Converting Informal Meta-data to VDM-SL: A Reverse Calculation Approach

F.L. Neves, J.C. Silva & J.N. Oliveira [⋆]

Informatics Department, University of Minho
Braga, Portugal

VDM in Practice!
A Workshop co-located with FM'99: The World Congress on Formal Methods
Toulouse, France, 20-21 September 1999

1 Introduction

Enterprise competitiveness in the information age is very much dependent on the quality of the underlying information systems. These provide crucial support to as important tasks as strategic business planning and decision support. Information system quality is, in turn, highly dependent upon consistency and reliability of stored data. However, it is hard to maintain the quality of fast-growing data in loosely structured information systems. These very rapidly become infected with so-called “dirty” data, a problem nowadays identified under the *data quality* heading.

In the last few years, many companies around the world have spent large amounts of resources on process re-engineering encompassing both applications and data repositories, in order to face the growing interest in *data mining* [1], *data warehouse* [12] and *Web marketing* systems [25].

The authors are currently engaged in a R&D project (KARMA) which aims at addressing data-quality from a *formal-method* viewpoint. This includes the adoption of formal techniques for meta-data representation and (reverse) calculation of data intensive applications. The KARMA-consortium involves three software houses [1] which contribute to the project with their experience in large data purification contracts.

2 About the KARMA Project

The system currently under development in KARMA builds upon past experience gathered in an academic reverse engineering project [21] and some experiments in software reuse [18] involving formal reverse data-structure calculation.

[⋆] Dep. Informática, Universidade do Minho, Campus de Gualtar, 4700-320 Braga, Portugal. The research described in this paper has been funded by the Portuguese Science and Technology Foundation under Research Contract KARMA (P060-P31B-09/97).

[1] These are NOVABASE PORTO *Ltd*, NOVABASE LISBON *Ltd* and SIDEREUS *Ltd* (Porto).

The adopted formal calculus [20, 22] has been developed as an alternative to standard normalization theory, framing database design into the wider area of *data refinement* [15]. Informal data models such as described by E-R diagrams, for instance, are turned into systems of datatype definitions in a systematic way [24]. Integrity constraints and business rules (which are so dear to the "data cleansing" practitioner) are identified with *abstraction invariants* [16] and *datatype invariants* [15], respectively, whose structural synthesis (analysis) by calculation is precisely the core of the calculus [22] and can be animated at rapid-prototyping level.

A prototype has emerged from the formal specification of a particular tool we have been designing for the KARMA software system. The ultimate goal of this tool is to deliver concise formal descriptions — written in the ISO standard formal notation VDM-SL [13, 8] — out of informal or poorly structured meta-data.

3 A Tool for Deriving VDM-SL from Informal Meta-data

Our experiments with the current prototype version of the KARMA toolset include a realistic example taken from [21]: the process of reversing the information system which supports, in ORACLE technology, the operation of the Student Records Office of Minho University. Out of a poorly documented relational database schema consisting of 53 tables operated by more than 230 units of code, the system delivers a compact VDM-SL description consisting of only 14 high-level finite-mapping structures.

It should be stressed that this tool is not fully automatic. It requires user guidance with respect to the selection of the formal laws of the calculus which should be applied in each particular step. In many situations this is actually a complex decision, as can be briefly explained: the calculus consists of inequations of the form $A \leq B$ (read: "*data type B implements, or refines data type A*") which abbreviate the fact that there is a surjection $A \xleftarrow{f} B$ (the *abstraction function*) which has a canonical right-inverse $A \xrightarrow{r} B$ (the *representation function*). That is to say, $f \cdot r = id_A$ holds [2]:

$$\begin{tikzcd} A \arrow[r, bend left, "r"] \arrow[r, phantom, "\leq"] & B \arrow[l, bend left, "f"] \end{tikzcd}$$

Let ϕ denote the *concrete invariant* which emerges as characteristic predicate of the range of r, that is to say, $\phi\, b$ holds wherever there exists at least one $a \in A$ such that $b = r\, a$. Because r is always injective, one can write $A \cong B_\phi$, where B_ϕ denotes the subset of B which satisfies concrete-invariant ϕ. So the replacement of low-level structure B by abstract structure A is safe provided ϕ is known to hold in the particular situation in hand — a kind of reverse specification via "concrete invariant discharge".

There may exist two or more candidate laws for invariant discharge, for instance $C \cong B_\psi$ competing with $A \cong B_\phi$. It is clear that B should be reversed either to A in case ϕ holds or to C in case ψ holds. Let us see a very short illustration, expressed in VDM-SL notation: a pair of relational tables sharing the same primary-key type,

[2] id_A denotes the identity function on data type A.

```
    CMOD :: T1: map A to B
            T2: map A to C;
```

(CMOD stands for “concrete model”) may either be isomorphic to

```
    AMOD1 = map A to BandC;
    BandC :: A1: B
             A2: C;
```

(AMOD1 stands for “abstract model 1”) in case concrete invariant

```
            inv mk_CMOD(t1,t2) == dom t1 = dom t2
```

happens to hold over CMOD, or be isomorphic to

```
    AMOD2 = map A to BorC;
    BorC = CaseB | CaseC;
    CaseB :: K: B;
    CaseC :: K: C;
```

in case concrete invariant

```
    inv mk_CMOD(t1,t2) == (dom t1) inter (dom t2) = {}
```

happens to hold over CMOD (AMOD2 stands for “abstract model 2”). These facts are concisely expressed in the “algebraic” notation of the calculus [19, 20] as follows, where the names of the relevant abstraction and representation functions are made explicit [3]:

$$\begin{tikzcd} A \rightharpoonup B \times C \arrow[r, bend left, "unjoin"] \arrow[r, phantom, "\leq"] & (A \rightharpoonup B) \times (A \rightharpoonup C) \arrow[l, bend left, "join"] \end{tikzcd} \tag{1}$$

ie. the fact which justifies the reversing of CMOD into AMOD1, and

$$\begin{tikzcd} A \rightharpoonup (B + C) \arrow[r, bend left, "uncojoin"] \arrow[r, phantom, "\leq"] & (A \rightharpoonup B) \times (A \rightharpoonup C) \arrow[l, bend left, "cojoin"] \end{tikzcd} \tag{2}$$

ie. the one underlying the reversing of CMOD into AMOD2.

[3] The finite-mapping datatype (bi)functor $A \rightharpoonup B$ translates to `map A to B` in VDM-SL. Functions $join$ and $unjoin$ are the finite-mapping instantiations of polytypic functions zip and $unzip$, respectively, see [14] and [23]. A library of about fifty such functions, written in the CAMILA [3] formal notation, is available from the calculus web-site `http://www.di.uminho.pt/˜jno/html/setshp.html`.

Relational database reverse-specification requires a relatively modest set of data-transformation laws [4]. Besides the two laws mentioned above, the *dom* function witnesses a very useful isomorphism between finite sets and partial finite mappings,

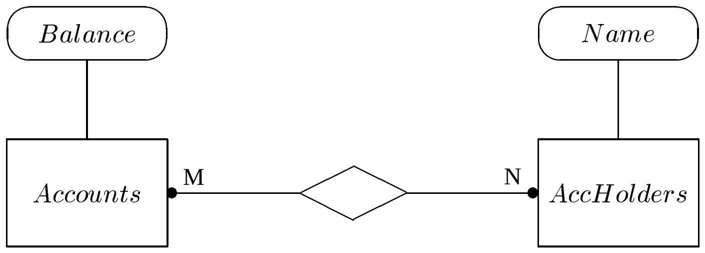
$$\begin{tikzcd} 2^A \arrow[r, bend left, "set2fm"] \arrow[r, phantom, "\cong"] & A \rightharpoonup 1 \arrow[l, bend left, "dom"] \end{tikzcd} \tag{3}$$

which expresses the equivalence between VDM-SL data models `set of A` and `map A to nil`. Another law which plays a prominent rôle in relational data formal calculation is

$$\begin{tikzcd} A \rightharpoonup (B \times (C \rightharpoonup D)) \arrow[r, bend left, "unnjoin"] \arrow[r, phantom, "\leq"] & (A \rightharpoonup B) \times (A \times C \rightharpoonup D) \arrow[l, bend left, "njoin"] \end{tikzcd} \tag{4}$$

From left to right one infers composite keys out of nested finite mappings. From right to left (reverse direction) it merges two tables which share a common (sub)key.

Let us see a very simple "toy" example. Suppose that we want to reverse the database structure of a naïve "bank account management system" informally described by the following E-R diagram:



This diagram is easy to express in terms of the KARMA formal meta-data standard. If we ask the tool to output the VDM-SL model corresponding to this diagram we get three relations (*ie.* sets of tuples) [5]:

```
----------------------------------------------------
-- VDM-SL type & data of Sets Abstract model.
----------------------------------------------------
-- Generated Automatically by KarmaSets tool.
```

[4] The synthesis of recursive data-models from relational, "flat" models, which is a standard result of the calculus [20], has not yet been incorporated in the tool.

[5] The particular choice of identifiers is not totally obvious and has to do with some details of the tool which are described later.

```
-----------------------------------------------------

types

Sets ::
      P1: set of Inf_1
      P2: set of Inf_2
      P3: set of Inf_3 ;

Inf_1 ::
      P1: AccountId_01
      P2: Balances_meta01 ;

Inf_2 ::
      P1: AccountId_01
      P2: AccHolderId_02 ;

Inf_3 ::
      P1: AccHolderId_02
      P2: Details_meta02 ;

Balances_meta01 ::
      P1: Balance;

Details_meta02 ::
      P1: Name;

AccountId_01 = seq of char;
AccHolderId_02 = seq of char;
Balance = seq of char;
Name = seq of char;
```

At any stage the tool can be asked to output the VDM-SL specification of the current version of the model under reversal. Once the integrity constraints of the diagram are taken into account, the main body of the model is converted to

```
Sets ::
      P1: Inf_1
      P2: map AccHolderId_02 to Details_meta02  ;

Inf_1 ::
      P1: map AccountId_01 to Balances_meta01
      P2: map Inf_2 to nil  ;

Inf_2 ::
      P1: AccountId_01
      P2: AccHolderId_02 ;
```

The two occurrences of attribute `AccountId_01` provide an opportunity to trigger law (4) above. The tool will therefore re-write the model to

```
Sets ::
      P1: map AccountId_01 to Inf_1
      P2: map AccHolderId_02 to Details_meta02  ;

Inf_1 ::
      P1: Balances_meta01
      P2: map AccHolderId_02 to nil  ;
```

Finally, law (3) can be applied to the second component of `Inf_1` and we obtain

```
Inf_1 ::
      P1: Balances_meta01
      P2: set of AccHolderId_02  ;
```

This is in fact how one would specify the "toy" bank account system above: two finite mappings, one describing the balance and set of account holders assigned to each account identifier, and the other one describing the details of account holders [6].

4 Application to Legacy Information Systems

Legacy information systems are normally poorly documented. Reverse-specification decisions such as the ones required above can only be made either by informal inspection (interviews with systems users) or by concrete-invariant exception statistics. In [21] the former approach was used because the whole task was carried out manually. With the advent of the KARMA formal standard for meta-data and of the associated tool [7] we started experimenting with the latter, which is in fact the one adopted (albeit very informally) by the companies of the KARMA consortium. The tool keeps track of both the "abstraction function so far" and of the associated representation function. The prototype makes it possible to bind sample data to its meta-data. By applying, for each candidate law, $r \cdot f$ to the sample data one is able to filter invalid records out and decide upon which law to adopt on a statistical basis (the less number of invalid records the better).

Some extra functionality is available which is of great help in coping with large information systems' meta-data and addresses a very important question: from which relational tables should one start the analysis? First, attributes which do not participate in the referential integrity of the information system are factored out as "meta" attributes (*eg.* `Details_meta02`). Based on the transitive closure of the graph which expresses such a referential integrity (this basically records the intertwining among primary and foreign keys), one is able to "cluster" closely related meta-data and restrict the analysis to each particular cluster. Clusters are identified by numbers which become apparent as figures which the tool attaches to the output VDM-SL identifiers (*eg.* `01` in `AccountId_01`).

[6] See [2] for details of the opposite forward engineering calculation, *ie.* the derivation, from this specification, of the relational model which we started from.

[7] Currently available as a prototype (about 8000 lines of CAMILA code, including some test data).

For instance, in the Student Records database one gets a fairly large cluster (20 tables, that is 38%) having to do with students, a second one (9 tables, 17%) having to do with courses and then a collection of smaller clusters corresponding to less relevant information structures.

5 Summary and Current Work

Reference [21] had shown how to obtain a formal reverse engineering discipline for free, simply by reversing the order of application of the laws of a data refinement calculus [20]. But this still required a lot of informal work and interaction with the target information system maintenance team.

The KARMA project has merged such a theoretical background with the experience gathered in the very large data quality contracts which the companies of the consortium have been involved into, in recent years. These companies expect to gain competitiveness on the methodological side, even within the scope of existing tools.

On the technical side, we would like to provide mechanical support for two particular aspects of the formal reverse engineering discipline sketched in [21]. First — and based on former work on formally specifying temporal information systems [10, 4] — the ability to spot and adequately reverse the *temporal dimension* of information. Second, we would like to combine formal reverse engineering with object-oriented re-engineering. The inference of an object-oriented data-model from a formal model, as sketched in [21], could be expressed in VDM++ [11] or combined with work on formalizing UML, for instance [5].

Last but not least, we would like to move further to reversing the operations (transactions). The clustering process on operations is described in [21]. The definition of an algebra supporting operation reasoning is, however, still incomplete and calculations are too lengthy — see *eg.* the algebra and calculations on *selective updating* presented in [22].

6 Related Work

Most work in the literature refers to formal reverse engineering of algorithmic code, in particular resorting to wp-calculi, see eg. [7, 9].

A clever use of type-inference techniques to *decompile* (that is, reverse engineer) C programs from target machine code is described in [17]. Type reconstruction is based on Milner's algorithm and includes the process of *data structure reconstruction*. As happens in our approach, this provides evidence that reverse engineering should "always" be preceded by datatype *reconstruction* — isn't *forward* engineering always based on datatype *construction*, after all?

Other applications of formal methods in the OO-analysis of existing software (eg. [6]) include the use of PVS (Prototype Verification Systems) formal tools but do not have a calculation flavour.

References

1. P. Adriaans, D. Zantinge, and Peter Adriaans Syllogic. *Data Mining*. Addison-Wesley, 1996.
2. J. J. Almeida, L. S. Barbosa, F. L. Neves, and J. N. Oliveira. Bringing CAMILA and SETS together — the `bams.cam` and `ppd.cam` CAMILA Toolset demos. Technical report, DI/UM, Braga, December 1997. *[45 p. doc.]*.
3. J. J. Almeida, L. S. Barbosa, F. L. Neves, and J. N. Oliveira. CAMILA: Prototyping and refinement of constructive specifications. In M. Johnson, editor, *Algebraic Methodology and Software Technology*, pages 554–559. Springer LNCS, December 1997. 6th International Conference, *AMAST'97*, Sydney, Australia, 13–17 December 1997, Proceedings.
4. L. S. Barbosa. The monadic structure of time. Technical report, Dep. Informática, University of Minho, 1996.
5. J.M. Bruel and R.B. France. Transforming UML models to specifications. In *OOPSLA'98 Workshop on Formalizing UML. Why? How?*, 1998.
6. B.H.C. Cheng and Brent Auernheimer. Applying formal methods and object-oriented analysis to existing flight software. Technical report, DCS, Michigan State University and DCS, California State University, 1993.
7. B.H.C. Cheng and G.C. Gannod. A two-phase approach to reverse engineering using formal methods. In *Formal Methods in Programming and Their Applications*. Springer-Verlag, 1993. Lecture Notes in Computer Science.
8. J. Fitzgerald and P.G. Larsen. *Modelling Systems: Practical Tools and Techniques* . Cambridge University Press, 1st edition, 1998.
9. G.C. Gannod and B.H.C. Cheng. Using informal and formal techniques for the reverse engineering of C programs. Technical report, DCS, Michigan State University, 1996.
10. INESC 2361 Group. Temporalização da NBDC. Technical report, © SONAE Company, Maia, May 1996. Consultancy report (in Portuguese).
11. The VDM Tool Group. The IFAD VDM++ language. Technical Report IFAD-VDM-44, IFAD, Forskerparken 10, DK-5230 Odense M, Denmark, September 1997.
12. Claudia Imhoff and Jon Geiger. Data quality in the data warehouse. In *Data Management Review*. April 1996.
13. ISO. Information technology — programmming languages, their environments and system software interfaces — Vienna Development Method — specification language — part 1: Base language, Dec. 1996. (ISO/IEC 13817-1, Geneva).
14. P. Jansson and J. Jeuring. Polylib — a library of polytypic functions. In *Workshop on Generic Programming (WGP'98), Marstrand, Sweden*, 1998.
15. C. B. Jones. *Software Development — A Rigorous Approach*. Series in Computer Science. Prentice-Hall International, 1980. C. A. R. Hoare.
16. C. Morgan. *Programming from Specification*. Series in Computer Science. Prentice-Hall International, 1990. C. A. R. Hoare, series editor.
17. A. Mycroft. Type-based decompilation. In S. D. Swierstra, editor, *ESOP'99 - European Symposium On Programming*, Lecture Notes in Computer Science. Springer, 1999.
18. F. L. Neves and J. N. Oliveira. Software Reuse by Model Reification . In *WISR'95 - 6th Annual Workshop on Software Reuse*, August 28–30 1995. Charles Il, Illinois, USA.
19. J. N. Oliveira. *A Reification Calculus for Model-Oriented Software Specification* . *Formal Aspects of Computing*, 2(1):1–23, April 1990.
20. J. N. Oliveira. *Software Reification using the SETS Calculus* . In *Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK*, pages 140–171. Springer-Verlag, 8–10 January 1992. (Invited paper).
21. J. N. Oliveira. A calculational approach to reverse specification, 1997. Seminar presented at *UNU/IIST*, Macau, May 13th, 1997, 22 p. [8].

[8] Available from `http://www.di.uminho.pt/~jno/ps/unu97s4.ps.gz`.

22. J. N. Oliveira. A data structuring calculus and its application to program development, May 1998. Lecture Notes of M.Sc. Course (150 p. [9]). Maestria em Ingeneria del Software, Departamento de Informatica, Facultad de Ciencias Fisico-Matematicas y Naturales, Universidad de San Luis, Argentina.
23. J. N. Oliveira. *'Fractal' Types: an Attempt to Generalize Hash Table Calculation.* In *Workshop on Generic Programming (WGP'98), Marstrand, Sweden*, June 1998.
24. J. N. Oliveira and A. M. Cruz. Formal calculi applied to software component knowledge elicitation. Technical Report C19-WP2D, DI/INESC, December 1993. IMI Contract *C.1.9. Sviluppo di Metodologie, Sistemi e Servizi Innovativi in Rete*. [10].
25. Jim Sterne. *World Wide Web Marketing*. John Wiley & Sons, 1998.

[9] Available from `http://www.di.uminho.pt/˜jno/ps/sanl98.ps.gz`.
[10] Available from `http://www.di.uminho.pt/˜jno/ps/c1-wp2b.ps.gz`.