Verification, Validation and Integrity Issues in Expert and Database Systems: Two Perspectives

Frans Coenen
Department of Computer Science,
University of Liverpool, UK
frans@csc.liv.ac.uk

Barry Eaglestone
Department of Information Studies,
University of Sheffield,UK
B.Eaglestone@shef.ac.uk

Mick Ridley
School of Computing and Mathematics,
University of Bradford, UK
M.J.Ridley@scm.brad.ac.uk

Abstract

This paper is directed at two central objectives. The first is to identify and establish areas of overlap between the expert and database system domains. The second is to present a view of existing and ongoing work concerning the Verification, Validation and Integrity (VV&I) of rule base and database systems. The paper combines reviews from the two perspectives of the expert systems and database systems communities, with the express aim of identifying possibilities where VV&I knowhow of the one may also be of value to the other (and vice-versa), especially with respect to the identified areas of overlap.

1 Introduction

Validation, verification and integrity (VV&I) issues are of primary importance to both expert and database systems (ESs and DBSs), since they relate to the fundamental question, "does a system provide a useful solution to the problems it sets out to solve?". This paper examines these issues from two perspectives, i.e. expert systems and database systems. The aim is to identify areas where there is common ground and solutions provided by one or other of these technologies are mutually beneficial.

The authors identify a number of areas where the domains of databases and expert systems interconnect. The first is concerned with the parallels in system structure/architecture that can be drawn when database and expert systems applications are compared. The second is founded on the generally acknowledged observation that the relational database model can be expressed in terms of clausal form logic. The third is in the field of deductive databases which seek to extend the functionality of the relational model by allowing predicate calculus rule constructs to be applied to data. Fourth is in the area of active databases where expert system style sets of rules are used to automatically update records given some "triggering" event. The fifth candidate for overlap is the link between Constraint Logic Programming (CLP) and constraint databases.

Further details concerning these areas of overlap are presented in Sections 3, after a brief defining overview of expert and database systems given in Section 2. The remainder of the paper is set out in a review style with sections covering spheres of interest within the expert systems V&V community, and the database integrity community. Each Section commences with a brief outline of the "state of the art", including significant references, and ends with a short discussion of possible "cross-over" technologies.

2 Expert and Database Systems

Expert and database systems provide complementary tools for representation and utilisation of information. In the following two subsections we present an overview of these two technologies.

2.1 Expert Systems

Expert systems typically comprise a rule base and an inference mechanism. Rules are generally described using predicate calculus, and more precisely a simplified version referred to as *clausal form*. Clausal form logic constructs are typically expressed as follows:

$$if < antecedent > then < consequent >$$

where the antecedent part comprises one or more propositions/predicates which, if they evaluate to `TRUE`, will establish the proposition/predicates contained in the consequent. Alternatively a production rule format, popular in many expert systems environments, may be adopted in which case the antecedent part tends to be referred to as the *condition part* and the consequent the *action* part:

$$if < condition > then < action >$$

In the case of PROLOG a special clausal form known as horn clauses:

$$\begin{array}{rcl} < consequent > & :- & < antecedent > . \\ (< action > & :- & < condition > .) \end{array}$$

is used. If a clause has no consequent part the antecedent part is considered to be unconditionally" true. In such cases the clause is usually referred to as a *fact*.

The principal advantage of clausal forms is that reasoning can be implemented using resolution techniques ([59]) where an inference engine attempts to satisfy the condition/antecedent part of a rule by replacing it with facts and/or other rules. This process continues until only unconditionally true facts remain, it which case the condition part is said to "succeed". If suitable facts are not found the condition is said to "fail". The process of finding a matching substitution that will make a condition part of a rule equal to an action/consequent part of a rule (or a fact) so that the resolution can be applied is called *unification*. Thus the engine moves from "if patterns" to "then patterns" using the if pattern to identify appropriate substitutions for the deduction of a new antecedent. This resolution mechanism is referred to as forward chaining (i.e. *goal driven* searching). The reverse is backward chaining (i.e. *data driven* searching) where a rule based system can form a hypothesis and use the rules to work backward towards hypothesis supporting assertions. A much more detailed discussion concerning the operation of rule based systems can be found in text books such as Winston [69] or Bench-Capon [8].

As a result of the principle of resolution, rules in a rule base can be arranged in a hierarchy comprising a top level (root) rule, any number of intermediate rules and a set of leaf rules (facts). In practice there may be several root rules, each node in the hierarchy may have any number of branches and the hierarchy may be extremely unbalanced. The satisfaction of the root rule can then be likened to the satisfaction of a compound query couched in terms of a number of sub-queries such that the result of one sub-query acts as input to another sub-query. Of course each sub-query may comprise further sub-queries, and consequently the resulting hierarchy is similar to that associated with an expert system's rule base.

2.2 Database Systems

Database technology can be viewed as a restricted form of rule base, in that it is concerned primarily with the storage of facts. However, the aims have a different focus to those of ES technology. Rather than providing support for reasoning, the aim of DB technology is more modest, i.e. to provide shared access to facts. It achieves this through the representation of facts as data in such a way as to maintain the illusion that persistent data looks like the information that it represents. This is done by describing stored data at different levels of abstraction and mapping between them, to provide "computer world" interpretations of data manipulations expressed in "real world" terms. The advantage should be that we maintain a natural representation of information as data, sharable by relevant applications, including ones not yet thought of, thus preserving investment in both data and applications [31]. If the facts are to be used for reasoning, this is done by the database application, not the database itself. The technology that delivers this approach has evolved through three generations of data models:

- hierarchical and network,
- relational, and
- object and object-relational.

Milestones in the evolution of DB technology are, an algebra for DBs (relational algebra [22]), extensible type systems (object-oriented features [4, 63]); and universal access to databases (the Internet). These respectively provide a basis for database languages and theories, an enormously extended range of potential applications embracing all forms of data object (not just numbers and text), and an enormously increased population of potential users. Relational and object databases currently constitute the mainstream of DB technology. However, we note that other data models have been developed by the research community. These include extended versions of the mainstream data models, for example to include strong spatial and temporal features, and data models based on different formalisms and paradigms such as functional, semantic, deductive, active and constraint data models.

3 overlap

In the following subsections a number of possible areas of overlap between database and expert systems are identified. Where necessary, in these section, rule fragments are presented using the PROLOG expert systems programming language ([21, 14]).

3.1 Structure

When considering the structure/architecture of database and expert system application some obvious parallels may be drawn as shown in Figure 1. Both have what we might refer to as an "operating" systems — the Data Base Management System (DBMS) in the case of a database application, and the inference engine in the case of and expert system. Both are traditional software systems in the sense that they are developed using mainstream software engineering tools and techniques. The verification, validation and testing of such systems is thus also undertaken using established techniques and thus fall outside of the scope of this paper. Both applications also have an "information repository" — a database in the case of a DB application, and a rule/fact base in the case of an expert system. In the case of deductive, active and constraint databases further information, in the form of rules and constraints, is added to the information repository thus enhancing the similarity between the two classes of system. In both cases it is the V&V and integrity issues concerned with these information repositories that is of concern in this paper. A further similarity that can be drawn from Figure 1 is at the system conceptualisation level where, in the case of a DB application, we have a *schema* and in the case of and expert system the idea of an *ontology* although the latter is a fairly recent concept. In the context of expert systems an ontology is a set of definitions of (hierarchically ordered) classes, objects, attributes, relations, and constraints. It's main function is to provide a vocabulary for the expression of domain knowledge. An ontology, like a DB schema, is thus a knowledge-level description (Newell [56]) in that it is independent of any representational formalism (Van Heijst [65]). Thus an overlap between the

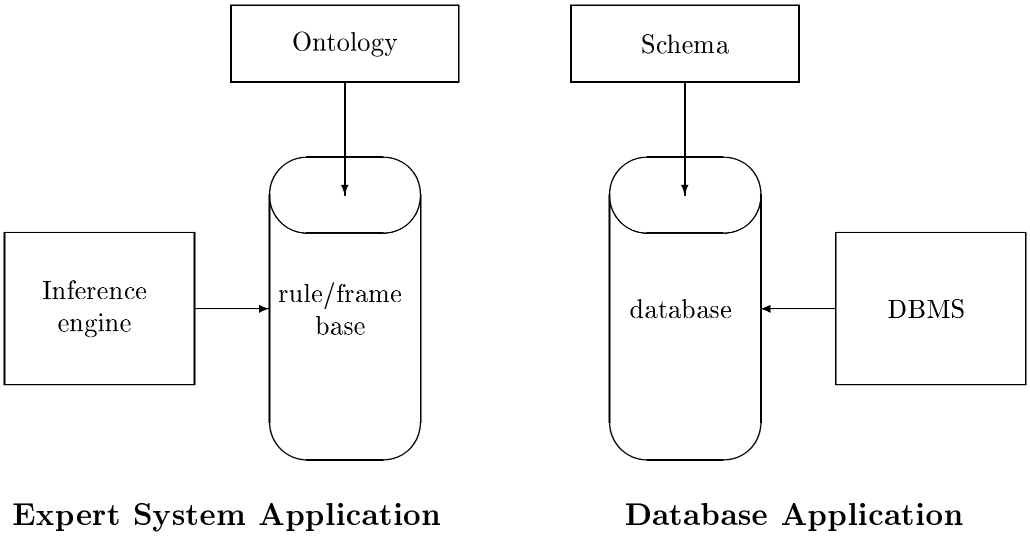

Figure 1: Parallels between structure/architecture of DB and ES applications

two technologies also exists at the conceptualisation level (especially where schemas are expressed in terms of rules). Note, however that there is a distinction between the concepts of ontology, i.e. an explicit conceptualisation of the domain, and schema, which describes the structures of the database itself. Recently, ontologies have also been introduced for DBSs as a conceptual layer above the schema. An advantage is that, by mapping elements of schemas to concepts within an ontology, the semantics of the database are made explicit and can be used as a basis for integration of schemas, for example to form multidatabase systems (Masood and Eaglestone [53]).

3.2 Relational Databases

The relational data model [22] represents data as tables of atomic values (e.g. Table 1) and defines an algebra for set-oriented associative manipulations. It specifies two general integrity rules: entity integrity prohibits null-valued keys, ensuring that all data objects include an identifying name; and referential integrity prohibits foreign keys values which do not also occur as primary keys, thus avoiding *dangling references*.

Relational databases [30] are the state of the art and benefit from the widely accepted APIs, ODBC and JDBC, and an international standard language, SQL, which provide the functionality of the relational algebra in a friendly executable form. To interact with a relational database we can therefore formulate expressions in the relational algebra as queries such as the following SQL query:

```
SELECT a.film b.director
FROM whatsOn a, films b
WHERE a.cinema = 'odean' and a.film = b.name
```

Though the third generation of database technology (object and object-relational databases) is emerging, it has yet to achieve the stability, reliability and performance to displace relational systems. A relational database can also viewed from an ES perspective as a special case of clausal form logic in which the tuple is regarded as a simple clause with constant arguments. Consider the example relational database given in Table 1 where we have three tables (relations), films, locations and whatsOn. This could be implemented as a series of PROLOG predicates or facts

films		
name	director	actor
the_trouble_with_harry	hitchcock	gwenn
the_trouble_with_harry	hitchcock	forsythe
the_trouble_with_harry	hitchcock	macLaine
cries_and_wispers	bergman	anderson

...

locations		
cinema	address	telephone
odean	lime_street	123456
philharmonic	hope_street	654321

...

whatsOn		
cinema	film	start_time
odean	cries_and_wispers	20.30
philharmonic	the_trouble_with_harry	20.15

...

Table 1: Example relational database tables

(Table 2). In Table 2 groups of facts are gathered together to form a "table" (in the relational database sense), using a unifying predicate (relation) name. Each line in the PROLOG fact base is then synonymous with a record or tuple in a relational database table.

The above example SQL query can therefore be expressed in logic, using PROLOG syntax, as follows:

```
select(Film, Director):-
    whatsOn(odean,Film,_),
    films(Film,Director,_).
```

Note that, as with relational Database Management Systems (DBMSs), expert systems environments and programming languages, such as PROLOG, typically also allow for the addition and deletion of facts. However, not all aspects of relational databases map neatly into clausal

```
films(the_trouble_with_harry,hitchcock,gwenn).
films(the_trouble_with_harry,hitchcock,forsythe).
films(the_trouble_with_harry,hitchcock,macLaine).
films(cries_and_wispers,bergman,anderson).
...

locations(odean,lime_street,123456).
locations(philharmonic,hope_street,654321).
...

whatsOn(odean,cries_and_wispers,20.30).
whatsOn(philharmonic,the_trouble_with_harry,20.15).
...
```

Table 2: PROLOG fact base

form logic. One particular problem is that the relational model is founded on the Closed World Assumption (CWA) - if a relationship is not known to be true then it is assumed to be false. This is not supported by pure logic, however CWA does correspond closely to the way in which languages such as PROLOG handle negation (i.e. negation by failure).

3.2.1 Note on object and object-relational databases

The emerging third generation (3G) of DB technology is predominantly object and object-relational. Respective de facto standards, i.e. ODMG [18] and the SQL3 standard proposals [4], can be viewed as object-oriented programming languages with added database facilities and relational databases systems with added object-oriented features. Both are technology (rather than theory) driven and have emerged in response to the realisation that the application of relational systems is impeded by a limited type system. Speculative requirements for these technologies were set out at the turn of the decade in the object-oriented and third generation database manifestos [5, 63]. 3G data models enrich structural semantics of DBs by supporting complex objects. Also they add behavioural semantics through encapsulation. The latter allows users to extend the type system, e.g. to represent non-textual and non-numeric objects.

3.3 Deductive Databases (DDB)

Deductive database systems provide mechanisms for managing knowledge as well as data. They are a natural extension to the relational model. As such deductive databases are able to derive new facts using existing information explicitly stored in the database [57]. This is achieved by generalising the type of information that may be stored in the databases. In addition to simple facts it is also possible to store expert system style rule sets. This then produces a database system with similar properties to that of a logic programming environment. For example, with respect to the PROLOG "relational database" presented in Table 2, we could store a rule of the form:

```
select(Cinema,Director,Film):-
    whatsOn(Cinema,Film,_),
    films(Film,Director,_).
```

which can be used in a number of different ways according to the manner in which the arguments are *instantiated*:

1. Return all films and their associated director currently being shown at all cinemas contained in the database (e.g. `select(X, Y, Z)`).

2. Return all the films, and the associated directors, currently being shown at a particular cinema (e.g. `select(odean, Y, Z)`).

3. Return all the films, and cinemas where they are showing, given a particular director (e.g. `select(X, bergman, Z)`).

4. Return all the cinemas where a particular film is showing and the director of the given film (e.g. `select(X, Y, cries_and_wispers)`).

5. Return all films by a particular director showing at a particular cinema (e.g. `select(odean, bergman, Z)`).

6. Return the name of the director of a particular film showing at a particular cinema (e.g. `select(odean, Y, cries_and_wispers)`).

7. Return all the cinemas where a particular film with a particular director is showing (e.g. `select(X, bergman, cries_and_wispers)`).

8. Confirm that a particular cinema is showing a given film with a given director (e.g. `select (odean, bergman, cries_and_wispers)`).

The distinction between *expert systems* and *expert database systems* is that the primary concern of the latter is the management of large amounts of data with the manipulation of that data as secondary [44]. Expert systems, on the other hand, are more concerned with the manipulation of "smaller" amounts of data and more complex amounts of knowledge, with the emphasis on manipulating that data in a manner that reflects the mode of working of a domain expert. The distinction is a fine one. The ratio of facts to rules may be a useful indicator to distinguishing between an expert system and an expert database - an expert systems will (generally) contain many more rules than facts while an expert database system will contain many more facts than rules.

3.4 Active Databases

Active databases support the automatic triggering of updates in response to internal or external events. This is generally achieved in an expert system" like manner where forward chaining of rules is used to accomplish the update. Rules typically follow the ECA (Event-Condition-Action) format of:

```
on <event> if <condition> then <action>
```

The similarity between this and expert system style rules, especially in the case of production rule systems, is self evident. The distinction is that in a rule base the triggering event is user supplied rather than expressly included. In a sophisticated active rule base the action part of the rule may entail calls to further rules in a manner identical to that supported by expert system style rule bases. The overlap here is so close that the rule set that forms a component of an active database is generally referred to as a rule base [1]. Active database behaviour has been applied to the relational model [62], the deductive model [70], and the Object Oriented model [42] of database systems.

3.5 Constraint Databases

Constraint databases [41] are an extension of research into CLP [40]. CLP combines the advantages of logic programming (declarative, semantics, nondeterminism and relational form) with efficient constraint solving by introducing richer data structures. As such it can be viewed as a generalisation of logic programming where unification (as used to support PROLOG) is replaced by constraint handling in a constraint system [26]. A particular advantage is considered to be that it supports "consistency techniques" [36], based on the concept of a-priori pruning, to perform intelligent searches of decision trees. The technique supports constraint propagation in such a way that it reduces a-priori the search space, thus limiting the computational time required. The basic idea of constraint databases is to replace the notion of a tuple in a relational databases by that of a generalised tuple, i.e. a conjunction of constraints. For example given a tuple:

$$(a,\ ...,\ an)$$

this can be regarded as a generalised tuple of the form:

$$(x1 = a1) \wedge ... \wedge (xn = an)$$

Of course this model also needs to be supported by an appropriate "constraint database" query language. This incorporation of CLP in a database model, where CLP can be viewed essentially as an extension to logic programming may be viewed as another area of overlap between the expert systems and database domains. The use of constraints is also recognised as an important tool for database integrity checking. In early work in this area constraints were expressed in terms of clausal logic. More recently much more sophisticated constructs have been adopted.

3.6 Discussion

DB and ES technologies can be viewed as complementary in that both are concerned with representation and utilisation of information. However, they are also distinct technologies. This distinction is not always clear. We see grey areas where an expert system ((mainly) all rules) fades to a deductive database (rules and (mainly) facts) and then fades to a DB ((mainly) all facts). However, a querying perspective provides better separation. DBs are primarily concerned with extensional querying, i.e. associative and navigational retrieval of stored facts. DDBs and ESs, on the other hand, are primarily concerned with intensional querying, i.e. application of the knowledge (rules) represented, within the context of the stored facts. Homogeneous representations of rules and facts are therefore used in ESs (and DDBs), commonly Horn clauses, whereas a DB has separate representation of data (the extension) and metadata (schemas). The differences in approach stem from the respective aims. ESs and DDBs are artificial intelligence (AI) applications and aim to provide "reasoning" and deductive capabilities in which stored knowledge is applied. DBs have a more modest aim, i.e. storage and retrieval of information stored as data. It is the DB application (which could be an ES) which then utilises the retrieved data, (e.g. for reasoning).

Grey areas are widening as the technologies borrow from each other to enhance functionality. This is occurring in mainstream database technology, for example, extensions to DB query languages to include intensional querying, such as the least fixed point facilities in object-relational database languages (e.g. SQL3's rule like transitive closure queries [4]). This cross fertilisation has produced a number of technologies which can be viewed as hybrid.

This growing overlap in features of DB, DDB and ESs suggests that the bringing together of solutions to VV&I related problems should be mutually beneficial. The following section clarifies the nature of the VV&I.

4 Validation, Verification and Integrity Issues

Validation and verification are terms more familiar within software engineering, whereas *integrity* is more widely viewed as a database issue. Validation and verification are often used as synonyms, but have distinct meanings. Verification tests that a product is constructed correctly (i.e. it meets stated requirements), whereas, validation checks that the product does what the user really wants. By maintaining the integrity of a product we ensure that it continues to be a "correct" product with respect to some specified constraints. For example, having formulated *integrity* constraints, we can *verify* that they are correctly enforced within the database system, and also *validate* that they correctly express semantics of the Universe of Discourse (UoD). The flavour of the distinction between the three terms is given concisely in the following definitions (the first two are from [10], to which we add the definition of integrity):

Validation: Are we building the right product?

Verification: Are we building the product right?

Integrity: Are we keeping the product right?

As with main stream software engineering VV&I of rule bases and databases has long been a concern in the ES and DB communities. However, work in these two areas has taken different perspectives on these issues. An overview of this work is presented in the following two Subsections.

4.1 The ES Perspective

For the ESs community, VV&I has been concerned mainly with getting the rules right prior to using the system. Accordingly the aim has been detecting and correcting intension errors and anomalies [25]. This is what we might expect, since knowledge, as distilled by observation and analysis from (past) data, should be stabilised and less prone to change. However, the situation is far from clear cut, as is shown by some of the problems with non monotonic logics and default reasoning where the notion of consequences due to updates in deductive databases have been extensively studied [55]. We also note that the formal semantics of both DDBs and ESs provide the mathematics for rigorous verification, whereas, much DB technology lacks a formal semantics. Early work on VV&I of rule bases originating at the commencement of the 1980s and now well established, sought to establish the nature of errors and anomalies that required detection. As a result many approaches and techniques were proposed to aid in the identification/prevention of such errors and anomalies ranging from design methodologies to V&V tool sets. Some of the proposed techniques and approaches have been adopted by industry, while others serve as markers in the chronology of rule base V&V research. Work is still continuing in this area. Further current work is also concerned with the refinement of rule bases, subsequent to the detection of errors, with the emphasis on tools to support automated or semi-automated refinement. It is also appropriate to note here that there have been a number of significant ESPRIT projects in the field - VIVA, VALID, VITAL [6]. Specific issues of VV&I for ES concern the errors and anomalies that can occur in rule bases, how they are detected, and what remedial action can be taken. These issues are each discussed below.

4.1.1 Errors and anomalies in rule bases

Broadly a rule base can be "wrong" in two ways, it may be structurally flawed or logically flawed. This is a distinction akin to the difference in traditional software engineering between the terms verification and validation [10] (given above). We can also make a distinction between structural errors and structural anomalies. Errors are clearly undesirable in that they will adversely effect the operation of a rule base; anomalies in contrast may not necessarily represent problems in themselves, but rather symptoms of genuine errors.

Much work has been done to establish and classify the nature of the structural errors and anomalies that may be present in rule bases (for example Ayel and Laurent [6] and Preece and Shinghal [58]). In the authors' view the most straight forward manner to classify structural errors/anomalies is to make a distinction between errors concerned with *inference* and those concerned with the *design* of the rule base.

Inference errors/anomalies are concerned with the connectivity of the rule base. A rule in a rule base is connected if it has at least one downward connection and one upward connection. If a rule has neither it is unconnected, if a rule has only upward connections it is a leaf rule and if it has only downward connections it is a root rule. A rule that has no upward or downward connections is sometimes referred to as a *redundant rule*, while a rule that is not a leaf rule but has no downward connection is sometimes referred to as a *dead end rule* (in forward chaining systems). Under the heading of connectivity we can also include auxiliary rules; these are defined as rules that have only one upward and one downward connection and therefore may (any desired isomorphism with the source data permitting) appropriately be subsumed into some other rule. Circularity, an urgent problem in some systems, is also considered to be an inference error.

Design errors/anomalies include subsumption, duplication and inconsistency. Subsumption is usually defined as the situation where one rule is a more specialised case of another. Thus if (for example) the antecedents (condition part) of two rules are identical except that one antecedent includes one or more additional propositions, and the consequents (action parts) are also identical, then subsumption exists. Duplication, where the antecedents and consequents of two rules are identical (except perhaps in the ordering of predicates) is then a specialised form of subsumption. Inconsistency then describes the situation where two (or more) rules result in contradictory consequents, i.e. where two rule antecedents are identical but their consequents are not co-tenable. Where the antecedents of two rules are such that subsumption may exist but the consequents are different subsumption and inconsistency are considered to exist together.

A similar classification of logic and design errors and anomalies can be attributed to the rule sets found in deductive and active databases, and relational databases expressed in clausal form. For example circularity has been identified as a problem in active database sets (see [2] or [42]). Although active databases display all the problems of experts system V&V, the issues are sharpened by imagining the consequences of a flawed distributed active database.

For a more detailed discussion concerning the errors and anomalies that may occur in rule bases interested readers are referred to [23]. It should also be recognised here that although V&V is an issue in DBMS, with respect to databases the greatest concern is to do with integrity of the data: the permitted types and ranges of items, possible conflicts and contradictions between fields and the inter-dependencies between tables and fields.

4.1.2 Error detection

There are two main approaches to identifying logical errors in rule bases. The first and most obvious approach is to exercise a set of appropriate test cases in a manner synonymous with traditional software testing. The operation of the rule base can then be said to be "correct" if the results produced compare favourably with those suggested by a domain expert. This approach has been adopted by many practitioners (e.g. [28] and [52], to give two examples). Work has also been done on the automatic generation of test cases ([20]). In the second approach information concerning integrity constraints (typically expressing incompatibilities between inputs and/or outputs) that are known to exist with respect to a particular domain are incorporated into the rule base and test cases generated which attempt to break these constraints. In this case a rule base is said to be correct if for all non-contradictory inputs no contradictory outputs result. Consistency checking techniques have been extensively studied (e.g. [48] and [33] to give two significant examples). Of course, as with traditional software testing, using either technique, the rule base can only be said to be correct in the sense that no test cases which produced wrong results or inconsistent results were found.

The risk of incorporating logical errors into rule bases can be significantly reduced by adopting appropriate software engineering techniques, especially during the conceptualisation of the rule base. To this end a number of expert system development environments/methodologies have been proposed of which the best known, and most widely used, is probably KADS ([68]). A number of formal expert system specification techniques have also been developed that result in an implementation independent (conceptual) model of the desired expert system which can be tested in isolation (see [34]). The perceived advantages are (1) that testing can take place early on in the development cycle and (2) that the nature of the testing is not cluttered with implementational detail ([32]).

Structural errors are generally detected through static inspection of the rule base. This is facilitated by the declarative logical formalism used to represent most rule bases. A great many tools and techniques have been proposed to identify errors and anomalies. These include techniques such as decision tables ([66]), incidence matrices ([24]), and rule base folding ([20]).

For further detail concerning error detection techniques interested readers are referred to text books such [6] or [23], or review papers such as ([67]).

It is suggested here that the nature of the techniques and approaches outlined above are equally applicable to the identified overlap between database and expert systems. Some work has been done in the field of active databases to identify infinite loops (*non-termination*) using an approach known as the *trigger graph method* ([2], see also [46]).

4.1.3 Remedial Action

Once an error or inconsistency has been detected in a rule base some corrective action is required. Typically this will involve the deletion, addition or modification of one or more rules. In addition, any remedial action, must be carried out in such a manner that the remainder of the rule base is not adversely effected. To this end it is desirable to refer to some revision strategy or plan to ensure that the corrective action is implemented in an appropriate manner (Bouali et al. [13] give a number of revision plans). A traditional software "debugging" element will also be involved.

There are two main approaches that may be adopted to repair a flawed rule base, (1) any necessary corrective action can be implemented "by hand" or (2) software tools can be used to automatically revise the rule base. The first is a consequence of the application of software tools that simply "flag" errors and anomalies. In many cases this is considered to be all that is required. Any necessary remedial action is then implemented by the creators of the system with appropriate reference to domain experts.

Much research has been directed at the automated refinement of rule bases (e.g. [28] and [13]) also sometimes referred to as rule base reduction ([33]). The question as to what extent this is desirable (or not) is still open to debate. For example in Ginsberg's rule base reduction approach subsumed rules are automatically identified and absorbed into appropriate existing rules. However, it may be that the existence of the subsumed rule was not simply a "left over" from an earlier version of the system, but an error in logic which would then require some alternative remedial action. Consequently it is generally acknowledged that rule bases cannot be entirely revised without the eventual intervention of a domain expert, however, it is also acknowledged that automated refinement/reduction of rule bases can aid the revision process. For example the KRUST system ([29]) has, as input, a single rule base and a set of training cases. If no fault is found using these cases the original rule base is returned, otherwise one or more refined rule bases are produced which must then be evaluated.

Similar considerations are applicable to deductive and active database rule sets and clausal form relational databases. The discussion as to whether automated remedial action is desirable is also significant, and warrants further investigation.

4.1.4 Summary

A summary of the expert system perspective is presented in Figure 2. The diagram is intended to show a break down of the viewpoint. The classification of the differing aspects is based on the experience of the authors and thus does not necessarily reflect the view of the entire expert system V&V community, indeed over the years there has been much discussion as to the classification of rule base errors. Much debate exists over what is an inference error and what is a design error. The classification given in Figure 2 is founded on that adopted in this paper which in turn is based on the nature of the algorithms required to identify the errors in question. The "boot lace" in Figure 2 is intended to indicate the "loose" connection between the different elements of V&V from the expert systems perspective.

4.2 The DB Perspective

The emphasis for the DB community on VV&I has been different to that in the ES community, as a consequence of the ongoing and changing nature of most databases. This volatility is in contrast to many ESs where little or no change is expected. Consequently, off the three terms, integrity is the most familiar in the DB world . In natural language, integrity is the property of honesty, uprightness or original perfect state (Collins Dictionary). In the DB world, it has a weaker meaning, i.e. plausibility. Integrity constraints are rules or conditions (Boolean-valued functions) that are required to be true, and thus limit the set of valid instances of a schema. They are defined to ensure that an instance is self-consistent, complete, and plausible with respect to known generic properties of the UoD.

VV&I issues are clearly fundamental to DBSs, since they concern establishing and maintaining the "correctness" of a DB, by ensuring that artifacts of a DBS collectively provide a plausible representation of the UoD and its behaviour.

Intensional correctness is also of concern for DBs and is addressed by methodologies for database design (e.g. [22]) and schema evolution management [43]. However, the focus has been on avoiding data errors and inconsistencies that can be introduced by data updates, thus ensuring that the time-varying extension of a database is maintained as a model of the UoD. The assumption implicit in this focus, i.e. of intension stability, is also disputable. Studies into the evolutionary nature [61] and stability [51] of database schemas show liability to frequent changes. This is particularly so for object databases which tend to target design applications which are by nature evolutionary, incremental, dynamic and experimental.

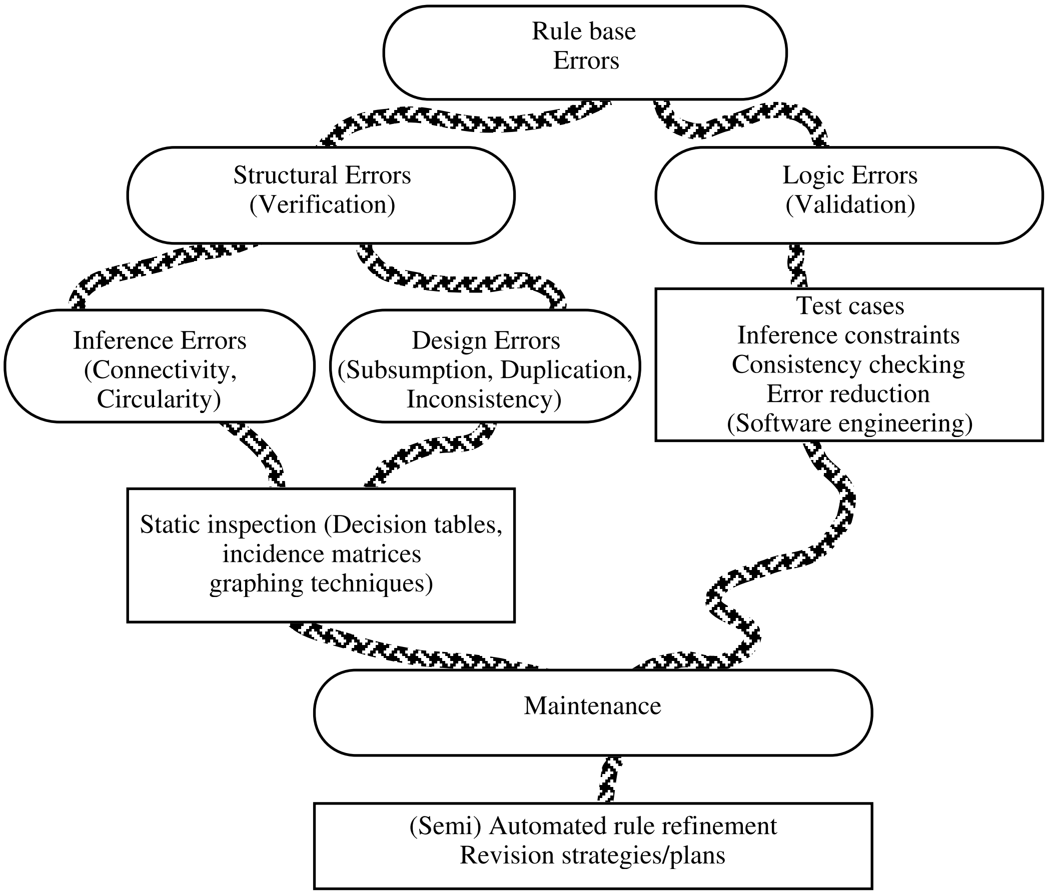

Figure 2: Summary of expert system V&V perspective

The varying nature of the VV&I database issues for the different generations of data models is discussed in the following subsections.

4.2.1 VV&I of relational databases

The formal definition of the relational data model has facilitated extensive research into issues of VV&I [1, 50]. In particular, the theory of relational databases includes a number of semantic features (domains, keys, dependencies between the columns (attributes) of tables (relations), etc.) which are a consequence of the meaning of the data represented, rather than the whims of the designer. Consequently, we can formulate properties of relational database which relate to the validity of the database, and which can be formally verified. Most of this theory has found little direct application in the design and operation of actual databases (e.g. the representation theory in [50] is of academic interest, but is not applied, and is in fact based upon the unrealistic assumption that relational DBs are projections from a universal relation). In fact, even the original definition of the basic relational data model is only imperfectly implemented in current relational DBMSs.

An exception is the theory of relational database design through normalisation, which is part of mainstream systems analysis and design methods. Normalisation allows verification of database designs through application of design constraints, called normal forms. Normal forms are formulated in terms of table keys and dependencies (functional, multivalues, join, template, etc.)

between columns. These constraints ensure the removal of redundant data, and thus avoid update anomalies. Update anomalies occur when an update creates inconsistent data or inadvertently removes data. Ironically, the normal forms are necessary only because of the lack of expressiveness of the relational model, which results from the underlying constraint that the values in tables are atomic (first normal form (1NF)). 1NF denies us the ability to directly model entities with complex structures and introduces potential design errors within the resulting collections of tables which are needed to collectively represent them. The formal specification of design constraints has also led to attempts to automate the design process (e.g. Bernstein's algorithm for synthesising 3NF [9]). However, the semantic features of the relational model provide only a poor representation of database semantics and can lead to theoretically sound designs which provide a poor representation of information as data (verified designs which have low validity!). In fact, Bernstein identified the need for a "semantic filter" to vet the validity of automatic deductions made using functional dependency theory, which amounts to human intervention.

Given the limitations of relational theory, validation of database designs relies on conventional software engineering techniques, such as "walk throughs" and testing with scenarios and test data sets. Improvements to the thoroughness of these techniques have filtered through from software engineering. For example, there have also been attempts to use formal specification languages and formal methods to support DB design verification (e.g. [35, 47]).

Maintenance of extensional database integrity is through application of the two general constraints, and also database specific constraints which can be formulated as Boolean functions, expressed in the relational algebra.

Maintenance of intensional integrity is through management of schema changes (schema evolution). Schema evolution is seen as a software engineering task which requires the usual software engineering validation and verification procedures. Other work concerns analysis of the impact of schema changes on the information capacity of a database system and shielding existing applications from the change, through schema modification and/or schema versioning.

However, in traditional relational systems support for domains is poor and typing is limited. Typing is provided by a number of built-in (basic) types (integer, float, character, etc.) and a single type constructor, the table (powerset of product), which can be applied to the built-in types just the once (1NF). This lack of support for complex structures (e.g. nested tables), and lack of an extensible type systems, has meant that many relational "integrity constraints" are simply supporting typing. This is particularly true for domain integrity constraints.

4.2.2 VV&I of object and object-relational databases

Validation issues for object and object-relational database design are again through software engineering techniques. Methodologies have been proposed to ensure validity of database designs, including those of Booch [11], Rumbaugh [60] and Jacobson [39] (design notations used in these methods have been integrated and rationalised into a single design language, the Unified Modeling Language (UML) [12]). However, lack of theoretical foundations impedes verification. There is no widely accepted formal semantics for 3G data models. In fact, there are aspects which are unsafe" and defy formalism (such as methods and their side effects).

Separate integrity constraint systems are superseded by generalised rule systems (as in Postgres [64] and active databases [70]) and by the ability to better express data semantics through a richer and extensible type system. With the superior typing of object (and object-relational) systems integrity constraint facilities are not as obvious as in relational system. In object DBSs they are not factored out into a separate category, "integrity". Instead, enforcement of data integrity is moved into the object's methods. However, this may open up a bigger problem of understanding and explaining DB behavioural semantics which are a consequence of individual methods and their interactions and side effects.

Ability to extend a type system introduces new problems by allowing representation of non-standard data objects. Content based integrity constraints are problematic because of the difficulty in inferring the semantics of images, sounds, etc. from their representations.

4.2.3 VV&I of legacy and multidatabase systems

VV&I issues for legacy systems [15] have come to the fore in recent years because of new possibilities for integrated exploitation of existing information sources through the internet and the WWW in particular, and through the use of multidatabase technologies [16].

The problems legacy systems present in the multidatabase context [27], are, firstly the reverse engineering problem of determining the integrity constraints built into the legacy systems, and secondly, how to derive a global mechanism within which the constraints of the component systems are reconciled and maintained.

4.3 Discussion

ESs have much in common with relational DBs (and DDBs). They can be viewed as being complementary, and are often used in conjunction. For example, relational DBs are used as backend fact stores by DDBs. This interoperability is possible because logic provides a unifying theory; rows (tuples) of a table (relation) can be viewed as facts which can be represented as Horn clauses. At a more general level, all three technologies are concerned with efficient storage and optimised querying of possibly large numbers of facts.

Similarities between relational DBs and ESs (and DDBs), i.e. common theoretical underpinning and functionality, suggests VV&I solutions should be transferable between the technologies. However, we find that there are fundamental differences in the nature of the VV&I problem as perceived by the different communities.

An advantage of DDBs and ESs from the VV&I point of view is the uniform and consistent treatment of all sorts of rules, whether default rules or integrity constraints. Integrity constraint research in DDBs has a long history (e.g. [45, 49]), but much of this has been aimed at efficient checking (i.e. limiting the search space) and is part of the general work on query optimisation in DDBs, which remains a problem area still. The other aspect of integrity checking is its use for informative query answering [38].

Within the DB community the focus has been on maintaining plausibility of facts represented as a model of the UoD. One insight that has come out of the DDB work on integrity is formalising the fact that integrity checking a DB is usually not about checking the DB itself, but about ensuring that changes to the DB do not destroy its (assumed) consistency. This has some implications for the introduction of integrity constraints into already built DBs. This also highlights the fact that integrity checking is more complex in DDBs, where many "facts" are implicit, i.e. generated from rules, and these facts need checking too.

DBs have a major advantage over ESs and DDBs for maintaining the validity and integrity of the stored facts. DB elements are typed (in Prolog and Datalog variables are untyped). A type provides a validity and integrity mechanism by acting as a "suit of armour" which stops data values from being accessed or manipulated in ways which are inconsistent with their meaning [17]. Though there are typed and OO logic languages (e.g. [3, 37]), those additions to logic languages move them away from their pure logic basis and that basis in logic may be a reason for their attractiveness i.e. the ability to "reason".

Another area where one might expect application of DB theories for ESs (and DDBs) is design theory. Normalisation is a formal approach to the refinement of relational DB design. (by ensuring no unnecessary replication of data). Normalisation, together with representational theory [50], provides a theory of "goodness of relational DB design". Again one would expect this to improve our ability to design good representations of facts within ESs (and DDBs). However, that is not the case, since normalisation is necessary for relational databases only because of the 1NF constraint which prohibits direct representations of complex objects. In DDB and ESs, complex structures can be modeled. For example, a list can be accommodated in a DDB fact (although physical storage may be harder if a relational DB is actually used for the fact storage).

This discussion has focused on apparent overlaps between the ES world and that of DBs, with regard to VV&I issues. However, on close inspection the two worlds have proved to be largely disjoint. Finally, we identify two DB problems that may be amenable to AI solutions also utilised in the ES world. The extensible type systems of modern DBSs make it possible to represent nonstandard types of data object, such as media object. Content-based querying for these is an open problem. For example, what algorithm would one use to retrieve images in which a dog featured? Media objects are large and inscrutable. Where there is no known deterministic algorithm, or the problem is believed to be intractable because of complexity, we have to revert to heuristics or other AI approaches, such as machine learning. For example, there is a growing body of work on the application of NNs to DB problems, such as automatic object classification [54].

A second area where AI solutions may prove applicable is where facts represented in a DB are insufficient to solve the problems at hand. In cases where querying is on the basis of incomplete or uncertain facts, AI techniques also used in ESs , such as Baysian reasoning may be appropriate.

Finally, we note that expertise is required by DB administrators and users, and perhaps one of the most useful VV&I related roles for ESs is in providing that expertise, particularly to improve the quality of usage of DB technology by the non-skilled operator.

5 Conclusions

In this paper the authors have attempted firstly to identify the area of overlap between the database and expert systems communities, and secondly to present a view of the "state of the art" of rule base and of database VV&I with a view to their application to the identified overlap.

Although some overlap between expert and database systems has been identified, there may be further areas where the experience of expert systems VV&I may be applicable to database systems and vice versa. For example, we have identified two VV&I areas where the AI techniques utilised by ESs are potentially applicable to DBs. These are integrity constraints for certain large and inscrutable types, such as media objects, and support for querying incomplete and uncertain information. This is a matter for further discussion.

Given the identified overlap between the expert systems and database communities, as indicated in this paper, it is suggested that the VV&I techniques established by the former may have some application within the database community. The possible nature of this applicability is as yet unclear. Also, the former may benefit from the latter by virtue of the interoperability of the systems, for example, where a DB is used by an ES as a fact store. Finally, we speculate that ESs potentially facilitate wider and better exploitation of DB technology by providing to non-expert DB designers and users the expertise currently used by the DB specialists.

It is also worth noting that our literature search for "verification and database, validation and database", and "integrity and database" produced almost no overlap. The latter was the largest category. Also, "validation and database" was big but not in the computing area, indicating that DB validation is seen mainly as an application specific activity, rather than a generic DB service.

References

[1] Abiteboul, S., Hull, R. and Vianu, V. (1995). Foundations of Databases. Addison Wesley, Reading, MA.

[2] Aiken, A., Widom, J. and Hellerstein (1992). Behaviour of Database Production Rules: Termination, Confluence and Observable Determinism. Proceedings ACM SIGMOD, International conference on the Management of Data, pp59-68.

[3] Alexiev, V. (1993). Bibliography on Integrating Object-Oriented and Logic Programming. ftp://ftp.cs.ualberta.ca/pub/oolog/oolog-bib.ps.gz.

[4] ANSI, (1998). SQL3 drafts and discussion documents can be found at ftp://speckle.mcsl.nist.gov/isowg3/dbl.

[5] Atkinson, M.P., et al., (1990). The Object-Oriented Database System Manifesto, in Kim, W., and Nicolas, J-M., and Nishio, S. eds., Deductive and Object-Oriented Databases, pp 223-239.

[6] Ayel, M. and Laurent, J.P. (1991). Validation, Verification and Testing of Knowledge Based Systems. John Wiley and Sons, England.

[7] Batini, C., Ceri, S. and Navathe, B. (1992). Conceptual Database Design. Benjamin-Cummings, Redwood, CA.

[8] Bench-Capon, T.J.M. (1990). Knowledge Representation: an Approach to Artificial Intelligence. Academic Press, London.

[9] Bernstein, P.A. (1976). Synthesising Third Normal Form Relations from Functional Dependencies. ACM Transactions on Database Systems, 1(4), pp 227-370.

[10] Bohem, B.W. (1981). Software Engineering Economics. Prentice-Hall, New York.

[11] Booch, G. (1994). Object-Oriented Analysis and Design with Applications, Second edition. Benjamin Cummings, Redwood.

[12] Booch, G., et al. (1996). UML Resource Center, http://www.rational.com/uml/index.html.

[13] Bouali, F., Loiseau, S. and Rousset, M-C (1997). Verification and Revision of Rule Bases. In Hunt, J. and Miles, R. (Eds.), Research and Development in Expert System XIV, SGES publications, pp253-264.

[14] Bratko, I. (1986). Prolog Programming for Artificial Intelligence. Addison-Wesley, Wokingham, England.

[15] Brodie, M. and Stonebraker, M. (1995). Migrating Legacy Systems: Gateways, Interfaces, and the Incremental Approach. Morgan Kaufmann.

[16] Bukhres, O.A. and Elmagarmid, A.K. (eds) (1996). Object-Oriented Multidatabase Systems: A Solution for Advanced Applications. Prentice Hall.

[17] Cardelli, L. and Wegner, P. (1985). On Understanding Types, Data Abstraction, and Polymorphism. ACM Computer Surveys, 17(4) pp 471-522.

[18] Cattell, R. and Barry, D.K. (eds) (1997). The Object Database Standard: ODMG 2.0. Morgan Kaufman, San Fransisco.

[19] Ceri, S., Gottlob, G. and Tanca, L. (1989). What You Wanted to Know About Datalog (And Never Dared to Ask). IEEE Transactions on Knowledge and Data Engineering, 1(1).

[20] Chang, C.L., Combs, J.B. and Stachowitz, R.A. (1990). A Report on the Expert Systems Validation Associate (EVA). Expert Systems with Applications, Vol 1, No 3, pp219-230.

[21] Clocksin, W.F. and Mellish, C.S. (1984). Programming in Prolog, 2nd Edition.

[22] Codd, E.F. (1970). A Relational Model for Large Shared Data Banks, Communications of the ACM,13(6), pp 377-387.

[23] Coenen, F. and Bench-Capon, T. (1993). Maintenance of Knowledge-Based Systems: Theory, Techniques and Tools. Academic Press, London.

[24] Coenen, F.P. (1995). An Advanced Binary Encoded Matrix Representation for Rule Base Verification. Journal of Knowledge-Based Systems, Vol 8, No 4, pp201-210.

[25] Coenen, F. (1998). Verification and Validation Issues in Expert and Database Systems: The Expert Systems Perspective. In Wagner, R.R. (Ed.), Database and Expert Systems Applications, Workshop proceedings DEXA'98 (Vienna), IEEE, pp16-21.

[26] Colmerauer, A. (1987). Opening the Prolog III Universe: a New Generation of Prolog Promises some Powerful Capabilities. BYTE, pp 177-182, Aug 1987.

[27] Conrad, S.,Eaglestone, B., Hasselbring, W., et al. (1997). Research Issues in Federated Database Systems: Report of EFDBS'97 Workshop, SIGMOD, December, 1997.

[28] Craw, S. and Sleeman, D. (1990). Automating the Refinement of KBS. Proceedings ECAI'90.

[29] Craw, S. (1996). Refinement Complements Verification and Validation. International Journal of Human Computer Studies, Vol 44, No 2, pp245-256.

[30] Eaglestone, B. (1991). Relational Databases. Stanley Thornes.

[31] Eaglestone, B. and Ridley, M. (1998). Object Databases: A Introduction. McGraw Hill.

[32] Fensel, D. (1995). Formal Specification Languages in Knowledge and Software Engineering. The Knowledge Engineering Review, Vol 10, No 4.

[33] Ginsberg, A. (1988). Knowledge Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy. Proceedings AAAI'88.

[34] van Harmelen, F. and Aben, M. (1996). Structure-preserving Specification Languages for Knowledge Based Systems. Journal of Human Computer Studies, No 44, pp187-212.

[35] Hennessey, P., Ibrahim, M.T. and Fedorec, A.M. (1996). Formal Specification, Object Oriented Design and Implementation of an Ephemeral Logger for Database Systems. DEXA'98, Lecture Notes in Computer Science 1134, Springer, pp 333-355.

[36] van Hentenryck, P. (1989). Constraint satisfaction in logic Programming. MIT Press.

[37] Hill, P.M. and Lloyd, J.W. (1994). The Godel Programming Language. MIT Press.

[38] Imielinski, T. (1988). Intelligent Query Answering in Rule Based Systems. Foundations of Deductive Databases and Logic Programming. Minker, J. (ed). Morgan Kaufmann, pp 275-312.

[39] Jacobson, I.M. (1992). Object-Oriented Software Engineering. Addison Wesley, Reading.

[40] Jaffer, J. and Lassez, J.-L. (1987). Constraint Logic Programming. In Proceedings of the 14th ACM Conference on Principles of Programming Languages (POPL), Munich.

[41] Kanellakis, P.C., Kuper, G.M. and Revesz, P.Z. (1995). Constraint query languages. Journal of Computer System Science, Vol 51, No 1, pp26-52.

[42] Karadimce, A.P. and Urban, S.D. (1996). Refined Trigger Graphs: A Logic-Based Approach to Termination Analysis in an Active Object-Oriented Database Management Systems. ICDE'96 pp384-391.

[43] Kaushal, R. and Eaglestone, B. (1998). View-Based support for Transparent Schema Evolution in Federated Database Systems. Workshop on Issues and Applications of Database Technology. Part of 3rd Biennial World Conference on Integrated Design & Process Technology, (IDPT'98).

[44] Kerschberg, L. (Ed.) (1986). Expert Database Systems: Proceedings from the First International Conference. Benjamin-Cummings.

[45] Kowalski, R., Sadri, F. and Soper, P. (1987). Integrity Checking in Deductive Databases. Proceedings of 13th VLDB Conf, Brighton, Morgan Kaufman, pp 61-69.

[46] Lee, S.Y and Ling, T.W. (1997). Refined Termination Decision in Active Databases. In Hameurlain, A. and Tjoa, A.M. (Eds), Database and Expert Systems Applications, (Proceedings DEXA'97), Lecture Notes in Computer Science 1308, Springer Verlag, pp182-191.

[47] Leonard, M. (1992). Database Design Theory. Macmillan Computer Science Series.

[48] Levy, A.Y. and Rousset, M-C (1996). Verification of Knowledge Bases Using Containment Checking. Proceedings of AAAI'96.

[49] Lloyd, J.W., Sonenberg, E.A. and Toper, R.W. (1987). Integrity Constraint Checking in Stratified Databases. Journal of Logic Programming, 4(4), pp 331-343.

[50] Maier, D. (1983). The Theory of Relational Databases. Pitman Science Press, Maryland.

[51] Marche, S. (1993). Measuring the stability of data models. European Journal of Information Systems, 2(1).

[52] Meseguer, P. (1993). Expert System Validation Through Knowledge Base Refinement. Proceedings IJCAI-93.

[53] Masood, N. and Eaglestone, B. (1998). Semantics Based Schema Integration. In Quirchmayer, G., Schweighofer, E. and Bench-Capon, T.J.M. (Eds), Database and Expert Systems Applications, proceedings DEXA'98 (Vienna), Lecture Notes in Computer Science 1460, Springer Verlag, pp80-89.

[54] Millns, I. and Eaglestone, B. (1998). An Integrity Constraint for Database Systems Containing Embedded Neural Networks. In Wagner, R.R. (Ed.), Database and Expert Systems Applications, Workshop proceedings DEXA'98 (Vienna), IEEE, pp56-63.

[55] Minker, J. (1993). An Overview of Nonmonotonic Reasoning and Logic Programming. Journal of Logic Programming, 17(2-4), pp 95-126.

[56] Newell, A. (1982). The knowledge level. Artificial Intelligence, 18, pp87-127.

[57] Paton, N., Cooper, R., Williams, H. and Trinder, P. (1996). Database Programming Languages. Prentice-Hall.

[58] Preece, A.D. and Shinghal, R. (1994). Foundations and applications of Knowledge Base Verification. International Journal of Intelligent Systems, Vol 9, pp683-701.

[59] Robinson, J.A. (1965). A Machine-Oriented Logic Based on the Principle of Resolution. Journal of the ACM, Vol 12, pp23-41.

[60] Rumbaugh, J., Premerlani, W. and Eddy, F. (1991). Object-Oriented Modeling and Design. Prentice-Hall, Englewood.

[61] Sjøderg, D. (1993). Quantifying schema evolution. Information and Software Technology, 35(1).

[62] Stonebraker, M., Rowe, L.A. and Hirohama, M. (1990). *The Implementation of POSTGRES.* IEEE Trans. Knowledge and Data Engineering, Vol 2, No 1, pp125-141

[63] Stonebraker, M., et al. (1990). The Committee for Advanced DBMS Function. Third Generation Database System Manifesto, SIGMOD Record, 19(3), 1990.

[64] Stonebraker, M., and Kemnitz, G., (1991). The Postgres Next Generation Database Management System. Communications of the ACM, 34(10), pp 78-92.

[65] Heijst, G. van (1995). The Role of Ontologies in Knowledge Engineering, Doctoral Thesis, University of Amsterdam, Amsterdam, The Netherlands.

[66] Vanthienen, J. and Wijsen, J. (1995). On the decomposition of Tabular Knowledge-Based System. The New Review of Applied Expert Systems, pp77-89.

[67] Vermesan, A.I. and Bench-Capon, T. (1995). Techniques for the Verification and Validation of Knowledge-Based Systems: A Survey Based on the Symbol/Knowledge Level Distinction. Software Testing, Verification and Reliability, Vol 5 no 4, 1995, pp233-72.

[68] Wielinga, B.J., Schreiber, A.T. and Breuker, J.A. (1992). KADS: A Modelling Approach to Knowledge Engineering. Knowledge Acquisition (Special Issue: The KADS approach to knowledge engineering), Vol 4, No 1, pp5-54.

[69] Winston, P.H. (1992). Artificial Intelligence, 3rd Edition. Addison-Wesley.

[70] Zaniola, C. (1994). A Unified Semantics for Active and Deductive Databases. In Paton, N.W. and Williams, M.H. (Eds), Rules in Database Systems, Springer-Verlag, pp 271-287.