Concurrent Engineering: Enabling Traceability and Mutual Understanding

Klaus Pohl, Stephan Jacobs

Informatik V, RWTH Aachen, Ahornstr.55, 52056 Aachen, Germany
email: {pohl, jacobs}@informatik.rwth-aachen.de

Concurrent Engineering requires the cooperation of people coming from different phases of the engineering process. Traceability between the different views (products), which exist in such cross-functional teams, is essential for enabling mutual understanding. Moreover, the different views must be related to each other and must be presented in a suitable way to support finding and resolving of inconsistencies, conflicts, and different opportunities.
We have developed and implemented a concurrent engineering environment, called PRO-ART, which is based on four main ideas: (1) record, use, and maintain the various products using formal product models; (2) capture the relationships between the products detected during process performance; (3) present the relations based on the ideas of the House of Quality; (4) provide a computer based environment, which hides the formal models and automates the recording of products and their interrelations.
The recording and situated retrieval of the information is enabled by a repository which has been implemented using the deductive and object oriented knowledgebase system ConceptBase.

1. Introduction

In contrast to sequential engineering, in concurrent engineering people stemming from different phases of traditional engineering processes must closely work together. Thus the setting for performing concurrent engineering process is heterogenous and the people within such cross-functional teams have different views and aims about the system, since they are responsible for different development 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.

However, this variety of methods and notations should not be replaced by new approaches, since each method and notation has its certain strength and weakness and users would not accept a complete change of their way of working. But mutual understanding is the key for success and therefore a systematic approach for enabling joint understanding is needed.

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 increases 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. Hence, the demand for a systematic approach to mutual understanding becomes even more evident.

On the technical level, several approaches were proposed to tackle the problem of common understanding. For concurrent engineering Reddy et al. [59] have classified them into five classes:

- Networking of workstations are used to *collocate* geographically dispersed team members and thus provide a mean for communication.
- *Coordination* methods [13, 41] are used to harmonize the access to common resources, to guarantee, that

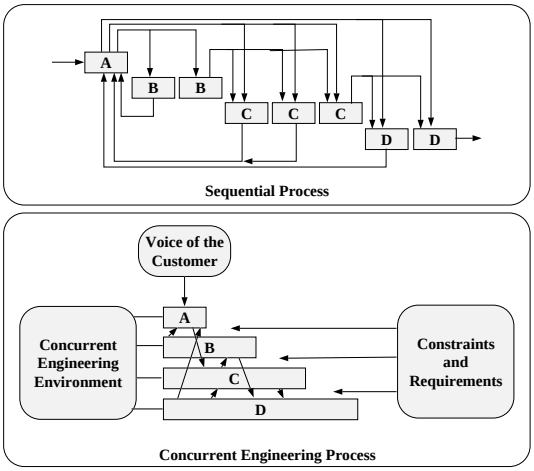

Figure 1 Sequential and concurrent engineering processes [66]

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

- *Information sharing* technologies like blackboards [38, 40] are used as common knowledge bases which contain all relevant information concerning a product.
- Management of the *design history* [37, 7, 55] shall capture the rationale behind design decisions and therefore not only the product of the process is recorded, but also it's development history, e.g. decisions made during the process.
- At last, attempts have been started for *integration* of frameworks, tools, and other services [66, 36] to enable transparent access to related application tools and to shared services.

These approaches provide technical means for a dispersed team to cooperate and by this improve the product of the concurrent engineering process.

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.

Requirements engineering is a domain, in which in some sense concurrent engineering has been performed since many years. Traditionally, people with different background and different roles, e.g., sales persons, managers, customers, developers are involved in the initial phase of the system (product) development. To enable the people to cooperate and communicate about the specification, traceability has been recognized as an important issue (cf. [46, 1, 14, 50, 19, 58, 49, 54]).

As in concurrent engineering processes, 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.

According to Davis [10] traceability can be defined as the ability to describe and follow the life of an artifact, in both a forward and backward direction, i.e. from its origin to its development and vice versa. The definition of traceability *"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." [20]* emphasizes the importance of traceability for concurrent engineering.

However, enabling traceability of the developed artifacts and supporting mutual understanding should be based on a clear definition of the concurrent engineering process.

We therefore define the typical process steps performed during a concurrent engineering process (cf. section 2). Based on this definition we derive the needs for enabling traceability between cross-functional teams and supporting mutual understanding 3. Our approach for enabling traceability, called PRO-ART, is described in section 4. CoDecide, our approach for supporting mutual understanding is outlined in section 5. A brief description of the current prototype implementation is given in section 6. Finally we illustrate our approaches using an example (section 7) and draw some conclusions (section 8).

2. Typical concurrent engineering steps

At the beginning of the process, each person (team) develops its own product using his/her preferred representation (Phase I). Dependencies which exist between the parts of one product are also created within this phase, e.g. dependencies between user requirements. At a certain time (t1), the products developed in parallel must be integrated, i.e. discrepancies and similarities must be detected through communication and conversation among the people involved (Phase II). Changes to the products have to be made, according to unresolved descrepancies. I.e. parts of the products are interrelated, integrated, and changed.

After having reached a common understanding of the system, 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 parts of other products.

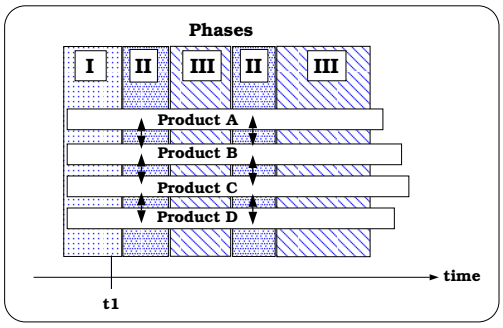

Figure 2 Typical steps in concurrent engineering processes

The improved products must again be 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"* is established, the manager of the company responsible for developing the library system 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 sales person, a system analyst, a programmer, a librarian, and the project manager.

2.1. Phase I: Parallel product development

Based on the Kano-Model (cf. [47]) work to be done during *phase I* can be divided. According to the Kano-Model 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.

However, during *phase I* the customers together with the sales person and the librarians define the (user) requirements using natural language. They concentrate on irregularities and shortcomings of the current system. Moreover, they focus on cases which are typically for university libraries.

The system analyst focus on the specification of an existing library systems, to identify parts which could be reused. He describes the elicited 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 *"proceedings should only be checked out for three days"*. This requirement is captured in his product, which is a hypertext document, where the requirements can be interrelated.

Within the specification of the system analyst, 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*" of the existing library system, the 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 *user* into *professor*, *research assistant*, *student*, and *staff*. 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.

2.2. Phase II: Interrelating products through cooperation

At a certain time, the products produced during *phase I* 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 therefore their interrela-

tions 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 discrepancies 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 *"proceedings should only be checked out for three days"* 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, or 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 into the products.

To enable traceability not only the products and their interrelations must be recorded, but also the decisions must be viewed as products and be interrelated with the other products. In the spirit of the IBIS model [7, 57] 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. [32, 52] 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.

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 has 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.

2.3. Phase III: Considering and maintaining product interrelations during parallel development

This phase consists of two parts. First, the decision made during *phase II* are integrated into the corresponding products. The relationships introduced during *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 the products are improved again 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.

After *phase III* 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.

2.4. 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 exist. 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 rationales are captured. In *phase III* people work independently again, but in contrast to *phase I*, existing relationships must be considered and maintained.

3. Requirements for enabling traceability and mutual understanding

Based on the three phases of concurrent engineering processes, we derive the requirements for enabling traceability between the different products developed by the various teams (section 3.1). Moreover, we outline the basic needs for enabling mutual understanding (section 3.2). In the following we use the term "product" to differ between the diverse views of the teams.

3.1. 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 capture 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 using (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.

3.1.1. 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 model, 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 4 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.

3.1.2. Recording and use of product interrelations

In practise, there exist many different kind of interrelations. These relationships must be formalized to enable situated use. To support the interrelation of the products, an approach is needed, which helps the engineers in detecting similarities and dissimilarities (cf. section 3.2).

As mentioned above, during the whole concurrent engineering process many relationships are created between or within 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 various links (cf. section3.2).

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 *"the maximum lend duration for books is 10 days."* stated by the librarian within a 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, an 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.

3.1.3. 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. 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. [52] for details).

3.1.4. Summary

Summarizing, traceability between the products must be based on formal product models. Moreover, the relationships between the products must be formalized to enable computer support. Capturing and maintaining these interrelations must be (as far as possible) automated. Therefore a computer based engineering environment is needed, which records the interrelations and supports the use and adaptation of existing relations during process performance. As a prerequisite, the environment must provide interoperable tools by which the various products are modified and the action offered by the tools must (as far as possible) automatically create new interrelations and maintain existing ones.

3.2. Supporting mutual understanding

Interrelated products are the prerequisite to support a team within concurrent engineering. However, the users must be able to deal with the dependencies, to trace the interrelations, i.e. to “read” the links. A graph browser containing all the formal products as nodes and their interrelation as links, is not a good means to present the interrelations in a clear way. To enable cooperation and communication about the different goals a common, intuitively usable presentation technique is required, which first enables the users to understand the goals by the other team members, and second allow them to place their own goals face-to-face with the other goals. This common presentation should serve as a "central forum" for discussion and information [23].

In the following the requirements for this forum are described in more detail.

3.2.1. Intuitive presentation

The presentations used for the products have been optimized. So they are well suited when working with one method or when presenting the results to another expert, who is familiar with the presentation. However, presenting the product to non-experts and wishing to place one product face-to-face with another product changes the situation. The optimized presentation are no longer suitable, as the non-experts are not familiar with it. That means, the criteria for estimating the quality of the presentation have changed. The most important criteria are now that

- first, non experts have to understand the specified product (at least they have to get an impression about it);
- second, the product must be presented face-to-face with another product. It is important to make correlations explicit, easy to identify, and intuitively to describe so that discussion and communication about the different products and their interrelations are provoked.

These criteria are in conflict. Whereas the demand for intuitively understandable and usable presentation requires a very simple notation, the wish for a notation that covers nearly all other notations requires a notation which is very expressive.

As mutual understanding requires the exchange of ideas, comments, and explanations the intuitive presentation of products have to be complemented by means to support informal communication, negotiation, and decision making. Therefore, different cooperation styles are required [29, 30].

3.2.2. Automatic transformation

Using different presentation techniques should offer aid to the cooperating users. Thus, there should be no additional work to transform one presentation into another one. Mapping presentations is often a time-consuming and boring task. So, it should be done automatically.

We therefore propose the use of a formal representation behind the presentations used for placing the different products face-to-face. Since all other products are recorded using a formal model, a mapping of the different products into the formal model used for the common presentation can be supported (automated).

3.2.3. Summary

Traceability has to be complemented by a well suited presentation, which supports mutual understanding. Presentational means used for performing a method are usually unsuitable as a medium for cooperation, at last for cross-functional team. A well suited presentation should enable the team members to intuitively express their own ideas, understand other ideas, and interrelate them. The presentation has to provoke communication about these ideas and should be complemented by corresponding technologies. To avoid time-consuming manual mappings between the single presentations, the cooperative presentation has to be formalized and to enable automatic transformation between the presentations.

4. PRO–ART: Enabling traceability

In this section we outline our approach for enabling traceability between cross functional teams, i.e. for recording and interrelating the various products produced during the concurrent engineering process using formal models. In section 4.1 we briefly describe the Information Resource Dictionary Standard (IRDS), which builds the foundation for recording the various information in a central repository. We outline a possible implementation platform for repositories (section 4.2) and provide two examples of formal product model in section 4.3. The dependency taxonomy established for recording dependency between the various products is outlined in section 4.4. Finally, the principle approach of the environment for enabling automated recording of the products and their interrelations is characterized in section 4.5.

4.1. Information Resource Dictionary Standard (IRDS)

In the field of information technology, ISO (International Standard Organization) and IEC (International Electrotechnical Commission) have established a joint technical committee, ISO/IEC JTC 1. The IRD Standard (ISO/IEC 10027 [28]) was prepared by this committee and belongs to a series of international standards on Information resource dictionary systems. The purpose of the family of international standards for Information Resource Dictionary Systems is to provide a common basis for the development of Information Resource Dictionaries. The IRDS Framework International Standard (ISO/IEC 10027, [28]) defines the context for the other standards of the family, e.g. IRDS Service Interface Standard ISO 10728:1993 or IRDS Reference Data Management Standard ISO 10032:1993.

An *information resource dictionary (IRD)* is a shareable repository for the definition of the information resources, relevant to all or part of an enterprise. An information resource dictionary system (IRDS) is a system which provides facilities for creating, maintaining and accessing an IRD and its definition. The IRDS standard does not provide a definition of all the information to be recorded in an IRD, but it does provide a framework for defining, representing, and managing such information.

The cornerstone of the IRDS framework is the concept of *four data levels* and the associated three *level pairs* (cf. figure 3). The purpose of these four data levels is to enable extensions of the data to be stored in the IRD. The four data levels are defined as follows:

IRD Definition Schema Level: At this level the types of objects are prescribed about which data may be recorded at the IRD Definition Level.

IRD Definition Level: Using the types established at the IRD Definition Schema Level, IRD Definitions are defined at this level. A part of an IRD Definition (called IRD Schema) prescribes the types of objects about which data may be stored in one ore more *IRD*. An IRD definition may contain one or more IRD Schemata. At any point in time an IRD Schema is a subset of the IRD definition, consisting of the parts that the dictionary administrator has chosen to make active. For example all object types (e.g., *entity-type*) defining a particular entity relationship diagram language may constitute an IRD Schema.

IRD Level: Application schemata are defined at this level using the IRD schema defined at the IRD Definition Level. The types of data at an IRD level are completely defined by the data held in the applicable IRD schema. There may be any number of IRDs existing, all described using one IRD Schema, e.g., an Entity Relationship model describing the application schema for a *library* system and another one describing an ER schema for a *petrol filling station*.

Application Level: At this level instances of business applications are recorded. For example, the people (e.g., *Bill Jones*) having loaned a particular book (e.g., *The Pelican Brief*).

In other words, IRDS is organized along the classification dimension of semantic data models abstraction. In a layered type universe of four levels, level *n+1* (called the *defining* level) constitutes a type system for the level *n* (the *defined* level); it defines the language in which level *n* can be specified. In increasing order of abstraction, the four levels are the Application Level, the IRD Level, the IRD Definition Level, and the IRD Definition Schema Level.

Consider the specification of a library system as an example. The description about actual *books* and *proceedings* as well as the people having some of the actual books on loan are described at the Application Level, e.g. they are recorded using a database application program. The record type or the table of a relational database for the entity *publication* with its specialization *book* and *proceedings* is defined at the IRD Level. The definition of the modelling terms in the Entity Relationship model are defined at the IRD Definition Level where in addition the terms for Structured Analysis models may be defined. The generic modelling terms, e.g. record types, are defined at the IRD Definition Schema Level, i.e. the modelling primitives used to represent the models at the IRD Definition Level.

4.1.1. Interlocking level pairs

Based on these four data levels, the IRDS standard promotes a so-called level pair architecture in which the rela-

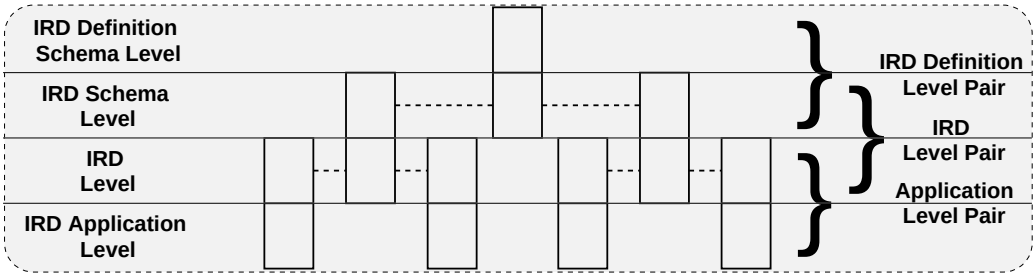

Figure 3 The three interlocking level pairs of the ISO/IEC IRDS framework.

tionships among the four levels are defined by a hierarchy of three database specifications on top of each other.

A level pair (figure 3 depicts four application level pairs, two IRD level pairs, and one IRD definition level pair) consists of two adjacent data levels. The upper level defines the *types* for the level below; it is called the schema for the lower level. At the lower level information is recorded as *instances* of the types defined at the level above. Hence a level pair can be interpreted as a database specification.

The IRD Definition Level Pair consists of the IRD Definition Schema Level and the IRD schema level. It contains the description of the dictionary schema as its instances, and a fixed way of defining such schemata as its own schema. This fixed way is defined in the IRDS Service Interface Standard (ISO 10728:1993) by a set of SQL tables, together with some basic features of the dictionary level pairs. In our example, these tables would contain tuples which say what, e.g. an *Entity* or a *Relationship* is.

The IRD Level Pair (often called dictionary) contains the schemata for the application level pairs as their data, and the definitions of the data models used for representing the application schemata as their own schema. For example, the dictionary (IRD level pair) may contain an Entity Relationship model of a library database, under a schema that defines the structure and constraints of Entity Relationship models.

The Application Level Pair corresponds to the actual applications to be integrated. For instance, the library database and various application programs running on it.

Based on the four data levels and the interlocking level pairs, the IRDS framework defines some facilities to be offered by an implementation of the IRDS standard, e.g., enforcement of constraints, access control, query and reporting facilities, etc.

4.1.2. IRDS and systems development processes

Each level pair can be associated with a particular kind of process and a particular class of people who use it:

- the Application Level Pairs with *usage processes* conducted by the *users*; e.g. an information system for lending books.
- the IRD Level Pairs with *development processes* carried out by the *application engineers*; the design of the information system.
- the IRD Definition Level Pair with the *change process* (sometimes called meta processes) executed by the method *engineers*; e.g. the definition of the methods used for developing the information system.

As a consequence of the level pair concept, only the three upper levels of the IRDS standard must be considered in a development environment. At the upper level, the concepts for expressing the methods and products are defined. Concrete product models are represented at the middle level and instantiated at the IRD level during process execution.

4.2. O-Telos and ConceptBase: An implementation platform for repositories

For implementing a repository, a language is needed in which the various models (schemata) can be formally represented. Such a language must provide the possibility to represent different level of abstractions and should offer all four kinds of abstraction mechanisms known from conceptual modelling: classification, generalization, specialization, attribution (aggregation). These mechanism can be used to represent the structure of a particular schema.

Beside the structural description, domain specific dependencies between objects must be represented. Therefore, the necessity of integrity constraints and rules was identified by various research contributions. By integrity constraints, the static dependency of the structure is defined, whereas rules are used to deduce dynamic dependencies between objects; dynamic in the sense, that a change of an object state leads to a change of an other object state whereas static dependencies assure that certain object properties can not be changed. A language should therefore provide the possibility of expressing domain specific integrity constraints and rules.

A third property needed for implementing a repository, is the dynamic clustering of objects (cf. [60]). Selection of "objects" should not only be based on their recorded memberships of classes, but also consider their current attributes and/or the actual attribute values. For example, there should be a possibility to retrieve all objects having a certain attribute or attribute value, independent of there actual class memberships. Short, query facilities are needed for application specific retrieval and selection of objects stored in the repository.

For the realization of a concrete repository, an implementation of the language must be available which is able to deal with a large amount of information in a consistent way and provide the above mentioned features.

The above mentioned characteristics to be met by an implementation platform, suggests to look for suitable languages and implementations within the area of knowledge representation or database technology.

One possibility for implementing a repository is to use relational database technology. As suggested by the IRDS framework, each interlocking level pair can be defined by a relational database system. The four modelling levels would then be defined as three database systems, which are defined at top of each other (cf. section 4.1). Another possibility, is to use an object oriented database system. One the one hand, both alternatives provide the needed query facilities and are excellent suited for recording large amounts of information. Moreover, the first alternative offers the advantage of widely available technology. But on the other hand, both alternatives have a very strong disadvantage: static as well as dynamic domain specific dependencies between objects can neither be expressed within a particular modelling level nor across different levels.

Because the consistency of the repository is an important prerequisite for supporting concurrent engineering processes, we have decided to choose an implementation platform, which enables us to represent domain specific dependencies and thereby assure domain specific consistency for a repository. We have chosen the deductive and object oriented knowledge representation language O-Telos (cf. [34]) and its prototypical implementation ConceptBase (cf. [31]), since O-Telos and ConceptBase provide all the needed concepts mentioned above (the different abstraction mechanism, integrity constraints and rules, dynamic clustering).

O-Telos (cf. [34]) is an extension of Telos (cf. [43]). Telos itself is based on CML (Concept Modeling Language), a further development of RML (Requirements Modeling Language) developed at the University of Toronto (cf. [21, 22]).

4.3. Formal definition of product models at the IRD definition level

As elicited in section 3.1 the products produced during the concurrent engineering process must be formally represented and integrated. As mentioned above, the product models are defined at the IRD Schema level.

In the following we present two formal product models provided by our concurrent engineering environment. These models have been formalized in O-Telos. Since one of the syntax variants of O-Telos is a typed and directed graph, the graphics we use for the models can be understood as direct representations of a formal O-Telos knowledge base with logical semantics (cf. [34]).

4.3.1. Recording informal information

For recording informal information we provide a hypertext model which is based on a quasi standard, namely the DEXTER model (cf. [24]).

"The concept of hypertext is quit simple: Windows on the screen are associated with objects in a database, and links are provided between these object." [6, p. 17]

In other words, the underlying data structure of a hypertext can be defined as a graph, where the information (independent of the representation format used) is represented as nodes and the interrelation of the information is represented as links of the graph. Consequently, a hypertext consists of two disjunct objects, namely *HT-Nodes* and *HT-Links*. These two concepts are refined by our hypertext model depicted in figure 4.

Atomic nodes (*HT-Atomic*) are primitives which can contain arbitrary information (graphic, text, pictures, sounds, ...). The information of an atomic node is stored in a Unix file, which name is recorded by the attribute stored-in. We distinguish between atomic nodes which can be split into a set of new atomic nodes (*HT-ChangeableAtom*) and nodes which can not be split (*HT-StaticAtom*). Splitting a node is necessary, if only certain parts of its content should be related to other information.

Composite nodes (*HT-Composite*) are nodes which are composed of *HT-Node*s (represented by the attribute consists-of). A composite node can not contain itself, neither directly nor indirectly (assured by a constraint). We differentiate between composite nodes by which a certain order of arbitrary nodes is represented (*HT-Sequence*) and composite nodes (called *HT-Set*) which are used the represent an unordered interrelation of nodes using the attribute *HT-RefLink*. An instance of *HT-Sequence* is created if an arbitrary hypertext node is split into two or more parts (nodes). The original node is transformed into a *HT-Sequence* (a composite node) which inherits all the links of the original node and consists of the nodes created during the split action. The new created nodes are related

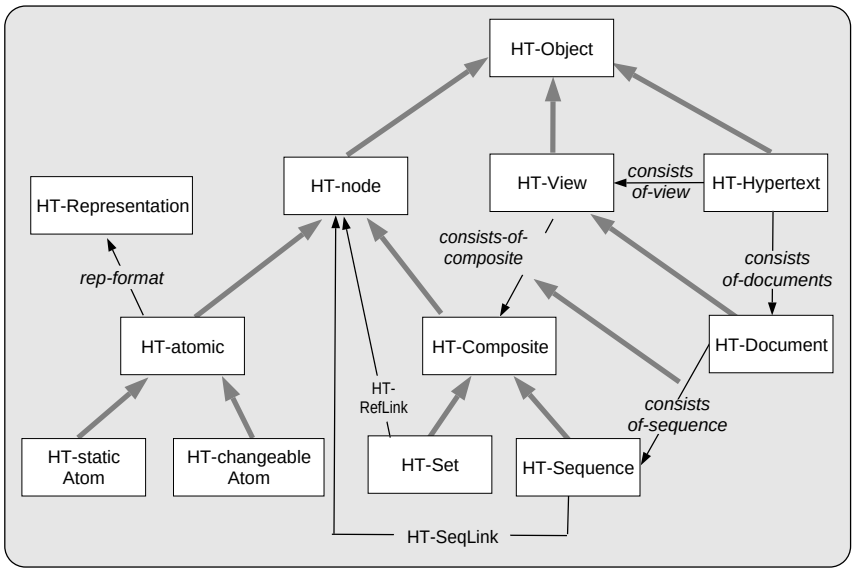

Figure 4 A formal hypertext model for recording informal informations.

to the sequence (representing the original node) using hypertext links of the type *HT-SeqLink*. In other words, splitting a node leads to the creation of a sequence, which guarantees that the new created composite node can be displayed as the original node. For example, if a protocol (which was entered in the system as a normal linear text) was split into several nodes to enable a detail interrelation with other representations, the *HT-Sequence* node, which represents the original protocol , enables to display or print the protocol in its original form, i.e. in the way it was entered in the system.

To be able to define different (disjunct) hypertexts the object *HT-Hypertext* is defined. A hypertext can consist of various *HT-Documents* and *HT-Views*. A *HT-Document* is always defined by a unique sequence, recorded by the attribute consists-of-sequence. Thus, a *HT-Document* can be used to represent normal textual documents in a linear manner, e.g. a specification according to the IEEE standard. In contrast, a *HT-View* can consist of *HT-Composite*, i.e. it can consists of *HT-Sequences* as well as *HT-Sets*. Therefore, among others, a *HT-View* can be created between different documents, e.g. between the *minutes* of a meeting and the results of an user *interview*. The introduction of these concepts goes along with the specification, that every *HT-atomic* node must be related to exactly one document, and that every *HT-Composite* must be related to either a *HT-View* or a *HT-Document* (modelled as constraint). This assures, that there exits no isolated instantiation of *HT-Node*.

As mentioned above, a hypertext node can contain all kind of information independent of the representation format used. Since, the action which can be applied to a hypertext node depend on the representation format, the attribute *rep-format* is used to record the kind of representation stored within a node. Currently we provide five representation formats (not shown in figure 4), namely *HT-Text, HT-Graphic, HT-Table, HT-Animation,* and *HT-Sound*. Depending on the representation format of the hypertext node, the actions for displaying and modifying the node are specialized. These actions are defined in [25].

Summarizing, a formal structure for informal information was defined by the hypertext model outlined above. Recording informal information using this structure, essentially enables the interrelation of informal information (like natural language text, graphics, sounds, etc.) to other products, e.g. entity relationship diagrams.

4.3.2. Recording the data view of an information system

For recording the data view of the system we provide an Entity Relationship (ER) model, which was defined according to the extended ER model proposed by Spaccapietra [65]. In addition to the ER model proposed by [5], our extended model provides the concepts of specializations of entities (together with the inheritance of the attributes) and the association of roles (*ER-Role*) and cardinalities (*ER-Card*) to the relationships (*ER-Relation*). Moreover, attributes (*ER-Attribute*) can be defined for relationships and entities (*ER-Entity*) and each entity an attribute can be defined by a data type (*DD-Type*). An Entity Relationship Diagram (*ER-Model*) consists of entities (*ER-Entity*) and relationships (*ER-Relation*) together with their attributes (*ER-Attribute*), cardinalities (*ER-Card*), roles (*ER-Role*), and data types (*DD-Type*). The graphical representation of the O-Telos model is depicted in figure 5.

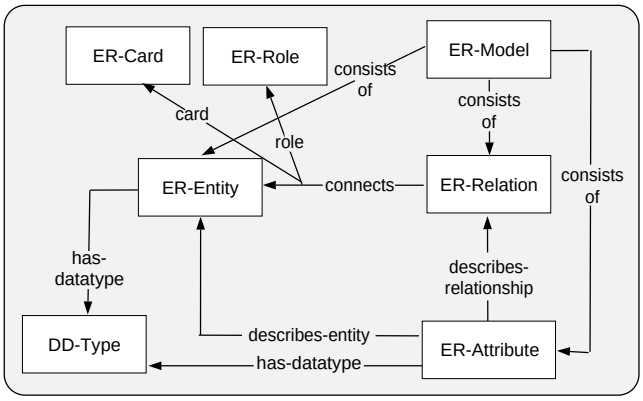

Figure 5 The Entity Relationship (ER) model

4.3.3. Assuring consistency of possible instantiations

We assure consistent instantiation be defining deductive rules and integrity constraints, which are checked within the implementation of O-Telos, every time an object is added or delete within the knowledge base. As an example for the definition of such constraints, which focus on the object *ER_Relation*, for which the frame notation of O-Telos is shown below.

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

Examples of integrity 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.

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

4.3.4. Other product models

We currently provide over ten product models for recording the information within a concurrent requirements engineering process. These models include a comprehensive specification model gained out of over 20 standards and guidelines, which is used to represent specifications as a basis for contracts, a decision model, which is based on the IBIS approach, and various semiformal (graphical) models including the model for Structured Analysis (SA) and the Object Oriented Modelling Technique (OMT).

Describing all these models goes behind the scope of this paper (cf. [48] for details).

4.4. Recording interrelations of products

As mentioned in section 3.1 formal product models are essential for enabling the recording of the interrelations of products. Since the products are recorded using the formal product models defined in the last section, we are able to record relationships between them. In the following we outline how such dependencies between products are represented.

A dependencies is described by five attributes. The *Action* by which the dependency was created is related to the dependency by the *created_by_RE* attribute. The type of the dependency is defined by the attribute *dependency_type*. Therefore, dependency types are itself object classes, which are organized in specialization hierarchy. The definition of our dependency hierarchy, which is described in [25, 48], is based on a comprehensive literature survey (e.g., *requirements engineering* [10, 14, 57, 56, 19, 22, 62], *text analysis* [42, 61], knowledge acquisition [45, 44], *recording design rationale* [7, 8, 57, 15], *hypertext* [11, 12, 3, 2, 16, 4, 67, 33, 17, 27, 39, 63, 68, 9, 35, 64, 69]).

The objects between which the dependency can be created are defined by the attribute *from_obj* and *to_obj*. In addition, an ranking can be defined for the dependency for which *temporary* is the default value.

The above described generic dependency type is defined in O-Telos as:

```
Product!has_dependency, MetaClass with
  created_by, necessary, single
     created_by_RE   : Action
  has_dep_prop, necessary, single
     dependency-type : RE_Dependency_Type;
     from_obj        : RequirementsObject;
     to_obj          : RequirementsObject;
     ranking         : Rank := temporary
  rule
     from_obj_rule :
       $ forall re-obj/RequirementsObject
           From (re-obj, this)
            ==> this from_obj re-obj  $
     to_obj_rule :
       $ forall re-obj/RequirementsObject
           To (re-obj, this)
            ==> this to_obj re-obj  $
end.</TelosFrame>
```

More specific dependency link types can defined as specialization of the generic dependency link type by

(1) restricting the allowed objects (e.g. to entities and hypertext nodes)

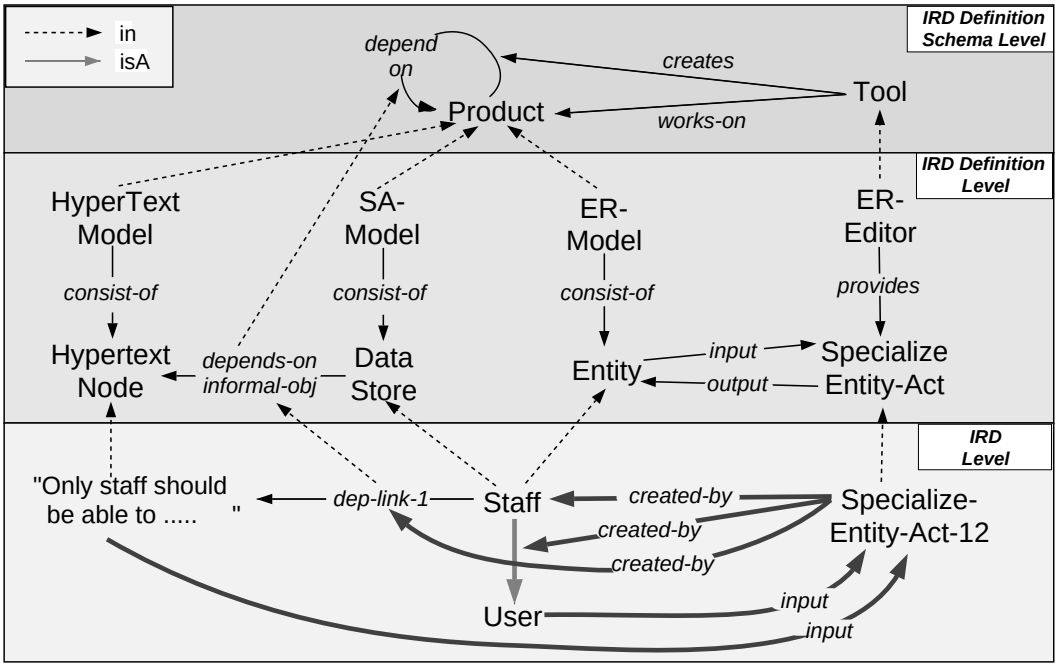

Figure 6 An (simplified) example of product instantiation and interrelation

(2) by defining a special dependency type of our dependency taxonomy (cf. [48] for the definition of this taxonomy which consists of fifteen dependency types)
(3) a combination of (1) and (2) and
(4) by defining further semantics of the dependencies by adding constraints and rules, e.g. restricting multiple instantiation for the objects or prohibiting the association of other dependency links to objects.

However, these interrelations have to be created an maintained by the process execution environment.

4.5. The process execution environment

Due to the large amount of information capturing trace information must be as far as possible automated. Therefore, the task of capturing the information and recording them in the predefined structure must be accomplished by the process execution environment, i.e. by the process centered engineering environment used for performing the process. Every computer based environment provides a fixed set of actions, by which the product is created, deleted and modified. This actions are mostly implemented by certain programs (modules, procedure); i.e. these actions are defined using a particular programming language.

The environment must assure, that beside recording product evolution (according to the defined formal product models) also the process execution and the relations to the product evolution are adequately recorded. Therefore, each action performed, together with the input used and the output produced, must be recorded and related to the product traces. Beside relating the process action to the product objects, the performance of an action can also lead to the creation of dependencies between objects.

We therefore have to extend the schemata provided at the IRD Definition level by the notions of tools (e.g. *ER-Editor*), the action they provide, and their possible inputs and outputs. The input and output of the action consists of the product parts and the dependencies between them. The execution of an action is then recorded as an instance of the defined action type (cf. figure 6) and the actual inputs and outputs are related to the action.

Taking the example from above, we now see, that the entity *Staff* was created by the *SpecializeEntity-Act-12* as a specialization (denoted by the *isA* link) of the entity *User* and that the action has used the entity *User* and the hypertext node *"Only staff should be able to ..."* as input. Moreover, two dependencies have been created between the trace objects: one dependency of the type *depends-on-informal-obj* between the object *Staff* and *"Only staff should be able to ..."*, labeled *dep-link-1* (cf. figure 6); the other between the entity *User* and the entity *Staff* of the type *depends-on-ER-object* (not shown in the figure 6).

Beside the recording of products as instantiation of the formal product models, the tool environment enables the engineer to browse between the different products. For example, using our tool environment the engineer is able

to retrieve all related objects of the entity *staff*. Using a query, the recorded interrelation *dep-link-1* is retrieved. Using our tool environment, the requirements engineer can now follow the interrelation which leads him to the sentence *"only staff should be able to ...."*.

Adding to complexity, in most cases for each kind of product a special tool is used, e.g., a textual editor for textual documents, an 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 the browsing between different products. In our example, for following the link from the entity *staff* to its destination (which is part of a textual document) the entity relationship editor must "ask" the hypertext editor to display the destination of the link, i.e. to highlight the part of the informal description which is related to the entity *staff*. This cooperation of the various tools is enabled by providing a communication manger, which trades the service requests of the tools (like highlighting of objects).

Thus, our tool environment helps the concurrent engineer in instantiating the formal product models, capturing and maintaining their interrelations during process execution, and enables the engineer to browse through the interrelated products using the recorded dependency links.

4.6. Resume

In this section we have outlined PRO-ART, our approach for enabling traceability between cross functional teams. Based on the IRDS standard we have proposed a repository, in which the formal product models are defined. The product produced are then recorded as instantiations of these models. The formal models at the IRD Definition level have been extended by formal tool models, which enable the automatic recording of the product produced during the process as well as their interrelations. Based on these interrelations, the environment supports the environment by browsing between the different products.

However, a suitable visualization of the recorded interrelations between the products is needed.

5. CoDecide: Supporting mutual understanding

So far we have shown, how traceability can be reached by interconnecting different views with dependency-links. However, these links are not a sufficient representation to enable cooperation and mutual understanding. A clear means to serve as an information forum has to be found.

In this chapter the idea of Quality Function Deployment (QFD) and the related form, the House-of-Quality (HoQ) is briefly outlined. We show that the ideas behind QFD can be generalized and be used for interconnecting different views and present them sufficiently to the users. The generalized HoQ is formalized and integrated into the formal framework discussed in chapter 4 using the idea of interconnecting products by a common presentation. Finally we show, how automatic mapping from the ER-model adapted from section 4.3 into the generalized HoQ presentation can be achieved using deductive queries.

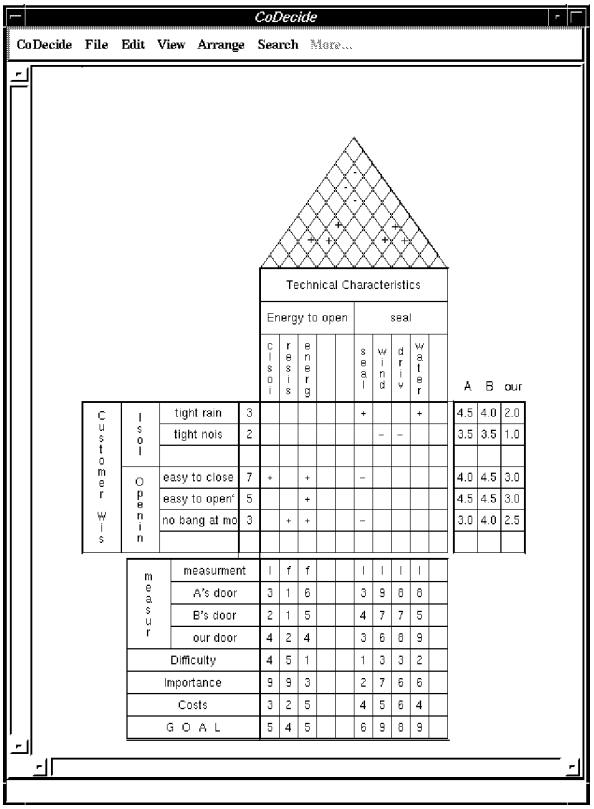

Figure 7 The House of Quality [26]

5.1. Quality Function Deployment

Quality Function Deployment has been successfully employed for about 20 years [26]. Since then, several joint ventures between Japanese and American companies have disseminated the method. The American Supplier Institute develops a variant of QFD that fitted into the traditional product development process. In particular, the quality table proposed by Akao in 1983 were changed into so-called houses of quality. mirroring the phases to be taken (cf. fig. 7). The goal of QFD is to ensure that the customers voice drives all actions concerned with quality. With the customer requirements at the basis of the whole process, technical characteristics, component

characteristics, process plan, and operation instructions are being deployed in four phases. To achieve this aim, the employes strive to come to an agreement concerning their actions. This fact is an important concept of the philosophy underlying QFD.

Focussing on the mapping of customer wishes into technical characteristics, i.e. the first phase of QFD, the two groups are e.g. the marketing representing the customers and a team of designer responsible for the technical characteristics. The main steps of QFD can be briefly described.

First, the customer wishes are filled in the left matrix. A treelike structure enables the definition of subwishes. Additionally weights can be offered to express the different importance of the customer wishes.

Second, technical characteristics are offered by the designer into the matrix on top of the central matrix. The characteristics can again be treelike structured.

Third, as the single technical characteristics are not independent but can correlate positively or negatively, the roof of the HoQ can be used to describe these interdependencies.

Fourth, the two groups together fill out the middle-matrix of the HoQ, i.e. they describe how the single customer wishes and the technical characteristics correlate.

Fifth, additional information can be offered in the matrix right of the HoQ and beneath the correlation matrix. These matrices contain information about competitive product in comparison to the own one.

Sixth, the technical characteristics are described in more detail by adding their costs and difficulties.

At last the relative importance of each characteristic has to be added. The goal now is, to decide which characteristics have to be reached to which degree. In a certain sense, the technical characteristics can be regarded as a set of alternatives which have to be fulfilled to a certain degree. The customer wishes serve as criteria which drive the decision process.

The success of QFD is mainly based on the intuitive usable and understandable form, the House-of-Quality (cf. figure 7). The whole QFD-process can be regarded as a complex decision process performed by at last two groups. Information gathering and analyzing is systematically performed, dependencies between the two different groups are explicitly mentioned. All in all, cooperation between different groups is reached by gaining mutual understanding.

In the following we generalize the idea and concepts of QFD to adopt their advantages in nearly all parts of the production process.

5.2. Interconnected products by a common presentation

The HoQ brings marketing and design face-to-face. The different views of marketing and design are interconnected by the correlation matrix. However, interconnecting two different views is a general phenomena in design. Typical examples are e.g.

- Validation: Two different representations, e.g. code and specification, are compared to look if there present the same thing. In this case the code and the specification represent two views, and the successive comparison interconnects the two views.
- Negotiation: The different criteria by the negotiation opponents can be regarded as views on the possible alternatives which form another view. The views are interconnected by the negotiation process.
- The whole Production Process: The single phases within the production process can be regarded as different, interconnected views on the production. E.g. marketing represents the customer's view, the analysts have the requirements view, etc. In sequential engineering there are interconnections between two following phases.

In comparison to traditional engineering, interconnections between different views are much more often in concurrent engineering (cf. chapter 2). As discussed before, these interconnections have to be made explicit in a suitable form.

Different products, e.g. hypertext or entity relationship diagrams, are presented to the user using a specialized presentation. These presentations are well suited for a particular product, but due not display the interrelations to other products. Thus a new *cooperative presentation* is needed to visualize interconnected products.

The generalized HoQ is a means to serve as the common presentation. The question arises, what the general HoQ looks like, i.e. how the *cooperative product* can be refined, how it can be formalized, and how the mapping rules to other products look like.

5.3. CoDecide: The generalized House of Quality

Within this section the main concepts of the generalized HoQ shall be informally motivated as a basis for the formal model in section 5.4.

Looking at various QFD variants using all slightly different HoQs, we have identified three different concepts. All concepts have different typical methods for modification [18].

- Fields: The field is the smallest unit in the HoQ. Each field contains a single string explaining a single

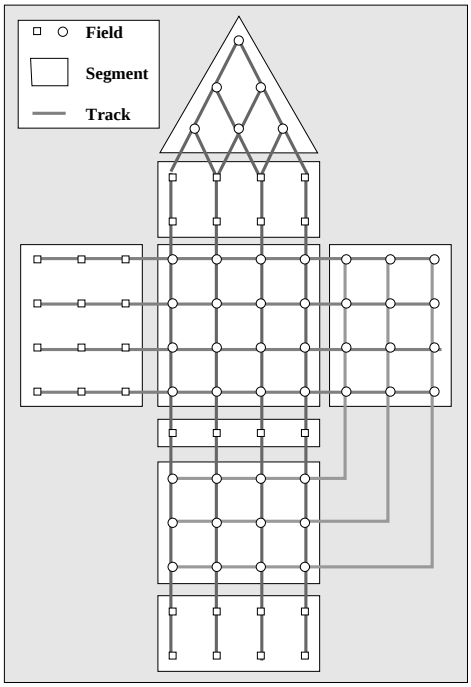

Figure 8 Concepts underlying the HoQ

customer wish, a correlation, a technical characteristic, etc. The attributes belonging to a field are the used shape, font, the fore- and background color etc. Methods executed on fields are the updating of the inserted strings.

- Segment: Fields are collected into segments, the so-called matrices. The matrices differ in their inner structure and shape. All customer wishes are e.g. collected in a segment offering a hierarchical structure. The correlation matrix offers identical fields. The roof, a special half-filled matrix is composed of diamonds. Attributes belonging to a segment are e.g. the inner structure (hierarchy, flat). Typical methods executed on segments are e.g. the export/import of product descriptions. There are two types of segments. The first one like the matrix for the customers represent a single view. The second one like the correlation matrix or the roof connect views. The view-matrices lie in only one dimension, whereas the connecting-segments lie in two dimensions.
- Tape: The dimensions are implemented in a concept called the tape. Tape glue single matrices. E.g. In the HoQ there are three tapes. The first one couples the "horizontal" matrices, i.e. the customer wishes, the correlation matrix, and the competitive products. The second one, combines the horizontal matrices, i.e. the roof, the technical characteristics and the correlation matrix, the comparison of competitive products and the estimation of the single technical characteristics. The last tape relates the two matrices dealing with competitive products. There are four segments which are two-dimensional: The correlation matrix, the two matrices with the competitive products, and the roof, which ties up the vertical tape with itself. (cf. figure 8)
  Each tape consists of several tracks. Whereas tapes combine segments, the track combines the fields of the single segments. Typical methods performed on tapes and tracks are insertion, deletion and reorganization of tracks. As these operations are performed on the tracks and not on the fields, structural consistency between the segments is guaranteed.

These concepts can be generalized in the following way. The segments serve as building blocks and the tapes serve as the gluing connection. So, various houses can be built, each well suited for a specific application.

5.4. A formal model for the generalized House of Quality

In this chapter the concepts of segment, tape, and field are formalized and interrelated to the products. The generalized HoQ, i.e. different kind of *Cooperative Presentations* formally *consists_of* the above mentioned concepts of tape and segments and their inner structure field and trace. There are mainly two kind of segments: the one dimensional and the two dimensional. According to their duty they are named *Product Segment* (one dimensional) and *Interrelating Segment* (two dimensional). As their name indicate product segments shall present different kind of products whereas interrelating segments focus on the dependency links between the products. Details of the model, e.g. the different kind of structures (flat, hierarchy), the different kind of interrelating segments (matrix, triangle) and the attributes are dropped for simplification.

5.5. Automatic Transformation

As the formal model of the HoQ is formalized within the repository, the formal structure provided, can be automatically instantiated with other parts of other products. For example, a segment for presenting all entities can be deduced by the following O-Telos frame.

```
MetaClass ER-list-segment isA Product
   rule
    instance-rule :
      $ forall ent/ER-Entity
        ==> ent in ER-list-segment   $
```

Moreover, these structures are automatically maintained, i.e. every time an entity is added or removed from the repository, the instances of the *ER-list-segment*

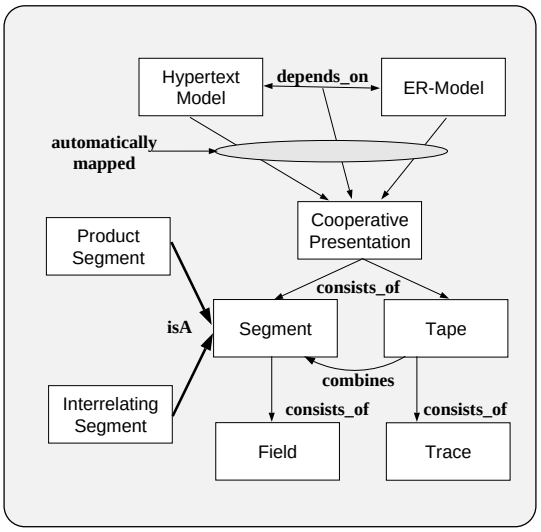

Figure 9 A formal model for the generalized House of Quality

are changed accordingly by the deductive component of our repository.

Deductive queries provide another possibility to transform the existing products into views presented to the user using the House of Quality. In contrast to the deductive rules, the views are created every time the query is answered, i.e. they are not recorded explicitly, but are computed every time the query is answered. Thus, the designer of an environment can choose on of the two alternatives, dependent on the usage, i.e. if a view is often being used, he should model this view using deductive rules, whereas in the case of sporadically used views should be defined using deductive queries.

Due to the presentation using the HoQ the formal model underlying these presentation is restricted to treelike structures, which is the mostly used principle for global structuring. Free structures, e.g. used in hypertexts, suffer from a lack of clarity which leads to effects well known as "lost in hypertext". Reducing the complexity of presentation leads to a simplification which just enables cooperation.

5.6. Resume

Based on the PRO-ART environment, which enables traceability by interrelating products through dependency links, we have outlined an approach to present these dependencies in an intuitive understandable form. Interfaces designed in the flavor of the HoQ serve as a cooperative information forum. The formal model behind the generalized HoQ enables to automatically map of the different products into the common representation model.

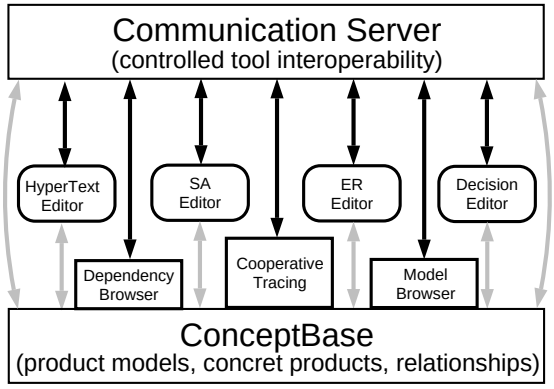

Figure 10 Architecture of our concurrent engineering environment

As the presentation is intuitively understandable it is well suited to serve as means for communication, negotiation, and mutual understanding .

6. Implementation issues

The architecture of our concurrent engineering environment is shown in figure 10. A detailed description can be found in [51, 52, 48]. The environment was implemented using C and C++. The graphical user interfaces were build using X11 R5 and the Andrew toolkit as well as the Tcl/Tk toolkit. To enable the use 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 O-Telos [43, 34] and its implementation ConceptBase Version 3.2 [31]. Currently the whole prototypical environment consist of more than 120.000 lines of source code. The formal product and tool models cover more than 250 class definitions.

7. Example[1]

We now pick up the example introduced in chapter 2. Several iterations of phase II and III have taken place. As the concurrent engineering team uses our environment, the different products (hypertext, ER-diagram, decision, etc.) are interrelated, i.e. dependencies which exist between the products are explicitly recorded. For example, the definition of the entity *book* and the informal requirement which led to the creation of attributes of the entity have been interrelated.

[1] This example focuses on the use of recorded interrelations. A detailed example for recording interrelations during process execution can be found in [48, 53, 52].

7.1. Integration of a new requirement using the HoQ

At a certain time, the products which have been improved in parallel, are again integrated, i.e. phase II is entered again. Thus, a new requirement stated by the customer *"Journals should not be checked out"*, recorded as informal text within the hypertext document, must be integrated with the other existing products, e.g. with the entity relationship model. This requirement was stated by the customer as an addition to the allready existing requirement *"Book loan duration = 10 days"*. Moreover, the two informal requirement were iterrelated using a hypertext link.

The new requirement should now be integrated with the other products. Therefore, the new requirement is loaded into CoDecide, as shown in figure 11. Since the requirement was related to another informal requirement, this related product part was also loaded into CoDecide. In other words, a special tool modus of the CoDecide tool enables the concurrent engineer to load a part of a product together with related parts of the same product. Moreover as depicted in figure 11, the existing interrelations are displayed within the triangle. As shown in the figure (for clarity, the interrelations are numbered), the exisiting relation (hypertext link) between the requirement *"journals should not be checked out"* and the requirement *"book loan duration = 10 days"* is displayed (number *1* in the triangle). In addition, all informal requirements which are interrelated with the requirement *"book loan duration = 10 days"* were loaded, as shown in figure 11 (relation number *2,3,4,5,6,7,8,9*).

Summarizing, the new informal requirement together with other related informal requirements was loaded into CoDecide.

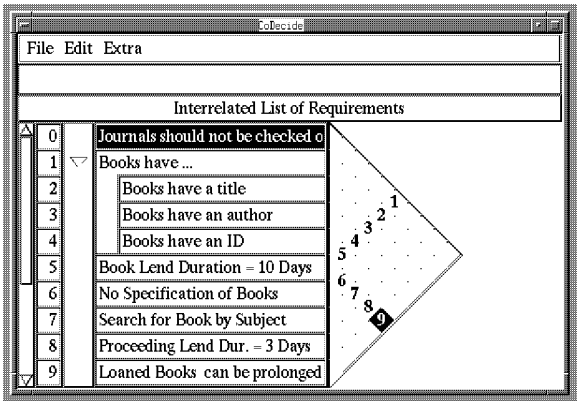

Figure 11 Displaying interrelated informal requirements

7.2. Displaying related product parts using interrelations

The concurrent engineer has now the possibility to retrieve product parts of other products which are related to the displayed information. Since, the requirement *"book loan duration = 10 days"* was directly interrelated with the new requirement *"journals should not be checked out"*, the concurrent engineers want to look at all parts of other products which are related to this requirement. Based on the recorded interrelation, CoDecide is able to retrieve the related products parts using the query `get-related-objects` with the hypertext node representing the informal requirement as actual parameter.[2]

```
GenericQueryClass get-related-objects isA
 ProductPart with
  paramter
    product : ProductPart
  constraint
    dep-constr:
      exist x/dependency
    $ this x product or
      product x this  $
end.
```

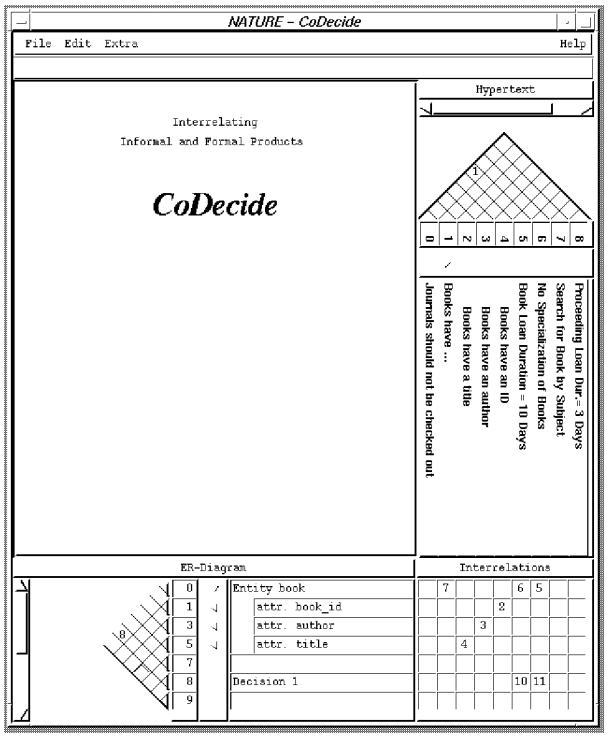

Figure 12 Displaying interrelated products parts of other products.

[2] `this` within the query class refers to the objects which are recorded as instances of the query, i.e. the answer to the query consists of all product parts which are interrelated with the product part specified as parameter of the query.

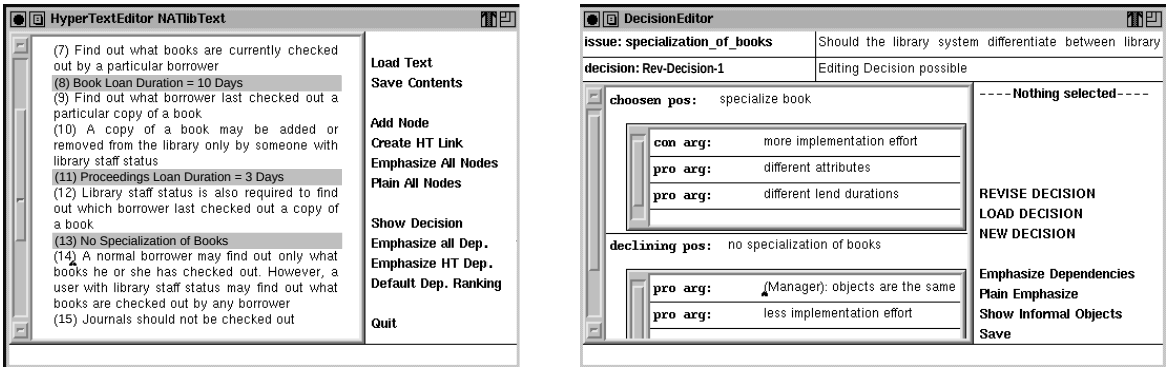

Figure 13 Revison of the decison using the PRO-ART environment.

The result of this query is displayed within CoDecide. As figure 12 depicts, CoDecide has changed its presentation. Now the informal requirements are displayed on top of the window, whereas the retrieved product parts, e.g. the entity *book* are displayed in horizontal positions. Beside the product parts, also existing interrelations are shown. On one side, these interrelations exist between the retrieved product parts (cf. left triangle of figure 12), e.g. the relation with the number *8*. On the other side, the relations to the informal requirements are displayed, e.g. the relations with the number *1*. Looking at the information the engineer recognizes, that the informal requirement was considered in *decision 1* (presented by the relation number *10*). Moreover, looking at the other relations of *decision 1* with informal requirements, the engineer detects, that there is a relation *11*, which relates the decision to the requirement *"No Specialization of Books"* and a relation *12*, which relates the decision to the requirement *"Proceedings Lend Duration = 3 Days"*.

Based on the presentation of the product interrelations in CoDecide, the concurrent engineering team have now detected a conflict. There are three requirements which implicitly suggest a specialization of the entity book, namely the new requirement *"journals should not be checked out"*, the requirement *"Book Lend Duration = 10 Days"*, and the requirement *"Proceedings Lend Duration = 3 Days"*. These requirements were detected based on the existing product interrelation. Moreover, the requirement *"No Specialization of Book"* and *decision 1*, which has introduced this requirement, were elicited.

Now the team has to decide, if a specialization of books should be made, or if the new requirement should be rejected.

The team decides that a specialization of books should be performed, which considers the different lend durations.

7.3. Integration of decisions (Phase III)

After phase II is finished, the engineers begin to integrate the decisions made in the last meeting into their products. First, they capture the decisions made during the meeting. Using the decision editor, they revised the old *decision 1* (which was about no specialization of the entity *book*). Thereby, the informal requirements are related to the new decision (cf. figure 13). After that, the system analyst specializes the entity book according to the informal requirements. The result of these specializations is shown in figure 14. Now, there are three subentities of book, namely *textbook*, *proceedings*, and *journal* each with a different lend duration.

Integrating these changes (revision of the decision *decision 1* and specialization of the entity *book*) has led to new interrelations of the product which were (almost automatically) created by the PRO-ART environment.

Figure 15 depicts the resulting dependencies between the new decision (*decision 2)* and the ER objects and the old decision (relations with number *3,4,5,6,7*) as well as

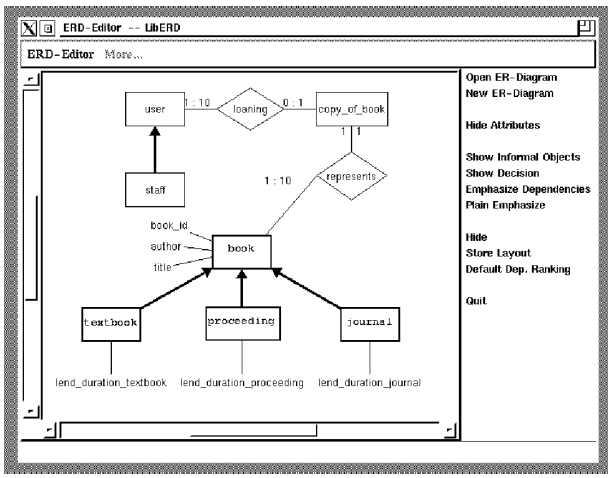

Figure 14 Specialization of an entity

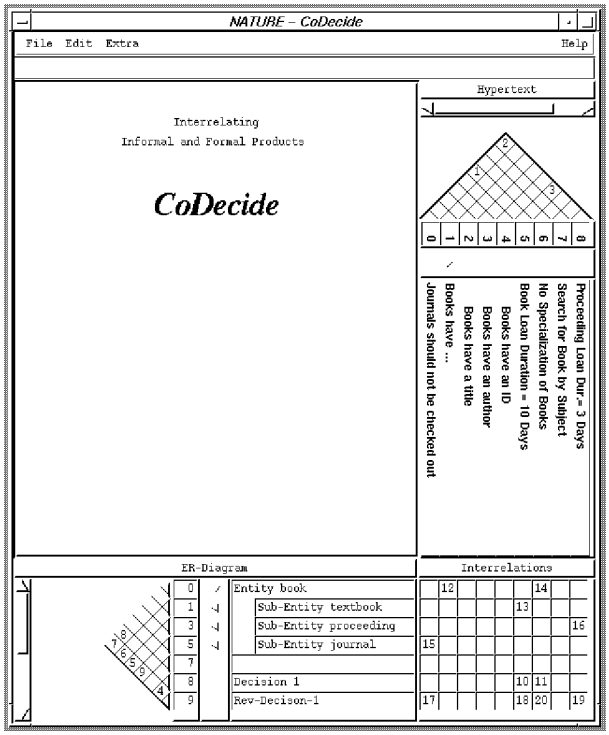

Figure 15 New created product interrelations.

between the new decision (*decision-2*) and the informal requirements (relations *17,18,19,20*).

Summarizing, the example has focused on the use of recorded interrelations of the various products which coexist in the concurrent engineering process. These interrelations are presented using CoDecide. As the example shows, the clear presentation provoked communication and therefore supports mutual understanding.

Further the example illustrates (without going into details), that the revision of the decision and the specialization of the entity *book* made using the PRO-ART environment led to the automatic creation of a whole set of product interrelations (as shown in figure 15). These interrelations have been created by the tools of the PRO-ART system, based on the formal tool definitions.

Hence, our environment supports the engineer in interrelating the various products and thereby enables traceability between the product, respectively their parts. Moreover, mutual understanding is supported by presenting the existing interrelations in a suitable way which provokes communication.

8. Summary

We defined three different phases of concurrent engineering processes. Within the first phase, the persons (teams) involved work in parallel and use different products to represent their views on the system. In the second phase, the products (views) are interrelated as a result of conversations and cooperations. Thereby similarities and dissimilarities are made explicit by representing relations between the various products. Moreover, decisions are made, by which the dissimilarities are resolved. In the third phase, people work in parallel again, but in contrast to the first phase, the existing interrelations must be considered and maintained. After the third phase an arbitrary number of loops, consisting of second and third phase, are performed, until the defined product fulfills the expectations. Based on the three phases we have elicited the requirements needed for enabling traceability between the various products and for supporting mutual understanding.

For enabling traceability between the various products, we have proposed PRO-ART, an approach for enabling traceability between cross-functional teams. Within the repository the products of the process are recorded as instantiation of formal product models. Additionally, the possible interrelations between the products are formally defined in the repository. Moreover, the tools constituting the environment have been based on formal tool models. For each action, both, the possible inputs and outputs, as well as the kind of interrelations to be created between the inputs and the outputs are defined. Therefore, PRO-ART is able to record the products produced as instances of the defined formal product models and to maintain and create product interrelations. Moreover, it supports the engineers by browsing through the product following the formal recorded interrelations.

For supporting mutual understanding we have proposed CoDecide, an approach which is based on the ideas of the House of Quality (HoQ). The concepts of HoQ have been generalized and formally represented within the repository. Thereby, an automation of the transformation between the various products and the formal model of the HoQ has been enabled. This transformation enables the team to present parts of the products and their interrelations in a presentation, which provokes communication and coordination.

Based on the two approaches, PRO-ART and CoDecide, a prototypical environment was implemented, which supports the engineers by

- recording the products using formal product models;
- detecting and recording product interrelations based on the formal models;
- maintaining and using the product interrelations;
- providing a suitable presentation of the interrelations which provokes communication and cooperation.

The prototypical implementation is currently being used in two settings. First, an evaluation takes place by applying the environment in requirements engineering.

a typical concurrent engineering setting, namely the requirements engineering process of software systems. As requirements engineering has suffers from different languages, from heterogenous teams, and a lack of integration of life-cycle-phases, it is a good candidate for concurrent engineering. First, premature experiences show, that the use of our environment enables traceability and supports mutual understanding and thereby leads to improved and more consistent process results (products).

Second, the implementation is adopted to another domain, namely factory layout planning. In this domain completely different product models are in use. We are currently developing formal models for corresponding products. Moreover, special interfaces as a means to enable customer participation are implemented using CoDecide. Difficulties are due to graphical representations of layouts with a great inherent complexity. The work done so far shows, that our approach is adoptable to other domains.

Acknowledgments

We like to thank Ralf Dömges for his valuable comments on a earlier version of this paper and all our students which helped us in implementing our prototype, namely: Bernd Adameit, Michael Gebhardt, Friedhelm Gibbels, Peter Haumer, Markus Hoofe, Christof Lenzen, Monika Pandey, Klaus Weidenhaupt, and Claudia Welter.

The work was supported by the ESPRIT Basic Research Action NATURE (#6353) and by the sate of Nordrhein-Westfalen, Germany (AI-Network (KI-NRW).

References

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

[2] James Bigelow. Hypertext and CASE. *IEEE Software*, pages 23–27, March 1988.

[3] T. Biggerstaff and R. Richter. Reusability Framework, Assessment and Directions. *IEEE Transaction on Software Engineering*, 13(2), 1987.

[4] P. Carando. Shadow: Fusing Hypertext with AI. *IEEE Expert*, 4(4):65–78, 1989.

[5] P.P.S. Chen. The Entity-Relationship Approach: Towards a Unified View of Data. *ACM Transactions on Database Systems*, 1(1), 1976.

[6] J. Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9):17–41, 1987.

[7] J. Conklin and M. J. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transaction on Office Information Systems*, 6(4):303–331, 1988.

[8] J. Conklin and K.B. Yakemovic. A Process-Oriented Approach to Design Rationale. *Human Computer Interaction*, 6(4):357–391, 1991.

[9] A. J. Cybulski and K. Reed. A Hypertext Based Software-Engineering Environment. *IEEE Software*, 9(3):62–68, 1992.

[10] 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.

[11] Norman M. Delisle and Mayer D. Schwartz. Neptune: A hypertext system for CAD applications. In *Proc. of ACM SIGMOD '86*, pages 132–142, Washington, 1986.

[12] Norman M. Delisle and Mayer D. Schwartz. Contexts – A Partitioning Concept for Hypertext. *ACM Transaction on Office Information Systems*, 5(2):168–186, April 1987.

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

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

[15] G. Fischer. Integrating Construction and Argumentation in Domain-Oriented Design Environments. In *Proceedings of the 1th Int. Symposium of Requirements Engineering*, page 284, San Diego, CA, 1993. IEEE Press.

[16] Pankaj K. Garg and Walt Scacchi. On Designing Intelligent Hypertext Systems for Information Management in Software Engineering. In *Proceedings of Hypertext '87, November 13–15, Chapel Hill, North Carolina*, pages 409–432, 1987.

[17] Pankaj K. Garg and Walt Scacchi. A Hypertext System to Manage Software Life-Cycle Documents. *IEEE Software*, pages 90–98, May 1990.

[18] Michael Gebhardt. Kohärentes Design durch Sichtenkopplung. Master's thesis, RWTH-Aachen, Germany, 1994. (in German).

[19] O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proc. of the 1st International Conference on Requirements Engineering (ICRE'94)*, pages 94–102, Colorado Springs, Colorado, April 1994. IEEE Computer Society Press.

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

[21] S.J. Greenspan. *Requirements Modeling: A Knowledge Representation Approach to Software Requirements Definition*. PhD thesis, Dept. of Computer Science, University of Toronto, 1984.

[22] Sol Greenspan, John Mylopoulos, and Alex Borgida. On Formal Requirements Modelling Languages: RML Revisited. In *Proc. of the 16th International Conference on Software Engieneering*, pages 135–148, Sorrento, Italy, May 1994. IEEE Computer Society Press.

[23] Robert Grob, Stephan Jacobs, and Stefanie Kethers. Towards Cooperative Information Systems in Quality Management - Integration of Agents and Methods. In *Proceedings of the 2nd International Conference on Cooperative Information Systems, CoopIS, Toronto*, 5 1994.

[24] F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. In J. Moline, D. Benigni, and J. Baronas, editors, *Proc. of the First NIST Hypertext Standardization Workshop*, pages 95–133, 1990.

[25] Peter Haumer. A Hypertext System for Structuring and Integrating Informal Requirements. Master's thesis, RWTH-Aachen, Germany, 1994. (in German).

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

[27] M. Hofman, U. Schreiweis, and H. Langendörfer. An Integrated Approach of Knowledge Acquisition by the Hypertext System CONCORDE. In *Proc. of the European Conference on Hypertext (ECHT-90). Hypertext: Concepts, Systems and Applications*, pages 166–179, Paris, France, October 1990. IEEE Computer Society Press.

[28] 10027 ISO/IEC. *Information Technology – Information Resource Dictionary Systems (IRDS) – Framework*. ISO/IEC International Standard, 1990.

[29] Stephan Jacobs. Methodenorientierte Entwicklung von CSCW. In *Proceedings of the D-CSCW 94, Marburg*, 9 1994. (in German).

[30] Stephan Jacobs and Stefanie Kethers. Improving Communication and Decision Making within Quality Function Deployment. In *Proceedings of the Conference on Concurrent Engineering, Research and Applications, CE94, Pittsburgh, Pennsylvania*, pages 203–212, 8 1994.

[31] Matthias Jarke, Stephan Eherer, Rainer Gallersdörfer, Manfred Jeusfeld, and Martin Staudt. ConceptBase – A deductive Object Base Manager. *Journal on Intelligent Information Systems*, 1994. also appeared as technical report, RWTH-Aachen, Aachner Informatik Berichte, No. 93–14.

[32] 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.

[33] Sahar Jarwa and M.-F. Bruandet. A Hypertext Database Model for Information Management in Software Engineering. In *Proc. of the Int. Conference on Database and Expert Systems*, pages 69–75. Springer Verlag, 1990.

[34] Manfred Jeusfeld. *Change Control in Deductive Object Bases*. INFIX Pub, Bad Honnef, Germany, 1992. in German.

[35] H. Kaindl. The Missing Link in Requirements Engineering. *ACM SIGSOFT Software Engineering Notes*, 19(2):30–39, 1993.

[36] 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.

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

[38] 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.

[39] L. Lafferty, M. Koller, G. Taylor, R. Schumann, and R. Evans. Techniques for capturing expert knowledge: An expert systems/hypertext approach. In *Proceedings of Applications of Artifical Intelligence VIII*, pages 181–191, Orlando, Florida, 1990.

[40] 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.

[41] 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.

[42] William C. Mann and S. A. Thompson. Rethorical Structure Theory: A Theory of Text Organization. Technical Report ISI/RS-87–190, Information Science Institute, University of California, 1987.

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

[44] Susanne Neubert. Model Construction in MIKE (Model Based and Incremental Knowledge Engineering). In *Proc. of the 7th European Knowledge Acquisition Workshop (EKAW'93)*, Toulouse, France, 1993.

[45] Susanne Neubert and Andreas Oberweis. *Einsatzmoeglichkeiten von HyperText beim Software Engineering und Knowledge Engineering*, pages 162–174. Springer Verlag, 1992.

[46] 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.

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

[48] Klaus Pohl. *A Process-Centered Requirements Engineering Environment*. PhD thesis, RWTH Aachen, Germany, 1994. forthcoming.

[49] Klaus Pohl. REFSQ'94: Session Summary on Requirements Traceability. In *Proc. the First International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'94)*, pages 12–15, Utrecht, Holland, 1994. Augustinus Verlag.

[50] Klaus Pohl. The Three Dimension of Requirements Engineering: A Framework and Its Application. *Information Systems*, 3(19):243–258, June 1994.

[51] 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 *ICSE-16–Workshop on Research Issues in the Intersection*

Between Software Engineering and Artificial Intelligence. University of Oregon, Technical Report, 1994.

[52] Klaus Pohl, Ralf Dömges, and Matthias Jarke. PRO-ART: Process Based Approach to Requirements Pre-Traceability. Technical Report 94-7-1, NATURE Report Series, RWTH-Aachen, Germany, May 1994. CAiSE'94 – Poster Outlines, University of Twente.

[53] Klaus Pohl and Stephan Jacobs. Traceability between Cross-Functional Teams. In *Proceedings 1st International Conference on Concurrent Engineering*, pages 223–234, Pittsburgh, PA, 1994. IEEE Press.

[54] Klaus Pohl, Gernot Starke, and Peter Peters, editors. *Proceedings of the First International Workshop on Requirements Engineering: Foundation of Software Quality*, Utrecht, Holland, 1994. Augustinus Verlag.

[55] C. Potts and G. Bruns. Recording the Reasons for Design Decisions. In *Proceedings 10th International Conference on Software Engineering*, 1988.

[56] B. Ramesh. A Model of Requirements Traceability for Systems development. Technical report, Naval Postgraduate School, Monterey, California, September 1993.

[57] B. Ramesh and V. Dhar. Process-Knowledge Based Group Support in Requirements Engineering. *IEEE Transactions on Software Engineering*, 18(6), 1992.

[58] B. Ramesh and M. Edwards. Issues in the Development of a Requirements Traceability Model. In *Proceedings of the 1st Int. Symposium on Requirements Engineering*, San Diego, CA, January 1993. IEEE Press.

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

[60] Thomas Rose, Matthias Jarke, and John Mylopoulos. Organizing Software Repositories: Modelling Requirements and Implementation Experiences. In D.B. Simmons, editor, *Proceedings of the 16th International Computer Software & Applications Conference CompSAC'92*, Chicago, IL, September 1992.

[61] Dietmar Rösner and M. Stede. Zur Struktur von Texten — eine Einführung in die Rhetorical Structure Theory. Technical Report FAW-TR-93005, FAW Ulm, 1993. in German.

[62] A. Salek, G. Sorenson, J.R. Tremblay, and J.M. Punshon. The REVIEW System: From Formal Specifications to Natural language. In *Proc. of the 1st International Conference on Requirements Engineering (ICRE'94)*, pages 200–229, Colorado Springs, Colorado, April 1994. IEEE Computer Society Press.

[63] W. Schuler and B. Smith. Author's Argumentation Assistant (AAA): A Hypertext-Based Authoring Tool for Argumentative Texts. In *Proc. European Conference on Hypertext (ECHT-90)*, pages 137–151, Paris, France, 1990.

[64] Helge Schütt. *Datenbankserver-Unterstützung für kooperative Hypermediasysteme*. PhD thesis, Reihnisch-Westfälische Technische Hochschule (RWTH), Aachen, 1993.

[65] S. Spaccapietra. Conceptual Data Modelling. In *Summer School on Conceptual Modelling, Databases & CASE*, Lausanne, Switzerland, July 1992.

[66] 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.

[67] N. Streitz, J. Hannemann, and M. Thüring. From Ideas and Arguments to Hyperdocuments: Travelling through Activity Spaces. In *Proc. of the Int. Hypertext Conference*, pages 343–364. ACM Press, 1989.

[68] E. Wilson. Links and Structures in Hypertext Databases for Law. In *Proc. European Conference on Hypertext (ECHT-90)*, pages 195–211, Paris, France, 1990.

[69] David P. Wood, M. G. Christel, and S. M. Stevens. A Multimedia Approach to Requirements Capture and Modelling. In *Proc. of the 1st International Conference on Requirements Engineering (ICRE'94)*, pages 53–56, Colorado Springs, Colorado, April 1994. IEEE Computer Society Press.