Towards the Boundary of Concurrency

R. Ortiz
Daimler Benz AG, Research Center Ulm, 89013 Ulm, Germany

P. Dadam
University of Ulm, Faculty of Computer Science, 89069 Ulm, Germany

Abstract

The support of highly concurrent engineering work proposed in this paper is achieved by means of an Event - Condition - Action (ECA) rule - based synchronization component. Process and product information are managed in a database according to the Concurrency Model, a semantical process and product data model. The definition of ECA rules allows the active database to recognize conflicts at run-time and to execute pre-defined conflict solving actions. Thus a high degree of concurrency can be achieved without demanding fixed workflows. The Concurrency Model approach supports the explicit control of workflows and data flows, concurrent access to data and control of redundancy. Scheduling decisions and cost evaluation for project planning and control are supported by the simulation capabilities of our approach.

1 Introduction

In our research project, the wiring design domain was defined as a test case for the support of engineering activities with a high degree of concurrency. This means that we aim to allow of a large number of tasks to run simultaneously with high overlap of underlying product data and time. High concurrency has gained little attention up to now by work in the Concurrent Engineering research field. Up to now, there is no distinction between the simultaneous work of two designers with little overlap of data and time and the concurrent design of an engineering device by several engineers, where the involved persons start their work nearly at the same time manipulating almost the same data. In the first case there is a low number of conflictive data manipulation. The second case is an example of highly concurrent work, characterized by a high complexity of the synchronization mechanism, needed.

Existing approaches to support concurrency, like workflow management in Computer Supported Cooperative Work and the definition of design spaces do not adequately reflect the nature of the engineering domain. They are based on a priori distinctions among synchronous and asynchronous tasks. But complex engineering processes are multi-causal. This means that they enforce a continuous redefinition what the problem and what the goals are. Therefore, it is impossible to fix the problem solving steps and their schedule in advance. In addition, it is difficult and often also impossible to decompose design problems and their solutions to completely independent sub-problems which can be pursued separately [1]. Whether some special case of engineering work must be performed synchronously or not, can only be determined at run-time according to the concrete situation described by means of the "current" product and process data. Thus deterministic approaches are not powerful enough to support high concurrency [2]. Instead, an autonomous synchronization component is required which recognizes conflicts and starts resolution strategies at run-time, if necessary.

The next section describes the electrical design domain in aircraft engineering. In section 3, the requirements of simultaneity for this particular area as well as the boundaries for the concurrency of engineering work are evaluated. Section 4 presents the characteristics of the proposed semantical process and product data model, the so-called Concurrency Model, which is the basis of our approach. Conflict recognition and resolution using Event-Condition-Action rules as well as the information requirements for rule checks are addressed in section 5. Section 6 discusses the functional components of the Concurrency Model approach, their role in an integrated framework architecture, and the current implementation status. Finally, in section 7 the obtained conclusions and open research issues are addressed.

2 The Aircraft Engineering Domain

Today, product development activities for aircraft engineering are already performed concurrently to some extent. This is done based on the principle of geometrical and functional design spaces (see figure 1). Working on design spaces means the geometrical division of the product in different parts (design spaces) with pre-defined interfaces. Later, during the real simultaneous work phase each engineer (or engineering team) works on the product specification within its individual design space. This is followed by the post-defining phase within which the integration of the individual design spaces into the final global design is performed [3,4,5]. The reduction in development time obtained by this principle is carried out at the expense of limiting the autonomy of engineer decisions to just one design space. Optimization is only possible within the pre-defined design spaces. The modification of interfaces between different design spaces causes parts of the design to become obsolete or incorrect during an intermediated period of time [6]. This problem is increased due to the fact that the work teams involved are usually distributed geographically and are using heterogeneous software systems such that the data exchange among systems has often to be performed by paper or by primitive data exchange interfaces with loss of data and semantics.

In addition, processes in the environment under consideration are very complex. They can be described as hierarchically structured activity trees in which dependencies among activities can span several subtrees [7]. The process knowledge is usually not centralized but distributed among several persons. The effects of different scheduling decisions are therefore difficult to estimate in advance.

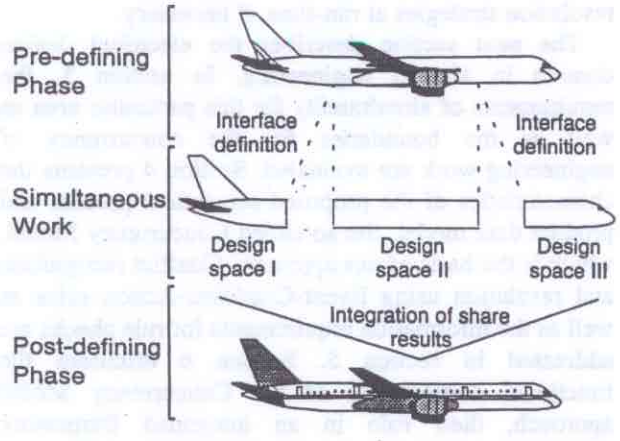

Figure 1 The design spaces principle

Wiring design differs from other areas in the domain of aircraft engineering. In comparison to the other disciplines, wiring design runs mainly sequentially and is therefore time-intensive and expensive. Because nearly each aircraft has a different device configuration, electrical design and manufacturing have the character of individual production. Important aspects are that the electrical equipment (cabling) is affecting several design spaces and has also a lower priority than other design disciplines such as hydraulic. For these reasons fixing of electrical interfaces among design spaces can not be done during the pre-definition phase. They are defined iteratively during progress of work [8]. This trial-and-error methodology causes a high number of iterations and aborts. Planning and execution of electrical engineering must therefore embody a high degree of flexibility. Exchanged electrical data are often saved redundantly such that global control of data actuality is very difficult to achieve. Drawings and lists are used as common "language" in this domain, which are usually not exposed in a preliminary state. This means that engineers often have to wait for data from other sources although they could proceed to a certain degree based on preliminary information.

Due to activity duration as well as the type of data (complex objects) the classical locking mechanisms of (central or distributed) database systems cannot be applied satisfactorily in this area. Advanced engineering database technology, communication mechanisms, and system support of cooperative engineering work are therefore required.

3 The Boundary of Concurrency

We want to improve concurrency support in the engineering domain described above. But first we must handle the question: Is there a boundary for the concurrency of engineering work? That is, how much concurrency can be introduced into an engineering process? To answer this "pointing-the-way" question we want to go into an imaginary engineering world:

In our imaginary world, there is a higher number of engineers involved in the development of an aircraft than today. We enter into a design office and look at the display of some engineer who is working on the specification of an electrical device, e.g. the wiring of a video equipment. Looking at the display of some other engineer we notice that he is also working on the same design. Even many more engineers are working on it. Modifications carried out by some designer are tracked back to the affected design activities in real time,

although the dependencies among the tasks are not obvious (e.g. there can be a dependency between the type of an electrical device and the allowed wire type). Whenever a conflict occurs the persons involved are automatically asked for interaction to resolve the conflict using an integrated communication facility.

Two engineers are simultaneously moving the same three-dimensional point. Both point movements are accepted partially and integrated in the resultant point position (see figure 2). But such a 3D point like in figure 2, is not limiting the concurrency. Two engineers can change the same axis value of a point at the same moment and both changes can be valid (e.g. if the real number given by a user is the top limit and the other user inputs a smaller value). But there are neither closed design spaces nor a fixed workflow definition. The engineers do not have to wait for locked data, and just one version of the video equipment results finally. Engineers finish a task within minutes for which a team in the real world of today would need many hours. The end solution is not the addition of propositions done by each engineer but it is an optimal combination of them. In this imaginary world, the boundary of concurrency is the atomic data type of the semantical data model. Such atomic data types are almost the simple data types integer, real, string, etc.

Two important conclusions can be drawn from this mental excursion: Firstly, a data manipulation that seems to be conflict free at first glance (like the change of an electrical device and of the used wire type) could generate an unexpected conflict. Secondly, the simultaneous data changes supposed to generate a conflict, e.g. when two engineers simultaneously move the same point, can be completely free of conflicts.

We asked the question whether the engineers in practice would accept to work with an information system supporting such a broad range of concurrency. There are profound occupational and sub-occupational differences in the way in which work groups share space and structure activities. Such occupational groups whose developments of physical objects are shared in posted drawings or sketches (e.g. artists, architects, mechanical and electrical engineers) tend to prefer open work spaces. Other occupational groups like software engineers, academics and writers prefer generally private workspaces with fewer intrusions and interruptions [9]. We, therefore, are quite confident to achieve a good acceptance of our highly concurrent engineering environment, once it is being tested in the wiring design practice.

Another important issue is whether meaningful results of simultaneous engineering work are possible without the definition of design spaces or fixed workflows. Experiments in three dimensional computer aided design showed that concurrent editing is not chaotic due to the intervention of social protocols [10]. Therefore the definition of goals for each engineering activity as well as the run-time control of work should be sufficient to guarantee successful engineering work.

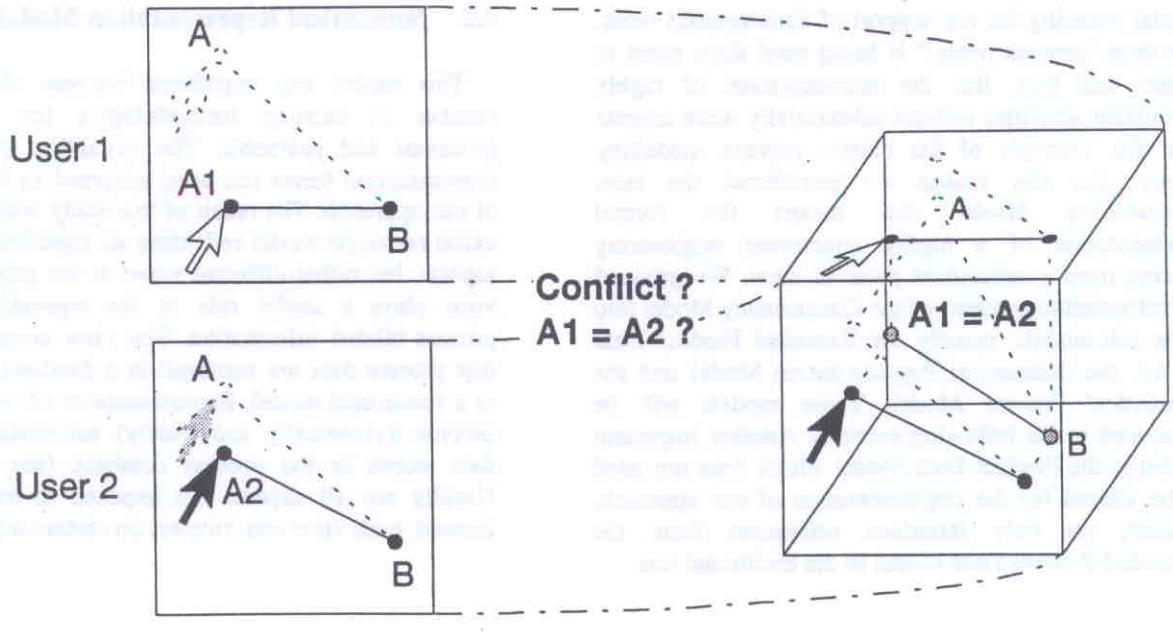

Figure 2 Working at the boundary of concurrency

Due to the fact that the goals of concurrent engineering (shorter time to market, reduced development costs and increased product quality [11]) are by far reached in the imaginary world outlined above, efforts on the Concurrent Engineering area should be pursued more ambitiously.

Summarizing the discussion above it can be said that highly concurrent engineering processes are characterized as follows:

- The (final) limit of concurrency is the atomic data type of the semantical product data model.
- There is not a fixed, pre-defined course of activities and steps.
- Changes are propagated immediately to concerned activities.
- Conflicts are recognized and resolved when they occur.
- Users are isolated from synchronization work.

The next sections show the proposed mechanisms for concurrency control of our approach. These concepts are based on the assumption that there exists a conflict recognition and conflict resolution expertise which is applicable to the semantical process data and product data model. This knowledge can be applied at run-time to recognize which activities have to run synchronously and to start conflict resolving strategies if necessary.

4 Concurrency Model Data Contents

The formal representation of the process data has a special meaning for the support of simultaneous work. The term "process model" is being used since years to reflect this fact. But the representation of highly concurrent activities reflects substantially more aspects than the concepts of the classic process modelling theory. For this reason we introduced the term *Concurrency Model* that means the formal representation of a highly concurrent engineering process from a semantical point of view. We grouped the information contents of the Concurrency Model into three sub-models, namely the Extended Product Data Model, the Semantical Representation Model and the Semantical Process Model. These models will be illustrated in the following sections. Another important model is the Product Data Model which does not need to be altered for the implementation of our approach. Instead, we only introduce references from the Extended Product Data Model to the traditional one.

4.1 Extended Product Data Model

There is much product information required for the support of highly concurrent engineering work which is not sufficiently reflected by the product data models used today [12]. To ensure that product data stay "clean" of such information and to avoid that the product data model has to be altered, we defined an extension of the product data model, the so-called Extended Product Data Model. This extension also contains references to data stored according to the product data model.

Through the support of further types of information the semantics of the application domain can be more adequately reflected. Thus the concurrency control capabilities of the engineering environment can be extended. Some examples of the additionally introduced information are: parametrical dependencies of product data (e.g. the length of a cylinder A is 2 times its diameter), quality stages of product data like preliminary, pre-released and released (e.g. the current value of the diameter of A is pre-released, its length is preliminary), range of values (e.g. the diameter of B must have a value between 25 and 30 mm), data versioning items (e.g. the current attribute values of C form its version 3), and the history of value changes in product data (diameter of D had the values 26, 29 and 28 mm) [13]. The Extended Product Data Model also reflects the hierarchical product structure according to the application terminology, e.g. a video equipment consists of video player, screens, wires, etc.

4.2 Semantical Representation Model

This model was introduced because of the high number of existing methodologies for describing processes and protocols. The suitability of selected representation forms has been analyzed in the context of our approach. The result of this study was that there exists no single model reflecting all significant process aspects, but rather different views at the process. Each view plays a useful role in the representations of process related information. The view concept means that process data are managed in a database according to a semantical model. Representations of this data are derived dynamically and (partly) automatically from data stored in the process database (see figure 3). Usually not all aspects are exposed in every view. Instead, each view concentrates on certain aspects.

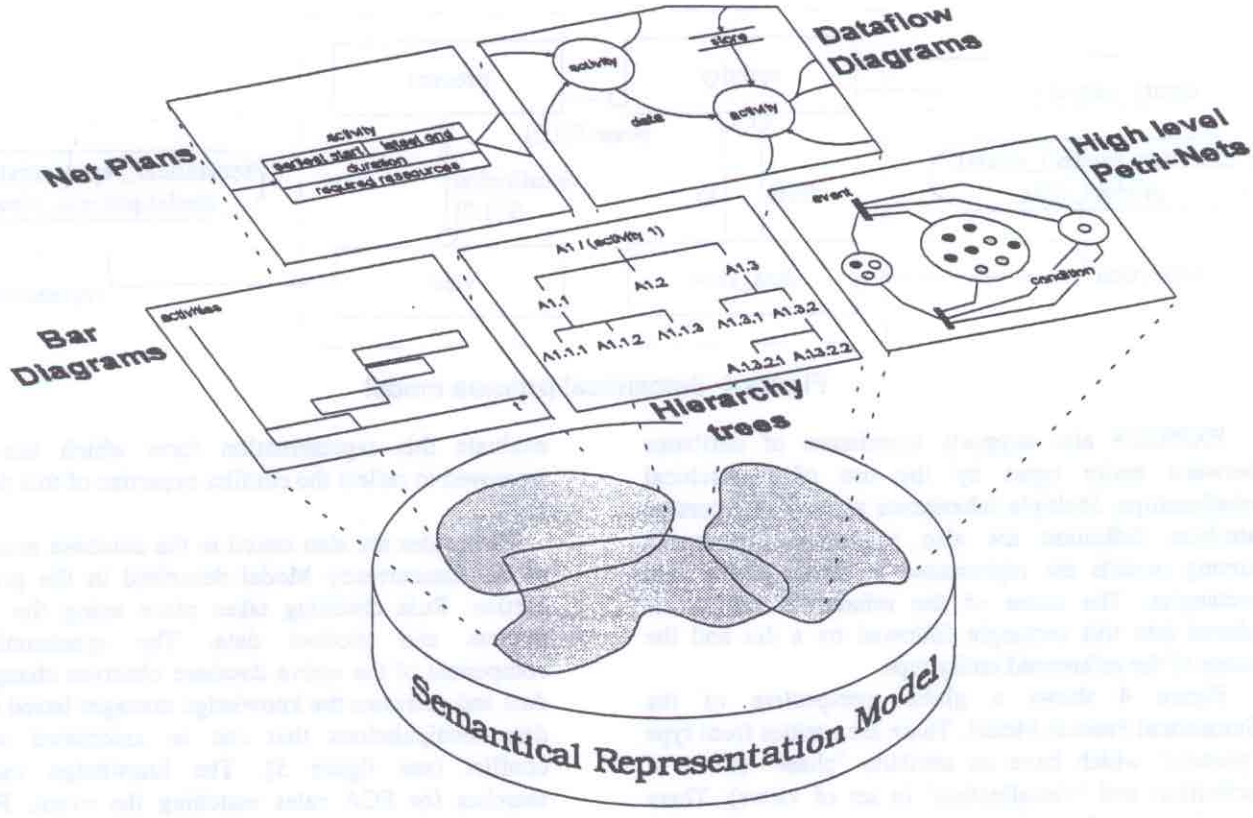

Figure 3 Process views as abstractions of process data

Different representation techniques have to be supported but the process viewer should also be extendible. Adequate representation forms are domain specific. For our studied case we decided to support process data representations in form of bar diagrams, net plans, data flow diagrams, hierarchy trees, and high level petri-nets.

Users never work directly with data of the semantical representation model but use logical views instead. According to the incremental modelling approach the person responsible for the project definition starts using a selected modelling technique. The input data are transferred later into the global schema of the process database. These data can be afterwards transformed into the next view, upon which further project aspects can be defined and entered.

4.3 Semantical Process Model

The Semantical Process Model is the core of the concurrency control in the presented approach. It describes the dependencies among the different steps of the overall process, e.g. data flow dependencies, time constraints, hierarchical relationships, etc. Activity classes have been determined as abstractions of comparable work phases. Activity states (in-work, resumed, restored, committed, aborted, etc.) as well as stage transitions have been also integrated into this model. A distinction between individual work (tasks) and collaborative work (interactions) was carried out. Common forms of interaction have also been defined. References among engineering activities and product data were introduced by the definition of data inputs and outputs for each activity. Further information contents are the factors time and cost for each activity type, which was introduced to support simple financial analyses.

An extract of the semantical process model is shown in figure 4. The employed notation is called EXPRESS-G, which was developed as part of the data exchange format STEP [14]. EXPRESS-G serves to build semantical data models through the definition of entity types, represented as rectangles in figure 4. Entity types are described by named attributes, which are drawn as thin lines among rectangles. Attributes must belong to a specific type shown by the small circles at the end of thin lines. The min-max numbers following the attribute name express the cardinality constrains of the attributes. There are also aggregated types in EXPRESS-G, which allow to define further constraints for attributes, e.g. in figure 4 a "process" has the attribute "phase" which is a "set" of activities.

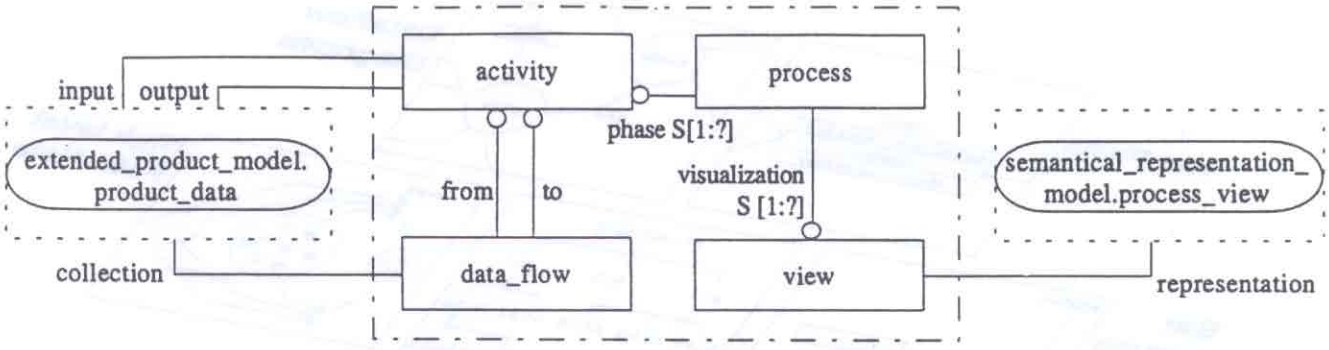

Figure 4 Semantical process model

EXPRESS also supports inheritance of attributes between entity types by the use of hierarchical relationships. Multiple inheritance as well as recursive attribute definition are also supported. References among models are represented as ovals placed into rectangles. The name of the referenced schema is placed into this rectangle followed by a dot and the name of the referenced entity type.

Figure 4 shows a global perspective of the Semantical Process Model. There are entities from type "process" which have an attribute "phase" (a set of activities) and "visualization" (a set of views). There are also data flows from one activity to another one containing collections of data which are references to the extended product data model. Activities are further described by inputs and outputs which are also references to the extended model. The attribute representation of entity view allows to define references to the semantical representation model.

5 Conflict Recognition and Resolution

Conflict resolution plays a central role in achieving support of simultaneous work. As described above, existing mechanisms for concurrency support are either based on a resolving possible conflicts in advance or on fixing of the point in time for conflict resolution in advance. Our approach is based on Klein's and Lu's [15] assumption that there is a rich collection of domain-independent conflict recognition and resolution expertise which can be identified in advance and applied at run-time. The separation of the conflict knowledge from the execution code allows an easier definition and update of the expertise as well as the reconstruction of system decisions. Due to the procedural nature of the process expertise we decided to experiment with the representation of knowledge in form of Event-Condition-Action (ECA) rules. Once the framework implementation is completed we will evaluate this representation form which has been improved to reflect the conflict expertise of this domain [16].

ECA rules are also stored in the database according to the Concurrency Model described in the previous section. Rule checking takes place using the stored process and product data. The synchronization component of the active database observes changes on data and activates the knowledge manager based on the data manipulations that can be associated with a conflict (see figure 5). The knowledge manager searches for ECA rules matching the event. For all rules matching in the event part, the condition part (which is a logical expression) is verified. If the result of the condition check is "True" the action part of the rule is communicated to the synchronization component which starts the conflict resolution executing the predefined action.

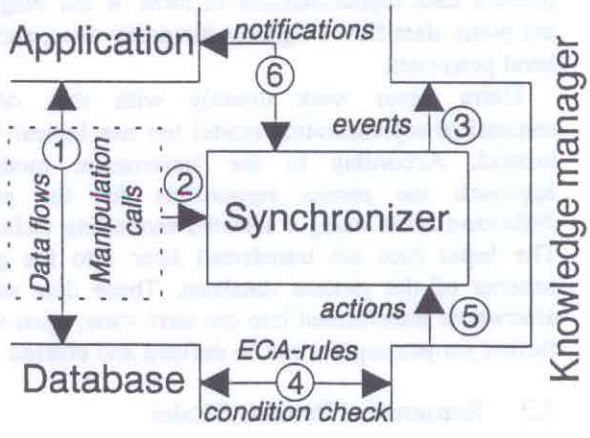

Figure 5 Rule check procedure

Figure 6 shows three examples of representative ECA rules. The first one is applied if the synchronization component detects that some new activity is being started. The activity identifier is passed on to the knowledge manager. This component

finds the rule referring to the current event (activity begins) and starts to check the condition body. In this case the data types which were defined as input of the concerned activity are identified and a request for instances of these data types is sent to the database. In case that there are not "enough" input data available, another query is generated which identifies the activities with corresponding output data types. If some predecessor activity is still running, the synchronization component receives a request to start the action part and the required information to execute the action (in this case the identifiers of involved activities and a description of the data which causes the conflict). The synchronizer starts the required conflict solving action. This means, it informs the predecessor activity about the conflict. The synchronizer then waits for an answer which can contain either data in some quality state or even an assumption of additionally required time. The answer is subsequently sent to the activity to be started.

The second example in figure 6 is a time trigged rule. The synchronization component observes that the pre-defined latest end time for an activity is reached. This event is further communicated to the knowledge manager together with the identifier of the activity. Later, the knowledge manager generates a database request for the current status of that activity. If the control of the activity shows that it is still running, the action part is communicated to the synchronizer. This component sends a commit request notification for the user on that activity. He can end the work task or he can answer with an assumption of the additional required time. In case this time is "too" long the synchronization component starts a new resource allocation, for example through the creation of a sub-task to be carried out separately.

The last example we want to handle here is applied when the active database obtains an update request. This event is noticed by the synchronizer, which communicates it to the knowledge manager for rule checking. A database query determines if such a request coming from another activity has already been accepted. In this case the identifier for the activity that is performing an update at present as well as a reference to the data instances is passed on to the synchronizer. This component starts the cooperation groupware and provides the data which causes the conflict. The users involved can interact using this software and can negotiate about a commonly accepted value for the data which are causing the conflict.

Event	Condition	Action
succesor activity begins	input instances missed	predecessor notification
latest end time reached	activity on run	further ressource allocation
update request	data already in write	negotiation start

Figure 6 Selected examples of ECA rules

Two types of rules can be identified in the conflict expertise: Rules that are case independent or general rules, and the case specific or special rules [17]. General rules are always known at the knowledge manager side. The special rules for the related case have to be entered before or during process execution to ensure that the specific process can be performed. To avoid that an important special rule remains undefined and that there are rules mutually eliminating each other, a process simulation must be started before the real process begins. Of course, a simulation can not analyze all possible cases, and therefore process deadlocks are nevertheless possible. For this reason, strategies for product and process data recovery have to be also improved. Process simulation can reveal that there are not sufficient instances of the semantical process model to ensure successful process control. In this case the synchronization component generates a list of information demands to be entered into the database before the process starts. If some information is unknown, the synchronization component releases a process start and later, by progress of work the information demand is addressed again.

As an "emergency exit", some tasks of the synchronization component can be suspended temporally by the project administrator. In this case the process control uses the conventional process management components, this means without the automatic execution of system actions. This status remains until a new consistent execution plan is formulated. The knowledge management system must have an explanation component which shows on demand which rules have led to a certain decision. We expect the number of general rules to be not more than 100. The number of project specific rules are expected to be even smaller.

6 Functional Architecture

In this section we want to outline the functional requirements for the implementation of the Concurrency Model approach. This catalogue can be used as a basis for further framework implementations. At present, an implementation is being done at the University of Ulm, partially using standard application software and existing software development tools. Figure 7 shows the required functional blocks, which are handled in detail in this section. The description of the framework components bases on the functions to be provided by the modules.

6.1 Data Repositories

The product data will be stored in a database according to the product data model. In an initial phase we are using a product data model that contains only electrical design related information. At present, the product database will store data only from one application program, namely SchematiX11. In the future we will implement a standardized product data model according to the ISO standard STEP (Standard for the Exchange of Product Model Data) [14]. Thus, also data from other applications can be stored in the product data repository. The exchange of product data between the Concurrent Engineering framework and other external systems will also be improved by the neutral data model. Concurrency related information will be also stored in this repository according to the Concurrency Model described in section 4.

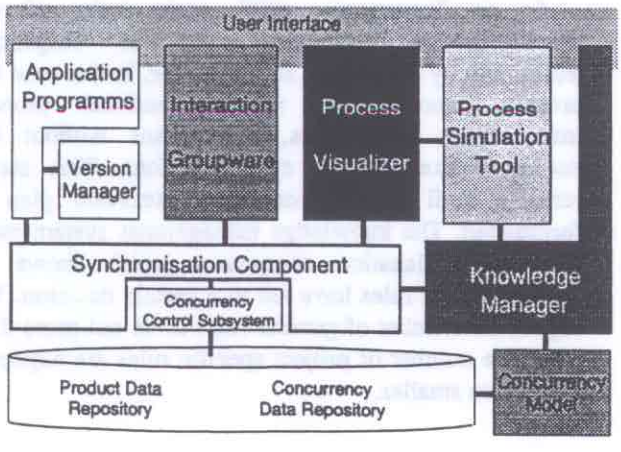

Figure 7 Functional system blocks

6.2 Concurrency Model

This component has been already discussed in detail in section 4 and will not be handled again at this point.

6.3 Database System and Concurrency Control Subsystem

For data management a standard database system has been integrated into the framework architecture. The concurrency control subsystem supports advanced concurrency control mechanisms like nested transactions and save-points which are used by the synchronization component. Versioning capabilities for product and process data have also to be provided by the database system. In our prototype implementation we are using ONTOS, a development of ONTOS Inc. USA, which satisfies most of the database related requirements. Direct access by the user to data stored in the database is not allowed. Instead, data manipulations are only possible through functions of a component with a user interface (the application programs, the groupware, the process visualizer, the simulation tool, and the knowledge manager).

6.4 Synchronization Component

Another functional software category in our Concurrent Engineering approach is the synchronizer of engineering activities. During the start of an activity this component starts the required application programs and provides the input data. The main function of this component is to observe modifications of product and process data and to activate the knowledge manager if events occur which can generate a conflict. Actions for conflict solving suggested by the knowledge manager are started by this component. The synchronizer also communicates with users when a commitment of an activity, output data, or assumptions concerning additionally required time must be requested.

6.5 Knowledge Manager

The function of this module is the support of input, update, and verification of the conflict expertise. The knowledge manager serves also as an inference engine which decides which rules must be applied by a given trigger. The required information for rule checking is contained either in the Concurrency Model, in the product data repository, or in the concurrency data repository. Therefore, the knowledge manager must also communicate with the database asking for product and process data, but also for meta data defined in

terms of the Concurrency Model. There are different intelligent system building tools that are being analysed for the implementation of this component. The implementation phase of the knowledge manager is planned for the end of 1995.

6.6 Application Programs

The definition and update of product data take place in the application programs such as computer aided design software. For our prototype we are integrating the electrical CAD system SchematiX11, a development of the Australian software house Softsmiths Pty. Ltd.. This system stores data continuously in ASCII files, one for each electrical schema. The product data contents of this ASCII file can then be stored in the database. Data which is required on the application program side is checked out from the database and written into an ASCII file so that it can be read by SchematiX11.

Because the achievable degree of concurrency depends on the frequency of data transfer from the application program to the database, the CAD system should update this file frequently. Another important issue is how often changes should be propagated or should become visible in other concerned applications programs. There are three alternatives for change propagation from the application to the database and from the database to other involved activities [18]: (1) broadcast all changes, (2) refresh periodically, and (3) refresh on demand. Option 1 was excluded because we are integrating standard application programs whose source program code is not available to us. We, therefore, decided to apply a combination of options 2 and 3. The synchronisation component waits during a pre-defined length of time that the user saves data into the ASCII file. When this time is exceeded a save request is sent. If there are changes that should be propagated to other application programs a "save-and-close" request is sent to the user. Saved data are then combined with data from the other application by the merge mechanism discussed below. Afterwards, the users can re-load the file.

6.7 Version Management Tool

This framework component handles the different versions of ASCII files updated by the concurrent activities. Its main function is to support the selection of instances from two or more object versions and to store them into a new one. The necessity for such a merge component has often been addressed in the literature [2,4,13,15,19], but there are until yet not significant advances on this research area. The mapping of logical representation to the merge of physical files has to be developed individually. The semantics of the application expresses the areas that are subject of changes. Non-conflicting areas ca be changed simultaneously. The version merge tool must be able to identify the modified parts and to update the master copy accordingly.

We defined a simple merge procedure to experiment with. This mechanism to combine data contained in the files A and B can be sketched as follow (see figure 8): For each object O_a in file A, find an object O_f in file B with the same object ID like O_a or find an object O_f in B with the same data type as O_a whose attributes are *comparable* to those of O_a. Find the difference (Δ) between O_a and O_f based on the attribute values that are not equal in both objects. For each attribute of Δ select a value using the semantics of data, e.g. check whether the attribute was defined as a field (see section 4) or whether some activity has a higher priority than the other ones. In case no selection can be carried out, ask „somebody" to do it. The selected attribute values of Δ are then added to the object O_m which is now defined as the merged object. Then O_m is added to the merged file. All Objects O_a and O_f that could not be matched are also included in the merged file. The core of our merge mechanism is the match of the objects among the files, this means to find out which objects in file A should be compared with objects in file B. But also the semantical selection of attribute values plays a central role. This merge procedure is being implemented at the present. Some simple tests have shown that most of the correct data contents of the files to be merged can be selected successfully.

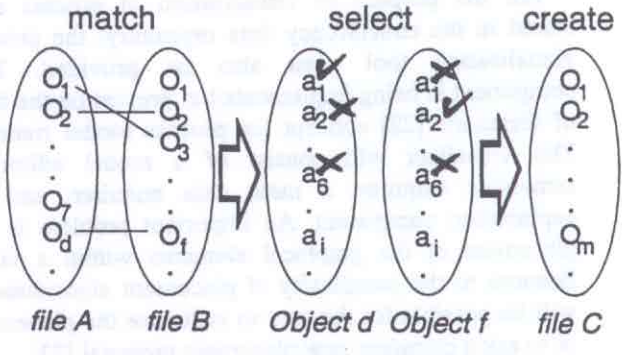

Figure 8 The merge procedure

6.8 Interaction Groupware

Traditionally, a discussion about some product or process characteristics is performed by two or more engineers working together in the same room having a drawing in front of their eyes. Each engineer can observe the progress of work and can also integrate it into his work. But the complexity of the engineering work has substantially increased. Cooperative work is not anymore a simple meeting of two or more experts. The interaction component should provide the same data view to all participants, regardless whether they work in the same room or not. Each party should have a mouse pointer in the groupware window. There must be also possible to interchange textual data or voice during the interaction.

In our implementation objects are not locked when they are being changed by an engineer. Instead, we use coloured representations to differentiate among: (1) data being manipulated in activity A but not involved in the conflict, (2) comparable data but from activity B, (3) data involved on the conflict, (4) data being changed by activity A at the moment, (5) data being changed by activity B, and finally, (6) data that are being changed simultaneously at the moment. This implementation will be based on a toolkit called Groupkit [20], a development of the University of Calgary. There is a high number of possible interaction forms, e.g. delegation, discussion, rejection, proposal, brainstorming, voting, etc. [21]. To reduce the complexity by the implementation of this component classes of interactions with comparable or even same behaviour will be built. For each class a set of cooperative steps can be defined and implemented in form of an interaction protocol.

6.9 Process Visualizer

For the purpose of visualization of process data stored in the concurrency data repository, the process vizualisation tool must also be provided. This component is being implemented at present on the base of Gassner's [22] concept for process model transfer. The visualizer will consist of a model editor, a semantics enquirer, a meta data enquirer, and an explanation component. An important problem is the placement of the graphical elements within a view. Because to the complexity of placement algorithms, it will be possible for the user to influence the placement or to ask a complete new placement proposal [23].

6.10 Process Simulation Tool

By introducing financial aspects into the Concurrency Model the implications of cost and time by a given project plan can be examined by a process simulation tool [24]. We also intend to implement such a simulator into the framework. This component provides the ability to examine different variations of the project plan as well as the implications of probable engineering iterations. Changes on any activity flow parameter can be estimated using the simulation tool.

7 Conclusions

The definition of fixed workflows does not reflect the "chaotic" approach that characterizes the real engineering world. Trying to determine a priori whether two or more parallel working activities must be performed synchronously or asynchronously is only possible by using methodologies that limit the engineer in his work. Therefore, only sub-optimal solutions can be achieved in this way. The demand for coordinated, synchronous work must be recognized at run-time when a conflict appears. The simultaneous manipulation of a small data type (like an integer) does not need necessarily to generate a conflict. Analogously conflicts caused by the parallel manipulation of two data types that can be independent from one other and should therefore be handled at the semantical data model level.

At the beginning, we described a visionary world that should be pursued to achieve significant advances in the concurrent engineering research area. This world is characterized by the atomic data type of the semantical product model as boundary of concurrency. Further characteristics are the non-deterministic process definition, run-time propagation of changes, run-time conflict recognition and resolution, and isolation of users from synchronization aspects. This imaginary world can partly become real using an advanced semantical process and product data model. The existence of rules applicable to this semantical model has been established. Thus the framework can activate rule checks and determine applicable conflict solving actions.

There are several open research issues in the area of high concurrency: e.g. how to avoid negative cascading effects; how to evaluate, model or express chances and (financial) risks; how to optimize the representation of the process data; which error and exception handling mechanisms can be used; how to achieve object recognition during version merge, etc.

8 Acknowledgements

Several useful suggestions and comments were provided by Dr. Johann Krammer and Dr. Josef Vilsmeier (Daimler-Benz Aerospace AG) as well as by Thomas Beuter and the team of students (University of Ulm). We would also like to thank Dr. Dieter Haban (head of the department CIM, Daimler-Benz Research Center Ulm) for his continuous support.

9 References

[1] Smithers, T.; Troxell, W.: Design is intelligent behaviour, but what is the formalism? In: Artificial Intelligence EDAM, (1990) 4(2), 89-98

[2] Prasad, B.; et al.: Information management for concurrent engineering - research issues. Concurrent Engineering - Research and Applications, 1(1993) 3-20

[3] Medland, A.J.; Janoes, A.: An approach to design based upon functional logic. In: Proc. 7th. Design Engineering Conference. Birmingham, UK, July 1984

[4] Anderl, R.; Malle, B. Schmidt, M.: Concurrent engineering based on a product data model. In: Proc. 3rd Conference on CALS and Information Management in Europe. Paris, France, 1992

[5] Grabowski, H.; Schmith, M.: Distributed Design - Working in Design Spaces. In: Proc. Of the CAD'92 GI Workshop. Berlin: Springer, 1992. In German

[6] Schmith, R.F.: Concurrent design - development time reduction through simultaneous design. In: Konstruktion, 45(1993)145-151. Berlin: Springer, 1993

[7] Krammer, J.: PEFF - Process chain optimization in the aerospace industry. Internal report Daimler-Benz Aerospace Corporation / LM, number 2. Sept. 1993. In German

[8] Schneider, P.; Ortiz, R.: Process chain integration using a common data model. In: Proc. CIM European Annual Conference 1994. Copenhagen, Denmark, Oct. 1994

[9] Reder, S.; Schwab, R.G.: The temporal structure of cooperative activity. In: Proc. ACM CSCW'90. Portland, OR, USA, September 1990

[10] Shu, L.; Flowers, W.: Teledesign: Groupware user experiments in three-dimensional computer aided design. In: Collaborative Computing, 1 (1994) 1-14

[11] Albano, L.D.; Nam, P.S.: Axiomatic design and concurrent engineering. In: Computer Aided Design, Volume 26, Number 7, July 1994, 499-504

[12] Biennier, F.; et al.: A hypermedia based model for concurrent engineering. In: Concurrent Engineering - Research and Applications, 1 (1995) 3-20

[13] Kirsche, T.; et al.: Functionality and architecture of a cooperative database system - A vision. In: Proc. 3rd int. Conf. on Information and Knowledge Management, CIKM'94. Gaithersburg, MD, USA. December 1994

[14] STEP ISO 10303: Product data representation and exchange. ISO TC184/SC4 N181. JAN93, 1993

[15] Klein, M.; Lu S.C.Y.: Conflict resolution in cooperative design. In: Artificial Intelligence in Engineering 1989, Vol. 4, No. 4, 168-180

[16] Krause, F.-L.; et al.: Implementation of technical rules in a feature based modeller. In: Akman, V.; et al.: Intelligent CAD Systems II - Implementation issues. Berlin, New York: Springer, 1989

[17] Delahaye, J.P.: Formal methods in artificial intelligence. London: North Oxford Academic, 1988

[18] Bean-Shaul I.Z.; et al.: An architecture for multi-user software development. In: Computing Systems, Vol. 6, No. 2. Berlin, New York: Springer, 1993

[19] Adams, E.W.: Object management in a CASE environment. In: Proc. Of 11th Int. Conference on Software Engineering. Computer Society Press, 1989

[20] Greenberg, S.; Marwood, D.: Real-time groupware as a distributed System - Concurrency control and its effect in the interface. In: Proc. ACM CSCW'94. Chapel Hill, NC, USA, October 1994

[21] Dürr, M.: Coordination mechanisms for team work - Model building and database support. Ph.D. - Doctoral thesis at the University of Karlsruhe, Germany. February, 1994. In German

[22] Gassner, K.: Formalization and transfer of the semantics of conceptual data models and process models. Ph.D. - Doctoral thesis at the University of Ulm, Germany, June, 1994. In German

[23] Paulisch, F.N.: The design of an extendible graph editor. Lecture Notes in Computing Science No.704. Berlin, New York: Springer, 1993

[24] Duffey, M.; et al.: Managing the product realization process - A model to aggregate cost and time-to-market evaluation. In: Concurrent Engineering - Research and Applications, 1 (1993) 51-59

Concurrent Engineering

A Global Perspective

Conference Proceedings
1995

Sponsored by:

Concurrent Technologies Corporation