MURDOCH RESEARCH REPOSITORY

http://researchrepository.murdoch.edu.au/1881/

Paper presented at the 14th Australasian Conference on Information Systems, in networked environments.

Hobbs, V.J., Pigott, D. and Toohey, D.P. (2003) *Situation abstraction modelling: a pattern language for databases.* In: 14 th Australasian Conference of Information Systems, 26 - 28 November , Perth.

Copyright: Valerie J. Hobbs, Diarmuid J. Pigott, and Daniel P. Toohey © 2003

Situated Abstraction Modelling: a pattern language for databases

Valerie J. Hobbs, Diarmuid J. Pigott and Daniel P. Toohey
School of Information Technology
Murdoch University
Perth, Western Australia
Email: {v.hobbs, d.pigott, d.toohey}@murdoch.edu.au

Abstract

Reusable, "standard" data models have long been an aim of the data modelling community, most recently with an emphasis on patterns. However, many standard pattern approaches call for a close modelling of the subject domain itself, and are not easily transferable to other domains. The approach we describe here, situated abstraction modelling (SAM) describes an approach to data modelling that defines a template for the functional "shape" of the solution, which is then instantiated for a particular set of circumstances. In effect, we have a template not for the data model, but for the situation to which it applies. We describe a set of six data model templates that can be instantiated over a wide range of application domains.

Keywords

Data modelling, entity-relationship modelling, template, pattern, situated abstraction modelling.

INTRODUCTION

When we design a database, we can call on formalisms in normalisation (to ensure that regularised structures are obtained), integrity constraints (to ensure integrity and consistency) and the relational algebra and calculus (to ensure the establishment of optimal design). These techniques have been established in theory and verified in implementation, and together make up the basis for education and practice. However, before such formalisms can be called upon, the initial task of analysis of any system for salient features must be carried out, and although we have a set of primitives and rules for combining them (entity-relationship diagrams, UML, data flow diagrams), we have nothing beyond that to guide the instantiation apart from verification against the world.

What is needed is a similar formalism that would enable us to have the same sort of rigorous approach to being able to model, based on first principles and experience. Reusable, "standard" data models have long been an aim of the data modelling community, for reasons of economy, reliability, and effective training. Rather than re-establish methods for every new database, the goal has been to capitalise on the intellectual and financial investment in previous design work, and the search has been for a framework within which this could be established.

One suggestion that addresses the problem of adapting experience both to further design and to education is to make use of pattern languages, a convention adopted from the arts of ornamentation and architecture by the "Gang of Four" (Gamma, Helm, Johnson, & Vlissides, 1995). Pattern languages are reusable elements of design derived from experience, but abstracted by an act of analysis into reusable components: pattern books formed the basis for design distribution in a pre-literate society (Alexander, 1979). If the expertise of designers could be stored in such a pattern language for databases, then the problem of design would then become the relatively straightforward task of recognising a need for a pattern and then applying the appropriate pattern to the situation in hand.

In architecture, use of patterns consists of selecting components at an appropriate level - from that of doors, windows and walls making rooms, up to the siting of particular buildings or aspects of buildings, up to creating communities. In hypothesising a pattern language for databases, however, the selection of an appropriate level for a pattern is left largely untouched by the various pattern authors.

Pattern languages of databases

Silverston (2001) addresses the problem at the table level: by producing a vast enumeration of tables, the designer (or educator) selects the table most likely to fill the need, trusting to the skill of the language preparer to have dealt successfully with all possible cases. However, the problem arises of recognising what selection amounts to a lossless decomposition: if this is the skill that is being taught to the students in the first place, selecting one of a number can't be guaranteed to teach the problem of noticing the salient, as a process of trial and error is not guaranteed to produce appropriate results. Moreover, a failure of the pattern language to meet a particular need is addressable only by referring ex cathedra to the source for either a new table, or a permissible extension to an existing one – this is not a substitute for the analysis skills needed.

Fowler (1997) borrows directly from the Gang of Four in observing components of database systems as functional subunits (Composite, Observer, Proxy, Singleton, Factory, Visitor etc) of a design whole. This concentration on the functionality is useful in the creation of a greater whole out of a set of components, but falls prey to the same problems that exist at a higher level: identification of the salient features within these pattern elements still remains to be done when the identification of the components has been made, and the bulk of Fowler's text is involved in showing how the components can be made to accommodate different data contexts.

Hay (1996) uses patterning to analyse the subcomponents at a higher level of abstraction than Silverston, and in a manner that is more data-centric than Fowler: using higraph formalisms he shows how the data components of various common database scenarios can be shown to have a commonality, and how these components can be reused. However, the problem remains of identification of data-relevant features in areas not previously enumerated, and ultimately suffers in the same way as Silverston's solution.

In the final analysis, the solutions Fowler provides are insufficient because they come from a systems analysis, rather than a database, perspective. On the other hand, Hay and Silverston begin with a problem domain such as accounting or stock control, and provide adaptable patterns within that domain, trusting in their skill to make similar solutions in other domains.

Interestingly enough, we can see the presence of similar levels in the automated design solutions offered by vendors such as Microsoft: we can see the wizards at the table, database and application level in Microsoft Access™ and Visual Basic corresponding to Silverstein, Hay and Fowler respectively. Such utilities, while useful as a shortcut for rapid application development, do not fulfil the required level of reused expertise either.

More importantly perhaps, Alexander (1979) stipulates that a pattern language should begin with the general, and only then work towards the specific. By establishing the work in the way that they have, the authors have already begun it at a level below that of general abstraction (or in the case of Fowler, in a functional abstraction that almost immediately has to be followed by a quest for instantiation). In other words, it is not possible to carry out an abstractive design of a database that makes use of experience via patterns using these methods: any attempt immediately returns to the concrete.

There is another, allied problem: not only are these patterns grounded in examples where the functional nature of the system is inextricable from its subject domain, but they are also tied far too closely to the sort of transactional systems that make up the majority of business data texts. The domain of science or humanities databases has proven to have different modelling problems (McCarty, 1998; Williams, Messina, Gagliardi, Darlington, & Aloisio, 1999), and business-grounded analyses are not going to provide reliable guides to analysis for those systems.

What we ultimately seek from pattern languages is a way in which the benefits of experience (situating; enframing; putting in context) can sit easily with the needs of abstraction in design (working with late-binding variables, models that can be placed with each other without reference to the specific). It was in search of such a modelling process that we began our analysis, described next.

What we needed

Our own approach to database pattern languages arose from a pedagogical need to be able to model non-transactional systems on any topic. Our situation was that of using entity-relationship modelling to create data models on a variety of themes in a course in multimedia databases. The issues we faced were:

- The models had to be fairly simple, since the exercise focused on the use of different media types for illustration, yet they still had to be able to support the media modelling precisely and to provide for varied and interesting views of the database.
- Students had a free choice of database theme, so selected anything that interested them (as long as it had potential for the multimedia component) – this resulted in a very wide range of themes not often met with in undergraduate database texts, such as cookery, horror movies, bonsai, wrestling, and frog ecology.
- Students seldom had much idea of how to begin modelling once they had chosen a topic – selecting (say) "cricket" is not sufficient in itself, as the focus could be on players and teams, or on details of matches in a competition.
- Most of the data modelling students had done in previous courses was of the transactional variety, and they had difficulty in applying the examples they had learned there to many of the situations they wanted to model.

We soon realized that there were certain common data models that applied no matter what the topic theme was, and that we could quickly recognize the appropriate model or suggest alternatives. We wanted to formalize this expertise so that we could make it available to the students, and so fast track the data modelling part of the exercise (since it was occupying a disproportionate amount of time). And, by providing a variety of models, we wanted the students to be able to consider the different perspectives on a topic and their implications explicitly, thus increasing their appreciation of the subtleties of their theme and its modelling potential.

We turned to the pattern literature (as discussed above) to see what comparable solutions had been met. We found that the typical pattern approaches did not meet our needs. And, as we have seen, modelling each system from first principles was not a viable option as it meant foregoing experience gained from previous similar systems.

Our method

We decided to identify genera of modelling situations and matching modelling templates. By concentrating on the logical entities, their interaction with the world, the system's portrayal of time and context, we established a set of descriptors with which to describe these circumstances. Using the situated activity framework (Clancey, 1993), we sought a level of abstraction that was firmly grounded in universal data practices, while at the same time permitting fully portable modelling tools: more abstracted than Fowler, Silverston and Hay, yet more situated than the fully abstract tools such as entity-relationship modelling (Chen, 1976) or UML (Object Management Group, 2003). We call this method Situated Abstraction Modelling (Figure 1).

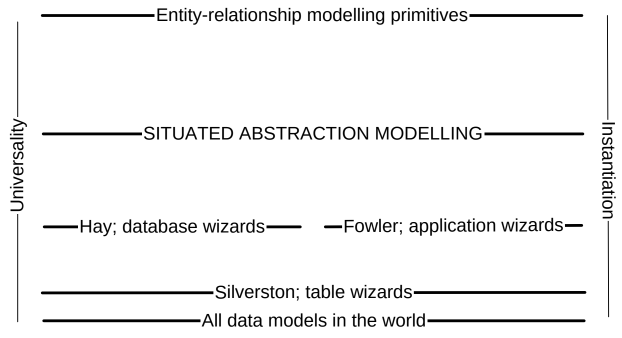
Entity-relationship modelling primitives
SITUATED ABSTRACTION MODELLING
Hay; database wizards
Fowler; application wizards
Silverston; table wizards
All data models in the world
Universality
Instantiation

Figure 1. Comparison of Situated Abstraction Modelling with other approaches to standard data models in terms of level of instantiation (situatedness) and universality (abstraction)

SITUATED ABSTRACTION MODELLING

We begin with the recognition that modelling is a situated activity because all cognitive activities are situated Clancey (1993; 1997). This means that we should not look for an absolute abstraction in our modelling, but look instead for how the modelling activity draws on norms to establish appropriate rules for analysis and optimisation. Any models formed must reflect the circumstances in which they were created, and are ultimately reflective of the world: models break down when they are taken too far from their point of formation. By the same token, the application of models in new domains consists of their contextualisation – the late-binding of value to slot that is part of the explicative and analytical process – and our modelling formalism must reflect all of these features.

Richards (2000) also draws on Clancey (1993) to make an analysis of conceptual modelling, and remarks on the close interaction of modelling and experience: "It is not just the external environment that will affect the context but that thinking itself modifies further action and context occurs at a conceptual level that exists within a social setting". While we do not support the sceptical position that Richards adopts, the inclusion of knowledge of circumstance and feedback from users is significant for our purposes: the elicitation of salient features in a database system begins with the knowledge of the system, and an awareness of rules that might apply and a knowledge of how those rules have applied in similar circumstances. We see the establishment and encoding of such rules, together with their appropriate domains of application, as the soundest approach to the creation of a pattern language for database design.

Thus, the templates that we have developed are not metamodels, but standardised rulesets of situations where data is used, and how those rules must be applied.

We identified six templates: Organisational, Events, Performance, Distribution, Sociological, and Construction, based on our analysis of common situations across a broad range of database designs. Within that template set, we defined a limited set of logical entities, which exist in a particular pattern of relationships that together characterise the particular template. The use of a template lies in establishing the genera of the particular database system, instantiating the logical data entities, and establishing the grounds for integrating with neighbouring data systems through the use of intersection entities.

The logical entities are replaced by actual entities when the template is instantiated. For example the logical entity AGENT may be instantiated as PERSON, SINGER, BRANCH, TEAM, as required for the particular domain. Most of these logical entities are specific to the template in which they are used, but some (notably AGENT) occur across multiple templates. Although they may typically be instantiated in different ways, the main features and principles of their use is the same no matter where they are used.

Within these logical entities, we then prescribe some required attributes – specifically primary keys, foreign keys and human-readable identifiers (names for the instantiated logical entities, labels for the typifying ones) and give rules for recursion in instantiation (described next).

Recursive hierarchies

Instantiation of the logical entity may well require its expansion into a hierarchy of entities (for example, the AGENT entity could represent the progressive grouping from Person to Department to Branch). This expansion is indicated on the template as a standard recursive relationship symbol (and is written in the text with an arrow symbol; e.g. Person ✐ Department ✐ Branch) but it is important to note that the recursion differs depending on the logical entity involved. We present a brief summary of the recursion types here, and will provide a fuller treatment of them elsewhere.

Several of the recursion types identified map on to types noted by other authors, for example Simsion (1994). Simsion recognized three different types of recursive relationship, 1:1, 1:N, and M:N, and within 1:N (which he called hierarchical) three different types: *contains*, *classifies* and *controls*.

- Traditional recursion. This is the “bill of materials” recursion, where we have hierarchies consisting of parts-subparts. Simsion calls this a *contains* hierarchy. In our templates, traditional recursion is found in the Construction template, around the COMPONENT and PROCESS entities.
- Hierarchical recursion. This type of recursion is a classification, in the sense of taxonomic classifications. It is characterized by inheritance from top to bottom: everything that is true at the highest level of the hierarchy is also true at all lower levels. A typical example would be Species ✐ Genus ✐ Family in a taxonomy. Simsion refers to this as *classifies*. In our templates Hierarchical recursion is found in SPECIES (in the Distribution template) and SITUATION (Distribution and Sociological). These hierarchies often tend to be permanent schemes that exist outside of the immediate domain of the data model.
- Constitutional recursion. In this type of recursion the hierarchy is rule-determined, for example in a bureaucratic management regime, such as Staff ✐ Department ✐ Branch; or performance-based hierarchies such as Episode ✐ Season ✐ Series. The rules for hierarchical membership are not automatically the same at each level in the hierarchy; moreover, the rules may be established by act of fiat, and a review of the rules possible at any moment, which would lead to an immediate reappraisal of the data. Simsion calls this a *controls* hierarchy. Constitutional recursion occurs in the AGENT, THING, PLAN and PERFORMANCE entities.
- Aggregative recursion. This type of recursion is similar to Hierarchical, but differs in that what is true of the highest level of the hierarchy is not necessarily true for every lower level. An example would be City ✐ State ✐ Country. Simsion calls this type of recursion *contains*. In our templates we find Aggregative recursion in LOCATION (Distribution and Sociological) and EVENT (Events template).

THE TEMPLATES

We now describe the six templates. For each template, we show entity-relationship diagrams (ERDs) of the template and an example instantiation, and a table summarising the main features of the template.

Organisational template

This template is used to represent individuals or hierarchical organisations of groups and individuals, and historicised facts about them.

The Organisational template (Figure 2) is probably the simplest of the templates. At its most extreme, it could be a single logical entity, Agent. Expansion of the Agent entity is in the form of recursion to a hierarchy of groups, and at the terminating end, to a weak entity (Agent-History) that records (usually summary) information about circumstances of interest to the Agent.

Typical instantiations are found in organisational hierarchies of companies, such as Staff ✐ Department ✐ Branch, or Employee ✐ Manager ✐ Franchise; but also in sporting domains such as Player ✐ Team ✐ Club ✐ League, or Driver ✐ Crew ✐ Team.

In the Organisational template time qualifies the recursive hierarchy for Agent: for example, players may belong to different clubs in different years. This is modelled by extending the hierarchy to many-to-many, with an indicative temporal attribute (and possibly a role) for the intersection entity.

Instantiations of the Organisational template may be found combined with those of others, such as Event or Sociological, where it effectively provides the recursive expansion seen in the Agent entity of those templates. However, the Organisational template exists in its own right, as it is required to model instances where there is insufficient detail about the weak entity to regularise it further (for example, the “fact sheets” assembled for pop stars and sporting figures).

Variations within the Organisational template include:

- Expansion of Agent into a hierarchy
- Inclusion or exclusion of Agent-History; or multiple Agent-Histories.

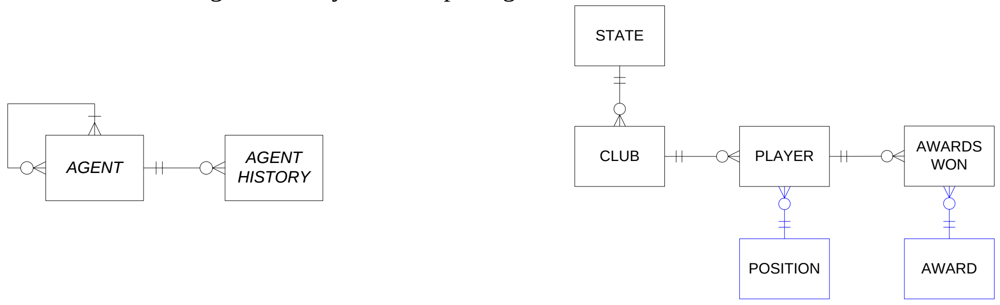

Figure 2. The Organisational template (left) and an example instantiation.

Logical entity	Subject matter represented by the logical entity	Nature of recursion	Candidate keys
AGENT	A named instance of a person or group	Constitutional	A unique identifier for the instance of the person or group
AGENT-HISTORY	A weak entity recording something of interest about the person or group, often summary in nature	No recursion	A unique identifier for the historical instance; possibly a compound key that includes the Agent candidate key

Table 1. Main features of the Organisational template.

Events template

This template is found where something of interest occurs in time: this may be either a regular occurrence, or a one-off historical moment. People or groups are always involved in the event. The basic template is thus a simple many-to-many relationship between the logical entities Agent and Event, with the entity Participation recording the circumstances of the involvement or its outcome (Figure 3).

The events of interest may be of a regular occurrence with a predefined structure (such as sporting fixtures, or the Oscars) or be part of chronicles of unique historical events such as wars or the space race. The events may also be transactions such as are encountered in financial or medical record systems, and this template is the one where typical transaction systems live.

Variations within the Events template include:

- Expansion of either of the outer entities into a hierarchy (such as Player ➪ Team ➪ Club, or Match ➪ Competition ➪ Season).
- Modelling the participation of Agents as an ordered pair (e.g. the home team and away team participating in a match)
- Enforcing rules for simultaneous participation (e.g. the number of runners in a heat)
- Modelling events that take place in a sequence (e.g. the draw for a tennis competition)
- The level of detail in recording the outcome of the event, especially for sporting events. (For example, in the instantiation in Figure 3 we could aggregate the goals scored by individual players as recorded in Plays-In, or store only the final score in Match.)

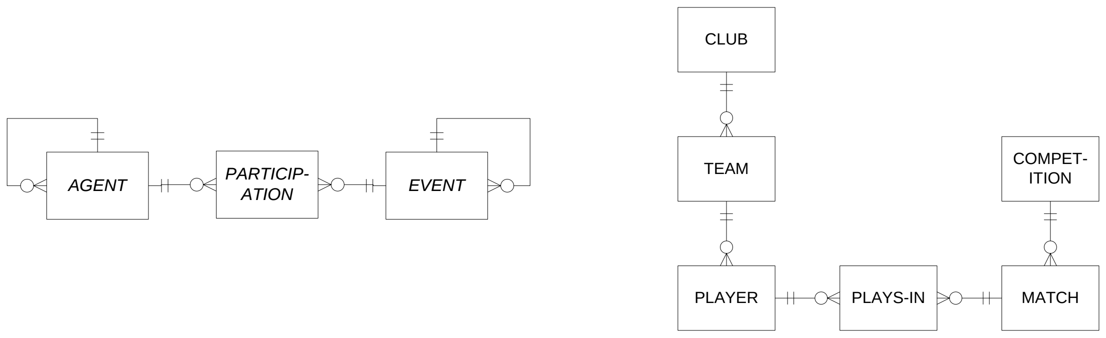

Figure 3. The Events template (left) and an example instantiation.

Logical entity	**Subject matter represented by the logical entity**	**Nature of recursion**	**Candidate keys**
AGENT	A named instance of a person or group	Constitutional	A unique identifier for the instance of the person or group
EVENT	A named and timed instance of an occurrence	Aggregative	A unique identifier for the instance
PARTICIPATION	The combination of a particular person/ group and occurrence	No recursion, but records may be able to be viewed in different grouping patterns.	The combination of agent and event primary keys is sufficient (no partial data is permitted).

Table 2. Main features of the Events template.

Performance template

This template is typically found where a performer or group of performers are following the instructions of a performance designer, which are set out as a blueprint or plan. It is characterised by two different sorts of agents, which are subtypes of the Agent logical entity: a Creator (such as a songwriter) who is responsible for producing the plan, and a Performer (such as a singer) responsible for performances of the plan (Figure 4).

The Performance template is used for databases where we are recording details about performances of music, film, plays, TV shows or other cultural artefacts. Typical instantiations would be a playlist database for a radio station, with songwriter, singer/group, song and rendition; or an online movie database with a complete cast and production details.

Variations within the Performance template include:

- Expansion of the Agent or Performance entities into a hierarchy (such as Singer ➾ Group, or Episode ➾ Series).
- The number of subtypes of Agent involved, and the way that they are subsequently converted to relational tables (see for example Elmasri and Navathe; 1999).
- Multiple variants of same performance – with the release of compilation albums, and the reissue of albums with extra tracks, physical artefacts (albums) may have components (tracks) shared with other albums (where the tracks may even have different names). Although this may seem counter-intuitive, it is straightforward to model with this template: the performance participates in a many-to-many hierarchical relationship.

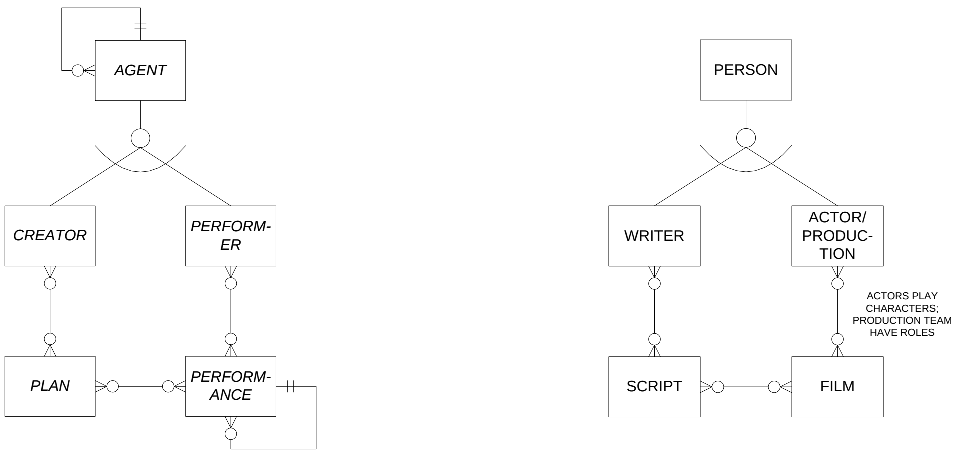

Figure 4. The Performance template (left) and an example instantiation.

Logical entity	Subject matter represented by the logical entity	Nature of recursion	Candidate keys
AGENT	A named instance of a person or group, with subtypes CREATOR and PERFORMER, each involved in separate relationships	Constitutional	A unique identifier for the instance of the person or group
PLAN	A named and timed instance of a plan	Constitutional	A unique identifier for the instance of the plan
PERFORMANCE	A named and timed instance of a performance	Constitutional	A unique identifier for the instance of the performance

Table 3. Main features of the Performance template.

Distribution template

This template is typically found where we wish to represent co-location that is independent of time or agency. It is characterised by three logical entities, Species, Situation and Location, which meet in a triple intersection as a fourth entity, Distribution (Figure 5).

The Distribution template is used for databases where we are recording instances of types of items and the manner in which they occur, but where the temporality of the instances is abstracted to time of day or year, rather than a chronological timeline.

Typical instantiations of the template are where a classified life form (such as a species of plant or animal) is to be found in terms of habitat and location, and where the temporality of the distribution represents migration or diurnal behaviour. Another example would be meteorology, where we observe the distribution of different weather phenomena (such as cyclones and ocean currents) across geographic regions and local topographical conditions.

Variations within this template include:

- Expansion of any of the outer entities into a hierarchy (such as Bird Species ➩ Family ➩ Order).
- Replacement of any of the outer entities by a lookup table to the central entity, thus enforcing a domain constraint where only the label or name of the record is of interest.
- Replacement of any of the outer entities to a business rule (effectively collapsing them to a single value); for example restricting to a particular geographical region.

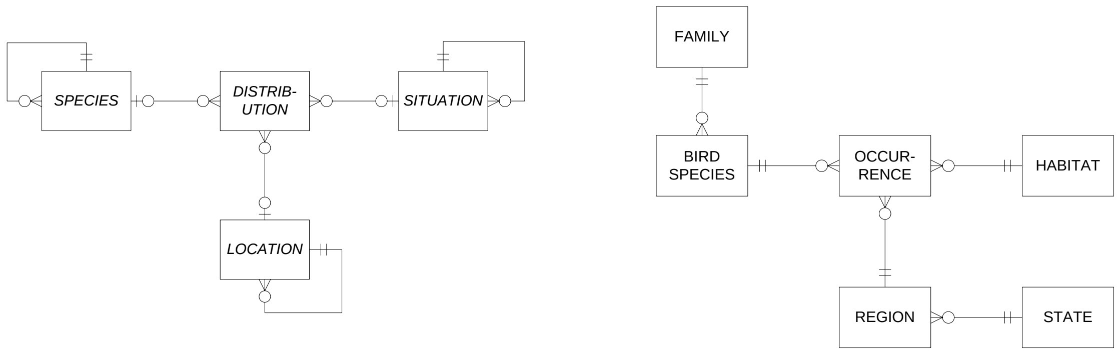

Figure 5. The Distribution template (left) and an example instantiation.

Logical entity	Subject matter represented by the logical entity	Nature of recursion	Candidate keys
SPECIES	A named type of item	Hierarchical	A unique identifier for the type of item
LOCATION	A named instance of a place, usually spatial	Aggregative	A unique identifier for the instance of place
SITUATION	A named type of context of the item occurrence	Hierarchical	A unique identifier for the type of situation
DISTRIBUTION	The combination of a particular item, place and context, resulting in a distribution pattern.	No recursion, but records may be able to be viewed in different grouping patterns.	A unique identifier for the distribution occurrence. A candidate key made up of the foreign keys of species, location and situation is inadequate because of the potential for partial data.

Table 4. Features of the Distribution template.

Sociological template

This template is similar to the Distribution template in that it represents co-occurrence, but it differs in that the notions of agency and time are now significant in determining the co-occurrence. Within the context determined by Situation and Location we have two entities, Agent and Thing, which are linked by an Actor-ActedUpon relationship (such as design or celebration). The result of this relationship is what is recorded in the central entity Occurrence, along with its date or sequence (Figure 6). Both Agent and Thing are individuated, so that their records are of named instances rather than types.

The Sociological template is typically used for databases where we are recording designed items or events of cultural significance: the important feature is that there is (explicitly or implicitly) a designer and something of significance to the model that is the result of that design.

Typical instantiations of the template are where we have a product item such as a car model that is designed by an architect. Another example would be a database of world festivals, where the celebration of a particular festival (such as Christmas) would differ according to cultural group, their geographic location and the context of the festival.

Variations within the Sociological template include:

- Expansion of any of the outer entities into a hierarchy (such as Car Model ➪ Marque ➪ Manufacturer).
- Replacement of any of the outer entities by a lookup table to the central entity, thus enforcing a domain constraint where only the label or name of the record is of interest.
- Replacement of any of the outer entities to a business rule (effectively collapsing them to a single value); for example restricting to a particular geographical region or a particular designer.

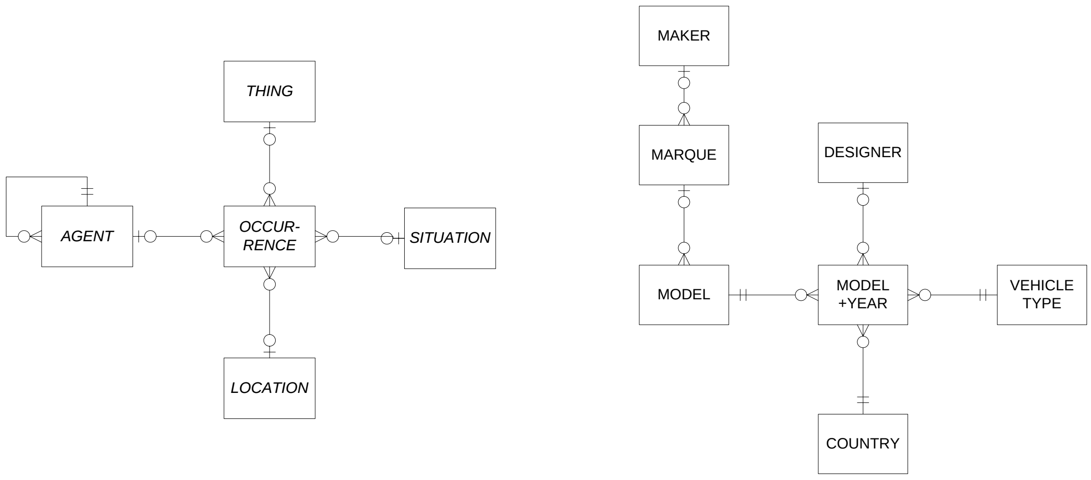

Figure 6. The Sociological template (left) and an example instantiation.

Logical entity	**Subject matter represented by the logical entity**	**Nature of recursion**	**Candidate keys**
AGENT	A named instance of a person or group	Constitutional	A unique identifier for the instance of the person or group
LOCATION	A named instance of a place, usually spatial	Aggregative	A unique identifier for the instance of place
SITUATION	A named type of context of the item occurrence	Hierarchical	A unique identifier for the type of situation
THING	A named instance of an item that is of interest to the person or group	Constitutional	A unique identifier for the instance of the item
OCCURRENCE	The combination of a particular person/group, item, place and context. Date and/or sequence are included.	No recursion	A unique identifier for the occurrence. A candidate key made up of the foreign keys of Agent, Thing, Location and Situation is inadequate because of the potential for partial data.

Table 5. Main features of the Sociological template.

Construction template

With this template, we model processes and the components involved with them; we make an internal ordering for the Process logical entity whereby we can stipulate listing order. Even though there are only two logical entities in this model, they interact in such a complex and subtle way that we have elected to show the expanded version as well as the simple one (Figure 7). The ordered components are part of each step, while the ordered steps take components and make them into compound structures which in turn become components. The recursive relationship each has is also complicated: a process may consist of many steps, with the output of one step being input to another, while a component is likely to be made of other components, and be combined into yet others.

Typical instantiations of the Construction template include any databases with the classic component assembly nature, such as model boat building; dance (where steps combine into figures into dances), and recipes, where raw ingredients are combined into parts of a dish and ultimately a complete meal.

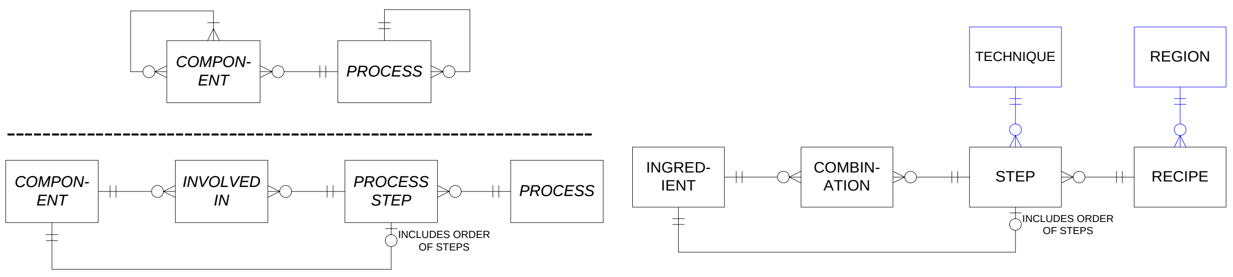

Figure 7. The Construction template (left) and an example instantiation.

Logical entity	Subject matter represented by the logical entity	Nature of recursion	Candidate keys
COMPONENT	A named type of component that can be simple or compound (made up of other components)	Traditional recursion	A unique identifier for the type of component
PROCESS	A named type of process	Traditional recursion	A unique identifier for the type of process
PROCESS-STEP	A named type of process step, plus the sequence order	(not applicable)	A unique identifier for the process step
INVOLVED-IN	The combination of a particular component with a process step	(not applicable)	The combination of component and process-step primary keys is sufficient (no partial data is permitted).

Table 6. Main features of the Construction template.

DISCUSSION

The formalisms of data modelling we can call upon provide both descriptive and normative roles in analysis. We call upon the normative aspects to give guidance, and the descriptive to assist in communication of the solution. By grounding the normative aspects in situated abstraction, we can provide workable precepts for both education and practice. By providing a minimal set of templates with rules for expansion and for combining, we have ensured that the functional requirements of the system are not lost by too early adoption of a model that is domain-oriented. In the words of Alexander (Alexander et al., 1977):

> "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

Instantiation thus becomes a matter of naming the logical entities, giving appropriate names to the required attributes, and then selecting the remaining attributes. Following Alexander et al's (1977) patterns methodology, we move from the highest level (templates) through the middle (logical entities) to the specific (required attributes and relations). One advantage of our approach is that the instantiated templates can then draw on the work of Fowler, Hay and Silverston at the appropriate point of the modelling exercise.

Teaching students and practitioners what template to use in their modelling work becomes a matter of showing how to identify common situations, how to implement them from the template, and how to combine individual templates to model complex data situations. Recognising the circumstances in which to use a particular template then becomes a matter of familiarisation with the appropriate recognitors, for example:

- Is this a situation where basic information is being collected about individuals? Then you should use the Organisational template.
- Is this a situation where goods are being bought and sold, and transactions recorded? Then you should use the Events template.
- Is this a situation where details are being recorded for a process? Then you should use the Construction template.

… and so on.

Our experience so far in using these templates to teach data modelling in courses where the emphasis is primarily on non-transactional data systems has been encouraging, and we intend to develop the approach further by developing such a set of guidelines for instantiation and combination.

REFERENCES

Alexander, C. (1979). *The timeless way of building*. New York: Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. A. (1977). *A Pattern Language*. New York: Oxford University Press.

Chen, P. P.-S. (1976). The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, *1*(1), 9-36.

Clancey, W. J. (1993). Situated action: A neuropsychological interpretation (Response to Vera and Simon). *Cognitive Science*, *17*(1), 87-107.

Clancey, W. J. (1997). *Situated Cognition: On Human Knowledge and Computer Representations (Learning in Doing)*: Cambridge University Press.

Elmasri, R. A., & Navathe, S. B. (1999). *Fundamentals of Database Systems* (3rd ed.): Addison-Wesley Publishing.

Fowler, M. (1997). *Analysis patterns: reusable object models*: Addison-Wesley Longman.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading, MA: Addison-Wesley.

Hay, D. C. (1996). *Data model patterns: conventions of thought*. New York: Dorset House Publishing.

McCarty, W. (1998). *Poem and algorithm: humanities computing in the life and place of the mind (Keynote speech).* Paper presented at the humanITies - Information technology in the arts and humanities: Present applications and future perspectives, The Open University, Milton Keynes.

Object Management Group, I. (2003, March 17, 2003). *UML Resource Page*. Retrieved June 3, 2003, from the World Wide Web: http://www.omg.org/uml/

Richards, D. (2000). *A Situated Cognition Approach to Conceptual Modelling.* Paper presented at the 33rd Hawaii International Conference on System Sciences.

Silverston, L. (2001). *The Data Model Resource Book, Revised Edition. Volume 1: A Library of Universal Data Models For All Enterprises; Volume 2: A Library of Universal Data Models by Industry Types*: Wiley Computer Publishing.

Simsion, G. C. (1994). *Data modelling essentials: analysis, design and innovation*: International Thomson Computer Press.

Williams, R., Messina, P., Gagliardi, F., Darlington, J., & Aloisio, G. (1999). *Report on European-United States joint workshop on Large Scientific Databases*. Annapolis, Maryland, USA: Center for Advanced Computing Research at the California Institute of Technology, European Laboratory for Particle Physics (CERN).

COPYRIGHT

Valerie J. Hobbs, Diarmuid J. Pigott, and Daniel P. Toohey © 2003. The authors assign to ACIS and educational and non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The authors also grant a non-exclusive licence to ACIS to publish this document in full in the Conference Papers and Proceedings. Those documents may be published on the World Wide Web, CD-ROM, in printed form, and on mirror sites on the World Wide Web. Any other usage is prohibited without the express permission of the authors.