

Generic Services for the Support of Evolving Building Model Data

Matthias Weise, Peter Katranuschkov, Raimar J. Scherer

Institute for Construction Informatics, University of Technology Dresden, Germany
{Matthias.Weise}, {Peter.Katranuschkov}, {Raimar.Scherer}@cib.bau.tu-dresden.de

Summary

Many problems related to data integration in AEC can be better tackled by an approach that takes into account the heterogeneity of tasks, models and applications but does not require continuous consistency of the evolving design data, at each data management operation. Such an approach must incorporate adequate services that can facilitate reintegration of concurrently modified data at reasonably selected coordination points.

In this paper we present a set of methods which, used in combination, can achieve that goal. After a description of the principal envisaged cooperative work scenario each of these methods is discussed in detail and current observations drawn from their software realisation are given. Whilst the suggested approach is valid for any EXPRESS-based data model, the practical focus of work has been on facilitating IFC-driven integration.

1 Current State and Problems of Data Integration in AEC

Model-based data exchange and sharing are commonly accepted as vital prerequisites for collaborative design work to date. A lot of efforts in the areas of conceptual modelling, software architectures and interoperability have been made in the last decades and valuable results have been achieved which are now approaching a degree of fruition with the development of the IFC project model of the IAI (Wix and Liebich 2001). However, seamless data sharing is still not available for wide practical use. There are various technical, economical, legal and even social reasons for that. In this paper we outline a suggested pragmatic and scalable approach to tackle *critical technical problems* related to model-based data integration in cooperative work. These problems are shortly outlined below. Throughout the discussions we use the terms *data model* (i.e. the schema according to which individual models can be created and described) and *model* (i.e. the data abstraction representing an actual facility, such as ‘the house of John Doe’) as defined in (Björk 1995).

1.1 The Heterogeneity Problem in Collaborative Work

The design of a building is a complex process that inevitably results in job specialisation and a certain simplification of real world problems. With regard to collaborative work this means to deal with heterogeneity in many areas involving different company profiles, different actors and roles, kinds of tasks, applications used etc. In view of data sharing, heavy abstractions of the planned ‘one-off’ facility are needed to allow efficient design (Turk 2001). Typically, each design tool uses a specific data model that is often different even for applications of the same domain. In fact, in design all domain and application models are conceptually interlinked by a continuously changing virtual (i.e. not yet existing) building, whereas one complete model which merges all needed views on the data does not exist. Moreover, the used models are not disjoint but contain partially the same pieces of technical information, represented redundantly and in different ways.

We have to accept that design professionals need different models and that applications will continue to define and maintain their own views on the building data. Consequently, achievement of data integration requires sophisticated *model mappings*, which in turn may produce additional data conflicts or even data loss.

1.2 The Problem of How to Achieve Consistent Data Sharing

The concept of data sharing is usually associated with the vision of consistent, up-to-date and complete design data. According to that, all design professionals should have access to a shared data repository to integrate their (partial) design solutions. Added or updated building data should be checked for consistency by using a set of design rules to resolve possible design conflicts.

Clearly, this level of integration is a desirable goal. It has been achieved by a number of software architectures based on a closed, dedicated modelling world or on sophisticated data integration services used upon a limited set of design tools (Eastman 1999). Typically such architectures require well harmonised applications capable to process agreed data models. Users have to accept the agreed environment and to adjust their work to it. Inclusion of 'foreign' legacy applications is hardly possible if this was not planned in advance. In short they work only for a limited, clearly defined scope. Scaling up to the full set of practical use cases in computer-integrated construction (CIC) is unlikely to be achieved due to the extreme increase of complexity with regard to modelling representations, model mappings and consistency (Amor and Faraj 2001). There is a need of open, scalable solutions agreed in consensus but this is not easy to achieve in the highly fragmented landscape of the construction industry.

We have to recognise that the road towards comprehensive life cycle data sharing will include a number of incremental steps seeking to find the optimal balance between fully automated consistent solutions for limited subsets of the design data and adequate interactive functions to fill in the gaps. Consequently, a concept allowing to mitigate hard demands associated with the problems of data sharing is required.

1.3 Problems Related to Standardisation

Standardisation is commonly accepted as a basis for improving cooperation. It is necessary in various areas related to data integration, such as data modelling, communication protocols, data exchange formats, data access APIs and so on. Major results are already available in all these areas but there does not yet exist a stable set of well-defined standards for CIC. Instead, there are many competing standards available and most are still under development, with constant updates happening in short periods of time. All this impedes their broad use in practice.

Specifically for data modelling there exists a rich family of standards known as ISO STEP, but due to its complexity STEP is used to a very limited extent in AEC. However, it provides a baseline for agreements of many industry-specific data models such as the internationally coordinated Industry Foundation Classes (IFC) and various domain-related developments as e.g. CIS/2 and the German PSS (for steel construction), OKSTRA (for road works) etc. All these models use the EXPRESS modelling language and are therefore based on the same modelling paradigm and data exchange technologies. Their scope ranges from domain-specific data exchange to full life cycle management of AEC projects. The latter is specifically addressed by the IFC project model which applies a layered architecture to enable integration of different design domains. However, in order to fulfil its designated scope IFC defines only the most important data needed for inter-discipline cooperation. It is a light-weight, incomplete data model whose primary purpose is in providing reusable concepts for various domain extensions to help harmonise the data structures and reduce redundancies and overlaps.

Currently, the IFC project model is supported by a large number of applications but they are all limited to file based data exchange and cannot be used very efficiently for collaborative work. The existing shortcomings are mainly due to the broad scope and the large amount of alternative options in IFC, the limitations of file-based data exchange and incomplete IFC conformance.

We have to recognise that standards enabling collaborative work have not yet fully penetrated design and construction practice. File-based data exchange of (partially) standardised model data is the current, quite insufficient common denominator with regard to data sharing. IFC provides a good basis for true database transactions but respective services are yet to be developed.

1.4 Database Problems

Already in the mid 1980s it was recognised that many of the problems experienced by CAD, CIM and CIC are in fact database problems (Robinson 1988). However, whilst research in database theory and artificial intelligence has brought about various new powerful methods to ensure data consistency, achieve multi-database integration and map data from one schema into another, progress in engineering data bases has not been so strong. Basically, this is due to the difficulties related to the support of long transactions. There exist many partial methods to support consistency, tackle various kinds of data conflicts, optimise check-out/check-in sequences etc., but a complete solution to this problem is not available yet (cf. Conrad 1997).

However, we need to accept that engineering work requires long transactions and, even worse, such transactions can (and should) happen concurrently. Consequently, the goal is to provide pragmatic methods which may not guarantee full consistency but should support the users to regain consistent model states.

In the following sections we describe a suggested new approach to tackle the above and other related problems. It is based on the conviction that full integration and consistency of the evolving design data are not needed continuously but only at specific coordination points, reasonably selected by the design team. We do not try to create a closed ideal world but provide an open solution which would allow to reduce data loss, improve data sharing quality and reach a 'reasonable' degree of consistency.

2 A Suggested Scalable Approach for Improving Data Integration

We propose an approach based on a set of inter-related services to support integrated data sharing in cooperative work environments. It does not promise a perfect environment providing faultless data integrity. The strategy is to mitigate the requirements to the involved engineering applications, reduce data loss caused by data mapping and other data conflicts, and at the same time take into account practical deficiencies in current data models and their software implementations. The suggested process is based on *long transactions* allowing off-line modifications and, in order to support parallel work, involving versioning and merging of concurrently changed data. Specifically considered are problems related to the use of a (potentially unlimited) number of *legacy applications* which are typically constrained by file based data exchange and their internally used data structures. Using legacy applications often results in unknown data changes and data loss, which are both very critical for collaborative work.

The suggested services comprise: (1) extraction of model subsets that are of interest for a specific design task, (2) mapping of the model data to tackle different modelling representations, (3) matching of two successive model versions to help recognise properly the latest modified data and (4) merging of concurrently made data changes. We assume a common modelling paradigm and a commonly agreed (standardised) data model to represent the data to be shared. Taking into account current practices and trends, the developed services are based on the broadly acknowledged EXPRESS modelling language and can thus be verified and used with the IFC project model and the legacy applications supporting IFC-based data exchange.

For all developed methods version management plays an important role. However, due to the size limitations of the paper it will not be separately discussed.

2.1 Principal Application Scenario

Whilst there are many different use cases where the developed services can be applied, it can be shown that they can all be derived from the principal scenario shown on Figure 1 below. It starts at time point t_i with the consistent shared model version M_i based on the product data model M defining the data that have to be shared and proceeds until the next coordination point t_{i+c} is reached.

The data processing sequence for a single designer is comprised of the following six steps:

- 1) model subset definition and subsequent extraction of the needed model data from M_i to a model subset Ms_i , which can be expressed as $Ms_i = \text{createSubset}(M_i, \text{subsetDef}(M_i))$
- 2) mapping of the model subset Ms_i to the domain model S_i which represents an instantiation of the domain data model S , i.e. $S_i = \text{map}(Ms_i, \text{mappingDef}(M, S))$
- 3) modification of S_i to S_{i+1} by the user via some legacy application that can be expressed abstractly as $S_{i+1} = \text{userModify}(S_i, \text{useApplication}(A, S_i))$
- 4) backward mapping of S_{i+1} to Ms_{i+1} , i.e. $Ms_{i+1} = \text{map}(S_{i+1}, \text{mappingDef}(S, M))$
- 5) model matching of Ms_i and Ms_{i+1} including object identification, comparison and evaluation resulting in the found differences $\Delta Ms_{i+1,i}$, i.e. $\Delta Ms_{i+1,i} = \text{match}(Ms_i, Ms_{i+1})$
- 6) reintegration of $\Delta Ms_{i+1,i}$ in M_i resulting in M_{i+1} , i.e. $M_{i+1} = \text{reintegrate}(M_i, \Delta Ms_{i+1,i})$.

The final consistent model M_{i+1} can then be merged with the divergent design data of the other designers (modified in parallel using the same procedure) at the coordination time point t_{i+c} to obtain a new stable model state M_{i+c} , i.e. $M_{i+c} = \text{merge}(M_{i+1}, M_{i+2}, \dots, M_{i+k})$, assuming that k = the number of concurrently changed checked out models.

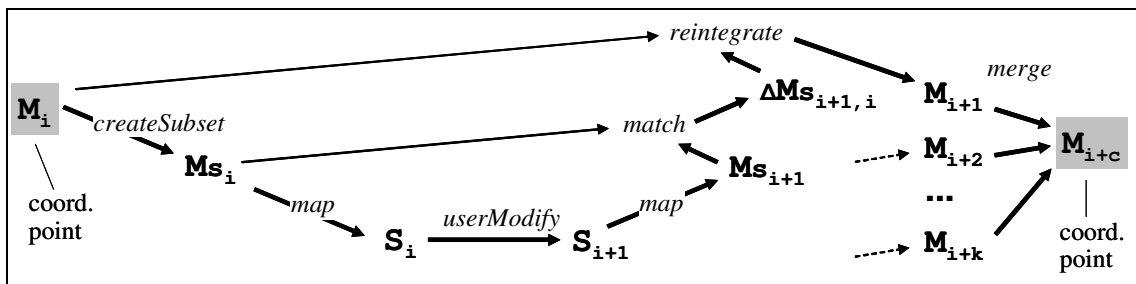


Figure 1: Principal application scenario for using the suggested generic services

In addition, the former state of the domain data model S at time t_{i-1} is used to fill in data that cannot be represented in M and hence will get lost by the mapping from S_{i-1} to Ms_{i-1} and hence in S_i too. By applying model matching and reintegration for S_i and S_{i-1} this data loss can be strongly reduced or even fully eliminated, avoiding time consuming and error-prone rework.

If a standardised data model that can be processed by the involved engineering applications is used, as e.g. IFC, then the model mapping shown in Figure 1 will not be necessary. Similarly, if all common data from M_i can be processed by the used application(s), subset creation can be skipped. Matching, reintegration and merging will always be needed to provide for concurrent work. Problems arising from the used applications may vary from misinterpreted object structures to data loss. Such problems can be tackled to a certain extent by additional correction of the data achieved by applying a dedicated model subset extraction reducing the resultant data set to semantically changed data and a subsequent adjusting model mapping.

Legacy applications supporting file-based data exchange can be integrated in the shown scenario without any re-engineering via an additional client application providing the suggested model management services, adapted to the shared data model used.

2.2 Tackling the Evolution of Building Model Data

Modelling objects are not static but evolve continuously during the design process, absorbing newly defined properties but also changing existing ones. Reasons for that are:

- Technical changes (selection of another design alternative),
- Changes through further detailing causing modifications in the data structures, and
- Changes through mappings.

Whilst the first two of these are fairly obvious, the last needs some more explanation. Indeed, model mapping is necessary to deal with different model representations but, like translating a sentence into another language where a backward translation most often results in a different sentence, it may cause changes of the data structure without semantic reasons. Such changes can be very subtle and therefore extremely difficult to detect.

Attempting to tackle the problems related to model data evolution by providing unique object identifiers and a version history for each object only helps in simpler situations where deeper referencing among the objects is not involved. Besides, in all practical data models identifiers are provided only for primary (design) objects whereas resource objects such as point, line, date, time, unit etc. are not assigned identifiers in order to reduce as much as possible the overall huge data volume and to simplify processing of the model data by the applications.

Therefore, more sophisticated methods are needed to deal with all changes that may occur by model data evolution. The suggested *model matching* method discussed further below provides an incomplete solution but it yields results that are much closer to the ideal case than currently used practical approaches.

3 Generic Services for the Support of the Evolving Model Data in Cooperative Work

3.1 Model Subset Extraction

The advantages of using model subsets are manifold. They can provide support to domain applications that are not capable to ‘understand’ the full shared global model, but they can also considerably reduce the quantity of the exchanged data, provide well-defined or even standardised domain views and simplify the data consistency and coordination problems. To be used in the envisaged scenario, a model subset should be (1) easily definable with as few as possible statements, (2) completely described within a single service request and (3) usable for reintegration of changed data by allowing elimination of mismanaged data sets. To meet these requirements a *Generalised Model Subset Definition* schema (GMSD) has been developed. It is a neutral definition format for EXPRESS-based models comprised of two subparts which are almost independent of each other with regard to data but are strongly inter-related in the overall process. These two parts are: (1) *object selection*, and (2) *view definition*. As the name suggests, the first part is purely focused on the selection of object instances using set theory as baseline. The second part is intended for post-processing of the selected data in accordance with a specific partial model view. Such views can be completely predefined to support standard domain/discipline perspectives. Figure 2 below presents the top level entities of the GMSD schema and illustrates the envisaged method of its use in run-time model server environments. More details on GMSD are provided in (Weise et al. 2003).

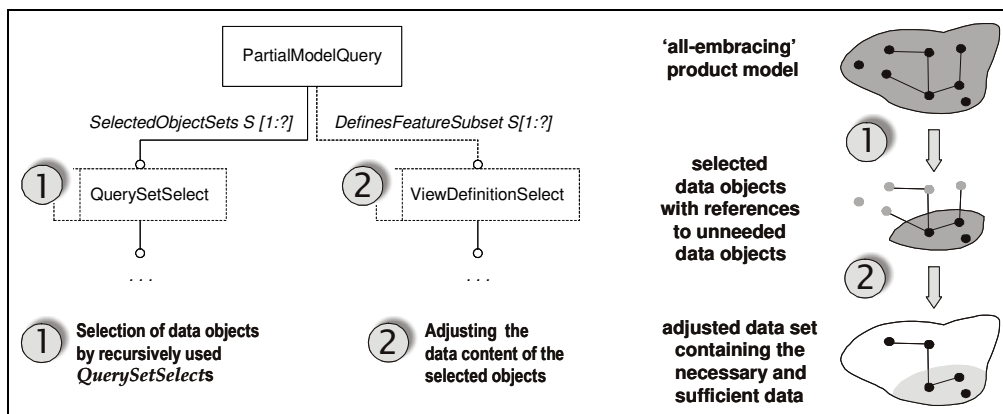


Figure 2: Top level structure of the GMSD schema and the related instance level operations

Figure 3 shows some possible uses of the GMSD schema. The first *createSubset()* operation generates the model subset Ms_i which is then modified by some legacy application to Ms_{i+1} . However, in the case of using a STEP physical file mandatory attributes have to be written to Ms_i even if not defined by the model subset request. This results in superfluous and often mismanaged information. Removing such attributes can be achieved by applying the same *createSubset()* operation to Ms_{i+1} too. The model subset can be further advantageously reduced by removing untouched data. For that purpose, a new subset extraction has to be specified and applied to both Ms_i and Ms_{i+1}^* so that the resulting model subsets Ms_i' and Ms_{i+1}' are processible by the following model matching service.

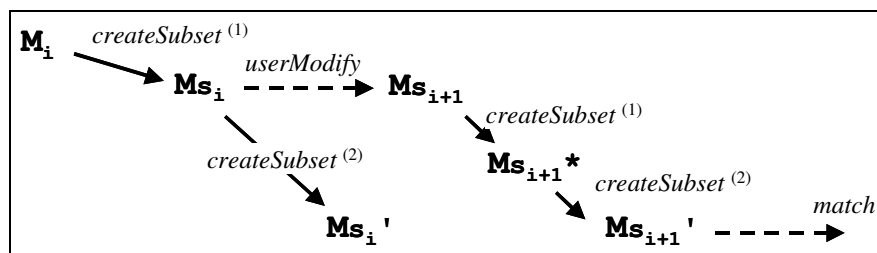


Figure 3: Applying model subset extraction to remove unchanged data before matching

3.2 Model Mapping

In principle, model mapping involves a procedure that is very similar to multidatabase integration known from the area of database research (Spaccapietra and Parent 1994). It consists of the following steps: (1) Detection of schema overlaps, (2) Detection of inter-schema conflicts, (3) Definition of the inter-schema correspondences with the help of formal mapping specifications, and (4) Use of appropriate mapping methods for the actual transformations on entity instance level at run-time.

The identification of typical *mapping patterns* allows to understand better the mapping task and to formalise what and how has to be mapped in each particular case. Specifically, by examining the theoretical background of object-oriented modelling as well as a number of practical cases, the following set of mapping patterns can be identified:

- *Unconditional class level mapping patterns*: These patterns depict the most general high-level mappings that affect all instances of the involved source and target classes.
- *Conditional instance level mapping patterns*: In contrast to the general definition of mappings on class level, instance level mappings may include certain conditions to select the set of instances to be mapped from the full set of available instances in the source model, or – in the case of multiple cardinality – from the cross product of all referenced sets of instances.
- *Attribute level mapping patterns*: These patterns depict how an attribute with a given data type should be mapped. However, in object-oriented models the data type of an attribute can be quite complex. Consequently, there are many different patterns that have to be considered, in many cases they can even govern the actual resulting cardinality of the mapping on entity level.

Attribute mapping patterns are further subdivided into: (1) basic attribute mapping patterns, (2) complex attribute mapping patterns, and (3) generative attribute mapping patterns. For each of these categories, several sub-cases have been identified, such as "*simple equivalence*", "*set equivalence*", "*functional equivalence*" for the first category; "*grouping*", "*ungrouping*", "*homomorphic mapping*", "*transitive mapping*", "*inverse transitive mapping*" for the second category; "*simple generative*", "*functional generative*" for the third category, and so on.

Figure 4 provides the formalism used to present these patterns graphically, along with respective typical examples.

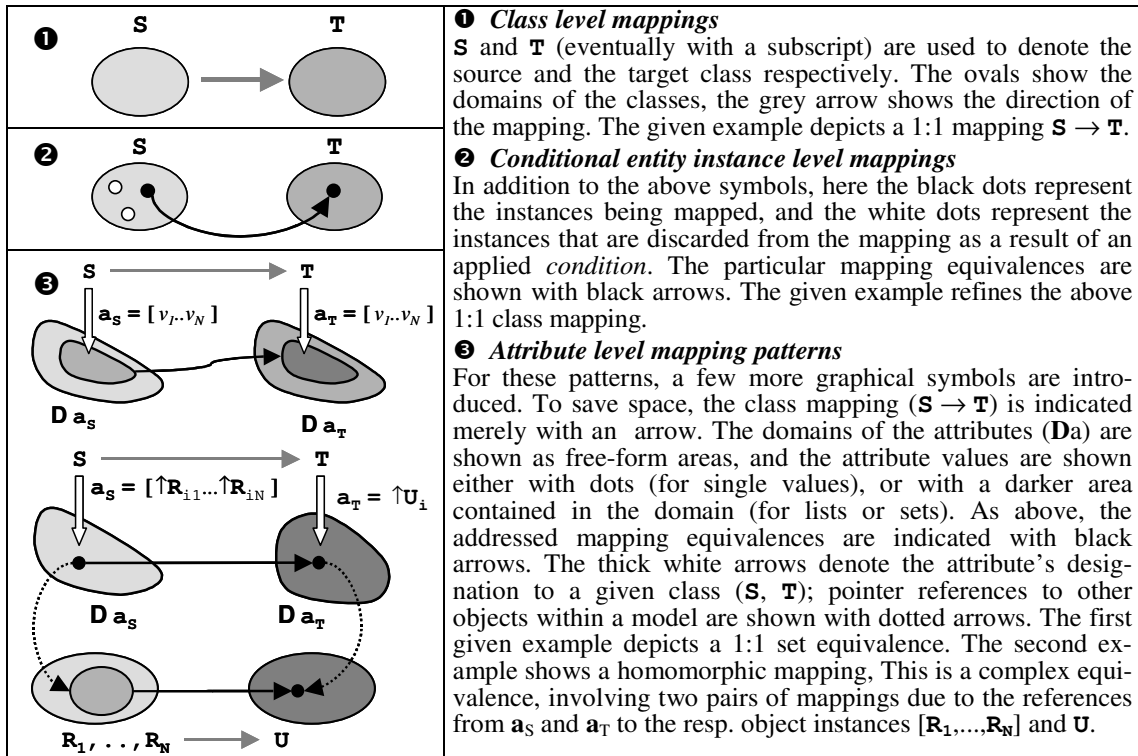


Figure 4: Graphical symbols used for the presentation of mapping patterns with illustrative pattern examples

As a whole 29 different mapping patterns have been defined in (Katranuschkov 2001). Some of these are relatively simple to implement but there are also several patterns that require quite sophisticated implementation methods. However, for the specific mapping cases regarding the IFC model only some of these patterns are really needed. IFC defines several pragmatic requirements restricting the use of EXPRESS modelling concepts which simplifies considerably the mapping task. More details on the specification of mappings and the respective mapping operations are provided in (Katranuschkov 2001) and in (Wagner and Scherer 2002).

3.3 Model Matching

Model matching has to deal with the identification of data changes and reintegration of the used model subsets into the shareable set of design data. In our approach it exploits the object structure without considering its semantic meaning and hence can be applied to different data models. However, the efficiency of this generic realisation of the mapping operation is dependent on the (structural) model definitions. In general we have to consider that (1) not all objects can be uniquely identified by a kind of a key value, (2) a structural difference in the data does not always imply a change in their semantic meaning, (3) eventual data loss caused by model mapping may emerge on object and attribute level and (4) errors from the used application may result in replacement of valid data (such as wrongly replaced IDs, replacements by default values etc.). As indicated in the previous sections, the problems (2), (3) and (4) are to be tackled by the model subset definition and adjusting model mappings. Thus, the suggested model matching method has to deal basically with the problems of object identification and reintegration of the changed data.

3.3.1 Comparison of model data

Comparison of the model data begins with the identification of pairs of potentially matching objects, established by using their unique identifiers or some other key value. Unfortunately, as mentioned before, such identifier are not available for all objects. Several approaches dealing with this problem are known. In object-oriented databases the so called *deep compare* method is used to compare two objects and all of their references. The latter can be quite useful for model

matching because comparisons can first be done for uniquely identifiable objects, leading – iteratively – to the remaining, non identifiable objects. However, in the case where referenced objects can not be uniquely identified a costly value-based evaluation of possible object pairs is necessary. Normally, an object tree created by tracking all references of an object will be countable but tends to become huge by more sophisticated product data models. Additionally, we have to be aware that unidentifiable objects may be shared, i.e. referenced by different identifiable objects. The general complexity of the problem is shown in (Spinner 1989) where a fully generic tree-matching algorithm is classified as NP-complete. Hanff et al. (1997) have analysed a variation of the deep compare method adapted to EXPRESS-based model data and applied to STEP AP 201. They have also outlined the problem of missing identifiers and conclude that unique identifiers are needed for all data objects to guarantee faultless matching. In the theory of *tree to tree correction* the minimal modification costs are used as criteria to decide the question of matching trees. For this purpose polynomial scaling algorithms are suggested (Schroeder 1994). Closely related to model matching is also the problem of detecting objects with the same meaning, contained in different data bases using (slightly) different representations (Neiling 2004). However, all these methods are only of limited use here because they are too ineffective or even not manageable in reasonable time for the given complexity of the matching problem we have to deal with.

Therefore we suggest an algorithm that provides a simpler scalable way for finding corresponding data objects. Its essence is in the iterative generation of object pairs by evaluation of equivalent references of already validated object pairs. The first set of valid object pairs is built by unambiguously definable object pairs. Any newly found object pair will then be validated in one of the following cycles depending on its weighting factor derived from the *type* of the reference responsible for its creation. Attribute values will only be used if ambiguities of *aggregated references* do not allow the generation of new object pairs. In order to avoid a costly evaluation of attribute values a hashcode is used, indicating identical references. In this way, the pairwise comparison of objects can be significantly reduced in respect to all other approaches.

Figure 5 illustrates the outlined procedure. Before starting any comparison of objects the set **VP** of validated object pairs and the set **UO** of unidentifiable objects will be initially created using available unique identifiers. After creation of VP and UO the object pairs of VP will be compared as shown on the right side of Figure 5 for $\{A_1, A_2\}$. Using their equivalent references ‘has_material’ a new object pair can be assumed for the objects E_1 and G_2 of UO. Additionally, a weighting factor indicating the validity of this assumption will be derived from the *reference type* of ‘has_material’ and added to the newly created object pair, which is then placed in the set **AP** containing all such assumed object pairs.

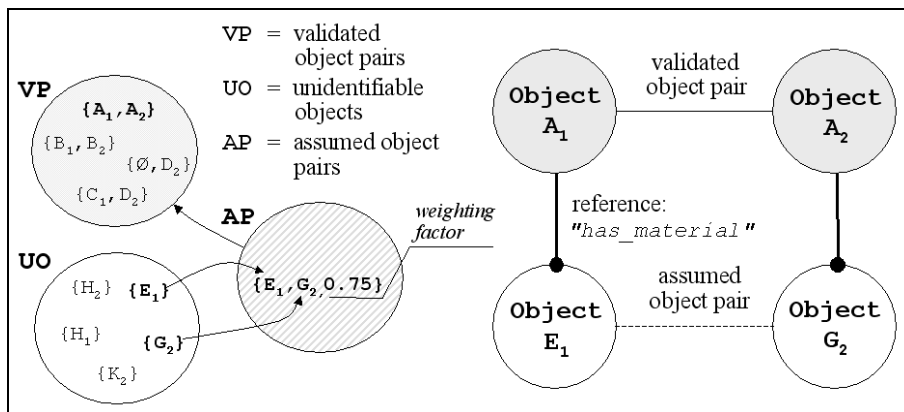


Figure 5: Iterative establishment of object pairs

Within this approach reference types are differentiated in (1) simple references, (2) ordered aggregated references and (3) unordered aggregated references. After comparison of all object

pairs of VP the highest weighted object pairs of AP will be moved from AP (and UO) to VP and a new iteration cycle will be started. However, now the weighting factors of newly created object pairs will be combined with the weighting factors of the validated object pairs.

It can be shown that the cycle will always terminate at most after $n-1$ steps, where n is the total number of objects and the process starts with only one valid pair. However, a terminated process may result in a non empty set UO. In that case a strategy similar to the assignment of unordered aggregated references can be applied to find further pairs of objects.

The described algorithm is heavily dependent on the evaluation of object references differentiated in the three mentioned reference types, as shown in Figure 6 below. The simple reference allows a well-defined creation of object pairs and hence is assigned the highest weighting factor. More interesting is the handling of ordered and unordered aggregated references as they include multiple combinations of object pairs. The order of references allows an assignment by position, if both aggregations are of the same size, or by the best matching sequence identified by using the hashcodes of the referenced objects. Such hashcodes are used also for the assignment of unordered aggregated references. However, as there are also some other criteria helping to create object pairs, such as multiple occurrence (of objects) and single non matching objects in both aggregations, the procedure is mainly limited to relate unchanged objects.


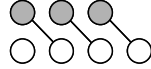
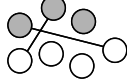
Type of reference	Object A ₁	Object A ₂	Object pairs	
Simple reference „has_material“	{hc342}	{hc923}	{hc342, hc923}	
Ordered aggregated reference „vertex“	{{1, hc23}, {2, hc12}, {3, hc56}}	{{1, hc09}, {2, hc23}, {3, hc10}, {4, hc56}}	{hc23, hc23}, {hc12, hc10}, {hc56, hc56}	
Unordered aggregated reference „neighbour_of“	{hc71, hc45, hc89}	{hc45, hc74, hc91, hc71}	{hc71, hc71} {hc45, hc45}	

Figure 6: Reference types and their relevance for the derivation of object pairs (hc = hashcode)

As shown, hashcodes are used for the decision of matching / non-matching objects if a well-defined assignment is not possible. The hashcode of an object is represented by an integer value and is calculated from the object state. Equal hashcodes are a strong indication for equal objects. The major benefit compared to other approaches is the avoidance of a costly pairwise attribute evaluation. However, when dealing with all references of an object, the calculation of hashcodes can also be quite time consuming. Therefore, a differentiation is made between a *qualified hashcode*, including the complete set of information, and a *simple hashcode*, constructed by ignoring the hashcodes of referenced objects.

3.3.2 Re-integration of changed model subsets

If model subsets are used, as shown in the discussed general scenario in section 2.1, the re-integration of changed data becomes necessary. As discussed in section 3.1 model subset extraction creates a model subset by removing data objects, cutting or reducing references, filtering attributes etc. Reintegration means to invert the process of model subset extraction, i.e. to add removed data objects, restore cut or reduced references and re-create filtered out attributes. It is heavily based on the model subset definition achieved via GMSD and on the results of the model comparison. Most critical for the reintegration process are wrongly established object pairs and rearrangements of the object structure without changes of the semantic meaning, such as changed units or coordinate systems.

Figure 7 below illustrates the process of using model subsets. By applying a GMSD-based *createSubset()* operation to a given model version some objects and attributes will be removed.

For object O_1 this results in a new version O_{S1} in which the simple reference 'a' is removed and the aggregated reference 'b' is downsized by one element. This object is then modified externally by some application to O_{S2} which differs from O_{S1} in the aggregated reference 'b', downsized by another element, and the simple references 'c' and 'd'. The reintegration (which is the second part of the shown *match()* operation) adds all objects and attributes from O_1 that have been removed according to the model subset definition. In this particular case, this will recreate the cut/downsized references 'a' and 'b'. However, it will take care not to add objects/attributes that have been modified by the used application.

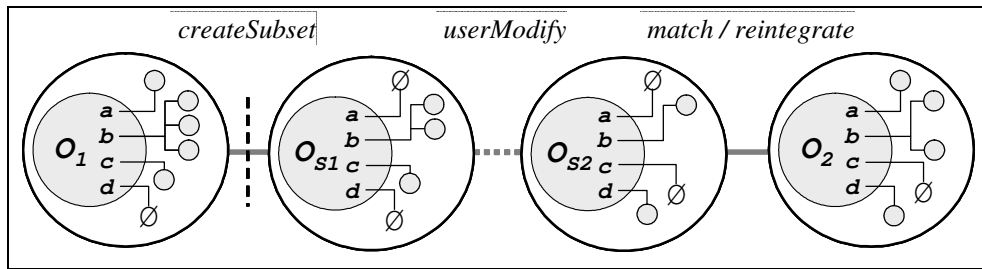


Figure 7: Reintegration of model subsets

The described reintegration scenario is limited to object pairs in 1:1 relationships. Other cases, such as n:m relationships or change of object types lead to more complex situations and may require some additional user interactions. Here it must also be noted that even in the restricted reintegration scenario shown above full consistency of the data cannot be guaranteed. For that purpose additional constraint definitions that can help to detect semantic conflicts are needed. However, in current practical data model specifications such definitions are mostly limited to simple rules providing restrictions of cardinality, uniqueness of values and so on. Therefore the degree of data consistency must finally be evaluated by the designers during the coordinating merging process discussed in the next section.

3.4 Merging of Divergent Model Data

Merging has to deal with the consistency of concurrently changed design data that exist in two or more divergent versions. It should be performed at a coordination point defined by the design team and has to be performed in cooperation of all involved designers. The aim is to provide a procedure by which modifications can be reconciled and appropriately adjusted to a consistent new model state marking the begin of a new stage of the concurrent design process. Figure 8 below illustrates the suggested concept of using available prior knowledge to enable efficient management of the iterative agreement process.

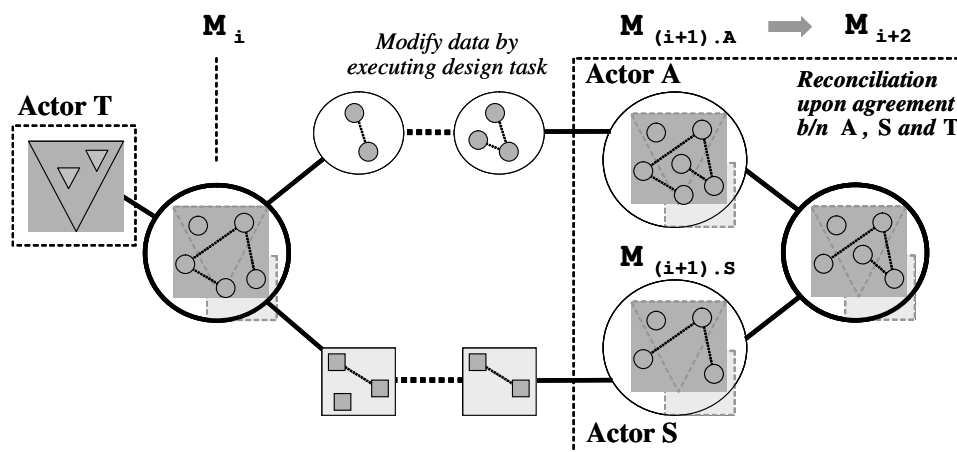


Figure 8: Principal schema for merging of concurrently changed design data

In the shown schema, the history of already finished transactions involving discipline-specific model subsets and respective data modifications plays a role of utmost importance. Specifically, in the example of Figure 8, both directly involved designers, named **A** and **S**, will have to agree to a conjointly accepted new model version M_{i+2} . However, these changes may often affect data used by other designers. For instance, a new object created by designer **A** can be also in the domain of designer **T** who must therefore be involved in the agreement process as well.

In principle, the merging operation is highly interactive as it requires active negotiations and approvals. However, the results of preceding matchings, identifying made modifications, and the change history of the model data can enable automated suggestions and consistent integration of approved changes. Here storing of model deltas and logging of the data used in each planning step can be helpful. This is on-going work which does not yet allow to draw detailed conclusions.

4 Discussion

In the preceding sections we outlined a novel approach enabling the tackling of evolving, concurrently changed design data in AEC. The developed schemas and methods constituting the suggested approach provide an overall framework allowing to solve many existing data integration problems in distributed cooperative work environments. However, there are also many open issues yet to be considered. The current implementation of the approach and performed test studies provided a basis to draw first observations and to identify further development steps.

The prototype software environment was realised in Java, using TCP/IP-based client/server interactions. The integration of a set of legacy applications with the developed model server implementing the described data integration services was done with the help of a model-independent client enabling the check-in/check-out of model data. Additionally, for the execution of practice relevant queries, a light-weight editor for view and mapping definitions and for model browsing was developed.

Specific tests of the developed services have been carried out for the IFC project model (versions 1.5.1, 2x and 2x2) and for the German PSS model. Creation, modification, reading and writing of IFC data was tested for the CAD applications Allplan (Nemetschek), ADT (Autodesk) with the IFC plug-in of Inopso, ArchiCAD (Graphisoft) and DDS. The goal of these tests was (1) to examine the degree to which changed/unchanged objects are correctly identified, and (2) to measure the performance of the developed algorithms.

The tests showed that: (1) at most 5% of the objects in an IFC 2x model are directly identifiable, (2) in most cases matching object pairs can be correctly established, and (3) the performance is approximately linearly dependent on the model size and seems adequate for practical needs. However, complete identification of all object pairs could only be reached by analysing all types of references described in section 3.3.1 above, whereby the performance of the algorithm appeared to be strongly related to the number of qualified hashcodes that need to be computed. Deliberately made changes could be recognised with varying success. The decisive factor here was the (re)integration of the changed objects in the existing object structure. Objects that could not be properly recognised as ‘changed’ were classified as ‘new’ thereby leading, cascadingly, to the same decision for objects referentially dependent on them. However, for the IFC data structure propagation of this effect seems to remain always on a limited scale.

Results for the PSS data model were better, due to its narrower scope and less complex structure. For example, in the studied test cases nearly 30% of all objects were directly identifiable.

Overall, the performed studies showed that the developed approach can be successfully applied to IFC. Whilst due to the different quality of processing of the IFC data by the examined applications, a satisfactorily stable procedure cannot yet be warranted for practical purposes, it is hoped that the ongoing harmonisation within IFC coupled with the suggested data integration

services can provide a level of automation enabling a better cooperative work process even without (the expected) further improvement of the engineering applications supporting IFC. Further refinement of the methodology will identify the degree to which this can be achieved.

5 Acknowledgements

The authors wish to acknowledge the support of the German Research Foundation (DFG), and the helpful involvement of FZK (Karlsruhe) in the testing of the developed services.

6 References

Amor, R. and 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.

Conrad, S. (1997): *Föderierte Datenbanksysteme: Konzepte der Datenintegration* (in German), Springer, Berlin-Heidelberg-New York, 331 p.

Eastman, C. M. (1999): *Building Product Models: Computer Environments Supporting Design and Construction*. CRC Press, Boca Raton, Florida.

Hanff, J. Freundt, M., Ilieva, D. and Pahl, P. J. (1997): *Das Problem der Identifizierung von Zeichnungsobjekten im Rahmen des Innovationsprojektes „Eine Datenbahn für das Bauen“* (in German), in: Fortschritt-Berichte VDI Reihe 4 Nr. 140. Düsseldorf: VDI Verlag, Germany.

Katranuschkov, P. (2001): *A Mapping Language for Concurrent Engineering Processes*. Ph.D. Thesis, TU Dresden, Germany.

Neiling, M. (2004): *Identifizierung von Realwelt-Objekten in multiplen Datenbanken*, (in German), Ph.D. Thesis, BTU Cottbus, Germany.

Robinson, P. (1988): *CIM's Missing Link: Object-Oriented Databases*, Computer Graphics World 10/88, pp. 53-58.

Schroeder, U. (1994): *Inkrementelle, syntaxbasierte Revisions- und Variantenkontrolle mit interaktiver Konfigurationsunterstützung* (in German), Ph.D. Thesis, TH Darmstadt, Germany.

Spaccapietra, S. and Parent, C. (1994): *View Integration: A Step Forward in Solving Structural Conflicts*, IEEE Transactions on Knowledge and Data Engineering 6(2), pp. 258-274.

Spinner, A. (1989): *Ein Lernsystem zur Erzeugung komplexer Kommandos in Programmierumgebungen* (in German), Ph.D. Thesis, Technische Hochschule Darmstadt, Germany.

Turk, Z. (2001): *Phenomenological Foundations of Conceptual Product Modelling in AEC*. International Journal of AI in Engineering, Vol. 15, pages 83-92.

Wagner, U. and Scherer, R. J. (2002): *Bericht über die 1. Phase des Projektes „Ein Kooperationsmodell für die Kontrolle divergierender Planungszustände“* (in German) available from: www.iib.bauing.tu-darmstadt.de/dfg-spp1103/de/zwischenberichte/Dresden.pdf

Weise, M. Katranuschkov, P., and Scherer, R. J. (2003): *Generalised Model Subset Definition Schema*, in: Proceedings of the CIB-W78 Conference 2003 – Information Technology for Construction, Auckland, available from: <https://www.cs.auckland.ac.nz/w78/papers/W78-54.pdf>

Wix, J. and Liebich, T. (2001): *Industry Foundation Classes IFC 2x*, © International Alliance for Interoperability, available from: <http://www.iai-ev.de/spezifikation/IFC2x/index.htm>.