Traceability between Cross-Functional-Teams

Klaus Pohl, Stephan Jacobs
Informatik V, RWTH Aachen
Ahornstr.55, 52056 Aachen, Germany
{pohl, jacobs}@informatik.rwth-aachen.de

Abstract

Traceability between different views, which exist in cross-functional teams, is essential for concurrent engineering. The views (products) of the various teams must be interrelated to each other and presented in a suitable way to emphasize inconsistencies, conflicts, different opportunities. Moreover, decisions together with their rationales must be made explicit. We have developed and implemented a concurrent engineering environment which is based on four basic ideas: (1) record, use, and maintain product interrelations during the concurrent engineering process; (2) capture the decisions and their rationale made during the process; (3) use formal product models to enable product interrelation; (4) provide suitable computer supported tools which hide the formal models and automate the recording of interrelations. First experiences show, that the use of our environment enables traceability of the product interrelations and the decision made within a concurrent engineering process and leads to improved and more consistent process results (products).

1: Introduction

In concurrent planning, mutual understanding of work is the key for success. As large scale projects involve many people, a systematic approach for joint understanding is necessary. Different people have different viewpoints. They are responsible for different production phases, like design, production or control. Each phase has its own language, its own objectives, its own tools and techniques, and produces a special product, e.g., specification, design documents, workflow definitions, quality control plans etc. This diversity is the core of the problem of mutual understanding.

Within traditional, sequential engineering the output of phase A is used as an input of phase B (cf. upper part of figure 1). Thus, the number of interfaces between different phases is essentially defined by the number of phases (number of phases minus one).

Concurrent engineering emphasizes the simultaneous enactment and interplay of all phases (cf. lower part of figure 1). The number of interfaces and so the number of possible mismatches increase to the order of N^2, where N is the number of phases. Moreover, in traditional engineering the communications between different groups is mostly restricted to the point in time where one phase is finished and the next phase is initiated, whereas in a concurrent engineering setting the communication should happen all over the place. The demand for a systematic approach to mutual understanding becomes evident.

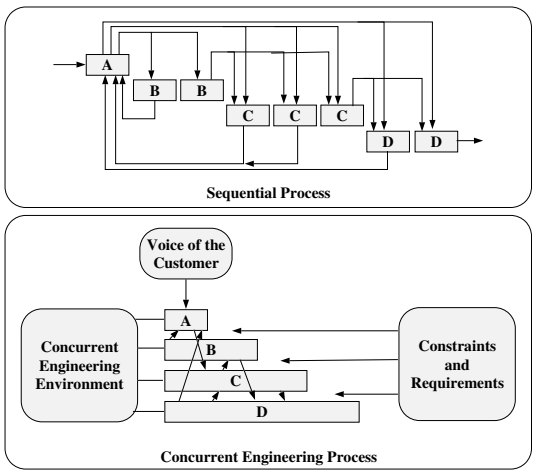

Figure 1 Sequential and concurrent engineering processes [29]

On the technical level, several approaches were proposed to tackle the problem of common understanding. For concurrent engineering Reddy et al. [28] have classified them into five classes:

- Networking of workstations are used to *collocate* geographically dispersed team members and thus provide a means for communication.
- *Coordination* methods [4, 17] are used to harmonize the access to common resources, to guarantee, that

updates are broadcasted to all team members, and to synchronize concurrent tasks.

- *Information sharing* technologies like blackboards [15, 16] are used as common knowledge bases which contain all relevant information concerning a product.
- Management of the *design history* [14, 2, 25] shall capture the rationale behind design decisions to offer not only the result of a process but it's background.
- At last, attempts have been started for *integration* of frameworks, tools, and other services [29, 13] to enable transparent access to related application tools and to shared services.

These approaches all provide technical means for a dispersed team to cooperate and by this enhance the possibilities of concurrent engineering. However, they do not help in joining goals, in mapping and interrelating different products nor in the integration of different methods. They do not provide any help for integrating or interrelating the different viewpoints which exist in concurrent engineering. Beside the technical possibilities for cooperation of different teams, methodical support is needed.

In this paper we describe an approach and implementation which enables *traceability* between cross-functional teams, i.e. between the products developed and maintained by the various teams as well as their histories and rationales.

According to Davis [3] traceability can be defined as the ability to describe and follow the life of an artefact, in both a forward and backward direction, i.e. from its origin to its development and vice versa. A definition of traceability by Greenspan and McGowan [7] emphasizes the importance of traceability for concurrent engineering.

> *"Traceability is a property of a system description technique that allows changes in one of the three system descriptions - requirements, specifications, implementation - to be traced to the corresponding portions of the other descriptions. The correspondence should be maintained through the lifetime of the system."*

For the case of concurrent engineering the phases of the lifetime have to be extended by activities like installation, maintenance, customer feedback studies etc., which are performed in sequential production processes after the implementation phase.

Requirements engineering is a domain, in which in some sense concurrent engineering has been performed since many years. Traditionally, people with different backgrounds and different roles, e.g., sales persons, managers, customers, developers, are involved in the initial phase of product development. To enable the people to cooperate and communicate about the product, traceability has been recognized as an important issue (cf. [19, 1, 5, 22, 6, 27]). Each of the people involved in the requirements engineering task have their own preferences for representations and products. For example, the customer uses natural language to express his needs, whereas the system specialist uses entity relationship and data flow diagrams.

This paper is organized as follows. In chapter 2 we divide the concurrent engineering process into three phases. We illustrate each phase using the specification of a library system as running example. Further, we derive the needs for enabling traceability between cross-functional teams. Based on this framework we describe our prototypical concurrent engineering environment which enables the recording of product dependencies, their use during process execution and their adoption to changes of the products (chapter 3). Finally we summarize our results and focus on extensions planned for the future (chapter 4).

2: A Framework for Traceability

In this chapter we first define typical process steps performed during concurrent engineering (cf. section 2.1). Based on these steps, we derive the requirements for enabling traceability between the different views on the common product developed by the various teams (section 2.2). In the following we use the term "product" to differ between the diverse views of the teams.

2.1: Typical concurrent engineering steps

Figure 2 depicts typical process steps performed during a concurrent engineering process. At the beginning, each team develops its own product (Phase I). At a certain time (t1), the products developed in parallel must be integrated, i.e. discrepancies and similarity must be detected through communication and conversations among the people involved (Phase II). To resolve the discrepancies changes to the products must be made, i.e. parts of the products are interrelated, integrated and changed. After having reached a common understanding of the sys-

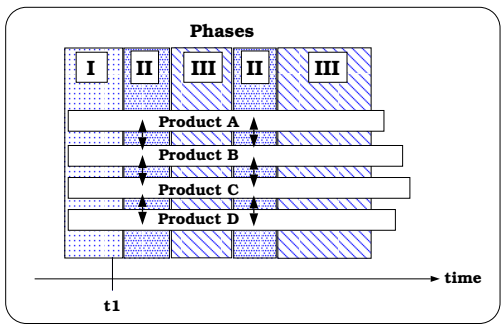

Figure 2 Typical steps in concurrent engineering processes

tem, the products are improved in parallel again (Phase III), i.e., each team improves its own product. But in contrast to phase I, the similarities and discrepancies detected during phase II must be considered. Furthermore, based on the known interrelations changes made to the products should be traceable, i.e. should allow reflection on related part of other products. The improved products must be again interrelated by performing phase II followed by another instantiation of phase III and so forth until the final product is reached.

In the following we explain each phase in more detail. For illustration we use a running example.

Suppose, a library system should be established within a university. After the vision *"build an information system for the library"* was established, the manager decides, that the information system should be developed using concurrent engineering. For eliciting the requirements for the overall system a team is settled up, composed of customers (students, professors), a system analyst, a programmer, a librarian, and the project manager.

2.1.1: Parallel product development (phase I): According to the Kano-Model (cf. figure 3) requirements can be classified into three categories:

- *Basic requirements* are evident for the product. As customers will not demand them explicitly, they have to be added by the specification experts. An example for our library information system would be the backups of the data stored on the disk-systems.
- *Performance requirements* are explicitly asked for by the customer. Whereas fulfilling the basic requirements is obvious, fulfilling performance requirements leads to customers satisfaction. Secure access to the library is an example for a typical performance requirement.
- *Excitement requirements* are not explicitly asked for nor expected by the customer. These requirements are mostly related to brand new and innovative techniques. Therefore excitement requirements are typically stated by technical experts. An example for an excitement requirement for the library would be the automatic sorting of returned books into the bookshelves.

If a stated requirement is categorized as basic, performance or excitement requirement is time-dependent. Requirements which were on the excitement level yesterday (e.g., multimedia) are nowadays on the performance level, and will be on the basic level tomorrow.

Based on the Kano-Model work to be done within phase I (cf. figure 3) is divided.

The customers and librarians define their requirements using natural language. They concentrate on irregularities

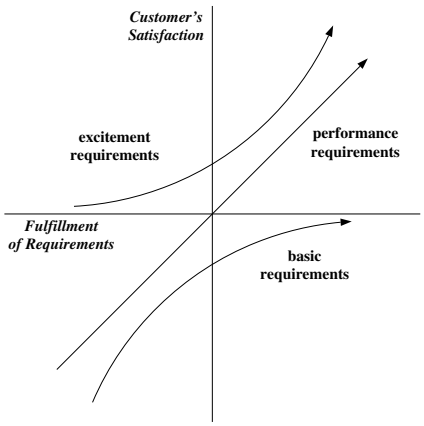

Figure 3 The Kano-Model [21]

and shortcomings of the current system. Moreover, they focus on cases which are typically for university libraries. The system analyst looks at other existing library systems, to identify parts which could be reused. He describes his requirements using data flow and entity relationship diagrams. Moreover, the manager and programmer also use their own representations and product models. For example, the programmer focuses on reusable modules and therefore fixes his understanding of the library system to be developed using architectural diagrams and module specifications.

For simplification, we reduce our example in the following to two concurrently acting persons, namely the librarian and the system analyst. Moreover we focus on the parts of the specification dealing with the object *book*.

Among others, the librarian states the requirement *"different loan durations for proceedings, journals, and normal books are needed*". This requirement is captured in his product, which is simply a normal textual document consisting of chapters, sections, and so forth.

The system analyst reuses a specification of an existing library system, which was established a few years ago within another company. Within this specification, an activity "*loaning of books*" (as an essential activity) was defined as a process in the data-flow diagram. Moreover, looking at the specification of the process "*loaning*", the system analyst recognizes that loaning of books are restricted to special groups of people. Integrating this observation into his current specification for the university library system, he specializes the entity *persons* into *professors, research assistants, students, and stuff*.

The requirements for the new library system stated by the system analyst are represented using entity relationship and data flow diagrams. Hence, his products (diagrams) are (for the moment) not comparable with the product (textual document) of the customer.

2.1.2: Interrelation of products through conversation (phase II): The results gained during the parallel development of the specification must be integrated. Therefore, the parallel development is interrupted, and phase II (cf. figure 2) is started.

In contrast to sequential processes, the products were developed fully independent and the interrelations are not explicit at all. It is completely unclear, to what degree the products represent the same things, which inconsistencies exist, and so forth. Moreover, as narrative documents are informal, automated knowledge acquisition techniques would fail in detecting similarities and differences.

The only chance to interrelate the descriptions is to provoke communication and cooperation between the people involved in the process. Starting with independent products the communication should

- lead to a common view of the current specification. The different products remain unchanged but consistencies and discrepancy between the products must be made explicit. This can be done by interrelating the different products, e.g., by interrelating the entity *book* (defined by the system analyst) and the sentence *"different loan durations for proceedings, journals, and normal books are needed"* stated by the librarian.
- identify isolated requirements, which have no corresponding or conflicting association.
- define activities (actions), by which existing conflicts could be solved. Conflicts have different reasons, e.g., naming problems like synonyms or homonyms, different goals and objectives. Conflicts can only be solved through conversations leading to common decisions. The decisions themselves cause changes to the products. To ensure, that the necessary product changes are made, change activities must be defined, by which the decisions are integrated in the products.

The decisions made during conversations must also be viewed as product of the concurrent engineering process. To enable traceability not only the interrelations of the products must be recorded, but also the decisions made must be interrelated with the other products. In the spirit of the IBIS model [2, 26] a decision chooses among known positions (alternatives) stated during discussions. For each position arguments exist which either support or decline the position. Hence, capturing the chosen position and the alternatives together with the various arguments enables traceability of agreement reached during process execution (cf. [11, 24] for detail).

Coming back to our example, the librarian and the system analyst start communication with the aim to interrelate their products, e.g., they focus on the objects (e.g., books, proceedings ...) of the library and the activities (e.g., loaning of objects). They browse through their products with the aim to detect similarities and dissimilarities. Thus they mutually enrich their understanding of the domain.

Within their meeting the activity *"loaning"* is adopted by the librarian together with the differentiation of people (professors, staff, students ...). Among others, they detect the conflict that the specification of librarian contains a differentiation of *books*, whereas the specification of the analyst does not. After a discussion, they agree on the distinction between *books, journals, and proceedings*. Further, they agree on the refinement of the action *"loaning"*. Now it has to be checked, if the user is allowed to loan a certain book. This depends on the kind of book (journal, proceeding, textbook) and to the group, the user belongs to (student, professor, ...).

Activities resulting from the decisions (e.g., specialization of the entity *book*) are marked as *pending activities*, to be performed. They must be performed to integrate the decisions in the current products after the meeting. Moreover, all parts of the products, dealing with the object *book* or the *loaning* activity are marked, to be checked during changes later on.

During the meeting, the isolated products available before the meeting, are interrelated. The detected relationships must be recorded to enable reuse in later meetings, but also during the further improvement of each specification (product). If the relationships are recorded, the persons are able to look at the other products from their own view. Furthermore, if a product A, which have relationships to a product B is changed, the people responsible for product B can immediately be notified about the changes. Moreover, the parts of their product B effected by the changes in product A can be identified.

Summarizing, after phase II, the isolated products are interrelated. The individual products are enhanced by adopting requirements from each other. Detected conflicts are solved and the decisions as well as their rationales are captured. Moreover, a list of activities to be performed after the meeting, is created.

2.1.3: Parallel development considering and maintaining relationships between products (phase III): This phase consist of two parts.

First, after the meeting the actions defined in phase II are performed. Thereby, the decisions made during the team meeting are integrated in the products. The relationship introduced during the phase II support this integration process. Corresponding items within the different products can be identified and considered during the integration of changes. The integrations can be made in parallel, i.e. each group can change their own documents (products).

Second, after the integration of the decisions each group improves its products in parallel. In contrast to

phase I, the relationships introduced in phase II must be considered. If, e.g., the product is changed the relationships must be changed accordingly. Thus, consistency of relationships is guaranteed over time.

Lets focus on our example. During the specialization of the entity *book* into *proceedings* and *journals* (performed by the system analyst within the ER-diagram), the relation between the entity *book* and the informal statement of the librarian *"different loan durations for proceedings, journals, and normal books are needed"* must be adopted. New relations between the two products must be generated, i.e. a relationship between *proceedings* and the statement *S1* as well as one between *journals* and the same statement. Only if the two new relations are created, the dependence of the entities *proceedings* and *journals* on the statement *S1* of the librarian is represented, and therefore can be used during later changes.

2.1.4: Summary: We have introduced three different phases of concurrent engineering. Each group uses special products to represent its results. Within the first phase, ideas, specifications etc. are collected independently and therefore no relations between the different product exists. In the second phase, the products are interrelated as a result of conversations. Similarities and dissimilarities are made explicit by expressing relations between the different products. Moreover, decisions made together with their rationale are captured. In phase III people work independently again, but in contrast to phase I, existing relationships must be considered and maintained.

After performing all phases an arbitrary number of loops, consisting of phase II (integrating the products, creating relationships) and phase III (parallel improvement of the products) is performed until the final product is reached.

2.2: Requirements for computer supported product interrelation

During a concurrent engineering process, many relations between the cross functional teams, i.e. their products, are created. Only computer supported creation and maintenance of this interrelation enables to deal with the products in a consistent manner and provides traceability for each team member. Traceability is simply enabled by following the product interrelations. Without computer support, known relations are forgotten over time, or become useless if they are not maintained. Offering computer support for recording, using, and maintaining product interrelations requires some prerequisites. In this section we characterize the main requirements to be fulfilled for enabling computer support.

2.2.1: Formal recording of products: Relations can only be recorded, if their source and their destinations are formally defined. In other words, for each kind of product a formal representation is needed, which can be used as the sources and destinations for recording the product interrelations. To enable the recording of relationships between product parts, also each part of the product must be represented by a formal object.

Hence, for each kind of product a formal product model is needed. Each product model should be composed of classes representing possible product parts, e.g., an entity relationship model should consist of classes for entities, relationships, attributes, cardinalities, etc. A concrete product is than represented as instantiation of the product models, i.e. each part of the concrete product is represented by an object which is an instance of the abstract class representing the product parts (cf. section 3.1 for more detail).

But, engineers involved in a concurrent engineering process, should not be confronted with formal models. They should concentrate on their normal task – building a system – and not be bothered with formal models. Since the models get very complex even experts are not able to handle them manually. Computer based tools are needed, which hide the formal models, but ensure a correct instantiation of the models during process execution. Furthermore, maintenance and changes of products get easier if computer support is offered.

Summarizing, the products must be formally represented to enable computer supported recording of product interrelations. Moreover, suitable tools are needed which hide the formal models and provide the engineers with comfortable functions supporting the task of product development.

2.2.2: Recording and use of product interrelations: In practise, there exist many different kind of interrelations. These relationships must be formalized to enable situated use. For each type of relationship an appropriate class must be defined. To support the interrelation of the products, special tools are needed, which helps the engineers in detecting similarities and dissimilarities. An example of such a tool is described in section 3.2. This tool supports the interrelation of the products mainly done during phase II.

As mentioned above, during the whole concurrent engineering process many relationships are created between the various products. The created structure can be viewed as a graph, in which the product parts are typed nodes and the relationships are represented by typed links. Problems of navigating in complex graphs are well known in the hypertext community. An appropriate medium to present

products and their relationship is needed, so that users are able to deal with the various links.

Finally, the user must be able to follow recorded relationships within his normal tool environment, i.e. the environment must provide functions to browse through different products following the recorded product interrelation. In the example introduced above, the user of an entity relationship editor should be able to follow the link from the entity *book* to the sentence *"different loan durations ..."* stated by the librarian with a normal textual document. Thus, each tool must provide actions which enables the user to follow the interrelations. Adding to complexity, in most cases for each kind of product a special tool is needed, e.g., a textual editor for textual documents, a entity relationship editor for entity relationship diagrams. Therefore, if the user wants to follow a link between different products different tools get involved; i.e. each tool must be able to communicate with other tools, to enable browsing between different products. In our example, to follow the link from the entity *book* to its destination (which is part of a textual document) the entity relationship editor must "ask" the text editor to display the destination of the link. Thus, the tools must be interoperable.

2.2.3: Maintaining interrelations: It is not enough to capture the interrelation between the products. The recorded relationships must be adopted to the changes made to the products.

This requires, that the tools record their execution, i.e. if an action is performed within a tool, not only the changes to the product, but also the changes to existing product interrelations, must be recorded. For example, the entity *book* was related to the informal requirement stated by the library *"different loan durations for proceedings, journals, and normal books are needed"*. If a specialization of the entity *book* is made, the informal requirement must be related to the output of the specification, the entities *proceedings* and *journals*. Only if this is done, changes to the informal requirement can be propagated to all parts of the related products, namely the entities *journald* and *proceedingd*.

Maintaining the interrelations must be assured by the tool environment. Beside assuring correct changes to the products, the actions provided by the tools must now additionally assure correct changes (adoption) of the existing product interrelations (cf. [24] for details).

2.3: Summary

We have identified three typical concurrent engineering steps. Based on this steps we have proposed a framework for enabling traceability between the products of cross functional teams. The traceability is based on the formal recording of the products developed as well as the relations which exists (are detected) between the products. Decisions are also viewed as products, which capture the agreement reached within the concurrent engineering team. Moreover, we have stated basic requirements which must be fulfilled to enable the use of this approach during concurrent engineer processes.

In the following we briefly describe our prototypical concurrent engineering environment, which enables traceability between cross-functional teams.

3: Supporting Traceability between Cross-Functional Teams

Based on the need of product integration we have developed an concurrent engineering environment, which enables the recording of relationships between products. Our environment supports the engineers not only during the creation of the relationships, but enables the traceability between the different products based on the recorded relations. Moreover, our environment assures, that if a product is changed, existing relationships between the products are change accordingly. We use the deductive object-oriented database language Telos [18, 12] and its implementation ConceptBase [9] for representing product models, their instantiation, and the product interrelations

In section 3.1 the foundations for enabling the recording of relationships between products are outlined. The recording and detection of product interrelations based on the adoption of the House of Quality is presented in section 3.2. Using and maintaining the recorded relationship during product improvement is depicted in section 3.3. Implementation issues of the prototypical environment are described in section 3.4.

3.1: Enabling product interrelations

As mentioned in section 2.2 formal product models are essential for enabling the recording of the interrelations of products. Within our concurrent engineering environment we use a knowledge base approach to formalize the product models, the products, and the relationships between them. In the following we briefly describe our product model and the way how the models are instantiated by our tool environment. Thus, the models are not visible to the people performing the concurrent engineering process.

3.1.1: Formal product models Describing our formal models in detail is beyond the scope of this paper. In figure 4 two extremely simplified models are shown: the

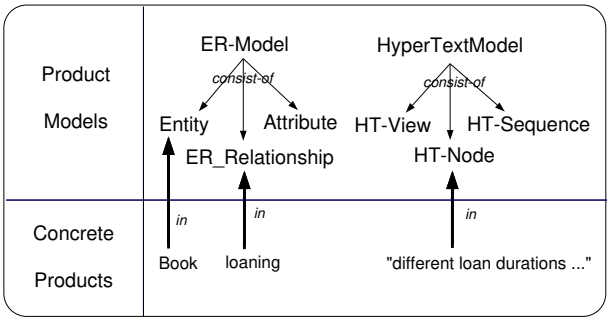

Figure 4 Simplified product models and their instantiation

entity relation ship model and our hypertext model by which textual documents can be structured. The graphical presentation can be understood as a direct representation of corresponding views on a Telos knowledge base and thus have a formal semantic. For example, the class *ER_relationship* is defined in Telos syntax as:

```
MetaClass ER_Relationship in Model_Component,
Class isA ER_Object with
  has_relationship_to, necessary
     connects : ER_Entity
end {ER-Relationship}
```

and the attribute *connects* of the class *ER_Relationship* is defined as

```
Attribute ER_Relationship!connects in Class with
  single, necessary
    min : Integer;
    max : Integer;
    role : String
  constraint
    non_negative :
     $ forall c/ER_Relationship!connects
       forall i/Integer
       (c min i  or c max i) ==>
      (i >= 0) $;
    min_LE_max :
     $ forall c/ER_Relationship!connects
       forall i1/Integer forall i2/Integer
       (c min i1   and   c max i2) ==>
       (i1 <= i2) $
end {end ER_Relationship!connects}
```

The correct instantiation of our product models is guaranteed through constraints which are expressed within our models. Examples of such constraints are the *non_negative* and the *min_LE_max* constraint, which assure, that (1) the cardinality of a *ER-Relationship* can not be negative and (2) that the *max* value is always greater or equal the *min* value.

Besides the products, the possible product interrelations are formally defined. For example, for the relation *refers_to*, a constraint is defined, which assures that all *refers_to* relations are symmetric, i.e. if a *refers_to* relations goes from object A to object B, an *refers_to* relation between object B and A is automatically introduced and vise versa

To enable the interrelation of all products of the concurrent engineering process, for each product a corresponding formal product model must be defined. For example, if the decisions made during the process (together with their rationales) should be recorded, a formal model for structuring the decision is needed. A particular decision is than recorded as instantiation of this model. Hence decisions can be interrelated with all the other products respectively product parts, e.g., an entity.

Coming back to our running example. The recording of the decision "specialization of the entity *books*" enables the relation of the decision to the output of the specialization activity, namely the entities *proceedings* and *journals* by a link of the type *depends_on_decision*. This interrelation is very important. For example, if the decision is revised later on, the effected product parts can be retrieved and presented to the engineers. Or, vice versa, if the entity *proceedings* should be deleted, the decision responsible for the creation can be retrieved and presented to the engineer. This situations are just two examples in which the recorded decision history can help the concurrent engineering team to understand the current product and choose among alternatives.

3.1.2: Instantiation of product models.

The models needed for all the products and different kind of interrelations get very huge; we have currently defined over 100 classes for product parts and about 15 different types of interrelations. To avoid additional work load for the engineers, the tools must be made responsible for correct product instantiation.

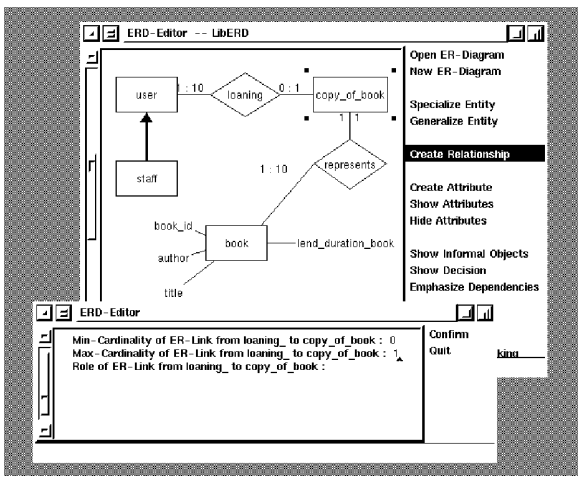

Figure 5 Entity relationship editor during the creation of a relationship

Let us focus on the creation of an relationship between the entity *copy_of_book* and *users*. Using our ER-editor (cf. figure 5), the relationship *loaning* was created. The input of the engineer consists of the name of the relationship (loaning) and the two cardinalities (0:10 a user is allowed to loan up to ten books; 0:1 a certain copy of book can only be loaned by one user at each time).

The action *create relationship* of the ER-editor creates the following formal description which is immediately stored within the knowledge base.

```
Class loaning in ER_Relationship,SimpleClass with
  connects
    e1 : user;
    e2 : copy_of_book
  end
Attribute loaning!e1 in ER_Relationship!connects
 with
   min
     min1 : 0
   max
     max1 : 10
end
Attribute loaning!e2 in ER_Relationship!connects
 with
   min
     min1 : 0
   max
     max1 : 1
  end
```

In the next section we focus on the support for the creation of product interrelation during the phase II. This task is supported by the House of Quality, which is only possible, since the products are recorded using formal models, and therefore the relationships between them can be stored as links between the corresponding product parts.

3.2: Detecting and recording product interrelations

The formal integration of different products has to be supported by informal communication and cooperation. As it is very cumbersome to figure out all traces by queries or browsing, an appropriated medium is needed to present the products as well as their relations within a clear form. This will support the mutual understanding, provoke communication, and help to detect conflicts. The House of Quality (HoQ), well known from Quality Function Deployment [8], [30], [20], can be used as a central forum for information and discussion. As the products are formally represented both products are automatically mapped from their original representation into the HoQ. The fact that most product models are based on tree like structures enables an easy automatic transformation into the HoQ-form. The narrative text of the analysts is structured into chapters, sections, etc. Data-Flow-Diagrams get the tree structure out of their recursive construction. Thus, the HoQ is filled on the one side with entities, their attributes,

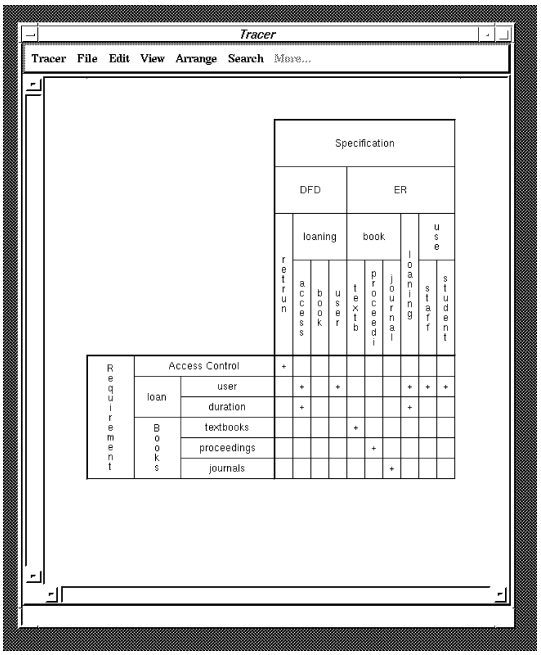

Figure 6 Visualisation of products and their interrelations

relationships and the like. The other side is filled with keywords representing requirements (cf. figure 6).

The HoQ enables the user to detect and define relationships between two products. First, because of the clear form relationships can easily be identified. Second, relationships can be recorded just by filling the corresponding field within the correlation matrix. Rules formalized in the environment guarantee the automatic translation of correlations into the database.

3.3: Using and maintaining product interrelations

The tool environment supports not only the instantiation of the product models and the interrelation of concrete products, but also their use during further development. If there exist interrelations, the engineer is enabled to browse between the different products. For example, using our tool environment the engineer is able to retrieve all related objects of the entity *book*. The recorded relationship between the entity *book* and the sentence *"different lending durations ..."* is retrieved from the knowledge base and the parts of the informal descriptions which are related to the entity *book* are highlighted (cf. figure 7).

Additionally, our tool environment supports the engineer by changing the products. It assures that the recorded relationships are adopted to the product changes. Focusing on our example, the system analyst specializes the entity

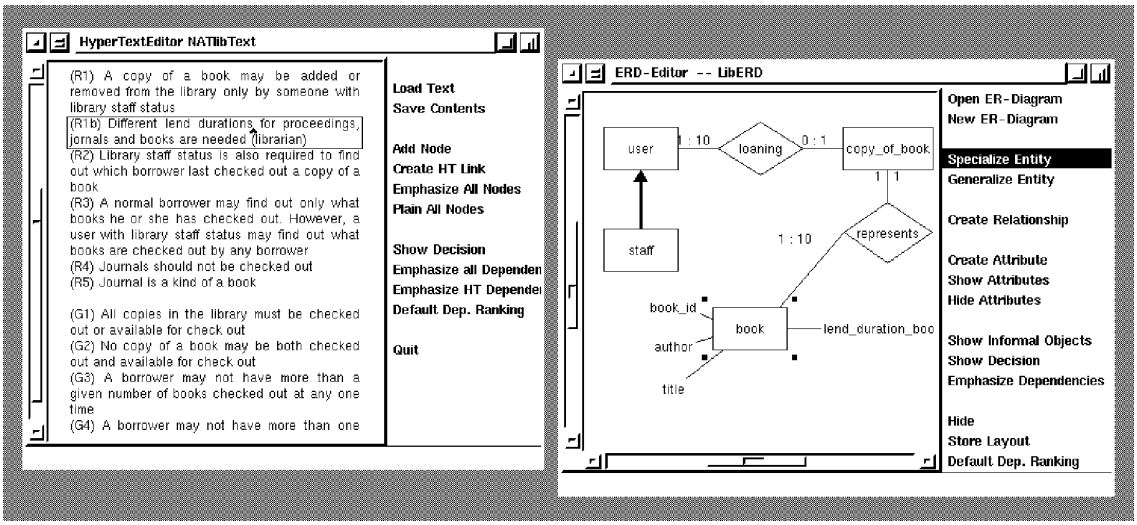

Figure 7 Browsing interrelations between products

book into *proceedings* and *journals*. During the execution of the action *specialize entity* (provided by the ER-editor) the tool assures, that all relationships between the entity *book* and other objects (product parts) are checked and, if necessary, new relationships between the object and the new created entities *proceedings* and *journals* are created. Among others, the decision made within phase II *"specialize entity book"* is related to the new created entities. Thus, the use and maintenance of the recorded product interrelations is supported by our tool environment.

3.4: Implementation Issues

The architecture of our concurrent engineering environment is shown in figure 8. A detailed description can be found in [23, 24]. The environment was implemented using C and C++. The graphical user interfaces were build using X11 R5 and the Andrew Toolkit. To enable the use

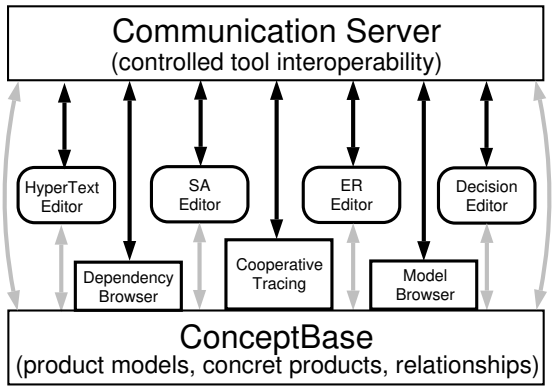

Figure 8 Architecture of our concurrent engineering environment

of the environment in a distributed setting, the communication between the tools as well as the communication between the tools and the knowledge base are based on IPC protocols. The formal models for products, relationships, and the tools where formalized using the knowledge representation language Telos [18, 12] and its implementation ConceptBase Version 3.2 [9, 10]. Currently the whole prototypical environment consist of more than 100.000 lines of source code. The formal models cover more than 100 different classes.

First experiments, which where performed in-house with students, have shown, that the availability and the use of product interrelations enhances the concurrent engineering processes. The results gained are more consistent than without our environment and the process as a whole becomes traceable.

4: Summary

We have presented an approach and described its implementation which enables traceability between cross-functional teams, i.e. between the products developed and maintained by the various teams involved in concurrent engineering. Moreover, using this approach the decisions made during the concurrent engineering process as well as their rationales can be recorded and interrelated with the various products. By recording the information using formal models, reasoning about the interrelations and the products is enabled. Furthermore, the engineers are able to browse through the interrelated products and the history of the concurrent process becomes traceable.

We have proposed a three phase approach for concurrent engineering. Our prototypical concurrent engineering

environment supports these phases by

- recording the parallel developed products using formal product models and appropriated tools;
- detecting and recording product interrelations based on the formal models using a tool which enhances communication and cooperation between different teams;
- maintaining and using the product interrelations during further development.

Currently the framework is evaluated in a typical concurrent engineering setting, the requirements engineering process for software systems. Experience shows, that the use of our environment enables traceability of the product interrelations and the decision made within a concurrent engineering process and leads to improved and more consistent process results (products). Another domain which serves as a test field, is factory layout planning. In the domain of factory layout planning completely different product models are in use. We are currently formalizing these models and extending our prototypical environment by appropriated tools. The work done so far shows, that our approach is adoptable to other domains.

Acknowledgments

We like to thank all our students which helped us in implementing our prototype, namely: Michael Gebhardt, Friedhelm Gibbels, Peter Haumer, Christof Lenzen, Monika Pandey, Klaus Weidenhaupt, and Claudia Welter.

The work was supported by the ESPRIT Basic Research Action NATURE (#6353) and the state of Nordrhein-Westfalen, Germany.

References

[1] *IEE Colloquium on Tools and Techniques for Maintaining Traceability During Design*. IEE London, UK, 12 1991.

[2] Jeff Conklin and Michael L. Begemann. gIBIS: A Hypertext Tool for Exploratoy Policy Discussion. *ACM TOOIS*, pages 140–151, 10 1988.

[3] Alan M. Davis. The Analysis and Specification of Systems and Software Requirements. In R.H. Thayer and M. Dorfman, editors, *Systems and Software Requirements Engineering*, pages 119–134. IEEE Computer Society Press — Tutorial, 1990.

[4] Prasun Dewan and John Riedl. Toward Computer-Supported Concurrent Software Engineering. *Computer*, 27(1):17–27, 1 1993.

[5] Merlin Dorfman and Richard H. Thayer. Standards, Guidelines and Examples on System and Software Requirements Engineering. *IEEE Computer Society Press - Tutorial*, 1990.

[6] O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability. In *IEEE International Conference on Requirements Engineering, Colorado Springs*, 4 1994.

[7] S. Greenspan and C. McGowan. Structuring Software Development for Reliability. *Microelectronics and Reliability*, 17:75–84, 1978.

[8] J.R. Hauser and D. Clausing. The House of Quality. *Harvard Business Review*, pages 63–73, 5 1988.

[9] Matthias Jarke. *Database Application Engineering with DAIDA*. Springer-Verlag, 1993.

[10] Matthias Jarke, Stefan Eherer, Rainer Gallersdörfer, Manfred Jeusfeld, and Martin Staudt. ConceptBase - A Deductive Object Base Manager. *International Journal of Intelligent Informations Systems, Special Issue on Advances in Deductive Object-Oriented Databases*, 1994.

[11] Matthias Jarke, Klaus Pohl, Colette Rolland, and Jean-Roch Schmitt. Experience-Based Method Evaluation and Improvement: A Process Modeling Approach. In *IFIP WG 8.1 Conference CRIS '94*, Maastricht, Netherlands, 1994. (submitted).

[12] Manfred Jeusfeld. *Änderungskontrolle in deduktiven Datenbanken*. Infix, 1992.

[13] T. Kiriyama, T. Tomiyama, and H. Yoshikawa. A Model Integration Framework for Cooperative Design. In *Proceedings MIT-JSME Workshop Computer-Aided Cooperative Product Development*, pages 126–139, 11 1989.

[14] Mark Klein. Capturing Design Rationale in Concurrent Engineering Teams. *Computer*, 27(1):39–47, 1 1993.

[15] F.L. Krause and B. Ochs. Potentials of Advanced Concurrent Engineering Methods. In *Proceedings IFIP Transactions B-6, Manufacturing in the Era of Concurrent Engineering, Herzlya, Israel*, pages 15–26, 4 1992.

[16] F. Londono, K.J. CLeetus, and Y.V. Reddy. A Blackboard Scheme for Cooperatice Problem-Solving by Human Experts. In *Proceedings MIT-JSME Workshop Computer-Aided Cooperative Product Development, Cambridge, USA*, pages 26–50, 11 1989.

[17] T.W. Malone, K.R. Grant, F.A. Turbak, S.A. Brobst, and M.D. Cohen. Intelligent Information-Sharing Systems. *Communications of the ACM*, 30(5):390–402, 5 1987.

[18] John Mylopoulos, Alex Borgida, Matthias Jarke, and Manolis Koubarakis. TELOS: Representing Knowledge about Information Systems. *ACM Transactions on Information Systems*, 8(4):325–362, 1990.

[19] Bashar Nuseibeh and Anthony Finkelstein. ViewPoint: A Vehicle for Method and Tool Integration. In *IEEE Proceedings of the International Workshop on CASE, CASE 92, Montreal*, pages 50–60, 7 1992.

[20] Lawrence P. Sullivan. Policy Management through Quality Function Deployment. *Quality Progress*, pages 18–20, 6 1988.

[21] Tilo Pfeifer. *Qualitätsmangement (in German)*. Karl Hanser Verlag, 1993.

[22] Klaus Pohl. The Three Dimension of Requirements Engineering. *Proc. of the CAiSE Conference, Paris, France, June 6-8*, pages 275–292, 6 1993.

[23] Klaus Pohl, Petia Assenova, Ralf Dömges, Paul Johannesson, Neil Maiden, Veronique Plihon, Jean-Roch Schmitt, and Giwrgos Spandoudakis. Applying AI Techniques to Requirements Engineering: The NATURE Prototype. In *Proceedings ICSE-Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence.*, 1994.

[24] Klaus Pohl, Ralf Dömges, and Matthias Jarke. PRO-ART: An Environment for Enabling Requirements Traceability. Technical report, Informatik V, RWTH-Aachen, 1994. (submitted).

[25] Colin Potts and Glenn Bruns. Recording the Reasons for Design Decisions. *10th International Conference on Software Engineering*, pages 418–427, 4 1988.

[26] Balasubramaniam Ramesh and Vasant Dhar. Group Support and Change Propagation in Requirements Engineering. In Matthias Jarke, editor, *Database Application Engineering with DAIDA*, pages 221–242. Springer, 1992.

[27] Balasubramaniam Ramesh and Michael Edwards. Issues in the Design of a Requirements Traceability Model. In *IEEE International Symposium on Requirements Engineering, San Diego*, 1 1993.

[28] Y.V. Ramana Reddy, Kanakanahalli Srinivas, V. Jagannathan, and Raghu Karinthi. Computer Support for Concurrent Engineering. *COMPUTER*, 27(1):12–16, 1 1993.

[29] R.A. Sprague, K.J. Singh, and R.T. Wood. Concurrent Engineering in Product Development. *IEEE Design and Test of Computers*, 8(1):6–13, 3 1991.

[30] L.P. Sullivan. Quality Function Deployment. *Quality Progress*, pages 39–50, 6 1986.