in: Tagungsband CAD’98 "Tele-CAD - Produktentwicklung in Netzen" , März 1998, Informatik Xpress 9, pp. 188-201

Applying ECA Rules in DB-based Design Environments

Joachim Reinert and Norbert Ritter

Department of Computer Science, University of Kaiserslautern
P.O.Box 3049, 67653 Kaiserslautern, Germany
e-mail: {jreinert|ritter}@informatik.uni-kl.de

Abstract. The goal of this paper is to bring together the worlds of design and active databases. Design databases have to provide a powerful data model for complex structured design data as well as a powerful activity model asserting basic data consistency and supporting workflow (in the design area we should better call it designflow) and cooperation among designers. There are numerous potential application areas for ECA rules within design demanding to incorporate active capabilities into design databases. We argue that ECA rules are well suited for error correction purposes providing a forward oriented handling of failure situations, even for repairing inconsistencies which were caused by cooperative data access. We will further argue that ECA rules do not suite for pre-planning, specifying, or implementing designflows. Beside these two areas, there are several application fields in design which can be supported by ECA rules, but the requirements of which could also be fulfilled by alternative mechanisms.

Keywords. Active Databases, ECA Rules, Design Databases, Activity Support, Designflow, Cooperation.

1 Introduction

Motivation

One of the long time perspectives of the DBMS research community is the support for engineering applications, especially support for an integrated view on the whole life cycle of a product. The very first phase of such a life cycle is the design of the product which is nowadays supported by a bunch of different tools, e.g., in software design there are upper and lower CASE tools, compilers, debuggers, etc. Not only the problem of supplying such a variety of tools with the data needed and incorporating the results back into the global database is a focus of interest, but also the support for the overall design process, i.e., basic building blocks for design methodologies, concurrent engineering and so on. Starting from the rigid ACID transaction paradigm (Atomicity, Consistency, Isolation, Durability) [19] numerous derivations have been developed to model the underlying processes often using ACID transactions as basic building blocks.

Active DBMS have been another research interest of the recent years promising modeling and run-time support for the dynamics of an application area opposed to the data model oriented approaches (e.g. relational data model). Even object-oriented approaches only provide the environment which allows to describe and execute dynamics through the methods of objects but the DBMS itself has no active role within this approach which is very different to what active DBMSs provide: The DBMS is thought to be an active mediator steering the design process w.r.t. the various tasks, i.e., controlling the cooperation of designers, starting design tools, matching the results of the design process with the design goals and much more.

The key concept developed in the active DBMS area is the so-called ***E**vent-**C**ondition-**A**ction* **rule**: After the occurrence of some event, the DBMS carries out an action, if a certain condition holds. Many authors propagating active DBMS facilities do also view this approach as a panacea of the problems in the design area. On the other hand, CAD framework developers consider the problems to be solved from a more abstract, application-oriented point of view. Hence, an evaluation of the potential of ECA rules w.r.t. design environments was hardly possible. This paper tries to bridge the gap between the high-level design problems and the more low level 'implementation' using ECA rules. We want to identify the domains within CAD environments, in which the application of such rules is likely to provide real benefit, in the sense of being adequate to solve the problems of that domain *and* providing a natural specification formalism.

Overview of the paper

Bringing together different 'worlds' is the difficult task of this paper. Starting with a brief characterization of the basic concepts of active databases and their origins (Sect. 2), we will outline a representative design model (Sect. 3). Thereafter, we discuss the areas within a design environment, for which the usage of rules is proposed (Sect. 4). For some of them the application of ECA rules will turn out to be very promising, others are possible application areas and for a third group ECA rules do not provide an adequate support at all. As the result of this research we suggest to concentrate on applying ECA rules in those areas identified highly promising (what else?) and to work on the languages for events, conditions and actions required in design to model tasks naturally and to support the entire process effectively.

2 The ECA-Rule Paradigm

Event-Condition-Action rules represent a paradigm rooted in multiple disciplines of DBMS research. To clarify the concepts of ECA rules it is helpful to look at these areas of research and analyze their influence on this amalgamated concept.

Real Life

The usage of 'rules' in every-day-situations is widespread. The example 'Only if you possess a driving license you are allowed to drive a car' shows that such rules can be considered to consist of a left-hand-side lhs (antecedent 'possessing driving license') and a right-hand-side rhs (consequence 'driving a car is allowed'). These rules can be used for both, forward-chaining/deduction (if I know someone owns a driving license, I can deduce that she/he is allowed to drive a car) or backward-chaining purposes (if I want to be allowed to drive a car, I have to apply for a driving license).

Deductive Databases

As mentioned before, rules are one basis for deduction in general. In the mathematical sense of deduction the lhs and the rhs of a rule are logic formulas (e.g. first order logic). Furthermore, the semantics of the rule itself has to be fixed (e.g. the lhs logically implies the rhs). In this sense rules without a lhs are facts, 'rules' without a rhs can be called queries. The central question is: Given a set of rules (including facts) and a query, is the query a logical consequence of the rules? The specialized field of deductive databases [13] copes with the problem of large numbers of facts, e.g. 'How can the facts be manipulated efficiently during the deduction process'.

Production Rules and Real-time Processing

Production rules [12, 11], stemming from productions forming formal languages, can be characterized as operating on a set of objects (called situation, working memory, ...). The lhs, which is a formula of some logical calculus, is checked for validity in the current situation. If it is valid, the rhs (called action), which manipulates the situation, may be applied. These changes result in non-monotonicity of the whole calculus (opposed to the deductive approach), which is a source of various problems: Because one rule may invalidate the results of another, the sequence of rule applications becomes relevant for the result. To achieve a deterministic behavior additional constructs are introduced (e.g. priorities) leading to complex nets of dependencies between rules. Hence, another attractive feature of rule sets is lost.

Although, compared to production rules, the focus of *real-time processing* [4] is slightly different (achieving a meaningful reaction within the *restricted timeframe* is the main goal), this area of rule usage inherits the properties (and problems) from the production-rules approach.

ECA Rules

The ECA-rule paradigm [8] is mainly based on that of production rules and on the event concepts of real-time processing, i.e., the lhs is the description of a specific situation consisting of an event and a logical condition. The event is specified as an event-algebra expression. An algebra is defined as a set of simple events in connection with constructors allowing to build complex events, e. g., the sequence of events (see e. g. [14]). The condition specification is given as a database query which (at least) may take the database state before the event and after the event into account. The rhs of a rule is a DML expression. Some authors suggest the use of a general purpose programming language with embedded usage of DML statements, especially for object-oriented approaches. The rule-ordering problem, i.e. the execution sequence of actions if rules fire 'at the same time', is mostly left to the user through the introduction of some sort of priorities.

A novelty of the ECA-rule paradigm in DBMS are the so-called ’coupling modes’ which allow to specify the execution contexts of the parts of an ECA rule [8]. Such coupling modes are defined between event and condition and between condition and action separately and each defines a time-coupling, i.e., whether the following part should be executed immediately or should be deferred. Transaction coupling defines whether or not the following part of the rule should be executed within the same transaction. Additionally, a new type of transaction is introduced namely the ’dependent transaction’. A transaction of this type has to be started by another transaction and is only allowed to commit if the starting transaction commits. For the purpose of this paper, we will use ECA rules in the widest sense possible to discuss whether or not such a paradigm is convenient within the development and usage of CAD. Hence, we will not propose a specific language for the rules but use intuitive expressions for events, conditions and actions. Furthermore, we will not elaborate on the problem of detecting complex events, testing non-trivial conditions and executing complicated actions. For general requirements for active DBMS see [15].

In the next section, we will switch to the second ‘world’, we want to elaborate on, the world of design applications. In Sect. 4 we will try to bring the two worlds together by discussing potential application areas of ECA rules in design.

3 Design Model(s)

This paragraph serves for briefly addressing the areas within design applications, for which the exploitation of rules as a supporting mechanism has been proposed. Since, it is not in the focus of this article to give a comprehensive view to design models, we selected one representative model, the CONCORD[1] design model [25, 26]. CONCORD allows to easily identify the mentioned areas, because it separates different aspects of design dynamics by distinguishing three hierarchically arranged operational levels which will be outlined in the remainder of this Sect. These operational levels are based on an version model allowing the management of explicit complex object versions and configurations [21].

Administration/Cooperation Level

This highest level of abstraction reflects the more creative and administrative part of design work. The focus is on the description and delegation of design tasks as well as on a controlled cooperation among the design tasks. The key concept at this level is the *design activity (DA)* being the operational unit representing a particular design task or sub-task. All relationships between DAs essential for cooperation are explicitly modeled, thus capturing task-splitting (*delegation*), exchange of (preliminary) design data (*usage*), and negotiation of design goals (*negotiation*).
Potential application areas for ECA rules at this level are pre-planning cooperation

1. The CONCORD acronym stands for: CONtrolling COopeRation in Design environments.

(see Sect. 4.4), handling inter-task dependencies (Sect. 4.5), implementing correctness criteria for cooperation (Sect. 4.6) and performing notifications (Sect. 4.7).

Designflow Level

Looking inside a DA reveals the designflow level. There, the organization of the particular actions to be performed in order to fulfill a certain (partial) design task is the subject of consideration (*designflow*). At this level, the *control/data flow* among several design actions performed within a DA is modeled. Usually, these actions are design tool applications which are applied to improve existing design states w.r.t. the corresponding design goal. The operational unit serving for the execution of a design tool is the *design operation (DOP*). In order to control the actions within the scope of a single DA, but without restricting the designers' creativity, flexible mechanisms for specifying and controlling designflows are provided [26].
Potential application areas for ECA rules at this level are design-task description (Sect. 4.1) and implementation (Sect. 4.2).

Transaction Level

From the viewpoint of the DBMS or data repository, a DOP is a long transaction having the properties of conventional transactions as far as DB-schema consistency, durability of updates, and isolation from concurrent transactions are concerned. Because of long duration, the atomicity property is relaxed; a transaction is internally structured by save/restore and suspend/resume facilities [18] to be able to rollback at the application level and to continue the design work after breaks. A DOP processes design object versions according to the load-operate-merge cycle.
Potential application areas for ECA rules at this level are maintaining design data consistency (Sect. 4.8) and error correction (Sect. 4.3).

4 Discussion: Application Areas of ECA Rules within Design

This chapter discusses the various areas where ECA rules might be applied within the CAD-framework. This discussion is required, because ECA rules are promoted for each of these areas, but seem inadequate for some of them.

4.1 Design-Process Description through ECA Rules

Some authors (e.g. [20]) propose the ECA-rule paradigm to model the whole design *process*, i.e., to use ECA rules to describe the dynamics of the designflow. To decide whether or not such an approach is useful, let us consider the major constituents of the design process. At the abstract level, the elements involved in the design process are designers, design objects and tools used to manipulate the design objects. To control the design process, some information on the status of the design has to be collected. In our design process model ([26], see Sect. 3), we suppose to store this information within the design objects, i.e., each object (version) is in a specific state w.r.t. the overall process. Other approaches propose to maintain this information separately. Those approaches allow to reference the state of a design process explicitly.

- Explicit modeling of the design process state

  The common approach for an explicit modeling is through the use of a design script. The elements of a script are tool invocations, which are connected through control structures (sequence, loop, parallel execution, etc.). These structures have to be expressed. For ECA rules, the occurrence of an event is a pre-condition for a rule to be effective. Hence, each rule (or each tool, if the tools are tightly integrated) has to post the correct events for other rules so that these rules get ‘fired’. In a software development environment such a rule may look like

  ```
  ON    end-of-compilation(sub-module)
  IF    no-syntax-errors(sub-module)
  DO    FOR EACH super-module OF submodule
        start INTEGRATIONTEST(sub-module, super-module)
  ```

  and

  ```
  ON    end-of-compilation(sub-module)
  IF    syntax-errors(sub-module)
  DO    start EDITOR(sub-module)
  ```

  These rules describe IF-THEN-ELSE branches and remain very simple because the only event is ‘end of design step’ and the only action is ‘initiate design step’. The condition is used to simulate the control sequences known from programming languages. However, the description of the process becomes very complex, because the number of rules will grow rapidly with the complexity of the design process modeled leading to a set of rules which will become rapidly unmanageable. To grasp the semantics of such large rule sets additional abstractions are required (e.g. stratification in the sense of [2]). Thus, the direct usage of ECA rules to model the design process explicitly is inadequate.

- Implicit modeling of the design process state

  Opposed to the first approach we propose implicit modeling, i.e., the progress of the process is not controlled explicitly but each object is in a specific state w.r.t. the overall process. This results in a data-driven approach. Hence ECA rules may be used to describe, which design step is next for a specific object. In order to decide which steps are applicable, the rules have to test which state the object has reached within the design process. Therefore, the concept of events is not so important, but the condition is the critical part of the description:

  ```
  ON    ANY CHANGE OF sub-module
  IF    compile(sub-module) = NO ERROR
  DO    FOR EACH supermodule OF sub-module
        start INTEGRATIONTEST(sub-module, super-module)
  ```

  and

  ```
  ON    ANY TIME
  IF    not-released(sub-module)
  DO    start EDITOR(sub-module)
  ```

In contrast to the former approach these rules represent *possibilities* of the next design step based on the actual state of an object. To face the problem of handling complex rule sets, rule processing obviously has to be guided by the designer. For example, in a situation where the compilation succeeded (without errors) but the sub-module under consideration is not (yet) released, both rules are applicable and, hence, the designer has to decide which step to do next, or there have to be meta-rules which control the rule execution (which, in turn, results in an explicit model of design states).

From this short description of the problems occurring by using ECA rules directly to model the design process, it should be obvious that such an approach suffers from the difficulties of any rule-based approach (gaining control over a large rule base) and would not be successful for a complex design process.

4.2 Implementation of the Design Process using ECA Rules

Another proposal is to use some model of the design process (e.g. the part of the CONCORD [26] model associated with the designflow level, see Sect. 3) and 'compile' this model into a set of ECA rules (e.g. [5]). This would allow to gain control over the set of rules through the compiler. The main advantage seems to be flexibility: Each concept of the design model is embodied by a (hopefully) small set of rules. The compiler combines these sets according to the overall structure of the specific design process. Enhancing the design model or altering the semantics of an element only requires some additional rules or the editing of a rule set. In our model, each (pre-planned or possible) design-state transformation would lead to a rule like

```
ON    ANY CHANGE OF sub-module
IF    compile(sub-module) = NO ERROR
DO    ADD   ( FOR EACH super-module OF sub-module
                start INTEGRATIONTEST(sub-module, super-module) )
      TO LIST-OF-POSSIBLE-ACTIONS(sub-module)
```

Then the designer interacts with each design object to select the next step in the design process. This approach may be successful but the real complexity is hidden in the compiler. From our point of view, the usage of rules to describe the semantics of a concept in the design process model is at least questionable. Aren't other approaches (functional specification, petri nets) more promising because of their well founded theoretical basis? Furthermore, the requirement of a compiler levels also the argument that rule bases can be extended easily even at run-time opposed to other paradigms. Another point is: if rules are used as the 'object code' of a compilation process, are they used according to the ideas of rule processing or will they be used as predefined pieces of code copied into the right places? As we think, the latter will happen, we conjecture that ECA rules are one possibility to implement a run-time system of the design-process model, but alternatives are available.

4.3 Using ECA Rules to Achieve Flexible Error Correction

As already mentioned (Sect.3), we suppose ACID transactions to encapsulate design steps (tool runs). In a CAD framework some sort of integration layer for tools is required. Up to now, the focus of these layers was often restricted to problems of data provisioning and propagation. We think, it also has to cope with the situation that tools do not terminate successfully. Fig. 1 depicts a simple design step. The tool application itself is encapsulated in a transaction which is not (automatically) terminated at the end of the tool run; we distinguish three different outcomes:

- The tool ends without an error condition, and the data generated fulfills the post-condition of the transition. In this case, the manipulated objects reach the state S_{n+1} which will be the basis for further manipulations.
- The tool ends with an error condition, e.g., the designer aborts editing a file. The simple approach is to rollback the transaction and restart in state S_n. But for some tools (mostly those which abort due to an automatically detected data inconsistency) it may be possible to adjust the object under design using a small set of (interactive) ECA rules so that, nevertheless, the goal (S_{n+1}) can be achieved.
- The tool ends without an error but the resulting data violates the post-condition of the transition in some sense. If it is possible to remedy this violation (maybe with designer interaction) such a situation is also a good candidate for the usage of a small rule base developed by the tool integrator especially for the particular transition (S_n, S_{n+1}).

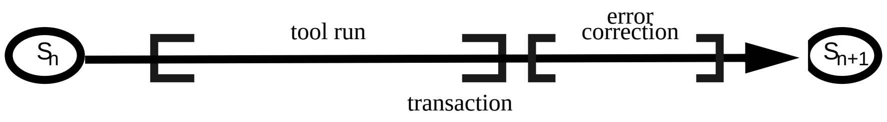

Fig. 1: A single tool run

With respect to these outcomes, why do ECA rules seem useful and adequate in such situations? As already mentioned, building a CAD framework requires the integration of tools not developed with a complete development cycle in mind. As long as the tool and the generated data are congruent with the overall design process, it is fine. But a CAD framework requires also an adequate exception handling. From our point of view, ECA rules provide capabilities to implement such an exception handling, because the termination of a tool (positively or negatively) may serve as the basic event, the condition part of rules may test various post-conditions and the actions may provoke (interactively) some repair actions to save the work already carried out. Hence, ECA rules provide the features of exception handling in a natural way. Furthermore, the defect of large rule sets being unmanageable does not apply to this application: The rules are structured according to the transition they are bound to, and the size of the rule set per transition remains small. Another problem of rule systems, namely the overall semantics of a set of rules, is also not relevant to this application scenario: The designer is a well educated engineer, so at any situation within the rule processing where a relevant decision is to be made he/she may be asked to provide (at least hints for) the right decision. For situations where the actual (intermediate) state provides enough information to deduce S_{n+1}, the reasoning possibilities of (E)CA rules may be used without user intervention.

So, the usage of ECA rules is promising in this area. However, the task of a good design of the required rule sets remains a difficult part during the integration of tools into the framework as well as during the definition of a specific designflow.

4.4 Using ECA Rules to Pre-Plan Cooperation

The CONCORD model (see Sect. 3) introduces DAs (design activities) as explicit operational units serving for the processing of design tasks or sub-tasks. Since design tasks usually cannot be completely pre-planned but require designer's creativity, we associate designers with DAs, who are working together with the designflow system in order to fulfill the corresponding design tasks. One possible way of cooperation between designers working on different design tasks is the exchange of preliminary, but in some application-specific sense meaningful design data among designers. This can be performed by granting cooperating designers access to certain private design-object states[2]. Here, people may see a further potential application area for ECA rules. A simple example could be the following:

```
ON    end-of-integrationtest(module)
IF    success(integrationtest(module))
DO    PERMIT (DA1, DA2, module, derivation-access)
```

This rule specifies an application-specific, cooperative-access control: as soon as the considered DA reaches a certain design state a certain kind of access (derivation of new versions) to a certain design object state is (automatically) granted to a cooperating DA. From our point of view, ECA rules may be used to pre-plan cooperative actions, but they are not especially suited to support this feature. Here, again, the usage of rules does not agree with the original ideas of rule processing. An alternative approach is to associate cooperation operations with design transitions, e.g. within a script. This alternative is at least as adequate as the exploitation of ECA rules.

A similar situation is given, when the designer associated with a DA wants to use rules to handle dynamic requests from other DAs, which often occur asynchronously to her/his design work. An example could be the following: when a request from a certain cooperating DA comes in and the corresponding design state has already been reached, the requested access is automatically granted without asking the designer. Here, again, ECA rules may be helpful, because they provide an intuitive and natural way of specifying reactions, but it is obvious that we are not considering an actual rule process, because a single event is handled by a single rule.

2. This is only one in a variety of cooperation control mechanisms provided by the CONCORD model [26].

4.5 ECA Rules Handling Inter-Task Dependencies

A problem inherent to task-splitting (cf. DA hierarchy, Sect. 3) is to take care that the partial solutions will fit together and that it will be possible that their integration will establish a consistent solution of the corresponding overall problem. In the literature, inter-task-dependencies have been addressed and several types of inter-task dependencies have been defined [1, 22, 7, 9]. For example, in [22] state dependencies are handled by special rules. The following rule is an example, allowing DA2 to start processing as soon as DA1 succeeds:

```
ON    DA1 reached final state
IF    (success(DA1.ouput)) AND (outval > 5)
DO    DA2.input := DA1.output, ENABLE(DA2);
```

This type of dependencies can only be regarded in application areas, where task-splitting and workflow specification can completely be done in advance. Since design applications do not allow a complete pre-planning of task-splitting (and corresponding designflow-specifications), such rules could just be applied in special cases. Handling these cases by rules, however, would result in describing (parts of) the process by using ECA rules, but this is not adequate as already argued in Sect. 4.1. We prefer to handle such constraints explicitly within a script, or dynamically through issuing delegation operations (creating sub-DAs, Sect. 3).

Besides process controlling aspects, inter-task dependencies also pertain design object properties, i.e. design quality. Dependencies between design object states of sibling DAs may become obvious already during the specification of the corresponding tasks and designflows. Furthermore, it is often possible to specify correction actions for the case that certain conditions are not fulfilled. A simple example could be the following. Two DAs are created to design a module each. Both DAs have derivation access to a sub-module, which has to be used by each of the modules to be designed by the DAs. Now, so-called *rendezvous points* can be defined for the two DAs. A rendezvous point refers to single design states in the designflow specifications of each of the DAs. A specified rendezvous point is handled by the system by blocking the DA which firstly reaches the corresponding design state until the other DA(s) have also reached their respective states. Reaching this situation creates an event, so that a rule can test dependencies between the design object states of the considered DAs. There are several ways of correcting detected inconsistencies. On one hand, designers can be informed and it can be left to them to start repair actions. On the other hand, a rule process can initiate these repair actions. Here, we have a similar situation as the one discussed in Sect. 4.3, and the same reasons apply for ECA rules being an adequate mechanism to repair inconsistencies.

4.6 ECA Rules to Implement ‘Correct’ Cooperation

Looking for further application areas for rules in design, people could mind exploiting rules to specify correctness criteria for cooperation. In the literature, many advanced transaction models ([10] gives a good overview) are proposed and new approaches try to provide frameworks for the specification of relaxed/extended transaction models [7, 16, 3, 17, 23]. Some of them, e.g. [17], consider ACID transactions as building blocks and exploit rules to express dependencies between transactions, others, e.g. [3, 23], provide primitives of transaction management and allow the specification of relaxed correctness criteria. These approaches, even those exploiting rules as specification mechanism, are well suited to specify extended transaction mechanisms for special environments. This means that specialists for transaction management use these concepts to specify the transactional model and the application programmer just has to be aware of the (newly specified) transaction manager's interface.

There is not much knowledge whether or not the specification mechanisms used in the above approaches can also be used in the design area to specify cooperation. In our opinion, however, the concepts in defining allowed patterns of cooperation must be similar to those used to specify designflows. For that reasons, rules do not seem to be adequate for specifying any kind of cooperation (see discussion in Sect. 4.1).

4.7 ECA Rules Performing Notifications

Due to the inherent complexity and the high dynamics of design problems it is frequently not possible to specify automatic reactions to all situations that may occur w.r.t. cooperation. For being able to handle such cases, a notification concept is necessary. We distinguish two classes of notifications. The first class contains notifications which are initiated automatically by the designflow management system. This is feasible for example, if access rights for meaningful design object states are changed (e.g. withdrawn). The second class contains notifications which are wanted and, therefore, especially specified by the designer. For example a designer might want to be informed, when a cooperating DA reaches a certain state.

The mentioned situations, in which notifications are helpful, can be addressed by special events. Thus, the following situations can be considered as special events: a DA is created or terminated; a DA reaches a new design state; specifications, e.g. designflow specifications, are changed; design objects are manipulated in a certain way, etc. Notifying designers about these situations can easily be handled by very simple rules. We just need EA rules, where the action is simply given by sending a notification to the corresponding designer. Although we do not have a typical situation for rule processing, EA rules are a natural and intuitive mechanism for the designer to specify wanted notifications.

4.8 Using ECA Rules to Maintain Design Data Consistency

As already mentioned several times, the ACID principle protecting the execution of design tools provides basic consistency for design data. At the higher levels of the CONCORD model (see Sect.3), we have a more application-specific notion of consistency concerning the quality of design data which is to be stepwise improved until the design goal is reached. To assure stepwise improvement, or to prevent worsening, respectively, the usage of rules can be considered. This usage, however, would lead to the problems discussed in Sect. 4.1 (implicit modeling of design states). Excluding the data driven approach (using ECA rules in the way mentioned in Sect. 4.1), maintaining design data consistency just means enforcing special assertions so that no rules are needed.

5 Conclusions

The main goal of this paper was to examine the potentials of *active* DBMS (ECA rules) in design environments. The presented, somewhat abstract models were needed to guide the discussion. As a conclusion, we can classify the examined application areas into three groups.

For a first group of applications, ECA rules do not provide an adequate support. The ECA paradigm may be used as a *supporting* mechanism but is inadequate when used without a 'surrounding' model which guides the rule development as well as the rule processing. This is especially valid for the areas 'design-process description' (Sect. 4.1) and 'achieving application-specific design-data consistency' (Sect. 4.8) but also holds for 'handling inter-task dependencies' (Sect. 4.6) as far as ordering of tasks is concerned.

Some of the discussed areas are indifferent w.r.t. the application of ECA rules; it is possible to use rules, but there are no striking arguments in favor for them ('design-process implementation' (Sect. 4.2) and 'pre-planning cooperation' (Sect. 4.4)).

The promising fields for the application of ECA rules in a design environment, 'achieving flexible error correction' (Sect. 4.3), 'performing notifications' (Sect. 4.7) and 'handling inter-task dependencies' (Sect. 4.6), share some common characteristics:

- The area requires flexibility and dynamic adaptability.
  In general, production rules are used in application areas where high flexibility is required and new knowledge, arising during run-time, has to be incorporated dynamically (e.g. changing the rule base). A sample area is handling inter-task dependencies concerning design-data quality (see Sect. 4.6). Here, constraints between design data of different tasks must be enforced.
- The overall set of rules can be structured in dependence on some application-oriented model.
  In order to gain control over the set of rules it is crucial to follow an application-oriented approach. For example, to achieve a flexible error correction, as discussed in Sect. 4.3, a single transition specification must be supplemented by a special set of rules handling the error correction for that transition. In this way, the overall set of rules can be structured and the subset of rules, which are enabled at the same time, becomes (or remains) manageable.

- The execution of rules does not change (design) data or can be guided by the designer.
  Facing the complexity of the application area and the difficulties inherent to the processing of rule sets, the execution of rules has to be safe in some sense. Within a design environment, we think a guarantee of 'safeness' cannot be provided automatically. Hence, either rules are not allowed to change the data (e.g. notifications only) or unclear situations are to be solved by the designer (rules include interactive parts) and not through a 'clever' resolution mechanism.

Furthermore, different applications stress different parts of ECA rules: While flexible control of tool results requires more or less CA rules, cooperation can better be supported by EA rules. Hence, it is questionable whether or not an integrated language is really helpful, especially in cases where CA rules are needed which have to be modeled by a bunch of ECA rules (generating rules for all 'condition-changing' events). Nevertheless the areas of promising usage of ECA rules have to be exploited further. Questions to be answered are:

- Which events are required?
  Due to the complexity of design and/or the various cooperation strategies, complex events will be necessary. Hence, an adequate event language for design environments has to be developed or a general purpose event language (e.g. [6, 14]) has to be adjusted to a specific design domain.
- How to structure rule sets?
  The sets of rules is structured by means of a designflow specification. How large will the error correcting rule sets grow? Will they remain manageable?

Finding answers to these questions are subject to further research. The application of ECA rules within the CONCORD design environment and the corresponding evaluation has to be carried out on the basis of a research prototype.

Acknowledgment

The authors would like to thank Prof. Dr. T. Härder for his helpful comments on an earlier version of this paper.

Literature

1. Attie, P., Singh, M., Sheth, A., Rusinkiewicz, M.: Specifying and Enforcing Intertask Dependencies, in: Proc. 19th VLDB Conf., Dublin, Irland, 1993, pp. 134-145.
2. Baralis, E., Ceri, S., Paraboschi, S.: Modularization Techniques for Active Rules Design, in: ACM TODS, Vol. 31, No. 1, 1996, pp. 1-29.
3. Biliris, A., Dar, S., Gehani, N., Jagadish, H.V., Ramamritham, K..: ASSET: A System for Supporting Extended Transactions, in: Proc. ACM SIGMOD, 1994, pp. 44-54.
4. Branding, H.: Real-time Scheduling in Database Systems: a Survey, in: Proc. Softwaretechnik in Automatisierung und Kommunikation (STAK '94), Ilmenau, March, 1994 (in German).

5. Bußler, C., Jablonski, S.: Implementing Agent Coordination for Workflow Management Systems Using Active Database Systems, in: Proc. 4th Int. RIDE Workshop, Houston, Texas, 1994, pp. 53-59.
6. Chakravarthy, S., Mishra, D.S.: An Expressive Event Specification Language For Active Databases, Technical Report, University of Florida, UF-CIS-TR-94-007, 1993.
7. Chrysantis, P., Ramamritham, K.: A Formalism for Extended Transaction Models, in: Proc. 17th Int. VLDB Conf., 1991.
8. Dayal, U.: Active Database Management Systems, in: Proc. 3rd Int. Conference on Data and Knowledge Bases, 1988, pp. 150-169.
9. Dayal, U., Hsu, M., Ladin, R.: Organizing Long Running Activities with Triggers and Transactions, in: Proc. ACM SIGMOD, 1990.
10. Elmagarmid, A. (ed.): Transaction Models for Advanced Database Applications, Morgan Kaufmann, San Mateo, CA, 1992.
11. Forgy, C.L.: OPS5 Users's Manual, Technical Report, Canergie-Mellon University, CMU-CS-81-135,1988.
12. Forgy, C.L., McDermott, J.: OPS - A Domain-Independent Production System Language, in: Proc. 5th Int. Conf. on Artificial Intelligence, Cambridge, 1977.
13. Gallaire, H., Minker, J., Nicolas, J.M.: Logic and databases: A deductive approach, in: Computing Surveys, Vol. 16, No. 2, 1984, pp. 153-185.
14. Gatziu, S., Dittrich, K.R., Events in an Active Object-Oriented Database System, in: Proc. 1st Int. Workshop on Rules in Database Systems, Edingburgh, 1993.
15. Gatziu, S., Geppert, A. Dittrich, K.R.: The Active Database Management System Manifesto: A Rulebase of ADBMS Features, in: Proc. 2nd Workshop on Rules in Databases, 1995.
16. Georgakopoulos, D., Hornick., M., Krychniak, P., Manola, F.: Specification and Managament of Extended Transactions in a Programmable Transaction Environment, in: Proc. 10th Int. IEEE Data Engineering Conference, Houston, Texas, 1994, pp. 462-477.
17. Günthör, R.: Extended Transaction Processing Based on Dependency Rules, in: Proc. 3rd Int. RIDE Workshop (RIDE-IMS'93), Int. Workshop on Multidatabase Systems, Vienna, 1993.
18. Härder, T., Hübel, C., Meyer-Wegener, K., Mitschang, B.: Processing and Transaction Concepts for Cooperation of Engineering Workstations and a Database Server, in: Data and Knowledge Engineering, Vol. 3, 1988, pp. 87-107.
19. Härder, T., Reuter, A.: Principles of Transaction-Oriented Database Recovery, in: ACM Computing Surveys, Vol. 15, No. 4, 1983, pp. 287-318.
20. Jasper, H., Active Databases for Active Repositories, in: Proc. 10th Int. IEEE Data Engineering Conference, Houston, Texas, 1994.
21. Käfer, W., Ritter, N., Schöning, H.: Configuration-Concepts for DB-based Design Environments, University of Kaiserslautern, 1997 (accepted for publication, in German).
22. Krishnakumar, N., Sheth, A.: Specification of Workflows with Heterogeneous Tasks in METEOR, in: Poster Paper Collection, 20th Int. VLDB Conf, Santiago, Chile, 1994.
23. Nodine, M.H., Zdonik, B.: Cooperative Transaction Hierarchies: Transaction Support for Design Applications; VLDB Journal 1, 1992, pp. 41-80.
24. Reinert, J.: A Rule System for Integrity Maintenance in Active Relational Database Systems, Doctoral Thesis, University of Kaiserslautern, April, 1996 (in German).

25. Ritter, N., Mitschang, B., Härder, T., Gesmann, M., Schöning, H.: Capturing Design Dynamics - The CONCORD Approach; Proc. 10th Int. IEEE Data Engineering Conference, Houston, Texas, 1994, pp. 440-451.
26. Ritter, N.: DB-based Cooperation Services for Technical Design Applications, Doctoral Thesis, University of Kaiserslautern, May, 1997 (in German).