A formal definition of the data dictionary for an extended entity-relationship data model is described. Extensions should allow greater semantic expressiveness and more precise modelling, while retaining ease of use, intuitiveness and flexibility. Basic concepts of the model are formally defined using the Z notation, and an enhanced graphical notation is also proposed. Furthermore, a simple transformation of the extended ER schema to a relational one is formally described as well.

Received March 1992, revised May 1992

1. INTRODUCTION

Formal methods are widely used in the specification phase of the software design cycle. The use of mathematically based techniques for description of system structure and behaviour was necessitated by the sheer size and complexity of modern software systems, which could not be handled by classical structured programming and design techniques. The ability to specify system properties in a precise and rigorous way, and to effectively prove that it satisfies its requirements, has been invaluable in the design and implementation of many software and hardware systems, regardless of their size.2 A number of formal specification techniques have been proposed thus far, most of them based on some well-developed mathematical notation.5 Among these, the most popular are Z,3 and VDM,4 which has even been recommended to become the official standard for software system specification.1

Data modelling at the conceptual level has always been a major part of database design, and of the design of information systems in general. A data model is a set of conceptual tools for describing relevant properties of the system under consideration, and it consists of three distinct, yet closely interrelated parts: a notation for data structure description, a set of constraints that data values must satisfy in order to be considered valid, and a set of operations for data update and retrieval.5,12 Many different data models have been proposed thus far, but only a few of them enjoy widespread acceptance. The relational model is undoubtedly one of the most popular, thanks to its simple structure and firm mathematical foundation, and many relational database management systems are commercially available. However, techniques for relational database design (including normalisation and normal forms) are not very well integrated with other system analysis and design methodologies, such as many variants of structured analysis, and relational simplicity is not always appropriate for accurate modelling of real-world concepts. Hence, other modelling techniques are used for modelling at the conceptual level, and later transformed to the relational model for implementation.

Most often, these techniques are based on Chen's entity-relationship (ER) model, or some variant thereof.3 Indeed, many of the so-called semantic data models7 and, more recently, object models,10 are rather close in spirit to the ER model. However, despite its apparent versatility and intuitiveness, some authors still consider the ER model to be just a diagramming technique, and not a data model at all, their main objection being that too much emphasis is put on the structure component, while lacking elaborate integrity constraints and operation definition. (This is the case with most other models, however.) Attempts have been made to enhance the basic ER model with dynamic concepts of operations, transactions and integrity constraints; some initial results in that direction have been reported elsewhere, for example in Ref. 9.

Another area of research would be to establish a formal basis for the ER model concepts, using modern formal specification techniques. The use of such techniques could provide a formal basis for a description of a complete data model at the conceptual level: not only the data structures, but a wide class of integrity constraints and update and retrieval operations as well. Furthermore, by integrating the model with dynamic integrity concepts, we could build a model at the external (user) level also, and allow specification of structure and behaviour of complex user-level objects. Part of this research deals with the formal specification of the extended entity-relationship (XER) data model. Z notation is used throughout, mainly because of its elegance and simplicity, and because its mathematical foundation – typed set theory – is well suited for specification of basic concepts of the XER data model. As the time of writing, the formal description of structural integrity constraints and update operations is almost complete, and we are finishing the definition of complex objects and applications. However, in this paper we present only the data dictionary component, due mainly to space limitations; other results will be published in forthcoming papers.

This paper is organised as follows: in Section 2 we present some preliminary definitions, and Sections 3 and 4 explain some basic concepts (objects and their relationships) of the XER model. In each case, a brief informal explanation precedes formal definitions, and illustrative examples serve to clarify the ideas presented. (Remember that mathematical theories must make use of
axioms which are just stated, and cannot be proved by means of that theory.) A detailed formal definition of various types of attribute is given in Section 5. Details of transformation from XER to relational model are given in Section 6, and the final section contains some conclusions and directions for further research. A short appendix contains a brief summary of the Z symbols used in the paper.

In addition, since data structure modelling with ER diagrams was one of the main reasons for the popularity of the ER model, we propose a graphical notation for the XER model. Brief descriptions of the symbols used may be found in the text.

2. BASICS OF THE MODEL

As noted in the introduction, the XER model is based on the well-known entity-relationship data model, with certain additional concepts intended to provide enhanced semantic expressiveness. All elements of the model are subsets of the basic set of

\[ \text{CONCEPTS} \]

which need not be defined any further, as is customary in Z specifications. We will also make use of a set of variable-length strings, referred to as

\[ \text{TExT} \]

The elements of the TEXT set will often be used for error reporting and similar purposes. These basic definitions will be referred to as Model schema, although they do not belong to any schema in the usual Z sense.

The CONCEPTS set includes persons, organisations, things, their properties, roles, relations, dependencies, links, events; in short, everything perceivable and distinguishable in the system we are modelling. (This system is sometimes called the universe of discourse.) Terms like ‘entity’ or ‘object’ are sometimes used, but we will reserve these for specific concepts, as will be seen shortly. The most significant types of concepts are the following:

- **Entities** are concepts representing perceivable things like persons, organisations, events and the like. In a ‘university’ database, distinct entities might be students, faculties, departments, courses, rooms, laboratories, hours, and other things relevant for our database. Similar entities may be grouped into entity sets or types (e.g. Student, Course), in which case all such entities are considered to be instances of a particular type. Throughout the paper we shall refer to sets of entities as entity types, and to singular entities as instances of their respective entity types. Entity sets will be depicted with rectangles, with inscribed name of the entity type.
- **Attributes** are concepts which denote properties of one or more entity types (e.g. Name of a Student, Rank of a Faculty member, Course.Id of a Course). Each entity instance has a set of values corresponding to these properties; the values either denote the way in which a particular property is maintained (i.e. qualitative measure), or the degree to which the property is fulfilled (i.e. quantitative measure). The set of values allowed for any particular attribute is referred to as its domain. Some graphical notations denote attributes with ovals, but at the expense of clarity; we will not show attributes on diagrams.
- **Roles** are concepts referring to ‘ends’ in a role or association. Each role must ‘end’ in exactly one association, and the respective entity type(s) may participate in other associations.
- **Associations** between entity types, and to singular entities as instances of their respective entity types. Association sets (or types) may be related by (participate in, form) an association, each of them having a distinct role. Associations formed by instances of entities of identical roles may be grouped in association sets (or types). Some associations have proprietary attributes (e.g. entities of the Student type and entities of the Course type might form the Exam association type, with Date and Grade attributes), while others have no attributes. Associations are depicted with ovals, sometimes inscribed within a rectangle (for reasons to be explained later). The name of the association type is written either within the oval, or in its immediate vicinity.

Entities and associations will be referred to as objects, since they are perceivable objects in the schema. The distinction between entities and associations is not quite sharp, since objects may exist that are both at the same time: associations formed by some entities may form (i.e. participate in) other associations, and thus, strictly speaking, behave as if they were entities themselves. An example of this type of behaviour will be shown later.

The following properties, then, hold:

- **Entities \( \subset \) CONCEPTS**
- **Associations \( \subset \) CONCEPTS**
- **Attributes \( \subset \) CONCEPTS**
- **Objects = Entities \Upsilon Associations**

where \( \subset \) stands for 'proper subset of', and \( \Upsilon \) is the familiar set union operator.

3. OBJECTS

3.1 Entities and associations

Formal definition of objects (entities and associations) is based on the concept of roles: each role ‘starts’ from an entity and ‘ends’ in an association. In order to represent the relation between roles and entities, and roles and associations, we introduce two binary relations: MappedTo and FormedBy. The former shows the ‘source’ entity from which a given role ‘starts’, while the latter shows the ‘destination’ (association) where the given role ‘ends’. Note that each association must represent the ‘destination’ for at least two roles (and maybe more than two), while each role must ‘end’ in exactly one association – roles with dangling ends must not exist. Therefore, binary, ternary, ..., associations are allowed (although higher-order associations are quite unlikely). These definitions may be formally defined with the RoleModel schema:*
A DATA DICTIONARY FOR AN EXTENDED ER DATA MODEL

An example is shown in Fig. 1, where Student and Course are entities linked by an association. As noted before, entity sets are depicted with rectangles, while associations are denoted with ovals. Association (and sometimes even role) names may be omitted from the diagrams, in cases where no ambiguity can result. Roles are graphically represented with directed lines, and role names are chosen to reflect their nature, e.g. a Student Attends a Course, and a Course is Attended_By a Student, respectively; this association might be named Stud_Course. Therefore, the following expressions hold:

\begin{align*}
\text{Student} &\in \text{Entities} \\
\text{Course} &\in \text{Entities} \\
\text{Stud}_\text{Course} &\in \text{Associations} \\
\text{Attends} &\in \text{MappedTo} \\
\text{Attended}_\text{By} &\in \text{MappedTo} \\
\text{Attends} &\in \text{FormedBy} \\
\text{Attended}_\text{By} &\in \text{FormedBy}
\end{align*}

which may be also written as

\begin{align*}
\text{Student} = \text{MappedTo}(\text{Attends}) \\
\text{Course} = \text{MappedTo}(\text{Attended}_\text{By}) \\
\text{Stud}_\text{Course} = \text{FormedBy}(\text{Attends}) \\
\text{Stud}_\text{Course} = \text{FormedBy}(\text{Attended}_\text{By})
\end{align*}

Note that an object may participate more than once in a single association, but in different roles, as shown in Fig. 2. In this case role names are mandatory, in order to be able to distinguish between roles. It may happen that a Course Requires another Course as a prerequisite, in which case the latter is Required_By the former. This association might be named Pre_Requisite.

For the present, we cannot claim that all objects (or even entities) belong to the range of the \text{MappedTo} relation; a more detailed definition will be given later.

3.2 Cycles in object definitions

As noted before, associations may form other associations (Fig. 3), in which case the symbol for association is inscribed within a rectangle, to stress the fact that this particular association exhibits entity behaviour. For example, although Stud_Course is an association formed by Student and Course, it behaves like an entity when participating in another association, which might be termed Hour. This makes more precise modelling of some real-world concepts possible; unfortunately, the same holds for circular definitions (e.g. an association may form itself). The circular definition might be either direct or indirect, via zero or more intermediate associations. It is beyond the scope of this paper to discuss methods to prevent circular definitions from appearing; it will suffice to show how such definitions can be detected directly from the model. In order to accomplish this, we will define a binary relation \text{Forming}, thus establishing a link between entities and associations in which they participate.

\begin{align*}
\text{ModelF} &\rightarrow \text{CONCEPTS} \\
\text{Forming} : \text{CONCEPTS} \leftrightarrow \text{CONCEPTS}
\end{align*}

which may be written in a compact, but somewhat opaque form as

\begin{align*}
\text{Forming} = \text{FormedBy} \circ \text{MappedTo}
\end{align*}

Special symbols \(\sim\) and \(\circ\) denote the inverse of a relation and relational composition, respectively; the meaning of these may be deduced easily by comparing the two definitions of the \text{Forming} relation.

The following schema expresses the fact that circular definitions are not allowed:

\begin{align*}
\forall x : \text{Associations} \bullet (x \mapsto x) \notin \text{Forming}^+
\end{align*}

where \(R^+\) denotes the transitive closure of the relation \(R\). An additional success-reporting schema may also be defined:

\begin{align*}
\text{RoleCycleSuccess} &\rightarrow \text{TEXT} \\
\text{report!} = "\text{No cycles in role definition}" \\
\end{align*}
Names ending in an exclamation mark denote output variables, as is customary in Z specifications.6,11

If no circular definitions are detected, we should report success:

\[ \text{NoRoleCycles} \land \text{RoleCycleSuccess} \]

However, if such definition is found, we must signal its presence to the user:

\[ \text{RoleCycleFound} \]

Previous schemas may be combined to give the following schema:

\[ \text{RoleCycleCheck} = (\text{NoRoleCycles} \land \text{RoleCycleSuccess}) \lor \text{RoleCycleFound} \]

which may be written in its entirety as:

\[ \begin{align*}
\text{RoleCycleCheck} & \quad \text{ModelF} \\
& \quad \text{report}! : \text{TEXT} \\
& \quad (\exists x : \text{Associations} \bullet (x \rightarrow x) \in \text{Forming}^* \land \\
& \quad \text{report}! = "\text{Cycle found in role definition!}")
\end{align*} \]

Previous schemas may be combined to give the following schema:

\[ \text{RoleCycleCheck} = (\text{NoRoleCycles} \land \text{RoleCycleSuccess}) \lor \text{RoleCycleFound} \]

which may be written in its entirety as:

\[ \begin{align*}
\text{RoleCycleCheck} & \quad \text{ModelF} \\
& \quad \text{report}! : \text{TEXT} \\
& \quad ((\forall x : \text{Associations} \bullet (x \rightarrow x) \in \text{Forming}^* \land \\
& \quad \text{report}! = "\text{No cycles in role definition!}") \\
& \quad \lor \\
& \quad ((\exists x : \text{Associations} \bullet (x \rightarrow x) \in \text{Forming}^* \land \\
& \quad \text{report}! = "\text{Cycle found in role definition!}"))
\end{align*} \]

### 3.3 Cardinality

The role concept (Subsection 3.1) has an additional property, its cardinality, which may be defined as an ordered pair \((lb, ub)\) with the following components:

- Each instance of the entity \(\text{MappedTo}\) the role must participate in at least \(lb\) instances of the association \(\text{FormedBy}\) that particular role. \(lb\) is called the lower cardinality bound.
- Each instance of the entity \(\text{MappedTo}\) the role may participate in at most \(ub\) (the upper cardinality bound) instances of the association \(\text{FormedBy}\) that role.

The lower bound may be 0, 1, ..., or \(k\), where \(k\) is a known integer, while the upper bound may be 1, 2, ..., \(k\) (a known integer), or an unknown integer \(M\). Obviously, the lower bound must not exceed the upper one; at most they may have equal values. Most frequently, cardinality pairs are \((0, 1), (0, M), (1, 1),\) or \((1, M)\), although other combinations may appear as well. The definition of cardinality is expressed with the following schema:

\[ \begin{align*}
\text{RoleCardinality} & \quad \text{RoleModel} \\
& \quad \text{Bounds} : \mathbb{N} \times (\mathbb{N} \cup \{M\}) \\
& \quad \text{Cardinality: CONCEPTS ++ Bounds} \\
& \quad \forall x : \text{Roles} \bullet (\exists y : \text{Cardinality} \bullet \text{first } y = x) \\
& \quad \forall(L, g) : \text{Cardinality} \bullet \\
& \quad \left( L \in \text{Roles} \Rightarrow (\text{second } g = M \lor \text{second } g \geq \text{first } g) \right)
\end{align*} \]

\(\mathbb{N}\) and \(\mathbb{N}_0\) denote the set of natural numbers with and without zero, respectively; \(\times\) is the familiar Cartesian product operator.

The true meaning of cardinality constraints cannot be defined at the object (model) level; it has to be expressed at the data level, where object instances are considered. For instance,

- \(\text{Cardinality}(\text{Requires}) = (0, 3)\) means that no Course may require more than three other Courses as \text{Pre_Requisites}; since the lower bound is zero, some Courses may exist that do not require any other Course as a \text{Pre_Requisite}.
- \(\text{Cardinality}(\text{Required_By}) = (0, M)\) means that each Course may be required by many other Courses as a \text{Pre_Requisite}; since the lower bound is zero, some Courses may exist that are not required by any other Course as \text{Pre_Requisites}.
- \(\text{Cardinality}(\text{Held_In}) = (1, 1)\) means that each \text{Stud_Course} must be Held_In at least one Room, and no more than one Room; in other words, it must be Held_In exactly one Room.
- \(\text{Cardinality}(\text{Attends}) = (4, 6)\) means that each \text{Student} must Attend at least four Courses, but no more than six of them.

![Figure 3. An association may form other associations.](image-url)
4. SPECIAL RELATIONSHIPS

Besides associations, objects may be related in other ways, which will be discussed in more detail in this section.

4.1 Generalisation and specialisation

One of the possible relationships between objects is the generalisation/specialisation, which relates one object type termed a supertype, and one or more object types, denoted as subtypes. Each instance of a subtype object has a corresponding instance of its super-type object, e.g. each instance of Professor is also a Faculty member. The classification criterion is a separate concept in our model, since any object may have zero, one or more groups of distinct subtypes, each group corresponding to a different classification criterion. The supertype object type is considered to be a generalisation of its subtypes, while subtype object types might be viewed as specialisations of their supertype. However, for clarity of presentation we shall introduce the classification criterion as a separate concept, and define generalisation and specialisation as binary relations between objects and appropriate criteria. These properties may be expressed with the following schema:

\[
\begin{align*}
\text{GenSpecModel} & \rightarrow \text{Model} \\
\text{Generalisation} : \text{CONCEPTS} & \rightarrow \text{CONCEPTS} \\
\text{Specialisation} : \text{CONCEPTS} & \rightarrow \text{CONCEPTS} \\
\text{Criteria} = \{x: \text{CONCEPTS} | (\exists y: \text{Objects} \cdot (x \rightarrow y) \in \text{Generalisation})\} \\
\text{SubTypes} = \{x: \text{Entities} | (\exists y: \text{Criteria} \cdot (y \rightarrow x) \in \text{Specialisation})\} \\
\forall x: \text{Criteria} \cdot (\exists y: \text{Objects} \cdot (x \rightarrow y) \in \text{Generalisation}) \\
\forall x: \text{Criteria} \cdot (\exists y: \text{SubTypes} \cdot (x \rightarrow y) \in \text{Specialisation})
\end{align*}
\]

An example of the generalisation/specialisation relationship is shown in Fig. 4, where the criterion symbol is a half-circle, and subtypes are depicted with rectangles with an oblique line in the upper left corner. Generalisation relationship is depicted with an undirected line from the supertype entity to the criterion, while specialisation relationships are denoted with directed lines from the criterion to its subtype entities. The example shown in Fig. 4 may be formally expressed with the following:

\[
\begin{align*}
\text{Position} & \in \text{Criteria} \\
\text{Faculty} & \in \text{Objects} \\
\text{Assistants} & \in \text{Objects} \\
\text{Professors} & \in \text{Objects} \\
(\text{Position} \rightarrow \text{Faculty}) & \in \text{Generalisation} \\
(\text{Position} \rightarrow \text{Assistants}) & \in \text{Specialisation} \\
(\text{Position} \rightarrow \text{Professors}) & \in \text{Specialisation}
\end{align*}
\]

which may also be written as

\[
\begin{align*}
\text{Faculty} = \text{Generalisation} (\text{Position}) \\
\text{Assistants} = \text{Specialisation} (\text{Position}) \\
\text{Professors} = \text{Specialisation} (\text{Position})
\end{align*}
\]

Note that each Criterion connects a supertype object type with one or more SubType object types, which in turn means that Generalisation is a total function from Criteria to Entities. On the other hand, the inverse of Specialisation is a total surjection (i.e. a total mapping from SubTypes onto Criteria):

\[
\begin{align*}
\text{Generalisation} & = \text{Criteria} \rightarrow \text{Entities} \\
\text{Specialisation} & = \text{SubTypes} \rightarrow \text{Criteria}
\end{align*}
\]

Circular definitions are prohibited here as well; prior to formulating the appropriate schema, we will define the SuperType function as:

\[
\begin{align*}
\text{SuperType} & : \text{CONCEPTS} \rightarrow \text{CONCEPTS} \\
\text{SuperType} = \text{Specialisation} \cup \text{Generalisation} \\
\text{dom} \text{SuperType} & = \text{SubTypes}
\end{align*}
\]

A schema similar to RoleCycleCheck will check for possible circular definitions:

\[
\begin{align*}
\text{GenSpecCycleCheck} & \rightarrow \text{ModelS} \rightarrow \text{TEXT} \\
\text{report!} & = \text{"No cycles in gen/spec definition"}
\end{align*}
\]

As is the case with Roles, each Criterion should have an associated cardinality pair, which may be interpreted as information about the exclusivity and optionality of the relationship.

If the lower bound \( lb \) is zero, the specialisation is optional, and instances of the supertype object may or may not have related instance(s) of any of the subtype objects. If \( lb = 1 \), the specialisation is mandatory, and each instance of the supertype object must have at least one related subtype instance. Values of \( lb \) greater than 1 do not make much sense anyway; if they appear to be needed, it might mean that our model is not correct, and additional concepts are to be introduced.
As for the upper bound $ub$, the value of 1 means that each instance of the supertype object is related to at most one instance of any of its subtypes. In that case, we say that the specialisation is of exclusive variety, and instances of different subtypes are related to disjoint subsets of supertype instances. If $ub$ is greater than 1, any instance of the supertype object may have one or more related subtype instances. Note that these instances must belong to distinct subtype objects, and $ub$ cannot exceed the total number of distinct subtypes of that particular criterion. We say that the specialisation is non-exclusive, and supertype instances corresponding to instances of its subtypes form overlapping subsets.

These properties may be formalised as follows:

$$\begin{align*}
\text{GenSpecCardinality} & \quad \text{GenSpecModel} \\
\text{SubTypeCount} : \text{CONCEPTS} \leftrightarrow \mathbb{N} \\
\forall x : \text{Criteria} \cdot (\exists y : \text{Cardinality} \cdot \text{first } y = x) \\
\forall y : \text{Criteria} \cdot \text{SubTypeCount}(y) = \\
\# \{ z : \text{SubTypes} \mid (z \mapsto y) \in \text{Specialisation} \} \\
\text{dom} \text{SubTypeCount} = \text{Criteria} \\
\forall (C, u : \text{Cardinality} \cdot \text{Criteria} = \\
(\text{first } g \in (0, 1) \land \\
\text{second } g \in (\text{1..SubTypeCount}(C)) \land \\
\text{second } g \geq \text{first } g))
\end{align*}$$

The symbol $\#$ stands for the count of elements in a set.

In the previous example, $\text{Cardinality} (\text{Position}) = (0, 1)$ (i.e. the specialisation is optional and exclusive) has the following meaning:

- the lower bound of 0 means that a Faculty member may exist without being either a Professor or an Assistant, while
- the upper bound of 1 means that each Faculty member may be a Professor, or an Assistant, but not both at the same time.

We might also distinguish between Faculty members according to their Sex, and let them be specialised as Females and Males (note that the diagram would be practically the same as that in Fig. 4). In this case, the $\text{Cardinality}$ of this $\text{Criterion}$ would be $(1, 1)$: i.e. each Faculty member must be either Female or Male, but not both at the same time.

4.2 Identification dependence

Another type of relationship is the identification dependence, which relates exactly two object types: instances of the dependent type cannot be distinguished without knowing the instance(s) of the object type they depend on. The identification-dependent object type is called characteristic, or weak entity type, and the object type it depends on is sometimes called its superordinate.

For example, a StreetAddress is a weak entity, since it is dependent on the Town entity: i.e. a StreetAddress is not a complete address without the information provided by the Town entity. The identification dependence relationship and $\text{WeakType}$ entities have special graphical symbols, as shown in Fig. 5. Weak entity rectangles have double edges, while the identification dependence is depicted with a directed line from the superordinate to the weak entity; a large triangle (pointing to the weak entity) is positioned somewhere along the line.

Formal representation of these objects and their relationships requires a separate binary relation, $\text{DependentOn}$, similar to $\text{Forming}$ or $\text{SuperType}$ relations:

$$\begin{align*}
\text{IdDepModel} & \quad \text{Model} \\
\text{DependentOn} : \text{CONCEPTS} \leftrightarrow \text{CONCEPTS} \\
\text{DependentOn} = \{ x \mapsto y \mid \\
x \in \text{Objects} \land y \in \text{WeakTypes} \} \\
\forall w : \text{WeakTypes} \cdot \\
(\exists u : \text{Objects} \cdot w = \text{DependentOn}(u))
\end{align*}$$

From the second predicate we may conclude that the inverse of $\text{DependentOn}$ is a total function from $\text{WeakTypes}$ to $\text{Objects}$, since each weak entity must depend on some other object.

From Fig. 5, the following expressions hold:

- $\text{University} \in \text{Entities}$
- $\text{Department} \in \text{WeakTypes}$
- $(\text{University} \mapsto \text{Department}) \in \text{DependentOn}$

Circular definitions are forbidden here as well:

$$\begin{align*}
\text{IdDepCycleCheck} & \quad \text{IdDepModel} \quad \text{report! : TEXT} \\
\text{report!} = "\text{No cycles in id.dep. definition}"
\end{align*}$$

As is the case with other relationship types, identification dependence may have an associated cardinality. In this case, cardinality bounds denote minimum and maximum number of weak entity instances that can be related to a single instance of the superordinate object.
4.3 Some final notes on objects

Finally, we can give a more complete definition of the Entities concept. Intuitively, entities could be understood as objects which exist by themselves, unlike associations (which require other objects to form them). However, isolated entities are not relevant for modelling, and only objects which are related to other objects may be considered entities. Relationships include Roles, Criteria and DependentOn relationships, and the following holds:

\[ \text{dom Cardinality} = \text{Roles} \cup \text{Criteria} \cup \text{DependentOn} \]

For example, the following expression means that each University must have no less than two Departments (there is no upper bound):

\[ \text{Cardinality (University} \rightarrow \text{Department)} = (2, M) \]

Since each weak entity instance corresponds to exactly one superordinate object instance, the inverse cardinality is always \((1, 1)\), e.g. each Department belongs to exactly one University.

In conclusion, since cardinality is defined for concepts of Roles, Criteria and DependentOn only, we may conclude that the following holds:

\[ \text{dom Relationship} = \text{Roles} \cup \text{Criteria} \cup \text{DependentOn} \]

5. Attributes

Attributes are named properties, which objects in each object type have in common. Each object instance has some properties, for which the values provide qualitative information (description), or sometimes a quantitative measure. In our model, attributes are CONCEPTS in their own right, just like objects, roles and other things, but different from any of them. For instance, attributes of the Student object type might be her/his name, address, phone number, student ID, and grade; the Course object type may be characterised by its name and code.

For different object types, attributes may appear in different capacities (or roles, but the use of this term could create confusion). For example, each kernel entity must have one or more KeyAttributes, the values of which uniquely determine instances of that entity:

\[ \text{Kernels} = \text{Objects} \setminus \text{DependentObjects} \]

It may be easily concluded that all object types defined thus far are mutually disjoint.

We will also make use of a binary relation which would link dependent objects with the objects they depend on:

\[ \text{Dependence} = \text{Forming} \cup \text{SuperType} \cup \text{DependentOn} \]

Note that all cycle checks shown so far did not account for cycles formed by relationships of different types. The following schema will account for such cycles:

\[ \text{CycleCheck} \]

\[ \text{ModelF} \]

\[ \text{ModelIS} \]

\[ \text{IdDepModel} \]
The second case (null values as inapplicable property) may be eliminated by proper use of the generalisation/specialisation relationship. Namely, if a property (i.e. an attribute) does not apply to some instances of an object, a new specialisation criterion should be introduced, together with at least two subtype entities. Then, all instances to which a particular property applies should be grouped into one subtype object, while instances of other subtype object(s) will not have that attribute at all. For example, some, but not all, Employees have the property TypingSpeed (and the appropriate attribute). In order to avoid null values for this attribute, we can introduce a JobTitle criterion, and partition all instances of Employees into two groups, or SubTypes: Secretaries, to which the TypingSpeed property applies, and Research Assistants, to which it does not apply.

Not all attributes tolerate null values. In particular, since one null value cannot be distinguished from the other, each KeyAttribute or PartialKey attribute must have a value, and, hence, null values are allowed for Descriptors only. This is reflected in the following schema:

<table>
<thead>
<tr>
<th>NullsModel</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NullsAllowed(x)</code></td>
<td><code>true, false</code></td>
</tr>
</tbody>
</table>

### 6. RELATIONAL IMPLEMENTATION

Most modern database management systems are based on the relational model. One possible realisation of our object model would utilise a commercially available relational database system. Therefore, it might be interesting to show the schema transformation from our model to the relational form. But first, a brief formal definition of the basic concepts of the relational model is in order. This definition will closely parallel the definition of our model, so as to facilitate the transformation, but without loss of generality.

A relational schema is a set of relations, each with a number of named attributes. A relation instance consists of a set of tuples, whose components correspond to the relation attributes. The primary key of a relation is a subset of its attributes for which the following properties hold:

1. **Uniqueness.** For each possible value of the key attributes, there can be at most one tuple with that value. In other words, the value of the key attribute(s) must be unique.
2. **Minimality.** No attribute can be excluded from the key without losing the uniqueness property (i.e. no proper subset has that property).

Other attributes are considered descriptive, and they may have duplicate values, or even no value (i.e. a null value). Note that a relation may not exist without having any descriptive attributes.

- **Attribute values** are elements of the corresponding domains, which may be considered as sets of allowed attribute values. In practice, it suffices to consider domains as elementary data types available, e.g. short and long integers, fixed-length character strings, floating-point numbers and the like, including special implementation-dependent data and time formats. However, these definitions may easily be extended to include structured data values like arrays, structures, lists, stacks, binary trees, and others as well.

Note that this definition is general enough to include even derived attributes; they are not explicitly mentioned, but neither are they prohibited. This is one of the topics to be investigated in further research.

Some of the domains require other information, i.e. whether null values are permitted or not. If an object instance has a null value for an attribute, it means that no value exists for that attribute, either because the proper value is as yet unknown, or because that property does not apply to that object. For example, when a Student registers for a Course, her/his Grade is unknown at the moment, but we must make provision for its value; it will become known as soon as she/he passes the appropriate exam.
In order to determine key attributes of relations, one must consider a number of different cases. The simplest one is when the object is a kernel entity: its Key Attributes will become RelationKeys of the corresponding relation, and that is all there is to it. On the other hand, relations corresponding to DependentObjects inherit key attributes from relations corresponding to objects they depend on. For example, key attributes of relations linked to objects which participate in an association will become RelationKeys of the relation linked to that association. Or, if the relation corresponds to a SubType entity, the keys of the relation associated to its supertype object will become its RelationKeys. Finally, keys of the relation associated to a WeakType entity are obtained as keys of the relation corresponding to its superordinate object, combined with its own PartialKeys. However, the real problem lies in the fact that a DependentObject may depend on some other DependentObject, and it is not always easy to determine which key attributes are to be inherited.

Note that we have already defined the Dependence relation, which links a dependent object with the object(s) it depends on. Since these objects may themselves depend on other objects, we have also used the transitive closure of that relation, Dependence*, to find out all objects (on all levels) that a given dependent object depends on. For each dependent object, the objects it depends on will be arranged in the form of an inverted tree; the root of the dependency tree will be the object itself, its nodes will be the objects it depends on, and branches would be relationships of either Forming, SuperType or DependentOn variety. All leaves, i.e. nodes which do not depend on any other objects, must belong to the Kernel entities.

As for key attributes, note that only Kernel entities have proprietary keys, and WeakTypes have partial keys; Associations and SubTypes have no key attributes of their own. Therefore, key attributes for a given dependent object could be obtained as the union of KeyAttributes of all Kernel entities, and PartialKeys of all WeakTypes, that belong to the dependency tree defined by the Dependence* relation. However, in order to account for proprietary PartialKeys, we must use the reflexive-transitive closure, Dependence*, which includes not only the usual transitive closure, but the identity relation as well.

\[
\forall x: \text{Objects}; y: \text{RelationKey} \quad (x \rightarrow y) \in \text{RelationKey} \Rightarrow
\]

The second property just forbids any attribute appearing as both key and descriptive attribute in the same relation.

It should be noted that we are not interested in why some attribute is part of the key; we are simply interested in whether an attribute is part of the key or not. Full formal description of the relational model, including functional, multivalued and other dependencies, is beyond the scope of this paper; normal forms which are the basis of relational model design will not be discussed either.

Rules for transformation of the XER model to a relational one will be presented now. A uniform solution would be to transform each object (entity or association) to a separate relation, whose attributes will be determined soon. This correspondence is described by the following schema:

\[
\text{TransformModel}
\]

\[
\text{RelationalModel}
\]

\[
XER2RM: \text{CONCEPTS} \rightarrow \text{CONCEPTS}
\]

\[
\text{Relations} = \{y: \text{CONCEPTS}\}
\]

\[
(3, x: \text{Objects} \bullet (x \rightarrow y) \in \text{XER2RM})
\]

\[
\forall x: \text{Objects} \bullet
\]

\[
(3, y: \text{Relations} \bullet (x \rightarrow y) \in \text{XER2RM})
\]

\[
\forall (x_1, y_1), (x_2, y_2): \text{XER2RM} \bullet y_1 = y_2 \Rightarrow x_1 = x_2
\]

These predicates mean that the XER2RM relation is, in fact, a bijection (a mapping which is both 1-1 and onto), since each Object has a corresponding Relation and vice versa, and, in addition, different Objects correspond to different Relations. The same statement may be given in a concise manner:

\[
\text{XER2RM} = \text{Objects} \rightarrow \text{Relations}
\]

Since both Relations and Objects are CONCEPTS, we might impose a condition that their names differ by, say, a unique prefix:

\[
\forall x: \text{Objects}; y: \text{Relations} \bullet (x \rightarrow y) \in \text{XER2RM} \Rightarrow y = " R." ^\vee x
\]

The symbol ^ stands for string concatenation.\text{1}\text{1}

As is the case with other formal definitions, the TransformModel schema just states that each object has an associated relation; it provides no description of the actual transformation process. However, if we substitute iteration (e.g. foreach or other similar construct) for universal quantifiers (\forall ... ) and set comprehension ([...])...}, and assignment for the equals sign '=' , the appropriate algorithm could be easily arrived at.

It remains to allocate Attributes of Objects to their appropriate Relations. Allocation is rather straightforward for Descriptors: they are simply inherited from the object in question, and linked to the corresponding Relation via RelationAttributes.

\[
\forall x: \text{Objects}; y: \text{Attributes} \bullet
\]

\[
(x \rightarrow y) \in \text{RelationKey} \Rightarrow
\]

\[
(x \rightarrow y) \notin \text{RelationAttributes}
\]

As a result of the use of reflexive-transitive closure, we were able to make our schema as general as possible: it includes even the case of Kernel entities.
The RelationKeys relation may be rearranged to read
\[ RelationKeys \triangleq XER2RM^{-\circ} \text{ Dependence}^{\circ} \quad \text{(Identifies} \cup \text{Discriminates)} \]

With this in mind, we could rewrite the last predicate of the RelationalAttributes schema as
\[ RelationAttributes = XER2RM^{-\circ} \text{ Describes} \]

Note that the inverse of the \( XER2RM \) relation is a binary function, since \( XER2RM \) is, in fact, a surjection.

As an example, consider the XER model shown in Fig. 6, parts of which have already been presented in the paper, and its associated attributes, summarised for convenience in Table 1. After the transformations described above are performed, we obtain the equivalent relational schema with the appropriate attributes, as shown in Table 2. Note that relations which correspond to DependentObjects have inherited key attributes from all objects they depend on. For example, Stud_Course depends on Student and Course, and Lecture...
Table 2. Relational attributes of the example model

<table>
<thead>
<tr>
<th>Relations</th>
<th>RelationKeys</th>
<th>RelationAttributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Faculty</td>
<td>SSN</td>
<td>FName, FAddress</td>
</tr>
<tr>
<td>R_Professor</td>
<td>SSN</td>
<td>FName, FAddress, Rank, PhD</td>
</tr>
<tr>
<td>R_Assistant</td>
<td>SSN</td>
<td>FName, FAddress, YearsExperience</td>
</tr>
<tr>
<td>R_Dependent</td>
<td>SSN, DName</td>
<td>Relationship, Age</td>
</tr>
<tr>
<td>R_Course</td>
<td>Course_Id</td>
<td>CName, Credits, RefText</td>
</tr>
<tr>
<td>R_Pre_Requisite</td>
<td>Course_Id_Requires, Course_Id_Required_By</td>
<td>—</td>
</tr>
<tr>
<td>R_Teacher_Course</td>
<td>SSN, Course_Id</td>
<td>Hours</td>
</tr>
<tr>
<td>R_Student</td>
<td>Student_Id</td>
<td>SName, SAddress</td>
</tr>
<tr>
<td>R_Stud_Course</td>
<td>Stud_Id, Course_Id</td>
<td>Grade</td>
</tr>
<tr>
<td>R_Room</td>
<td>RoomNo</td>
<td>NoSeats</td>
</tr>
<tr>
<td>R_Lecture</td>
<td>RoomNo, Stud_Id, Course_Id</td>
<td>Hour</td>
</tr>
</tbody>
</table>

depends on Student_Course (i.e. on Student and Course) and Room; therefore, their KeyAttributes appear as RelationKeys of the R_Stud_Course and R_Lecture relations.

The Table also reveals a potential problem which we have not discussed as yet. Note that RelationKeys of the R_Pre_Requisite relation contain two instances of the Course_Id attribute, which is perfectly legal; remember that the Course object participates in the Pre_Requisite association in two different Roles. However, the names of these attribute instances must differ, since the RelationKeys relation cannot contain more than a single maplet of the form

\[ R_{Pre-Requisite} \rightarrow \text{Course}_\text{Id} \]

RelationKeys is a proper set, not a bag, and multiple instances of a single element are not allowed. This case might be handled by augmenting the name of either attribute instance (or both of them) with the name(s) of the appropriate mapping(s), so as to make these names different, as shown in Table 2.

7. FINAL REMARKS

A synopsis of data dictionary for an extended entity-relationship data model is presented using the well-known Z notation. Various concepts of the model are formally defined, and some of their properties are derived. The data dictionary comprises various object types, including entities of different kinds, and their relationships. The relationships are not limited to common associations, but include generalisation/specialisation relationships and identification dependence as well. Different attribute types and their relationships with objects are also discussed. Finally, we have shown that the XER model may easily be transformed to a relational model, in order to implement it; a formal specification of this transformation is also presented. In all cases, the use of formal specification techniques contributes to the clarity and conciseness of specifications, and enables formal derivation of model properties to be performed easily.

Obviously, the work reported is but a beginning of formal development of an extended ER data model, since it includes just the description of the model structure. Indeed, such a formal description of the XER model is almost complete at the time of this writing, and it includes update and retrieval operations, and integrity constraints. Attribute domains, entity integrity (i.e. uniqueness based on defined keys), and structural integrity constraints are covered. A methodology for definition of structure and behaviour of complex objects, corresponding to external user views, would exploit the benefits of semantic expressiveness and formal foundations of the XER model. However, these results will be the subject of another paper.
APPENDIX: Z NOTATION USED

Z is a specification language based on weakly typed set algebra; a detailed description of the notation is given in Ref 11. We will present only a brief informal overview, in order to clarify the notation which indeed may not be easy to follow at first.

Most common mathematical symbols retain their meaning in Z specifications, e.g. notation for numbers (N and \( \mathbb{N} \)), first-order logic operations (\( \land, \lor, \Rightarrow \)) and quantifiers (\( \forall, \exists \), e.g. there exists exactly one, \( \exists ! \)). Set relations and operations, such as \( \in \) and \( \notin \) for membership and its negation, \( \subseteq \) and \( \subset \) for ‘subset of’ and ‘proper subset of’, and \( \cup, \cap, \setminus \) and \( \times \), for union, intersection, set difference and Cartesian product, are also provided.

Set comprehension allows definition of sets with arbitrary characteristics, e.g.

\[
\text{Roles} = \{ x \in \text{CONCEPTS} | (\exists y : \text{Objects} \cdot (x \mapsto y) \in \text{MappedTo}) \}
\]

It should be read as: \( \text{Roles} \) is the set of all elements \( x \) of the \( \text{CONCEPTS} \) set, such that there exists some element \( y \) from the \( \text{Objects} \) set with the property that the maplet \( x \mapsto y \) (or the ordered pair \( (x, y) \)) belongs to the \( \text{MappedTo} \) relation.

Z has a multitude of symbols for denoting various binary relations and functions. \( A \rightarrow B \) stands for a binary relation \( R \), i.e. a set of pairs \( (a, b) \) such that \( a \in A \) and \( b \in B \). The set of \( as \) which appear as first components of pairs in \( R \) form its domain, \( \text{dom} R \), and the set of \( bs \) which appear as second components form its range or co-domain \( R \). The inverse of a relation, \( R^{-1} \), is a set of pairs \( (b, a) \), i.e. the components have swapped places. Components of a given pair may be extracted with the aid of the functions \( \text{first} \) and \( \text{second} \).

Relations in which no element of the first set appears more than once are called functions, which is written as \( F = A \rightarrow B \) or, sometimes, as \( B = F(A) \). If all elements of the first set appear in the function, the function is a total one, otherwise it is partial. A function \( F = A \rightarrow B \) is an injection, \( A \rightarrowrightarrow B \), if different \( bs \) correspond to different

as (this is also termed a 1–1 mapping); it is a surjection, \( A \rightarrowrightarrow B \), if the entire \( B \) set is its range (this mapping is onto). Injections and surjection may be total as well; a total injective surjection is termed a bijection, or a total mapping which is both 1–1 and onto.

A named Z schema defines some properties, or invariants; it consists of a declaration part (where variables of various types are defined) and a predicate part, where properties of these variables (i.e. predicates in which they take part) are formally defined, and it may be read as: given these declarations..., the following predicates... hold. The schema may also describe an operation, in which case the predicate part includes both the precondition and the postcondition of the operation. Schemas are a convenient way of enhancing the readability of specifications, and of structuring them, albeit at a very rudimentary level. Schemas may be combined using logical connectives ‘and’, ‘or’ and ‘not’; various other schema-combining operators (under the common name of schema calculus) are available as well. Furthermore, a schema may include one or more other schemas in its declaration part, e.g.

\[
\text{ModelF} \quad \text{RoleModel} \quad \text{Forming} : \text{CONCEPTS} \leftarrow \text{CONCEPTS} \\
\text{Forming} = \text{FormedBy}^* \_ \_ \_ \text{MappedTo}
\]

in which case the declarations and predicates of the included schema(s) are merged with the declarations and predicates, respectively, of the main one.

A detailed account of the Z notation is beyond the scope of this Appendix, and the interested reader may consult the excellent texts available.5, 11

Acknowledgements

The authors gratefully acknowledge the comments of the anonymous reviewers, which considerably improved the readability of the paper, and the kind donation of the oz.sty \LaTeX\ style by Paul King of University of Queensland, Australia.