PARDES- A Data-Driven Oriented Active Database Model

Opher Etzion
Faculty of Industrial and Management Engineering
Technion - Israel Institute of Technology
Haifa, 32000, Israel
e-mail: ieretzn@tx.technion.ac.il

Abstract

Most active database models adopted an *event-driven* approach in which whenever a given event occurs the database triggers some actions. Many derivations are *data-driven* by nature, deriving the values of data-elements as a function of the values of other derived data-elements. The handling of such rules by current active databases suffers from semantic and pragmatic fallacies. This paper explores these fallacies and reports about the PARDES language and supporting architecture, aiming at the support of *data-driven* rules, in an active database framework.

1 Introduction and Motivation

The term *Derived Data Item* has been defined as a "data item that appear to exist in its declared form, but it is actually derived from related data in the database" [KOENIG 81]. It was mentioned in the DBTG proposal [CODASYL 71], in which it is interpreted procedurally, i.e. whenever the derived data item is approached, a procedure is invoked to calculate its value. The Semantic Data Model SDM [HAMMER 81] views derived information in a similar fashion. Although in many cases a *lazy evaluation* of derived information is sufficient, later models acknowledged the necessity of calculating some of the derived data-items using an *eager evaluation*. This is justified either because of efficiency reasons (many retrieval operations vs. a relatively few updates) or due to effectiveness considerations (such as: real-time or other temporal constraints on the derived value's currency). The emerging technology of *active databases* is a likely candidate to support derived data.

The paradigm of *active databases* [MORGENSTERN 83] has gained a substantial interest in recent years, resulting in a few research prototypes. Among the objectives stated in the various works were:

1. Extending the modelling capabilities of a database to capture derived information, integrity constraints, alerters etc.
2. Tight-coupling between a DBMS and production system rules.
3. Using high-level declarative language in the database update process to reduce the development time of complex database applications.

The leading and most described architecture in current active database models is the *E-C-A* (*Event-Condition-Action*)[MCCARTHY 89]. Under this architecture a rule is composed of three components:

Event : either a database transition (when an Employee is Hired:) or an external event, such as a clock triggered event (each morning at 7:00 am do:).

Condition : a collection of queries, example: (If Employee.Salary > Employee.Manager.Salary).

Action : a user defined program.

The execution of a rule in an active database complies with the following scheme: Whenever an *event* occurs, find the rules associated with it, for each such rule, evaluate the *condition*, if it is satisfied then trigger the *action*. The execution of any rule may create other *events* and thus trigger more actions.

This research is motivated by several observations about the way that active databases handle derived data:

1. All the rules are treated as *event-driven*, while rules that are used to derive data are *data-driven* by nature. *Data-driven* rules are rules that define derived data-elements as functions of other data-elements and thus are triggered by modifications to certain data-elements in the database. *Data-driven* rules can be supported by restricted type of events and

restricted type of actions. Events are always modifications to a data-element that participates in a derivation, while actions are always update operations to a derived data-element. We can take advantage of the semantics of these dependencies to improve the performance of the execution process and the reasoning capabilities about it, and to reduce the size of the rule set.

2. The *E-C-A* model accommodates only partial information about the entire update process. While the *event* and *condition* parts are explicitly represented within the model, the logic of the *action* part is hidden inside a user defined program.

3. Since the model does not "know" about the logic of the *action* part, it is difficult to reason about the transitive closure of an event, i.e. to determine all the consequences of a single update. This knowledge is important for the analysis of the application's behavior and for optimization purposes[HUDSON 86]; [SEGEV 91]; [BOTZER 92].

The goal of the PARDES model, described in this paper, is to pose an alternative approach to the *E-C-A* model so that these deficiencies can be reduced. This is done by first exploring *data-driven* rules and obtaining a better modelling and architecture to handle them and then extending this model to support *event-driven* rules that achieve the full expressive power of the *E-C-A* model.

2 The Support of Derived Information in Current Active Databases

2.1 Models Supporting Derived Information

Several models which are either *active databases* or models whose main goal is the support of *derived information* were reported in the literature:

POSTGRES [STONEBRAKER 91] is an extended relational DBMS system. It implements production rules as queries that are constantly active. A rule is fired when its preconditions are satisfied.

HiPac [XAIT 89] is an object-oriented database, aimed for real-time applications. Rules are implemented as objects. The *E-C-A* paradigm was conceived within the HiPac project.

Sapiens [SAPIENS 90] is a commercially implemented system based on an hybrid representation. Sapiens implements rules associated with data elements, using implicit conditions and actions in some cases.

Madam [KOENIG 81] is a binary functional database model. Derivations are not implemented as rules, instead, the model supplies a SETL-like language, and a pre-compiler on top of it. The pre-compiler accepts as an input derivation assignments, that may be implemented in an incremental way. Like Sapiens the conditions and actions are inferred in some cases from the derivation definition. Unlike all the other described models, the control is not an internal mechanism of the system, instead it is explicitly coded in the SETL-like language.[1]

Cactis [HUDSON 86]; [HUDSON 89] is an Object oriented database which supplies a mechanism to control the order of calculations. The calculation itself in Cactis is a combination of user-defined functions and systems primitives. The control is determined by ordering the arguments to the object-oriented methods in an attribute grammar graph and applying a control algorithm.

Ariel [HANSON 92] is a relational active database model that includes an optimization model for condition testing based on rule indexing and a discrimination network. The basic architecture conforms with the *E-C-A* paradigm.

Other implementations of the *active database* paradigm include **sturburst** [LOHMAN 91], **DIPS** [SELLIS 89], **Ode** [GEHANI 91].

Most of these models are compatible with the *E-C-A* paradigm in the sense of the common problems discussed below. Exceptions are noted in the problems' descriptions.

2.2 Semantic Problems in Current Implementations.

Throughout this paper we denote a *data-driven* derivation rule as an assignment consisting of the derived variable on the left hand side and the derivers (variables that participate in the derivation) along with their roles on the right hand side. Example: the expression:
a := $b_1 + b_2 \times b_3$
is interpreted as: for all the instances in the database, a is computed as a function of b_1, b_2, b_3 according to this assignment. a is referred to as **PDI**(Persistent Derived Information), while $b_1, \ldots, b_n$ are referred to as the **derivers** of a.

The semantics of derived values in the *E-C-A* models is equivocal in the following cases:

I. *The Multiple Relationship Problem.* Example: Let Y be defined by two different rules as:
Y := X + 3

[1] Actually it is an hybrid of a persistent language and an active database.

Y := X + 5

This situation, in the general case, produces a non-deterministic result because it is not clear which rule is fired first. In **Sapiens** and **Cactis** the control mechanism executes the two rules according to their location in the rule set, thus the last assignment determines the persistent value, but the first assignment is useless. In **Madam** one can use the programming language to make conflicting assignments, the assignments are carried out according to their order in the program. In **Postgres** and **HiPac** the outcome is non deterministic either because an arbitrary rule is selected, or because both rules are selected in an unknown order.

II. *The Conflict Resolution Problem.* The firing conditions are not mutually exclusive. An example is:
Y: = X + 3 when Z > 0
Y: = X + 5 when Z > 5
Unlike case **I.**, these assignments are conditioned. The conditions, however, are not mutually exclusive. Some strategies have been developed for run-time detection and determination of conflict resolution (see [IOANNIDIS 89] for a survey). In the discussed systems, **Postgres** and **Ariel** enable (but do not enforce) the definition of priorities among rules. In the other models, the case is similar to case **I.**

III. *The Non Determinism Problem.* When a PDI is derived from two or more rules, even if they have mutually exclusive derivers, the relationship between the derived value and its derivers is not clearly defined. An example is:
Y := X + 3
Y := Z + 1

The value of any instance of Y is determined by the last rule that has been applied to calculate it, but we cannot reconstruct the value of Y by looking at the values of X and Z. Furthermore, if X and Z are updated in the course of the same transaction, problem **I.** returns. This problem can be generated in all the discussed models. See the discussion of case **I.**

2.3 Pragmatic Problems:

IV. *The Redundancy Problem.* An implementation of triggers may result in redundant computations. An example is:
Y:= X + 3
Z: = X + Y + 5
W: = X + 2*Y + 3*Z
Under a standard evaluation of triggers, if X is modified then W is modified four times.[2] In this case the correct value is obtained, but the update time complexity is exponential instead of linear[HUDSON 86]. This problem is common to any intensive interconnected application. As a result of this problem additional semantic problems may arise. Consider the following case:
Y: = X + 3
Z: = X + Y + 5
W: = W + 1 , when Z is modified.

Under a standard evaluation of triggers, a modification of X results in 2 modifications of Z, thus W counts the update twice. Consequently, the value of W is wrong. Only the **Cactis** model addresses this problem.

V. *The Halting Problem.* It is possible to generate infinite loops such as:
Y: = X + 3
X: = Y + 2

Sapiens halts after one loop, unless both derivations are defined as recursive. Other models permit infinite loops.

VI. *The Direct Update Problem.* Whenever a computed value is directly updated, the dependency between the derived value and its derivers is lost. An example is:
Y:= sum(X)
When an instance of Y is modified directly, bypassing the derivation, this instance looses the connection with the values of its derivers. **Cactis** distinguishes between an *intrinsic* attribute and a *derived* attribute. The latter type cannot be modified directly in any case, which leaves no room for exceptions. All the other models take the other extreme and enable direct update of derived values without any indication that such an update has occurred.

VII. *The Multiple Situations Problem.* The *E-C-A* approach leaves all the different implications of a dependency to the system designer's discretion. For example the dependency:
Activity.Activity-Cost := Project.Overhead-Cost + Activity.Cost-Per-Day * Activity.Duration
is represented by a collection of rules, that are invoked by the following events:

1. Project.Overhead-Cost is modified.
2. An activity is removed from a project.

[2] once as a direct result of the update of X, once by Y, and twice by Z.

3. An activity is transferred to a new project.
4. An activity refers to a non-existing project.
5. A project is deleted.
6. Cost-Per-Day is modified.
7. Duration is modified.
8. Activity-Cost is directly updated.

Though a reasonable system designer should realize that all these cases must be handled and generate the correct action in each case, the result is sensitive to the system designer's capabilities and the consistency of the system w.r.t. the dependency specification has to be verified for each dependency. Furthermore, 8 rules are required to express a single dependency.

VIII. *The Global Picture Problem.* In most of the models there is no way to model the *global picture*, i.e., to reason about the possible consequences of update operations. This is capability is useful in the validation of an application and in verifying that there are no undesired implications to an update operation. It is also useful for various optimizations: optimization in the application's execution (eliminating redundant updates, problem IV.) and optimization of the transaction mode.

3 The PARDES Model- Concepts and Facilities

3.1 Goals

Derivation Closure : Constructing an update model that guarantees the database's consistency with respect to its derivation rules at the required currency, and is able to support consistency at all times of any rule if required.

Programming Style : Designing a rule language that satisfies two distinct goals: semantic accuracy and high-level abstractions. We propose the *invariant approach* as an appropriate programming style. This approach is further discussed in Section 4.

The Behavior Model : The behavior of the update process should be represented as an integral part of the database and provide reasoning capabilities about this process.

Update Control : The update control mechanism should achieve minimality in the number of update operations, eliminating redundancy and establishing determinism in the execution.

Exception Handling : Exception handling should be integrally embedded in the rule language using high-level abstractions.

Flexible Transaction Model: The transaction model should support a flexible transaction model that handles different requirements for degrees of consistency.

3.2 The PARDES architecture

An application designed in PARDES is constructed and executed using the following phases:

Definition: The database is a structural object oriented database. The database schema is defined by the system designer using a data definition language. Invariants are part of the schema definition. Figure 1 shows a simple example:

CLASS	ATTRIBUTES
Order	Customer, Product, Items (Product, Quantity) Order-Price(PDI)
Customer	Status, Credit, Debit(PDI), Balance(PDI)
Product	Price, Manufactured-Quantity, Available(PDI)

INVARIANTS

D1: Order-Price := sum (Quantity * Price)
D2: Debit := sum (Order-Price)*0.8
when Status = preferred
Sum (Order-Price)
otherwise
D3: Balance := Credit - Debit
D4: Available := Manufactured-Quantity - sum(Quantity)

C1: Balance > -2 * Average (Order-Price)
C2: Available > 0

Figure 1- Schema Definition

D1...D4 are derivation rules while C1,C2 are constraints. Note that inferable referential details are omitted (example: D1 is interpreted as $\forall x : [Order(x) \rightarrow Order\text{-}Price(x) = \sum_{y \in Items(x)}(Quantity(y) * Price(Product(y))]$

Translation: The schema and invariants are being translated into a **dependency graph**, which is an executable data resolving all the required references among different classes. This graph determines the transitive closure of an update operation, namely, all the derived update operations that might be invoked by the execution of this update operation and facilitates the reasoning about the update process

including optimizations. A similar graph has been proposed in [SHETH 92] in the context of transaction control, however there are two major differences:

1. In PARDES the graph is inferred and not given explicitly.
2. In PARDES the graph includes ALL the information required for execution and not control information only.

Situation Interpreter: At run-time the **dependency graph** is used to infer the action to be taken. Example: for the operation:
D1: Order-Price:= sum (Quantity * Price),
the following actions are inferred:

1. when new Item is created in the Order: calculate its contribution to the Order-Price.
2. when an Item is removed from the Order: subtract its added contribution from the Order-Price
3. When a Quantity in an Order is modified: add the difference between the new and old values multiplied by the Price.
4. When Price is modified: add the difference between the new and old values multiplied by the Quantity (this should be done for any Order referring to this Product.

Run-Time Controller: The Update Control mechanism uses a technique equivalent to topological order on the dependency graph, to eliminate redundancy in update operations.

Exception Handling Component: All the models discussed above allow the specification of Exception-Handlers *only* in an *imperative* way, at the system designer's discretion. The PARDES model provides Exception-Handling abstractions as a part of the invariant language[ETZION 91b], as well as support of different types of null and defaults [ETZION 91a]. The abstractions stand for generic strategies for handling exceptions in cases of: syntactic (range) violations, referential integrity violations, constraints violations and direct modifications of derived updates (bypassing the derivations). The basic strategies are:

restrict - Any violation results in a transaction rollback.

nullify - The violating value is replaced by a null value.

cascade - Other modifications occur in the database to restore consistency (these three strategies are generalization of referential integrity modes as defined in [DATE 81].)

forgive - The violating value is persisted and recorded as an exception [BORGIDA 85]; this strategy has been referred to in [BALZER 91] as "pollution marker".

disconnect - Disconnect the violating value from the rule. Is it possible to attach a private exception-handler in this case.

Figure 2 shows example of exception-handling modes interpretation for a given case. Note that *nullify* is not a meaningful mode in this case.

For Customer 112, The Balance is directly updated, bypassing the derivation D2, decreasing the Balance. Note that nullify is not a valid mode in this case. The actions to be taken according to the modes are:

Restrict	Rollback this update operation
Cascade	Add the difference to an *Order* with name unknown.
Forgive	Permit the update and mark it as an exception.
Disconnect	Disconnect this instance from the rule's range.

Figure 2: Exception-Handling Modes

Flexible Transaction Mechanism: There are two decision problems in the transaction model of an active database:

1. Should an operation to update a PDI instance be triggered by modifications of any of its derivers?
2. If the answer to the first decision is positive then Should the PDI be updated synchronously and ensure consistency at all times w.r.t. the deriver's update operation? (these are two separate issues).

The results of these two decisions determine the materialization strategy. It should be noted that the decision is local, it may differ from one derived data-element to another. The **materialization strategy** of any derived update can be defined in a global level (w.r.t the original update operation), local level (w.r.t to all its derivers) or link level (w.r.t. a specific direct or indirect deriver). The system designer can use any consistent combination of these levels, achieving maximum flexibility. Materialization Strategies are:

fully consistent : the update operation should reside in the same atomic transaction along with the operations updating the base operations.

quasi consistent : the update operation may reside in a different transaction than the base operations, however inconsistency is allowed only until the later transaction has been executed. In case of failure the system emulates a retroactive rollback, by issuing compensating transactions.

loosely consistent : the update operations may reside in a different transaction than the base operations. No compensation is issued on failure, this is a case of a *weak invariant.*

periodically coordinated : the derived update operation is not executed within the regular transaction system. A refresh operation periodically derives these values. The frequency may be determined by temporal or other coherency conditions [ALONSO 90]; [SHETH 92].

semi consistent : the derived update operation is executed on the first retrieval operation that requires the derived value. This mode is coupled either with the **quasi consistent** mode or with the **loosely consistent** mode.

Passive : The derived value is always virtual.

The topic and associated algorithms are discussed in [ETZION 92]. The decision about assigning a consistency mode to a rule may be recommended by an optimization model that improves its own results by getting feedback based upon the application's behavior. This issue is discussed in [BOTZER 92]. This flexible mechanism becomes possible only due to existence of the **dependency graph** and the **executability** property of this graph.

4 The Invariant Language

Rule languages tend to be formal. Some languages such as Telos [MYLOPOULOS 90] use well-formed formulae in first-order-logic or its subset.
Example : $\forall x \exists y : P(x, y)$.
In other languages assertions are written in a IF...THEN... syntax, which is equivalent to a subset of first-order-logic (usually Horn Clauses). The advantage of logic-like languages is their expressive power, as well as their accuracy. Specifications written in logic are executable. Empirical studies [BRAINE 78] have indicated that the concepts of formulating are difficult for most people to understand even after an extensive training. Our design-goal is to make it feasible for a systems analyst who is not necessarily trained in formal logic to state the invariant assertions. Consequently the required language should be simpler. We also require our language to support data independence. In addition to the data independence features that are used in most database languages (independence of physical location, independence of indices etc.) we employ additional two types of independence:

Situation Independence: All possible modifications are inferred by the system and not specified on a case-by-case basis. This is a consequence (and one of the main benefits) of the programming-by-invariants style.

Referential Independence: The matching of two classes is determined by the system using semantic equivalences among class-attributes. Unlike the relational operation *join* that requires explicit matching, the matching is implicit. This feature enables independence of the invariant language of the physical structure. Any change of location of a data-element (such as splitting classes or unifying classes) is transparent to the invariant language. Only the translation process should be re-activated.

This section concentrates upon the invariant language features.

4.1 The Invariant Principle

The style of programming proposed in the PARDES model for expressing dependencies is based upon programming by invariants. Each dependency is stated by an invariant definition. Programming by invariants consists of two main principles:

1. Abstract language of assertions that states the dependencies in a simple way, hiding structural and referential aspects.

2. A computational process that enforces the database to satisfy all the invariants in an active fashion.

4.2 Invariant Types

There are two types of invariants:

Logical Invariants : C1, C2 in the Figure 1 are logical invariants (constraints, assertions) [NICHOLAS 82]; [MORGENSTERN 84]. These are assertions that a consistent database must satisfy. These assertions are handled as data-driven rules, because modifications of the participating attributes trigger the re-evaluation of the assertion.

Computational Invariants D1...D4 in Figure 1 are computational invariants. They are used in Calculation of data-elements that are arithmetic functions of other data-elements.

Note that conditional and referential dependencies are embedded in logical and computational invariants.

4.3 Invariant Execution Rules

Several rules apply to the implementation of programming-by-invariants.

1. The invariant definition is executable. No extra programming is needed to maintain the dependency.

2. An Invariant may include a conditional expression such as:

   D2: Debit := sum (Order-Price)*0.8
   when Status = preferred
   Sum (Order-Price) otherwise

   The conditions must be mutually exclusive. Mutual exclusiveness is enforced by conjuncting each condition with the negations of all the previous conditions, thus assuming a priority ordering. Example, in the rule:
   x:= y when z >0 ; y+5 when w > 0,
   the second condition is interpreted as:
   $w > 0 \wedge \neg (z > 0)$.

3. Each PDI appears on the left hand side of exactly one invariant definition.

These rules enforce semantic accuracy. The definition of each derived element is well-defined and deterministic. Under this set of rules each derived data-element is uniquely determined in a given database.

4.4 Properties of the Invariant Approach:

Compactness: The language is clear and concise.

Extended Data Independence: The user is liberated from aspects of location and reference while defining the invariant.

Semantic Clarity: The invariant language ensures unique interpretation of each invariant.

4.5 The Support of Event-Driven Rules

The PARDES model was constructed to accommodate *data-driven rules*. Support in *event-driven rules* is required in order to achieve a general active database model. Thus, entities of types: **Event** and **Action** are also supported by the language and control model. *event-driven rules* are partitioned into:

Event-driven derivations that are expressed in invariant syntax, with the addition of **Event-list.** These derivations are triggered by events that belong to this list and not by the modification of any of the derivers.

Event-driven UDA (user defined actions) that are programs external to the model. The definition of UDA requires to specify all the possible inputs and outputs of the UDA so the UDA is also modelled by the **dependency graph.**

5 Summary

In this paper we have described a paradigm for handling a data-driven rules in active databases and sketched an architecture to support this paradigm. The main contribution of this research is supplying higher level language and extended modelling capabilities for data-driven rules. A prototype implementation of PARDES has been constructed using C++. Further research is done in several directions: the main directions are: extending the model to support a distributed environment, and extending the model to cope with temporal environment. On the application side the PARDES model is being used to construct an intelligent CIM database.

Acknowledgements

Giorgio Ingargiola has helped to shape the ideas and features of the first version of PARDES. Various parts in the PARDES research project has been performed in the Technion by David Botzer, Avigdor Gal, Gali Raznik, Anna Rubinshtein and Ronnen Armon, the latter has constructed the prototype implementation of PARDES.

References

[ALONSO 90]: R. Alonso, D. Barbara, H. Garcia-Molina- Data Caching Issues in Information Retrieval System. ACM TODS 15(3), pp 359-384, Sep 1990

[BALZER 91]: R. Balzer- Tolerating Inconsistencies. Proc 13th Int. conf. on. Software Engineering. May 1991, pp 158-165.

[BORGIDA 85] : A. Borgida- Language Features for Flexible Handling of Exceptions in Information Systems. *ACM TODS* 10(4), Dec 1985, pp. 107-131.

[BOTZER 92]: D. Botzer- Active Database Optimization, M.sc. Thesis, Technion- Israel Institute of Technology, Sep 1992.

[BRAINE 78]: M. Braine- On The Relation between Natural Logic of Reasoning and Standard Logic. Psychological Review, 1978.

[CODASYL 71]: CODASYL. *Data Base Task Group Report.* ACM 1971.

[DATE 81] : C.J. Date- Referential Integrity *proc. VLDB* 1981.

[ETZION 91a] : O. Etzion- Handling Active Databases with partial inconsistencies. *Lecture Notes on Computer Science, 548, pp 171-175,*, Oct 1991.

[ETZION 91b]: O. Etzion- Active Handling of Incomplete or Exceptional Information in Database Systems. *Proc WITS 91, pp 46-60*, Dec 1991.

[ETZION 92]: O. Etzion- Flexible Consistency Modes for Active Database Applications. *Technion- Israel Institute of Technology, Technical Report ISE-TR-92-2.* January 1992.

[GEHANI 91]: N.H. Gehani, H.V. Jagadish- Ode as an active database: Constraints and Triggers. *Proc. VLDB 91, PP 327-336*, 1991.

[HAMMER 81]: M. Hammer, D. Mcleod- Data Base Description with SDM: A Semantic Data Base Model. ACM TODS 6(3), 1981.

[HANSON 92]: E.N. Hanson- Rule Condition Testing and Action Execution in Ariel, *Proc SIGMOD 92, pp 49-58*, June 1992.

[HUDSON 86] : S. Hudson, R. King- CACTIS: A Database System for Specifying Functionally Defined Data. *proc. IEEE OODBS Workshop* 1986.

[HUDSON 89]: S. Hudson, R. King- CACTIS: A Self-Adaptive, Concurrent Implementation of Object Oriented Database Management System. *ACM TODS 14(3), pp 291-312*, Sep 1989.

[IOANNIDIS 89]: Y.E. Ioannidis, T.K. Sellis- Conflict Resolution of Rules Assigning Values to Virtual Attributes. *Proc. ACM SIGMOD conf, pp 205-214*, 1989.

[KOENIG 81]: S. Koenig, R. Paige- A Transformational Framework for the Automatic Control of Derived Data. *Proc VLDB 81, pp 306-318*, 1981.

[LOHMAN 91]: G.M. Lohman, B. Lindsay, H. Pirahesh, K.B.Schiefer- Extensions to Starburst: Objects, Types, Functions and Rules, *CACM 34(10), pp 94-109*, Oct 1991.

[MCCARTHY 89] : D. McCarthy, U. Dayal- The Architecture of an Active Data Base Management System. *Proc. 1989 ACM SIGMOD conf.* June 1989, pp 215-224.

[MORGENSTERN 83]: M. Morgenstern- Active Database As a Paradigm for Enhanced Computing Environment. *Proc VLDB 83, pp 24-42*, 1983.

[MORGENSTERN 84]: M. Morgenstern- Constraint Equations: Declarative Expression of Constraints with Automatic Enforcement. *Proc VLDB* 1984 pp 291-200

[MYLOPOULOS 90]: J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis- Telos: A Language for Representing Knowledge About Information-Systems. *ACM TOIS, 8(4), pp 325-362*, 1990.

[NICHOLAS 82]: J.M. Nicholas- Logic for Improving Integrity Checking in Relational Database models. *Acta Informatica* 18(3). pp 227-253, 1982

[SAPIENS 90]: What is SAPIENS. *Sapiens Technologies LTD.*

[SEGEV 91]: A. Segev, J.L. Zaho- Data Management for Large Rule Systems. *proc VLDB 91.*

[SELLIS 90]: T. Sellis, C.C. Lin, L. Raschid- Data Intensive Production Systems: The DIPS Approach. *Sigmod Record, Sep 1989.*

[SHETH 92]: A. Sheth, M. Rusinkiewicz, G. Karabatis- Using Polytransactions to Manage Interdependent Data. Chapter 14 in: A. Elmagarmid (ed)- Transaction Models for Advanced Database Applications, Morgan-Kaufmann 1992.

[STONEBRAKER 91]: M. Stonebraker, G. Kemnitz- The POSTGRES Next-Generation Database Management System. *CACM, 34(10), pp 78-93*, Oct 1991.

[XAIT 89]: Xerox Advanced Information Technologies- HiPac: A Research Project in Active, Time-Constrained Database Management. *Final Technical Report. XAIT-89-02.*