Seminar-Report 9412 on

Active Database System

Alex Buchmann
Technische Hochschule Darmstadt

Sharma Chakravarthy
University of Florida

Klaus Dittrich
Universität Zürich

March 21–25, 1994, Schloß Dagstuhl

List of paritcipants:

Elena Baralis, Politecnico di Torino, Italy
Daniel Barbar´, Matsushita Information Technology Lab., Princeton, USA
Renato Barrera, Intergraph Corporation, Huntsville, USA
Rudolf Bayer, TU München, Germany
Mikael Berndtsson, U of Skövde, Sweden
Mokrane Bouzeghoub, U de Versailles, France
Holger Branding, TH Darmstadt, Germany
Alex Buchmann, TH Darmstadt, Germany
Stefano Ceri, Poliecnico di Milano, Italy
Sharma Chakravarthy, U of Florida, Gainesville, USA
Umeshwar Dayal, HP Labs, Palo Alto, USA
Oscar Diaz, U of the Basque Country, San Sevastian, Spain
Klaus R. Dittrich, U Zürich, Switzerland
Suzanne Embury, U of Aberdeen – King's College, Great Britain
Opher Etzion, Technion – Israel Institute of Technology, Haifa, Israel
Johann Christoph Freytag, Humbolt U zu Berlin, Germany
Stella Gatziu, U Zürich, Switzerland
Michael Gertz, U Hannover, Germany
Peter Gray, U of Aberdeen – King's College, Great Britain
Ulrike Griefahn, U Bonn, Germany
Thomas Grotehen, U Zürich, Switzerland
Ulrike Jaeger, HU Berlin, Germany
Heinrich Jasper, U Oldenburg, Germany
Dirk Jonscher, U Zürich, Switzerland
Martin Kersten, CWI, Amsterdam, The Netherlands
Angelika Kotz-Dittrich, Schweizerische Bankgesellschaft Zürich, Switzerland
Thomas Kudraß, TH Darmstadt, Germany
Brian Lings, U of Exeter, Great Britain
Rainer Manthey, U Bonn Germany
Hans Nissen, RWTH Aachen, Germany
M. Tamer Özsu, U of Alberta, Canada
Norman Paton, Heriot–Watt U, Edinburgh,Great Britain
Tore Risch, Linköping U, Sweden
Claudia Roncancio, BULL–IMAG / Grenoble U, Gieres, France
Gunter Saake, TU Braunschweig, Germany
Scarlet Schwiderski, Cambridge U, Great Britain
Timos Sellis, Nat. Techn. U of Athens, Greece
Eric Simon, INRIA, Le Chesnay Cedex, France
Martin Sköld, Linköping U, Sweden
Salvatore Stolfo, Columbia U, New York, USA
Jean–Raymond Velten, Centre d'Etudes de la Navigation Aerienne, Toulouse, France
Jennifer Widom, Stanford U, USA
Jüregen Zimmermann, TH Darmstadt, Germany
Günter von Bültzingsloewen, FZI Karlsruhe, Germany

Contents

Active mediator systems in a heterogenous environment
Brian Lings
ER1C: A study in Events and Rules as 1st Class
Rainer Manthey
Active DBs and reactive programming: How much of active DB research is really DB-specific?
Hans Nissen
Active Rules in a Deductive and Object-Oriented Setting
M. Tamer Özsu
Active Capabilities in the TIGUKAT objectbase Management System
Norman Paton
On Applications and Active Database Research
Tore Risch
Compiling Active Object-Relational Rule Conditions into Partially Differentiated Relations
Claudia Roncancio
An active object oriented DBPL
Gunter Saake
Abstract Description of Object Interaction and Object Activity
Scarlet Schwiderski
Composite Events in Distributed Systems
Timos Sellis
Concurrency Control Issues in Active Database Systems
Eric Simon
Optimizing Incremental Evaluation of Rules in an Active DBMS
Martin Sköld
Compiling Active Object-Relational Rule Conditions into Partially Differentiated Relations
Salvatore Stolfo
Scaling Database Rule Processing
Jean–Raymond Velten
Active DBMS for Air Traffic Control Systems?
Günter von Bültzingsloewen
Active DBMS Support for Workflows and Derivation Processes
Jennifer Widom
Research in Active Database Systems
Jürgen Zimmermann
Designing an Active Database System

"On active databases: issues and evolution"

Stefano Ceri, Politecnico di Milano

Active databases are vehicles for expressing data-driven reactive computations which are much needed by real-life applications; yet, they are still not adequately supported by commercial systems, while research prototypes are not sufficiently uniform and/or consolidated. The above considerations, more or less agreed in the opening session of the workshop, "trigger" my current perception of hot issues in the field and of how it could evolve:

1. I believe we need to provide a better abstraction mechanism for reactive computation, and to integrate it into the application development life cycle. This requires understanding critical design issues and steps, providing methodological guidance, helping critical design steps by means of tools. This is useful, in my opinion, even if the target implementation does not use active dbms technology.
2. We need to facilitate the process of rising the functionalities being supported by the active components of commercial systems, which are currently very limited, by having them incorporate some of the innovations currently being experienced in research prototypes. As researchers in the field, it is our responsibility to identify such critical functionalities, provide them with a clean and uniform semantics, and experience them through concrete implementation and use.

"Diversity of rule management: The issue and an approach for object-oriented systems"

Oscar Diaz, University of the Basque Country

Active DBMSs are a fast-growing area of research mainly due to the large number of applications which can benefit from this active dimension. These applications are far from being homogeneous, requiring different kinds of funtionalities. However, most of the active DBMSs provide a fixed, hard-wired execution model to support the active dimension. So my contentions are: 1) different applications require distinct execution models for rule management, 2) it is the user who, besides defining the rules, should specify how these rules have to be managed, and 3) control information rarely refers to individual rules. Rather it is shared by a set of rules supporting the same functionality. Hence we aim to provide an execution model mechanism which exhibits heterogeneity, extensibility and declarativeness.

In an object-oriented context, the above features can partially be achieved through metaclasses. Metaclasses allow the advantages of the object-oriented paradigm to be applied to the DBMS itself. The execution model is then defined using classes and methods. Thus, not only are rules (i.e. the knowledge model) described using an OO approach but so is the rule management strategy (i.e. the execution model). Heterogeneity and extensibility can be seen as the main advantages obtained from this approach. To enhance declarativeness the execution model is described through a set of parameters. In this way, the user just provides the pararmeters for each dimension that best fit the application and the system generates the appropriate methods to support this execution model. These are the problems we face in our active system EXACT.

"Active DBs and reactive programming: How much of active DB research is really DB-specific?"

Rainer Manthey, University of Bonn, Germany

When looking at topics and issues currently under investigation in the active DB community, it appears to me that many (if not most) of them are not that strongly depending on "typical" DB characteristics, such as the presence of a large amount of persistent data, managed in a centralized manner and shared by many users. Features like event specification and detection, coupling modes or execution models, not to mention semantics, are more or less orthogonal to the above-mentioned characteristics. They are much more related to the problem of organizing a general-purpose programming paradigm, which I would like to call "reactive programming". It is characterized by the existence of two computational processes linked by means of a set of event-reaction specifications. A "foreground" process, specified in terms of a "normal" imperative program is automatically monitored by an extended run-time system for the occurrence of situations corresponding to a relevant event. Reactions are automatically activated once such an event has been observed, resulting in a "background" process that can be viewed as a marge of the foreground process and the reactions triggered by its individual steps. Thus reactive programming adds a layer of implicit activation of imperative procedures (controlled by an "intelligent" system) to the explicit activation ("call") of procedures under the control of a programmer. I believe that understanding interaction with an active DB in terms of such a very general model of computation will help us avoiding to develop methods and techniques that are too closely depending on a (narrow) DB-focussed point of view. This does not mean that active DB research should move to programming language research, though. Once the general-purpose aspect of reactive processing has been understood, it will be much easier to properly identify and to successfully highlight those problems and solutions, that are really specific to and unique within the DB context we are interested in.

"Scaling Database Rule Processing"

Salvatore J. Stolfo, Columbia University

As large organizations inevitably embrace distributed database technologies, and the promised age of data superhighways becomes a reality, databases will grow to enormous sizes (and breadth of topics) and will be available anywhere and anytime to a very large user community. Judging from the interest in active / expert / deductive databases, it is natural to pose the question of whether or not the current approaches to database rule processing scale to databases of sizes that are orders of magnitude larger than common today. We call this the scaling problem.

We posit that solving the scaling problem requires new approaches to parallel and distributed processing of database rule programs. PARADISER (PARallel And DIStirbuted Environment for Rules) is an operational programming environment that compiles and implements a rule program over a database substrate in a distributed computing system. PARULEL (PArallel RULE Language), the target rule language, is a set-oriented rule language with parallel execution semantics. An initial implementation of PARADISER, including a PARULEL compiler, has been demonstrated and its performance evaluated over a test suite of application programs, PARADISER introduces a new form of optimization called Predictive Dynamic Load Balancing (PDLB) and a new parallel join algorithm that balances the workload of rule evaluation over a number of processing sites, thereby increasing utilization and speed up of parallel resources.

"Research in Active Database Systems"

Jennifer Widom, Stanford University

Earlier work in active database systems focused on:

- Design and implementation of a new active rule language in the context of the Starburst extensible relational DBMS

- Techniques for specifying so-called "internal" active database applications in high-level languages and compiling them to rules (integrity constraints, materialized views, deductive rules, etc.)

- Techniques for statically analyzing the behavior of active rules.

- Mechanisms for active rule processing in tightly-coupled distributed database systems

Current work in active database systems is in the direction of:

- Exploiting active rules for constraint management in loosely-coupled heterogeneous database systems

- Better techniques for statically analyzing the behavior of active rules and techniques for statically and/or dynamically optimizing active rule execution

- Semantic foundations for active rules

"Active DBMS for Air Traffic Control Systems?"

Jean-R Velten, CENA Toulouse, France

The main responsability of the Air Traffic Controller is to prevent collisions between the aircrafts he is in charge of while ensuring an orderly, but as efficient as possible, traffic flow. To provide such a service, he performs:

* control tasks, issuing clearances and instructions to pilots;

* monitoring tasks, surveying some critical traffic situations;

* information tasks, delivering informations about other traffic to pilots.

Within the DAARWIN project, CENA is studying a distributed architecture and new functions for the ATC system. In order to master the complexity of such a system and to ensure its easy evolution while hiding architecture choices, the Client-Server model has been choosen for the software architecture of this system, providing the basis for system decomposition and incremental design. Due to the very dynamic features of Air Traffic Control field, where most of the application programs need in fact to receive data in due time or to be notified of pertinent events, the active server approach has been choosen. So servers have to filter events and notify clients, in accordance with event subscriptions. The condition mechanism is used to refine the set of related data.

The improvement of these systems will be based on the automation of high level functions while keeping the human operator in the ATC loop. The functions which are currently covered by research programmes are the monitoring functions as well as the cooperative tools, the objective of which is to assist controllers in their decision making process. All these advanced functions put a stronger emphasis on the active capabilities to be provided by the system kernel (mainly primitive events, multi-user notification of event, object status changes as events, appropriate transaction model ...)

”Active DBMS Support for Workflows and Derivation Processes”

Günther von Bültzingsloewen, FZI Karlsruhe

A typical scenario for many applications consists of a net of tools which are integrated into an environment and which cooperate to serve the application. Examples are CAD design environments, manufacturing environments or scientific information systems. The whole environment is supported by a component (potentially based on an active DBMS) managing meta-data (like design object hierarchies, versioning, etc.) and controlling the execution of activities (i.e. eventually the invocation of tools). This control component has to capture and ensure dynamic constraints on activity execution, i.e. which activities are enforced / allowed / forbidden to occur. Further requirements are:

- Visualization of the flow of / constraints on activities
- Abstraction by resting activities
- Flexible online changes of flows/constraints
- Automatic rederivation of a design or data product upon changes
- Querying the derivation history and
- Failure Handling.

While transactional workflow management systems based on nested (trans-)actions and triggers support dynamic constraints, abstraction, and failure handling, they do not enable visualization of system status, etc. Therefore, we propose to combine them with high-level petri-nets (e.g. predicate / transition nets): the flow of (sub-)activities within each activity is captured by the net as well as the constraints on activity execution. The individual nets are related to each other in the way that a transition corresponds to the invocation of a subactivity which in turn corresponds to invoking the subactivity net by placing call parameters onto input places. The History can be captured by means of a trace net which records transition and token instances; this history is scanned for rederivation of data products and supports querying. Our next step will be to work out complex modelling examples in order to check which kinds of extensions to the basic petri net are required and to validate that the approach is appropriate from a user perspective.

”A Constraint-Based Approach to the Repair of Long-Duration Design Transactions.”

Suzanne M. Embury & Peter M. D. Gray, University of Aberdeen, Scottland

The transaction repair process involves the active use of semantic domain information (usually in the form of integrity constraints) to generate a sequence of updates that will restore validity to an invalid database state. In the context of traditional, high-volume transaction-processing applications, the aim of the repair process is to recover from errors automatically so that transaction processing can continue with the minimun of intervention from the user. In the context of design applications, on the other hand, and in particular in applications where transactions are ”user-controlled”, the aim is to assist the designer/user in building a consistent design by suggesting a set of possible updates for she/he to select from. In this way, the transaction repair mechanism becomes another tool in the design process, which can lead but does not constrain the designer.

Our approach is to view the transaction repair process as a constrained search problem in which the result of the search is a (set of) sequences of updates that will restore the consistency of an invalid database state. Based on this view, we are currently implementing an extension to our transaction mechanism which responds to constraint violation (detected at commit-time) by generating a piece of code describing the repair problem, and passing this to a special-purpose constraint solver (CHIP) for manipulation. The constraint solver is able to retrieve data selectively from the DBMS, as it is required, or to evaluate database methods or to retrieve extra metadata information in order to guide the search process.

"Active Rules in a Deductive and Object-Oriented Setting"

Hans Nissen, RWTH Aachen, Germany

Event-Condion-Action (ECA) Rules are a well-known mechanism to define the active (reactive) behavior of a system. Active databases are able to handle and evaluate such rules in an efficient way. For our purposes we look at a system as a provider of several kinds of services. Such a system can be an application tool like a text editor, or a database system. We use active rules to describe the behavior of a system, i.e. a rule specifying how the system should react if a specified event was detected. We propose a service oriented model of such rules, where the action is the execution of some services while an event can be seen as an already executed service. In this context we propose to view the evaluation/interpretation of an active rule again as a service offered by the knowledge base. We integrated this model into the knowledge-representation language Telos. Currently we have two applications in mind. In the first place we want to use active rules to specify the internal behavior of our KBMS itself, e.g. how it should compile and evaluate integrity constraints. Our long-term goal is to use active rules to implement extended functionality of our system, including such things as view maintenance, view update, constraint repair and partial evaluation of meta formulas.

In the second place we want to describe the way-of-working of application programs, such as tools of our usage environment. They then become servers, where the KB conrols the execution of the services. To overcome the problem that they become a bottleneck, we want to compile the behavior-specification into code (e.g. C++), which can be linked to the application tool and then controls it's behavior. The change of the way-of-working can easily be done by changing the rules and recompiling the control code. No modification on the application code is needed.

"Dynamic Workflows in Distributed, Autonomous Systems"

Daniel Barbar^´, M.I.T.L., Princeton N.S., USA

Workflows are long-duration, multi-step activities that involve a coordinated execution of steps at muliple stations inside an organization. We are interested in worklfows that can run in environments with the following requirements

- Autonomous stations : each processing station is autonomous in executing a task on behalf of the workfow, that is, a processing station must be treated as a "black-box" whose mode of operation cannot be changed or fully known to the workflow designer
- Dynamic behavior: Processing stations evolve in time resulting to changes in the control flow. New rules can be added, changing the course of the workflow. This means that no workflow can be fully specified a priori.
- Partial automation: the environment in which the workflow executes may only be partially automated.

We are currently designing a model and system to manage workflow in such environments. The model is heavily based on rules that drive the flow of the activities.

The environment poses very interesting problems to the design. For instance, event detection becomes complex because the events can be composite over primitive events on processing stations.

"Concurrency Control Issues in Active Database Systems"

Timos Sellis, Nat. Technical University of Athens, Athens, Greece

Relational systems are extended with active capabilities. Languages have been developed and execution models have been investigated. Our presentation describes our research on the concurrent execution of rules in a database environment.

Traditionally, the serializability criterion of correctness is defined on the basis of read/write conflicts. With rules however the conditions must be true for the actions to execute, and rules must fail when their actions are no longer true. A different correctness criterion is therefore needed based on the conflicts among conditions/actions. We have developed a locking protocol and the necessary extensions to the transaction managers for the above reasons. One extension is a new lock compatibility matrix which provides greater concurrent access. The second extension is to allow concurrent execution within a transaction. We also present a simulation-based performance study where we identify characteristic features of the rules and study their effect on performance.

"On Applications And Active Database Research"

Norman Paton, Heriot-Watt University, Edinburgh

As illustrated by the wide range of topics addressed at this workshop, the extension of database systems with active capabilities is a fertile area for theoretical and practical research. It is also clear, however, that much of this research is proceeding on the basis of "technology push" rather than "application pull". In the early phase of research on a topic, this may well be healthy, as many ideas can be generated and paths explored in order to identify possibilities. However, many significant issues which are presently topics of debate can only be effectively addressed in the context of comprehensive example applications. For example, the following questions have been considered at this workshop:

- Which parts of an application are most effectively expressed as active rules, and which using some other paradigm?
- What tools or methodologies are most effective for designing applications involving active behaviour?
- What forms of rule analysis are necessary/helpful for developers of active systems?
- Which of the wide range of execution model dimensions proposal are useful in different contexts?
- Are different kinds of database system most effectively served by different flavours of active rule system?

It is difficult to see how such questions, which have a significant bearing on much of the ongoing research into active databases, can be effectively addressed without meaningful experience with significant applications. A preliminary framework for the comparison of rule systems and applications is presented in [1]. This, however, is not sufficiently detailed or associated with a wide enough range of applications to enable most of the above questions to be answered. It is thus proposed that a portfolio of detailed descriptions of implemented applications would be a significant contribution to the active databases literature. It is planned, therefore, to put together such a portfolio under the anspices of the ACT-NET Network of the European Community. The plan is to proceed as follows:

1. ACT-NET members will draw up a framework within which active applications can be described.
2. A "call for description" will be made.
3. The resulting portfolio will be made available through the ACT-NET ftp server.

It is intended that the portfolio will extend as new contributions are made. Please support this venture if you can!

[1] N. Paton, O. Diaz, H. Williams, J. Campin, A. Dinn and A. Jaime, "Dimensions of Active Behaviour", Proc. 1st Int. Wshp. on Rules in Database Systems, Springer-Verlag, 40-ST, 1994.

"Event Signaling in an OO system"

Renato Barrera, Intergraph Corporation

We are interested in providing active capabilities to an OO system that manages caches of C++ objects and uses a passive DBMS for persistency. Our system considers flat, short transactions only and is tailored for an environment in which events can be defined on the fly, and in which events can be associated to or dissociated from rules dynamically. Our system lumps conditions and actions into a single procedure, considers both primitive and composite events, and can execute rules in an immediate or in a deferred mode. In our treatment of composite events, we have traded expressive power for speed of execution, thus achieving a design in which both events and their ancilliary constructors are created "just in time" and cease to exist when no longer needed.

"Active Database Functionality - Experiences and Requirements in a real-world Environment"

Angelika Kotz-Dittrich, Union Bank of Switzerland

In contrast to the increasingly sophisticated rule models the active database community is now coming up with, the state of the art in the industrial environment is fairly modest. We currently find a huge amount of rule checking embedded in application programs or hidden "in the heads of people". In our environment, which is a large bank, we made the observation that as soon as a DBMS with some active functionality is available to them, developers and users are very keen on applying these new features (we are talking here about the relational products with basic trigger mechanisms). The main problems and requirements we encountered in the corresponding projects are as follows:

There is an urgent need for design methodologies and practical guidelines. Users demand tools that support maintenance, reuse and extension of rule sets (we find sets of more than 100 rules that are very hard to maintain). Performance is a big issue and there are applications exhibiting real-time requirements like e.g. in the area of security trading. Security problems (i.e. all aspects regarding validating and controlling rules) are a main hindrance to apply active

DB functionality in mission-critical applications. Furthermore, we are living in a distributed heterogeneous environment with huge legacy systems and active functionality will have to cope with that fact.

We are currently advocating the use of active DB functionality in non-mission-critical applications with moderate data volumes and workloads. Appropriate fields we are building databases for are financial analysis and economic research as well as infrastructural components for database interoperability or workflow control.

"Optimization Issues in ADBs - The first steps"

Johann Christoph Freytag, Humboldt Universität zu Berlin

Based on already existing languages and concepts for ADBs we focus on the following three issues in our group.

1. Fundamental Issues in ADBs.
   Since ideas for ADBs use concepts coming from databases, the transaction area and "cooperate systems" area, it is important for us to understand the concepts in each area separately before "mixing" them together. The goal of this work is to develop a uniform framework that allows us to describe all aspects for ADBs coherently.

2. Optimization issues:
   Based on 1., we shall investigate the different possibilities in ADBs. Optimization should cover both inter- and intra-rule optimization. We hope to take advantage of already existing formalisms and models (such as Petri Nets) for dealing with this issue.

3. Applying ADB technology to applications
   Based on our interest in CIM we would like to model (part of) an application using the ADB paradigm. This task will help us to learn about the suitability of the ADB formalism, at the same time verifying the approach taken in 1. and 2..

A long way to go . . . . .

"Active Capabilities in the TIGUKAT objectbase Management System"

M. Tamer Özsu, University of Alberta

Many of the applications that require the functionality of an objectbase management system seem to require active capabilities as well. We are therefore, looking at the development of a distributed active objectbase management system. Our work is conducted within the context of the TIGUKAT system. TIGUKAT (which means "objects" in the Canadian Inuit (Eskimo) language) is a distributed objectbase management system under development in the Laboratory for Database Systems Research of the University of Alberta. It has a novel object model whose identifying characteristics include a purely behavioral semantics and a uniform approach to objects. Everything in the system (including the schema) is a first-class object with well-defined semantics; thus the system is reflective. The computational model supported is one of applying behaviors to objects.

We are introducing rules as objects. Consistent with TIGUKAT philosophy, rule components (events, conditions and actions) are also modeled as objects. At the moment we have considered simple events based on behavior application. The conditions and actions are modeled as functions which are first-class objects. At the moment we consider conditions expressed as queries in the TIGUKAT query language (TQL). Query type is a subtype of function type in TIGUKAT. This work is in its initial stages and many of the system problems have yet to be solved.

”The AIMS Approach to Active Information Management Systems”

Ulrike Jaeger, Humboldt Universität zu Berlin, Germany

AIMS approaches the problems of cooperative applications from different angles. An example for a cooperative application is a hospital information system. A set of self-centered units perform local tasks and cooperate for the sake of the global task to heal the patient. A unit is aware of its own task and context and the information it requires from other tasks in order to proceed. In exchange it will externalize part of its local information to other units.

The AIMS model explores three dimensions of the problem:

- a cooperation model,
- an activity model,
- a db model.

The cooperation model follows a proposition of J. Klein (COMPCON 1991) and R. Obermarck who presented an event synchronization model for the specification and implementation of (advanced) transaction models. The model provides a slim bidirectional event interface to the units. The execution is event driven and handles complex events. It has temporal notions of *eventually,* as well as *delay* and *deadline.*

The activity model in general follows the E-C-A semantics of rules. Here notions of event, complex event, situations come up again.

The db model is characterized by tx as the execution model.

AIMS investigates how far those concepts are identical, related or contradictory. We'd like to know which power we gain it we decide in favor of one or the other contradictory semantics?

”Abstract Description of Object Interaction and Object Activity”

Gunter Saake, TU Braunschweig

Conceptual modelling of information systems requires besides the design of the database structure also the modelling of dynamic application behaviour. Part of this behaviour description may later on be implemented using active database technology. Such a modelling approach has to abstract from implementation details. It should offer a logic-based, declarative framework for describing all relevant aspects of system behaviour to enable consistency checks and application verification.

As part of the conceptual modelling language TROLL, event calling is proposed to capture the interaction aspect of dynamic system behaviour. Event calling bases on a set-based execution model (a set of events occurs simultaneously) which is semantically based on logical implications between event occurences. A set-at-a-time execution model avoids problems like confluence or non-deterministic semantics, leaves freedom for optimization, and the set property ”no duplicates” makes it possible to check consistency and termination conditions for calling chains. Based on parameter flow, a partial order of called events may be determined defining correct execution orders.

The presented work is joint work with Thorsten Hartmann. For receiving a detailed report on it, please contact the author.

”Phoenix: a database programming language with active rules”

Ulrike Griefahn, Universität Bonn, Germany

Though most active database languages are based on the ECA-paradigm, there is still no agreed understanding of the role of the individual rule components and their relationships. When developing an active rule concept for our DBPL Phoenix, we therefore aimed at providing syntactical features for expressing various such roles and relationships within a common framework. Active rules in Phoenix consist of two parts, a trigger and a reaction. The trigger is composed of an event pattern and an event condition. The reaction consists of an arbitrary imperative statement of the DBPL and of an execution mode that specifies the time at which the reaction is to be executed. Conditions appear in two places: The event condition is only used to specify the triggering event in more detail. Conditions that generate bindings for the rule‘s action have to be specified within an imperative statement in the reaction part, e.g. for each C do A. This is possible, because control structures in Phoenix may include arbitrary queries in their condition part.

Events that are observed in Phoenix are instance-oriented. An event may be an arbitrary procedure call, the exit of a procedure execution, or some other event (e.g. a clock event). Each event uniquely defines a specific point in time. In this sense the trigger part of a rule specifies a point in time at which the respective rule is to be activated. In a similar way the execution mode denotes the point in time at which the reaction execution is to be started. This may either be immediately after event detection, or instead or after the execution of the invoked procedure. In addition, the execution mode may specify a later event (e.g. 11:00am or the end of the current transaction), immediately after which the reaction is to be executed.

”The Pardes Project”

Opher Etzion, Technion, Israel

The Pardes Project was originally aimed at supporting spreadsheet-type programming and constraint enforcement in an active fashion. These types of applications have inherent semantic properties that enable their specification in higher-level language, and execution optimization techniques that are not applicable in the general case of active databases such as: avoiding redundant updates, detecting cycles and incremental recomputations. Current activities in the Pardes project tackle the following issues

1. The ”TAPUZ” extension add the capabilities of general active database rules and supporting composite events.

2. A temporal component of Pardes has been designed to integrate the capabilities of active and temporal databases and to allow multiple views of the past, present and future, and retroactive (+ proactive) updates and queries.

3. A mutual consistency theory of derived data-elements that combines the active database coupling modes with materialization modes.

4. An extension to the disributed case, and the use of Pardes as a coordinator for interdependancies in a federated database.

5. A high level language and model for active exception-handling in databases.

”Composite Events in Distributed Systems”

Scarlet Schwiderski, University of Cambridge

In many application areas it will be unavoidable to use the active database paradigm in a distributed environment (e.g. banking systems). My particular interests include cooperative working and multimedia applications (e.g. multimedia conferencing, hypermedia).

Distributing ECA rules to different sites of a distributed system brings considerable difficulties in comparison to "centralized" ECA rules. At present I am looking at composite events in a distributed system. I assume that the constituent basic events of a composite event occur at different sites in the system. A certain site is responsible for monitoring a certain rule and therefore for detecting the corresponding composite event. This site is called the observer site. On the one hand the observer site detects relevant events locally and processes them and/or sends them to remote "interested" sites; on the other hand it receives relevant events from remote sites. Events are first put into an event queue at the local rule monitor and then processed. One problem is that the arrival order of events at the local rule monitor does not coincide with their occurrence order (in general). Another issue to consider is that events on different sites can occur "in parallel" (no global time in distributed systems; approximately synchronized clocks) and therefore cannot be totally ordered. A local event monitor must therefore consist of two components:

- a stabilizer, which sorts incoming events (topological sort)
- an event detector, which detects composite events from the sorted stream.

W.r.t. these problems I found an interesting analogy in the field of distributed debugging. I argue that some of these results apply to our case. On the whole, I want to tackle the following topics:

- naming of basic events
- construction of composite events
- detection of composite events

"ER1C: A study in Events and Rules as 1st Class."

Brian Lings, University of Exeter
Mikael Berndtsson, University of Skövde

Presents the design of an active database system built in a layered architecture on Ontos. Major features include logical events with local condition checks, together with rule conditions which are SQL statements. ER1C is a '123' system according to the seminar definitions. Other important features include event and rule inheritance which is intergrated with the inheritance concepts of C++ methods, and composite event detection.

The implementation design relies heavily on the active data dictionary facilities of Ontos; this provides good support for primitive event detection. However, the transaction level gives poor support.

"Optimizing Incremental Evaluation of Rules in an Active DBMS."

Eric Simon, INRIA, 78153, Le Chesnay, France

”Compiling Active Object-Relational Rule Conditions into Partially Differentiated Relations”

Martin Sköld, Tore Risch
Dept. Comp. and Inf. Science, Linköping University, Sweden

Presents ongoing work on tightly integrating active rules with a next generation object oriented (OO) database system having transactions and a relationally complete OO query language.

The rules are defined as Condition Action (CA) where Events (E) can be specified as an option. The condition part of a rule is defined as a declarative OO query and the action as procedural statements.

For efficient rule condition monitoring, a technique called Partial Differention of relations which is based on incremental evaluation techniques. The rule compiler generates partial Δ-relations that detect changes to a derived relation given changes to one of the relations it is derived from. The technique is based on the assumption that the number of updates in a transaction is usually small and therefore only small effects on rule conditions will occur. Thus, the changes will only affect some of the partially differentiated relations. The partial Δ-relations are optimized using cost based query optimization techniques. Changes are propagated in a network at a check phase usually at commit time (deterred rules). The propagation algorithm uses bottom-up, breadth-first propagation to correctly and efficiently propagate both positive (insertions) changes and negative (deletions) changes. The technique does not assume permanent materializations, but this can be added as an optimization option.

”Reactive Integrity Enforcement in Active Databases”

Michael Gertz, University of Hannover, Germany

It is well accepted that active databases provide a suitable framework to implement efficient integrity maintaining subsystems by the use of triggers. Recent approaches suggest to utilize also triggers to perform repair actions in case of constraint violations instead of rolling back a whole transaction. Since these approaches mainly follow fixed and automated strategies the designer often has to refine or even to revise already derived repairing triggers to incorporate more semantical knowledge on the application. Herewith often the efficiency of the derived triggers decreases.

The main problem on repairing constraint violations is that even simple integrity constraints can be violated through several db operations and also often a multitude of possible repair actions exists.

We argue that modelling the reactive behavior as a design task should be clearly seperated from the automated derivation of respective triggers. For this we provide an expressive declarative specification language for repair actions which includes the specification of rollbacks of single violating operations, attribute replacements on violating tuples, additional modifications (propagations) and exceptional cases for constraint violations. The advantage of reaction specifications is that violations can be analyzed and repair actions then may depend on respective results.

Notwithstanding, it is undisputed that modelling the reactive behavior or constraint violations builds a complex design task which should be supported by design tools tailored to ananlyze constraints, reactions and their dependencies (e.g., a repair action for a constraint can violate another constraint). Although we provide a clear concept on modelling the behavior we focus our further work on more sophisticated analyzing techniques of constraints and corresponding reactions. Finally, for triggers automatically derived from constraint and reaction specifications we want to develop tools to simulate and debug repair processes in order to provide the designer a complete framework to enrich his/her application by efficient repair actions (triggers) on violations of the specified constraints.

"Active Rules in the Context of Access Control"

Dirk Jonscher, Klaus R. Dittrich, Universität Zürich, Switzerland

Concerning authorization active rules are a special case of protection objects, and it has to be specified who is allowed to create, alter, drop and/or enable/disable an active rule. The chosen scheme must be compatible with the security policy of the system. In particular, it has to be taken into account that rules are usually associated with other protection objects (relations or object types) which may have been created by a different user possibly having a control-right for this object. Furthermore, active rules (conditon and action part) are units of activity. Thus, it has to be verified whether the operations executed by a rule are permitted or not. The following approaches are possible: the rule inherits the access rights of its creator/owner (the standard for most existing systems); the rule inherits the access rights of its "invoker" (probably not very useful); or a rule is an authorization unit of its own and can directly get the required access rights. The latter is the most preferable approach, because it allows for a seemless integration of authorization for active rules into existing access control policies. This way it is possible that different users design an active rule and authorize it to be executed as production data.

Besides access control for active rules, there is another interesting aspect, namely access control with active rules. Active rules are powerful mechanisms to realize elaborated security policies. Two examples are Chinese Wall policies (Brewer/Nash-Model) and the realization of duties (or obligations). Both policies depend on the reaction to events. Chinese Wall policies are history-dependent access rights. A user is free to choose which data he/she wants to access. However, once an access has been done, accesses to other data which belong to a conflict of interest domain (e.g. data of a competing company) are forbidden. An active rule can be used to monitor the access behaviour of users and react with a proper authorization (revocation of permissions or granting of prohibitions).

Duties are special access relationships where a user is obliged to execute a transaction under certain circumstances. These transactions require human interaction such that they cannot be automatically scheduled. However, the system can monitor the fulfilment of a duty and can schedule a contingency action in case a duty is not fulfilled.

"SENTINEL: An object-oriented active DBMS"

Sharma Chakravarthy, University of Florida, Gainesville

Making a database System active entails not only developing an expressive event specification language (with well-defined semantics), algorithms for event detection, but also a viable architecture that is extensible. Sentinel adbms is an attempt at this. Sentinel uses Snoop as the event specification language consisting of not, or, and, sequence, any, A, A*, P, and P* event operators. We support various event consumption modes (referred to as contexts in Snoop), such as unrestricted, recent, chronicle, continuous, and cumulative. These event-consumption modes have different storage and computational requirements.

We use the execution semantics of HiPAC slightly tailored to the object-oriented enviroment (e.g. class level rules vs. instance level rules, subscription and notification mechanism). A nested transaction model is supported and is used to implement serializable execution of rules.

Sentinel is built using open OODB (from TI, Dallas) and Exodus (from Univ. of Wisconsion, Madison). Currently, a planning and a financial application has been implemented to demonstrate the functionality supported by Sentinel. They will be expanded to reasonable size applications in the immediate future to perform benchmarking and performance analysis of the implementation.

"Modelling Events and Reactions for External Active Database Applications"

Heinrich Jasper, Universität Oldenburg, Germany

Whereas triggers and ECA-rules have been studied in detail for internal database applications, e.g. integrity checking and auditing, there is a lack of experiences with external, real world applications. Our research at Universität Oldenburg suggests that an explicit notion of EVENT (and reactions thereof) should be used within the huge amount of modelling activities neccessary for real world applications. Although events are already incorporated in several modelling techniques (OMT, Martin/Odell,...) they play a minor role in these. In opposite to that, we want to "lift" the EVENT to a first class entity w.r.t. modelling, just as objects are in the aforementioned techniques. From object and event structures the behaviour (methods for objects, activities for events) is derived and subsequently validated. Gradually refinements lead to an object-event-behaviour model of the application domain that can be transformed into ECA-rules of an underlying active (database) system.

"Active Databases: A Perspective"

Umeshwar Dayal, Hewlett Packard Laboratories, Palo Alto, Califormia, USA

Over the past ten years or so, active databases have matured to the point where there is now an active research community. The first research prototypes have been built. Some rudimentary functionality has made its way into products, and is reflected in emerging standards such as SQL3 and OMG. This is a good time to reflect on where the field should be headed.

In my opinion, there has been altogether too much emphasis on rule models and languages, with a broad spectrum of execution semantics. Instead of inventing yet others with more and more esoteric features, this community should turn its energies towards engineering concerns.

There is a dearth of experience in building real applications using active databases. Much work needs to be done in developing methodologies and tools for design and analysis of active database applications. As we push more semantics and functionality into the database system, where is the boundary between the application program and the database? How are the active capabilities made available through API's and user interfaces? Active database systems are different from large scale production rule systems or other kinds of expert systems (which might profit from active database support). What are the canonical active database applications? More work also needs to be done on architecture, implementation, and performance issues; and in defining benchmarks.

The technology must enter the mainstream of DBMS practice, so that "active database systems" are no longer a curiousity; rather, <u>all</u> DBMSs must exhibit "active" capabilities.

Finally, we must face the challenge that applications in the real world will be constructed and deployed in open, distributed computing environments that include a variety of components and sevices such as database managers, transaction managers, naming services, security services, workflow services, event monitoring and notification services, and so on. Some of the functionality currently bundled in DBMSs will be unbundled and provided as components and services, together with other functions not typically associated with DBMSs. Active DBMSs will have to become citizens of such environments if our technology is to become widely used in practice.

”An active object oriented DBPL”

Claudia Roncancio, Bull & University of Grenoble, France

We are concerned by the problem of writing a complete database application. We consider that it is essential to provide the programmer with a powerful declarative query language as well as with a turing complete language, for updates and complex calculations. We propose Paplom which is an OO DBPL extended with active and passive rules. The language offers a smooth combination of an imperative language and a (declarative) deductive one.

The model proposed includes classes and modules. Classes include as usual, the structure and behaviour of objects as well as the ”active behaviour” of the objects. The active behaviour is expressed as ECA rules which can react to any event concerning the objects (even its private part).

Modules offer a means for structuring applications. A module includes global variables, operations (which are untargetted to a specific class) and active rules. The definition of active rules in classes as well as in modules allows to organize rules according to the role they play. For instance, particular users can customize the ”rule set” used by their application, by defining its rules in the modules used by their applications.

With our proposal we are not pushing the triggers out of the DB scheme but we are offering a means for defining database triggers and application triggers in an uniform way. We consider problemes of supporting active rules in an OO context as the one related to complex structures and inheritance. We also propose a new ”activation mode” for triggers which allows to support a set oriented execution of rules in the object context.

The implementation of (Active) Paplom is in progress and our main goal is to provide good performance.

Note: this work is done together with P. Dechamboux.

”Object Model Extensions for Active Design”

Thomas Grotehen, Klaus R. Dittrich, University of Zürich, Swiss Bank Cooperation

During oo-sw-development application specifications are expressed in terms of an object model. The ”active” part of the specification may be expressed in a rule language like SAMOS. It is not a trivial task to define such a specification, so conceptual object models sometime are used as a tool to build a higher level specification that is mapped to a concrete (logical) object model. Most of these conceptual models do not provide for concepts that can express situations the system has to recognise and react on. We propose an extension for conceptual object models that can be used to express such situations and the connections to actions.

The basic modelling concepts in our approach are event and condition nodes that can be connected to methods via ”event vertices” (conjunction, disjunction, negation, sequence, ...).

We prefer the ”unary” approach : one tool - the object model, one product - the oo-schema. Up to now we have not seen any reason why our concepts should be restricted to database modelling.

”The Architecture of the SAMOS Prototype”

Stella Gatziu, Klaus R. Dittrich, Universität Zürich, Switzerland

We investigate the architecture at the active ooDBMS SAMOS. SAMOS provides a rule definition language as a means to specify ECA-rules. Events may be specified in a primitive way (method, time, transactions and abstract events) or in a composite way using a set of event constructors such as sequence or negation.

The SAMOS prototype consists of two blocks:

- an underlying object-oriented DBS (ObjectStore)
- an add-on layer on top of ObjectStore implementing the active functionality

The add-on layer consists of:

- an analyzer for compiling event and rule definitions
- a rule manager for the retrieval of information about events and rule definitions. In an object-oriented environment rule and event definitions are represented as objects.
- a detector for primitive events. For example, time events are detected using the cron mechanism of UNIX. Or, method events are detected by recompilation of the implementation of the methods of interest.
- a detector for composite events. In SAMOS we use the model of Coloured Petri Nets for the modelling and the detection of composite events.
- a rule execution component for condition evaluation and action execution. Using the transaction model offered by ObjectStore we identify the problem and the restrictions concerning, e.g., the support of decoupled coupling mode.

"Some requirements to make active databases a viable technology"

Alex Buchmann, Technische Hochschule Darmstadt, Germany

Basic active database capabilities are appearing in relational products. At the same time, the first active object-oriented prototypes are emerging. A variety of issues must be resolved to make active database technology useful to a wide community of users and to accelerate the development of new and more robust object-oriented active DBMS prototypes.

The entry price to do meaningful work in active object-oriented DB systems is rather high since a stable oo-platform is required. Therefore, an extensible platform into which individual groups can insert partial developments is required. We require further a reference architecture that can serve both as a way to describe active databases systems in terms of the level of functionality offered, and to identify necessary interfaces to be provided by the DBMS vendors to layer active capabilities on top of existing DBMSs. As prototypes are emerging it becomes increasingly necessary to define yardsticks and benchmarks for testing. Given the difficulty in defining provable correctness criteria for full-fledged active DBMSs we will depend an well-defined test-suites.

In order to make this technology widely acceptable to users we require active DB design methodologies and tools. Unless we support the users early on we run the risk of causing disillusionment with this potentially powerful technology.

"Active DBS and RTDBS: bringing both camps together"

Holger Branding, Technische Hochschule Darmstadt, Germany

Many applications proposed to be gaining from active DBMS functionalities need support, first, in modeling the intended temporal behavior, and second, the enforcement of modeled temporal behavior.

I try to bring together the two camps of real-time DBS and active DBS. Both fields are rather new research issues. They lack even a common understanding of basic notions. Apart from these difficulties there are some basic functionalities that turned out to be essential to build efficient and real working systems that are capable to support real applications.

Combination of the two fields involves, first, to develop a RTDBMS with restricted functinality, i.e. a restricted data model, or reducing unpredictable behavior by restricting to main memory DBS, etc.

In a second step, a restricted DBMS is enhanced with active capabilities. As in RTDBS, the functionality must be restricted to control complexity and predictability. It is essential to derive cost formulas in order to provide a more predictable temporal behavior than it can be found in such systems nowadays. In the active part the event set to which rules react must be restricted.

Both fields need the support of the underlying operating system because consumption of time is due to the use of resources, namely CPU, main memory, and I/O.

"Designing an Active Database System"

Jürgen Zimmermann, Technische Hochschule Darmstadt, Germany

In order to build an active DBMS we first tried to use an underlaying object-oriented database system and made several negative experiences. One problem is the detection of method events. A typical solution is developing a preprocessor which wraps all methods. This is both inefficient and sometimes incorrect. When overridden methods in their body call the original method two wrappers have to be passed causing the creation of two events.

To avoid the problems of a layered architecture we use Texas Instruments' Open OODB and have access to their sources. Now we're able to adapt both the transaction management and the method invocation mechanism to the requirements of an active DBMS.

Another subject of the REACH project is having high parallelism inside the DBMS, namely in composing events and in firing rules. Therefore, we use the new operating system Solaris offering threads which yield better performance results than conventional child processes. Firing rules in parallel is realized in firing one rule in two subtransactions running in one Solaris thread. Since rule firing starts with a subtransaction for condition evaluation having typically read-only methods we are working on optimistic synchronization between sibling subtransactions. A rule's 2nd subtransaction executes the action associated to the rule. The next step to enable/offer different synchronizations we'll extend TI's Open OODB with several transaction managers which provide the functionality required by active DBMS.

"Active mediator systems in a heterogenous environment"

Thomas Kudraß, Technische Hochschule Darmstadt, Germany

A heterogeneous database system can be considered as a well-suited application of active database capabilities. A federation of multiple database systems can be viewed as a space of distributed active objects. The task of a mediator within a federation is to enforce global integrity constraints as well as to control long-running activities spanning database boundaries.

The necessity arises to cope with the problem of the autonomy of the component systems (e.g. design autonomy, execution autonomy). A promising attempt is to define autonomy in terms of the functions they are providing at their interface. It cannot be assumed that all participants of the federation offer all the needed hooks to allow the mediator to control every state of them in order to guarantee global consistency, for example, the occurrence of local updates as events that has to be monitored. A solution must be found how to handle the conflict between consistency conditions that have to be enforced globally and the autonomy the local systems still preserve. So there is a motivation for the notion of weaker consistency requirements that have to be expressed in multi database rules.

The testbed to be implemented uses the active rule system of REACH based on the Open OODB database system with C++ as canonical data mode. The component systems comprise relational as well as object oriented systems (Sybase, ObjectStore).

”An Algebraic Approach to Rule Analysis in Active Database Systems”

Elena Baralis, Politecnico di Torino

While active database systems are very powerful, developing even small applications can be a difficult task, due to the unstructured behaviour and unpredictable nature of rule processing. During rule processing, rules can activate and deactivate each other, and the intermediate and final states of the database can depend on which rules are activated and executed in which order. It is highly beneficial if the rule programmer can predict in advance some aspects of rule behaviour. This can be achieved by providing a facility that statically analyzes a set of rules, before installing the rules in the database. Two important properties of rule behaviour are termination and confluence. A rule set is guaranteed to terminate if, for any database state and set of modifications, rule processing can not continue forever. A rule set is confluent if, for any database state and set of modifications, the final database state after rule processing is independent of the order in which activated rules are executed.

We propose a generally applicable algorithm for determining when the action of one rule can affect the condition of another rule. This algorithm is useful for analyzing termination, since it can determine when one rule may activate another rule, and for analyzing confluence, since it can determine when the execution order of two rules is significant. Since we take an approach based on relatonal algebra, our method is applicable to most active database systems that use the relational model.