Abstract and concrete scenarios in concurrent engineering

Dr.-Ing. Michael Huhn, Bergische Universität Wuppertal, 42285 Wuppertal, Germany
(huhn@uni-wuppertal.de)

Summary
An architecture of a distributed planning system for the building industry has been developed. The emphasis is on highly collaborative environments in steelwork, timber construction etc. where designers concurrently handle 3D models. The overall system connects local design systems by the so-called Design Framework DFW. This framework consists of the definition of distributed components and protocols which make the collaborative design work. The process of collaborative design has been formalized on an abstract level. This paper describes how this has been done. A sample is given to illustrate the mapping of concrete scenarios of the 'real design world' to an abstract scenario level. This work is funded by the Deutsche Forschungsgemeinschaft DFG as part of the project SPP1103 (Meißner et al. 2003).

1 Introduction

1.1 The need for parallel design
At least in the structural steelwork area, classical boundaries of the design chain are melting. Several engineers are working on the same structure in parallel (Claus and Kazakov 2003). Some structures of a building are already pre-fabricated while other areas are still modeled or drafted. Parts of the structures are already released, others are changed again.

The engineer faces a new challenge. Beside the regular design, he has to coordinate his own work with the overall design **permanently**. Usual approaches like 'export & re-import' fail as the merge happens too late. A planning environment must be able to support transactions at the level of a single design step like the creation of a joint. On the other hand, the designer must be relieved of the burden of 'manually' handling the model complexity and distribution. This means, for instance, that the designer should **not** be forced
- to 'check out' model parts in advance,
- to 'manually' synchronize redundant, distributed model parts,
- to inform other designers of every modification,
- to ask for changes by others etc.

A planning environment should perform these tasks as global as possible. Administrative tasks have to be extracted, centralized and compressed to a minimum. Nevertheless, the designer knows about the model distribution, he still must manage a number of matters. The question is: What is it he/she still has to deal with ?

1.2 The architecture of a planning environment
As part of this research project, a number of approaches of model distribution has been evaluated (Pegels and Huhn 2004). Criterias have been the expected network traffic, location and bottleneck effects of object model mappings, expected location and sequence of synchronisation, support of online notification, support of offline design etc. As result, a target architecture of a planning environment has been developed, decomposed and mapped onto Internet technologies, see figure 1. Basically, it is a peer-to-peer architecture of design systems plus central mangement components. Each design system may use it's own native object model, but needs a wrapper to a common framework. At the semantic level of the building design, the collaborative design will be aligned by using a product model.

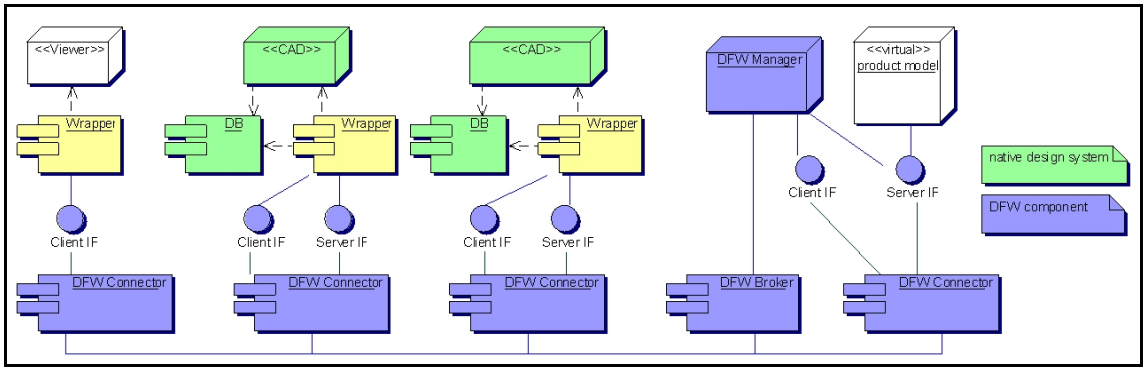

Figure 1

Distributed planning environment

A wrapper works as an adaptor to the framework as it encapsulates specifics of the design system. It does the mapping between the local and the common product model, it implements or maps the notification functionality etc. Each system is both client and server: The own model part will be served, other model parts are used. Viewer components are pure clients.

Connectors link the design systems by using a broker as message distributor. Other central manager components host project information, cache data, control notifications etc.

1.3 The Design Framework

The Design Framework DFW is both an architecture of a distributed building design system and a definition of protocols. The framework assumes that the participating design systems (resp. their wrappers) communicate in terms of a common product model, e.g. the IFC. This means that this product model is used as the vocabulary for exchanging data of common interest. It does **not** mean that the design result is persistent in terms of the product model.

The DFW defines two levels of general messaging protocols. These protocols are abstract in the way that they neither depend on a certain design or CAD system, nor on a specific product model. They just make some assumptions which are described in the next chapter. Concrete design scenarios and concrete local CAD commands can be mapped onto these protocols. Real-life scenarios have been analyzed conc. the question if they can be represented by the DFW.

2 Idea of this work

2.1 Levels of abstraction

The Design Framework consists of two levels of abstraction, i.e. the system level and the component level. Additionally, the term ‘project level’ will be used in order to describe real-life actions.

At the project level, people actually design the building in terms of columns, slabs, walls. People change a column, they release a drawing (which means they release a model subsystem), they are notified that a certain column has been changed (again).

The system level abstracts from concrete building elements, it just uses general concepts like object, model, object owner etc., see below. The single designer 'sees' the DFW as one system which deals with the (building design) project. He/she can use and control that system at a local workstation. Of course, the user knows that he owns only a part of the whole project resp. of the digital building model.

At the component level, the DFW is decomposed into units which are distributed and loosely coupled via the Internet. In terms of software development, each unit can be developed independently, it is described by interfaces only.

2.2 Level description

Each level is described by using a modified UML (Booch 1998) representation. Use cases offer a general overview of the functionality of the system. Each level uses its own object model. The behaviour of these objects is described by using (unified) activity and sequence diagrams. The different states of the objects during these interactions are described by statechart diagrams. Figure 2 shows the meta-model of the level description.

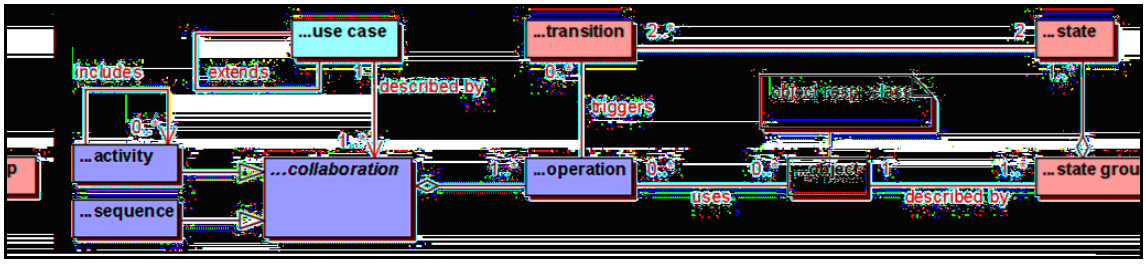

Figure 2

Meta-model of a DFW level description

Thus, a level can be fully described. Use cases are decomposed into collaboration scenes, scenes are decomposed into operations which affect objects by changing their states.This includes that these essential states are determined. By following that description pattern, a certain degree of quality of level description can be achieved: If all instances of a level fulfil the cardinality constraints of the meta model, then we have a closed set of elements. This, of course, does not check the level semantics.

2.3 Level mapping and scenarios

The real-life design process is represented by mapping it to the Design Framework.

- Daily tasks are classified according to the kind of interaction between different design workplaces. (Design workplace means here a single designer with his/her design system resp. partial model.) Thus, business cases are defined like ‘Releasing a subsystem’, ‘Complementing a foreign subsystem’ etc.
- Business cases are decomposed into use cases by using the concepts of the system level. A use case of the system level is more or less equivalent to a command at the user interface.
- At the component level, the same functionality is decomposed into the processes between the distributed components of the Design Framework.

By mapping real-life scenarios onto the abstract framework, we make sure that the Design Framework can represent the concurrent design process. By internally mapping to the component level, we make sure that the framework can be implemented.

3 Scenarios

3.1 Concrete scenarios

3.1.1 Concepts of the project level

Even on the project level, a few concepts must be introduced in order to describe the scenarios:

Each designer is responsible for (at least) one part of the project. In terms of the building model instance, he recognizes these **model parts** as disjoint parts of that model instance. He differentiates between the **own** model part and **foreign** model parts.

Structural engineers, detailer etc. work on different abstractions of the building, they work on **partial model**s of the overall product model.

3.1.2 Project level scenarios

Concrete design scenarios describe typical business cases. They will be mapped onto the Design Framework. The following scenarios have been determined as crucial by a survey (Pegels and Koch 2002). The samples describe just the starting situation. See a complete sample in chapter 3.3.

- Completion in a foreign model part : A detailer needs to attach some stair landing to an existing column, modifying this element of a foreign model part.
- Change of parts which are predefined by another partial model : A detailer needs to modify the section of a beam which has been dimensioned by the structural analysis.
- Request for acceptance : The detailer has changed the pre-dimensioned beam in a significant way and needs a reaction of the structural engineer.
- Management of consistency : A detailer has changed the model, no matter what model part the elements belong to. Now the system should take care of all release cycles.
- Reservation : Several details work on the same column. A detailer wants to reserve the column while it is edited by others.
- Release : The detailer wants to release a model part, ‘freezing’ that state as ‘valid’. Each release must be archived, subsequent changes will require another release.

3.1.3 Interaction types

In highly collaborative environments, designers follow basic interaction types. The DFW supports all these types.

Using **direct interaction**, two partners work on the same design target (i.e. building model). They expect to be able to edit ‘everything’, and they immediately want to ‘see’ what the partner does. There is just one –always current– target state.

A variant of direct interaction is the **invited interaction**. One partner allows for write access (of “his” part) unilaterally.

Using **lazy interaction**, a designer notices changes by partners as late as possible. Of course, design conflicts are avoided as the model is managed synchronously. But changes by partners are only communicated

- if the designer decides to get this information, or
- if the system forces the designer to notice the changes as this is ‘necessary’. This can be caused by actions of the designer himself (e.g. release a subsystem) or by actions of others (e.g. request for acceptance).

Finally, **offline interaction** means that partners work independently on each other. They trust in any conflict management, and they expect that their model parts are synchronized by the next online session.

3.2 Abstract scenarios

3.2.1 Concepts of the system level

At each level, the scenarios work against the objects of that level. The system level uses the following concepts, figure 3.

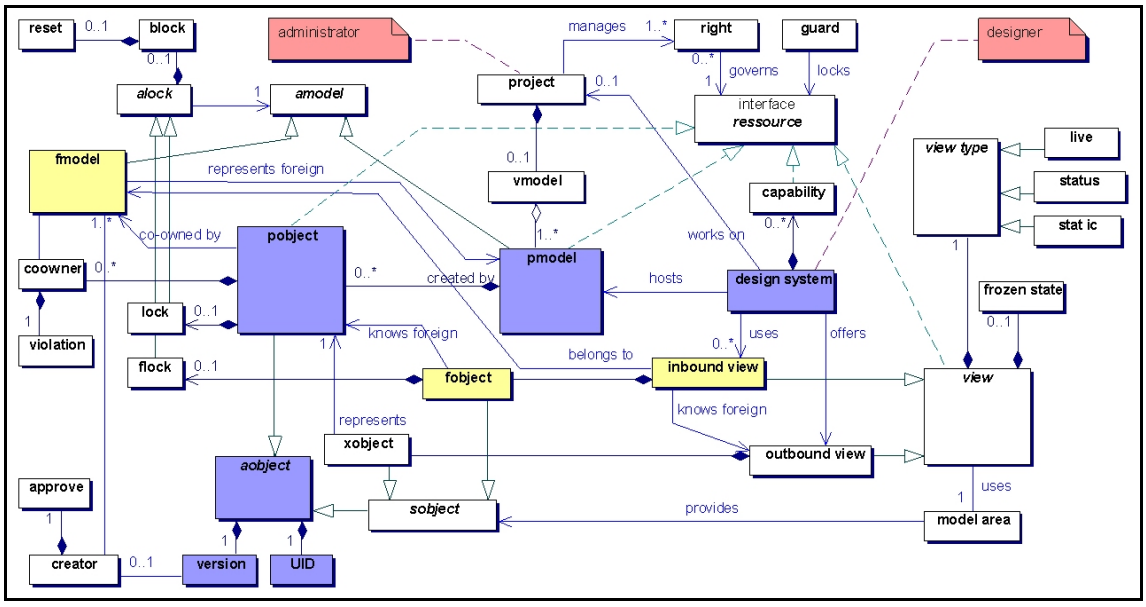

Figure 3

Analysis classes of the DFW system level

A **pmodel** is the local part of the product model instance. It is hosted by a single design system and owned by a single designer. It does not comprises any derived documents. The pmodel consists of **pobject**s. Pmodels are disjoint by definition.

The **vmodel** is the overall instance of the product model. It is called virtual as it is distributed over a number of pmodels, it is the sum of it's pmodels.

The **project** is the aggregation of the all data belonging to a building design. This comprises (exactly one) vmodel, the document vault, right definitions, role definitions and assignments of rights, user definitions and assignments of roles.

A **model area** is a named collection of vmodel queries. By applying the queries, it provides a collection of pobjects. Usually, this happens dynamically, the collection will be updated after any pmodel change. Model areas are used as a generic means to choose pobjects of interest in the vmodel or in a specific foreign pmodel.

A **model capability** is the ability to provide certain services to the DFW. Different design systems might have different features. For instance, a system might be able to notify the DFW of model changes or not.
In practice, model capabilities are used for a loose coupling of tasks. Instead of invoking a task 'blindly', a client asks for capabilities and choose a task to perform.

Each access to resources is ruled by a management of **rights**. In other words: A designer cannot do anything with a foreign pmodel without permission. This even includes any read access. The DFW supports generic rights on the level of model areas.

Every resource is protected by a locking **guard**. A writing client needs to own the appropriate right in principle, and he must have write access. In Online phases, the DFW supports short lock

transactions. This means, that the resource will be freed immediately after modification. The DFW supports locking on the pobject level.

Each pobject has an identificator which is unique in the vmodel, the **UID**.

Over the lifetime of a pobject, an object **version** is maintained. On the vmodel level, there is no design (and version) branching. A designer might try different variants, but he must publish exactly one final version branch only. On the pmodel level, a client can always decide which aobject version (see below) is newer.

Pobjects have a **location**. Each pobject belongs to exactly one pmodel, it spends the whole life cycle inside that pmodel. But by defining views of the vmodel, the designer may fade in objects of other pmodels into his pmodel. These objects are said to be 'foreign' to the local pmodel, they are **fobjects**. Both pobjects and fobjects generalize to **aobjects**.
Each and every pobject exists just once at the semantic level of the vmodel. Locally, a fobject could be seen as a temporary copy of a 'remotely local' pobject. Both instances have different life cycles, and they may be of a different state at the same time. Especially in offline phases, both instances exist in parallel, and they must be synchronized during the next online period.

Pobjects hold **ownerhip** information. Normally, each pobject is 'owned' by just the pmodel it is located in. But pmodels might belong to different partial models, like structural analysis and detailing. In this case, some pobjects of the two pmodels are 'interrelated'. A column as detailing pobject might be part of the detailing pmodel, but coordinates etc. are co-owned by the analysis pmodel. This pobject has two owners !
The concept of ownership supports the management of rights. Multiple ownership allows for general notification and check services.

Each designer directly 'sees' his own pmodel 'through' the local design system. Additionally, he might blend portions of other pmodels, i.e. **inbound views**. The definition of these portions is done by the model area concept. On the other hand, each designer provides a view of his pmodel to others. Actually, he might provide different **outbound views** to different partners. Outbound views may present xobjects instead of pbjects to clients. (A xobject is a proxy of a pobject, representing an older version of it.)
Thus, the final view into a foreign pmodel is defined by two actors, i.e. the owner of the foreign pmodel and the user of the local pmodel.
The term '**view'** does not necessarily refers to a physical representation in a 2D drawing or a 3D model. It means just representation which, for instance, could be a BOM entry as well.

The DFW introduces the following view types, to be combined by the collaborators into one resulting behaviour. A **static view** is just a snapshot of a pmodel portion. The view client is responsible for any update. The **status view** enhances the static view in the way : Modifications are reported automatically to the client. The client knows that an object has been changed, but he does not know how. Again, he is responsible for any update. In a **live view**, all pmodel changes are immediately communicated to clients.

3.2.2 System level use cases and scenarios

Use cases provide a kind of interface of (and into) the system level. The business cases of the project level will be mapped onto that given set of use cases.

The Design Framework defines a set of 40+ use cases (Pegels and Huhn 2004). They cover
- the design process itself (create, modify, delete, reserve etc.),
- pmodel management (open, release, close, view etc.),
- project and platform configuration (assignments, rights, server management etc.),
- network influence (online, offline).

Use cases have been defined by applying the following rules:
- Offer high-level functionality and hide the internal complexity of possible conditions and combinations.
- Try to choose the command layer of the design system as the level of granularity.
- Generalize (building) design operations.
- Support operations of single as well as multiple elements.
- Consider operations always symmetrically: A remote peer can do the same as the local peer.
- Include the network as actor: It can switch to offline as well as the user.
- Reuse use cases by including if possible.

Each use case is realized by one or more abstract scenarios which can be seen as implementing building blocks. Each scenario is defined by an activity diagram. The Design Framework defines a set of 60+ scenarios. They have been defined by applying the following rules:
- Split use cases into scenarios which can be reused.
- Split scenarios at points of user interaction.

3.2.3 Abstract scenario samples

'Open model' is a sample of a command layer use case. It belongs to the pmodel management type and hides the variants
- pmodel not yet assign to a project,
- pmodel assigned to project, pmodel was offline when closed,
- pmodel assigned to project, project is offline, pmodel was online when closed,
- pmodel assigned to project, project is online, pmodel was online when closed.

The use case is implemented by the abstract scenarios 'open model offline', see figure 4, and 'switch online'.

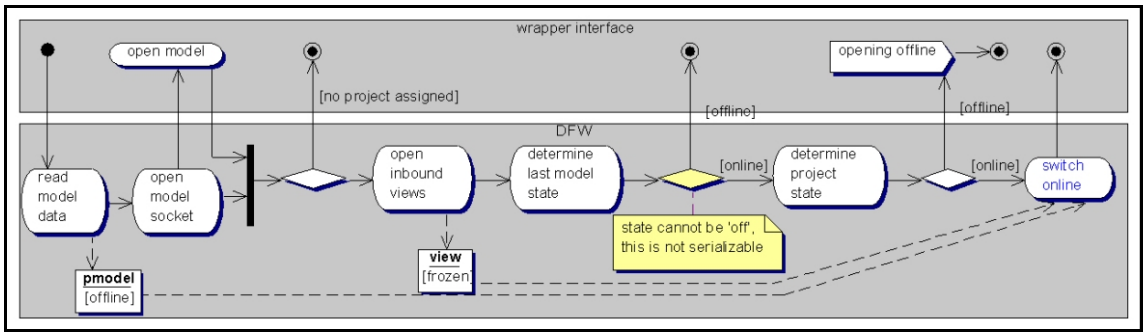

Figure 4

Abstract scenario 'Open model offline'

Concrete design steps are seen as collection of atomic steps which can all be abstracted to a high degree: Every design is just creation, modification or deletion of pobjects. The appropriate use cases always describe the handling of a single pobject. On the other hand, concrete scenarios also address groups of pobjects. In case of possible user interaction, scenarios have to be disconnected for clustering: Before moving a building part, all elements must be reserved for that change. The actual move happens **after** a successful reservation only. At the system level, this means that the check&lock sub-scenario must be performed for all pobjects first. It must be disconnected from the actual modification sub-scenario.

Figure 5 and 6 show the two subsequent sub-scenarios of the 'change object' scenario which are disconnected and thereby useable independently.

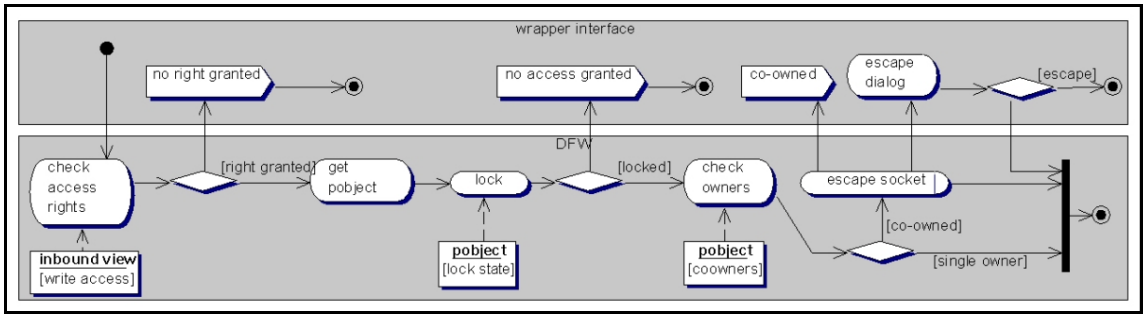

Figure 5

Abstract scenario ‘Change object pre’

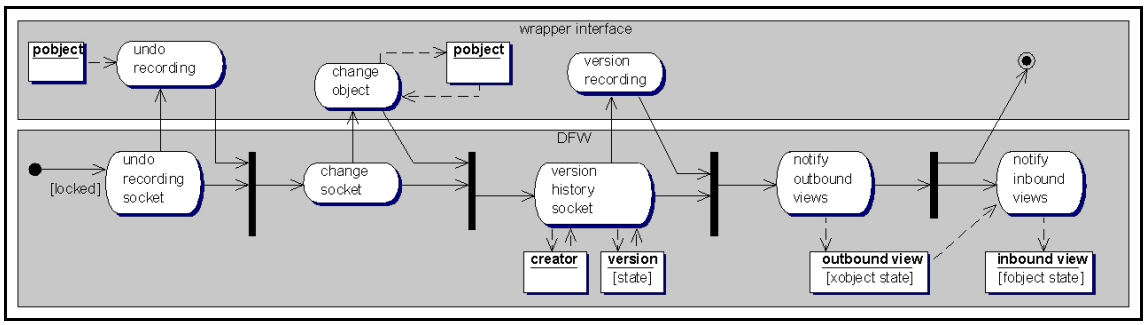

Figure 6

Abstract scenario ‘Change object do’

In a distributed environment, connectivity impacts need to be considered. The designer may decide to work offline, he may return to a connected workplace, the network may crash etc. Even a spontaneous loss of connection is not handled as event but as group of regular use cases. The following figures show two connectivity scenarios.

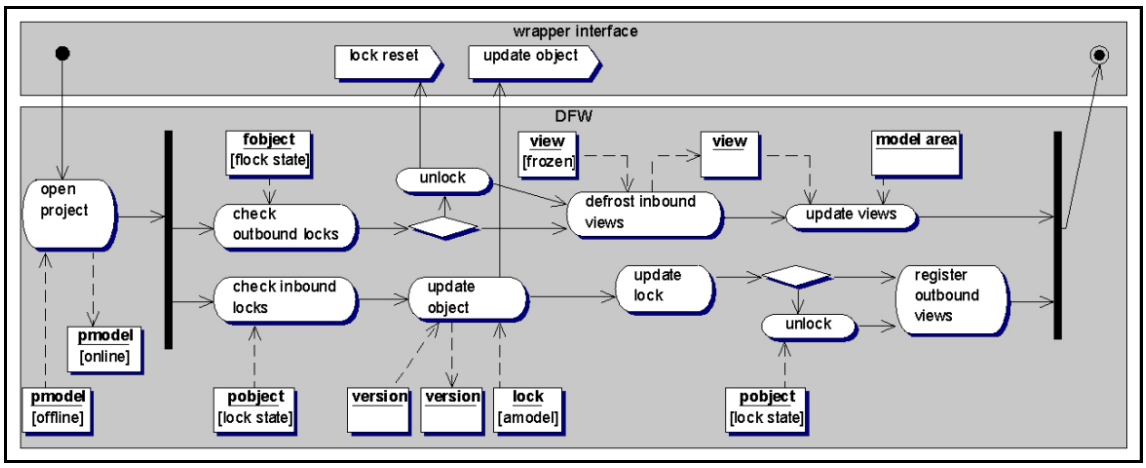

Figure 7

Abstract scenario ‘Switch online’

‘Switch online’ is invoked by a designer command, ‘Spontaneous offline (foreign)’ describes what happens if a remote pmodel gets offline by fault. These samples also illustrate the symmetry: Almost every use case can happen in the local as well as in any remote system.

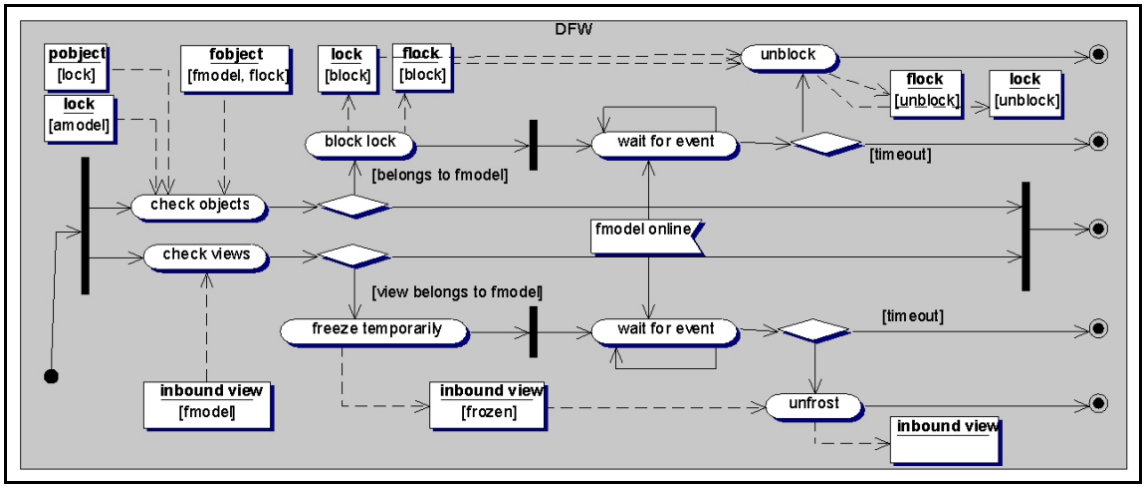

Figure 8

Abstract scenario ‘Spontaneous offline (foreign)’

3.3 Mapping concrete to abstract scenarios

3.3.1 Sample

Concrete scenarios can now be represented by using abstract scenarios as “building blocks”. In the following sample, two designers work in parallel at a structure, see figure 9.

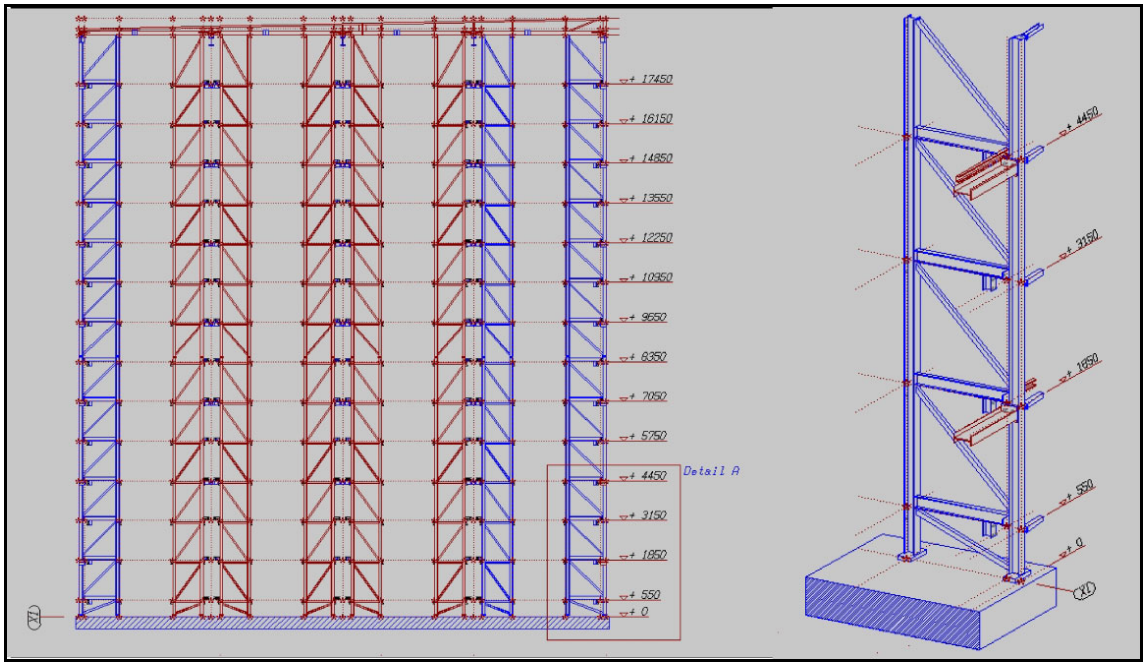

Figure 9

Concurrent design façade / steelwork

Designer F attaches beams (see detail, red) to the existing steelwork. These beams will support façade elements. The joints (endplates) influence the columns, designer F cannot release his model part without clearance by designer S (steelwork).

Figure 10 shows the simplest form of the this scenario.

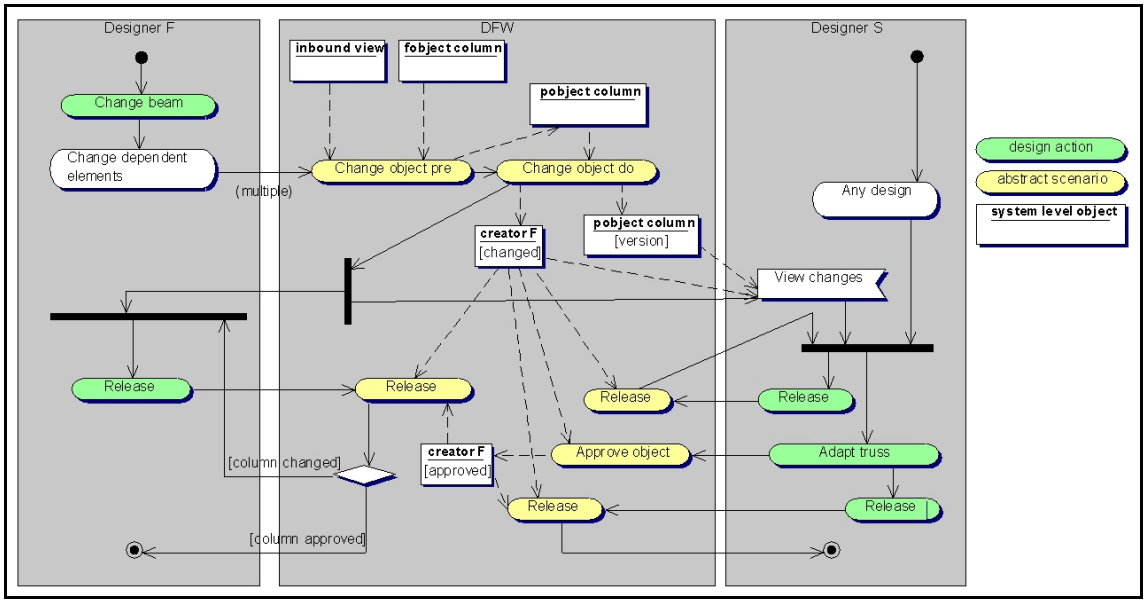

Figure 10

Concrete scenario ‘Consistency management’

We assume that both designers use already a live view in order to work in parallel. Designer F attachs the beam in model F to the column in model S. Usually, he uses a CAD macro. Inside the macro, an endplate will be created, and both the beam and the column will be changed. Scenario ‘Change object pre’ checks all conditions, navigates from the column proxy in model F to the original pobject in model S and locks all objects. Scenario ‘Change object do’ does the changes, updates the versions and saves the information that model F has changed the column. If designer F tried to release his model part right now, the scenario ‘Release’ would deny this. The same applies to designer S. But S will be notified anyway, and he ‘sees’ the changes and can adapt the truss. In the simpliest case, he just approves the new column state. In both cases, the scenario ‘Approve object’ resets the pobject’s creator state. Now, both designers may release their model parts.

3.3.2 More mapping

The abstract scenarios are again mapped onto the objects of the component level. The appropriate scenarios describe the message flow between the components. This must be skipped here for reasons of brevity.

4 Future work

Minor parts of the component level (which has not been covered in detail here) still need to be defined. Essential parts of the design framework will be implemented as part of the SPP1103 project (Meißner et al. 2003) . It is intended that the steelwork domain of the IFC ST4 (ST-4) shall be used as the underlying product model. A reasonable and language-independent GUI will be developed.

Early prototypes show that web services (Christensen et al. 2001) can be used as communication means between the DFW components. More work is necessary in order to simplify asynchronous communication (Zdun et al. 2003). The web service endpoint declarations must become stable.

5 Conclusions

The implementation of a series of crucial scenarios of collaborative design at an abstract level is possible. By using meta models, behavioural models can be checked to a high degree.

The Design Framework can connect design systems, if these systems are satisfied by the quality of the underlying product model in terms of exchange of information.

6 References

ST-4 Structural Analysis Model and Steel Construction; IFC Project 2003 http://cib.bau.tu-dresden.de/icss/structural.html

Booch, G.; Rumbaugh, J; Jacobson, I. (1998). *The Unified Modeling Language User Guide*. Reading: Addison-Wesley

Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S. (2001). *Web Services Description Langauge (WSDL)*. http://www.w3.org/TR/wsdl

Claus, R.; Kazakov, M. (2003). *OMG CAD Services V1.0 Standard – An approach to CAD-Cax integration*. In: Proc. of the 10th ISPE International Conference on Concurrent Engineering (Enhanced Interoperable Systems); Lisse: Balkema, ISBN 90-5809-623-8, pages 565-571

Meißner, U.F. et al. (2003). *Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau*. DFG-Schwerpunktprogramm 1103. http://www.iib.bauing.tu-darmstadt.de/dfg-spp1103

Pegels, G.; Huhn, M. (2004). *Grundlagen vernetzt-kooperativer Planungsprozesse für Komplettbau mit Stahl, Metall, Holz und Glas*, Report II. Bergische Universität Wuppertal

Pegels, G.; Koch, A. (2002). *Grundlagen vernetzt-kooperativer Planungsprozesse für Komplettbau mit Stahl, Metall, Holz und Glas*. Report I. Bergische Universität Wuppertal

Zdun, U.; Voelter, M.; Kircher, M. (2003). *Design and implementation of an asynchronous invocation framework for web services*. In : Proc. Of the International Conference on Web Services ICWS-Europe 2003, Vol. 2853 of Lecture Notes in Computer Science, pages 64-78, Springer