The TriGS Active Object-Oriented Database System - An Overview

G. KAPPEL, W. RETSCHITZEGGER

Department of Information Systems, University of Linz, Altenbergerstraße 69, A-4040 Linz, Austria
<{gerti, werner}@ifs.uni-linz.ac.at>

Abstract

The active object-oriented database system TriGS has been developed as part of a larger EC ESPRIT project aiming at the development of next-generation production scheduling and control systems [Huem93]. The goal of this paper is to summarize the work on TriGS which comprises both aspects concerning the development of the active system itself, and guidelines concerning the design of active databases.

1 Motivation

Non-standard application areas like workflow management, multimedia and computer integrated manufacturing require *timely responses to critical situations*, *sophisticated constraint management* and *adaptiveness to changing business policies*. For these reasons, database systems are required to be "*active*" in the sense that they are able to react automatically to certain events by means of knowledge stored in terms of triggers, i.e., Event/Condition/Action rules (ECA rules), inside the database. Besides activeness, "*object-orientation*" is equally important in non-standard application areas, since it helps to intuitively model the universe of discourse and to easily adapt to changing requirements. Consequently *active object-oriented databases (AOODB)* are a commonly accepted solution for smoothly capturing context-dependent and time-dependent organizational knowledge of large enterprises [Ceri96].

Two problems, however, are well known in the research community concerning the development and use of AOODBS. First, the problem of *reflecting the semantics of object-oriented concepts in the definition of ECA rules* and second, the problem of *providing appropriate support for the design of active databases*. To tackle the first problem, we have developed the active object-oriented database system *TriGS (Triggersystem for GemStone)*. To cope with the second problem, we have introduced the notion of so called "*Rule Patterns*" in analogy to the solution metaphor of design patterns in object-oriented system development. In the following, an overview of TriGS and its design environment based on rule patterns is given.

2 Overview of TriGS

TriGS has been implemented on top of the object-oriented database system GemStone and focuses on overcoming the shortcomings of existing active object-oriented systems as well as on advanced requirements of non-standard application areas. In this section, we outline the concepts used in TriGS for specifying (re)active behavior, i.e., its knowledge model, aspects concerning the execution of rules, i.e., its execution model, and some implementation aspects of the TriGS prototype system.

2.1 The Knowledge Model

Like most active systems, TriGS is designed according to the ECA paradigm [Daya88]. Rules and their components are implemented as first-class objects allowing both the definition and modification of rules during run time. Unlike other active systems, TriGS makes explicit use of objects, message passing, inheritance, and overriding to provide a *seamless integration* between rules and an object-oriented data model [Kapp94]. Rules can be *activated at different levels of granularity* ranging from the object instance level to the object class level and on to the application level. Figure 1 shows the basic structure for specifying rules in TriGS using the Backus-Naur Form (BNF). The symbols : : = [ ] are meta symbols belonging to the BNF formalism. Angle brackets denote non-terminal symbols.

```
<rule_definition> ::=
  DEFINE RULE <rule_name> AS
  ON <Esel_C>                              /* Condition event selector*/
       IF <bool_expr> THEN                 /* Condition part */
  [[WAIT UNTIL] ON <Esel_A>]               /* Action event selector */
       EXECUTE [INSTEAD] <action>          /* Action part */
  [WITH PRIORITY <number>]
  [TRANSACTION MODES(C:(serial|parallel),A:(serial|parallel))]
  END RULE <rule_name>.
```

Figure 1. BNF of an ECA Rule in TriGS

The event part of a rule is represented by a *condition event selector ($Esel_C$)* and an optional *action event selector ($Esel_A$)* determining the events (e.g., a machine breakdown) which are able to trigger the rule's condition and action, respectively. Triggering a rule's condition (i.e., an event corresponding to the $Esel_C$ is signaled) implies that the condition has to be evaluated. If the condition evalu-

ates to true, and an event corresponding to the $Esel_A$ is also signaled, the rule's action is executed. If the $Esel_A$ is not specified, the action is executed immediately after the condition has been evaluated to true. By default, the thread of control signaling the condition triggering event is not blocked while the triggered rule is waiting for the action triggering event to occur. Blocking can be specified by the keyword `WAIT UNTIL`.

In TriGS, any message sent to an object may signal a *message event.* Rules incorporating message events are able to monitor the behaviour of objects and can be attached to specific classes or defined independently of any class hierarchy, realizing both, *object-local behaviour* and *system-global behaviour.* Besides message events, rules in TriGS are able to monitor *time events, explicit events*, and also *composite events* similar to the approach described in SNOOP [Chak94]. Composite events consist of *component events* which may be primitive or composite and which are combined by different *event operators* such as conjunction, sequence and disjunction. For each event, a *guard* similar to masks in ODE [Lieu96], i.e., a predicate over the event's parameters, may be specified, which further restricts the events able to trigger a condition or an action, respectively. The condition part of a rule is specified by a boolean expression, possibly based on the result of a database query (e.g., are there some scheduled jobs on the damaged machine?). The action part is specified again in terms of messages (e.g., display all jobs scheduled on the damaged machine and reschedule them on another machine). Considering message-based rules, the keyword `INSTEAD` allows to specify that the action should be executed instead of the method corresponding to the message triggering the condition evaluation.

Figure 2 shows the basic rule structure by means of an example rule which is responsible for monitoring the threshold of an inventory. Condition and action descriptions are given in a self-descriptive pseudocode, closely resembling Smalltalk syntax. Note that, for ease of explanation it is assumed that a class automatically maintains its extent and the extent can be accessed by sending messages to the class itself (cf. `Inventory`).

DEFINE RULE InventoryRule_1 **AS**	Name
ON POST (Inventory,remove:part quantity:quant) **DO**	$Esel_C$
IF (trgObj itemsOfType:(part category)) quant < trgObj constantThreshold **THEN**	C
ON REL (conditionEvent time. MyDateTime endOfCurrentMonth) **DO**	$Esel_A$
EXECUTE conditionEvent trgObj reorder:20 for:(part category)	A
END RULE InventoryRule_1.	

Figure 2. Example Rule

Every time parts are removed from the inventory (determined by the rule's $Esel_C$), the condition checks whether the quantity is below a certain threshold. If true, parts are reordered (denoted by the rule's action) at the end of the current month (determined by the rule's $Esel_A$). The $Esel_A$ is represented by a relative time event specifying the condition evaluation time as the initiating event and the end of the current month as the event which is finally triggering action execution.

2.2 The Execution Model

Concerning the execution of rules, most existing active systems use coupling modes in order to specify time semantics and transaction semantics for relating different rule components. TriGS, in contrast, makes a clear distinction between the specification of time semantics and the specification of transaction semantics allowing an orthogonal definition of these properties. *Time semantics*, on the one hand, is specified by means of an event-based approach [Kapp94] which allows the specification of *arbitrary points in time for condition evaluation and action execution.* This is in contrast to coupling modes, which support only two points in time called immediate and deferred, i.e., at the end of the respective transaction. *Transaction semantics*, on the other hand, is defined by so called *transaction modes*, which can be separately specified for condition and action, respectively (cf. Figure 1). The transaction mode specified for the condition called *EC-Mode* is relative to the condition triggering event whereas the transaction mode specified for the action called *EA-Mode* is relative to the action triggering event.

TriGS supports two transaction modes, *serial* and *parallel.* A serial transaction mode (default) specifies that a condition/action is to be executed directly within the triggering transaction. Note, that some active systems [Gatz95] are using nested transactions [Härd93] instead. This has not been possible in TriGS due to the lack of nested transactions in GemStone. A parallel transaction mode defines that a new independent top-level transaction in which the condition/action is executed is generated as soon as an appropriate event occurs. That is, both the triggering transaction and the rule transaction are allowed to proceed immediately after the event has been signaled. If both, condition and action, are triggered by the same event and for both a parallel transaction mode is specified, condition and action are processed within a single top level transaction. Multiple triggered rules, i.e., conditions and actions of different rules which are triggered at the same time, are scheduled according to priorities. Due to their parallel execution, however, it is not guaranteed that they are also finished in the scheduled order.

2.3 The Prototype System

Research efforts in the area of active object-oriented databases have been put mainly into the development of

knowledge models and execution models without considering the consequences for their implementation. Thus, one part of our work was dedicated to implementation issues within TriGS [Kapp98]. The architecture of the TriGS prototype follows a layered approach, i.e., the active capabilities are implemented on top of the Smalltalk-based object-oriented database system GemStone, which provides traditional database functionality, as is required for an active system [ACT96]. The basic components of the prototype system are illustrated in Figure 3.

Concerning the implementation, one emphasis is put on the detection of composite events whose *components are allowed to span* not only different transactions, but also *different applications* as long as they share the same database. Another emphasis is put on performance issues, in particular for rules which are defined to be executed in parallel.

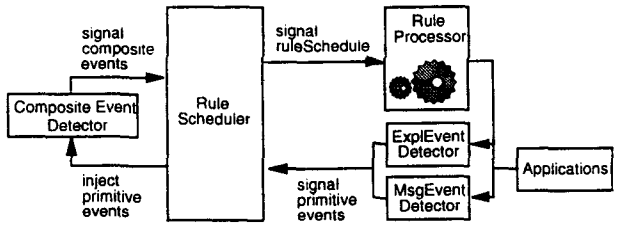

Figure 3. Components of the TriGS Prototype

To realize these goals, first, *composite event detection* as well as *rule scheduling* is done *in parallel* to the event signaling application. Second, events which occur within an application transaction are *immediately signaled*, thus *being visible across transaction boundaries*. However, one has to be aware that increased efficiency is at the cost of reduced reliability due to the relaxed isolation property of the event signaling transactions. Third, rule processing is made efficient by means of *several rule processors*, each running within a separate thread. The number of these threads is dynamically controlled and depends on the utilization of the corresponding rule processors.

3 Designing Active Object-Oriented Databases

Not least since the remarkable panel at the RIDE'94 workshop on active database systems [Wido94] has the design issue of active databases been known as one of the most pressing open research problems. This is largely because of the expressive power and flexibility of existing active object-oriented data models together with a lack of design guidelines on how to apply these models. Active database designers still have to worry about *which parts of an application* should be realized by means of rules as well as *how to do rule design* most efficiently. One reason for this situation is that in object-oriented database systems where behaviour is initially specified together with data, it is not obvious what should be put into rules and what into methods [Kapp96a]. Another reason is that the expressive power of existing active systems often results in several possibilities for implementing one and the same design problem.

Existing approaches on designing active databases mainly adhere to a *top-down approach*. They focus on the graphical representation of rules at the *conceptual level* by integrating them into existing conceptual design models such as extended ER-models [Petr94], [Nava92] and object-oriented models [Bich94], [Grot94], [Rubi94], [Silv96]. *Logical active database design*, i.e., the transformation of a conceptual active model into an active database schema, has been dealt with in only a few approaches [Bich94]. In contrast to all these top-down approaches, we follow a *bottom-up approach* to the design of active databases, thus facilitating reuse of existing, previously developed and already tested design solutions. Our approach concentrates on the logical level of active database design.

3.1 Bottom-Up Design by Means of Rule Patterns

Learning from analogous design problems in object-oriented system development and borrowing their solution metaphor, we introduce rule patterns in analogy to design patterns [Gamm94]. Rule patterns provide templates for the specification of business policies, which allows the *bottom-up design of active databases* [Rets97]. They both categorize rules according to different types of business policies and constraints, and at the same time provide an abstraction mechanism for specifying rules in an application independent manner.

The motivation for introducing rule patterns as abstraction of rules is at least threefold. First, and as already mentioned, there exist no design guidelines for applying rules in application development. Second, the amount of business rules found in simple case studies reported in literature is beyond several hundreds [Knol94]. Clearly, one needs some kind of structure and/or classification in order to manage such an amount of rules. And third, we have found out that rules realizing business policies can be abstracted in that they are no more restricted to a specific application domain but rather can be easily applied to other domains with little adaptation effort. Consequently, when looking at a rule realizing a specific policy, one can find components which are applicable for a number of application domains as well as components specific to a single application domain. Let us consider the following example. A business policy in a warehouse could be that every time the stock-keeper takes out goods, the number of goods in stock has to be checked and, if fallen below a given limit, new goods have to be ordered. A similar business policy in a bank could be that every time a customer withdraws a certain amount from his/her account, the balance is checked and if overdrawn, the bank charges are increased. It can be seen that there are a lot of similarities

between these two policies. Consequently, it is possible to factor out common components valid for both application domains. An abstract formulation of these two policies could be: As soon as a certain value is changed and this value falls below a certain limit some reaction has to be undertaken.

To avoid that the application designer has to consider such fixed, i.e., common components of a rule realizing some business policy again and again for each application domain and to support the adaptation of such a rule to other application domains, rule patterns are introduced. *Rule patterns* are descriptions of rules, predefining certain event/condition/action-pairs. Some rule patterns realize business policies by abstracting from a single rule only. At a higher level of abstraction, however, rule patterns are a *composition* of several rules working together to realize some specific kind of business policy. *Parameterization* makes rule patterns even more general and versatile than they could be otherwise. Parameterized rule patterns consist of components independent from particular applications, i.e., *predefined components*, as well as components specific to particular applications, i.e., *parameterized components*. The kinds of parameters as well as the degree of parameterization, i.e., the relation between parameterized components and predefined components, vary according to the different kinds of policies.

Figure 4 illustrates the process of working with rules and rule patterns.

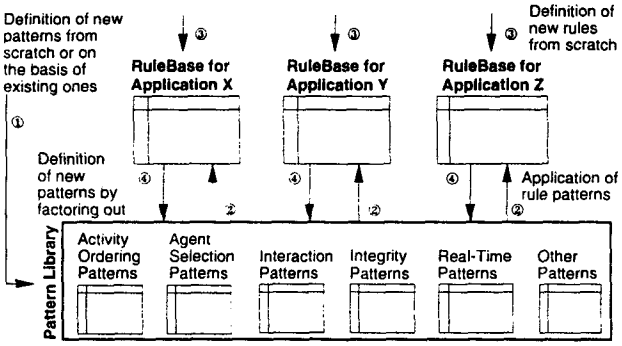

Figure 4. Working with Rules and Rule Patterns

In the long run, for each kind of business policy rule patterns are provided within a pattern library. Within our research prototype, this library is organized as a set of dictionaries wherein the patterns are all stored as first-class objects. At this time, of course, we are far from claiming that our set of rule patterns is complete. Consequently, this pattern library must be extensible by both defining new patterns and specializing existing ones (① in Figure 4). At the same time, existing patterns or parts of them can be easily reused. In order to use rule patterns within an application, they have to be customized by the application designer. This is done by binding the parameters of a selected pattern on the basis of the application semantics. The system guides the application designer during the process of parameter binding by providing on-line help for each required parameter and by restricting the binding alternatives to those that do not contradict the specification of the underlying pattern.

Once a rule has been fully and correctly specified, it can be automatically generated and stored in the rule base attached to the corresponding application (② in Figure 4). Furthermore, it is still possible to specify rules without using a pattern and to store them directly within the appropriate rule base (③ in Figure 4). This is normally done for rules without reusability in mind. If sometimes later an application designer recognizes that a specific design situation recurs and thus is worth to be specified by a corresponding rule pattern, existing rules can be used for this abstraction process (④ in Figure 4).

3.2 A Design Environment for Rules and Rule Patterns

As with all other design pattern approaches in pure object-oriented system development, it is crucial for the successful application of rule patterns that they come along with a proper design and development tool facilitating at least the four tasks depicted in Figure 4. Such a design environment called $\text{TriGS}_{\text{Designer}}$ based on our prototype system TriGS is already operational (cf. Figure 5).

First of all, $\text{TriGS}_{\text{Designer}}$ represents the functionality of a *Rule Editor*, in that it allows the definition of rules and their components from scratch or by retrieval and modification of existing ones. This simple rule editor is further extended by a so called *Pattern2RuleEditor*, which focuses on the reuse of existing rule patterns in the definition of new rules. It looks like the pure rule editor, except that as soon as a new rule pattern is loaded, the editor is dynamically configured with the syntactic and semantic restrictions defined by this rule pattern. For example, if the rule pattern defines that the transaction mode for condition evaluation has to be serial, then the editor may not allow the selection of the parallel mode as soon as the rule designer creates a new rule on the basis of the respective rule pattern. For this, metadata about each rule pattern is stored within the database.

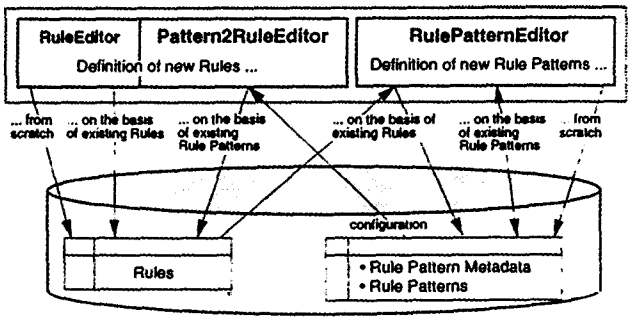

Figure 5. Architecture of $\text{TriGS}_{\text{Designer}}$

The last component of $TriGS_{Designer}$ constitutes the so called *RulePatternEditor*, which is used for the definition of new rule patterns either from scratch, or by the modification of existing rule patterns, or even by factoring out the common components of existing rules. Currently, we are extending $TriGS_{Designer}$ with debugging and animation facilities.

4 Summary and Ongoing Work

In this paper, we have summarized the research efforts behind TriGS, a Triggersystem for GemStone, focusing on

(1) the *seamless integration* of active concepts into an existing object-oriented database system, a powerful execution model by using an *event-based approach*, a prototype implementation including *composite event detection spanning several applications*, and *rule execution in parallel* to both the application and to other rules, and

(2) the introduction of *rule patterns* as a means for designing active object-oriented databases by reusing existing design experience together with the implementation of a corresponding *design environment*.

Although a first prototype system of TriGS is already operational, there are many conceptual questions which still have to be solved concerning both the active system itself and the design of active databases based on rule patterns. Ongoing work concentrates especially on the following issues:

(1) *Inheritance of Rules*. Up to now, not much work has been done in the literature concerning rule inheritance and overriding. Some systems, such as [Beer91], [Coll94], [Geha96], [Mede91], [Shyy94], propose that rules are always inherited and can never be overridden, which corresponds to *strict inheritance* [Wegn88]. Another possibility, which is followed by some systems like TriGS, is that rules are specialized arbitrarily, which is called *arbitrary* or *implementation inheritance* [Wegn88]. The motivation for using implementation inheritance in TriGS was, that the underlying object-oriented model of GemStone, which is Smalltalk, adheres to this kind of inheritance, and, in our opinion, the kind of inheritance used for rules should not contradict the kind of inheritance supported by the underlying object-oriented model. However, what should be investigated in the future is, how far other kinds of inheritance such as *specification inheritance* and *specialization inheritance* [Wegn88] are useful for being applied to rules.

(2) *Advanced Transaction Support*. Active object-oriented databases need advanced transaction support, of which the nested transaction model has proven very promising [Gatz95]. It has also been shown, however, that the nested transaction model is appropriate for simple events only, but too restrictive in case of composite events. This is due to the fact that the nested transaction model does not provide any concept to relate several event signalling transactions, i.e., parent transactions, to a single rule transaction, i.e., child transaction. What is actually required is that the *rule transaction can be made a subtransaction of more than one event signalling transaction*, and, that the rule transaction is able to *cooperate with them* when accessing shared data [Kapp96b]. What we want to do is to extend the nested transaction model of [Härd93] in this direction since it seems to be that these extensions allow combining the reliability and flexibility of the nested transaction model with the expressive power of composite events.

(3) *Low-level versus High-level Rule Patterns*. So far, we have stocked our pattern library with interaction rule patterns stemming from workflow applications and concurrent object-oriented applications [Kapp95], [Rets97]. However, interaction rule patterns may be classified as rather low-level, system-oriented patterns since they provide very basic interaction mechanisms. The identification of higher-level, more application-oriented patterns in the area of workflow management, such as patterns for agent selection and activity ordering, is part of ongoing work.

REFERENCES

[ACT96] ACT-NET Consortium, *The Active Database Management System Manifesto: A Rulebase of ADBMS Features*, in SIGMOD Records, Vol. 25, No.3, Sept. 1996

[Beer91] C. Beeri, T. Milo, *A Model for Active Object-Oriented Database*, in Proceedings of the 17th International Conference on Very Large Data Bases (VLDB '91), G.M. Lohman, A. Sernadas, R. Camps (eds.), Morgan Kaufmann, Barcelona, Spain, Sept. 1991

[Bich94] P. Bichler, M. Schrefl, *Active Object-Oriented Database Design Using Active Object/Behavior Diagrams*, in Proceedings of the 4th International Workshop on Research Issues in Data Engineering (RIDE '94), Active Database Systems, J. Widom, S. Chakravarthy (eds.), IEEE-CS, Houston, Texas, Feb. 1994

[Ceri96] S. Ceri, J. Widom, *Applications of Active Databases*, in Active Database Systems - Triggers and Rules For Advanced Database Processing, J. Widom, S. Ceri (eds.), Morgan Kaufmann, San Francisco, California, 1996

[Chak94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, S.-K. Kim, *Composite Events for Active Databases: Semantics, Contexts, and Detection*, in Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94), J.B. Bocca, M. Jarke, C. Zaniolo (eds.), Morgan Kaufmann, Santiago, Chile, Sept. 1994

[Coll94] C. Collet, T. Coupaye, T. Svenson, *NAOS - Efficient and modular reactive capabilities in an Object-Oriented Database System*, in Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94), J.B. Bocca, M. Jarke, C. Zaniolo (eds.), Morgan Kaufmann, Santiago, Chile, Sept. 1994

[Daya88] U. Dayal et al., *The HiPAC Project: Combining Active Databases and Timing Constraints*, in Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, H. Boral, Per-Åke Larson (eds.), SIGMOD Record, 17(3), Chicago, Illinois, Sept. 1988

[Gamm94] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994

[Gatz95] S. Gatziu, *Events in an Active Object-Oriented Database System*, PhD, Kovac, 1995

[Geha96] N.H. Gehani, H.V. Jagadish, *Active Database Facilities in ODE*, in Active Database Systems - Triggers and Rules for Advanced Database Processing, J. Widom, S. Ceri (eds.), Morgan Kaufmann, San Francisco, California, 1996

[Grot94] T. Grotehen, K.R. Dittrich, *Erweiterung konzeptueller Objektmodelle für den Entwurf aktiver Systeme*, Technical Report, Inst. of Computer Science, University of Zurich, 1994 (in german)

[Härd93] T. Härder, K. Rothermel, *Concurrency Control Issues in Nested Transactions*, VLDB Journal, 2(1), G. Schlageter (ed.), 1993

[Huem93] Ch. Huemer, G. Kappel, S. Rausch-Schott, W. Retschitzegger, A M. Tjoa, S. Vieweg, R. Wagner, *OODB Technology for KBL*, Final Deliverable (Milestone 4) for Esprit Project 5161 KBL, Linz/Vienna, 1993

[Kapp94] G. Kappel, S. Rausch-Schott, W. Retschitzegger, S. Vieweg, *TriGS - Making a Passive Object-Oriented Database System Active*, Journal of Object-Oriented Programming (JOOP), July/Aug. 1994

[Kapp95] G. Kappel, P. Lang, S. Rausch-Schott, W. Retschitzegger, *Workflow Management based on Objects, Rules and Roles*, IEEE Data Engineering Bulletin, Special Issue on Workflow Systems, 18(1), March 1995

[Kapp96a] G. Kappel, M. Schrefl, Modelling Object Behavior: To Use Methods Or Rules Or Both? Proc. of the International Conference on Database and Expert Systems Applications (DEXA'96), M. Thoma, and R.R. Wagner (eds.), Zürich, Springer LNCS 1134, September 1996

[Kapp96b] G. Kappel, S. Rausch-Schott, W. Retschitzegger, M. Sakkinen, *A Transaction Model For Handling Composite Events*, in Proceedings of the Third Int. Workshop of the Moscow ACM SIGMOD Chapter on Advances in Databases and Information Systems (ADBIS '96), Moscow, Sept. 1996

[Kapp98] G. Kappel, S. Rausch-Schott, W. Retschitzegger, *A Tour on the TriGS Active Database System - Architecture and Implementation*, Accepted for Publication at: ACM Symposium on Applied Computing (SAC'98), (to appear)

[Knol94] G. Knolmayer, H. Herbst, M. Schlesinger, *Enforcing Business Rules by the Application of Trigger Concepts*, in Proceedings Priority Programme Informatics Research, Information Conference Module 1, Swiss National Science Foundation, Bern, 1994

[Lieu96] D.F. Lieuwen, N. Gehani, R. Arlein, *The Ode Active Database: Trigger Semantics and Implementation*, in Proceedings of the 12th International Conference on Data Engineering (ICDE '96), IEEE Computer Society Press, New Orleans, Louisiana, March 1996

[Mede91] C. B. Medeiros, P. Pfeffer, *Object Integrity Using Rules*, in Proceedings of the 5th European Conference on Object-Oriented Programming (ECOOP '91), P. America (ed.), Springer LNCS 512, Geneva, Switzerland, 1991

[Nava92] S.B. Navathe, A. Tanaka, S. Chakravarthy, *Active Database Modelling and Design Tools: Issues, Approach and Architecture*, IEEE Data Engineering Bulletin, Special Issue on Active Databases, 15(4), Dec. 1992

[Petr94] I. Petrounias, P. Loucopoulos, *A Rule-Based Approach for the Design and Implementation of Information Systems*, in Proceedings of the 4th Int. Conference on Extending Database Technology (EDBT '94), M. Jarke, J. Bubenko, K. Jeffery (eds.), Springer LNCS 779, UK, March 1994

[Rets97] W. Retschitzegger, *A Tour on TriGS - Development of an Active System and Application of Rule Patterns for Active Database Design*, infix-Verlag, Germany, 1997

[Rubi94] K.S. Rubin, P. McClaughry, D. Pellegrini, *Modelling rules using Object Behaviour Analysis and Design*, Object Magazin, June 1994

[Shyy94] Y.-M. Shyy, S.Y.W. Su, *Refinement Preservation for Rule Selection in Active Object-Oriented Database Systems*, in Proceedings of the 4th International Workshop on Research Issues in Data Engineering (RIDE '94), Active Database Systems, J. Widom, S. Chakravarthy (eds.), Texas, Feb. 1994

[Silv96] M.J.V. Silva, C.R. Carlson, *Conceptual Design of Active Object-Oriented Database Applications Using Multi-Level Diagrams*, in Proceedings of the 10th European Conference on Object-Oriented Programming (ECOOP '96), P. Cointe (ed.), Springer LNCS 1098, Linz, Austria, July 1996

[Wegn88] P. Wegner, S. B. Zdonik, *Inheritance as an Incremental Modification Mechanism or What Like Is and Isn't Like*, in Proc. of the 2nd European Conf. on Object-Oriented Programming (ECOOP '88), S. Gjessing, et al. (eds.), Springer LNCS 322, Oslo, Norway, Aug. 1988

[Wido94] J. Widom, S. Chakravarthy (eds.), Proceedings of the 4th International Workshop on Research Issues in Data Engineering (RIDE '94), Active Database Systems, IEEE-CS, Houston, Texas, Feb. 1994