PRODUCT MODEL BASED COLLABORATION

R. J. Scherer
Technische Universität Dresden, Institute of Construction Informatics, Dresden, Germany

ABSTRACT: In collaborative design, we can distinguish between synchronous and asynchronous methods of collaborative working. Both are important and complementary in AEC. We will show that an explicit product model and a multi-layered system, with components linked together by an information logistics system, are the necessary prerequisites to effectively tackle the complexity of the information flow for asynchronous collaborative working. The information flow requires well-grounded model-based coordination to enable transparent and guided discussion and decision-making processes. The corresponding product model management services are identified and discussed. These are (1) model view extraction, (2) mapping to design specific models, (3) backward mapping after the design session, (4) identification of design changes through matching, (5) reintegration of extracted and changed models and (6) merging of the mutual design changes and identification of conflicts.

1 INTRODUCTION

Collaboration can be supported by methods to improve the exchange of information, the communication, or the coordination of two or more persons cooperating in intra- or inter-enterprise teams. Collaboration is goal-oriented and hence the coordination of activities and intentions of all persons involved is the main objective of the supporting actions. This includes also tracking the fulfilment of past activities and the conflicts arising from them. Hence, conflict management is an important issue of collaborative work (Scherer et al 1997).

In the past, collaboration support was mainly focused on the improvement of communication including multi-media representation forms, like video-conferencing, drawings, virtual reality and diagrams and complemented through coordinated formalized common workflows. Until recently, the retrieval of the right information, the set-up and continuous adaptation of the workflow according to the evolving work process and in particular to the work content had to be done by the persons themselves through interpretation of the communicated information. This was not only time-consuming but also led to misinterpretations (i.e. different interpretations due to the different perceptions of the involved persons), and often resulted in bad coordination or even discoordination.

Product models available today provide the advantage to better evaluate the contents of the collaboration issue and thereby deduce various coordination-related supportive means. They provide for representing the information content in a formalized way, which allows the computer (program) to interpret properly the information (avoiding different interpretation by different programs), to reason about it by logic or by algorithmic reasoning methods, and to activate related workflow patterns. Since nowadays product models are well developed and applied in various domains of design but they are in a very early stage with regard to the production process and in a similar, slightly more advanced state in the domain of facilities management (IAI 2006), we will concentrate our further considerations on the design domain.

2 COLLABORATIVE TEAMWORK

In distributed teamwork, we have to distinguish between two main ways of collaborative working, namely(Scherer 2004):

- *synchronous* collaborative teamwork, and
- *asynchronous* collaborative teamwork.

Synchronous collaborative teamwork means that all members of a team are working on the same product at the same time and exchanging their expert knowledge simultaneously for problem solving. This is a relatively rare case in AEC practice. Such collaborative work is mostly employed to search for a new innovative design solution or for the solution of very complex problems. The complexity of a design problem and/or the degree of novelty calls for personal communication, discussion and inspiration among the team members. Communication happens in the “human world”.

Asynchronous collaborative work means that expert knowledge from all team members is necessary and they may work at the same time on the some product component, but it is sufficient that they provide their contributions without direct and immediate communication with other team members. Communication can occur via computer in a formalized way, by exchanging ideas and suggestions such as the inherent knowledge in written, graphical and multi-media representation, or nowadays in product model data form. When communication takes

place asynchronously in the "computer world", it does not necessarily mean that the team members do not inspire each other but that inspiration may happen on a lower level than by synchronous working. Asynchronous working is sufficient for most routine design tasks, which is the bulk of design work carried out in the AEC domain. Usually, the design process in AEC requires that different experts develop their work in parallel but independently, using roundtable meetings at specific discrete points in time to coordinate their work as illustrated in Fig. 2.1 (Scherer 1997 et al, Katranuschkov 2001). Today such round-table meetings are carried out as physical meetings or as video conferences. The latter only works satisfactorily if minor technical problems have to be solved.

Asynchronous collaborative teamwork provides an environment that enables concentrated and efficient work of all team members. It protects team members from permanent communication requests so that every designer concentrates on his/her own specific tasks but at the same time will be informed and keeps track of others' solutions and informs the others about his/her decisions in a timely fashion, individually deciding upon when and how to spend time on keeping track. The shortcoming of asynchronous teamwork is that the coordination between the coordination points is weak. Hence, that coordination has to be done more often and preferably all designers should take part in it. A requirement that often can not be fulfilled on such a quality level.

The main difference between synchronous and asynchronous work is that in the first case communication among team members takes place predominantly in the "human communication world" whereas in the second case it occurs in the "computer communication world", via information and knowledge represented in semantically highly structured data, as provided by product models. Modern computer communication is understood as the representation of information and knowledge in data structures on high semantic level, and not as the application of computers and networks transferring only low-structured text and multi-media data, or even pixel files. The goal is to increase the quality of the computer-communicated information, its expressiveness and its granularity, and hence the retrieval of the actually needed information pieces. Computer communication does not require the experts to communicate with each other at the same time because the information content is now stored and can be retrieved – repeatedly – later on, with any appropriate time-shift by each team member depending on his/her specific needs to share knowledge and data. Therefore, computer communication can be considered a one-sided communication on demand.

However, communication and coordination via computer is only one of the two major aspects that are necessary to make asynchronous collaborative working happen. The second aspect is the management of the time-stretched discussion process. This can be carried out manually, in a very time-consuming way by a person, e.g. the team leader, or with the help of organizational collaboration tools supporting him. Such tools can remarkably reduce the organizational workload on the team leader and at the same time minimise errors that might occur by overlooking or misunderstanding something. They will also reduce the workload on every team member who searches for the information required to make his contributions to the right problems at the right time. Common teamwork or discussion panel tools can be beneficially applied and are already widely used in practice (Schulz 1996, Weisberg 2001). However, the development of specific tools for AEC based upon a common conflict management system as outlined below has the potential to enhance strongly collaboration as such tools can be tailored to the specific culture of the domain.

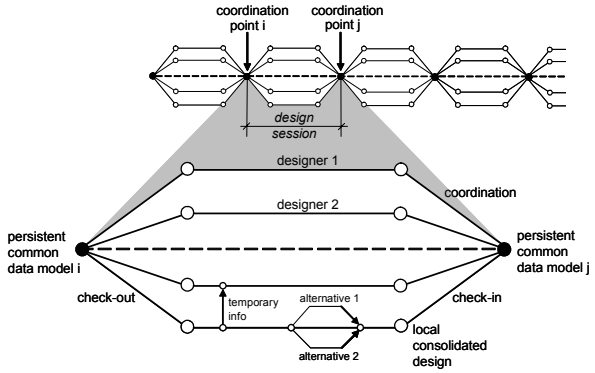

Figure 2.1. Asynchronous collaborative teamwork with mandatory coordination points.

3 THE COORDINATION PROCESS

The collaboration processes can be described by coordination scenarios. Fig. 2.1 shows the standard coordination scenario with common coordination points that are mandatory for all. It is typical for today's roundtable meetings where all accumulated conflicts and open issues are discussed and solved at once, or at least solutions and deadlines are agreed upon. This means that conflicts have to be managed explicitly by the team members – individually, or by a central responsible person, the project manager or a specialised conflict or risk manager. The latter was practised e.g. by the British Airport Authority (BAA) for the design and construction of the fast subway connection from Heathrow Airport to central London as well as for Terminal V of Heathrow Airport (John Gill 1998, 2002). Basically, conflict management is performed with the help of evolving to-do lists, or in a more elaborated form by risk lists (open issues) and contingency plans (solution strategies expressed by a workflow containing the first tasks at least), using paper, spread sheets or specialised data management systems (Fig. 3.1). In all cases, the cognitive work is solely done by humans. However, if the computer were able to understand and reason about the content, valuable parts of this work could be taken over by the computer, namely the definition of problems, the set-up of solution workflows – but not the solution content –, and the management of the solution workflow.

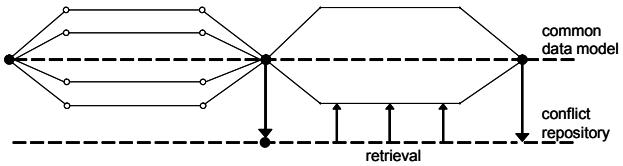

Figure 3.1. Complementing the common data model with a conflict management system.

This demands a workflow system, a conflict repository and a conflict management system. With these three IT components, we would be able to disband the constraints of common mandatory coordination points, substituting them by more flexible coordination lines aligned with the evolving model data (Fig. 3.2), and the product model will become the basis of a real building information management system. In such a system, coordination will be individualized. Each team member can decide upon when and what he wants to coordinate with the help of the system and hence, via the system, with all other team members over time-stretched coordination lines.

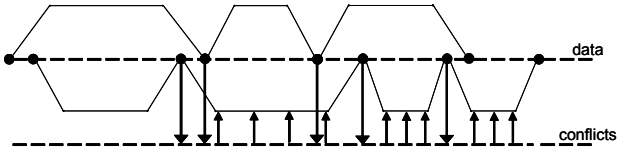

Figure 3.2. Disbanding common to individual co-ordination points thanks to the conflict management system.

Going more into detail and taking into account more serious and complex conflict problems that may be not solvable solely by formalized information exchange, we will finally come up with the coordination scenario shown in Fig. 3.3 where mandatory coordination points still exist but with considerably longer time spans between them in comparison to the traditional scenario from Fig. 3.1. They are introduced to consolidate the common data model and in addition to benefit from eye-to-eye communication, i.e. synchronous collaboration on demand. Between these coordination points, non eye-to-eye collaboration procedures of asynchronous working are applied to benefit from the individualized coordination process, where scenarios for individual work and close teamwork are shown.

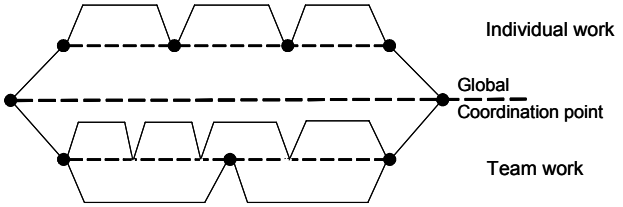

Figure 3.3. Hybrid local and global co-ordination points according to the kind of working (conflict system avoided for convenience).

4 COORDINATION SUPPORT

From the scenarios above (Fig. 3.1 – Fig. 3.3) the requirements on the product model and the coordination supporting IT services can be drawn. The first supporting service would be to provide the designer with the information of all his design modifications he has to inform other designers about, and to classify them into (1) problematic modifications and (2) normal modifications that have just to be approved. Both these are conflicts, but the first ones are identified as expected conflicts, whereas the second are only potential conflicts. They have to be recorded in a checklist or, as suggested above, in a conflict repository. Until today this has to be done by hand, which is a highly time-consuming task. Therefore, the designer usually prefers to draw all his modifications in a separate file, which is then sent either to all other designers for approval or to the design team coordinator, typically the architect, who does the approval on behalf of the other designers and only requests additional approvals from them on demand.

This task may be taken over by one or several computer services. These services must first find out all modification done in a design session. Secondly, they have to classify the design modifications into three classes, namely the two above-mentioned classes “for approval”, and an “approval free” class. The latter comprises design modifications that are only in the responsibility of the designer himself. This is the case for instance by the design of the reinforcement in a RC element. A further support would be, if not all modifications for approval have to be sent to everybody, but are classified according to the designers who have to approve the modifications. This would be also very helpful in the case when a design coordinator is doing the approvals on behalf of the other designers, in order to reduce error or avoid overlooking something. Precise checklists can be very helpful here as well, reducing errors to zero and reducing time for the check.

In cases when more than one designer has to approve a design modification, a workflow must be set up. However, even in the case of one designer there is already a workflow, namely with the one worktask “approval”. When a real conflict occurs, i.e. when the approval is not positive and the design modification is rejected, a conflict procedure has to be set up. This is a workflow containing at least one worktask for the original designer with the request of a design change, complemented with a change proposal. It is conceivable that several cycles of proposals and approvals may follow that cannot be all defined in advance, but are evolving, i.e. we do have an evolving workflow. It is also a recursive workflow that has to be monitored by the design coordinator to ensure that it comes to an end. Each design modification to be approved by another designer is a potential design conflict, and due to its recursive, non-foreseeable duration, it is a potential risk for the design process as well. Therefore, it is reasonable to handle serious design conflicts as design risks. This kind of working has been proven by BAA for the fast subway connection (Gill 1998) and the Terminal V of the Heathrow Airport (Gill 2002). Such risks may influence the overall workflow seriously. Therefore, the monitoring of design risks and the optimal merging of the related consecutive workflows in the overall design process have to be supported by appropriate risk management methods as we develop in an ongoing research project (Sharmak et al. 2007).

In the last years, we developed a procedure that does not require a separate approval file but merges the modifications directly in the common product model (Weise 2006). This strategy was selected in order to (1) circumvent locking of parts of the model during a check-out, i.e. during a so-called long transaction, and (2) avoid deadlocks due to not having a valid product model or diverging evolving conflicts. Thereby it is important that there exists only one common model, which is binding for all designers and which is complemented by a separate conflict repository that contains the above-defined conflicts resulting from parallel working and design alternatives, here also formulated as conflicts. Thus, each designer has

a sound ground and can individually inform himself about open design issues, i.e. open conflicts and design alternatives, stored in the conflict repository.

However, a consistent common data model cannot be achieved by simply checking-in the modified shared model data that was checked out and modified by the designer for and during his design session. At first changes of the data has to be classified in (1) design modifications and (2) alterations done by the design tools. The latter has to be taken into account because it would not be possible to implement totally error-free product model interfaces. An interface is not only the technical implementation of the product model classes but contains also a model mapping from the external (common) data model to the internal one. Even if external and internal models are based on the same data schema, they may show different versions. Therefore, the service has to identify design tool alterations and correct them. These automatic corrections have to be shown to the designer for approval (Fig. 4.1, top level).

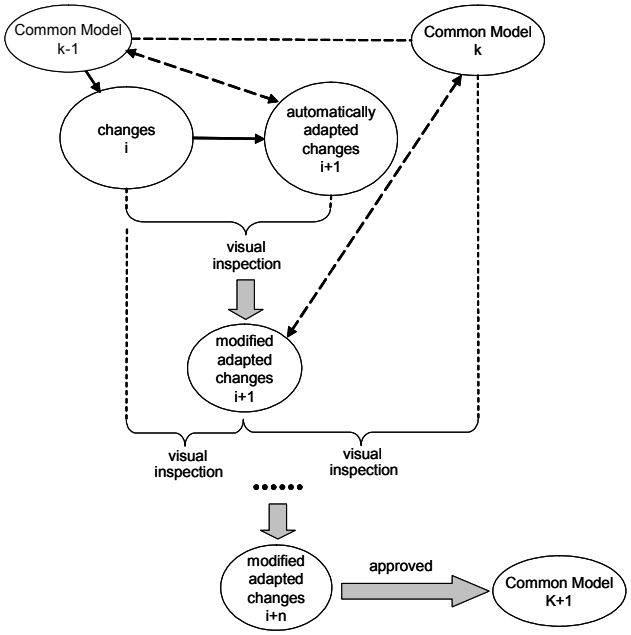

Figure 4.1. Visual inspection cycles in the context of model merging.

The approved design modifications are merged in the common data model in the next step. In order to achieve consistency, several modifications are necessary (1) due to the propagation of design modification into the common model, and (2) due to the fact that the common model was modified several times by other designers since the check-out, as indicated in Fig. 3.2. All automatically generated modifications done to achieve consistency and all open conflicts where a unique solution can be generated have to be presented to the designer for approval, or for finding an alternative solution. This can lead to approval cycles as indicated by the lower level on Fig. 4.1, before the modified data set is merged in the common product model and the remaining conflicts are stored in the conflict repository.

5 PRODUCT MODEL MANAGEMENT METHODS

Product data management has to provide several important functions for cooperative work as discussed in the previous chapter. Their availability and quality is essential for the value of collaborative working because they should warrant the consistency of the data for the typical long transactions in AEC without data locks. Thus, we have to deal with check-out/check-in cycles where the same data can be modified asynchronously and in parallel, changes have to be properly tracked and managed, and users have to be notified in accordance with their roles. Such functions must also be appropriately synchronised with other information management services, e.g. organised around a dedicated project portal. We argue that they may be provided by one or more third-party vendors as web services and may not be tightly integrated with any particular Model Server. Thus, instead of developing a new mode-based project environment we aim to provide these functions as a set of basic product data management services.

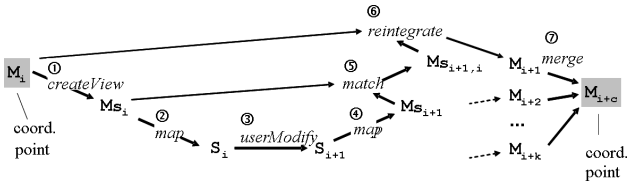

Figure 5.1. Generalised cooperation scenario to support long transactions.

Each design session, starting with a check-out activity and ending with a check-in activity comprises up to 6 functions including the coordination step with the other designers (Scherer et al 1997b). They correspond to the 7 activities of the generalized cooperation scenario shown on Fig. 5.1, except for activity 3, which is the design modification carried out by the designer with his specialised design tool.

1. *Model view extraction*:
   Ms_i = extractView (M_i , viewDef (M_i))
2. *Mapping* of the model view Ms_i to the discipline-specific, in most cases proprietary model S_i, which is an instantiation of the data model S:
   S_i = map (Ms_i , mappingDef (M, S))
3. *Modification by the user* of S_i to S_{i+1} via some legacy application, which can be expressed abstractly as:
   S_{i+1} = userModify (S_i , useApplication (A , S_i))
4. *Backward mapping* of S_{i+1} to M_{si+1}, i.e.:
   Ms_{i+1} = map (S_{i+1}, mappingDef (S , M))
5. *Matching* of Msi and Msi+1 to find the differences:
   $\Delta Ms_{i+1,i}$ = match (Ms_i , Ms_{i+1})
6. *Reintegration* of $\Delta Ms_{i+1,i}$ into the model:
   M_{i+1} = reintegrate (M_i , $\Delta Ms_{i+1,i}$)
7. *Merging* of the final consistent model M_{i+1} with the data of other users that may concurrently have changed the model, to obtain a new stable model state, i.e.
   M_{i+c} = merge (M_{i+1}, M_{i+2}, … , M_{i+k}),
   with k = the number of concurrently changed checked out models.

The essence of these functions is shortly explained in the following sections. They can be offered by a product data

management system or as dedicated services, which can be flexibly orchestrated and integrated into existing collaboration systems such as today's Web-based project environments.

5.1 *Model view extraction*

Model View Extraction is the first step in the generalized collaboration scenario shown on Fig. 5.1. To be usefully applied, a model view should (1) be easily definable with as few as possible formal statements, (2) allow for adequate (run-time) flexibility on instance and attribute level and (3) provide adequate constructs for subsequent reintegration of the data into the originating model. To meet these requirements, a Generalised Model Subset Definition Schema (GMSD) has been developed (Weise et. al. 2003). It is a neutral definition format for EXPRESS-based models comprised of two subparts, which are almost independent of each other with regard to the data but are strongly inter-related in the overall process. These two parts are: (1) object selection, and (2) view definition. The first is purely focused on the selection of object instances, such as all objects belonging to storey 2 using set theory as baseline. The second is intended for post-processing (filtering, projection, folding) of the selected data in accordance with the specific partial model view, such as only class column and only attribute height. Fig. 5.2 shows the top-level entities of the GMSD schema and illustrates the envisaged method of its use in a run-time model server environment. More details on GMSD are provided in (Weise et al. 2003) and (Weise 2006), along with references to other related efforts such as the PMQL language developed by Adachi (Adachi 2002).

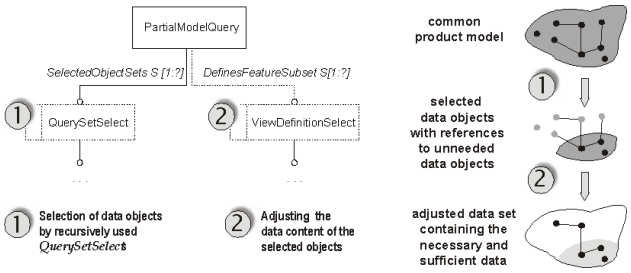

Figure 5.2. Top-level entities of the GMSD schema (left) and its principal use in a model server environment (right).

5.2 *Model mapping*

Model Mapping is needed by the transformation of the data from one model schema to another. Typically, this would happen in the transition from/to an agreed shared model or model view to/from the proprietary model of the application the user works with (Figure 5.1). The overall mapping process consists of four 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.

Mapping patterns allow understanding the mapping task better and to formalize what and how has to be mapped in each particular case. By examining the theoretical background of object-oriented modelling the following types of mapping patterns can be identified: (1) Unconditional class level mapping patterns, depicting the most general high-level mappings, (2) Conditional instance level mapping patterns, including logical conditions to select the set of instances to map from the full set of instances in the source model, and (3) Attribute level mapping patterns, specifying how an attribute with a given data type should be mapped. For each of these categories, several subcases have been identified in (Katranuschkov 2001). Examples on attribute level include simple equivalence, set equivalence, functional equivalence, transitive mapping, inverse transitive mapping, functional generative mapping, and so on.

All mapping patterns can be defined by means of the developed formal mapping language CSML (Katranuschkov 2001) or by another suitable specification language such as VML (Amor 1997). Performing the mappings is typically done with the help of a Mapping Engine. In general, this is a difficult task but there exist several known support tools that can be drawn up to tackle the issue (Katranuschkov 2001, Weise et al. 2004).

5.3 *User modifications on the model view*

User modifications on the model view can hardly be accomplished as reusable generic services. They represent an essential part of the actual value-adding work of each designer in the development and iterative detailing of the design solution. With common software tools the user works on his sub model and makes changes in that model in accordance to his field of activity, e.g. on the architecture or the HVAC systems of the designed building. This work often leads to inconsistencies between the separate used model views, especially if two or more users make parallel changes to the design.

Whilst this subtask of the overall cooperation process cannot be supported by any generic product data management methods, it can be well aligned in the scenario and consequently its integration with the other services can be generalised within a common framework.

5.4 *Backward mapping*

Backward mapping is needed to translate the application data representing the changed design state back into the data schema of the shared model in order to create a new model view that can then be matched with its initial version created in the first step of the cooperative work process, i.e. Model View extraction (Fig. 5.1).

In principle, there is no difference between this step and the forward mapping outlined in section 5.2 above. Some mapping languages as e.g. CSML require for that step a separate mapping specification whereas others, such as VML, claim to be capable of directly specifying bi-directional mapping. However, in both cases the mapping process is basically the same. It requires the same mapping tools and has the same position in the overall scenario. Moreover, the same mapping service can be used for both the forward and the backward mapping tasks.

If not an explicit mapping to/from the specific design model is made but an implicit one, encapsulated in the design tool, a model normalization has to be carried out (Weise 2006) in order to flatten the alteration of the de-

sign tool and to prepare the shared data model for matching.

5.5 *Model matching*

Model matching has to deal with the identification of the changes made by the user in his design session with the help of one or more design tools (Fig 5.1). This may be done by a dedicated client application but a much more natural implementation is a server-side procedure.

In our approach, matching exploits the object structure without considering its semantic meaning (Weise et al. 2004). Hence, it can be applied to different data models and different engineering tasks. It does not require nor involve specific engineering knowledge.

Comparison of the model data of the old and new model versions begins with the identification of pairs of potentially matching objects, established by using their unique identifiers or some other key value. However, identifiers may not always be available for all objects due to shortcomings in the data model, e.g. most of the resource objects in IFC do not have identifiers or the identifiers are not properly implemented in IFC interfaces. Such unidentifiable objects may be shared via references from different identifiable objects. The general complexity of this problem is shown in (Spinner 1989) where a fully generic tree-matching algorithm is shown to be NP-complete. Therefore, we have developed a pragmatic algorithm that provides a simple scalable way for finding matching data objects. Its essence is in the iterative generation of corresponding object pairs by evaluating equivalent references of already validated object pairs. The first set of valid object pairs is built by unambiguously definable object pairs. Any new found object pair is then validated in a following iteration cycle, depending on its weighting factor derived from the type of the reference responsible for its creation. Attribute values are only used if ambiguities of aggregated references do not allow the generation of new object pairs. To avoid costly evaluation of attribute values a hash code is used indicating identical references. In this way, the pairwise comparison of objects can be significantly reduced.

Fig. 5.3 illustrates the outlined procedure. Before starting any comparison of objects the set VP of validated object pairs and the set UO of unidentifiable objects are initially created using available unique identifiers. After that, the object pairs of VP are compared as shown on the right side of the figure for {A1, A2}. Using their equivalent references *has_material,* a new object pair can be assumed for the objects E1 and G2 of UO.

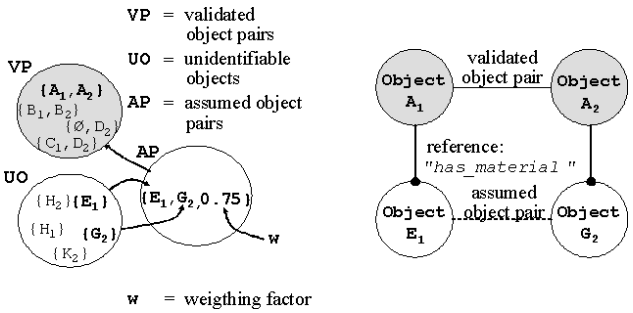

Figure 5.3. Schematic presentation of the model mapping process.

A weighting factor indicating the quality of this assumption is then applied according to 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 derived matching pairs. After comparison of all object pairs of VP, the highest weighted object pairs of AP are moved from AP (and UO) to VP and a new iteration cycle is started. Now the weighting factors of the new created object pairs can be combined with the weighting factors of already validated object pairs from which they have been deduced, thereby allowing for a global ordering. More details on the developed algorithm along with an overview of related efforts are provided in (Weise 2006).

5.6 *Model re-integration*

Reintegration of the changed model data is always necessary when model views are used, as shown in the generalized collaboration scenario in Fig. 5.4. Model view extraction creates a model view by removing data objects, cutting off or reducing references, filtering attributes etc. Reintegration means to invert the process, i.e. to add in a consistent way removed data objects, restore cut-off or reduced references and re-create filtered out attributes considering all related design changes. In other words, the design changes altering cut-off parts of the product model have to be propagated into the common model. This demands automatic adaption and propagation of design changes, which have to be approved by the designer (Fig. 4.1) This is strongly dependent on the model view definition achieved via GMSD, the results of the model comparison and the adopted version management.

Figure 5.4 illustrates the reintegration process on an example, assuming that by applying a GMSD-based *extractView( )* operation to a given model some objects and attributes will be removed. For object O1 this results in a new version OS1 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 user application to OS2 which differs from OS1 in the aggregated reference ‘b’, downsized by another element, and the simple references ‘c’ and ‘d’. The reintegration process adds all objects and attributes from O1 that have been removed according to the model view 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.

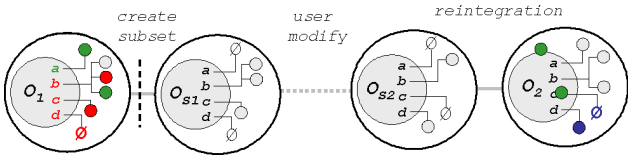

Figure 5.4. Example of the re-integration process.

Whilst this procedure is quite clear and more or less the same for various different scenarios, there exist several detailed problems that have to be dealt with. They can be sub-divided into: (1) structural problems (1:m version relationships for reference attributes, n:1 version relationships, change of the object type in a version relationship), and (2) semantic problems. The latter cannot be resolved

solely by generic server-side procedures but require domain knowledge and respective user interaction. They occur typically when a change to a model view requires propagation of changes to another part of the overall model in order to restore consistency. In such cases, data consistency must be evaluated by all involved actors during a final merging process (Weise et al 2004; Weise 2006).

5.7 *Model merging*

Model Merging has to deal with the consistency of concurrently changed data that exist in two or more shared models. It should be performed at a commonly agreed coordination point in cooperation of all involved users in order to reduce complexity. The aim is to provide a procedure by which modifications can be reconciled and appropriately adjusted to a consistent new model state, marking the beginning of a new collaboration cycle.

The method developed (Weise 2006) is based on a commonly agreed data model M_i as shown in Fig. 5.3. All changes carried out by the participating designers are represented as delta values. Conflicts on attribute, object and object topology levels are resolved in an automatic way based upon predefined rules, such as (1) an object is deleted if it was deleted by all designers, (2) an object is deleted if it was deleted by one designer and no other designer changed it in any way, (3) an object that was changed by only one designer and not modified by any other designer is integrated with these changes, (4) if an object is changed by two designers, all the attributes changed are integrated.

It is obvious that a commonly acceptable data model cannot be achieved automatically. Hence alterations that are not in line with a designer's intention hold potential conflicts and need inspection cycles as outlined in Fig. 4.1. Remaining conflicts have to be stored in the conflict repository and resolved later on in collaborative workflows. Not all designers need to be involved for all conflicts but based upon resolution workflows, their authorization and the modifications they have carried out, the conflict set can be portioned into sub-sets of mutually involved 2, 3 and so on designers. Starting phase of these evolving workflows can be automatically set up and propagated to the individual work lists of each designer. In the scenario sketched in Fig. 5.3, designer A and B had carried out currently design modification. However due to authorization demands designer T has to be involved in the coordination process, here solved through a pre-merging step.

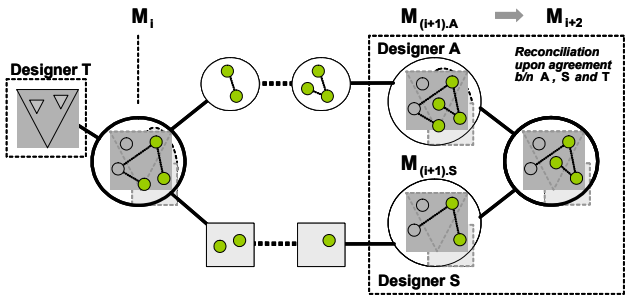

Figure 5.5. Schematic presentation of the model merging process.

6 ARCHITECTURE OF A COLLABORATION SYSTEM

In the previous chapter, we identified various basic methods that are needed for efficient product model based collaboration. These methods can be best implemented in the form of *software services*, exploiting the flexibility, adaptability and extensibility of the modern Service-Oriented Architecture (SOA) approach, in the spirit of a model-based IT environment (Carter 2007). Appropriately orchestrated, the services can be used for a variety of complex AEC tasks, especially in conjunction with the IDM approach of using Building Information Models (Wix 2005).

Various platform configurations can be envisaged for the realisation of such orchestration and coherent service use – from low-level "classical" client-server architectures (Scherer 1995) with their typical bottleneck problems to dedicated multi-server systems (Scherer 1998), project portals (Scherer 2000, Katranuschkov 2001), P2P systems, and modern Grid-based architectures at the high end. However, due to the nature of construction projects, typically performed by ad hoc created virtual organisations, the Grid-based approach is considered the one providing greatest benefits (Turk et al. 2006). Fig. 6.1 below shows the suggested principal architecture of a grid-based service-oriented platform. It is comprised of four layers to which an external client application layer is "plugged in" at the front end, and an external data storage layer is interlinked at the back end.

In principle, it should be possible to plug in any kind of client applications to the platform using one and the same approach. However, greatest advantages and most consistent integrated use of the collaboration services can be achieved by using a Workflow or Process Management Client that provides for direct goal-oriented support of the discussed collaboration scenarios. The principal functionality of such a client is outlined in the next chapter.

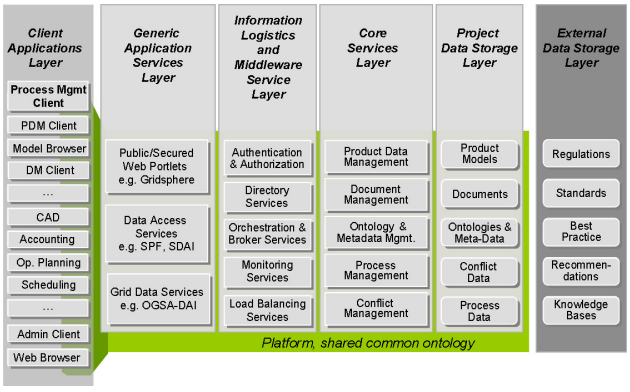

Figure 6.1. Principal System Architecture.

"Plugging-in" of all external applications to the platform is realised with the help of the services on the Generic Application Services Layer, which act as proxies to the external applications, thereby enabling their interoperability with the "deeper" platform services via high-level WSDL-based interfaces.

The Middleware Service Layer ensures the basic operability of the platform by a set of services ensuring the adequate use of the underlying Grid technology, based e.g.

on Globus or Unicore. Via these services, the connection of the client applications to the layer of (distributed) core semantic services of the platform can be established and efficiently managed. These Core Services include in particular the product model management methods described in Chapter 5, but can also be extended by dedicated document management, process management and especially conflict management services (Gehre et al 2007). They are responsible for the proper management, interpretation and processing of the shared project data on the Project Data Storage Layer, as well as for their external links to further data sources such as regulations, best practice cases and templates, local organisational knowledge bases etc.

This stepwise delegation of tasks between the service layers considerably reduces the complexity of the overall platform and improves its manageability, maintainability and extensibility. However, this alone is not sufficient for achievement of adequate semantic interoperability as required by any more sophisticated data management methods. For that purpose, a high level, environment wide process-centred ontology framework encompassing all entities of the IT system is suggested. It can provide the necessary semantic proxies of the entities in the "real" AEC environment, which allows to capture semantic meta-data about all "things" in the environment in a coherent way, and appropriately delegate detailed processing tasks to the specialised collaboration services using high-level concepts defined e.g. in the OWL language and communicated via WSDL interfaces, SPARQL queries or other suitable methods (Gehre et al. 2007).

A successful prototype of such process-centred environment ontology has been realised recently in the EU project InteliGrid (Gehre et al. 2006, Gehre et al. 2007).

7 WORKFLOW MANAGEMENT SYSTEM

For an efficient error-prone quality-controlled design process, the work of the members of a distributed virtual team must be organized in terms of worktasks (Wasserfuhr & Scherer 1997). Worktasks are globally identifiable (like other objects of the environment) and linked to actor roles (e.g. architect, structural engineer, etc.), to the required input (documents, product data), to the expected or delivered output (documents/views of the product model/single objects of the product model), and to the time schedule of a project.

Tasks are grouped into different levels as shown in Fig. 7.1 starting from project level and ending on the Internet level:

- *Services* are typically web services.
- *Activities* include one or more services. They are carried out by one person with one role. They can be modelled as business process objects (Gehre 2007).
- *Worktasks* consist of one or more activities. They are carried out by one or more persons owning different roles.
- *Workflows* include one or more worktasks. Several workflows of the same type may be performed in one project. The workflows themselves can be derived from process patterns defined by means of a process ontology and then respectively adapted and configured e.g. by the project manager. They may be applicable for the whole project (cross-company) or company specific (Katranuschkov et al. 2006).
- *Process Patterns* consist of one or more workflows that are company-specific instantiations of process modules but are generic from the company's point of view.
- *Process Modules* are company-independent descriptions of work and consist of one or more process patterns (Keller 2007).
- *Process Module Chains* consist of one or more process patterns and are project-specific.
- *Service packages* consist of one or more process module chains and structure the project process as value chains.

During the overall work process, the process management system continuously updates the work lists for the different users, which contain exactly those worktasks that are relevant for that user.

The user indicates that he wants to start the execution of a task by activating the corresponding worktask in his work list. Then the system provides him with a list of all relevant documents , shared data models and the corresponding tools, from which he can select a document, a shared data model or an appropriate tool – e.g. CAD, a structural analysis tool, or an office application. When the user finishes a worktask, he assigns the results to the process management tool, which then updates the status of all possible follow-up worktasks for all other users.

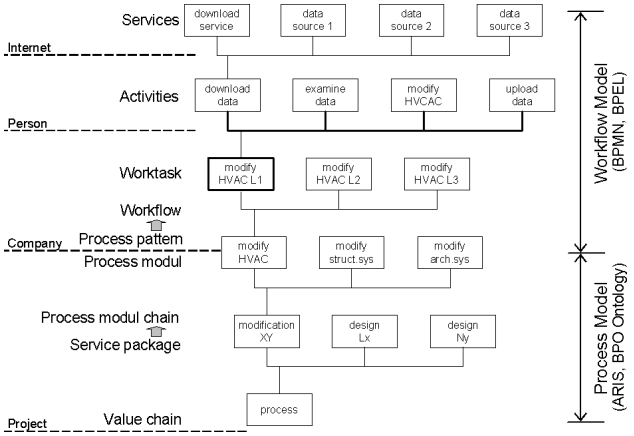

Figure 7.1. Hierarchical sub-structuring of value chains down the web services.

Each worktask can change its state during its life cycle. Initially, a task is either suspended or ready for execution. A task is suspended if it requires additional data to be executable, e.g. the calculation of loads may be suspended because data about the building geometry and the location of building elements is missing. As soon as all required input data is available, a task is marked as *readyForExecution*. If the user actually starts it, the internal operation *fetchForUnify( )* is performed, ensuring exclusive access to this task and changing its state to *inExecution*. Then, the user may switch the state between *inExecution* and *interrupted*, as often as he needs to.

The workflow system can be configured to check whether a user performs tasks simultaneously, and to provide a notice, if this happens. When a task is finished, the results

are linked to that task and the state becomes *finished*. All tasks, which are not finished, can be *aborted*. Abortion can also be performed for the whole workflow that includes the task.

For a dynamic set-up of workflows, activities, and worktasks as well as their refinement on demand during runtime, a tool named Process Wizard was designed to support project managers in the coordination of actors (Wasserfuhr & Scherer 1997). A process definition methodology was later developed to achieve a parametric description of worktask patterns, based on workflow templates as described above. Fig. 7.2 shows a screenshot of an example session with the Process Wizard. Each task of a user role is modelled as a worktask (a node in the process network) and the dependencies are represented as arrows. The main window (1) of the Process Wizard shows the worktasks. By selecting a worktask, the properties of that worktask can be modified in a separate window (2). For each worktask, the users and roles can be specified by selecting them from overview lists that are interrelated according to a defined actor matrix (3).

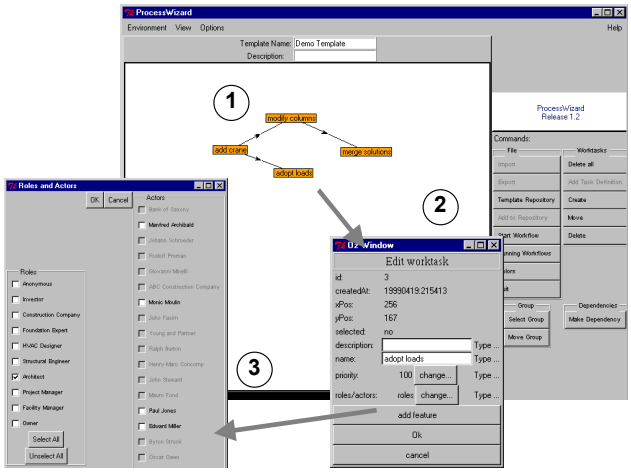

Figure 7.2. Process Management Tool taken from the ToCEE Environment.

8 CONCLUSIONS

More than 15 years of research on collaborative engineering have clearly revealed that a high human-machine interaction in the sharing of work is important to tackle the complexity. The machine has to take over cognitive parts of the co-ordination and collaboration work but still has to remain in the position of an assistant. Many decision supports, if not ultimately all may be taken over by the machine and decision-making can be prepared but ultimately it has to be approved by the end-user to whom the support is finally directed. A high level of formalization of the co-ordination and the content of co-ordination are necessary in order to provide the machine with generic methods and services, applicable and adaptable to the various design situations. Only a high-semantic presentation level, on which all the achieved results were built in the past, has led to an ever-growing product and process model, nowadays called building information model in the scope of buildingsmart (IAI 2006), which is sometimes hard to manage as the STEP model has already revealed. The strict modularization of the IFC model, the smart version management with a separate platform and extended model versioning like 2x.3, i.e. platform version 2, extended model, version 3 may considerably help postpone this problem further to the future. However, descriptive interoperability as propagated by the IAI is limited. The reported methods are based upon functional interoperability and underpin the big advantages received. Still more flexibility is needed and the ongoing research in applying ontologies for data as well as service interoperability have proved that ontologies are a promising methodology to widen our scope in particular to more independent and distributed data models as well as distributed data sources and distributed functionality in form of web services, Grid and P2P technologies. However logic reasoning ─ attractive as this may be at the first glance ─ is strongly limited to the necessary computer power and the state of the art of algorithms. Hybrid ontologies with a strong part of semantically described knowledge, pre-evaluated in patterns, may be one of the pragmatic answers to that topic. Our recent research results in the scope of the EU project InteliGrid (Gehre 2006) may be considered a proof of the concept.

ACKNOWLEDGMENTS

The research work was mainly carried out together with co-workers: Alexander Gehre, Markus Hauser, Martin Keller, Karsten Menzel, Rainer Wasserfuhr, Matthias Weise, and in particular with my senior researcher Peter Katranuschkov, who has shown a never-ending enthusiasm. Their efforts are highly acknowledged. The research projects were funded by the European Commission, the German ministry of research (BMBF) and the German research foundation (DFG) through several contracts, which is very much appreciated.

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. (http://cic.vtt.fi/projects/ifcsvr/tec/VTT-TEC-ADA-12.pdf)

Amor R. (1997). *A Generalised Framework for the Design and Construction of Integrated Design Systems.* Ph. D. Thesis, University of Auckland, New Zealand, 350 p.

Carter S. (2007). *The New Language of Business: SOA and Web 2.0.* Prentice-Hall, ISBN 0-13195-654-X, 320 p.

Gehre A., Katranuschkov P., Wix J. & Beetz J. (2006). *InteliGrid Deliverable D31: Ontology Specification.* The InteliGrid Consortium, c/o Univ. Ljubljana, Slovenia.

Gehre A., Katranuschkov P. & Scherer R.J. (2007). *Managing Virtual Organization Processes by Semantic Web Ontologies*. To appear in: Proc. of the 24th CIB-W78 Conference, 26-29 June 2007, Maribor, Slovenia.

Gill J. (1998). *Risk Management for fast subway connections Heathrow Airport to central London* (personal communication).

Gill J. (2002). *Risk Management for Heathrow Airport Terminal V* (personal communication).

IAI (2006). *IFC2x Edition 3 Online documentation.* © International Alliance for Interoperability, 1996-2006.

(http://www.iai-international.org/Model/R2x3_final/index.htm)

Katranuschkov P. (2001). *A Mapping Language for Concurrent Engineering Processes*, Doctoral Thesis, Institute of construction Informatics, Report 1 (Ed. R. J. Scherer), Technische Universität Dresden, Germany, ISBN 3-86005-280-2, 385 p.

Katranuschkov P., Scherer R.J., Turk, Z. (2001). *Intelligent Services and Tools for Concurrent Engineering - An Approach towards the Next Generation of Collaboration Platforms*, e-journal ITcon, www.itcon.org.

Katranuschkov P. Gehre A., Keller M., Schapke S.-E. & Scherer R.J. (2006). Ontology-Based Reusable Process Patterns for Collaborative Work Environments in the Consutrction Industry. In: Cunnigham P. & Cunningham M. (eds.) "*Exploiting the Knowledge Economy*", IOS Press, ISBN 1-58603-682-3.

Keller M. (2007). *Informationstechnisch unterstützte Kooperation bei Bauprojekten* (Information-technology supported co-operation in construction project), Doctoral Thesis, Institute of construction Informatics, Report 5 (Ed. R. J. Scherer), Technische Universität Dresden, Germany.

Scherer R.J. (1995). *COMBI ─ Overview and Objectives*, Proceedings of the First European Conference on Product and Process Modelling in the Building Industry, Oktober 1994, Dresden, S. 503-510, R. Scherer (ed.), Balkema, Rotterdam.

Scherer R.J., Wasserfuhr R., Katranuschkov P., Hamann D., Amor R., Hannus M. & Turk Z. (1997a). A Concurrent Engineering IT Environment for the Building Construction Industry. In: Fichtner D. & MacKay R. (eds.) "*Facilitating Deployment of Information and Communication Technologies for Competitive Manufacturing*", Proc. IiM'97, 24-26 Sept. 1997, Dresden, Germany, ISBN 3-86005-192-X.

Scherer R.J., Katranuschkov P.(1997b). *Framework for Interoperability of Building Product Models in Collaborative Work Environments*, Proc. of the 8th International Conference on Civil and Building Engineering, pp. 627 - 632, C.-K. Choi, C.-B. Yun, H.-G. Kwak (eds.), Seoul, Korea.

Scherer, R. J. (1998). *A Framework for the Concurrent Engineering Environment*. in: Amor R. (ed) „Product and Process Modelling in the Building Industry", Proc. of the 2nd European Conf. on Product and Process Modelling in the Building Industry ECPPM'98, Building Research Establishment, Watford, 19-21 Oct. 1998, Clowes Group, Beccles, Suffolk, UK, 1998

Scherer R. J. (2000). *Towards a Personalized Concurrent Engineering Internet Services Platform*, in: Goncalves R., Steiger-Garcao A. & Scherer R. (eds.) "Product and Process Modelling in Building and Construction", Proc. of the 3rd ECPPM 2000, Lisbon, Portugal, 25-27 Sept. 2000, A.A. Balkema, Rotterdam, ISBN 90 5809 179 1, pp. 91-96.

Scherer R.J. (2004). *Information Logistics for Supporting the Collaborative Design Process*, Chapter 7 in: Bento, J.; Duarte, J.P.; Heitor, V. M.; Mitchell, W.J.; (eds.) Remote Collaborative Design, Praeger Publishers, New York, U.S.A.

Schulz R. C. (1996). *Computer Mediated Communications in Architecture, Engineering and Construction*, Res. Report, Dept. Civil Eng., Cal. State Univ., CA.

Sharmak W., Scherer R.J. & Katranuschkov P. (2007). *Configurable Knowledge-Based Risk Management Process Model within the General Construction Project Process Model*. To appear in: Proc. of the 24th CIB-W78 Conference, 26-29 June 2007, Maribor, Slovenia.

Spinner A. (1989). Ein Lernsystem zur Erzeugung komplexer Kommandos in Programmierumgebungen (A learning system for generating complex commands in program environments). Ph.D. Thesis, (in German), Technische Hochschule Darmstadt, Germany.

Turk Z., Dolenc M., Gehre A., Katranuschkov P., Klinc R., Kurowski K. & Scherer R.J. (2006). A Generic Architectural Framework for CIC. Invited paper at the "*World Conference on IT in Design and Construction"*, November 15-17 2006, New Delhi, India.

Wasserfuhr R. & Scherer R.J. (1997). Information Management in the Concurrent Design Process. In: Proc. Int. Colloquim IKM'97, Weimar, Germany.

Weisberg S. (2001). *i-Collaboration ─ State of the Industry*, CADALYST Magazine 9/2001 (last visited at: http://209.208.199.147:85//features/0901icollab/0901icolab.htm).

Weise M., Katranuschkov P. & Scherer R. J. (2003). Generalised Model Subset Definition Schema. In: Amor R. (ed.) "*Construction IT: Bridging the Distance*", Proc. of the CIB-W78 Workshop, 23-25 April 2003, Waiheke Island, Auckland, New Zealand, ISBN 0-908689-71-3. CIB Publication 284.

Weise M., Katranuschkov P. & Scherer R. J. (2004). Generic Services for the Support of Evolving Building Model Dat*a*, in: Beucke K. et al. (eds.) *Proc. of the Xth Int. Conf. on Computing in Civil and Building Engineering (ICCCBE-X)*, Weimar, Germany, June 02-04, ISBN 3-86068-213-X.

Weise M. (2006). *Ein Ansatz zur Abbildung von Änderungen in der model-basierten Objektplanung.* (An approach for the representation of changes in model-based design) Ph. D. Thesis (in German). Doctoral Thesis, Institute of construction Informatics, Report 4 (Ed. R. J. Scherer), Technische Universität Dresden, Germany, ISBN 3-86005-557-7, 213 p.

Wix J. (ed), 2005. *Information Delivery Manual: Using IFC to Build Smart*, (www.iai.no/idm/learningpackage/idm_home.htm)