Inconsistency Management in Schema Evolution and Data Base Migration

Fernando Arango Izasa

UNIVERSIDAD NACIONAL DE COLOMBIA. Facultad de Minas.
Systems and Informatics School, Medellín - Colombia
farango@unalmed.edu.co

Recibido para revisión Abr 2005, aceptado Abr 2005, versión final recibida May 2005

Abstract: Automation of conceptual schema evolution process should allow database management systems to smoothly perform schema *transformation* and data *migration*. However, current approaches lack the ability to perform well in all circumstances. They can't manage, for example, schema restrictions strengthening without lost of information or the aid of an external migration tool, which, besides of being costly, have to assume database correctness responsibility. In this paper we present a novelty approach to schema evolution automation that performs well in all circumstances. It allows the schema transformation and data migration to be performed in the realm of day by day system use without lost of information or the use of an external migration tool. The approach key idea is to make database management system, not only capable of performing schema changes and data migration, but capable of managing schema evolution/data migration inconsistencies. Instead of pretending an abrupt elimination of evolution related inconsistencies, our approach opts for predict, permit, detect and gradually eliminate them. The approach elements are stated in the framework of the formal approach to the object oriented software development, OASIS. OASIS allows us to rigorously specify and to automatically obtain a software ambient prototype oriented to the creation, animation, evolution, versioning and versions interoperation, of OASIS conceptual schemas, that has, in addition, inconsistencies management capabilities.

Keywords: Software Maintenance, Schema Evolution, Object Oriented Development, Inconsistencies management.

1 SCHEMA EVOLUTION AUTOMATION

Software maintenance has shown to require 65%-70% of personnel resources and 80% of total software development cost [Yourdon (1989) and Meyer (1988)]. Maintenance is related to several factors, i.e. software errors, requirement changes, or the use of a prototype oriented development approach.

A Database Management System (DBMS) offers persistence, integrity, input/output, and access control to several, probably different, databases (DB). A "Conceptual Schema" specifies to DBMS the particularities of each database. The maintenance of most databases consists, therefore, in changing its conceptual schema and adapting the database to the changed schema. This maintenance process has been called *schema evolution* [Banerjee, Kim, Kim y Korth (1987)].

The most commonly used approach to schema evolution is done by first rewriting the schema, and then "migrating" the DB in a process outside the realm of the DBMS, using specifically written programs or specialized tools. This approach, however, makes migration an abrupt process that may involve massive data transformation, lost of information, large amount of resources, and interfere severely with normal system operation.

A more modern approach pretends to smooth evolution making the DBMS capable of automatically perform the schema transformation and data migration. To that end several approaches incorporate to the DBMS schema transformation operators, mechanism to build and apply data adaptation functions, and mechanism to create schema versions capable of isolate the programs form the schema changes.

2 CURRENT APPROACHES, REQUIREMENTS & REMAINING PROBLEMS

A large amount of research and development is related to schema evolution automation: [Banerjee et al. (1987), Barbedette (1991), Brèche, Ferrandina y Kuklok (1995), Carsí (1999), Clamen (1992), Kajal, Claypool y Rundensteiner (1998), Lautemann (1997*b*), Lautemann (1997*a*), Lerner y Habermann (1990), Lerner (1996), Odberg (1995), Penney y Stein (1987),Rundensteiner (1994), Scherrer, Geppert y Dittrich (1993), Scholl, Laasch y Tresch (1991), Scholl, Laasch y Tresch (1994), Tresch y Scholl (1992), Waller (1991) and Young-Gook y Rundensteiner (1997)].

Whatever is the approach, it requires answering specific questions. Among them are relevant to our - approach the following ones:

- Which transformation operators will be offered?
- Which adaptation mechanisms will be implemented?
- How schema, data and programs correction will be maintained?

2.1 State of Art Concepts

To answer the questions, most approaches rely in a small set of specific ideas and concepts defined as requirements to schema evolution automation. Understanding those ideas is a simple way to comprehend the different approaches. Consequently, in this paragraph analyze the state of art from the point of view of the requirements the approximations are based upon.

Completenessis the capability of a schema transformation operation set to assume every possible schema change [Banerjee et al. (1987)]. Completeness has guided the definition (or adoption) of *complete* sets of elementary (or *primitive*) schema transformation operators in most approximations [Banerjee et al. (1987), Brèche et al. (1995), Carsí (1999), Penney y Stein (1987), Scherrer et al. (1993), Tresch y Scholl (1992) and Young-Gook y Rundensteiner (1997)]. A proof of completeness is sketched in Banerjee et al. (1987) for its particular primitive set.

Robustness is the capacity of a schema transformation operation set to offer evolution operators - applying a more *abstract level* evolution step [Brèche et al. (1995)]. Completeness has motivated the conception of all sort of complex evolution operators, involving schema transformations (or reorganization) and database migration [Brèche et al. (1995), Carsí (1999), Lerner (1996), Rundensteiner (1994) and Tresch y Scholl (1992)].

Flexibility is the capability of the transformation operation set user, to incorporate new evolution operators that can be added to the already existing ones. Flexibility has been supported by the use of Meta level modeling (reflection) [Tresch y Scholl (1992)] or specialized schema evolution *frameworks* [Kajal et al. (1998)].

Operator Correctness obligates transformation operator to leave schema and database correct after been applied [Banerjee et al. (1987)]. Correctness has driven to make schema transformation operator (primitive or complex), capable of doing complementary, by default, schema changes and database adaptations in order to assure that schema an database stay in correct states [Banerjee et al. (1987), [Brèche et al. (1995), Carsí (1999) and Scherrer et al. (1993)]. In what follows we call operators that warrant correctness *AutoContent* operators[1].

DBMS Correctness obligates the DBMS to reject any evolution operator application if it leaves the schema or database in an inconsistent state. Most - approaches reject the operator application if there is any chance for inconsistencies being introduced [Banerjee et al. (1987) and Scherrer et al. (1993)]. The approach in Carsí (1999), however, rejects the operator application only if inconsistencies are actually introduced.

Schema Invariants are rules considered necessary and sufficient for a conceptual schema being considered correct [Banerjee et al. (1987)]. Most approaches define a specific fix set of invariants tailored to its particular schema model [Banerjee et al. (1987), Brèche et al. (1995), Carsí (1999) and Scherrer et al. (1993)]. Violations to the invariants are used to trigger complementary changes o to reject transformation operator application.

Designer defined adaptation function allows the modification of the complementary by default, adaptation functions, associated to schema transformation operators. Some approaches allow only the further application of additional adaptation functions, leaving to the default ones the responsibility for schema and database correction [Brèche et al. (1995) and Kajal et al. (1998)]. Other approaches allow the modeler to substitute the default functions leaving the correction responsibility to the modeler (or to the tool) [Carsí (1999). Lerner y Habermann (1990) and Penney y Stein (1987)].

Schema versioning allows the programs to interact with the database trough the schema they where created with. Some approaches maintain old schema versions as database *views* [Brèche et al. (1995), Scholl et al. (1991) and Young-Gook y Rundensteiner (1997)]. Other approaches extend versioning to the database, keeping all the database-schema versions required by the programs Clamen (1992), Lautemann (1997*b*) and Odberg (1995)].

Schema version interoperation is the capability of database changes to propagate into the different schema versions. In the views approach this propagation is implicit to the existence of only one database version. In the database-schema versioning approach the database changes propagation is supported for a set of attribute inter version dependency functions specifically defined to the purpose [Clamen (1992), Lautemann (1997*a*) and Odberg (1995)].

[1]So not call them incorrect operators.

2.2 Unresolved Issues

In spite of the solutions' diversity offered, some problems still remains unsolved. In fact, different solutions to different problems are in conflict to each other, making compromise necessary.

The use of *AutoContent* operators, as a way to warrant correctness, is in opposition to completeness and use simplicity, and is, in addition, of no use in the case of schema versioning and schema versions interoperation. Particularly:

- The operator correctness requirement obligates the elementary operators to be robust (and not very "elementary"), associating complementary changes to the basic one, in contradiction to its natural aim for simplicity.

- If complementary changes are determined by predefined '*rules* [Banerjee et al. (1987) and Brèche et al. (1995)], they don't necessarily agree to the modeler's intentions (i.e. they can induce severe information loss [Banerjee et al. (1987)]), making difficult to conduce the evolution process.

- If complementary changes are under modeler control [Carsí (1999), Lerner y Habermann (1990) and Penney y Stein (1987)], they leave open the possibility for introducing inconsistencies, in opposition to operator correctness.

- The DBMS correctness can make some evolution steps impossible or difficult. Consider, for example, an evolution step strengthening database restrictions and, consequently, opening the possibility for leaving many database objects in an inconsistent state[2]. In this circumstance, some approaches opt for reject the changes [see Banerjee et al. (1987) and Scherrer et al. (1993)], making the evolution step impossible. Whereas other approaches [Carsí (1999)] opt for verifying the actual occurrence of inconsistencies before rejection, making the evolution step possible but only if database is pre corrected. In fact [Banerjee et al. (1987)] proof for its primitive operator set completeness, did not consider operations applicability restrictions, nor does any consideration for database adaptation completeness exist in the literature.

- As is the case of an outside DBMS migration process, current approaches perform the database adaptation in an abrupt way, making difficult a gradual particularized object by object adaptation approach. Consequently the schema evolution team has to identify and collect this kind of changes, possibly using a specialized tool, and, in the best case, include them in an evolution operator application.

- Operator correctness requires to pre consider the effect of each schema change on schema invariants, so complementary corrective actions can be planed in advance. This rules out the introduction of new schema invariants, and so a fix invariant set is the norm.

- Modern OO models allow the specification of several types of static and dynamic restrictions that are susceptible to change during evolution. In those circumstances, an acceptable data actualization, from the point of view of a given version, could be unacceptable, from the point of view of another version. Consequently, database correctness can not be assured under schema versions interoperation.

[2]This is the case, for example, of changing a one to many relation cardinality to a one to one, or introducing a new non null key attribute.

2.3 Requirements

To point our approach contribution, and to compare it to state of art, we defined a set of requirements to schema evolution automation. Those requirements elaborate on the already existing ones, by making additions or changes oriented to solve the remaining problems.

To keep it short, in what follows we only mention the new requirements and the modified ones.

We renamed Banerjee's Completeness as **Transformative Completeness**, and introduced **Adaptive Completeness**, as the capability of adaptation mechanisms to assume every possible database adaptation to a transformed (or new) schema.

Tranformative Gradualness is the capability of a schema transformation operation set to change the schema at the lowest level of granularity possible. It allows grater control on changes and the step by step introduction of complex schema constructions.

Adaptive Gradualness is the capability of performing any particularized object by object adaptation, without relying in an external tool. It subrogates Adaptive Completeness, allows grater control on database adaptation, and imposes the need to include in the DBMS mechanisms to point the required adaptations.

Transformation operators' orthogonality is the possibility of applying individual schema transformation operators without considering correction issues. It allows the free application of schema transformation operators assuring transformative completeness.

Schema transformation - database adaptation orthogonality, is the possibility for independently defines schema transformations and data adaptation aspects of schema evolution. It gives total control on database adaptation to the modeler.

Flexibility of invariant set is the possibility for including new schema invariants when considered necessary by modeler. A flexible invariant set allow the introduction of method oriented invariants.

Operator Correctness is changed to **Evolution Step Correctness** in order to release the individual operator from the obligation of maintaining schema and database correction, and to transfer it to the set of operations that comprise an evolution step.

Schema versioning and **schema versions interoperation** are maintained as a way to make possible evolution when programs modification is difficult, and to set the basis for the integration of database having different schemas.

Database inconsistencies awareness is the capability for each schema version to know database inconsistencies relative to its own ruling. This requirement allows schemas to incorporate protection mechanisms and work properly in the realm of database - schema version mismatches.

3 INCONSISTENCIES MANAGEMENT AS A - SOLUTION

The contribution of our approach to the stated requirements is a consequence of three key factors: an inconsistencies management system, a controlled versioning and interoperation model, and the use of reflection for the specification of evolution operators and schema invariants.

Our inconsistencies management system doesn't try to avoid inconsistencies, but it rather **accepts, predicts**, **detects** and **advises** its occurrence, and then points the involved elements in order to help correction. The effect of this approach on requirements is derived for the following factors:

- Schema transformation operators are no longer rejected on the basis of schema and database correction issues. This increases its applicability and assures transformational completeness.
- Complementary, by default, schema changes and database adaptations are no longer required after schema elementary transformations. This allows very low granularity schema transformation and database adaptations giving modeler more control on changes.
- It makes possible a particularized object by object adaptation approach assuring adaptive completeness.
- Inconsistency detection and advice performs the identifications and collection of needed adaptations role, so external specialized tools are no longer needed.
- Inconsistency advice gives schema versions the capability to know database inconsistencies relative to its own ruling, allowing schemas to incorporate protection mechanisms and work properly even if database inconsistencies occur.

Our approximation to schema versioning and interoperation is based in extending the versioning to the database [Clamen (1992) and Odberg (1995)], and introducing inter version relationships as new schema rules (see 4.3.5). Violations to these new schema rules can be managed as any other inconsistency, giving the modeler the possibility for controlling the results of interoperation. The effect of this approach on requirements is derived for the following factors:

- It makes possible schema versions interoperation even if they have significative differences.
- It gives modeler control on the way database changes propagate into different schema versions. This allows modeler to choose among inter versions object compatibility, and database-schema compatibility in the different versions.
- Allows the modeler to plan a gradual database adaptation process and carrying it out with minimum interference to system operation.

The use of reflection in the OO model (see 4.3.2), for the specification of evolution operators and schema invariants, takes advantage of the model's expressive power making possible the flexible specifications of invariants and new complex evolution operators.

4 MAGIC: SOFTWARE ENVIRONMENT FOR OO CONCEPTUAL SCHEMA ANIMATION, EVOLUTION, VERSIONING, AND INTEROPERATION

Our approach is stated as a software environment oriented to the creation, animation, modification, versioning and versions interoperation of OO conceptual schemas under the OASIS model and language. We named our software environment proposal with the acronym MAGIC[3].

In this section we present the environment features and the main concepts that support its realization.

4.1 OASIS: MAGIC's OO Formal Model

OASIS approach belongs to a declarative school line of thinking trying to fuse the expressive capability of the OO model, with the semantic precision of formal languages.

[3]From its Spanish description: "Medio Ambiente para la Gestión de InConsistencias ..."

OASIS model is based in the traditional concepts of the object oriented development approach Booch (1994) and Meyer (1988), which see information systems as composed of a set of multi related, interacting and class classified objects. Particularly:

- OASIS objects pass trough several states, defined by its *attribute values* and *relations*, as a consequence of object interactions (asking and giving *services*). An object *event* is one of its most elemental possible state changes, and an *operation* is a complex state change made up several events. *Integrity restrictions* rule the possible object states, *triggers* rule when an event has to occur, and *pre conditions* and *post conditions* rule the possible events. *Protocols* restrict additionally the possible sequences of state changes (the object life).

- OASIS classes classify the objets and allow the specification of objets features. OASIS classes are classified as *primitive*, *elemental* and *complex* classes. Primitive classes represent abstract data types, elemental classes are defined using primitive classes, and complex classes are defined using other elemental and complex classes. Complex classes can be defined using other classes trough specialization/generalization, and aggregation operators.

OASIS language is used in class specifications. A set of class specifications is a *conceptual schema*. An OASIS class specification is a theory in Dynamics logic [*Dynamic Logic* (1984)], and is divided in sections. The section named "**constant attributes**", "**variable attributes**" and "**derived attributes**" have the theory symbols alphabet. The section named "**derivations**", "**constrains**", "**triggers**", "**preconditions**" and "**valuations**" have the logic well formed formulas corresponding to the inter attributes dependencies, integrity restrictions, trigger conditions, and events pre and post conditions, respectively.

For an extended description of OASIS see Torres, Ramos, Sánchez y Pastor (1998).

We adapted OASIS to the needs of our multi inter operating schema version approach. In particular:

- We introduced warning level Integrity restrictions, to cope with inconsistencies.

- We introduced two new specification sections ("**forward dependencies**" and "**backward dependencies**") to cope with inter version relationships (see 4.3.5).

4.2 MAGIC Functions

MAGIC main functions are as follows:

- **Conceptual Schema Animation:** MAGIC allows the creation and use of a database in accordance with any given OASIS conceptual schema. That is achieved simply by allowing users to activate the events and queries described in the schema, on a given compliant database (which can be empty in the beginning).

- **Conceptual Schema Creation & Transformation:** MAGIC allows the creation and transformation of an OASIS conceptual schema, trough a flexible set of schema transformation operators that satisfies the transformational completeness requirement. A MAGIC "schema" is then a sequence of schema states that evolves continuously.

- **Schema Versioning:** MAGIC allows any conceptual schema state to be set as a schema "version", and to be used on the associated database. It requires, however, fully compliance whit schema invariants from any schema state, to be set as a version.

- **Database Adaptation:** MAGIC allows adapting the (old) associate database to the (new) schema version in any conceivable way.

- **Schema Versions Interoperation:** MAGIC allows the schema versions to be used simultaneously, and allows the controlled (and flexible) propagation of any database actualization to the any (other) active schema version.

- **Inconsistencies Management:** MAGIC implements an Inconsistencies Management System as described in the former section. The inconsistencies managed comprise, violation to schema invariants, violations to schema version ruling, and violations to inter version relationships.

4.3 MAGIC Concepts

Several concepts support the realization of MAGIC functions. Some of them appeared in the literature, and some where created for, (or adapted to) our approach. In what follows we present the main concepts supporting each MAGIC function.

4.3.1 Conceptual Schema Animation

MAGIC capability for conceptual schema animation is a direct consequence of the OO model and specification language adopted. An important goal of OASIS OO model formalization was, in fact, the creation of a specification prototyping environment.

OASIS specification prototyping capability is a direct consequence of the following factors:

- Any OASIS conceptual schema defines a mathematical theory and any OASIS database is an interpretation, which is model of the theory. Any OASIS database query corresponds to a demonstration in the theory. Any OASIS database actualization corresponds to a localized change in the theory's model that can be validated[4] by only evaluating the affected theory well-formed formulas.

- Demonstration and well-formed formulas validation in OASIS theories can be automated by reifying the OASIS logic in other logic having operational semantic [Toval (1994)].

The MAGIC schema animation capability is then obtained reifying OASIS logic in a first order as described further (section 5).

4.3.2 Conceptual Schema Creation & Transformation

MAGIC approximation to schema creation & transformation is based in the Ramos, Pelechano, Penadés, Bonet, Canós y Pastor (1993) and Carsí (1999) approach to the specification of an OASIS conceptual schemas editor and evolution tool. These approaches use reflection in the OASIS model to view any OASIS specification as a set of OASIS object (or "MetaObjects"). The structural and dynamical properties of the MetaObjets are then specified in an OASIS schema (or "MetaSchema") taking advantage of the expressive power of the OASIS model.

MAGIC approximation has, however, several differences from Ramos et al. (1993) and Carsí (1999) - approach, specifically:

- Elementary evolution operators are no longer *Auto-Content*, and, consequently, they are simpler: they implement the operation ***create***, ***delete*** and ***change*** for any schema component (simple or composite). MAGIC's elementary operators satisfy the requirements of transformational completeness and gradualness.

- MetaSchema is explicitly coded and animated by MAGIC's conceptual schema animation capability. This allows modeler to easily introduce and modify complex schema transformations and schema invariant, satisfying evolution operator's flexibility and robustness, and schema invariant's flexibility requirements.

4.3.3 Schema Versioning

As any database objects, schema components evolve passing by a succession of states, as schema transformation operators are applied. MAGIC allows any conceptual schema state to be set as a schema "version", and to be used on the associated database.

To that end MAGIC's MetaSchema includes a "fix-version" event. The fix-version event has as precondition the satisfaction of all schema invariants, so only correct schemas can be fixed as versions.

4.3.4 Database Adaptation

For a newly introduced schema version being capable of interaction with the database, it has to establish how its vocabulary elements project on the database components. In MAGIC this is defined by inter version schema vocabulary elements relationships (in what follows for short "inter version relationships"). Inter version relationships define the vocabulary elements that are the "same" in two (consecutive) versions[5], the elements that are functionally related, and the newly introduced or deleted elements.

Inter version relationships, are specified in two special schema section created for that purpose, namely, "**forward dependencies**" and "**backward dependencies**", corresponding to dependencies to next and last versions respectively. Most inter version relationships are, however, assumed by default, and only non-equality relationships, name changes and vocabulary elements meaning changes, has to be specified. For a complete description of this schema section see Arango (2002).

Inter version relationships completely define how old database objects will be projected to the new database objects, and how the new database objects will be defined, in particular:

- How new database objects will be derived from the old database objects, including which objects remain, which objects have to be created, and which have to be destroyed or modified.

- How new objects attribute values will be derived from the old objects attribute values, including which attribute value remains the same, which have to be deleted or introduced, and which have to be recalculated (from old attribute values).

The MAGIC inter version relationships together with the inconsistencies management system assures satisfaction of adaptive completeness requirement.

4.3.5 Schema Versions Interoperation

MAGIC approach to schema version interoperation is based in the concept of objects with *facets* [Clamen (1992)], which extends to database the schema versioning. In this approach every object offers a different

[4]It must be verified if new interpretation even models the theory.

[5]They must be projected to the same database elements.

facet to each schema version, forming a sequence of *adjacent* facets. The object facet corresponding to a schema version has all properties declared in the version object class, and manages all interactions with that schema version. Properties in a facet are conceptually different from the properties in a different facet.

The effects of an event *activated* in a facet, from its correspondent schema version, can propagate to other facets as an *induced* event, in a controlled way, through inter version relationships.

An induced event is simply defined from another adjacent facet event by the satisfaction of a related inter version relationship. The propagation is controlled by interoperation facet policies ruling if the facet accepts induced events, if the induced events can violate facet rules, and which inter version relationship defines the event.

4.3.6 Inconsistencies Management

By suppressing operator correctness requirement and accepting schema version interoperation we accepted schema and database inconsistencies. Inconsistencies belong to one of the following types:

- Schema invariant violations caused by application of schema transformation operators.
- Database - schema mismatches caused by schema rules strengthening during evolution.
- Database - schema mismatches caused by induced events during schema version interoperation.
- Inter version relationships mismatches caused by the facets interoperation policies.

The MAGIC inconsistency management system helps controlling and correcting inconsistencies by implementing mechanisms that predict, detect, advice and allow inconsistencies correction. These mechanisms operate as follows:

- The schema invariant violations are advised when schema components are queried. To that end, each schema invariants related to a specific component is evaluated when the component is queried. Correction of these inconsistencies is done through schema transformation operators.
- Strengthened schema rules are pointed as *susceptible* as a way to predict this kind of inconsistencies. Susceptible rules are verified when data elements affected by the rule are queried. If the verification fails a warning advice is attached to the data value. Change of attribute values is possible by using special generic events. The use of these events is preconditioned to the attribute participation in an inconsistency.
- For each version schema rules that are *susceptible* to violation by induced events are pointed as a way to predict this kind of inconsistencies. The formal characterization of these rules is described in Arango (2000). Susceptible rules are verified and violations are advised and corrected as before.
- *Susceptible* Inter version schema rules are identified by examining the facet's interoperation policies. Verification and advice is done, however, only when requested by system users. No correction is possible for these inconsistencies.

5 OASIS REIFICATION IN MAUDE & MAGIC SPECIFICATION IN OASIS

As mentioned before, MAGIC capability for conceptual schema animation was obtained by reifying OASIS logic in a first order equational logic following the ideas in Toval (1994). The chosen logic was the *membership equational logic* supported by the language MAUDE, developed in the Stanford Research Institute [Clavel, Durán, Eker, Lincoln, Martí-Oliet, Mesenger y Quesada (2000)]. OASIS reification gives as a result an equational logic theory susceptible of automatic inference under MAUDE equational rewriting engine. Any OASIS database query or actualization is, in fact, done in MAUDE as a demonstration in the resulting theory.

Following Toval's approach, OASIS reification is based in creating MAUDE sorts representing the different types of schema components. The main differences among MAGIC and Toval's reification are the greater complexity of the used OASIS model, and the possibility for using any MAUDE sort as an OASIS primitive class. Our reification covers the following aspects of MAGIC functionality: database queries and actualizations, use of MAUDE sorts as OASIS primitive classes, conceptual schema versioning and interoperation, and the capability for OASIS Meta level schemas animation (see 4.3.2).

Taking advantage of the last aspect of MAGIC functionality, we obtained MAGIC capabilities for schema creation, transformation and versioning by its specification in a OASIS metal-level schema.

A more detailed description of OASIS reification in MAUDE and MAGIC specification in OASIS will be the subject of future papers. The interested reader can see [Arango (2002)].

6 EXAMPLE

OO software development process has been characterized as *iterative, incremental* and *prototypic* [Booch (1994)]. In this context rather than obtaining the "*right*" system from the beginning, we pretend to get a useful one that can be easily modified without loosing information. In our example[6] we show how MAGIC inconsistencies management can help improving an initially poor design with minimum trauma.

To have control on books loan, the following information has been stored for each title in a library: *title*, *author*, number of *volumes* owned, a list of *borrowers'* names and a list of the correspondent borrowers' *phones*. To keep data actualized, two events have been defined: a *loan* event and a *devolution* event, both affecting the lists of borrowers' names and phones. The corresponding OASIS schema has, consequently, only one class (see figure 1).

After a while the consequences of a poor, non-normalized, design are noticed: no control on borrowers' behavior is possible, borrower phones can be different in different books, and borrowers track is lost when books are returned.

To correct problems, the database is normalized separating borrowers and titles data. To that end the borrowers' data is stored separately in objects of class *member*, and the loans are registered as a relation.

Database adaptation to new schema is driven by inter version relationships as follows:

- Inter version relationship in the "**backward**" section of *member* new class, defines the *member* class population.
- Inter version relationship in the "**backward**" section of *title* class new version, defines the *member* objects related to each *title* object.
- Inter version relationship in the "**forward**" section of *title* class old version, assures that all member phones registered in the old *title* objects will be registered in the new *member* objects.

Normalization strengthen schema restriction, making phones functionally dependent on names, and restricting to one the number of different phones associated to each member. That implies the possibility of inconsistencies on the cardinality restriction of the *phone* attribute in *member* objects. Database inconsistencies management helps to gradually correct this type of inconsistency, so a fully adapted correct new database can be obtained.

This correction is driven by inter version relationships as follows:

- Inconsistencies in *phone* attribute of member objects will be detected and advised when the attribute is queried.
- Corrections can be made directly by special generic correction events (assignation of a value to the attribute).

[6]It's an adaptation of Arango (2002).

In addition, if old schema version remains in use, the inconsistencies management systems will help the old schema to behave as a fully updateable view:

- When in phone attribute inconsistencies are corrected, inter version relationships inconsistencies, related to old version *title* class "**forward**" restrictions will appear. To correct these inconsistencies user has to correct *phones* in the old *title* objects.
- Updates to *phones* or new loans, driven by the old schema versions will trigger the whole inconsistencies driven corrections.

7 CONCLUSIONS

In this paper we show how the inclusion of an inconsistencies management system, besides a complete set of schema transformation operators and database adaptation mechanisms, enhance DBMS schema evolution capabilities making it capable of copping with all possible evolution cases. Those cases include all possible conceptual schema transformation forms, and all possible forms of adapting the database to the transformed schema. An important case covered by our approach is the gradual object by object adaptation that is needed when schema transformation strengthens schema restrictions, leaving many objects in an inconsistent state. The inconsistencies management system makes possible, in addition, schema version interoperation even in the case where contradictions among schema versions exist.

REFERENCIAS

Arango, F. (2000), Analysis and management of inconsistencies in the interoperability of oo conceptual schemas versions, *in* 'Workshop, IDEAS 2000, Cancún - Mexico'.

Arango, F. (2002), Gestión de las Inconsistencias en la Evolución e Interoperación de los Esquema Conceptuales OO, en el marco formal de OASIS, PhD thesis, Department of Informatic Systems and Computation, Polytechnic University of Valencia, Valencia - Spain.

Banerjee, J., Kim, W., Kim, H. y Korth, H. F. (1987), 'Semantics and implementation of schema evolution in object-oriented databases', *SIGMOD* pp. 311–322.

Barbedette, G. (1991), Schema modifications in the lispo2 persistent object-oriented language, *in* 'European Conf. of Object-Oriented Programming, Geneva - Switzerland'.

Booch, G. (1994), *Object Oriented Analysis and Design With Applications*, 2th edition edn, Bejamin/Cummings, Menlo Park (Calif.).

Brèche, P., Ferrandina, F. y Kuklok, M. (1995), Simulation of schema change using views, *in* 'Proceedings of the 6th Int'l Conf. On Data Base and Expert Systems Applications, London'.

Verslon 1

```
class Libro
   Constant attributes
      Titulo : char (' ) towards (1,1);
      Autor : char (' ) towards (1,1);
      Volumenes : int (0) towards (1,1);
   variable attributes
      Nombres : char (nil) towards (0,*);
      Telefonos : int (nil) towards (0,*);
   events
      prestar(Nom:char, Tel:int);
      devolver(Nom:char);
   valuations
      [prestar(Nom:char, Tel:int)] insert Nom in Nombres;
      [prestar(Nom:char, Tel:int)] insert Tel in Telefonos;
      [devolver(Nom:char, Tel:int)] remove Nom in
Nombres;
      [devolver(Nom:char, Tel:int)] remove Tel in
Telefonos;
   end class
```

Verslon 2

```
class Libro aggregation of
   dynamic relational Prestamo alias for Socio towards (0,*) from(0,1) list;
   constant attributes
      Titulo : char (' ) towards (1,1);
      Autor : char (' ) towards (1,1);
      Volumenes : int (0) towards (1,1);
   Events
      prestar(Nom:char, Tel:int);
      devolver(Nom:char);
   valuations
      [prestar(Nom:char, Tel:int)] insert Socio(Nom) i ı Prestamo;
      [devolver(Nom:char, Tel:int)] remove Socio(Nom) in Prestamo;
   backwards dependencies
      {Socio(N)∊ Prestamo)} if {N∊Nombres};
   forward dependencies
      {(S.Nombre∊ Nombres) ∧ (Telefonos[index (S.Nombre) in Nombres]
in T}
                                        if {S∊ Prestamo and T=S.Telefono }
end class

class Socio
   constant attributes
      Nombre : char (' ) towards (1,1);
      Telefono : int (0) towards (1,1);
   backwards dependencies
      ∃S∊ Socio{S.Nombre=N ∧ T∊ S.Telefono} if
         ∃L∊ Libro{N∊ L.Nombres ∧ T=(L.Telefonos)[indice N in
L.Nombres]}
   forward dependencies
end class
```

Figura 1: OASIS Schema

Carsí, J. (1999), OASIS como marco conceptual para la evolución del software, PhD thesis, Universidad Politécnica de Valencia - Spain.

Clamen, S. (1992), Type evolution and instance adaptation, Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. Tech. Report, CMU-CS-92-133R.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Mesenger, J. y Quesada, J. (2000), A maude tutorial, *in* 'CICA, Sevilla - Spain'.

Dynamic Logic (1984), Reidel, chapter Handbook of Philosophical Logic II, D.M.Gabbay & F.Guenthner, pp. 497–694.

Kajal, T., Claypool, J. y Rundensteiner, E. (1998), Serf: Schema evolution through an extensible, re-usable and flexible framework, Technical report, Computer Science Department, Worcester Polytechnic institute. Computer Science Technical Report Series,WPI-CS-TR-88-9.

Lautemann, S.-E. (1997*a*), A propagation mechanism for populated schema versions, *in* 'Proc. of the 13th Int'l Conf. On Data Engineering (ICDE), Birmingha - U.K.'.

Lautemann, S.-E. (1997*b*), Schema versions in object-oriented database systems, *in* 'Proc. of the 5th Int'l Conf. On Database Systems for Advanced Applications (DASFAA), Melbourne - Australia'.

Lerner, B. (1996), A model of compound type changes encountered in schema evolution, Technical report, Computer Science Department, University of Massachusetts at Amherst. 96-044.

Lerner, B. y Habermann, A. (1990), Beyond schema evolution to database reorganization, *in* 'Proceedings of the ACM Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA), ACM SIGPLAN Notices', Vol. 25, Otawa - Canada.

Meyer, B. (1988), *Object Oriented Software Construction*, Prentice Hall, Hemel Hempstead.

Odberg, E. (1995), Multi Perspectives: Object Evolution and Schema Modification Management for Object-Oriented Databases, PhD thesis, Norwegian Institute of Technology.

Penney, J. y Stein, J. (1987), Class modification in the gemstone object oriented dbms, *in* e. Norman Meyorwitz, ed., 'Proc. of the 2nd Conf. On Object Oriented Programming Systems, Languages and Applications (OOPSLA)', Orlando, Florida, pp. 111–117. ACM, ACM Press, Special Issue of SIGPLAN Notices. V. 22, No. 12, Dec. 1987.

Ramos, I., Pelechano, V., Penadés. M., Bonet, B., Canós, J. y Pastor, O. (1993), Análisis y diseño orientado a objetos de un entorno de prototipación automática basado en oasis, Technical report, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia.

Rundensteiner, E. (1994), A clasification algorithm for supporting object-oriented views, *in* 'Proc. Int'l Conf. Information and Knowledge Management', pp. 18–25.

Scherrer, S., Geppert, A. y Dittrich, K. (1993), Schema evolution in no2, Technical report, Institut für Informatik Universität Zürich. Technical Report Nr. 93.12.

Scholl, M., Laasch, C. y Tresch, M. (1991), Updatable views in object-oriented databases, *in* 'Proc. Second DOOD Conf.', pp. 189–207.

Scholl, M., Laasch, C. y Tresch, M. (1994), Evolution towards, in, and beyond object databases, *in* 'Proc. 3rd GI Workshop Information Systems and Artificial Intelligence, Hamburg'.

Torres, P., Ramos, I., Sánchez, P. y Pastor, O. (1998), Oasis version 3.0: Un enfoque formal para el modelado conceptual orientado a objeto, Technical report, Departamento de Sistemas Informáticos y Computación, Facultad de Informática, Universidad Politécnica de Valencia. Servicio de Publicaciones, SPUPV-98.4011.

Toval, J. (1994), Formalización Algebraica de un Entorno de Producción Automática de Prototipos Orientados por Objetos, PhD thesis, Universidad Politécnica de Valencia.

Tresch, M. y Scholl, M. (1992), Meta object management and its application to database evolution, *in* '11th Int'l Conf. On the Entity Relationship Approach, Karlsruhe - Germany'.

Waller, E. (1991), Schema updates and consistency, *in* 'Second Intn'l Conference DOOD'91', pp. 167–188.

Young-Gook, R. y Rundensteiner, E. (1997), 'A transparent schema-evolution system based on object-oriented view technology', *IEEE Transaction on Knowledge and Data Engineering* 9(4).

Yourdon, E. (1989), 'Re-3: re-engineering, reestructuring and reverse engineering', *American Programmer* 2(4), 3–10.