).

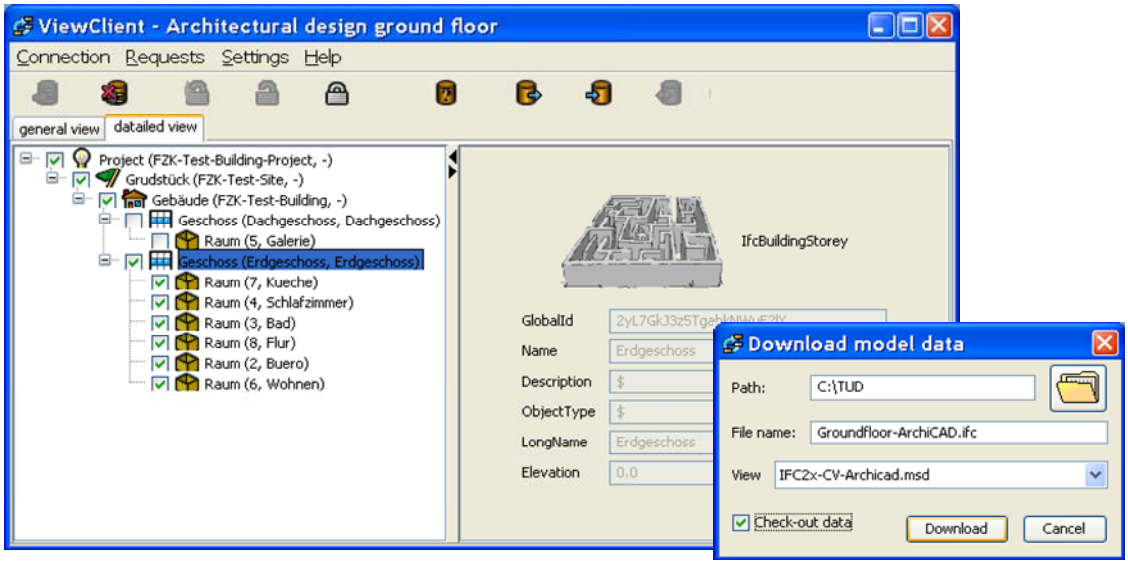

Figure 5: Screenshot of IFC-ViewClient supporting the selection of single storeys

BENEFITS FOR THE MANAGEMENT OF LONG TRANSACTIONS

The cooperation model shown in figure 1 is conceptually clear but comes up technically with a couple of questions regarding the update of the shared product model. These are: (1) the *matching* of (concurrently created) model versions, and (2) their *merging* into the common shared model. This final step of a long transaction can be a problem even if no parallel changes have to be synchronized. As already outlined, adaptable views allow concentrating on the really needed design data which frees CAD applications from the burden of handling the complete data set, which often results in data loss. The contribution of this research is to provide possibilities to easily create and adapt views for specific needs. We see two benefits of the developed approach:

- it provides the basis to detect design changes, and
- it supports the management of conflict resolution.

SUPPORT FOR CHANGE DETECTION

Change detection is necessary whenever a CAD application is creating a new design state. In all such cases matching of imported and exported design data is needed to detect the design

changes in terms of the used product model instances. This is related to a number of problems, starting with the lack of object identifiers for all object instances in the case of IFC. The compare algorithm presented in (Weise et al. 2005) is capable to compensate missing object identifiers but in order to properly detect deleted design data it relies on having a new design state without data loss. However, data exchange is still mostly file based and exchangeable model subsets are restricted by the conventions defined for the widely used STEP physical file format (ISO 10303-21). Consequently, requested model subsets have to be extended to ensure compatibility to the data exchange conventions. If such *added* data cannot be handled by the CAD application, the information will be lost in the file containing the changed design state. Therefore, matching requires model subsets according to the originally requested design data. Hence, the model subset definition needs to be applicable not only for requested but also for changed design states. In our approach, negative effects of changes to the model subset definition can be avoided in several ways, as e.g. by enforcing the use of invariant properties such as the identifier of an object instead of a user given name.

SUPPORT FOR CONFLICT MANAGEMENT

Applying changes to the shared product model can be done as described in (Weise & Katranuschkov 2005). To ensure consistency, data conflicts with the shared product model have to be detected and resolved. To some extend this step can be automated, normally limited to the level of the underlying data structure. However, at least for their meaning to the overall building design these changes have to be reviewed by other designers. With the help of model subsets design changes can be assigned to potentially affected designers, who have previously used or changed this kind of information. The design changes can then be evaluated with regard to these model subsets, allowing to derive responsibilities for review and conflict resolution. Compared to the typically preset definition of access rights, this dynamic evaluation of used model subsets enables a more natural way of working, requires no additional time for system configurations, and results in less time needed for change review.

CONCLUSIONS AND FURTHER RESEARCH NEEDS

The presented adaptable view approach has proven to work for a number of performed test cases, clearly demonstrating the abovementioned benefits. Nevertheless, further research is needed (1) *to avoid improper use of adaptability* by defining constraints for the combination of generic views with selected objects instances, and (2) *to extend the capabilities for defining generic views* by allowing more sophisticated conditions for picking up related object instances and more flexible treatment of references.

Improper use of generic views is currently handled with the help of the requesting client application supervising the selection of object instances. This is additionally restrained by invariant object identifiers enabling the handling of changed model subsets. It worked fine in all test examples but we should also have possibilities to check client requests on the server side, without external help. The second mentioned issue requires to enhance subview definition. This is currently limited to handling referenced object instances by static specifications on class level only. More flexible extending/cutting of references dynamically, depending on the actual state, can enable more precise view definitions with even fewer user interactions.

ACKNOWLEDGMENTS

The authors wish to acknowledge the support of the German Research Foundation (DFG) providing funding for this project in the frame of the priority program 1103.

REFERENCES

Adachi Y. (2002). *Overview of Partial Model Query Language*, VTT Building and Transport / SECOM Co. Ltd., Intelligent Systems Lab., VTT Report VTT-TEC-ADA-12.

Amor R. & Faraj I. (2001). *Misconceptions about Integrated Project Databases.* ITcon Vol. 6. Available from: http://itcon.org/2001/5/.

Björk B.-C. (1995). *Requirements and Information Structures for Building Product Data Models*, VTT Publ. No. 245, Espoo, Finland.

Denno P. /ed/ (1999). *EXPRESS-X Language Reference Manual*, ISO/TC184/SC4/WG11/ N088. Available from: http://www.steptools.com/library/express-x/n088.pdf

Hietanen J. /ed./ (2002). *Building Lifecycle Interoperable Software (BLIS).* Available from: http://www.blis-project.org/index2.html

ISO 10303-21 IS (1994) /Cor.1:1999/. *Industrial Automation Systems and Integration – Product Data Representation and Exchange -- Part 21: Implementation Methods: Clear Text Encoding of the Exchange Structure*, ISO TC 184/SC4, Geneva, Switzerland.

Liebich T., Adachi Y., Forester J., Hyvarinen J., Karstila K. & Wix J. (2006). *Industry Foundation Classes IFC 2x3*, © International Alliance for Interoperability. Available from: http://www.iai-international.org/Model/IFC(ifcXML)Specs.html

Fowler J. (1995): *STEP for Data Management, Exchange and Sharing*, Technology Appraisals Ltd., Twickenham, UK.

ProIT (2004). *ProIT: Product Model Data in the Construction Process.* © IAI Int. Solutions.

Steinmann R. (2006). *Web-site of IAI's ISG, the Implementer Support Group.* http://www.bauwesen.fh-muenchen.de/iai/iai_isg/.

Weise M., Katranuschkov P. & Scherer R. J. (2003). *Generalised Model Subset Definition Schema*, Proc. CIB-W78 Conference 2003 – Information Technology for Construction, Auckland, NZ. Available from: https://www.cs.auckland.ac.nz/w78/papers/W78-54.pdf.

Weise M., Katranuschkov P. & Scherer, R. J. (2004). *Generic Services for the Support of Evolving Building Model Data*, Proc. ICCCBE-X Conference, Weimar, Germany.

Weise M. & Katranuschkov P. (2005). *Supporting State-based Transactions in Collaborative Product Modelling Environments*, Proc. CIB-W78 Conference 2005 - Distributing Knowledge in Building, Dresden, Germany.

Wix J. /ed./ (2005). *Information Delivery and Framework*, Presentation at the IAI International Council in Oslo, Norway, 31 May 2005 http://www.nibs.org/FMOC/71305/2_InformationDeliveryAndFramework.pdf