15

Managing Constraint Violations in Administrative Information Systems

Isabelle BOYDENS
SMALS-MVM Research
102 Rue du Prince Royal
1050 Brussels, Belgium
isabelle.boydens@smals-mvm.be

Alain PIROTTE
Université catholique de Louvain
IAG, 1 place des Doyens
1348 Louvain-la-Neuve, Belgium
pirotte@info.ucl.ac.be

Esteban ZIMANYI
Ecole Polytechnique Fédérale de Lausanne
Laboratoire de Bases de Données, IN-Ecublens
1015 Lausanne, Switzerland
Esteban.Zimanyi@epfl.ch

Abstract

This paper motivates a comprehensive methodological framework for dealing with some aspects of real-world complexity in information system analysis and design. By complex application problem, we mean a problem that cannot be solved by the current technology in the way that it is perceived and analyzed by application domain specialists. The paper focuses on a motivating case study, the analysis of constraint violations in database management at the Belgian agency for social security. We then re-interpret practices and their problems in terms of current information system technology. Recommendations are derived both for suitable developments of the technology, that would allow a better treatment of complex real-world problems, and for methodological improvements in data management practices in the application domain, that would take better advantage of the current technology.

Keywords

Information systems, complex application domains, research methodology, integrity constraints, management of inconsistency, data quality

1 INTRODUCTION

In spite of the continuous and spectacular progresses of computer technology, information systems, even moderately complex ones, often answer less than perfectly the needs that they are supposed to satisfy. Often, computing tech-

Data Mining and Reverse Engineering S. Spaccapietra & F. Maryanski (Eds.)

nology gets the blame ("computers did it again") but, of course, in most cases, the real problem comes from human errors or from methodological inadequacies, as in the recent failure of the Ariane 5 rocket* or the growing worries about the "Year 2000" problems, for example.

Software engineering is the discipline concerned with the practical problems of developing large software systems through a collection of methodological processes. The phrase "software engineering" was coined precisely to foster more appropriate methodological research. The area has become one of the most active in computer science and engineering.

Still, the complexity contemplated for modern information system applications has consistently kept pace with very real productivity gains, obtained mostly from the ever-increasing performance of computer technology. Thus, developers of complex real-world applications have kept being faced with problems whose solutions exceed the capabilities of current technology, when the application requirements are analyzed with all the relevant detail.

Another source of inadequacy of computer solutions lies in the traditional underestimation of the difficulties, hence of the cost, of application development. Often, real-world complexity is insufficiently analyzed in the early stages of system development. Problems only surface with the operational systems, which then require costly adjustments. The abstraction process of best selecting, in a complex real world, what is relevant to the needs of an application will remain creative and difficult. The more carefully this modeling transition is performed, the more adequate and efficient the corresponding computer solutions will be.

The idea for the methodological framework proposed in this paper originated from a study of information management practices at the Belgian social security agency (ONSS–RSZ) (Boydens 1992). Our subsequent analysis revealed much more complexity than anticipated, to the point that we concluded that that complexity could not be handled directly by current computer technology.

The paper is structured as follows. Section 2 introduces data management at the ONSS–RSZ. The pragmatic complexity in the application domain leads to accept data that do not comply with the database schema. Violations of database constraints are treated as "anomalies" by social security practitioners and handled in an adhoc manner. The basic idea of our work is that those anomalies should be carefully analyzed in terms of the current information system technology and of some of its plausible extensions. Section 3 re-analyzes constraint violations as the introduction into the database of data complying with a database schema weaker than the "ideal" schema implementing the rules of operation of the agency. Then the process of "correcting the faulty data" is presented as an update process through which data are progressively led to comply with the "ideal" schema. Section 5 is a short survey of research

*See our reading of the official report at the URL http://yeroos.qant.ucl.ac.be/dummies/ariane.html.

work about inconsistency management in knowledge-based systems. Finally, Section 6 summarizes the paper and suggests avenues for further validating our methodological proposals.

2 CONSTRAINT MANAGEMENT IN PRACTICE

2.1 Data Management at the ONSS–RSZ

The ONSS–RSZ (Office National de Sécurité Sociale – Rijksdienst voor Sociale Zekerheid) is the Belgian social security agency. The ONSS–RSZ collects, stores, processes, and redistributes social security contributions (more than a thousand billion Belgian francs each year from about three million workers and two hundred thousand employers) as, e.g., unemployment and retirement benefits (Boydens 1992, Boydens 1995).

The social security agency wishes to collect contributions and redistribute benefits as quickly as possible. Practically, the agency cannot compel two hundred thousand employers to supply correct information only. Thus, to operate within a reasonable time frame, it must tolerate imperfect (i.e., missing or erroneous) data in the database.

If anomalies in an entry document do not exceed a specified level and if no prohibited violation of constraint is detected (like, e.g., an erroneous amount of social security tax), then the document is integrated into the database, with the idea that it will be corrected later.

Still, it may be very difficult to detect, and a fortiori to correct, erroneous administrative data. Some anomalies are detected by control programs, while others are communicated informally by the individuals whose data are faulty. Some data cannot be corrected automatically, even if they are clearly erroneous, because of their juridical "evidential value". For instance, erroneous data contained in an original document signed by an employer may be integrated in the database and only corrected when a new original is received. Section 2.4 further analyzes types of constraint violations at the social security agency.

2.2 Database Constraints

A database stores information about some part of an application domain in the real world. A model of that part of the real world is represented as the database schema through the database design process. The better the schema agrees with the features of the real world relevant to application needs, the more adequate and efficient the resulting information system will be. However, the popular data models (typically, relational and entity-relationship), which can be viewed as languages for expressing database schemas, were defined as a

compromise, among very different stakeholders, between power of expression, ease of use, and facility of implementation. Weighed against the complexity of applications like social security management, the modeling power of those popular data models is clearly insufficient.

Thus, to better model application domains, the data structure part of database schemas (e.g., relations in the relational model, entities and relationships in the entity-relationship model) has to be supplemented with *integrity constraints*, also called *consistency constraints*, that are prescriptions (or assertions) that tighten the semantics of those data structures. Like data structures, constraints express general (i.e., type-level) semantics of the application domain and they constrain the allowed extensions of a database. Operationally, constraints control updates to the database so that the updated database conforms to the schema. Operational versions of constraints must thus be produced either manually, as part of the application programs, or automatically, as a function of database management systems.

Thus, constraints have a declarative version, as part of or supplementing the database schema, and an operational version, controling database updates. Still, constraints are fundamentally properties of the data, rather than of application programs.

A distinction is sometimes made between inherent, implicit, and explicit constraints. *Inherent constraints* are tightly linked to the data structures themselves: they express properties of the data structures and they need not be specified explicitly in the schema. In fact, they can also be viewed as part of the data structures. For example, in the entity-relationship model, it is inherent, that is, it holds without explicitly saying, that every relationship instance associates entities of specific entity types.

Implicit constraints are implied by the various specifications that accompany the data structure definition in the schema. For example, the mention, in a relational schema, that an attribute is a key in a relation implies the constraint that there will never be two distinct tuples in that relation with different values for that attribute. Another example of implicit constraint is the cardinality of relationships in the entity-relationship model.

Database management systems provide automatic support for some of the inherent and implicit constraints. For example, automatic support of the implicit constraints associated with relational referential integrity (an inherent constraint in entity-relationship models) was incorporated into the major commercial relational systems a few years ago. But many constraints are not automatically supported and they must be coded in the application programs.

Explicit or *adhoc constraints*, sometimes called *business rules*, are adhoc in the sense that they are application-dependent pieces of data semantics, that cannot be captured in the data structures and that are necessary for a faithful model of the application domain. Depending on their complexity, they are expressed declaratively in some specification language or procedurally, when they tightly affect some database transactions.

It is not always clear in the literature whether constraints are part of the schema or whether they supplement the schema, which, in that case, consists of just data structure definitions. We prefer the former presentation, as there is no difference in nature, from the point of view of the application domain, between information conveyed by the data structure part of the schema and information conveyed by constraints. In particular, constraints, like data structures, may be queried (Pirotte, Roelants and Zimányi 1991). From now on, unless otherwise specified, "schema" will be taken to mean data structures plus constraints.

Thus, in orthodox database management, updates violating constraints are prohibited as a matter of principle, just like updates not complying with the data structure definition are prohibited. A major goal of this paper was to try to reconcile database management principles with the frequent practice of tolerating errors and inconsistencies in administrative information systems.

2.3 A Simple Example

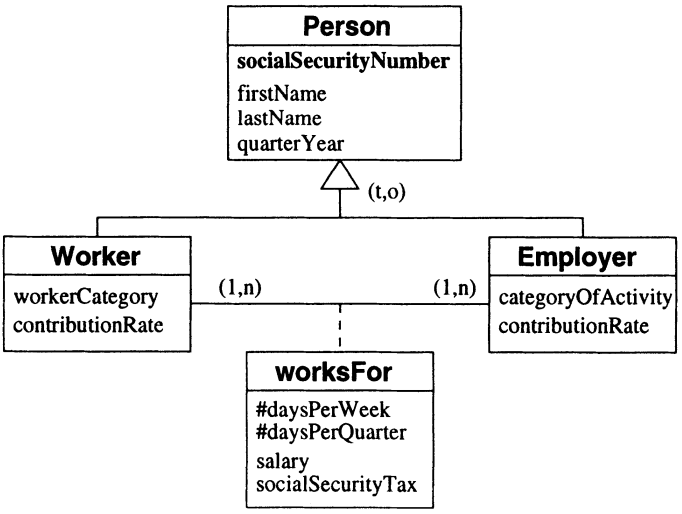

Figure 1 Simplified administrative information system.

The idealized example shown in Figure 1 will be used for illustrating some aspects of data management at the social security agency. A Person has attributes socialSecurityNumber (the identifier of Person), firstName, and lastName. The is-a generalization, total and overlapping, says that a Person is a Worker, an Employer, or both. A Worker has a workerCategory (e.g., workman, secretary) and a corresponding contributionRate. Similarly, an Employer has

a categoryOfActivity (e.g., building industry, retail distribution) and a corresponding contributionRate.

Data about the work of a Worker for an Employer during a quarter (quarterYear) are kept in an instance of the worksFor relationship. Attributes of worksFor comprise: #daysPerWeek (a multivalued attribute with the number of days worked by the Worker for the Employer for some or all weeks of the quarter), #daysPerQuarter (the number of days worked by the Worker for the Employer during the quarter, that is, the sum of the values in #daysPerWeek), salary (the accumulated salary paid by the Employer to the Worker for the quarter), and socialSecurityTax (the social security contribution due for that salary).

The data structures in Figure 1 are supplemented with the following constraints:

- IC_1: firstName and lastName of a Person must match the corresponding data in the national register (see later) for the same socialSecurityNumber
- IC_2: the contributionRate of an Employer cannot be null
- IC_3: the contributionRate of an Employer is determined by the categoryOfActivity (functional dependency)
- IC_4: the contributionRate of a Worker cannot be null
- IC_5: the contributionRate of a Worker is determined by the workerCategory (functional dependency)
- IC_6: for an instance of worksFor relating a Worker and an Employer, the number of days worked for the weeks in a quarter (the values in the multivalued attribute #daysPerWeek) must add up to #daysPerQuarter, the number of days worked during the quarter for the Employer
- IC_7: the number of days worked by a Worker during a quarter (the sum of the values of #daysPerQuarter for all the Employers for whom the Worker has done some work) must be less than or equal to 70
- IC_8: the socialSecurityTax is equal to the salary of the worker multiplied by the contributionRate of the Employer and by the contributionRate of the Worker

2.4 A Typology of Constraint Violations

In the simple example of Figure 1, constraints are of three types: some constraints (like IC_2, IC_4, IC_6, and IC_8) are meant to test correctness and consistency of data supplied to the agency by the employers; others (like IC_3 and IC_5) impose consistency between data supplied by the employers and data known by the agency; still others (like IC_7) express business rules from the application domain. This section proposes a broad classification of various

types of constraint violations that occur in the practice of social security data management.

Internal Inconsistencies. Data may be internally inconsistent, that is, inconsistent within the agency, in a single database or across several databases managed by the agency. When testing some data A for correctness against other data B, it may be difficult to determine which of A or B is safer, and the only sure conclusion may turn out to be the agreement or disagreement of A and B.

For example, if the category of activity declared by an employer does not match the category of activity already known by the agency for the same employer, then two contribution rates may be considered (a violation of IC_3). Correcting the violation requires further investigation, like a contact with the employer or more extensive comparisons with older data.

External Inconsistencies. Some data may be internally (that is, within the agency) consistent, but externally inconsistent in the scope of a federation of databases.

For example, data about a particular person may be consistent in the agency databases, but the internal data (i.e., social security number, first name, last name) may not match data in the national register, thus violating IC_1.

The national register is a repository for data on all people registered in Belgium. All government agencies refer to that information. Therefore, the national register plays the role of "master file" for the agency databases and other databases in the federation. This dependency bears some resemblance with referential integrity in relational databases.

Propagation of External Corrections. The national register itself also contains errors. But, while the social security agency is of course responsible for the data in its databases, it has no means of directly modifying the national register.

Each person is supposed to appear exactly once in the register, but some people appear two or more times, for example under their maiden name and under their married person name. In addition, some people do not appear at all in the register, for example, because they were not residents of Belgium when it was first established.

Errors are corrected little by little, when they are discovered by the people whose data are faulty or by the agency personnel. As the national register acts as master file, corrections must be propagated to all other databases referring to its information, a complex process that may uncover inconsistencies. Referring again to relational databases, this propagation resembles the updating of a relation key that appears as foreign key in other relations.

Semantic Ambiguities. Other errors result from erroneous interpretations of administrative policies, because of their complexity and their frequent changes. In their operational context, information systems perform two mappings, or transformations, of information: a *representation* or modeling transformation, that interprets information in the real world through a model of relevant aspects in the application domain, and an *interpretation* transformation, that supplies data back into the application domain, after processing in the information system. Thus data follow a path from outside or upstream of the information system, then they are processed within the information system, to be exploited outside or downstream of the system. A basic condition for the adequacy of the information system, or, in short, for data quality, is that these mappings ensure that the user view of the application domain be faithfully captured by the information system (Wand and Wang 1996).

- Upstream of the information system (that is, before going through the representation mapping), decisions made by legislators are translated into laws by lawyers. But there are sometimes semantic ambiguities in legislative definitions. For instance, in the Belgian social security law, the concept of "day of work" does not have a single meaning. From 1987 to 1990, for agencies in charge of tax collection, every day at work counted as one "day of work" as long as it was started, whatever the number of hours worked that day. On the contrary, for agencies handling the distribution of social benefits, a "day of work" necessarily had to comprise at least three hours worked for the same employer. Thus, people who worked fewer than three hours in a day for the same employer paid taxes that did not contribute to their benefits entitlement ...
- Within the information system (or, rather, systems), the concept of day of work is represented differently in the various administrative databases. Thus, different administrative data will correspond to the same reality.
- Downstream of the information system (that is, when data have gone through the interpretation mapping to be used in the application domain), for example for statistical processing, data from different databases are merged but, since a single concept, namely day of work, does not have a single meaning across different databases, such mergings may be inconsistent.
- Finally, statistical results (forecasts and previsions, for instance) serve as inputs to the preparation of new laws, which in turn will be reflected into the information system. Thus, a circular relationship has been created between output quality and input quality of the information.

Semantic Gaps. Some errors originate from contextual knowledge which would be very hard to formulate completely in the database schema.

For example, the official juridical nomenclatures for categories of activity described in legislative policies are not always complete. Then IC_2 is violated

for all employers whose category of activity is missing. One approach to this problem goes through creating fictitious codes for those employers.

Accuracy of Constraints. Constraints do not always adequately capture the semantics of the application domain, which leads to several problems.

- Exceptional but correct data may be treated as incorrect. For example, a worker may work weekdays and weekends in a quarter, so that the number of days worked during that quarter exceeds the number set by IC_7. Here, it is the constraint that is incorrect. The error results from the difficulty of delimiting exceptional situations during database design.
- Conversely, an erroneous data item may satisfy the constraints and still be accepted at input. For example, if an employer sends incorrect information about wages, all data derived from that information (e.g., social security tax) will be incorrect too, although this will go undetected as none of the constraints is violated.

The latter problem illustrates a fundamental limitation of the modeling process (Demolombe and Jones 1993). Constraints cannot be tight enough to catch all possible errors. More knowledge about the real world would be needed, as well as knowledge about the correspondence between the database and the real world. But the necessary amount of such knowledge is arbitrary large in general.

Incomplete Information. Anomalies commonly arise from missing data. If, for example, the attribute categoryOfActivity comes in with a null value, then IC_3 cannot be checked and it is impossible to compute the contribution for the employer. Further investigation (like a direct contact with the employer) is necessary for full information. A more drastic example of incomplete information is when an employer did not return tax forms for the due date.

Incompleteness can be difficult to detect on the database only. For example, if an employer did not register a worker for a particular quarter, while having registered the same worker for previous and subsequent quarters, this can either be an anomaly due to an omission or express the reality, namely that the worker did not work for the employer during that quarter. Here again, the problems are linked to fundamental limitations of the modeling process.

Correlation of Errors. Anomalies can be correlated for the same data, as one constraint violation (say, of IC_7) may entail a fictitious violation of other constraints (for instance, IC_6).

Federated databases, especially when some data are replicated in several databases, also contribute their share of complexity to the management of constraint violations. Logically interdependent data can be updated, syn-

chronously or asynchronously, by several agencies, each of them correcting data according to their specific competence. The management of transaction concurrency is a substantial problem.

For instance, the correctness of IC_2 may be managed by agency A, while IC_3 is treated by agency B, and IC_5 by agency C. But the correction of an IC_3 violation by agency B may entail new IC_2 violations to be corrected by agency A, while agency A may not know when the correction process by agency B has been completed.

In summary, data correction is a long and complex process, that takes place concurrently with the introduction of new data in the database. Data of different quality levels naturally coexist in administrative information systems. The correction process is diachronic, dynamic, and discontinuous. The data exploitation process, on the contrary, is synchronous and static. The quality of the output data, through the interpretation mapping, is therefore difficult to assess precisely.

2.5 Handling Inconsistency in Practice

In administrative information systems, inconsistent and incomplete information is typically managed (i.e., detected, then corrected and validated) through a process that schematically comprises the following steps (DOS 1992):

1. a distinction is made between constraints allowed to be violated (e.g., an inconsistency between a total amount of days worked and a total amount of hours worked) and those that are not (e.g., a faulty amount of social security tax); also, some constraint violations are tolerated only to a certain degree (e.g., a rate of allowed anomalies per record is set);
2. types of constraint violations are defined:

   - an "anomaly code" specifies that a constraint violation results from the absence of a data value (incomplete information) or from the incompatibility of two data values (inconsistent information);
   - a "treatment process code" specifies delays of correction: some anomalies have to be corrected within 48 hours while others may be corrected only after days or weeks, depending on the complexity of the investigations needed;

3. a prohibited violation or an excess rate of allowed anomalies cause the rejection of the information: the employer will have to resubmit corrected data;
4. upon detection of an allowed constraint violation linked to a data value A:

- A is time-stamped, and tagged with anomaly code(s) and treatment process code(s);
- a record is created in an "anomaly file" (which is part of the database) with the following information: date and time when A entered the database, date and time when the violation was detected, anomaly code(s) and treatment process code(s), date and time of correction, identification of the correcting agency;

5. whenever a data value A is corrected or validated:

- a time-stamped record is created in the anomaly file to remember the history of the correction process (in view of possible juridical contest or lawsuit);
- consistency of the new value of A is checked against the relevant constraints and any new inconsistency is reported.

Despite this quite elaborated organization, problems subsist, because of the complexity of the application domain. Programming is complex and expensive, since laws and regulations, and thus constraints that reflect them, frequently change. In our analysis, a more comprehensive methodology, supported by CASE tools, at a suitably high conceptual level, would allow an easier and more rational management of constraint violations. Such a methology could guide and simplify the correction process as follows:

- decide on an order for correcting a set of constraint violations;
- identify possibly equivalent correction paths;
- select and evaluate the best paths, according to technical, semantic, and pragmatic organizational constraints.

These points are elaborated upon in the next sections.

3 REVISITING THE NOTION OF CONSTRAINT VIOLATION

Our initial formulation of the problem kept revolving around the question: "How do we deal with an inconsistent database in practice?"

This is practically hopeless, as Section 5 will conclude, within the bounds of current technology, as consistency is a basic hypothesis of mainstream data modeling (see, e.g, (Hulin, Pirotte, Roelants and Vauclair 1989)). So, in a way, the story (or, at least, the scientific story) ended there and we were left with the only possibility of adhoc solutions for correcting inconsistencies.

At the same time, we felt that more principled work could be applied to the reconstruction of consistent data. A revealing fact was that domain specialists

are unwilling to call "errors" these violations of consistency and call them "anomalies" instead.

Another approach turned out to be more fruitful. In fact, as soon as a constraint is violated, in a sense, it loses its status of constraint. As recalled in Section 2.2, information expressed by constraints is of the same nature as data structure information and violating a constraint is thus equivalent to violating the database schema.

The central idea of our approach is to reinterpret constraint violations as follows. Consider a database schema S composed of a data structure definition DS and a set of constraints IC. If some data D in a database of schema S violate some prescription of S, then D can be viewed as being in agreement with another schema S_w weaker than S. We will call S the "ideal" schema and S_w the "weak" schema. Correcting the violations of D with respect to schema S then amounts to updating the data (all or part of D and/or other data) of schema S_w so that the modified data comply with S. Thus the unpalatable problem of dealing with data that are inconsistent with their schema has been transformed into a problem of updates, which seems more comfortable to tackle.

The transformed problem is further simplified if the consistency violations of data D with respect to schema S are further analyzed as several simpler violations. The update process then has to go through a sequence S_w, S_1, ..., S_n, S of schemas, where each schema in the sequence is tighter (i.e., more constraining) than its predecessor.

The problem has become one of building a sequence of schemas, certainly not unique in general, of defining the correctness of a sequence, of studying the convergence of sequences, of identifying correct sequences, etc. These questions bear some resemblance with properties of sets of rules in active databases. They are elaborated upon in the next section.

4 DATABASE DESIGN AND INCONSISTENCIES

If constraint violations must be accepted, then the process of database design should be expanded for managing inconsistency, to include the following tasks:

1. determine which constraint violations will be accepted and which ones will not;
2. analyze dependencies among the constraints that may be violated;
3. organize the necessary (meta)information for managing violations.

4.1 Deciding on Acceptable Constraint Violations

The constraints C in a schema S are partitioned into two disjoint subsets C_{enf} and C_{viol}, where C_{enf} are the constraints actually enforced and C_{viol} are those for which violations are accepted.

Constraints C_{enf} are enforced in two ways. A few of them (typically some of the implicit constraints like key and relational referential integrity) are automatically enforced by the database management system. However, most constraints have to be reexpressed operationally in application programs to check that database consistency is preserved upon data entry and update.

Thus, if DS defines the structural part of a database schema S, then $DS + C_{enf}$ constitutes the weak schema, while $DS + C_{enf} + C_{viol}$ is the ideal schema.

In the scenario where the management of constraint violations is added to an operational database, deciding which violations are allowed is akin to a reverse engineering activity. Inputs for that activity are provided by the programs that are currently run against the database to detect and correct erroneous data. In practice, the decision is made according to business policies that vary from one application to another.

4.2 Precedence Among Constraint Violations

The second step in our strategy of inconsistency management consists in analyzing dependencies among constraint violations. For the example of Section 2, suppose that null values are accepted for the contributionRate of employers, a violation of IC_2. Checking the violation of, say, IC_3 or IC_8 for the corresponding employers becomes meaningless.

To exploit such relationships among constraints in C_{viol}, a precedence relationship P_{ic}, noted $C_1 \rightsquigarrow C_2$, is defined between two constraints C_1 and C_2, as follows: for a test of whether some data D satisfies C_2 to be meaningful, then D must satisfy C_1. In other words, the precondition for being able to test C_2 on D is that C_1 be satisfied by D.

P_{ic} is clearly non reflexive ($C_1 \not\rightsquigarrow C_1$) and transitive (if $C_1 \rightsquigarrow C_2$ and $C_2 \rightsquigarrow C_3$, then $C_1 \rightsquigarrow C_3$). Therefore, P_{ic} defines a partial order among constraints.

The precedence graph of Figure 2 corresponds to the constraints about the schema of Figure 1. For example, if some data D violate constraint IC_4, checking the violation of successors IC_5 and IC_8 of IC_4 is pointless.

The precedence graph permits a sharper localization of faulty data. Without the precedence graph, an investigation of which data violate a constraint C will return data violating constraints C' such that $C \rightsquigarrow C'$, in addition to data actually violating C. The precedence graph allows to tune the investigation process and distinguish C from C', and thus obtain a finer perception of data quality levels.

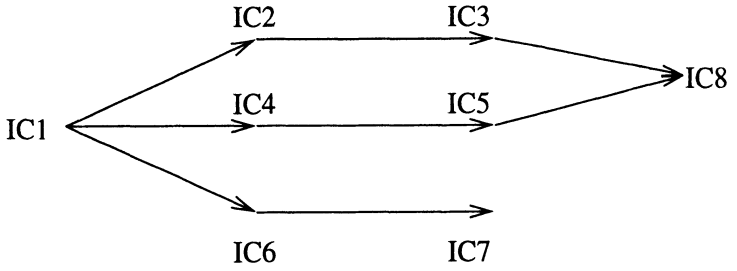

Figure 2 Precedence graph for constraints of Figure 1.

The precedence graph is used to correct an inconsistent database: the correction process can be viewed as evolving data down the precedence graph. Consider a database schema $S = DS + C_{enf} + C_{viol}$ and let $C_1, \ldots, C_k$ be a total order of constraints in C_{viol}, compatible with the partial order defined by the precedence relationship P_{ic}. The sequence $C_1, \ldots, C_k$ defines a sequence $S_0, \ldots, S_k$ of schemas between the weak schema $S_0 = DS + C_{enf}$ and the ideal schema $S_k = DS + C_{enf} + C_{viol}$, where $S_i = S_{i-1} + C_i$ for $i = 1, \ldots, k$.

Thus, if some data D violate some constraints in C_{viol}, the transition from S_{i-1} to S_i corresponds to correcting the violations of constraint C_i. Of course, in general, several total orders for constraints C_{viol} can be derived by topological sorting from the precedence graph. Each one corresponds to a sequence of schemas, that is, to a sequencing for the correction process.

Choosing a sequence of schemas for the correction process depends on many factors. The most crucial one may be the availibility of information. For example, in a federation of databases, data to be corrected may reside in a database over which the correcting agency does not have control. As another example, when data that were missing are received or when data are corrected, and the violation of some constraint C is thereby suppressed, constraints C' that are successors of C in the precedence graph can be tested.

Another factor that influences the choice of a sequence of corrections of some data D is the cost of testing constraints and correcting the data. Cost comprises computational resources (e.g., the time needed to run programs against the database to locate the data violating a constraint C) as well as business processes (e.g., correcting violations of C may involve requesting information from thousands of employers).

In practice, a large number of constraints may be allowed to be violated. Thus, computer support is needed to assist database designers in defining the precedence relationship P_{ic} between constraints. A CASE tool could provide the following functions:

- management of a set of constraints partitioned as C_{enf} and C_{viol}, with the possibility of constraints migrating from C_{enf} to C_{viol} and vice-versa;
- checks of consistency, redundancy, minimality, etc. of a set of constraints (if they are expressed formally in some specification language);

- support for the specification of the precedence relationship P_{ic}; for example, the tool could eliminate redundant links (e.g., those that can be deduced by transitivity) and test whether the precedence graph is acyclic.

Our analysis suggests several research avenues. For example, how are violated constraints identified for an inconsistent database? How can a sequence of updates be characterized precisely in terms of the schemas satisfied by the data? These questions are complex in their general form.

4.3 Building a Meta-Database

In an ideal world, the main recommendation to domain specialists and people managing a database would be: "Don't do it, keep your data consistent." But, as we have seen, this goal can often not be achieved in practice. Thus, in the real world, the recommendation becomes: "If you are compelled to accept some inconsistency, characterize it as fully as you can, so that enough information is provided for subsequent data correction."

As mentioned in Section 2.5, administrative agencies have done and are doing work in that direction, by means of elaborate application programs. However, this implies a programming task which is intensive, tedious, costly, and risky.

We advocate a more principled approach, through a comprehensive meta-database to store and manage data needed for the detection of errors and their correction.

In addition to the information needed for identifying errors, the meta-database could also record additional information useful for their management, like date and time of error detection, actions undertaken upon error detection (e.g., phone calls, letters sent, forms returned to the sender), personnel in the agency responsible for managing the inconsistency, date and time of error correction, and so on.

The creation of such a meta-database goes through the classical steps of database design, namely: (1) conceptual design, to identify the required information from an application domain perspective; (2) logical design translating the conceptual schema resulting from the previous step into a schema in the data model of a database management system (e.g., the relational model), and (3) physical design producing a schema for the target platform (e.g., Oracle).

Conceptual design of the meta-database involves determining the allowed violations (i.e., constraints C_{viol}), as well as the information necessary for adequately managing violations.

Figure 3 shows parts of a relational schema of a meta-database for the example of Figure 1. The relations store information about violations as follows*:

*Not all information is shown in Figure 3.

```
MissingEmployers(employerSSN,quarter,year)
MissingWorkers(workerSSN,quarter,year)
FictitiousCategories(categoryCode,categoryOfActivity,contribRate)
MissingCategory(employerSSN)
IC1Violations(SSN,firstName,lastName,firstNameNR,lastNameNR)
IC6Violations(workerSSN,employerSSN,quarter,year,...
      sum#daysPerWeek,#daysPerQuarter)
IC8Violations(workerSSN,salary,workContribRate,...
      emplContribRate,taxComputed,taxDeclared)
```

Figure 3 A sketch of schema for the meta-database.

- MissingEmployers records the employers who did not file forms for a given quarter;
- MissingWorkers records the workers for whom no declaration was filed;
- FictitiousCategories records the categories of activities with no special code in the categorization, and for which the agency creates a fictitious code;
- MissingCategory stores employers with a null value for their category of activity;
- IC1Violations keeps track of people whose identification data in the agency database does not match the identification data in the national register;
- IC6Violations keeps track of the workers for whom the sum of values in the multivalued attribute #daysPerWeek do not match the number of days work during the quarter;
- IC8Violations keeps track of the workers for whom the amount of tax, as computed by the agency, does not match the amount of tax declared in the entry forms.

Some information for the meta-database can be extracted from current data in the database through queries and views. Also, programs run against the database to detect and correct erroneous data produce information useful for the detection/correction process that should be integrated into the meta-database.

Still, implementing a large meta-database involves technical issues for which tools are not available off the shelf, and also elaborate (and costly) conceptual and organizational activities (Boydens 1996). Domain-specific cost/benefit analyses must be conducted to determine the breadth of metadata management that is appropriate for achieving the main objectives of the information system.

5 RELATED WORK

One area relevant to our work is that of exception handling. An in-depth study of an operating information process in a Fortune 100 organization is

reported in (Strong and Miller 1995). The goal of the study was to develop understanding about exceptions and derive managerial recommendations for treating them. Their definition of exceptions covers those generated by incomplete and erroneous information in inputs and outputs, requests to deviate from standard procedures and situations that computer-based systems were never designed to handle. They present several perspectives on considering exceptions: as infrequent random events, as errors to be eliminated, and as a normal part of organizational process.

In (Borgida 1985) integrity constraints are considered as normalcy conditions — ones that may occasionally conflict with the actual state of the world. Violations of constraints are allowed to accommodate exceptional facts and occurrences. The paper develops an exception-handling mechanism for a programming language allowing the definition of information systems at a conceptual level. It also investigates a first-order logic of constraints with exceptions.

There are some similarities between our treatment of database integrity and the work on reactive integrity maintenance (e.g., (Karadimce and Urban 1991, Urban and Delcambre 1990, Urban and Lim 1993, Fraternali and Paraboschi 1993, Fraternali and Paraboschi 1997)). In this context, constraints are repaired by active rules which can be sequenced to obtain the flexible behaviour needed for real-world informations systems. In (Baralis, Ceri and Paraboschi 1996) are given several ways to modularize/control active rules to obtain a better enforcement of integrity. Also, the use of rules for better conceptual modelling of informations systems is treated in (Ceri and Fraternali 1997, Ross 1994).

Traditional business data management handles simple fact databases only, typically ordinary relational databases. An early extension has consisted in adding deductive or active rules, that express general information (or knowledge). Rules allow to reduce redundancy in the fact database. The resulting systems have been known as knowledge-based systems, deductive database systems, or expert systems. With a simple form of rules (called definite rules), modeling the information content as a theory of classical first-order logic is now well understood (see, e.g., (Hulin et al. 1989) for a tutorial presentation). Data modeling obeys the closed world assumption, that is, all the data in the database are certain and consistent, and facts not present in the database are interpreted as false.

Things complicate considerably with the introduction of imperfect information (missing, disjunctive, uncertain data) (Zimányi and Pirotte 1997), negative information or inconsistency. Standard logic cannot deal with contradictory information: a single contradiction destroys the entire theory (*ex contradictione sequitur quodlibet*).

Query answering has been approached with logics generalizing standard first-order logic. The inference system that constructs query answers must be sound (i.e., produce only correct answers) in some well-defined logic and

complete (i.e., produce all correct answers) with respect to some intuitively acceptable notion of completeness. Inference in knowledge-based systems, unlike first-order logic, is nonmonotonic (i.e., not all consequences are preserved when a knowledge base is updated), which substantially complicates the semantic mechanisms of the extended logics.

Update operations (i.e., insert, modify, delete) can create conflicts between new information and information already present in the knowledge base. Several problems arise when assimilating new knowledge in knowledge bases. One of them is maintaining the consistency of the data, another is that the changes required are not necessarily unique.

Update and inference both have to be able to deal with inconsistent information according to certain rationality principles. Several inconsistency-tolerant logics have been proposed in the literature, including the four-valued logic of Belnap (Belnap 1977), the nonmonotonic logic of minimal inconsistency of Priest (Priest 1989), and the paraconsistent constructive logic of Nelson (Almukdad and Nelson 1984). Other systems discussed in the philosophical literature are presented, e.g., in (Priest 1979, Rescher and Brandom 1989). An interesting collection of papers can be found in (Wagner 1997) and in the references mentioned there.

A crucial question in inconsistency-tolerant logics is whether they enforce consistent inference or, rather, whether they allow for inconsistent conclusions. Traditional paraconsistent logics yield both p and its negation $\neg p$ as valid conclusions whenever p is contradictory with the premise set, which is undesirable from an information processing point of view.

A solution is to isolate contradictory pieces of information so that they are not used as valid premises in further inferences (*ex contradictione nihil sequitur*) (Wagner 1991). Ordered theories with an explicit ordering of rules allow to define the priorities among competing rules. If two arguments are in conflict with each other, but both have the same conclusive force, they neutralize (or block) each other. If an argument has a stronger conclusive force than all other conflicting ones, it defeats them, and qualifies as a justified argument.

Another approach, when handling conflicts, is to modify existing information, while obeying some principle of minimal change or "minimal mutilation" of the knowledge base. To comply with this requirement, a knowledge-based system needs a measure of change for comparing different candidates for the result of an update. Such a measure depends on a notion of information content for a knowledge base.

For example, a deductive database can be updated by changing facts but not rules, so that all constraints remain satisfied (Lobo and Trajcevski 1997). Users may be involved in this process to choose the appropriate change from a set of alternatives whenever there is more than one possibility to perform the update. A common notion of minimal change involves a preference of positive

information over negative one: a possible change is preferred over another one if it deletes fewer facts.

In another approach (Demolombe and Jones 1993), the management of constraint violations is expressed in terms of updates that remove situations where the database contains wrong beliefs about the real world (validity violation) or where some beliefs that logically derive from the database are missing (completeness violation).

Finally, the connection between belief revision and nonmonotonic reasoning was formally clarified, thus throwing light on the connection between consistency-restoring and reasoning-from-inconsistency approaches discussed in (Benferhat, Dubois and Prade 1995).

Most work in logic traditionally was done on the study of reasoning on the basis of a fixed theory. It is only recently that research has addressed the assimilation of new information into a changing theory, in particular in philosophical logic. A large body of theoretical results in this field has become known under the name of AGM theory, after its originators Alchourón, Gärdenfors, and Makinson (Alchurrón, Gärdenfors and Makinson 1984). In particular, it has been adapted as a theoretical basis for modeling knowledge assimilation.

The main criterion in deciding whether to preserve certain beliefs in the face of revision is whether they can be held consistently after the new information is incorporated. Thus, revision can be seen as a two-step process (del Val 1997). In the first step, the new information is checked for consistency with existing beliefs. In the case of inconsistency, as few beliefs as possible are withdrawn to restore consistency. In the second step, the beliefs kept in the first stage are merged with the new information.

However, as stated in (Tennant 1997), the AGM theory is rooted on the view that a classical logic theory is the paradigm of knowledge-based systems. The AGM theory does not take into account that knowledge assimilation, unlike theory change, is concerned with non-classical, nonmonotonic knowledge-based systems. The inadequacy of the AGM recovery postulate, which captures the minimal mutilation principle, is discussed in (Tennant 1997). The conclusions are that the AGM contraction and revision operations based on the postulate of recovery are inadequate, and that the remaining AGM postulates are too weak to capture any interesting property.

As follows from the above discussion, operational solutions for handling inconsistency in logical theories and knowledge-based systems exist neither for inconsistency-tolerant inference, nor for theory change and knowledge assimilation. Results obtained for idealized settings (e.g., a set of sentences in classical propositional logic) do not extend to realistic knowledge-based systems with non-classical inference operations. Many theoretical questions remain open and the current research results have little direct relevance for the practical management of inconsistency.

6 METHODOLOGICAL SUMMARY AND FURTHER WORK

6.1 Summary

Business policies and real-world requirements often make erroneous data inescapable in realistic information system applications.

An analysis of constraint violations in the databases of the Belgian social security agency (ONSS–RSZ) was performed in the light of current practices for coping with and resolving the resulting inconsistencies. The management of inconsistent information was then re-interpreted in terms of current database technology, as (1) introducing the erroneous data in a less constraining database (i.e., with a weaker schema) than the intended database (with the ideal schema); (2) progressively updating the data from the weaker schema to make it comply with the ideal schema.

Since data of different quality coexist in the database, many possible intermediate schemas may exist between the weaker schema and the ideal one. Then, the solution was described as determining a sequence of schemas, each more constraining than its predecessor in the sequence, as defining the correctness of a sequence, as studying the convergence of sequences, as identifying correct sequences, etc.

It is our tenet that the only practical way to cope with inconsistent data, given the current state database technology, is to systematically keep track of which data are inconsistent in the database as well as of all the information needed for its management. We argued that this should be done explicitly as a step of database design or reengineered on the operational database. We recommended the following activities:

- decide on constraints that are allowed to be violated;
- build a precedence relationship among those constraints to help construct "semantically meaningful" sequences of corrections of erroneous data;
- design a meta-database for keeping track of all necessary information needed for the management of inconsistency.

Finally, we reviewed how inconsistency has been managed in knowledge-based systems, and we concluded that more work is needed, from both a theoretical and a practical perspective.

6.2 Methodology

The paper motivates a methodological framework for information system research, that aims at improving the quality of solutions provided by application programs. The methodology addresses complex problems, that is, problems

that are not easily solved by the current technology in the form that they arise in the application domain.

Clearly, more fundamental research on the theory of information systems is needed to be able to handle more satifactorily applications like social security data management. A contribution of our analysis is the identification of desirable technological advances. For constraint violations studied in this paper, flexible ways to deal with inconsistency would clearly improve the situation, if database technology aims at dealing more adequately with a real world, where inconsistency manifests itself in several ways.

This paper suggests a general methodological approach to managing complex problems that cannot be directly handled by the current technology. Complexity should be addressed in four, not strictly sequential, stages:

- a thorough analysis of actual requirements (both those satisfied and those not satisfied by the existing system) through a partnership between specialists in information system development and specialists of the application domain;
- a precise formulation that carefully identifies, in terms of the functions offered by current technology, those requirements that cannot be addressed by a direct application of the technology and how those requirements are currently being addressed;
- the identification of realistic requirements on suitable evolutions of the technology to better address the complex problems;
- the formulation of recommendations to specialists of the application domain to modify or adapt their practices in order to take the best advantage of current technology and its foreseeable evolutions.

In order to be effective, our methodological framework for dealing with inconsistency should be integrated into the database management process and supported by CASE tools coupled with it.

6.3 Further Work

Our methodological approach lends itself to the analysis of other aspects of social security database management. The basic rules of the game for collecting employers' contributions and distributing social security benefits change with time, this is real life.

Thus, reinterpreted in terms of current technology, the task of the administrative information system is to manage a collection of time-stamped databases, i.e., a collection of schemas with their associated data. This can be modeled as a collection of versions of databases, where each version has a fixed schema (i.e., a schema that does not vary with time) and one database is the current database. As time passes, rules will evolve and a new schema

will be defined from the schema of the current database, which will become a past database.

Current technology does not deal well with versions of schemas. Relevant research on sequences of versions has been conducted in the computer-aided design area (see, e.g., (Katz 1990)). The concept of "workspace models" provides a mechanism through which new versions are made visible to a community of designers. The idea is that "private workspaces" can contain incomplete work in progress: before being validated and made available, new versions are checked until no more errors are detected. A first important difference with our approach is that, in administrative data management, there is no clear private workspace and information has to be made available for public use as quickly as possible, and often before being completely checked and corrected. Another difference with design problems is that, in our case, it is often difficult to know when the correction process has been completed.

Another outcome of our methodological approach is the identification of suggestions for improving the practices of actors in the application domain. To improve data management of administrative databases, the actors in the application domain who define the new rules of operation should be required to accompany the definition of new rules with enough information to enable the applications that operate on the data to correctly relate the various databases (the past ones and the current one) that coexist and cooperate for obtaining useful output.

The problem is compounded with retroactive rules. Here database technology cannot help if the new rules are not supplemented with enough information to convert the data in the first past database (the most recent one except for the current one) into data that complies with the new rules. Database technology can help in converting declarative rule information into operational programs that transform the data.

We are currently studying other similarly complex problems and revisiting, with that methodological framework in mind, a number of case studies that were conducted in our YEROOS* research group.

REFERENCES

Alchurrón, C., Gärdenfors, P. and Makinson, D.: 1984, On the logic of theory change: Partial meet functions for contraction and revision, *Journal Symbolic Logic* **50**, 510–530.

Almukdad, A. and Nelson, D.: 1984, Constructible falsity and inexact predicates, *Journal Symbolic Logic* **49**(1), 231–233.

Baralis, E., Ceri, S. and Paraboschi, S.: 1996, Modularization techniques for active rules design, *ACM Trans. on Database Systems* **21**(1).

*YEROOS (Yet another project on Evaluation and Research on Object-Oriented Strategies), http://yeroos.qant.ucl.ac.be

Belnap, N.: 1977, A useful four-valued logic, *in* G. Epstein and J. Dunn (eds), *Modern Uses of Many-valued Logic*, Reidel, pp. 8–37.

Benferhat, S., Dubois, D. and Prade, H.: 1995, How to infer from inconsistent beliefs without revising, *in* C. Mellish (ed.), *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence*, Morgan Kaufmann, Montreal, Canada, pp. 1449–1455.

Borgida, A.: 1985, Language features for flexible handling of exceptions in information systems, *ACM Trans. on Database Systems* **10**(4), 565–603.

Boydens, I.: 1992, La banque de données LATG de l'ONSS. Les flux de l'information traitée à partir d'une banque de données : étude critique. Mémoire de Licence Spéciale en Sciences de l'Information et de la Documentation, INFODOC, Université Libre de Bruxelles. Lauréat du Concours des Bourses de Voyage de la Communauté Française de Belgique.

Boydens, I.: 1995, Statistical exploitation method for administrative databases, *Proc. of the Int. Conf. on New Techniques and Technologies and Statistics*, Bonn, Germany, pp. 63–70.

Boydens, I.: 1996, Meta-information systems, critical interpretation tools for computer sources, *History and Computing* **8**(1), 11–23. In French.

Ceri, S. and Fraternali, P.: 1997, *Designing Database Applications with Object and Rules: The IDEA Methodology*, Addison Wesley Longman.

del Val, A.: 1997, Nonmonotonic reasoning and belief revision: Syntactic, semantic, foundational, and coherence approaches, *Journal of Applied Non-Classical Logics* **7**(1–2), 213–240. Special issue on Handling Inconsistency in Knowledge Systems.

Demolombe, R. and Jones, A.: 1993, Integrity constraints revisited, *in* A. Olivé (ed.), *Proc. of the 4th Int. Workshop on the Deductive Approach to Information Systems and Databases*, Barcelona, Spain, pp. 309–333.

DOS: 1992, Utilization of administrative databanks as a mean to produce statistical information. DOSES-EUROSTAT (action A4), Final report. Torino, Richerche e progretti.

Fraternali, P. and Paraboschi, S.: 1993, A review of repairing techniques for integrity maintenance, *in* N. Paton and M. Williams (eds), *Proc. of 1st Workshop on Rules in Database Systems*, WICS, Springer-Verlag, Edinburgh, Scotland, pp. 333–346.

Fraternali, P. and Paraboschi, S.: 1997, Selecting production rules for constraint maintenance: Complexity and heuristic solution, *IEEE Trans. on Knowledge and Data Engineering* **9**(1).

Hulin, G., Pirotte, A., Roelants, D. and Vauclair, M.: 1989, Logic and databases, *in* A. Thayse (ed.), *From Modal Logic to Deductive Databases*, John Wiley & Sons, pp. 279–350. In French: Logique et bases de données, In: Approche logique de l'intelligence artificielle, vol. 2, Dunod, 1989, pp. 311–294.

Karadimce, A. and Urban, S.: 1991, Diagnosing anomalous rule behavior in databases with integrity maintenance production rules, *Proc. of the 3rd Int. Workshop on Foundations of Models and Languages for Data and Objects*, Aigen, Austria, pp. 77–102.

Katz, R.: 1990, Towards a unified framework for version modeling in engineering databases, *ACM Computing Surveys* **22**(4), 375–408.

Lobo, J. and Trajcevski, G.: 1997, Minimal and consistent evolution of knowledge bases, *Journal of Applied Non-Classical Logics* **7**(1–2), 117–146. Special issue on Handling Inconsistency in Knowledge Systems.

Pirotte, A., Roelants, D. and Zimányi, E.: 1991, Controled generation of intensional answers, *IEEE Trans. on Knowledge and Data Engineering* **3**(2), 221–236. Short version in P-91/02.

Priest, G.: 1979, Logic of paradox, *Journal of Philosophical Logic* **8**, 219–241.

Priest, G.: 1989, Reasoning about truth, *Artificial Intelligence* **39**, 231–244.

Rescher, N. and Brandom, R.: 1989, *The Logic of Inconsistency*, Blackwell.

Ross, R.: 1994, *The Business Rule Book: Classifying, Defining and Modeling Rules*, Database Research Group, Inc.

Strong, D. and Miller, S.: 1995, Exceptions and exception handling in computerized information processes, *ACM Trans. on Office Information Systems* **13**(2), 206–233.

Tennant, N.: 1997, On having bad contractions, or: No room for recovery, *Journal of Applied Non-Classical Logics* **7**(1–2), 241–266. Special issue on Handling Inconsistency in Knowledge Systems.

Urban, S. and Delcambre, L.: 1990, Constraint analysis: A design process for specifying operations on objects, *IEEE Trans. on Knowledge and Data Engineering* **2**(4), 391–400.

Urban, S. and Lim, B.: 1993, An intelligent framework for active support of database semantics, *International Journal of Expert Systems* **6**(1), 1–37.

Wagner, G.: 1991, Ex contradictione nihil sequitur, *in* R. Reiter and J. Mylopoulos (eds), *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence*, Morgan Kaufmann, Sydney, Australia, pp. 538–546.

Wagner, G. (ed.): 1997, *Handling Inconsistency in Knowledge Systems*, Editions Hermès. Special issue of the Journal of Applied Non-Classical Logics, 7(1–2).

Wand, Y. and Wang, R.: 1996, Anchoring data quality dimensions in ontological foundations, *Comm. of the Assoc. for Computing Machinery* **39**, 86–95.

Zimányi, E. and Pirotte, A.: 1997, Imperfect knowledge in databases, *in* A. Motro and P. Smets (eds), *Uncertainty Management in Information Systems: from Needs to Solutions*, Kluwer, pp. 35–87.

7 BIOGRAPHY

Isabelle Boydens has been consultant at the research department of the SmalS-MvM in Belgium since 1996. From 1991 to 1996, she was researcher at the Université de Liège in Belgium. Her scientific interests include conceptual modeling, data quality analysis, and improvement methods for large administrative information systems.

Alain Pirotte has been a professor in computer science at the Université catholique de Louvain in Belgium since 1991. From 1969 to 1991, he was a researcher at the Philips Research Laboratory in Brussels. His scientific interests include database management, analysis methodology for information systems, and advanced applications of those technologies.

Esteban Zimányi has been lecturer in computer science at the Université Libre de Bruxelles in Belgium since 1992. He was visiting researcher at the Ecole Polytechnique Fédérale de Lausanne in Switzerland during 1997. His research interests include conceptual modeling, geographical information systems, temporal databases, and analysis methods for system development.