Modeling for Data and Business Rules

An Interview with Terry Halpin

This interview appeared in Data Base Newsletter*, vol. 25, no. 5, (Sep/Oct 1997), ed. R. G. Ross, Database Research Group Inc. and is reproduced by permission.*

Modeling Data

Newsletter: Many practitioners today still approach data modeling as higher-level database design, rather than as conceptual modeling for the business. Why do you feel that approach is inappropriate?

Halpin: I assume by 'high level database design' you mean designing at the logical level (e.g., going straight to relational table schemes, or using a corrupted version of E/R where entity types are thought of as tables). The main problem with that approach is that it makes it hard to communicate with the people who do understand the business. Data modeling should be a cooperative activity involving both the modeling expert and the business expert. To facilitate this process, the modeler needs to speak to the business expert in plain language using business terms, not arcane database terminology.

Newsletter: What are the dangers to the business in not designing a database based on a conceptual model?

Halpin: If you don't begin with a conceptual model, it's much harder to ensure that your model actually does reflect the way your business works. Also it's a lot harder to cope with changes to your business. Conceptual models are much more stable than lower level database designs.

Newsletter: What are the technical dangers in not designing a database based in a conceptual model?

Halpin: Conceptual modeling makes it easier to capture and validate the business rules at a high level, from which you consciously can make informed choices about how to implement the rules. Designing at a lower level makes it easy to misconstrue or even miss some rules, and often means you commit too early to some implementation strategy.

Without a conceptual model of your business, the task of maintaining or evolving your database application can be a nightmare. If your technical designer leaves, somebody else has to wade through low level documentation to understand the application, and if you decide to port the application to a new platform, the re-engineering task can be daunting. A conceptual model provides a readily understandable structure, to which you may apply different mapping algorithms to implement it in whatever kind of system you like.

Newsletter: Do you believe there is a clear distinction between a conceptual model and a logical model?

Halpin: Yes.

Newsletter: What criteria would you use to make that distinction?

Halpin: A conceptual model is expressed in terms of concepts and language familiar to the business person. It breaks information down into elementary facts, and makes no commitment about how these facts are grouped into structures. A logical model commits itself to grouping facts in one specific way into those specific structures provided by the chosen underlying data model (e.g., hierarchic, network, relational, object-relational, nested-relational).

For example, conceptually we may have facts of type: Employee manages Department; Employee drives Car. Suppose each employee manages at most one department, each department has at most one manager, and each employee may drive many cars (and vice versa). In a hierarchic or network model, some of these facts might be stored as an inter-record link or inside a record structure. In the relational model, the 'drives' fact would be stored in a Drives table, but the 'manages' fact could be stored in an Employee table, a Department table, or a separate Management table. In an object-relational model, we might store both facts inside an Employee table by using a set type for cars. All of these choices about how to store the conceptual facts are implementation decisions (at the logical level), and have nothing to do with how a human being conceives of the business.

Newsletter: Can a single modeling technique be used for both purposes?

Halpin: Yes. Using a method such as ORM (Object-Role Modeling) you can model the elementary facts conceptually, and then use various mapping procedures to group these into the logical structures.

Newsletter: What marks the transition from one phase to the next?

Halpin: Essentially, a logical model groups conceptual facts into structures that hold one or more fact types. These logical fact structures are often compound. For example, suppose that each employee works for some department and maybe has a mobile phone. We could group these fact types into a single relational table. Employees with mobiles would have two facts recorded on their row. Employees without a mobile would have only one fact recorded, thus giving rise to the awkward notion of a null value. Null values don't appear at the conceptual level because all facts are elementary.

Newsletter: Do criteria exist for 'proving' that a conceptual model is correct or consistent?

Halpin: Up to a point. A conceptual model is correct if it accurately models those aspects of the real world that are of interest for the application. Using ORM, a modeler can validate various features of the model with the business expert by discussing them in natural language and using sample populations to check various constraints. Although this makes it a lot easier to get a correct model, it is always possible that the business

expert might not really understand some aspect of the business, and an error could slip through. In this sense, no method can really guarantee correctness.

ORM includes a number of formal checks to help the modeler detect any mistakes in the model. Also, formal checks can be made to determine that the business rules are consistent (i.e., they don't contradict one another).

For example, suppose your model includes the exclusion constraint: no Person who wrote an Article also reviews that Article. Now suppose we add the subset constraint: each Person who wrote an Article also reviews that Article. These two constraints are 'population-inconsistent' (i.e., you can't populate the fact types without generating a logical contradiction).

ORM tools such as InfoModeler will detect such constraint violations automatically. Although most business constraints can be checked in this way, the general problem of determining consistency for all possible constraint patterns is undecidable.

Newsletter: How well do other techniques– e.g., E/R or object orientation– measure up on these same criteria?

Halpin: Not very well. They are not designed for natural communication or for population checks, and commercial tools that support them are very weak in the range of business constraints that can be expressed or validated.

Techniques for Modeling

Newsletter: Various forms of Entity-Relationship (E/R) techniques represent the most popular approach to high-level modeling for databases. Do you consider E/R to be a conceptual modeling technique?

Halpin: Done properly, E/R could be considered conceptual, although not to the same degree that ORM is. Unfortunately, E/R as supported by typical commercial tools is not much different from logical modeling. For example, IDEF1X discusses foreign keys, a relational notion that is not part of true E/R.

Newsletter: What are the strengths of the E/R approach?

Halpin: It enables you to talk about some objects and relationships, as well as a few constraints, in a reasonably high level way. E/R diagrams also are compact enough to display a lot of information in a small space.

Newsletter: What deficiencies does the E/R approach have?

Halpin: Because E/R uses attributes, its models are inherently unstable. If you later decide to record something about an attribute, it has to be remodeled as an entity type or relationship. Moreover, E/R models are too awkward for most population checks. You also need additional ways of specifying constraints (one way for relationships, and another way for attributes). As commonly implemented, E/R supports very few constraints, and has many kinds of restrictions that make it hard to describe the business

naturally. For example, relationships typically must be binary infix, and usually can't be nested in other relationships.

Newsletter: You are known as the originator of the Object-Role Modeling (ORM) approach. How does ORM differ from E/R?

Halpin: Actually ORM was developed originally in Europe by a group of people, including Falkenberg and Nijssen. What I did do was formalize the method (e.g., so logical proofs of schema consistency and equivalence could be undertaken), and extend and refine the method for practical use.

Unlike E/R, ORM was designed from the ground level up to facilitate communication between the modeler and the subject matter expert, and to facilitate schema evolution. It allows you to talk in plain sentences, using sample populations to validate a wide variety of business rules. Sensibly, it makes no use of attributes as a base construct. This not only makes conceptual models and queries more stable, but also leads to a simpler and more uniform way of expressing business rules.

Newsletter: ORM features the modeling of business facts. How is that different from relationships?

Halpin: A relationship is either a fact or a reference. A reference indicates how some objects (typically values, such as character strings or numbers) are used to identify other objects. All other relationships are called facts. For example, employees and departments might be identified by the reference types: Employee has Employeenr; Department has Deptname. Two fact types based on these object types might be: Employee works for Department; Employee heads Department.

ORM is so-called because it pictures the world in terms of objects playing roles. A role is just a part in a relationship, and a relationship may have one or more roles. A unary has one role, e.g., Employee smokes. A binary has two roles, e.g., Employee drives Car. A ternary has three roles, e.g., Athlete obtained Position in Event. And so on. If you replace each object term in a fact sentence by an ellipsis, you see the logical predicate– e.g., '... smokes,' '... drives ...,' '... obtained ... in ....'.

Notice that the ellipses or 'object holes' may appear in any position in the sentence. This is known as 'mixfix' notation, since you can mix the objects up with the predicate in whatever way it seems natural to verbalize the relationship. This is essential for languages such as Japanese, where verbs come at the end. An n-ary relationship may be given n different readings, so you can verbalize starting at any role. For example, the ternary above may be given the alternate readings: Position in Event was obtained by Athlete; Event resulted in Position for Athlete.

Newsletter: What do the distinctions between ORM and E/R translate into how the practitioner approaches the task of modeling?

Halpin: Using ORM, modelers speak to the domain experts in their own language. Rather than agonizing over what features should be modeled as attributes, they express the facts and rules in plain language, and use sample populations to help validate the rules. Unrestricted by a binary-infix straitjacket, they can verbalize the facts in a natural

way. In short, they can concentrate on modeling the business as they see it, rather than having to recast it in terms of artificial constructs.

ORM facilitates the development of accurate, detailed models that are impacted minimally by later changes to the business. Once you have an ORM model, you easily can generate an E/R model from it. A tool such as InfoModeler can automate this for you. Though less detailed, the E/R model is useful in providing a more compact overview. A practitioner who wants to keep using E/R can get the best of both worlds by using ORM to do the original conceptual model, and then abstracting this to E/R. This leads to higher quality E/R models, and also reveals the details that they lack.

Newsletter: Does the additional rigor make the task more, or less, difficult?

Halpin: A lot less difficult.

Newsletter: Doesn't the additional rigor make the resulting diagrams more difficult to read and portray?

Halpin: No. ORM diagrams are more thorough, but easier to understand. Among other cryptic features, E/R diagrams in practice often have lots of artificial entity types introduced to model n-aries in stages indirectly, or to get around restrictions such as not being able to have a relationship to which an attribute or relationship can be attached.

If your application has some information that you naturally would verbalize as a quaternary (e.g., Product in Quarter in Region was sold by Retailer) why make life difficult for yourself by binarizing it? Apart from simplicity, populating the quaternary as it is makes it easier to check out constraints (e.g., is it all-key?). Suppose the quaternary is all-key, and we now want to record details about it. In ORM we simply objectify the quaternary (as Sale, say) and attach the relevant relationships to it (e.g., Sale has Quantity).

The Challenge of Change

Newsletter: One of the points you mentioned briefly above was the issue of schema evolution. In simple technical terms, what does that mean?

Halpin: Over time the structure of your business application typically changes, so your model should evolve to reflect these changes.

Newsletter: What is a common example of this type of problem?

Halpin: The most common change is to add a new fact type that needs to be recorded. Sometimes a constraint or derivation rule changes. And sometimes we need to change an identification scheme.

Newsletter: Why is this problem so important to running the business?

Halpin: If your model doesn't reflect your business accurately, it's easy to make wrong decisions.

Newsletter: How well is the problem understood today?

Halpin: Fairly well.

Newsletter: How well do current techniques and technologies address it?

Halpin: Typically not very well.

Newsletter: What is necessary to address it at the requirements or modeling level?

Halpin: Use ORM to do your base modeling, because it was designed to minimize the impact of change. If you still want to use E/R or OO, treat these as abstractions of ORM.

Newsletter: What is necessary to address it at the database engine level?

Halpin: It's best to do your modeling at the conceptual level, then annotate your conceptual model with any desired mapping choices, and use mapping algorithms to transform to the lower levels.

Modeling Business Rules

Newsletter: How do you define 'business rule?'

Halpin: Some people include fact types or even object terms as rules, but I prefer to think of a business rule as either a constraint (e.g., each Person was born on at most one Date) or a derivation rule (e.g., Person1 is father of Person2 if Person1 is parent of Person2 and Person1 is male).

Newsletter: In what way does the business rule approach extend beyond data modeling?

Halpin: It's still essentially data modeling, but done properly. If you include dynamic business rules, it also brings in process and event modeling.

Newsletter: Does your definition place business rules at the conceptual level, the logical level, or both?

Halpin: They are best specified at the conceptual level but can be mapped to lower levels.

Newsletter: How does the business rule approach affect the way we think about processes?

Halpin: I like to think of processes in terms of operations being performed on data.

Newsletter: What advantages does that hold for the business?

Halpin: Data models are a lot more stable than processes, so that facilitates coping with change.

Newsletter: A business rules approach places emphasis on the declarative capture of rules. How well can that be integrated with a conceptual approach to modeling?

Halpin: It's a perfect fit.

Newsletter: What extensions and/or new techniques are needed to address the constraints?

Halpin: I think that ORM provides the best approach here.

Newsletter: What do business rules look like in ORM?

Halpin: You can express them either diagrammatically or in formalized natural language. For example, Figure 1 depicts three fact types: Room provides Facility; Activity requires Facility; Room at Time is used for Activity. Object types are shown as named ellipses, with their reference schemes in parenthesis. Logical predicates appear as named sequences of roles, where each role appears as a box. Roles are connected by line segments to the object types that play them.

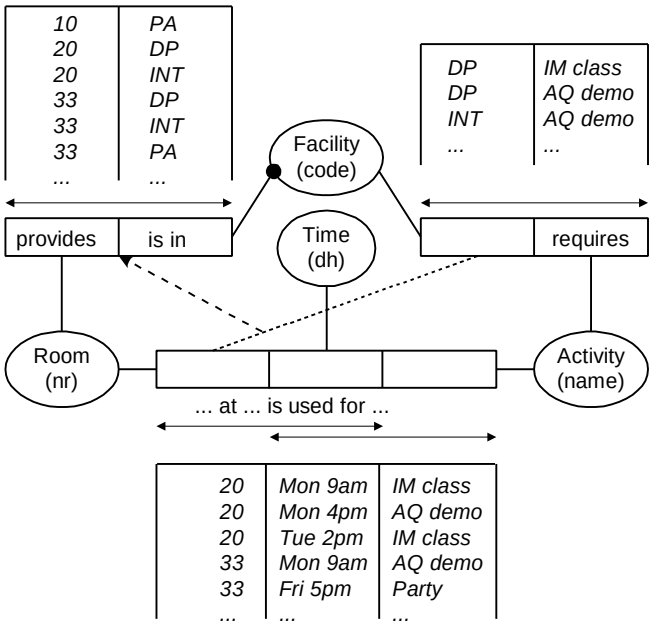

Figure 1: Example of a business rule in ORM

The black dot is a mandatory role constraint. In our example this verbalizes as: each Facility is in at least one Room. The arrow-tipped bars are uniqueness constraints (analogous to cardinality constraints in E/R). In our example each uniqueness constraint spans two roles, indicating that pairs that instantiate them must be unique. The three tables shown provide sample populations that satisfy all the constraints, and are not part of the data model structure itself.

The two binary predicates are m:n (an inverse reading is shown for one of these). The ternary predicate has two uniqueness constraints. The one over the first two roles indicates the common rule that each room at each time is used for at most one activity. The constraint over the last two roles indicates that each activity at any given time uses at most one room. For some applications this constraint might not apply; you can check it

with the client by saying the rule in English and by adding a counterexample to the population (e.g., add the tuple "40, Fri 5pm, Party" and ask if it's possible to have both the last two rows together). As an exercise, remodel the ternary in binary E/R and try to capture both the uniqueness constraints. Most versions of E/R simply can't do this.

Finally, the dotted arrow between two role pairs is a join-subset constraint. It declares that if a room at a time is used for an activity that requires a facility then that room must provide that facility (e.g., public address system (PA), data projection facility (DP), internet access (INT)). A tool such InfoModeler can verbalize all the constraints in English and allow sample populations to be entered and checked.

This small example shows just a few ORM constraints. Lots of other constraints are expressible (exclusion, frequency, ring, etc.).

Newsletter: Can any type of rule be expressed in ORM?

Halpin: No. Currently ORM is restricted to static constraints and derivation rules that are expressible in first order logic. But that covers a lot of ground.

Newsletter: Are the rules expressed graphically, or is some other means employed to express them?

Halpin: Most rules can be expressed graphically, but some can only be expressed textually. The textual language is a formalized subset of natural language. InfoModeler can verbalize the graphical constraints in textual form automatically.

Newsletter: Is this approach primarily for the IT professional, or is it suited for business analysts?

Halpin: It's suitable for both.

Newsletter: To use such extensions and/or techniques successfully, what new approaches will practitioners need to take?

Halpin: Modelers and domain experts need to view the modeling activity as a cooperative effort, exploiting natural language and examples to communicate.

Newsletter: What impact do you foresee these having on system development methodologies?

Halpin: Although the basic ideas are simple, practical experience has shown that vast improvements in modeling can be achieved.

Knowledge Support

Newsletter: In your opinion, where do knowledge acquisition techniques and expert systems fit, if at all, with business rules and a conceptual approach to modeling?

Halpin: Usually the modeler has access to one or more clients who collectively are a business expert, and it's easier to get the rules from them than by applying automated

inductive analyses of data. If you don't have access to such people, then such systems can be used to suggest rule patterns, but you still need to check these out with business users.

Newsletter: Do they address the same problem, a different problem, or an overlapping problem?

Halpin: There's a small overlap.

Newsletter: Is a conceptual approach to data appropriate where decision-oriented or inference-style requirements exist?

Halpin: Yes.

Newsletter: Why?

Halpin: The higher the level at which such requirements are specified, the easier it is to get them right.

Newsletter: Actually, isn't it true that most workflow or operational-level business applications do include such decision-oriented or inference-style requirements?

Halpin: Yes.

Newsletter: How should that impact what approach is taken?

Halpin: A data modeling method should be able to capture efficiently as many requirements as possible. Decision-oriented and inference-style requirements are no exception.

Newsletter: What problems do you see with current DSS and business intelligence tools?

Halpin: They are tied far too closely to the relational model. Instead of making information available in the user's terms, they require users to have a detailed understanding of how the database is implemented. Sometimes users are unable even to attempt to get the information they want. Worse still, they may get back incorrect results because the question they actually formulated was not the one they thought they were asking.

Automated Support

Newsletter: How can automated tools assist in the capture and support of business rules?

Halpin: Tools such as InfoModeler allow you to specify rules at a conceptual level, and to validate the rules using both positive and negative examples.

Newsletter: What opportunities exist for the automatic generation of code?

Halpin: Substantial.

Newsletter: What is the current state of the art in such tools?

Halpin: InfoModeler represents the state of the art in DDL code generation. Some other tools also are capable of generating a significant portion of the 4GL code for forms as well as 3GL procedures.

Newsletter: Are they proven for commercial grade applications?

Halpin: Some are.

Newsletter: What different categories of tools do you see?

Halpin: Different tools work to varying degrees at one or more of the conceptual, logical, internal, and external levels.

Newsletter: What trends do you anticipate?

Halpin: More and more will be done at higher levels, with automatic mapping to the lower levels.

Object Orientation

Newsletter: What strengths or innovations do object-oriented models offer?

Halpin: They incorporate some semantic modeling features (e.g., subtyping) and their support for complex objects and encapsulation facilitates re-use in programming environments.

Newsletter: Do object-oriented models offer a conceptual or a logical view of requirements for a business application problem?

Halpin: Mostly logical.

Newsletter: How good are they as a database technique?

Halpin: Poor.

Newsletter: Is that a result of the way they are applied by practitioners, or does it represent some deeper flaw of the approach?

Halpin: It's a fundamental problem with the approach.

Newsletter: How good are they for representing business rules, especially constraints?

Halpin: Pretty woeful.

Newsletter: Are there formal rules for 'proving' that an object-oriented design is correct or consistent?

Halpin: Not really.

Newsletter: Do you find that object-oriented models have advantages for managing change once a system is implemented?

Halpin: Their use of oids (hidden object identifiers) can help with cases where an object changes its external value-based identification scheme. But you typically still need value-based schemes that are visible to humans, so the problem of changing identifiers has only been ameliorated, not dissolved.

Newsletter: Does object orientation offer effective solutions or potential for the problem of schema evolution?

Halpin: It tends to exacerbate the problem rather than solve it. Object-oriented schemas tend to make heavy use of pointers to optimize performance for the current complex object structures. If these structures evolve, it's a very hard problem to realign the navigation pathways to avoid performance degradation.

Newsletter: What other difficulties do you see with object orientation as a suitable approach to operational business problems where data and knowledge is a significant factor?

Halpin: To begin with, OO relationships tend to be binary only, so lots of artificial objects tend to be introduced. More importantly, if OO models are to support non-trivial constraint specification and enforcement, they have a major problem in doing this efficiently. Part of the problem is that if facts are duplicated effectively in different objects, then constraints on them may have to be duplicated too. As a simple example, a book object may include a set of authors and a set of reviewers, so we need to constrain these sets to be disjoint. At the same time, a person object may have a set of books written and a set of books reviewed. So we need to constrain those sets to be disjoint too. The more complex the constraint, the worse the problem gets.

Constraint Language

Newsletter: One approach to defining constraints is using slightly extended forms of SQL. Do you feel that higher-level forms of expression are possible?

Halpin: Definitely.

Newsletter: Do you feel that natural language– e.g., unstructured English– can ever be suitable as a means to express rules?

Halpin: No. It's too ambiguous to be safe.

Newsletter: What about more structured forms of English?

Halpin: Yes.

Newsletter: In your view, can these forms of structured English simply be developed 'down' from unstructured, native English, by removing ambiguities and tightening the language?

Halpin: That would be a very inefficient way to do it.

Newsletter: How important is an underlying formal specification approach to developing such a language?

Halpin: It's essential.

Newsletter: Many constraints will be tedious and lengthy to express even in these higher forms of language. Moreover, there are clearly common patterns or symmetries across many complex constraints. Can these patterns or symmetries be exploited to streamline the constraint language for the more experienced user?

Halpin: Yes. These can be defined effectively as macros.

Newsletter: Do you believe that constraints can be typed to reflect these symmetries or patterns?

Halpin: Yes.

Newsletter: Wouldn't that provide even greater expressive power?

Halpin: It would certainly make the language easier to learn.

Conceptual Query

Newsletter: A constraint language must 'understand' the semantics of the underlying conceptual model for the data, and also– in one fashion or another– be able to make any arbitrary reference to it. A query language must be able to do exactly the same thing. Are constraint languages and query languages actually two sides of the same coin?

Halpin: Yes. A conceptual schema basically comprises fact types, constraints and derivation rules. A derivation rule is simply a way of defining one predicate in terms of existing predicates, so it can be thought of as a query applied to those predicates. Any constraint can be thought of as a check that a query to find a violation of the constraint will return the null set.

Newsletter: What are the advantages in approaching the problem in that fashion?

Halpin: It means you can have a uniform approach that uses the same language to specify a model and to perform queries on it.

Newsletter: What are the advantages in using a query language based on a conceptual model?

Halpin: It frees the user from needing to have a detailed understanding of how the model actually is implemented. The conceptual model is typically far easier to understand than the logical model. Moreover, conceptual queries can be much simpler and easier to formulate than the corresponding queries in a language like SQL.

If the conceptual language is based on ORM, you get added advantages such as semantic stability. ORM queries are semantically stable since they make no use of

attributes, and hence are unaffected by schema changes that recast an E/R or OO attribute as a relationship or entity type (hence requiring an E/R or OO query to be also recast).

InfoModelers are soon to release an ORM query tool that enables users to easily formulate queries directly in ConQuer (a conceptual query language based on ORM) without requiring the user to have any prior knowledge of the ORM model. The ConQuer queries are transformed automatically into SQL and executed on the target relational DBMS.

Newsletter: As such capabilities become available, how do you anticipate they will change the roles of business users?

Halpin: Non-technical users will be able to easily formulate almost any query they could imagine, and have it automatically transformed into optimized SQL.

Newsletter: How will they change the roles of IT professionals?

Halpin: IT professionals can focus their efforts on formulating clear, high level models and queries readily understandable to others. Most of the boring work to transform these high level specifications to an executable form will be automated.

Newsletter: What opportunities exist in this for Data Administrators?

Halpin: Data Administrators who embrace this truly conceptual technology will be empowered as never before. Their efforts will be directed more and more to understanding the high level semantics of their business, communicating this understanding, and exploiting its potential. And they should enjoy doing it.

About Terry Halpin

Dr. Terry Halpin, BSc, DipEd, BA, MLitStud, PhD, is a tenured senior lecturer in computer science at the University of Queensland, and is also a technical consultant for InfoModelers Inc., a leading developer of ORM-based tools for modeling and querying databases. InfoModelers recently was formed from the database division of Asymetrix Corporation, where Dr. Halpin worked in 1995-6 as Head of Database Research while on leave from academia. His research focuses on conceptual modeling and conceptual query technology for information systems.

Dr. Halpin has presented papers and tutorials at many international conferences. His doctoral thesis provided the first full formalization of Object-Role Modeling (ORM/NIAM), and his publications include over fifty technical papers, as well as three books, including the second edition of *Conceptual Schema and Relational Database Design* (Prentice Hall Australia, 1995).