Generalising the Consistent Database Sampling Process *

Jesús Bisbal, and Jane Grimson
Computer Science Department, Trinity College Dublin, Ireland.
E-mail address: Jesus.Bisbal@cs.tcd.ie, Jane.Grimson@cs.tcd.ie

Abstract

Database sampling is widely used in many database applications when, for efficiency reasons, an entire database cannot be used. Although random sampling is the most commonly used sampling strategy, this paper analyses the use of *consistent sampling*, that is, sampling according to certain criteria (e.g. satisfaction of integrity constraints) used to evaluate the consistency of the resulting sample. This alternative to random sampling is particularly appropriate in the context of constructing prototype databases to support *Information System Development*.

This paper firstly presents a framework for evaluation of prototype database construction methods. Then a general description of the Consistent Database Sampling Process is introduced. Finally the paper outlines a sampling tool which implements this Process.

1 Introduction

Database Sampling is commonly used in a wide range of applications when the size of the database makes impractical, in terms of time or resources, to

*Extended version of a paper published in the Proceedings of the Joint Meeting of the 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000) and the 6th International Conference on Information Systems Analysis and Synthesis (ISAS'2000), Orlando, USA, July 23-26, 2000. Published by the International Institute of Informatics and Systemics (IIIS).

use the entire database. Examples of such applications include:

Data Mining When discovering useful information from large amounts of data a trade-off between efficiency and accuracy can be achieved by analysing only a sample of the database [KM94].

Query Estimation Approximate answers to aggregate queries (e.g. number of tuples satisfying a particular predicate) can be efficiently computed by answering these queries for a sample of the database [Olk93].

Information System Development Data-intensive applications development requires prototype databases to support several stages of the development process, e.g. validation, testing, users training. In particular, in the context of Legacy Information System Migration [BLWG99], a *Sample Database* has been identified as essential to the success of the migration process [WLB+97].

Most of these applications of database sampling use random sampling to select the tuples to be included in the Sample. In some cases, e.g. query estimation[1], this is clearly the appropriate approach as it is the simplest approach, and more complicate sampling mechanisms will not provide better results. This paper, however, is concerned with the use of database sampling to support the development of information systems. In particular when an existing, operational database is already available at development time but using the entire database for development purposes is too costly, as is the case in Legacy Information System Migration. In such situations random sampling leads to prototype databases without any particular semantics, as the integrity constraints the database must satisfy during operation are not considered during sampling. The resulting prototype database is not considered appropriate to support the information system development process [WJ91, BWLG98].

This paper proposes that when operational data is available it should be used to build the database prototype(s) to support the development process. Throughout the paper the term *Prototype Database* will refer to any database used to support the development of an information system, independently of how this database has been constructed. *Sample Database* is used to refer

[1] An alternative to random sampling in the context of data mining has never, to the best of authors' knowledge, been proposed. This possibility will be briefly analysed in Section 6.

to a Prototype Database built by sampling an existing database. The term *Consistent Database Sampling* refers to the extraction of a sample from a database enforcing a predefined set of integrity constraints in the resulting database. For the purposes of this paper consistent sampling is considered to be the opposite to random sampling. Finally, Prototype Databases which are not Sample Databases will be called here *Test Databases*.

This paper presents a general description of the Consistent Database *Sampling Process* (CDSP) and a working prototype of a database sampling tool as an implementation of this Process. This general description provides a better understanding the key issues and challenges that must be addressed when consistently sampling from a database. Also, it provides the foundations for a well-defined *Sampling Protocol* that allows for an incremental construction of a database sampling tool, which is capable of constructing Sample Databases satisfying multiple criteria simultaneously.

The reminder of this paper is organised as follows. The next Section defines a framework for evaluation of Prototype Database construction methods and justifies the use of database sampling in the case of Legacy Information System Migration. Section 3 describes the Consistent Database Sampling Process, which is the basis for the Sampling Protocol analysed in Section 4. In Section 5 an implementation of this Process, termed Consistent Database Sampling Tool, will be outlined. The final Section summarises the paper and gives a number of future directions for this research.

2 Database Prototyping in Information System Development

When developing data-intensive applications there is a need to prototype the database that the applications will use during operation. Such a database can be used to support several stages of the development process [Som95]:

Validation of database and application requirements Different designs can be checked for completeness and correctness [NML93].

User Training Applications will need a database on which to operate with during user training.

Testing As needed in, for example, system functional test [Bei95], and performance evaluation [GSE+94].

Only a few approaches for building Prototype Databases have been reported in the literature. Refer to [BG00] for a brief description of what the authors consider the most relevant solutions. Existing approaches can be classified according to two orthogonal criteria: (1) the origin of the data used to populate the Prototype Database; and (2) the amount of semantic information the Prototype Database contains:

(1) Synthetic vs. Operational Data When a Prototype Database is populated from predefined domains which have no relationship to its application domain, this data is commonly referred to as *synthetic data*. In contrast, the term *operational data* is used in cases where the prototype database is populated from data of a working database. Test Databases, as referred to in Section 1, are those populated using synthetic data; Sample Databases result when using operational data.

(2) Poor vs. Rich Semantics Existing solutions enforce different sets of integrity constraints in the resulting Prototype Database. Random values generation [GSE+94] and random sampling [KM94], for example, would result in databases with poor semantic information as no particular constraints would be enforced. An approach which includes a highly expressive language (e.g. first-order-logic or FOL) used to define the set of integrity constraints being enforced would produce databases with richer semantic information.

The kind of support that a Prototype Database built using a particular method provides to the information system development process can be assessed by identifying where this method falls according to the two classification criteria given above. These criteria can, therefore, be seen as a framework for evaluation of Prototype Database construction methods. Figure 1 shows a graphical representation of such framework, where the two criteria, data origin and semantic content, have been displayed along each of the axes. Given one concrete Prototype Database construction method the relative semantic richness it can enforce will provide a value for its X coordinate in Figure 1. Similarly, the percentage[2] of operational data used to populate a prototype database will define a value for its Y coordinate. These two coordinates define a point in Figure 1. This Figure has been divided

[2]A method could combine operational with synthetic data to populate a Prototype Database.

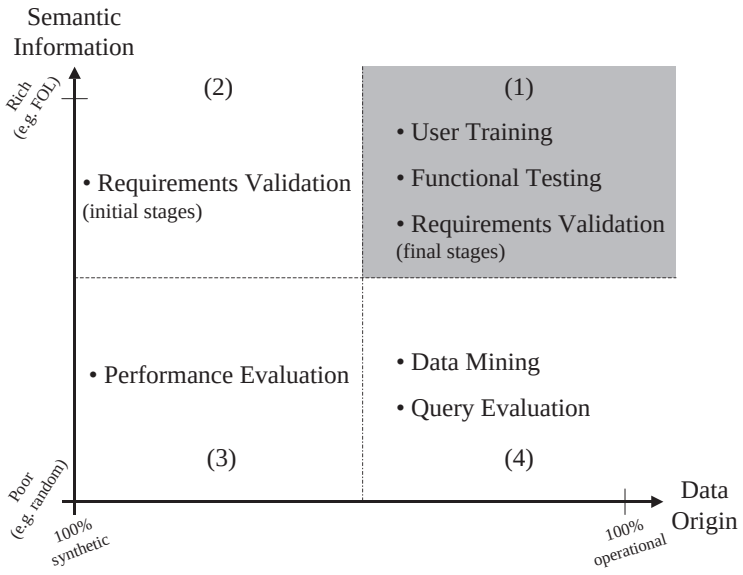

Figure 1: Framework for Evaluation of Prototype Database Construction Methods

into four quadrants that indicate which methods are more appropriate for the various stages of the information system development process.

Methods in quadrant (1) (e.g. [BWLG98]) lead to databases highly similar to those the information system will use in production mode. For this reason the resulting Prototype Databases will be useful for user training, system functional testing and at the later stages of requirements validation. Methods that fall in quadrant (3) (e.g. [GSE+94]) do not enforce complex integrity constraints and do not need to query other data sources to populate the constructing Prototype Database. Such methods can, therefore, efficiently generate large volumes of data, which makes them particularly appropriate for performance evaluation of information systems. Quadrant (2) represents a trade-off between efficiency (quadrant (3)) and faithfulness (quadrant (1)). For this reason methods in this quadrant (e.g. [NML93]) are appropriate for initial database requirements analysis where different alternative designs must be explored (which requires information) and therefore several Prototype Databases may need to be generated (need for efficiency). Finally, methods in quadrant (4) (e.g. [Olk93, KM94]) would not be useful to support information systems development. This quadrant indicates the

use of operational data and the inclusion of little semantic information in the Prototype Database being populated. Using operational data results in prototype databases being very similar to the production database. Therefore ignoring semantic information would defeat the whole purpose of using operational data. Methods within this quadrant are, however, the only ones used in applications of sampling such as data mining and approximate query evaluation.

The above discussion places Sample Databases and Test Databases, as referred to in Section 1, in the context of information systems development. Sample Databases are those produced by methods that fall in quadrants (1) and (4), and Test Databases by those in quadrants (2) and (3). This paper is concerned with the construction of *consistent* Sample Databases (see Section 1), which result when using methods in quadrant (1), grayed in Figure 1. It is clear from this Figure that this class of Prototype Databases can be used to support more stages of the development process than those that result from using methods in other quadrants. Although operational data is not always available when developing information systems, it is available in the case of Legacy Information System Migration, and therefore Consistent Sample Databases are the preferred Prototype Database type to be used in a migration project. This is further discussed in the Section 2.1.

2.1 Database Sampling in Legacy Information System Migration

The previous Section justified the use of operational data and rich semantic information to build a Prototype Database that will support a legacy information system migration project. It remains to be clarified why, in this context, a *Sample* of an existing (migrating) database, as opposed to the entire database, must be used.

The legacy migration process can be defined as follows [BLWG99]:

Legacy Information System Migration Moving an information system to a more flexible environment which allows information systems to be easily maintained and adapted to new requirements, retaining original system data and functionality without having to completely redevelop them.

In the context of legacy migration the operational (legacy) database will have to be migrated to a new environment. This data must be transformed from

the existing to the new schema[3] [WLB+97]. The transformation mechanism must be developed and tested. Using the entire database for testing purposes only would be too costly, therefore a Sample Database is required. Also, when developing the new information system a Prototype Database will be needed. As discussed in Section 2 operational data should be used in this context, and again it would not be cost-effective to migrate the entire database simply for development purposes. A Sample of this database is sufficient.

The above discussion justifies that, in the context of legacy information system migration, the term Sample Database is used interchangeably with Consistent Sample Database. For the purposes of this paper a *(Consistent) Sample Database* is defined as follows:

(Consistent) Sample Database Any consistent subset of the data stored in a Source Database.

The term *Source Database* refers to the database which is being sampled. The word *subset* has been used here to illustrate the fact that a sample database is expected to be smaller than its Source Database, as discussed before. A Sample Database must be *consistent* with an specified set of integrity constraints, ideally the same set as the Source Database. Finally, the word *any* found in this definition relates to the actual size of the sample (e.g. for a relational database, the number of tables, and the number of tuples in each table). No minimum or maximum size is required. As long as a database is built from data of a Source Database and it meets the specified set of integrity constraints, it can be considered to be a Sample of this Source Database. Restrictions on the Sample size may be required in the context of a particular application of database prototyping.

When sampling from an existing database, the most significant challenge regards the correctness of the resulting database, according to the set of integrity constraints being enforced. Figure 2 illustrates the context in which the sampling process takes place. The database itself is not sufficient to produce a consistent Sample Database. The extraction of such a Sample requires knowledge about the set of constraints the Sample must meet, since this information is used to guide the sampling process. A representation of integrity constraints particularly well suited for sampling purposes, termed *Insertions Chain Graph*, was reported in [BWLG98]. This is a graphical

[3]These two schemas would be, in general, significantly different, and the transformation mechanism highly complex.

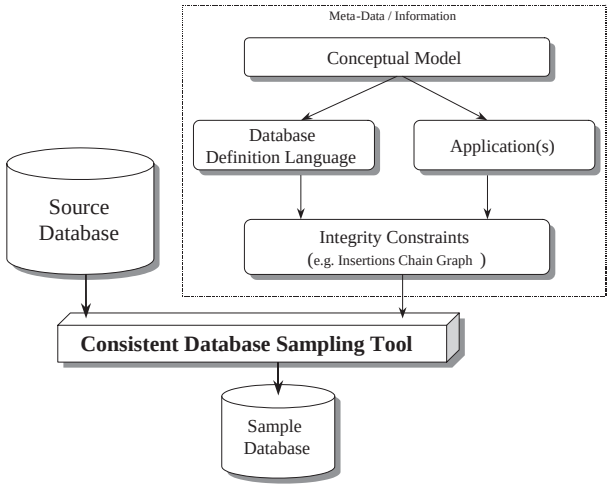

Figure 2: Information sources involved in Consistent Database Sampling

description of the set of insertions into the Sample Database that are required after performing one particular insertion, in order to keep the Sample Database consistent.

As shown in Figure 2, the set of integrity constraints held by a database is buried within both the Database Definition Language (DDL) and the applications. These constraints can be expressed in a conceptual model, which may not be available in the context of legacy information system migration.

The process of consistently sampling from a Database can be described abstracting the details of a concrete database model or integrity constraints type. This is done next in Section 3.

3 Consistent Database Sampling Process

A generalised description of the consistent database sampling process is used here to identify the key issues to be addressed during sampling and therefore providing a better understanding of the process itself. The General Consistent Database Sampling Process, hereafter CDSP, is shown in Figure 3. It is general enough to be applied to any kind of database and with an arbitrary collection of integrity constraints as it does not make any assumptions on this respect. This Figure shows, in fact, the structure of any iterative program: initialisation, condition, action, and preparation for next iteration.

```
void DatabaseSampler::ExtractSample(){
    InitialiseProcess();   //Needed information.
    while(!StopSampling()){
        currentInstance = SelectInstance();  //From sourceDB.
        SampleDB.insertInstance(currentEntity, currentInstance);
        Synchronise();      // Inter-Sampler consistency.
        UpdateProcess();  // Intra-Sampler consistency.
    }
}
```

Figure 3: General Consistent Database Sampling Process (CDSP)

However, it makes explicit the fact that the process is extracting a Sample from a database. The initialisation step is represented by a call to the *InitialiseProcess()* method. It has been identified that in all types of sampling there is a need for some kind of data structure that must be used to guide the sampling process. This initialisation step is responsible for setting up this information. An example of such data structure would be an Insertions Chain Graph (see Section 2.1).

The next step of the Process involves a condition that determines when to stop the sampling process, and it is represented by a call to method *StopSampling()*. This condition will be satisfied, for example, when the Sample Database satisfies the required set of integrity constraints, a predefined sample size is reached, etc.

Sampling a database is about selecting the appropriate set of instances[4] *according to given criteria.* The first of these selections, which does not depend on previous selections, is represented by a call to *SelectInstance()*. The criteria used to make this initial selection would depend on the database sampling application at hand. Examples of possible criteria would include: random selection, selection of an instance based on the number of additional instances that must be inserted into the Sample in order to maintain consistency (see [BWLG98] for details), etc. The selected instance must be inserted into the Sample Database, calling to *InsertInstance()*. This step has been included only to indicate that an appropriate interface to the Sample Database is required.

[4]The term *instance* is used here to refer to a data item in the database. In a relational database, for example, that would be a *tuple*.

The rest of the required selections, which depend on previous selections, are represented by method *UpdateProcess()*, which ensures that the instance selected by *SelectInstance()* is kept consistent with the specified criteria (e.g. set of integrity constraints). A concrete implementation of this method for a wide range of integrity constraint types was reported in [BWLG98]. This implementation was based on using a representation of integrity constraints termed Insertions Chain Graph (see Section 2.1).

The information being used to guide the sampling process must be updated after each insertion as this will determine which other selections are required. *UpdateProcess()* is also responsible for updating this information.

Finally, *Synchronise()* is the central part of the Sampling Protocol underlying this Model and is described in the next Section.

4 Consistent Database Sampling Protocol

The CDSP described in the previous section was developed as a general description of the set of activities and their relationships common to any database sampling process, independent of the criteria used to evaluate the consistency of the resulting Sample Database. Since these activities are common to any type of sampling, they can be seen as the functionality a *Consistent Database Sampling Tool* (or CDST), see Figure 2, should support. The design principle behind this general description is that a CDST would be composed of different sampling modules. Hereafter, such sampling modules will be referred to as *(Database) Samplers*. Each sampler extracts a sample from a database according to some criteria (e.g. concrete integrity constraint type, random[5]). A CDST would include several of these samplers. Using the CDSP described in Section 3 existing samplers can be integrated to create more sophisticated samplers which would sample a database according to several criteria *simultaneously*. This allows a CDST to be constructed *incrementally*, developing different samplers *independently* and then integrating them to build more complete samplers.

For the Database Sampler Integration Mechanism (see Section 5) to work, the CDSP relies on the fact that any Database Sampler will adhere to the

[5]References to random sampling are included here for completeness only. The preferred consistency evaluation criteria would be the satisfaction of sets of integrity constraints. Including random sampling would, however, extend the range of applications where a database sampling tool could be used.

following *Sampling Protocol*:

1. **Interface** All complying samplers must *implement* the following five methods: *InitialiseProcess()*, *StopSampling*, *SelectInstance()*, *Synchronise()*, and *UpdateProcess()*.

2. **Semantics** Their *semantics* must be as explained in Section 3.

3. **Inter-operation** *After each insertion* into the Sample Database methods *Synchronise()* and *UpdateProcess()*, in that order, must be called.

The inter-operation of database samplers that adhere to this Protocol relies on the semantics of methods *Synchronise()* and *UpdateProcess()*:

UpdateProcess() As described in the previous Section, this step ensures that the last insertion into the Sample Database is kept consistent with respect to the integrity constraints enforced by a single sampler.

Synchronise() A sampler must always call to method *Synchronise()* after performing an insertion into the Sample Database. This will, in case this sampler is integrated with other samplers, ensure that this insertion maintains consistency with the integrity constraints the other samplers enforce. For this reason *Synchronise()* can be seen as the enforcement of Inter-Sampler Consistency (see Figure 3), as opposed to Intra-Sampler Consistency which is achieved by *UpdateProcess()*.

By implementing the interface defined in Section 3, and following the Protocol analysed in this Section, a set of Database Samplers can be developed independently and concurrently, and then integrated according to the requirements of a particular Prototype Database application. This process has been tested creating an initial prototype of a generic CDST. This tool is described in the next Section.

5 Implementing the Sampling Process: A Consistent Database Sampling Tool

The initial step when building a Consistent Database Sampling Tool (CDST) is the construction of a set of *Basic Database Samplers*, that is, Database Samplers which are built independently without integrating other existing

samplers. The appropriate set of Basic Samplers to be used depends on the Prototype Database application. Each of them should sample a database according to one single criterion only. The current implementation of the CDST supports the following Basic Samplers:

Random Sampler Randomly selects the specified number of tuples from a specified table. Several instances of this type of sampler could be used, each of them sampling different tables.

Sampler with Functional Dependencies Selects the minimum size sample of a specified table that satisfies the specified set of functional dependencies.

Random Sampler with Functional Dependencies Randomly samples a specified table until the specified set of functional dependencies is satisfied. This sampler does not search for the minimum size Sample Database as the previous sampler.

Sampler with Referential Integrity Constraints Using an Insertions Chain Graph (see Section 2.1) to represent the set of referential integrity constraints that must hold in the database, it extracts a sample that contains at least the specified number of tuples in each table and that satisfies this set of constraints.

Once the set of Basic Samplers has been developed, they can be integrated to create more sophisticated samplers that sample a database according to several criteria simultaneously. Which Basic Database Samplers are integrated depends on the particular application of database sampling at hand. By having a set of Samplers which implement simple sampling criteria and then integrating them, as opposed to creating one single sample that implements all criteria, Basic Samplers can be re-used and integrated in different ways for different database sampling applications. This design makes the CDST more flexible and extensible.

The Database Sampler Integration Mechanism, hereafter DSIM, for aggregating the behaviour of Database Samplers which adhere to the Sampling Protocol relies on an additional Database Sampler type called *SamplerIntegrator*. This Sampler takes the list of all (instances of) Database Samplers to be integrated, and uses the fact that each of them adheres to the Sampling Protocol to reach a database which satisfies the sampling criteria (e.g.

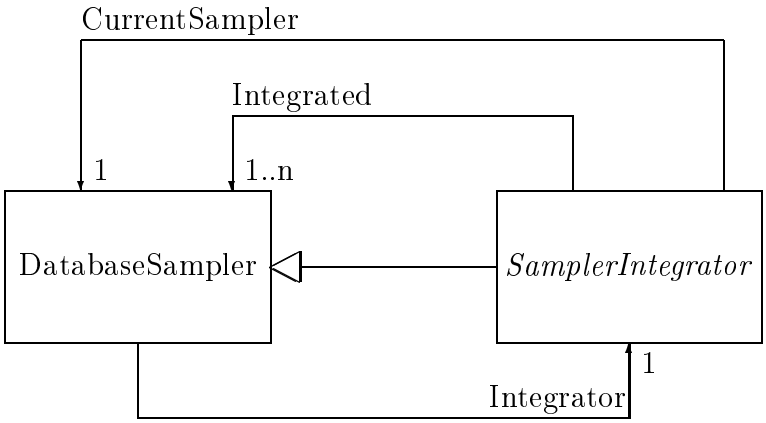

Figure 4: UML class diagram for the design of the *SamplerIntegrator* module

integrity constraints types) of all those Samplers simultaneously. Figure 4 shows a UML [MK97] Class Diagram for the design of SamplerIntegrator.

As shown in Figure 4, SamplerIntegrator inherits from class DatabaseSampler. This class represents the abstract concept of database sampler and therefore it implements the CDSP shown in Figure 3 of Section 3. All samplers that form part of a CDST, including SamplerIntegrator, inherit from class DatabaseSampler and therefore they must implement this CDSP and follow the Sampling Protocol of Section 4. Since all samplers, basic and integrated, implement that interface several levels of integration are possible. That is, when an instance of SamplerIntegrator integrates a set of samplers, it does not need to know whether they are basic database samplers or other instances of SamplerIntegrator. This design leads to a very flexible, transparent and incremental process of building comprehensive database samplers that reflect the complexity of the source database.

In Figure 4, the association named *Integrated* represents the list of samplers to be integrated by a particular instance of SamplerIntegrator, using what is known as *Delegation* [Szy98] in object-oriented systems design. In particular, it applies a design pattern known as *Command* [GHJV95], which makes the design of SamplerIntegrator more extensible and transparent, as indicated above. *CurrentSampler* refers to the sampler, of association *Integrated*, that performed the very last insertion into the Sample Database, that is, the one to be kept consistent by other samplers. Finally, association named *Integrator* represents the association with an instance of Sampler-

Integrator that is integrating this instance of DatabaseSampler with other samplers.

In summary, assume a set of samplers $Samp_1, \ldots, Samp_n$ are to be integrated. Using the design of *SamplerIntegrator* of Figure 4, the resulting sampler would iteratively construct a Sample Database as follows. Starting with an empty database, SDB_0, it would use $Samp_1$ to perform the initial insertion(s), leading to a new Sample Database SDB_1. Then it would iterate through all other samplers, $Samp_2, \ldots, Samp_n$, to maintain this insertion consistent, resulting in a set of databases $SDB_2, \ldots, SDB_n$. Each SDB_i would result from keeping the very last insertion of $Samp_{i-1}$ consistent with the criteria of $Samp_i$. This would be a recursive process, as database SDB_i would also need to be kept consistent with the very last insertion of $Samp_{i-2}$, then with $Samp_{i-3}$, and so on, each step resulting in a new database. This process leads to a set of increasingly larger databases $SDB_0 \subset SDB_1 \subset \ldots \subset SDB_m$ with the last one, SDB_m, consistent according to the criteria of all samplers $Samp_1, \ldots, Samp_n$.

The DSIM explained in this Section has been tested integrating two samplers which extract a sample of a database according to two orthogonal criteria:

Sampler with Referential Integrity Constraints and Functional Dependencies
: Samples a multi-table database so that the resulting sample satisfies sets of functional dependencies and referential integrity constraints simultaneously.

The resulting sampler enforces two completely different types of integrity constraints. In addition, it does so without any knowledge of how the integrated samplers are built. The DSIM is the same independently of whether each sampler is basic or integrated.

6 Summary and Future Work

Database sampling is a technique widely used in many different database applications when, for efficiency reasons, the entire database cannot be used. This paper has clearly identified the role database sampling plays in the area of database prototyping to support information system development. A framework for evaluation of database prototyping construction methods has

been presented, which has then justified the use of database sampling in the context of legacy information system migration.

The focus of this paper has been on what has been called *consistent database sampling*, that is, the process of sampling a database according to certain criteria, which are used to evaluate the consistency of the resulting Sample Database. A prototypical example of such criteria would be the satisfaction of sets of integrity constraints. In this context, the key issues involved in database sampling have been indentified by providing a general description of the Consistent Database Sampling Process (CDSP). Any implementation of this Process allowing for different types of criteria to be enforced in a modular and extensible fashion clearly requires the definition of a sampling Protocol each of these modules, termed here *Database Samplers*, must follow. This Protocol has been described, and the use of the Sampling Process together with the Sampling Protocol in order to allow different Database Samplers to co-operate has been analysed. This co-operation leads to Sample Databases satisfying the criteria enforced by several Database Samplers simultaneously.

The implementation of a tool, termed *Consistent Database Sampling Tool* (CDST), has been reported. This tool represents an implementation of all the above concepts: Sampling Process, Sampling Protocol, and co-operation between independently built Database Samplers. The co-operation between Database Samplers has been achieved by a combination of inheritance and delegation, leading to a design pattern which, although applied here to the context of database sampling, is generic.

Future work is currently focused on extending the existing prototype of a CDST so that more database samplers, which sample according to different criteria, will become available. This will widen the application domains where the tool can be used.

A promising line of research is the use of consistent database sampling in data mining. To the best of authors' knowledge only random sampling has been used to date in this context. By randomly sampling a database, some patterns can be lost as related data items may not always be sampled together. Consistent sampling will lead to Sample Databases which, as a whole, are more representative of the database being analysed than if random sampling is used. Therefore interesting patterns are more likely to be found in Consistent Sample Databases than in Test Databases. Consistent database sampling in data mining would lead to an iterative process as follows: results of an iteration of data mining would suggest criteria to be included in a

new sample of the database, this new sample, when mined, could reveal new patterns which could, in turn, be included in the extraction of a new sample.

References

[Bei95] B. Beizer. *Black-box testing : techniques for functional testing of software and systems.* Wiley, 1995.

[BG00] J. Bisbal and J. Grimson. Database prototyping through consistent sampling. In *the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR'2000).* Scuola Superiore Guglielmo Reiss Romoli (SSGRR), 2000.

[BLWG99] J. Bisbal, D. Lawless, B. Wu, and J. Grimson. Legacy information systems: Issues and directions. *IEEE Software*, 16(5):103–111, September/October 1999.

[BWLG98] J. Bisbal, B. Wu, D. Lawless, and J. Grimson. Building consistent sample databases to support information system evolution and migration. In G. Quirchmayr, E. Schweighofer, and T. J.M. Bench-Capon, editors, *Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA'98)*, volume 1460 of *Lecture Notes in Computer Science*, pages 196–205. Springer-Verlag, 1998.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns.* Addison-Wesley, 1995.

[GSE+94] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-record synthetic databases. In *Proceedings of ACM SIGMOD'94*, pages 243–252, 1994.

[KM94] J. Kivinen and H. Mannila. The power of sampling in knowledge discovery. In *Proceedings of the 1994 ACM SIGMOD-SIGACT Symposium on Principles of Database Theory (PODS'94)*, pages 77–85, 1994.

[MK97] F. Martin and S. Kendall. *UML distilled: applying the standard object modeling language*. Addison-Wesley, 1997.

[NML93] A. Neufeld, G. Moerkotte, and P. C. Lockemann. Generating consistent test data: Restricting the search space by a generator formula. *VLDB Journal*, 2(2):173–213, April 1993.

[Olk93] F. Olken. *Random Sampling from Databases*. PhD thesis, University of California, April 1993.

[Som95] I. Sommerville. *Software Engineering*. Addison-Wesley, fifth edition, 1995.

[Szy98] C. Szyperski. *Component Software*. Addison-Wesley, 1998.

[WJ91] E. J. Weyuker and B. Jeng. Analyzing partition testing techniques. *IEEE Transactions on Software Engineering*, 17(7):703–711, July 1991.

[WLB+97] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, and D. O'Sullivan. The butterfly methodology: A gateway-free approach for migrating legacy information systems. In B. Werner, editor, *Proceedings of the 3rd IEEE Conference on Engineering of Complex Computer Systems (ICECCS'97)*, pages 200–205. IEEE Computer Society Press, 1997.