Rule Contexts in Active Databases
- A Mechanism for Dynamic Rule Grouping

Martin Sköld, Esa Falkenroth, Tore Risch
Department of Computer and Information Science, Linköping University
S-581 83 Linköping, Sweden
e-mail: {marsk,esafa,torri}@ida.liu.se

Abstract. Engineering applications that use Active DBMSs (ADBMSs) often need to group activities into modes that are shifted during the execution of different tasks. This paper presents a mechanism for grouping rules into *contexts* that can be activated and deactivated dynamically. The ADBMS monitors only those events that affect rules of activated contexts.
By dynamic rule grouping the rules to be monitored can be changed during the transactions. This can be contrasted by static rule grouping where the rules are associated with specific objects during the schema definition.
A rule is always activated into a previously defined context. The same rule can be activated with different parameters and into several different contexts. Rules in a context are not enabled for triggering until the context is activated. However, rules can be directly activated by activating them into a previously activated context. When rule contexts are deactivated all the rules in that context are disabled from triggering.
User defined contexts can be checked at any time in a transaction. Rule contexts can be used as a representation of coupling modes, where the ADBMS has built-in contexts for immediate, deferred, and detached rule processing. These built-in coupling modes are always active and are automatically checked by the ADBMS.
Contexts and rules are first-class objects in the ADBMS. Database procedures can be defined that dynamically activate and deactivate contexts and rules to support dynamically changing behaviours of complex applications.
The context mechanism has been implemented in the AMOS ADBMS. The paper concludes with an example of a manufacturing control application that highlights the need for rule contexts.

1 Introduction

A system for building manufacturing control applications was implemented using an ADBMS [10]. In the system active rules control the manufacturing tasks. Details about the system and examples of active rules are presented in section 4. Experiences from this system integration are:

- These type of engineering applications need to group activities into modes that are shifting during the execution of different tasks.

- Since the ADBMS did not initially have mechanisms for handling mode changes the application had to implement this functionality by introducing state variables in the rule conditions.

- The state variables caused the rules to become complex and unintuitive. A rule would often need to refer to several different state variables.

- The state variables represent control knowledge. It is better to separate rules representing domain knowledge from rules for control knowledge, e.g. by defining *meta-rules* [1][2].

- Implementing mode changes by altering state variables is inefficient since the total number of simultaneously monitored rules will be unnecessary large. By having the ADBMS support mode changes internally the overhead for rule checking can be kept low.

This paper presents an ADBMS mechanism for dynamically grouping rules into *contexts* that are activated and deactivated dynamically. The contexts are associated with different modes in the applications. When the application shifts between modes, the ADBMS is ordered to shift attention, or *focus*, to the associated rule context. Shifting between contexts means that all rules in the old context are ignored and the rules in the new context are monitored instead. Applications sometimes need to work with modes on different levels and one mode can consist of many sub-modes in a hierarchical manner. This means that the ADBMS must be able to handle several rule contexts simultaneously and to support modelling of contexts in the schema. This is accomplished by defining contexts (and rules) as first class objects in the ADBMS. This approach also supports the definition of *meta-rules* that are defined over rules and contexts.

Applications must not only have the possibility to create and delete contexts and activate and deactivate them, but must also be able to control when the rules are to be checked. For example, the application might initiate a series of operations and then check if any rules were triggered. This usually falls outside of the general coupling modes defined in ADBMSs. Our contexts therefore have *rule processing points,* which allow applications to define their own coupling modes where the rules can be checked at a user specified time in a transaction.

The contexts are also used internally in the ADBMS to implement system coupling modes. System coupling modes are associated with predefined contexts that are automatically checked by the system.

The paper presents rules and rule contexts as implemented in the AMOS (Active Mediating Object System)[5][13] ADBMS. The paper concludes with an example of a manufacturing control application that highlights the need for rule contexts.

2 Related Work

The idea of grouping rules dynamically into different contexts was initially developed in expert systems[1][17]. Other names for these groups of rules include *worlds* and *viewpoints*. In expert systems these rule contexts are usually used for organizing different *hypothesis* during a deduction process. In an ADBMS the issue is more of organizing different *activities*.

The contexts were also supported in the rule based expert system Mycin and its successor Oncosin [1]. In Mycin contexts had to be specified as special context variables in rule conditions; in Oncosin a special CONTEXT clause on each rule referred to the context variables. By contrast our contexts completely separate the context specifications (i.e. the control information) from the rules (i.e. the knowledge) and therefore the same rule can occur in many contexts with different control strategies.

The rules in an ADBMS are often defined as first-class objects in the database schemas [3]. In Object Oriented (OO) systems the rules can often be grouped as belonging to a class and rules can be associated with other classes similarly as class methods. KEE [8] used this model for grouping rules into *worlds*. This classification is useful when associating rules with specific objects statically, e.g. when associating some constraint on the possible values of a class attribute or reacting to a user defined event that is associated with an object. These kinds of rules are usually always active and are triggered when a method is invoked of an instance of the class. However, in many applications there is a need to dynamically group rules that are associated with many different classes of objects.

Both POSTGRES[15] and Starburst[18] allow rules to be members of *rule sets,* which can be ordered hierarchically and where complete rule sets can be activated and deactivated. Rule sets are checked at certain *rule processing points*. The contexts in AMOS are more dynamic since the same rule can be activated in different contexts for different parameters, i.e. for different object instances. The contexts are objects and thus can be stored in any data structure and can be used for relating different data to different contexts. In AMOS contexts are also used for defining built-in coupling modes for rule execution. This means that these contexts have rule processing points that are automatically executed by the system. Since the same rule can be activated in different contexts the same rule can also be given many coupling modes.

In [16] a model is presented for defining applications in terms of *brokers* that represent reactive system components and *roles* that specify the responsibilities of brokers in various situational and organizational contexts. A proposal was made to implement rules using rules and special role dependant state variables. As was mentioned earlier, we believe this kind of modelling is better supported by rule contexts in the ADBMS.

We define rule contexts as first-class objects that enable parameterizing functions with contexts, organizing them hierarchically in data structures, and defining rules that manage (create/delete, activate/deactivate) other contexts than their own. This makes it possible to define *meta-rules* as in [1][2] where the meta data consists of other rules and contexts.

3 Rules and Contexts in the AMOS Active DBMS

The active rules have been introduced into AMOS[5][13], an Object Relational DBMS. The data model of AMOS is based on the functional data model of Daplex[14] and Iris[6]. AMOSQL, the query language of AMOS, is a derivative of OSQL[11]. The data model of AMOS is based on objects, types, functions, and rules. Everything in the data model is an object, including types, functions, and rules[3]. All objects are classified as belonging to one or several types, i.e. classes. Functions can be stored, derived, or foreign. Stored functions correspond to object attributes in an Object Oriented system and to base tables in a relational system, derived functions correspond to methods and relational views, and foreign functions are functions written in some procedural language[1]. Database procedures are defined as functions that have side-effects. AMOSQL extends OSQL[11] with active rules, a richer type system, and multidatabase functionality.

3.1 Contexts

When rules are activated in AMOS, they are always associated with rule contexts. Contexts are first-class objects and are created by the statement:

create context *context-name*

where the *context-name* is a global name. Contexts are deleted by:

delete context *context-name*

Contexts are initially *inactive* which means that before a context is *activated* the events affecting its rules are not monitored (unless the events are monitored by another already active context). Contexts are activated by:

activate context *context-name*

which enables all the activated rules in the context to be monitored. Contexts are deactivated by:

deactivate context *context-name*

which disables all the activated rules in the context from being monitored. Two built-in contexts, named `deferred` and `detached`, are predefined and always active for deferred and detached rules, respectively. These are checked automatically by the system. Deferred rules are checked immediately before transaction-commit and detached immediately after.

3.2 Rules

AMOSQL supports Condition Action (CA) rules. The Condition is defined as an AMOSQL query and the Action as an AMOSQL procedural expression.
The syntax for rules is as follows:

create rule *rule-name parameter-specification* **as**
  when *for-each-clause* | *predicate-expression*
  do *procedure-expression*
where
for-each-clause ::=
  for each *variable-declaration-commalist* **where** *predicate-expression*

1. In AMOS foreign functions can be written in Lisp or C.

The *predicate-expression* can contain any boolean expression, including conjunction, disjunction and negation. Rules are deleted by:

delete rule *rule-name*

Rules are activated and deactivated separately for different parameters. Rules are activated in different contexts, where the default context is the `deferred` context:

activate rule *rule-name parameter-list* [**strict**] [**priority** 0|1|2|3|4|5] [**into** *context-name*]

deactivate rule *rule-name parameter-list* [**from** *context-name*]

The semantics of a rule in an active context is as follows: If an event in the database changes the truth value for some instance of the condition to *true*, the rule is marked as *triggered* for that instance. If something happens later in the transaction which causes the condition to become false again, the rule is no longer triggered. This ensures that we only react to net changes, i.e. *logical events*. A non-empty result of the query of the condition is regarded as *true* and an empty result is regarded as *false*. Since events are not monitored in inactive contexts, rules in them will not trigger until the context is activated and some event happens that causes the condition to become true. *Strict* rule processing means that the system guarantees that a rule is triggered only once for each change that causes its condition to become true. Rule priorities can be used for defining conflict resolution between rules that are triggered simultaneously in the same context.

3.3 Rule Contexts and Rule Processing Points

Each context in AMOS has a separate *rule processing point* where the conditions of the rules in the context are checked and where the corresponding actions are executed if the condition is true. (For strict semantics the action is executed only if the condition was false in the previous processing point of the context).

A processing point is either *explicit* or *implicit*. Explicit processing points are issued by explicit calls from applications to a *check* system procedure. Implicit processing points are issued by the ADBMS at specific database states, e.g. just before (deferred rule processing) and after (detached rule processing) each commit point.

Rule contexts can be used as a representation of coupling modes [4]. The coupling modes are defined as named contexts with implicit processing points. All rules that are activated in the same context also have the same coupling mode, i.e. the same rule processing point. Traditionally coupling modes have been associated directly with individual rules. By associating the coupling modes with rule contexts a more flexible model can be achieved. Since rules can be activated into several contexts they can also be given several coupling modes. Coupling modes for *immediate*, *deferred*, and *detached* rule processing can be defined as built-in contexts that are automatically checked by the transaction mechanism of the ADBMS (fig. 1). In [4] a separation was made between E-C and C-A coupling modes. When we refer to immediate coupling mode here, we really mean immediate-immediate, and by deferred we mean deferred-deferred. Contexts for other E-C and C-A combinations could also be defined. Immediate rule checking is currently not supported in AMOS, but its processing points would have to be just after (or before) triggering database operations. User defined contexts with explicit processing points can be checked at any time within a transaction. The detached coupling mode is important in a multidatabase architecture like AMOS. In such an architecture one agent or broker may need to monitor the behaviour of another agent[12]. This monitoring must be made on commited data. By using a detached coupling mode the rules that perform the monitoring will never trigger on uncommitted changes.

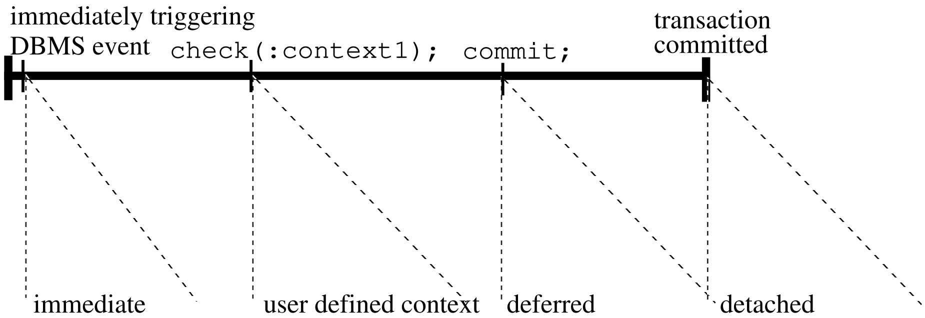

Fig. 1. Contexts as a representation of coupling modes

Decoupled and *causally dependent decoupled* coupling modes [4] can be implemented using general transaction mechanisms for creating sub-transactions and synchronizing transactions.

4 A Manufacturing Control System

The need for a context mechanism became apparent when an ADBMS was used in the implementation of a system for building manufacturing control applications [10]. ARAMIS (A Robot And Manufacturing Instruction System) [9] is a high-level language and a set of tools for designing intelligent behaviour of control systems. The ARAMIS language has similarities with workflow languages [7], but is oriented towards specifying the high-level activities of control applications. The low-level control programs that interact with the physical hardware are isolated from the application programmer by the World Model (WM) metaphor. All the objects in the model of the manufacturing task can be observed and manipulated as objects in the WM. The original ARAMIS system [9] was fully implemented (controlling a robot with various sensors), but with a primitive ADBMS. In [10] a three-level architecture combining the ARAMIS language and an ADBMS is presented. In CAMOS (Control Applications Mediating Object System), see fig. 2, a manufacturing task is expressed in a high-level task language that is partly compiled into an AMOS database that stores the WM and monitors changes to the objects in the WM.

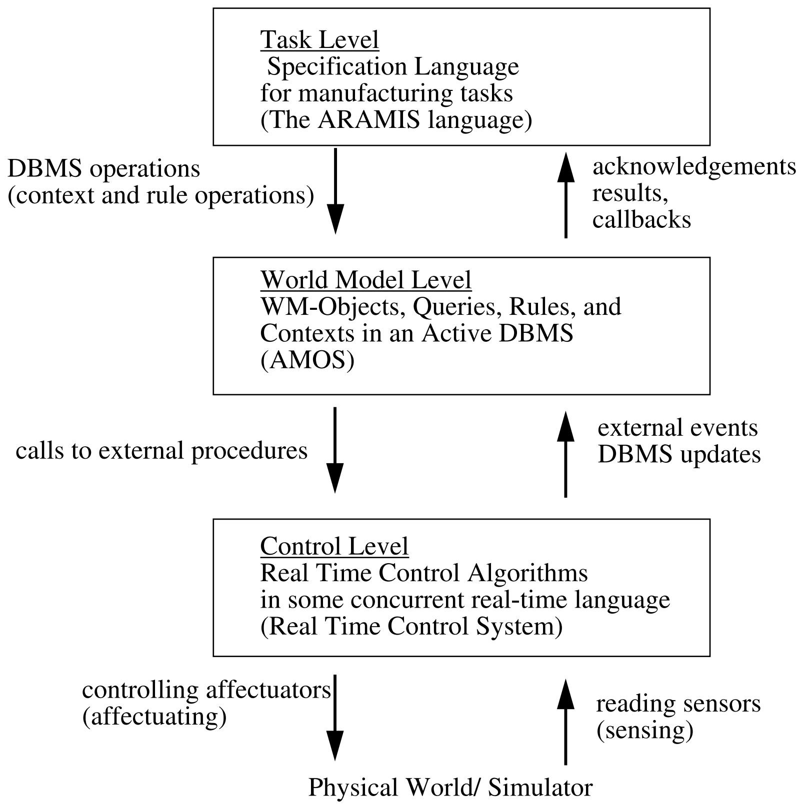

Fig. 2. The three-level architecture of CAMOS

The WM is synchronized with a *physical world* or a *simulator* by cooperation between a *control system* and an ADBMS through a servo mechanism. When the task level updates the WM, the control level affects the physical world to correspond to the WM. Likewise, when the control level sense changes in the physical world, it updates the WM. In the CAMOS architecture the high-level query language and active rules of AMOS are used to support much of the functionality in the WM, e.g. to monitor changes to the WM. Parts of this architecture have been implemented to verify the ideas. Instead of using actual hardware, a simulator of a production cell was used[1]. In the initial implementation state variables were used to model mode changes. Below follows an example of how rule contexts in AMOS can be used instead.

1. Based on a simulator developed by Artur Brauer at University of Karlsruhe

4.1 A Production Cell Simulation

A production cell consisting of a two-armed robot, an elevating rotary table, a press, a crane, and two conveyor-belts produces body parts for cars (fig. 3). Unprocessed parts arrive from the left on the lower conveyor-belt and are transported to the elevating rotary table that puts them into gripping position for the first arm (`:arm1`) of the robot. The robot moves a part to the press that presses it into a finished body part. The robot then moves the part, using the second arm (`:arm2`), to the top conveyor-belt that moves to the left. A crane finally picks up the parts and place them on a pallet (lower left of fig. 3).

This is an application that requires a database for storage data relating to the different parts in stock and also active database management for the actual control of the production task. Another requirement is that the setup should be flexible and the production cell should easily be reconfigured for production of different parts.

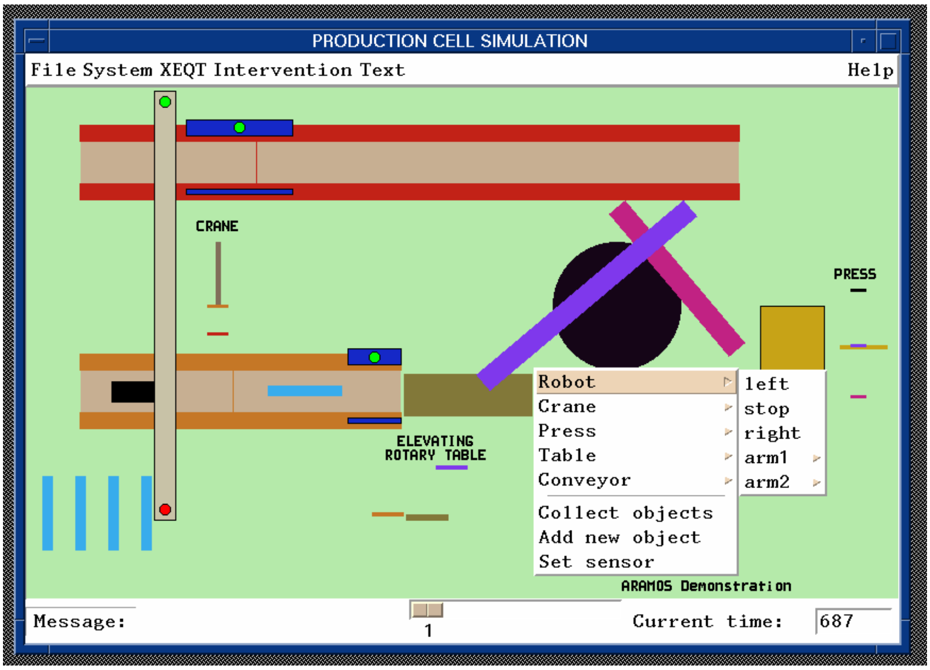

Fig. 3. A top-view of a simulated production cell for manufacturing car body parts

Take a scenario where the production cell can alternate between the production of two different parts. This can be modelled by two different contexts (fig. 4). Each context is used to relate to data needed for each part. Rules that are specific for each different part are activated into the respective contexts. Sub-contexts can also be defined for different activities within the cell. This is illustrated here by two contexts used in both production tasks, one for rules relating to the elevating rotary table and one for the press. There will of course be more contexts and rules, but these are enough to illustrate the idea.

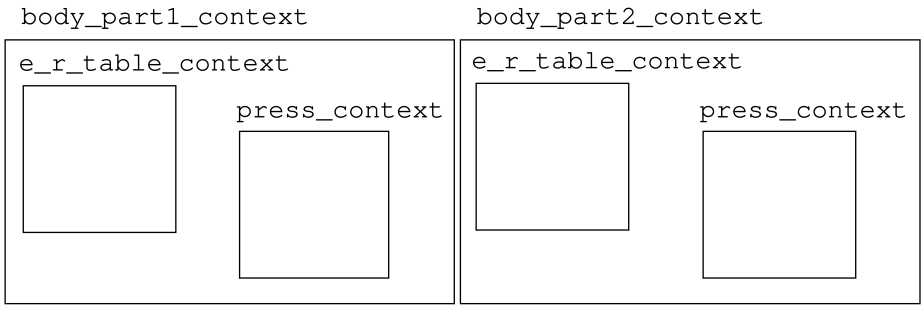

Fig. 4. Example contexts for producing two different parts and two general subcontexts

An example of a task program for producing `part1` can be seen in fig. 4. It is a cyclic program that keeps producing parts until it is stopped explicitly.

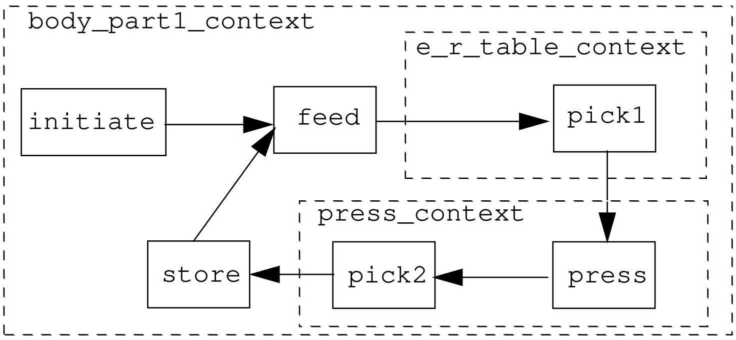

Fig. 5. An example of a task program for producing `part1`

Below follows part of an example schema in AMOSQL that illustrates the example above. The two main contexts are first defined followed by a context for the elevating rotary table. A rule that defines when the robot can grip a part on the table is activated into the context for the first arm of the robot (`:arm1`). A context for the press is then created along with a rule that specifies when it is safe to operate the press. The first rule is also activated into this context, but for the second arm of the robot (`:arm2`) instead. It specifies when the robot can grip an object in the press. Here follows extracts of the context related parts of the schema for this application:

```
create context body_part1_context;
/* Definitions of rules related to part1 */
...
create context body_part2_context;
/* Definitions of rules related to part2 */
...
create context e_r_table_context;
create rule grip_rule(robot_arm a) as
            when for each part prt
            where above(position(a), prt)
            do robot_grip(a, prt);
activate rule grip_rule(:arm1) into e_r_table_context;

create context press_context;
create rule press_rule(robot r, press p) as
            when for each robot_arm a
            where a = arm(r) and
                   outside(position(a), p) and
                   part_in_press_position(p)
            do close_press(p);
activate rule press_rule(:robot, :press) into
press_context;
activate rule grip_rule(:arm2) into press_context;
```

During the execution the task program for producing `part1` the order of database operations initiated from the task level might be:

```
activate context body_part1_context;
...
         check(:body_part1_context);
         ...
         activate context e_r_table_context;
         ...
         check(:e_r_table_context);
         ...
         deactivate context e_r_table_context;
         ....
         activate context press_context;
         ...
         check(:press_context);
         ...
         deactivate context press_context;
```

5 Conclusions and Future Work

The paper presented rule contexts as a mechanism for dynamically grouping rules. Rules are activated into contexts and are deactivated from contexts. When a context is activated it enables all its rules for monitoring. In deactivated contexts all the rules are disabled from being monitored. Events are only monitored if they are referenced from some rule in an active context.

Contexts are used to represent coupling modes where all rules in the same context also share the same coupling mode. Predefined contexts are defined for the usual system coupling modes.

Contexts are first-class objects, which makes it possible to store them in any data structure and define meta-rules that activate and deactivate them.

Future work includes investigating the need for several contexts belonging to the same coupling mode. This will cause a need for ordering the execution order of different contexts. Using priorities is one way of doing this, but since the conflict resolution between different rules inside the same context is also done with priorities this might lead to an unnecessary complicated model.

The issue of event consumption is also important. If checking of one context consumes events then rules in consecutively checked contexts might not trigger the way they were intended.

Defining meta-rules that manage other contexts is another subject for future research.

6 References

[1] Buchanan B. G., Shortliffe E. H.: Rule-based Expert Systems, *The Mycin Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, 1984

[2] Davis R., Meta-rules: Reasoning about Control, *AI*, vol. 15, 1980, pp. 179-222

[3] Dayal U., Buchman A.P., McCarthy D.R.: Rules are objects too: A Knowledge Model for an Active, Object-Oriented Database System, *Proc. 2nd Intl. Workshop on Object-Oriented Database Systems*, Lecture Notes in Computer Science 334, Springer 1988

[4] Dayal U., McCarthy D.: The Architecture of an Active Database Management System, *ACM SIGMOD conf.*, 1989, pp. 215-224

[5] Fahl G., Risch T., Sköld M.: AMOS - An Architecture for Active Mediators, *Intl. Workshop on Next Generation Information Technologies and Systems (NGITS '93)* Haifa, Israel, June 1993, pp. 47-53

[6] Fishman D. et. al: Overview of the Iris DBMS, *Object-Oriented Concepts, Databases, and Applications*, ACM press, Addison-Wesley Publ. Comp., 1989

[7] Georgakopoulos D., Hornick M., Sheth A.: An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure, *Distributed and Parallel Databases*, 3, 2, April 1995, pp. 119-153

[8] Hedberg S., Steizner M.: Knowledge Engineering Environment (KEE) System: Summary of Release 3.1, Intellicorp Inc. July 1987

[9] Loborg P., Holmbom P., Sköld M., Törne A.: A Model for the Execution of Task-Level Specifications for Intelligent and Flexible Manufacturing Systems, *Integrated Computer-Aided Engineering* 1(3) pp. 185-194, John Wiley & Sons, Inc., 1994

[10] Loborg P., Risch T., Sköld M., Törne A., Active Object Oriented Databases in Control Applications, *19th Euromicro Conference of Microprocessing and Microprogramming*, vol. 38, 1-5, pp. 255-264, Barcelona, Spain 1993

[11] Lyngbaek P., OSQL: A Language for Object Databases, tech. rep. HPL-DTD-91-4, *Hewlett-Packard Company*, Jan. 1991

[12] Risch T.: Monitoring Database Objects, *Proc. VLDB conf.* Amsterdam 1989

[13] Risch T., Sköld M.: Active Rules based on Object Oriented Queries, *IEEE Data Engineering bulletin*, Vol. 15, No. 1-4, Dec. 1992, pp. 27-30

[14] Shipman D. W.: The Functional Data Model and the Data Language DAPLEX, *ACM Transactions on Database Systems*, 6(1), March 1981

[15] Stonebraker M., Hearst M., Potamianos S.: A Commentary on the POSTGRES Rules System, *SIGMOD RECORD*, vol. 18, no. 13, Sept. 1989

[16] Tombros D., Geppert A., Dittrich K. R.: SEAMAN: Implementing Process-Centered Software Development Environments on Top of an Active Database Management System, *Technical Report 95.03, Comp. Science Dept., University of Zürich*, Jan. 1995

[17] Walters J.R, Nielsen N.R., Crafting Knowledge-based Systems - Expert Systems Made Easy/ Realistic, *John Wiley & Sons*, 1988, pp. 253-284

[18] Widom J.: The Starburst Rule System: Language Design, Implementation, and Applications, *IEEE Data Engineering*, vol. 15, no. 1 - 4, Dec. 1992