Information System Using Active Database- a Novel Approach

Ayushi Jain
CS Dept., Vivekananda
Institute of Technology, East,
Jagatpura, Jaipur, India

Pankaj Chittora
CS Dept., Vivekananda
Institute of Technology,
East,Jagatpura, Jaipur, India

Tushar Vyas
CS Dept., Vivekananda
Institute of Technology, East,
Jagatpura, Jaipur, India

Abstract-Active Database Management Systems (ADBMSs) have been developed to supportapplications with sensing changes in databases. Database driven Information Systems are divided into two integrity concerns: staticand dynamic integrity. Static integrity is one that is based on the situations within a particular databasestate and dynamic integrity addresses situations with state sequences.Dynamic integrity issues come to the bow whenservices are intended over the database. Information systems are collection of services and dynamic integrity is then an indispensable part of IS design. Therefore, designers of information systems have been regularlyconcerned with dynamic integrity.An active DBMS is one whichautomatically executes specified actions when specified conditions rise. Active Databases (ADBs) that enhance an element of dynamics into database systems, have mainly addressed static integrity concerns. Though there have been some attempts to join dynamic integrity concerns inside the agenda of active databases. This paper speeches dynamic integrity concerns in active databases by changing the perception of an active database from a database system enlarged with rules to a database driven information system that offers rule- based services. This change in perspective offers anauspicious approach for addressing active databasesinadequacies, and discloses a roadmap of directions for future active database research.

Keywords— Database, Information System, Termination, Confluence, Observable Determinism, Reactive System, Multi-stream Interaction

I. INTRODUCTION

Active database includes support forspecifying active rules that monitor changes to data and rules that performsome control tasks for the applications. Active rules can also be used for specifyingconstraints that must be met to maintain the integrity of the data, formaintaining long-running transactions, and for authorization control. Traditionally database systems were designed to manage large collections of static data. Data values in any database instance are related through certain dependencies and restricted through certain constraints. The emphasis of database design was to efficiently maintain these relationships among data elements whenever the database needs to update.

The community of information system researchers speak to issues of managing information within a larger dynamic system. The general model of an information system is in the form of a collection of semantic services.

Services were initially defined and modelled as sequences of tasks resulting in various flow based representations like flow diagrams, workflow diagrams, activity flowcharts, etc. However, there is a rising realization that services are best designed using models of interaction, developed largely in the reactive systems domain [16,24]. This change in perspective comes from the observation that semantic services are usually interactive in nature, involving intermediate exchanges with one or more external environments.

Nowadays, most database systems offer features that go beyond management of static data and most information systems are powered by a database. As a result the distinction between the two is blurred and the terms are sometimes used interchangeably. Several issues that have been addressed in one of the fields are reinvented in the context of the other.

The difference between a database and an information system is best appreciated when we consider their function. The job of a database is to store data and answer queries. This entails addressing issues like data models, schema design, handling distributed data, maintaining data consistency, answering queries, etc. Updates to the stored data set happen in atomic operations, where each update is isolated from all other update operations. Query answering is a history less, atomic operation. As long as there are no updates, a given query will result in the same response, regardless of how many other queries are running concurrently and how many times the same query has been asked before. Such behaviour is representative of closed algorithmic computation.

The job of an information system is to provide a service, which are semantic entities involvingthoughts that span the life cycle of the larger system. Except in trivially small information systems, services are interactive in nature involving user sessions with one or more users. An interactive service need not be a closed, atomic operation. It may involve many intermediate states where the external environment may influence the flow of the computation. Services are also not isolated from one another. Two or more interactive services may be intertwined in such a way that their operations cannot be serialized[25].

The following table compares Databases and Information Systems.

TABLE I. DATABASE V/S INFORMATION SYSTEM

	Databases	**Information Systems**
Nature:	Algorithmic	Interactive
State:	User Data	User data, logs and histories, user profiles
Job:	Updates and queries of data	Data backed services to users
Output:	Determined completely by Query/update specification.	Individualized based on user history/preferences

These considerations can perhaps be summarized as Information System is equal toDatabase plus Interaction.Here, the term "interaction" concerns the set of all issues that go into modelling one or more multi-channel interactive processes that characterize services provided by the system.Active databases are a combination of traditional static databases and active rules, meant to be automated mechanisms to maintain data integrity and facilitate in providing database functionalities. Active Database is a combination of Database and Active Rules.

However, soon their rules came to support a variety of tasks like integrity constraints, materialized views, and derived data, coordination of distributed computation, transaction models, and progressive data modelling constructs and automatic screen update in the context of database change. Despite this, integrity management in ADBs have been mainly concerned with static integrity that concern data, with very little support for dynamic integrity that concern services.

We contend that the traditional view of active databases is unduly limiting. To simulate the behaviour of a passive database using active rules not only results in ad hoc rule design, but may also cause footraces in translating functionality requirements of the information system into rules in the Active databases. We argue for a radical change of viewpoint, so active databases are embraced as a special restricted type of an information system rather than a special type of a database. In order to demonstrate the impact of this change in viewpoint, we consider a running example of a conference management system which similar to the one addressed by Essnk and Erhart in explaining stages of development of an information system [1].

II. ACTIVE DATABASES

A. Features of Active Databases

Database systems have been passive, storing and retrieving data in direct response to user requests without initiating any operations on their own. As the scale and complexity of data management increased, interest has grown in bringing active behaviour into databases, allowing them to respond independently to data-related events. Typically this behaviour is described by event-condition-action rules.

Event-condition-action rules comprise three components: event E, condition C, and action A. The event describes an external happening to which the rule may be able to respond. The condition examines the context in which the event has taken place. The action describes the task to be carried out by the rule if the relevant event has taken place and the condition has evaluated to true [2]. In sum, if the specified event E occurs and if the condition C is true then the specified action A is executed.While there is agreement that all ADBs must detect event occurrences, support rule management, and execute actions [3], there is no consensus on how the events, conditions and actions are specified. Rule conditions may get arbitrarily complex and rule conditions may have to be monitored in one of many different ways [4].

B. Rule Analysis

Rule analysis deals with predicting how a set of rules behaves at run-time. The following are the three properties of rule behaviour [2]:

1) *Termination:*Active database rules are terminating only if there is no recursive firing of rules. The ADB itself does not guarantee rule termination.
2) *Confluence:* Confluence property of rules decides whether the execution order of non-prioritized rules make any difference in the final database state. ADBs do not guarantee any rule execution order.
3) *Observable Determinism:*A rule set is observably deterministic if the effect of rule processing as observed by the user of the system independent of the order in which the triggered rules are selected for processing.

The following two rules implement a response which is confluent but observably nondeterministic.

On <event E1>
if <condition C1>
do <send message to user>

On <event E1>
if <condition C1>
do <abort>

In this example, if the first rule is scheduled for firing before the second, then the user receives a message and the transaction is aborted. By contrast, if the second rule is scheduled before the first, then the transaction is aborted but no message is sent to the user.

Some examples can be considered from the conference management system as follows. The following are some Integrity Constraints and their corresponding rule sets are given below:

(a) A Referee cannot review any papers written by him.
  On (insert into Referee)
      if (Referee.name == Paper.authorname)
      do (abort)

(b)A submitted paper cannot be withdrawn after the withdrawal date.
      On (delete from Paper)
      if (CurrentDate > Paper.withdrawaldate)
      do (abort)
      On (delete from Paper)
      if (CurrentDate > Paper.withdrawaldate)
      do (Send message to Author that withdrawal date has passed and he cannot withdraw paper)

(c) The PC Chair can be an author of at most one paper.
      On (insert into Paper)
      if (contains (Paper.authors, PC-Chair) and PC- Chair.papers > 1)
      do (Abort)
      On (insert into Paper)
      if (contains (Paper.authors, PC-Chair))
      do (PC-Chair.papers += 1)

From the above examples, (b) is confluent, but not noticeably deterministic. Rules in (c) are not confluent. If the PC chair sends a paper having him as the first author and the paper has more than 5000 words, then depending on the order of execution of the rules, the author may or may not receive a message about the word count in the paper. Also, depending on how rules in (c) are scheduled, the count of the number of papers submitted by the PC-chair may or may not be incremented after submission of the paper. All the above rule sets are terminating since there is no recursive invocation of rules.

While several strategies like dynamic grouping of rules have been suggested to ensure termination and confluence, it has not been found effective for all rule sets. ADBs simply assume that rules are terminating and confluent [4], but this is not always the case and user interaction is some- times required to ensure their termination. The development of effective rule analysis systems awaits further work on communicating the results of analysis to users. This suggests that user should be modelled in a stronger sense in ADBs to realize the full capabilities of ADBs. For example,in nested transactions, the termination of rules may sometime require user interaction. User interactions and interaction among rules are conspicuous by their absence in active databases.

III.   INTEGRITY CONSTRAINTS AND TEMPORAL LOGIC

Active database rules provide a powerful mechanism for integrity constraint enforcement. While active databases began by concentrating on static integrity, support for several kinds of dynamic integrity issues are also present in many ADBs [5].

A. Static and Dynamic Integrity

Static integrity constraints are used to describe properties of database states; they restrict the set of possible states to those states, which are admissible concerning the constraints [6].Dynamic integrity constraints are constraints on state sequences instead of single states as in the static case [7].

While static integrity constraints express conditions on each state the system, dynamic integrity constraints express conditions for traversals in the state space of the system. An example of a dynamic integrity constraint from Conference management system (CMS) is that the state of the paper cannot be changed to "accepted" as long as the previous state is not "submitted".

B. Dynamic Integrity and Temporal Logic

In most commercial databases dynamic integrity support only single state transitions can be checked. The database provides support by which a rule can check for the "old" and "new" value of an updated data item and act accordingly. It cannot however check the history of updates on the data item, or arbitrary sequences of state transitions.

The need for reasoning based on history in ADBs has been felt by various researchers. In order to specify properties of state sequences, temporal logic has been explored, which extends predicate logic by special operators relating states

within sequences. Dynamic integrity constraints are also known as temporal integrity constraints [8]. Temporal formulae are built from non-temporal formulae, by iteratively applying logical connectives [6]:

- Quantifiers (∀, ∃) over possible or current objects and data;
- next operator referring to the next state;
- Temporal quantifiers always, sometime referring to all or some states in the future;
- Bounded temporal quantifiers always /sometime, be- fore /until.

The following is an example of dynamic constraint over CMS relations p: PAPER, r: REFEREE, pc:PROGRAMC, c:CONF, a:AUTHOR, expressed in our variant of temporal logic.

When (p.paperid=r.paperid and r.memname=pc.memname)
After (p.state="submitted")
(Always (pc.memname) and
Eventually (p.state="accepted" or p.state="rejected"))

This constraint specifies that a referee cannot leave the program committee once allotted a paper for review, and that review assignments cannot be changed; furthermore, that the state of the paper should eventually be changed from "submitted" to either "accepted" or "rejected".

C. Temporal Logic in Active Databases

In Active databases, dynamic integrity constraints formulated in temporal logic are converted to ECA rules.

For example, in [8], architecture is proposed for implementing temporal integrity constraints by compiling them into a set of active DBMS rules. This compiler allows automatic translation of integrity constraints formulated in past temporal logic into rules of an active DBMS. During compilation, the set of constraints is checked for the safe evaluation property. The result is a set of SQL statements that includes all the necessary rules needed for enforcing the original constraints. When the rules are activated, all up- dates to the database that violate any of the constraints are automatically rejected (meaning the corresponding transaction is aborted). This method converts past temporal logic formulae into a set of SQL statements. For past temporal logic formulas, the truth of the formula in state n depends only on the finite history D0; D1; ::::; Dn of the database.

[9] Uses past temporal logic for specifying conditions and events in the rules for active database system. An algorithm is presented for incremental evaluation of temporal conditions and a new framework for processing temporal constraints. This method stores the history of the database. In [6], monitoring schemes for dynamic integrity constraints are developed. Generating triggers for monitoring integrity from dynamic constraint formulae are addressed.

Referring to the "job" description of a database from the first section, we see that all of these approaches go beyond what is meant to be supported by a database. In fact, they address precisely some of the issues that are required for modelling the interactive behaviour of information systems that are defined by their service descriptions.

IV. SERVICE ORIENTED INFORMATION SYSTEMS

The community of IS researchers were historically concerned with the problem of information management in any dynamic system. A common abstraction for an IS was to view it as a collection of services. A "service" is a semantic process that makes up the functionality of the system. In the CMS example, activities like authors uploading a pa- per, the chair choosing reviewers, etc. are all semantic activities of the system.Mostly, the main problem addressed here is the description of services. A service description specifies how tasks in the service ought to be sequenced. The system is assumed to be made of actors who execute the services, and infrastructure that is used by the actors. The job of the information system is to facilitate interaction between an actor and the infrastructure or other actors of the system to ensure that the service description is honoured.

Languages for specifying service descriptions constitute the meta-model of the system. Some of the early meta-models used different variants of flowcharts. A flowchart is a sequence of tasks with various branching and looping conditions.A major problem with flowchart based approaches is that they tend to become too rigid when the amount of activity in the system increases. When a system has several concurrent activities with actors involved in more than one activity, the possibility of an actor being unable to exactly follow the rules of a flowchart becomes higher. As a result, handling "exceptions" becomes a major issue [10, 11, 12].

Information Systems as Reactive Systems: The weakness of the flowchart meta-model arises from the "closed" mental model that they exemplify. A flow chart assumes that semantic activities can only be performed by specific sequences of steps (which they describe) and nothing else. Any deviation from these sequences is considered illegal and an exception is raised.There is a growing realization that information systems are "open" systems which have only partial control over their dynamics. An instance of a service cannot unilaterally control the behaviour of its actors, nor can it control the behaviour of other services in the system, even though they share the same infrastructure (or the system state).

In other words:

An information system has to maintain its dynamic integrity even when it has only partial control over its own dynamics.

Perhaps the closest meta-model that comes to meeting such requirements is the reactive system.

Reactive systems are systems whose main role is to maintain an ongoing interaction with their environment, rather than transform a given problem to a solution.

A transformational system is the conventional type of system, whose role is to produce a final result at the end of a terminating computation. They can be modelled as a multi valued function from an initial state to a final state [13]. The behaviour of a transformational system is closed or algorithmic in nature [14].

On the other hand, the fundamental model of a reactive system is the Labelled Transition System (LTS), comprising of a set of states S and a set of transitions $\delta = \{ f \mid f: S \rightarrow \}$. At any instance, the system is in one of the state's $s \epsilon S$ and its response to external inputs depends on its current state. An interactive service constitutes a walk in the LTS graph where each transition may involve external interaction.

The open nature of reactive systems has resulted in duals of most computational paradigms like algebras, inductive reasoning and well-founded sets. Rather than being able to recursively specifies a computational domain from an initial set of axioms, reactive systems require the system model to iteratively enumerate all permissible states of the system where it can accept external stimuli and respond to them.Since it may not be possible to enumerate all possible initial states, a good design of an open system should allow for extending or contracting the system model (by adding or deleting states and their associated behaviours) dynamically.

Multi-stream interaction: Information systems contain a specific issue of multiple channels that is not generally addressed in reactive systems. Reactive systems are usually designed to react to one environment or a single interaction stream. If there is more than one external agent using the reactive system, the system does not differentiate between any of the agents.However, having a reactive system discriminate between its various channels of interaction increases it's expressiveness in such a way that it cannot be reduced to a system having a single stream of interaction [14, 15, 16].

In a multi-channel interactive system, the behaviour is not only history sensitive, but also channel sensitive. Multi-stream interaction further reduces the control a system has over its own dynamics.Some meta-models provide abstractions by which channels can be specify explicitly. However, these meta- models are unduly limiting in that, they statically fix the number of channels at the time the model is constructed. In reality, the number of channels in an IS cannot be known a priori, and channels can be created and destroyed dynamically.

The notion of contracts is increasingly being used to model interactive behaviour having multiple streams of interaction [17, 18, 19, 20]. A contract specifies dynamic integrity in the form of one or more normative constructs like duties, rights and prohibitions. An interactive service can follow any trajectory as long as it honours the contractual norms.

In the equation "IS = DB + Interaction", interaction modelling hence refers to modelling the contractual norms that govern the dynamic integrity of the IS.For the sake of argument, we introduce a simple norms based meta-model. We shall refer to this meta-model in conjunction with an underlying database system whenever we talk of the term "information system" further on.

The service descriptions of an information system aremodelled by an "interaction space" which is as follows: IS= (S, X, F, ψ). Here, S is the (finite) "state space" of all activities in the system, X is a finite but unbounded set of "channels" with which the system interacts with its external world, F is a set of functions or "methods" that relate states in the state space, and ψ is a set of constraints.

For each state $s\epsilon$ S the "attributes" of s, denoted byattr(s), is a set of state variables and their values.

Any channel x ϵX is in a specific state $s \epsilon S$ at any given point in time, and the channel x is said to "inhabit" state s. The attribute of a channel attr(x) is the attribute of the state that the channel inhabits. For each channel x, the term hist(x) associates it with a sequence denoting the history of its state traversals along with their corresponding attributes. The IS can call the function purgehist(x) for any channel x to purge its history and set it to null. Any channel x is also associated with two data streams called the "input" and "output" streams respectively. A channel can read from its input stream and write to its output stream. They are represented as in(x) and out(x)respectively.

Each state $s \epsilon S$ is defined by a set of "entry" conditions E(s). A channel may enter this state only if its history satisfies the constraints in the entry conditions.

F is a set of functions, each of which is of the form f: $S \rightarrow 2^s$. Depending on the specifications of the application, a function may be either called by the external environment or invoked by the system itself. A function represents an atomic transition.

Constraints in ψ are of the form $s_0 \wedge .... \wedge s_n M[s]$. This is read as follows: if there are channels inhabiting states $s_0 .... s_n$then state s gets modality M. Here M is one of the following:

- O[s] – it is obligated for a channel to inhabit s. If s is uninhabited, then all channels in s0....sn block, waiting for the obligation to hold.
- P [s] – it is permitted for a channel to inhabit s
- F [s] – it is forbidden for a channel to inhabit s. If one or more channels already inhabiting s, then they are blocked.

In the above, when a channel is blocked, its input and output streams are disabled. It cannot read any input and all the output it generates are buffered. Function calls are also disabled, until the state is enabled again.

V. CONTRASTING IS AND ADBs

Given a meta-model of an IS, we can have two different views of ADBs as

(a). Databases extended with rules

(b). Simple information systems providing.

A. Individualization and Active Views

The following table illustrates the two views of Active Databases. This table is essentially the same as the table in Section 1, except for the italicized words.

TABLE III. TWO VIEWS OF ACTIVE DATABASE

	Active Databases as Databases with rules	**Active Databases as specialized IS**
Nature:	Algorithmic	Interactive
State:	User data	User data, *rule-related* logs and histories, *rule related* user profiles
Job:	Updates and queries of data *by user as well as rule-driven*	Data backed rule-based services to users
Output:	Determined completely by query/update specification.	Individualized based on user history/preferences

While the first view is the standard one [2, 3], only the second view naturally endows Active Databases with needed functionality, such as such as individualization, use of logs and profiles, and user interaction.Individualization entails enabling users to formulate customized requests, as well as allowing them to customize their view of the feedback. It requires awareness of user characteristics and user preferences as well as of the history of user interaction with the system.

Active views in ADBs are motivated by the need to mediate between users and data stored in the database [21]. Active views can be used to define complex objects, events and conditions of interest. They mitigate the need for users to constantly issue queries to verify the existence of certain events or condition in the system. Active views also provide active caching of materialized views so as to support subscription services, notification services and to update user profiles. Further, materialized views can be updated automatically. Thus we have active views in ADBs which are analogous to the concept of individualization in IS.

B. Types of Operations

We use the term "operation" to mean a logical unit of dynamics. In database systems, operations are transactions and in information systems they correspond to services. Wedefine three kinds of logical operations and see how they would be typically designed in a database mental model and in an IS mental model.

We differentiate between three types of operations: user operations, system operations and prompted operations.

Consider the Conference Management System (CMS). The author submitting the paper is a user operation. User operations involve one or more external environments and are initiated by the environment. The system cannot predict when a user operation is initiated and in many cases, also what is the trajectory of the operation.Deleting a member from the members list when his membership expired is an automatic system operation. A system operation is one that is completely internal to the system. Environments do not control, or may not even be aware of system operations.

The system should have knowledge that a paper needs reviewers. After the details of the submitted paper are entered into the conference database, the system is able toprompt the chairperson of the program committee that reviewers are needed but the actual reviewer selection is left to the chairperson. Such an operation is a prompted operation. Prompted operations are initiated by the system, but involve one or more external environments for its execution. While a system may control when a prompted operation may begin, it may not have any control over how the operation itself proceeds.

Traditional databases support only user operations. Even here, only atomic operations are permitted – while the user may initiate an operation, she will have no control over the operation once it is initiated. Passive databases have no support for system and prompted operations. They are relegated to application programs running over the databases.Typical information systems on the other hand, support all the three kinds of operations.

ADBs fall somewhere midway. They support all three operations, but in a disconnected manner. Each rule that fires in an ADB is logically independent of all other rules. They can support system and prompted operations involving a single rule. The firing of a single rule may cause side- effects causing other rules to fire. However, they are all logically independent of one another. When a user operation creates a side effect firing a system operation, there is no direct way to correlate the two operations.

C. CMS as ADB and IS

The following are some examples from CMS that show limitations of ADBs from a database mental model:

Example 1: If an author submits a paper and the author is also a PC member, then in the same session, the author is not allowed to move between the author and PC member areas. For instance, when submitting a paper, an author is notsupposed to know what other papers were submitted, who are the reviewers, etc.

CMS as ADB: In ADBs, it is not possible to track a specific channel of interaction and reason based on its history. The simplest way to implement such a policy in an ADB is to have separate logins for authors and PC members. Hence, even though a PC member is an author of a paper (s). He has to login separately for these different roles.

CMS as IS: In an IS, the simplest way to implement this policy is to identify entry states for the PC member area and create an entry condition which checks to see that any in- coming channel has not visited any of the author specific states. Similarly, in all entry states of the author area, channels should be checked to verify that they have not visited any PC member areas.

Example 2: When the CFP is closed, the system should no longer allow users to upload new papers. However, if there are any users who have already started the process of registering and uploading a new paper, they should be allowed to complete their process.

CMS as ADB: In order to meet this constraint, there needs to be different status messages printed in a separate table that tracks where a particular user (given his/her login session id) is in the process state space. The ADB should allow for updates after the closing time if the user has al- ready an entry in the current session id showing that (s)he has come past the starting state.

CMS as IS: With an IS model based on norms, it is simply a matter of disabling the entry state(s) for the upload process after the time elapses. All channels that have passed this state continue to be in the system without being denied access.

Example 3: In some cases where there are more than one PC Chairs, the system should ensure that every logical operation like assigning a paper for review or finalizing the decision on a paper is done fully by one of the PC Chairs.

CMS as ADB: In the database perspective, this can be achieved by an advisory locking mechanism. When a PC chair begins a logical operation, a lock is set in the database. When the other PC chair logs on and tries to perform something, the system raises an exception based on the lock. However, since each update is considered independent of one another, the lock should contain sufficient information to determine who requesting operation is. Otherwise, it would deny permission to the holder of the lock itself.

CMS as IS: From an IS perspective, it is simply a question of maintaining different instances of services (in this case channels). Each channel takes up a service and exe- cutes it independently from the other channels. Since the context of a service is maintained within the instance, there is no need for any external coordination mechanism. Such an issue is complicated in ADBs since they do not have the notion of services that extend across rules and instances ofservices.

VI.   A ROADMAP FOR FUTURE ADBS

Based on the above differences in viewpoints we enumerate the following roadmap for future ADB research.

Rule instances: ADBs should support rule instances that run in their own contextual space and are logically isolated from other rule instances.Presently rule instances can be contrasted only when the condition part of their ECA rules differ. However, two rule instances are different even if they match in all the three parts of an ECA rule.Rule instances obviate the need for ad hoc approaches to differentiate rules from one another. They can also directly be mapped to different instances of services offered by the larger IS.

State space for rules: In addition to rule instances, a notion of a state space of a rule instance helps in tracking each individual instance of a service more precisely.A state space (or the "condition space") is a set of all different conditions under which events trigger actions. In many cases, the state space would be too complex for the database designer to specify a priori. However, there exist many other problems where it is not only possible, but also desirable to know the complete state space of a rule.

Given the state space of a rule, it is possible to incorporate integrity constraints across rules depending on how many instances are there in which state of the state space.

Instance specific ECA history: A means of storing the history of events, conditions and actions executed by a rule instance helps in tracking the trajectory of rule instances more precisely. Transactions may have to roll back or perform compensating operations if a rule instance is found to have violated dynamic integrity norms based on its ECA history.

Enabling, disabling and waiting on rules: It should be possible for individual rules (or rule instances) to explicitly enable, disable or wait on the triggering of another rule. It is of course possible to embed this functionality into the condition part of existing ECA rules. However, it is desirable to have this as a separate functionality for the following reasons:

1. Enabling and disabling of rules are not controlled by the rules themselves. Rules need not be aware of this external control over their behaviour. The condition part of an ECA rule will have to simply contend with integrity constraints pertinent to its context and not be concerned about constraints in the larger system.

2. Enabling, disabling and waiting primitives help in easier formulation of synchronization across disparate semantic services. Such constructs are especially helpful when rule instances are also incorporated in the model.

VII.   CONCLUSIONS

Active databases have been traditionally considered as data transformation systems, with research methods that borrow from traditional database arsenal. This is despite the fact that one of the design issues in active databases is in bringing application-level integrity concerns to the database level. As a result, designing the rule system of ADBs is a challenge and the mapping between application requirements and the system of rules remains complicated.

We argue for a change in viewpoint, so active databases is embraced as a special (restricted) type of an information system rather than a special (augmented) type of a database. As such, they are interactiveservice-providing systems rather than mere data transformation engines. This change in perspective offers a promising approach for ad- dressing ADB shortcomings, and reveals a roadmap of additional features for ADBs.

Note that our argument can also be applied to object- oriented database systems (OODBs) [22]. In contrast with object-relational databases, a "pure" OODB is best viewed as an IS. This change of perspective will yield a roadmap of desired OODB research directions analogous to that for ADBs.

REFERENCES

[1]   L.J.B.Essink, W.J.Erhart. Object Modelling and System Dynamics in the Conceptualization Stages of IS Dev., in Object Oriented Approach in IS. F.VanAssche, B.Moulin, C.Rolland (ed.) Elsevier Pub.1991 IFIP.

[2]   N. W.Paton, Os. Diaz. Active Database Systems, ACM Computing Surveys, Vol 31, No 1, March 1999.

[3]   K.R.Dittrich, S.Giatziu, A.Geppert. The Active Database System Manifesto: A Rule base of ADBMS features, In T.Sellis (ed.): Proc. (RIDS), Athens, Greece, Septem- ber 1995. LNCS, Springer 1995.

[4]   T. Risch, M. Skold.Monitoring Complex Rule Conditions. N. W. Paton (Ed.): Active Rules in Database Systems. Springer, New York 1999, pp. 81-102.

[5]   F. Casati, S. Castano, and M. Fugini.Managing Work flow Authorization Constraints through Active Database Technology. Information Systems Frontiers 3:3, Sep. 2001.

[6]   M.Gertz, U.W.Lipeck. Deriving optimized monitoring triggers from dynamic integrity constraints. IEEE Proc. of Data and Knowledge Eng.,Vol. 20(2), pp. 163-193, 1996.

[7]   F. Bry. Query Answering for Information Systems with Integrity constraints. Proc. IICIS, 1997.

[8]   D. Goldin. Persistent Turing Machines as a Model of Interactive Computation.in: K-D. Schewe and B. Thalheim (Eds.), First Int'l Symposium (FoIKS'2000). LNCS, Vol.1762, Springer-Verlag, Berlin 2000, pp. 116-135.

[9]   A.P. Sistla, O. Wolfson. Temporal Conditions and Integrity Constraints in Active Database Systems. Proc. 1995 ACM SIGMOD Int'l Conf. on Management of Data, pp. 269– 280, 1995.

[10]   F. Casati, S. Ceri, S. Paraboschi, G. Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. ACM Trans. on Database Systems, 24:3, 1999, pp. 405-451.

[11]   A. Borgida, Y. Murata. Tolerating exceptions in workflows: a unified framework for data and processes. In Int'l Joint Conf. on Work Activities, Coordination and Collabora- tion (WACC'99), ACM Press, pp. 59-68, 1999.

[12]   Z. Luo, A. Sheth, K. Kochut, J. Miller. Exception handling in workflow systems. Applied Intelligence: the Int'l J. of AI, Neural Networks, and Complex Problem-Solving Technologies 13:2, pp. 125-147, Sep. 2000.

[13]   Z. Manna, A. Pnueli. The Temporal Verification of Reactive and Concurrent Systems. Springer-Verlag, 1992.

[14]   D. Goldin, S. Srinivasa, B. Thalheim. Information Systems=DBS+Interaction: Towards principles of Information System Design. Proc. ER 2000, Utah, USA, Oct. 2000.

[15]   P.Wegner, D.Goldin. Interaction as a Framework for Modelling. In Chen, et. al. (Eds.) Conceptual Modelling: Current Issues and Future Directions, LNCS 1565, April 1999.

[16]   S. Srinivasa. Channel Sensitivity in Reactive Systems. Proc. Int'l Conf. on High Performance Computing (HiPC), Embedded Systems Workshop, India, Dec. 2001.

[17]   B. Meyer. Applying "Design by Contracts". Comm. of the ACM 25:10, Oct. 1992.

[18]   G. Weiss (Ed). Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, 2000.

[19]   W-J.v.d. Heuvel, H. Weigand. Cross-OrganizationalWorkflow Integration using Contracts. Business Objects and Component Design and Implementation Workshop. Proc. ACM OOPSLA 2000, Minneapolis, USA, Oct. 2000.

[20]   A. Reuter, F. Schwenkreis. Contracts -a low-level mechanism for building general purpose workflow management systems. IEEE Data Eng. Bulletin, 18(1), 1995.

[21]   L. Kerschberg, D. J.Weishar. Conceptual Models and Architectures for Advanced Information Systems. Applied Intelligence, Vol. 13,pp. 149-164,2000.

[22]   M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik. The Object-Oriented Database System Manifesto. Proc.First Int'l Conf. on Deductive and OODBMS, 1989.

[23]   J. Chomicki, D. Toman. Implementing temporal Integrity constraints using an Active DBMS. IEEE Proc. on Data and Knowledge Eng., 1994.

[24]   H.D.Ehrich, U.Goltz, J.Meseguer. Information Sys- tems as Reactive Systems. SchlossDagstuhl Intl Conf. and Research Centre for Computer Science, 16.02.-20.02.98,Seminar No:98071, Report No:200.

[25]   H.J.Schneider, A.I.Wasserman. Automated Tools for In- formation Systems Design, IFIP WG 8.1 Working Conf., Jan. 82.

[26]   Y. Saito, M. Shapiro, Optimistic Replication. Microsoft Research Tech Report MSR-TR-2003-60 , Oct. 2003.