Standards for Assured Legacy Systems

B.Nick Rossiter
Computing Science
Newcastle University, NE1 7RU

Michael Heather
University of Northumbria at Newcastle, NE1 8ST, UK

February 17, 1997

Abstract

A formal model, based on the ISO standards for Information Resource Dictionary Systems (IRDS), is developed with the aim of providing a complete definition of an information system from the physical data values held to the concepts employed for data and function representation and real–world abstractions. It is discussed how such a multi–level model can be used to control the evolution of information systems by creating an environment where heterogeneous systems can be compared. Current trends towards more structured programming techniques and more disciplined software engineering environments lead to the potential for considerable benefits from using an IRDS approach. This potential, however, will only be realized if a formal underpinning of the IRDS standard is achieved.

About the author

Nick Rossiter is lecturer in the Department of Computing Science with particular interests in databases and systems analysis.

Michael Heather is senior lecturer in law where he has been responsible for computers and law since 1979.

Suggested Keywords

legacy systems, standards, reference model, data dictionaries, categorical representations.

1 Background

With continuous evolution of existing systems, the discontinuance of old and the continual creation of new ones, a user needs to rely on a persistency that can transcend any particular system. This can only be satisfied if exact knowledge of any system is available. This is exact knowledge in the sense of mathematics and needs to be expressed in a language that can cope with large quantities of complex data over long periods and in different contexts. This requires the existence of some higher–level reference system which exhibits these characteristics with the necessary persistency[1].

In the simplest case it is always necessary to go one level up to address the physical level from a logical standpoint. A minimum requirement is to know names, constraints, typing, etc. In modern database systems, this is the function of the system catalogue which in a relational system includes descriptions of the relation names, attribute names, attribute domains or types, key attributes, other constraints, views, storage structures and indexes. In a network database, information held in the catalogue would include descriptions of record types, set types and physical organization. In a COBOL program, the data division could be held in a catalogue giving information on record types, field types and their pictures.

Today nearly all system catalogues are *active* (rather than passive) in the sense that they are an automatic source of naming and typing information for programs accessing the system. This means that the catalogues are the cornerstone of the information system often being subjected to a very high rate of searching activity. Clearly a major advantage of active catalogues is that system type changes are made once in the catalogue rather than replicated through many source programs.

However, a logical level is incomplete without an associated formal reference level to describe how it represents real–world entities, properties, relationships, typing, etc. This reference level deals with meta–meta data. Another way of looking at this level is to view it as dealing with the policy of the model: why is a particular construction in the model available and what semantic capability does it handle?

This comparison of logic and principles is performed against a reference or neutral layer which contains a superset of the facilities in the application models. The reference layer contains as data the principles for data and activity representation and for

[1]These ideas on the reference data model were first presented in a paper given at the Computing and Control Division Colloquium held by the Institution of Electrical Engineers at Savoy Place, London, on Tuesday 13 December 1994 on the topic Legacy Information System-Barriers to Business Process Re-engineering and organized by Professional Group C3 (Information systems and networks):
Rossiter, B N, & Heather, M A, The Reference Data Model applied to
Migrating Information Systems, Digest No: 1994/246 7/1 IEE, London.
The further significance of this approach for legacy systems is being published as the first chapter in the monograph:
Rossiter, B N, & Heather, M A, Data Modelling for Migrating Information
Systems in: Legacy Systems, A W Booth (ed), UNICOM, London.

each application model the concepts which capture these principles. There is therefore a mapping from principles to schema intension for each application model in turn.

In relational systems with all data held in tables, the meta–meta level is implicitly part of the model. The freeness of the object–oriented paradigm, on the other hand, means that the meta–meta level needs to be constructed explicitly to control the representation of classes, objects, properties, inheritance, composition, methods, etc. In a COBOL system, *wrapper* constructions are increasingly used to provide encapsulation of programs with pre–determined interfaces to emulate object constructions. However, these wrappers are at the meta level: the meta–meta level would describe the purpose of wrappers and the facilities used for their construction.

Figure 1: Levels and the IRDS

These are particular examples of the message level which provides a top layer of consciousness which has to be developed further to deal with the widest possible type of information system. It was for this kind of purpose that the ANSI Standard Reference Model (X3.138) was developed in the 1980s [1], emerging in 1993 [2] as an associated part of the standard for a framework for Information Resource Dictionary Systems (IRDS) developed in 1990 and 1993 [3]. Language bindings can in principle be defined for any language and among the early specifications were Pascal and SQL.

The IRDS standard provides for four levels given below with our interpretation:

1. the meta–meta or policy level defining facilities available in terms of computing

science abstractions (aggregation, composition, inheritance, association, etc);

2. the meta or construction level defining the techniques available in a particular system for representing the abstractions at the policy level;
3. intension level naming data instances and types and providing constraints to satisfy the application users;
4. extension level holding data values conforming to the intension.

As an example, in the relational model, level 1 would be real–world abstractions (such as aggregation), 2 would be constructs available (such as tables and attributes), 3 would be data names (such as table names) and 4 would be data values.

Between each level there is a mapping, termed a *level pair*, which enables data at one level to be related to data at a lower level as shown in Figure 1. In the same relational example, the level pair between levels 1 and 2 would be the association of an abstraction, say aggregation, with a constructive facility in the model, say table in this case; the level pair between 2 and 3 would be the association of a construction in the model, say table, with the name of a table; and the level pair between 3 and 4 would be the association of a data name with a data value.

In this paper, we describe how a formal representation of the IRDS can be used to control transformations in migrating legacy systems, written in either a database or programming paradigm, with consistency and integrity. First we give an informal view of our work.

2 Levels and Mappings

We construct an IRDS at four levels:

1. The top level is IRDDS, in which concepts relating to policy and philosophy are defined; for example, object–oriented abstractions would be declared at this level. In principle, only one instance of an IRDDS need be defined for an organization: there is only one collection of such concepts but with the open–ended nature of object–oriented structures, some extensibility is required.
2. The second level is IRDD in which schema facilities are defined. There will be one definition of IRDD for each paradigm employed in an organization, one for COBOL, another for SQL, etc.
3. The third level is IRD which defines the intension for an application, giving names and constraints. There will clearly be many intensions defined in an organization, one for each application.

4. The fourth level is APP which gives the extension, the data values. There will be one extension for each intension, the values being consistent with the names and constraints of the intension.

Between each level there is a mapping as shown in Figure 2:

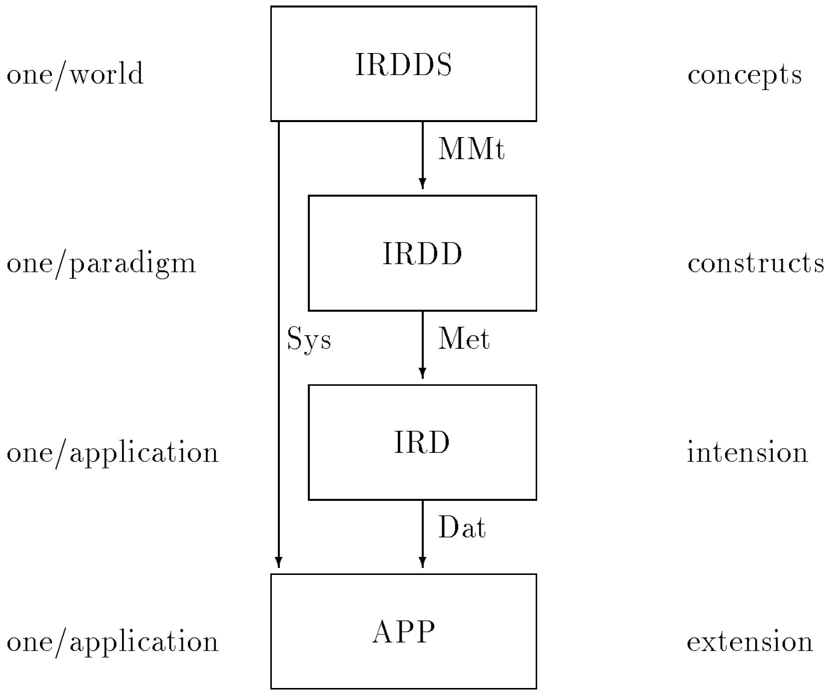

Figure 2: The IRDS in Schematic Form

- Between levels 1 and 2 (IRDDS and IRDD), there is the mapping MMt acting as a level pair. This level pair exists only in IRDS–type systems in which constructive facilities in a system are related to real–world abstractions.
- Between levels 2 and 3 (IRDD and IRD), there is the mapping Met acting as a level pair. This level pair provides a standard data dictionary function of, for instance, saying which tables are available in a relational system.
- Between levels 3 and 4 (IRD and APP), there is the mapping Dat acting as a level pair. This level pair can be thought of as the normal function of an information system: to link values to names so that data can be addressed by name rather than by physical location.

- Between levels 1 and 4 (IRDDS and APP), there is the mapping Sys acting as a level pair. This level pair short–circuits the navigation through four levels by giving a direct mapping from real–world abstractions to data values.

The IRDS standard is the basis for relating heterogeneous systems, that is systems based on different paradigms. While there is only one instance of the top level (the IRDDS), this level is extensible and new concepts and abstractions can be added as desired. From the point of view of legacy systems, the IRDS provides the ability to run an organization with many different paradigms all integrated through the type of structure shown in Figure 2. The critical mapping is Sys, that is the arrow from IRDDS to APP, relating concepts to values. By determining this mapping for all types of system, the problem of migrating legacy systems is avoided to some extent as all types of approach to information systems can be accomodated and run in an integrated fashion. In this sense, the knowledge gained in managing federated database systems is clearly relevant [4].

Our next task is to formalize the diagram in Figure 2 so that a sound scientific basis can be developed for our IRDS model to handle heterogeneous systems.

3 Categorical IRDS Constructions

Constructive mathematics attempts to develop logically what works in practice and can perhaps provide the necessary universality for interoperability of heterogeneous data systems with consistency and quality assurance in the real–world. Category theory [5] is particularly appropriate for modelling multi–level relationships for it is essentially concerned with links between objects.

Category theory provides the universal construction which we have used at Newcastle for formalizing information systems. An example is the development of the Categorical Product Data Model (CPDM) which can formalize, for instance, the object–relational model [6]. A brief introduction to categories, functors and natural transformations is given in Appendix I.

In the CPDM data model, each class definition can be identified as a collection of arrows (functions) forming a category **INT** and each family of object values conforming to a particular class definition as a category **EXT**. The mapping from the intension (class definition) to extension (object values) is made by a functor *Dat* which enforces the various constraints specified in **INT**. Category **INT** is the intension corresponding to the third level in IRDS and **EXT** is the extension corresponding to the fourth level.

In CPDM the intension category **INT** is actually a family of categories, representing definitions of classes, associations (relationships) and coproduct structures indicating inheritance hierarchies. The arrows within it may be methods as in object–based systems, logical connections as in network databases, or functional dependencies as in relational database schemas. It should be emphasised that categorical approaches naturally include procedures and functions through the underlying arrow concept ensuring that both structure and activity can be modelled in a multi–level manner. The category **EXT** is also a family of categories, representing object values and association instances. The functor D mapping from the intension to the extension not only connects a name to its corresponding set of values but also ensures that constraints specified in the schema, such as functionalities of relationships, membership classes and functional dependencies, hold in the extension.

It is relatively straight–forward in category theory to extend the product data model CPDM, defined already as a two–level structure, in a universal manner to handle the four levels of IRDS. Each of the four levels of IRDS is defined as a category. Between each level there is a higher–order function, termed a functor, which ensures that certain consistency requirements are met in the mapping between the source and target categories. The abstractions level (top) is a category **IRDDS** which defines the various abstractions available for modelling real–world data. The next level is a category **IRDD** defining the various construction facilities available for representing abstractions and data in a particular system. There is therefore, for one instance of **IRDDS**, many instances of **IRDD**, one for each type of model (relational, network, COBOL, etc).

The meta–meta data functor pair (level pair) MMt maps target objects and arrows in the category **IRDDS** to image objects in the category **IRDD** for each type of system. This mapping provides at the meta-meta level the data for each kind of system, that is to say how each abstraction is to be represented. We also define a functor pair Met relating for each system the constructions in **IRDD** with the names in a particular application in **IRD**. Combining these new constructions with our product ones above and renaming our **INT** as **IRD** and **EXT** as **APP**, we have the following direct and universal representation of IRDS as shown in Figure 3.

$$\mathbf{IRDDS} \overset{MMt}{\rightleftarrows} \mathbf{IRDD} \overset{Met}{\rightleftarrows} \mathbf{IRD} \overset{Dat}{\rightleftarrows} \mathbf{APP}$$

Figure 3: IRDS Levels in Functorial Terms

In category theory, we can consider the above to be a composition of functors with Sys as the overall functor from **IRDDS** $\longrightarrow$ **APP**, such that for each type of information system:

$$Sys = Dat \circ Met \circ MMt$$

In order to relate concepts in different models, we need to compare the functors Sys : **IRDDS** $\longrightarrow$ **APP** for each of our COBOL, network, relational, object–oriented systems, etc. This comparison is a natural transformation, say η, if for each arrow $f : A \longrightarrow B$ in **IRDDS** the diagram in Figure 4 commutes where we distinguish the two systems as Sys and Sys'. A diagram commutes when all the different paths between any two objects give the same result.

$$\begin{tikzcd}
Sys(A) \arrow[r, "\eta_A"] \arrow[d, "Sys(f)"'] & Sys'(A) \arrow[d, "Sys'(f)"] \\
Sys(B) \arrow[r, "\eta_B"'] & Sys'(B)
\end{tikzcd}$$

Figure 4: Commuting Target Square with Covariant Natural Transformation

4 Discussion

We envisage that constructions of the IRDS in systems based on CPDM offer much potential for improving our ability to deal with heterogeneity. The universal basis gives us a predictable and provable behaviour which is such an important aspect if legacy systems are to be reliable and productive. Essentially the formal basis for IRDS assists in developing a reference model to relate policy statements in different models and environments with consistency and integrity. This is the crux of legacy systems.

There are ways in which a categorically–based IRDS would be more powerful than the ones envisaged in the standards. For instance Spurr [7] comments on the difficulties of using the IRDS standard dictionary with CASE tools because of the lack in IRDS of a natural way of modelling object structures. Our work [6] has shown the natural correspondence between categorical databases and object structures so that a complete and faithful representation of the object–oriented paradigm should be possible throughout the four levels.

In addition, category theory provides natural typing facilities for the effective management of a data dictionary using the concept of a terminal object, which is an object with an arrow to it from every other object in the category in which it is held. For example, denoting the terminal object in a category $\mathbf{I}$ by $\mathbf{1_I}$, then $\mathbf{1_I}$ types a database entity in general, $\mathbf{1}_{\mathbf{I}_1}$ types the database entity *suppliers* and $\mathbf{1}_{\mathbf{I}_2}$ types the database entity *parts* with $\mathbf{I_1}$ as the class of *suppliers* and $\mathbf{I_2}$ the class of *parts*.

A potential difficulty with our approach is that it assumes a complete knowledge of all features of every type of system is available. This may well be true for systems developed by current software engineering techniques but is unlikely to be true for some of the worst examples of old legacy systems. The top level IRDDS is always available but we cannot assume there will be an IRDD available for all paradigms. There is clearly a limit to the range of paradigms that could be covered in IRDD structures. This limit is governed by the extent to which a paradigm can be conceptualized. IRDD are likely to be defined in reasonable detail provided the paradigm used is not too idiosyncratic, has clearly–defined schema facilities and these have been used in the applications. Old COBOL programs, where data and program divisions are defined, should present no problem here but assembler programs with *ad hoc* data definition structures may contain too little logical structure to be adapted to our architecture.

Even if we have a paradigm clearly–defined by an IRDD, there may still be difficulties if, in the original system, there are problems with incomplete or conflicting schema information or code has been written bypassing the conceptual tools of a system.

We therefore see our techniques being more useful in coping with well–defined legacy systems, even if they are using outdated paradigms, than poorly–engineered historical software where the case for a complete rewrite is obviously stronger. The improvements in languages, for instance Microfocus COBOL and Visual BASIC, and the greater emphasis on formal schema design in current software engineering both indicate that future problems will be more amenable to our four–level approach.

It is therefore suggested that current trends towards more structured programming techniques and more disciplined software engineering environments should be leading to greater use of an IRDS approach. The full potential of such a strategy, however, will only be realized if a formal underpinning of the IRDS standard is achieved.

Our conclusion is that, for situations where legacy software is coherently designed and implemented, there is no other way forward than by this categorical approach or an equivalent framework. Legacy systems presuppose universal techniques to ensure consistency and integrity across time and distribution. A formal underpinning is needed of the four–level IRDS standard.

At present we are implementing a prototype [8] of CPDM using the platform of P/FDM [9], an implementation of the functional database model in Prolog, from Aberdeen University. A demonstrator project is planned to illustrate the advantages of our approach and to explore further the management of the four–level architecture for assisting in the control of legacy systems.

5 References:

1. D.Gradwell, Developments in Data Dictionary Standard, *Computer Bulletin*, September 1987.

2. Information technology – *Reference Model of Data Management*, Standard ISO/IEC 10032 (1993).

3. Information technology – *Information Resource Dictionary System (IRDS) framework*, Standard ISO/IEC 10027 (1990); 10728 (1993).

4. N.J.Fiddian, W.A.Gray, A.Ramfos & A.Cooke, Database Meta-Translation Technology: Integration, Status and Application, *Database Technology* **4**, 259–263 (1992).

5. M.Barr & C.Wells, *Category Theory for Computing Science*, Prentice–Hall (1990).

6. B.N.Rossiter, D.A.Nelson & M.A.Heather, *The Categorical Product Data Model as a Formalism for Object–Relational Databases,* Technical Report, Computing Science, Newcastle University, no.505, 41pp (1995).

7. K.Spurr, CASE Tools, Does the ISO Standard IRDS provide Sufficient Support? in: *Fourth Generation Systems*, ed. S.Holloway, Chapman and Hall, Unicom 36–47 (1990).

8. Nelson, D A, & Rossiter, B N, Prototyping a Categorical Database in P/FDM. *Proceedings of the Second International Workshop on Advances in Databases and Information Systems (ADBIS'95)*, Moscow, 27-30 June 1995, Springer-Verlag Workshops in Computing, edd. J. Eder and L.A. Kalinichenko, ISBN 3-540-76014-8, 432–456 (1996).

9. P.M.D.Gray, K.G.Kulkarni & N.W.Paton, *Object–Oriented Databases: A Semantic Data Model Approach*, Prentice Hall (1992).

Appendix I: Brief Review of Category Theory

Category theory is based not on the set as a fundamental but on the concept of a morphism, generally thought of as an arrow and represented by $\longrightarrow$. The arrow represents any dynamic operation or static condition and can cope therefore with descriptive/ prescriptive equivalent views. For example, the arrow is a generalization of mathematical symbols like $=, \in, \subset, \leq, f(x), \ldots$ with the usual respective meaning of equality, membership, partition, comparison, functional image, etc.

The arrow can never be free–standing: it must have some source and target, often conveniently named domain (dom) and codomain (cod) respectively. A category is a collection of arrows.

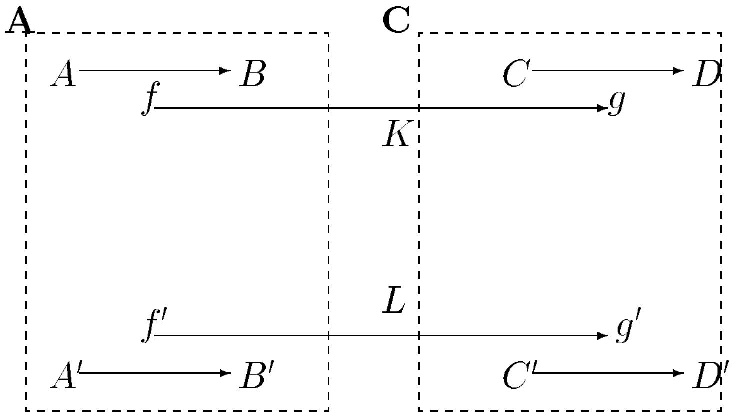

Figure 5: Functors compare Categories

The basic constructs of category theory are quite simple:

1. The identity arrow 1_A identifies an object A. That is $1_A : A \longrightarrow A$.
2. Arrows are composable if the codomain of the one forms the domain of the other.
3. Identity arrows can be distinguished by unitary composition with some arrow f.
4. Composition of arrows is associative. Arrows may be composed so that the codomain of one arrow may become the domain of another. Standard category theory requires composition to be associative. For the arrows:

$$A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D \xrightarrow{i} E$$

$$i \circ (hgf) = (ih) \circ (gf) = (ihg) \circ f$$

Conventionally then a category in this context is a collection of arrows between objects which may be named. In Figure 5, we show a category **A** with two arrows f and g. Where categories are given names, we use the convention throughout the paper of denoting them in bold upper–case letters.

$$f : A \longrightarrow B \qquad g : C \longrightarrow D$$

A functor relates one category to another: it is a mapping between categories. Figure 5 shows functor arrows K, L between categories **A** and **C** containing objects $A, B, C, \ldots$ interrelated by arrows $f, g, \ldots$. In Figure 5, K assigns from the source object A the target object K(A) to C and from a source arrow f the target arrow K(f) to g. Components of an information system may be represented by categories $\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$.

An arrow between functors is termed a natural transformation as shown in Figure 6 where there is a natural transformation α from K to L, written:

$$\alpha : K \longrightarrow L$$

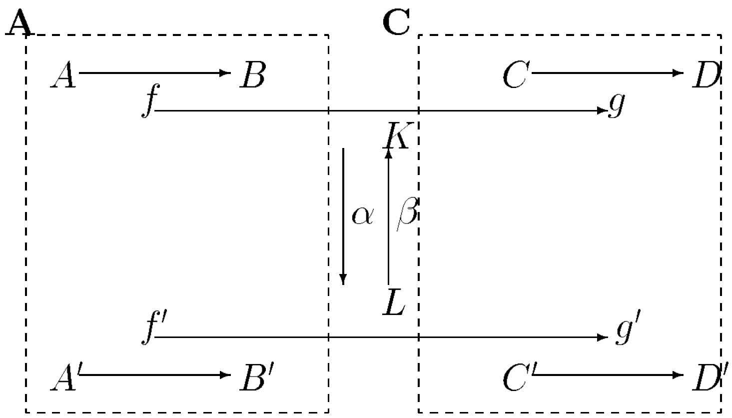

Figure 6: Natural Transformations compare Functors

This natural transformation assigns to each source object A a target arrow

$$\alpha_A : K(A) \longrightarrow L(A)$$

such that for each source arrow $A \longrightarrow B$ the target square shown in Figure 7 commutes.

Note the tight inter-relationship between the levels in category theory: morphisms and objects of categories at the lowest level are part of the expressions at the highest level of natural morphisms.

$$
\begin{tikzcd}
K(A) \arrow[r, "\alpha_A"] \arrow[d, "K(f)"'] & L(A) \arrow[d, "L(f)"] \\
K(B) \arrow[r, "\alpha_B"'] & L(B)
\end{tikzcd}
$$

Figure 7: Commuting Target Square with Covariant Natural Transformation