TECHNICAL REPORT NO. 404

Leveled Entity Relationship Model

Munish Gandhi * Edward L. Robertson*

gandhim@cs.indiana.edu edrbtsn@cs.indiana.edu

Dirk Van Gucht*

vgucht@cs.indiana.edu

May 25, 1994

*Computer Science Department, Indiana University, Bloomington, IN-47405.

Abstract

The Entity-Relationship formalism, introduced in the mid-seventies, is an extensively used tool for database design. The database community is now involved in building the next generation of database systems. However, there is no effective formalism similar to ER for modeling the complex data in these systems. We propose the *Leveled Entity Relationship (LER)* formalism as a step towards fulfilling such a need.

An essential characteristic of these next-generation systems is that a data element is no longer assumed to be atomic, instead it could have a complex internal structure. The LER formalism models structured data by leveling ER diagrams such that deeper layers provide greater structural detail. Moreover, elements deep inside one structure reference elements deep inside another structure. LER cleanly formalizes the relation between such deep structural elements.

Contents

1 Introduction **6**

1.1 A simple LER example 8

2 Definition **9**

2.1 Concepts 9

2.1.1 Entity 10

2.1.2 Aspect 11

2.1.3 Relationship 12

3 LER Formalism **13**

3.1 Schema 13

3.2 Instance 16

4 Representations **18**

4.1 LER diagram 18

4.2 Textual Representation 22

5 LER and the Real World **25**

5.1 Detailed Example: A Software Development Enterprise . . . 25

5.2 Mapping the concepts 28

5.2.1 Mapping to entities and sub-entities 28

5.2.2 Mapping to relationships 30

5.2.3 Mapping to aspects 31

6 Related Work and Comparisons **32**

6.1 Non-encroaching relationships 33

6.2 Consistent refinements 33

6.3 Well-defined refinement 34

7 Conclusion **35**

List of Figures

1 A simple LER diagram 9

2 Schema L = $(\mathcal{E}, \mathcal{A}, \mathcal{R}, \epsilon, \alpha, \rho)$ 15

3 Instance I = $(\iota_{\mathcal{E}}, \iota_{\mathcal{R}})$ 19

4 Notation for aspects 19

5 Notation for entities 20

6 Entity Definition: METHOD, CLASS and DEFINITION . . 21

7 CLASS DEFINITION top-level schema 21

8 Textual syntax 23

9 Textual representation for example 24

10 Software Deliverables 25

11 Glass-box view: PRODUCT 26

12 Glass-box view: USER MANUAL 27

1 Introduction

A model is an abstract representation of some objective reality. For a model to effectively communicate, the mismatch between our conceptualization of the reality and its representation in the model should be minimal. Consider a software engineering example. An engineer views software as a hierarchy. For instance, a software package is a collection of programs, which in turn is a structured set of classes, each of which has some methods. However, an ER diagram would not reflect the hierarchical nature of the above. This "cognitive dissonance" reduces the communication capability of the model. Continuing our example, a manager interested in knowing about software packages would be distracted by the unnecessary complexity in the schema arising from elements associated with methods. A formalism which permits layering would enable one to concentrate on entities and relationships at one layer of abstraction and tune out elements from other layers.

These benefits of layering ER diagrams have been recognized and many models [FM86, Har88, TWBK89, LV89, CJA90] which layer ER diagrams have been proposed (section 6 gives their salient features). A common thread runs through these approaches. An ER diagram at a lower level is abstracted to an entity at a higher level. However, these formalisms have some drawbacks.

- The exact nature of the association between an entity at a higher level and a sub-entity at a lower level remains poorly defined. Thus, such models are used only as communication tools – the database corresponds to only the lowermost layer and can not take advantage of the layering information in the design.

- A relationship between a subentity S in E and an entity on the same layer as E can not be established without breaking the encapsulation of E. This disallows a clean abstraction mechanism.

- Refining an entity (abstracting an entity cluster) results in addition (deletion) of relationships to entities other than the entity being refined. This makes successive refinement a process which cascades changes throughout the enterprise.

We defer a more detailed discussion of these drawbacks, their implications and how the current proposal overcomes them to a section (section 6) which occurs after we have defined our model.

The *Leveled Entity Relationship (LER)* model is an adaptation of the traditional ER layering models with two core features to overcome the above drawbacks. The first drawback may be overcome by necessitating an entity at a higher level to be present at a level below as *self*. The *self* represents the higher level entity in its elemental form, and is used as a focal point in organizing the ER diagram below. The second and third drawbacks are overcome by adding a mechanism to make sub-entities deep inside an entity visible to the external world without unnecessary complexities – an ER analog of information hiding. This is done by making the lower level element appear as an *aspect* of the higher level entity.

In this paper, the model is considered from the diagrammatic standpoint. We first define the various elements in the LER model (section 2), but consign the formalism to the appendix (appendix 3) where we formalize the notion of an LER schema (section 3.1) and give it semantics by considering its instances (section 3.2). After a discussion of the elements in the LER

model, we present a diagrammatic representation to serve as an intuitive tool for communicating designs of leveled systems (section 4.1).

Having defined LER diagrams, we turn to the task of exposing how they may be used to model real-world structures (section 5). A detailed example illustrates various LER concepts (section 5.1). The example also prompts a discussion of the mapping between real-world concepts and LER constructs (section 5.2). At this point, we are in a position to discuss related work and how this model overcomes the drawbacks in the previous models (section 6).

1.1 A simple LER example

Before we begin the formalism, we develop the intuitions behind LER using an example. The example reflects a software engineering need to coordinate a software development effort using a central repository. In fact, this research was prompted by the inadequacy of the current tools to naturally model software engineering repository schemas. Figure 1 represents a schema about classes, their methods and the definition of the methods.

There are three entities of interest: `CLASS`, `METHOD` and `DEFINITION`. The properties of entities are represented using *aspects*[1]. For a `CLASS`, we are interested in knowing its name and methods. Thus, a `CLASS` has an aspect `Name`, and an aspect `Method`. For a `DEFINITION`, we are interested in its `Body`. Similarly, for a `METHOD`, we are interested in the method's `Declaration`.

[1]The word 'aspect' denotes the perception of a certain property of an item or thing. In this sense, it is an *attribute* of the abstract entity. The word is also used to connote the view of an item from a particular position. In this sense, an aspect is a view from the outside of an abstract entity to only a selected entity inside the abstract entity.

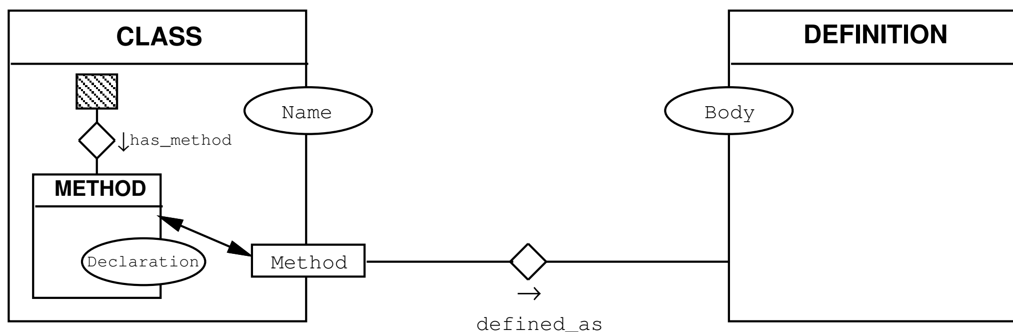

Figure 1: A simple LER diagram

Consider the internal structure for a CLASS. The internal relationship has_method specifies the METHODs associated with a CLASS. For each CLASS instance, the aspect Method should contain exactly those METHOD instances that are related by the has_method relationship to that CLASS. This is established using a correspondence between the aspect Method and the internal entity METHOD.

Each METHOD has a definition and we would like to represent that without breaking the CLASS encapsulation. This is done using the defined_as relationship, which relates the aspect Method with the DEFINITION.

2 Definition

2.1 Concepts

The Leveled Entity Relationship (LER) model extends the essential concepts in the ER model [Che76] by enriching them to fulfill the special requirements for modeling structured data. An LER *entity* may be atomic (like an ER entity) or it may have an internal structure. An *aspect* of an entity may

be a direct property of the entity (like an ER attribute) or it may reflect an internal facet of the entity. An LER *relationship* may directly associate two entities (like an ER relationship) or it may associate entities by linking subentities within them.

We now consider the LER concepts in detail. First, we provide an informal description. This is followed by a formal (symbolic) definition for schema and instances. An example schema and instance illustrate the definitions.

2.1.1 Entity

Informal definition: An objective or conceptual reality that has a distinct existence.

The world of entities may be observed at different levels of abstraction. At the highest level, an entity is perceived atomically. However, it may have a structure which becomes apparent only when observed at lower levels of abstraction. For example, a C++ class may be considered an atomic entity at some granularity level. At a lower granularity, the class would declare many methods. An entity with such a structure at a lower level of abstraction is a *complex entity.* An entity with no such structure is a *simple entity.*

For a simple entity, we may specify only its external structure, that is, its aspects. For a complex entity, we also need to specify its internal structure. This is done using a specially constrained LER and *correspondences* linking the internal and external structures. An entity type is defined by the internal and external structures for entities with that type, and references to the type are made using an *entity name.*

2.1.2 Aspect

Informal definition: The outward appearance of an inherent feature of the entity.

The values that these features may take come from a domain of "printable" values or OIDs. Aspects having a domain of printable values are akin to attributes of the ER model. We refer to these aspects as being *value-based*. Aspects with a domain of OIDs are *oid-based*. Each entity has an oid-based aspect which reflects the core identity of the entity. This concept of a self-specifying aspect for every entity is similar to a surrogate for entity every relation in RM/T [Cod79]. We'll sometimes refer to this aspect as "self", even though it has the same name as its entity.

An aspect may be *single-valued* or *multi-valued* depending on whether the corresponding feature needs a single element or a set of elements from its domain to describe itself completely. As an example, the name of class may be single-valued, while the methods multi-valued.

The informal definition of an aspect suggests that an aspect is more than just a property of a entity. It may also serve as a window exposing some internal features of the entity. For example, the names of a class's methods may correspond to names of sub-entities internal to the class abstraction. Since such aspects derive their values from a deeper structure, we label such aspects as *deep* aspects. Those which directly describe the entity are labeled *immediate* aspects.

As far the external structure is concerned, there should be no need to know whether an aspect derives its values from an internal element or not. Thus, a deep aspect serves to externalize the internals of an entity without

breaking its encapsulation. This proves to be critical. Breaking the encapsulation to reach the deeper elements renders all structuring essentially artificial.

2.1.3 Relationship

Informal definition: An association between two entities.

The association is formalized by linking two oid-based aspects. While This corresponds to the ER formalism because the entity identities are captured by oid-based aspects.

A relationship between the self-describing aspects of two entities represents a direct association between the two entities. Frequently, one needs to associate with an entity *within* another entity. This is done by bringing the identity (oid) of the interior entity to the periphery of the exterior entity using an aspect. For example, a `method` declaration inside a `class` is associated with its `definition` outside the `class` by relating `definition` to the `class` aspect corresponding to the internal `method` entity. Thus, one may associate entities internal to other entities without breaking the encapsulation of the containing entities.

Continuing our example, it may seem that a relationship associates a set of `methods` of a `class` with some `definition`. However, the semantics ensure that *each* `method` has its `definition`. In general, a relationship with a multi-valued aspect is interpreted as an association with *each* value of that aspect.

3 LER Formalism

We assume the existence of the following countably infinite and pairwise disjoint sets of atomic elements:

- *Aspect names. Entity names* are a subset of aspect names.
- *Relationship names.*
- *Entity identifiers.*
- *Values.*

The entity identifiers and values are organized into a set of *domains* $\mathcal{D}$, where $\mathcal{D} = \{OID, D_1, D_2, \ldots\}$ such that OID is the set of entity identifiers, and each D_i is a subset of the set of values. An LER schema is defined in terms of aspect names, relationship names and domains, and an LER instance in terms of entity identifiers and values.

In this section, we will denote aspect names by $A, A', A_1, A_2, \ldots$, entity names by $E, E', E_1, E_2, \ldots$, relationship names by $R, R', R_1, R_2, \ldots$, entity identifiers by $e, e', e_1, e_2, \ldots$ and values by $v, v', v_1, v_2, \ldots$.

3.1 Schema

An LER Schema L is defined by $(\mathcal{E}, \mathcal{A}, \mathcal{R}, \epsilon, \alpha, \rho)$ where

1. $\mathcal{E}$ is a finite set of entity names.
2. $\mathcal{A}$ is a finite set of aspect names. Also, $\mathcal{E} \subset \mathcal{A}$.
3. $\mathcal{R}$ is a finite set of relationship names.

4. α is a function from $\mathcal{A}$ into $\mathcal{D} \times \{s, m\}$. If $A \in \mathcal{E}$, $\alpha(A) = (OID, s)$.

   If the first component of $\alpha(A)$ is D, then we say the domain of A is D (or, $dom(A) = D$). A is said to be *oid-based* if the domain of A is OID, else A is *value-based.* If the second component of $\alpha(A)$ is s, then we say that the aspect is *single-valued.* Otherwise, the aspect is *multi-valued.*

   We refer to the set of oid-based aspect names as $\mathcal{O}$ and to elements from this set as $O, O', O_1, O_2, \ldots$. Note that $\mathcal{E} \subset \mathcal{O} \subset \mathcal{A}$.

5. ρ is a total function from $\mathcal{R}$ into $\mathcal{O} \times \mathcal{O}$.

6. ϵ is a total function from $\mathcal{E}$ onto a set of quadruples such that $\epsilon(E) = (\mathcal{A}_E, \mathcal{E}_E, \mathcal{R}_E, \kappa_E)$ where

   (a) $\mathcal{A}_E \subset \mathcal{A}$ is the set of aspects *of* an entity. Also, $E \in \mathcal{A}_E$.

   (b) $\mathcal{E}_E \subset \mathcal{E}$ is the set of entities *within* an entity. We refer to $E \cup \bigcup_{E' \in \mathcal{E}_E} A'_E$ (say, $= \mathcal{B}$) as the set of aspects *within* E.

   (c) $\mathcal{R}_E \subset \mathcal{R}$ is a set of relationships *within* an entity. Also, for each $R \in \mathcal{R}_E$ such that $\rho(R) = (O, O')$, O, O' are aspects within E.

   (d) κ_E is a (partial) function from $\mathcal{A}_E - \{E\}$ to $\mathcal{B}$, such that for $\kappa_E(A) = A', dom(A) = dom(A')$. It defines a *correspondence* between aspects of E and aspects within E.

   Those aspects in A_E for which κ_E is defined are *deep* aspects. Aspects which are not deep are *immediate* aspects.

An entity E such that $\mathcal{E}_E = \mathcal{R}_E = \kappa_E = \phi$ is a *simple entity.* Other entities are *complex entities.*

$\mathcal{E} = \{METHOD, DEFINITION, CLASS\}$

$\mathcal{A} = \{Method, Name, Body, Declaration\} \cup \{\mathcal{E}\}$

$\mathcal{R} = \{defined_as, has_method\}$

ϵ has the following values (over $\epsilon(E) = (\mathcal{A}_E, \mathcal{E}_E, \mathcal{R}_E, \kappa_E)$)

$$\begin{aligned}
\epsilon(METHOD) &= \{\{METHOD, Name\}, \phi, \phi, \phi\} \\
\epsilon(DEFINITION) &= \{\{DEFINITION, Body\}, \phi, \phi, \phi\} \\
\epsilon(CLASS) &= \{\{CLASS, Name, Method\}, \{METHOD\}, \\
&\quad \{has_method\}, \{(Method, METHOD)\}\}
\end{aligned}$$

α has the following values (over $\mathcal{D} \times \{s, m\}$)

$$\begin{aligned}
\alpha(METHOD) &= \alpha(DEFINITION) = \alpha(CLASS) = (OID, s) \\
\alpha(Method) &= (OID, m) \\
\alpha(Name) &= (String, s) \\
\alpha(Body) &= (String, s) \\
\alpha(Declaration) &= (String, s)
\end{aligned}$$

ρ has the following values

$$\begin{aligned}
\rho(has_method) &= (CLASS, METHOD) \\
\rho(defined_as) &= (CLASS, DEFINITION)
\end{aligned}$$

Figure 2: Schema L = $(\mathcal{E}, \mathcal{A}, \mathcal{R}, \epsilon, \alpha, \rho)$

Figure 2 presents the schema for our introductory example.

3.2 Instance

An instance of an LER Schema L = $(\mathcal{E}, \mathcal{A}, \mathcal{R}, \epsilon, \alpha, \rho)$ is a pair $\iota = (\iota_{\mathcal{E}}, \iota_{\mathcal{R}})$ denoting the sets of entity instances and relationship instances respectively, where

- $\iota_{\mathcal{E}}$ maps each entity name in $\mathcal{E}$ to a finite set of *entity instances*. An entity instance for an entity name E, where $\epsilon(E) = (\mathcal{A}_E, \mathcal{E}_E, \mathcal{R}_E, \kappa_E)$ and $\mathcal{A}_E = \{A_1, A_2, \ldots, A_n\}$, is a set containing n aspect-value pairs $\{A_1 : v_1, A_2 : v_2, \ldots, A_n : v_n\}$, where $v_i \subset dom(A_i)$ and, if A_i is single valued, v_i is a singleton. We refer to $\iota_{\mathcal{E}}(E)$ as the *extent* of E. Values for aspects are further constrained below.

- $\iota_{\mathcal{R}}$ maps relationship names in $\mathcal{R}$ to a set of *relationship instances*. A relationship instance for an relationship name R, $\rho(R) = (O, O')$, is a pair (o, o') where $o \in \{\nu(e, O) | e \in \iota_{OID}(E), E \in \mathcal{E}\}$ and $o' \in \{\nu(e, O') | e \in \iota_{OID}(E), E \in \mathcal{E}\}$. These conditions ensure that instances of R use identifiers which are currently values for aspects O and O'.

Since each entity instance carries its own oid as a value of its "self" attribute, $\iota_{\mathcal{E}}$ suffices to delineate $\iota_{OID}(E)$, the entity identifiers for the instances of any entity type E, using $\iota_{OID}(E) \stackrel{\text{def}}{=} \{e \; | \{E : \{e\}, \ldots\} \in \iota_{\mathcal{E}}(E)\}$.

The following two conditions are required for oids to uniquely identify entity instances. The first constrains entity identifiers of distinct instances in the extent of E to be distinct. The latter dissallows the same entity identifier to be used in distinct extents.

$$|\iota_{\mathcal{E}}(E)| = |\iota_{OID}(E)|$$

$$\iota_{OID}(E) \cap \iota_{OID}(E') = \phi,\ \ for\ E' \neq E$$

The entity instances $\iota_{\mathcal{E}}$ also allow a direct definition of valuation. Define ν, the value of an aspect A for the instance given by an entity identifier e, as

$$\nu(e, A) \stackrel{\text{def}}{=} \begin{cases} v & if\ \{E : \{e\}, A : v, \ldots\} \in \iota_{\mathcal{E}}(E) \\ \phi & otherwise \end{cases}$$

Expression of further constraints require functions $\delta(e)$ and $\omega(e)$, where $\delta(e)$ is the set of entity identifiers "internal" to e and $\omega(e)$ is the set of relationship instances which directly or indirectly relate e to other instances. Formally, for an entity identifier $e \in \iota_{OID}(E)$, define $\delta(e)$ and $\omega(e)$ as the least sets, such that

$$\delta(e) \stackrel{\text{def}}{=} \{e_i \mid (e_i, e_j) \text{ or } (e_j, e_i) \in \omega(e)\} \cup \{e\}$$

$$\omega(e) \stackrel{\text{def}}{=} \left\{ (e_i, e_j) \;\middle|\; \begin{array}{l} \exists R_k \in \mathcal{R}_E, (e_i, e_j) \in \iota_{\mathcal{R}}(R_k) \\ and\ e_i \in \delta(e) \text{ or } e_j \in \delta(e) \end{array} \right\}$$

We now use $\delta(e)$ to fix the value for a deep aspect as the union of the corresponding aspect values for instances reachable from e. For $\{E : \{e\}, A : v, \ldots\} \in \iota_{\mathcal{E}}(E)$ and $\kappa_E(A) = A'$, $v = \{\nu(e_l, A') \mid e_l \in \delta(e)\}$.

Figure 3 is an instance of the schema outlined in section 3.1. It consists of classes `Stack` and `Dummy`. `Stack` has two methods declared as "Push(T)" and "T Pop()". In its definition, `Pop` returns the top element in the stack – an item of type T. `Push` does a simple return after adding an item of type T. The `Dummy` class has no methods.

4 Representations

4.1 LER diagram

In this section, we outline a diagrammatic notation for representing an LER schema. Figure 4 summarizes the notation for representing aspects in an LER schema. While there are four different aspects that we need to distinguish, we do not distinguish the two oid-based aspects. This is because the only single-valued oid-based aspect has the same name as the name of the entity itself. Other oid-based aspects reflect the contents of the deep structure; hence the single-valuedness of a deep oid-based aspect should be specified as a constraint on the deep structure.

Figure 5 summarizes the notation for representing entities and relationships. A relationship is simply represented by a diamond and its name and direction specified alongside. There are two representations for an entity. The black box view is an abstract representation of the entity and shows only its name and its aspects. The glass-box view is a more complete view of the entity. In this view, the internal entities, relationships, and aspects are displayed. Double arrowed edges represent the correspondence between aspects of the entity and aspects within the entity.

The self aspect of an entity is used to define both the internal and external structures of an entity. Externally, it identifies the entity. Internally, it is the focal point in the internal organization of the entity. In this sense, the self bridges the internal and external perspectives of the entity. Our representation distinguishes the two uses of self. Internally, the self is represented using a shaded box. Externally, it is represented as just another aspect.

A complete LER diagram consists of two sets of figures:

For each entity name in $\mathcal{E}$, the entity instances given by $\iota_{\mathcal{E}}$ are:

$$\iota_{\mathcal{E}}(\text{METHOD}) = \left\{ \begin{array}{l} \{\text{METHOD} : \{\#3\}, \text{Declaration} : \{Push(T)\}\}, \\ \{\text{METHOD} : \{\#4\}, \text{Declaration} : \{TPop()\}\} \end{array} \right\}$$

$$\iota_{\mathcal{E}}(\text{DEFINITION}) = \left\{ \begin{array}{l} \{\text{DEFINITION} : \{\#5\}, \text{Body} : \{\text{``}\ldots return;\text{''}\}\}, \\ \{\text{DEFINITION} : \{\#6\}, \text{Body} : \{\text{``}\ldots return\ item;\text{''}\}\} \end{array} \right\}$$

$$\iota_{\mathcal{E}}(\text{CLASS}) = \left\{ \begin{array}{l} \{\text{CLASS} : \{\#1\}, \text{Name} : \{Stack\}, \text{Method} : \{\#3, \#4\}\}, \\ \{\text{CLASS} : \{\#2\}, \text{Name} : \{Dummy\}, \text{Method} : \{\}\} \end{array} \right\}$$

For each relationship name in $\mathcal{R}$, the relationship instances given by $\iota_{\mathcal{R}}$ are:

$$\iota_{\mathcal{R}}(\text{has_method}) = \left\{ \begin{array}{l} \{\text{CLASS} : \#1, \text{METHOD} : \#3\}, \\ \{\text{CLASS} : \#1, \text{METHOD} : \#4\}, \end{array} \right\}$$

$$\iota_{\mathcal{R}}(\text{defined_as}) = \left\{ \begin{array}{l} \{\text{METHOD} : \#3, \text{DEFINITION} : \#5\}, \\ \{\text{METHOD} : \#4, \text{DEFINITION} : \#6\}, \end{array} \right\}$$

Figure 3: Instance I = $(\iota_{\mathcal{E}}, \iota_{\mathcal{R}})$

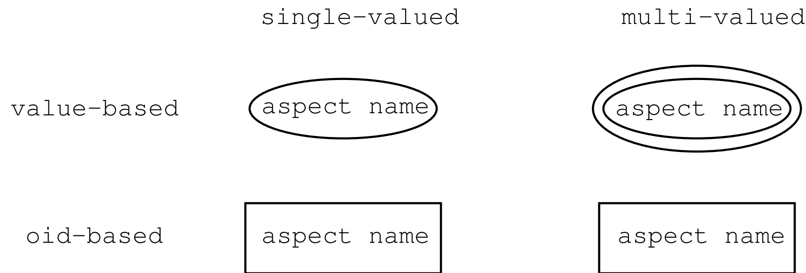

Figure 4: Notation for aspects

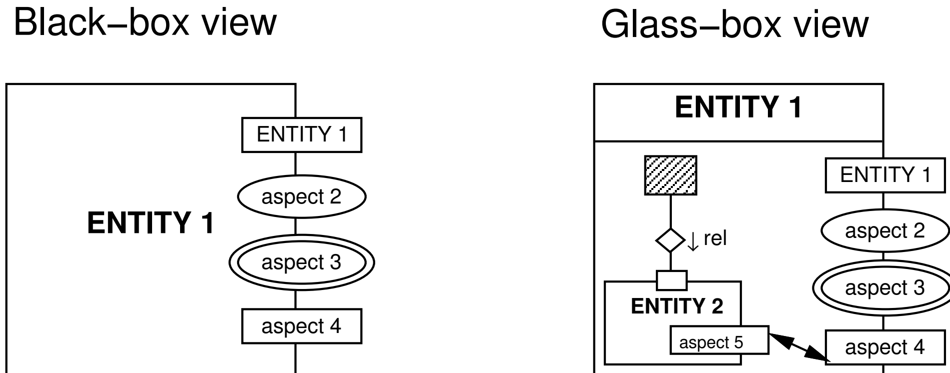

Figure 5: Notation for entities

- A set of glass-box views defining the entities of interest in a particular schema. For the example in section 3.1, figure 6 defines the glass-box views of a METHOD, CLASS and DEFINITION entities.
- A top-level diagram of the system. This captures relationships which were not within the definitions of entities. Entities may be represented in this diagram using their black-box views. Figure 7 defines the top-level diagram for our example schema.

It may be noted that the glass-box views completely define $\mathcal{E}$, $\mathcal{A}$ and ϵ. The glass-box views together with the top-level diagram completely define $\mathcal{R}$ and ρ. In the spirit of ER, we do not specify the exact domains for each aspect. If we assume that there are only two domains – the domain of values and the domain of oids – then the glass-box views also completely define α. However, it is not difficult to add a facility to specify the domain to a more precise level.

A few shortcuts may be adopted to make LER diagrams simple:

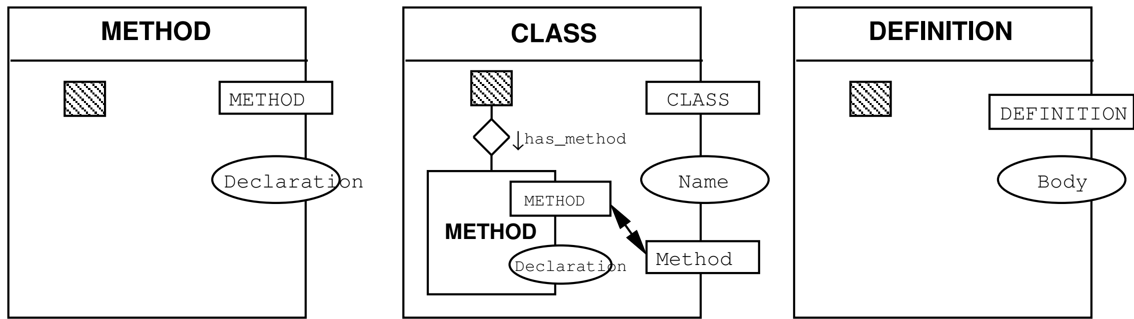

Figure 6: Entity Definition: METHOD, CLASS and DEFINITION

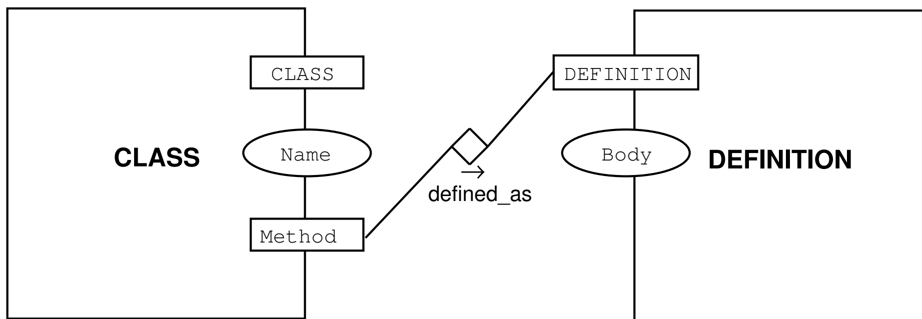

Figure 7: CLASS DEFINITION top-level schema

- Since every entity has a self-specifying aspect, we omit that from both the glass-box and the black-box views of the entity. However, we do retain the shaded box representation of the self. Furthermore, any relationship which self at one end may now directly connect to the entity itself (like a relationship in an ER diagram).

- The glass-box and the block-box views may be collapsed for compactness. Furthermore, we need not show the shaded box in the glass-box view for a simple entity.

- We need not have globally unique aspect names. Thus, the same name may be used by different entities. Globally unique names for each aspect may be obtained by simply prefixing an aspect name with its entity name.

Applications of these shortcuts results in the figure 1 in section 1.1.

4.2 Textual Representation

In this section, we outline a textual notation for representing an LER schema. A textual representation could be useful in communicating the schema between two programs.

Figure 9 uses the syntax in figure 8 to represent the schema in section 5.1. It may be noticed that the textual representation is similar to the diagrammatic representation. It includes two sets of definitions. The entity definitions correspond to the glass-box views, and the relationship definitions correspond to the top-level view. However, in the textual representation, one can specify the domain of an aspect with greater precision.

ler:

entity-defn-list relationship-defn-list

entity-defn-list:

entity-defn [*entity-defn*]*

relationship-defn-list:

`define relationships` {*relationship-defn* [, *relationship-defn*]* ;}

entity-defn:

`define entity` *entity-name* {*interface*; [*internal-ents*; *internal-rels*; *internal-cors*;] }

interface:

`interface:` *aspect-defn* [, *aspect-defn*]*

aspect-defn:

`single-valued` *domain* :: *aspect-name*

`multi-valued` *domain* :: *aspect-name*

internal-ents:

`internal entities:` *entity-name* [, *entity-name*]* ;

internal-rels:

`internal relationships:` *relationship-defn* [, *relationship-defn*]* ;

internal-cors:

`internal correspondences:` *correspondence-defn* [, *correspondence-defn*]* ;

relationship-defn:

[*entity-name*.]*aspect-name relationship-name* [*entity-name*.]*aspect-name*

correspondence-defn:

aspect-name `corresponds to` [*entity-name*.]*aspect-name*

Figure 8: Textual syntax

```
define entity METHOD{
interface:
        single-valued string:: Declaration;
}
define entity CLASS{
interface:
        single-valued string:: Name,
        multi-valued OID:: Method;
internal entities:
        METHOD;
internal relationships:
        METHOD method_of CLASS;
internal correspondences
        METHOD corresponds to Method;
}
define entity DEFINITION{
interface:
        single-valued string:: Body;
}
define relationships{
        CLASS.Method defined_as DEFINITION;
}
```

Figure 9: Textual representation for example

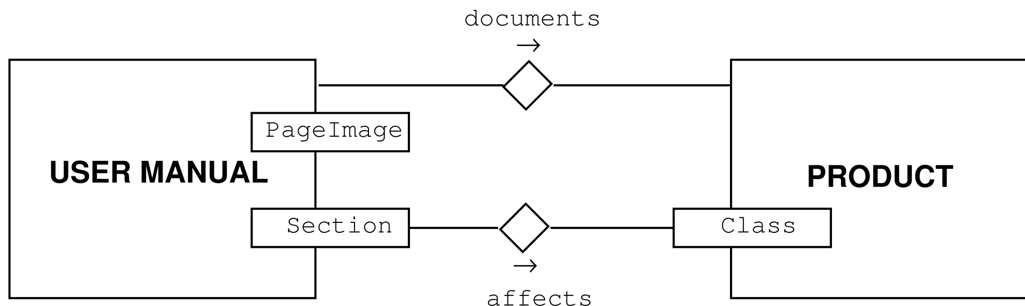

Figure 10: Software Deliverables

5 LER and the Real World

A database designer repeatedly encounters certain patterns while modeling the data of an enterprise. In this section, we indicate how these may be modeled using the LER approach. We start out by extending the previous example to display some of these patterns. This is followed by a discussion of how the enterprise data may be mapped to concepts in LER.

5.1 Detailed Example: A Software Development Enterprise

Consider how a software development enterprise may organize the data relating to its project deliverables.

The firm has its personnel working on two distinct concerns for each project (Figure 10). Some personnel develop the software `PRODUCT`, while others write the documents which make up the `USER MANUAL`. While a product and its manuals are related to each other (both at the top-level and internally), the internal organization of one should have a minimal influence on the organization of the other. To achieve this we organize the deliverables as follows. At the coarsest level of granularity, a `USER MANUAL documents`

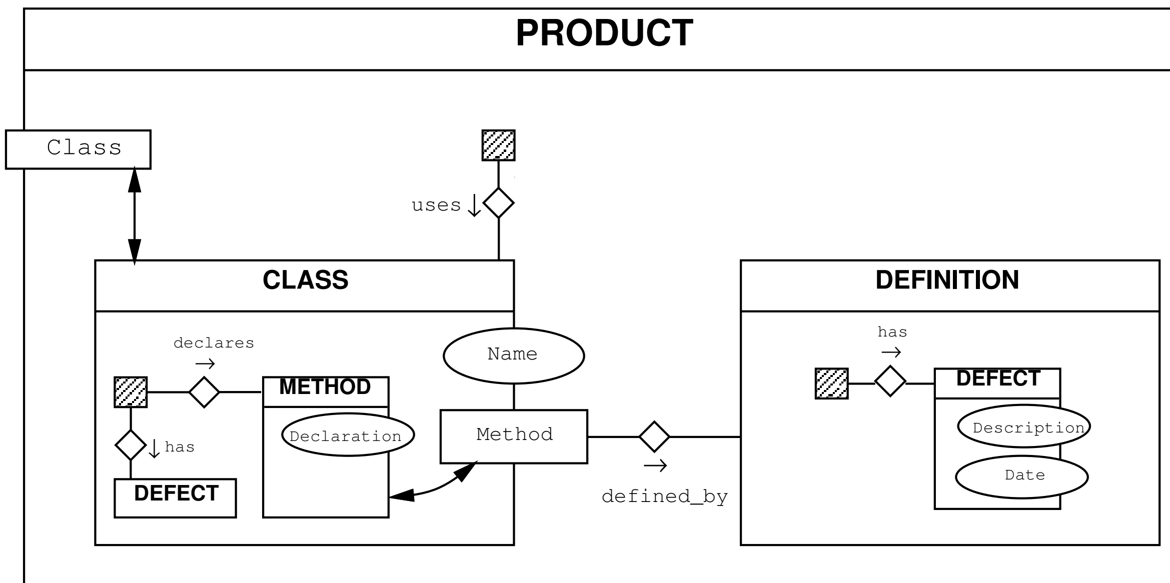

Figure 11: Glass-box view: PRODUCT

a PRODUCT. From the outside, we know that the USER MANUAL has aspects Sections and PageImages. The correlation between these is reflected in the internal structure of the manual. An important aspect of a PRODUCT is the set of Classes which are used to build the product. A class in the product affects manual sections in the sense that altering the class requires revision of those sections. Notice that we are able to relate elements at a fine granularity using a higher level of abstraction.

Let us now examine the glass-box views of the PRODUCT and the USER MANUAL. First the PRODUCT (Figure 11). As suggested by the Class aspect in the black-box view of PRODUCT, the products in the firm use an object-oriented language (say, C++). Thus, a PRODUCT is made up of many interrelated CLASSes. (For simplicity, we do not show any relationships be-

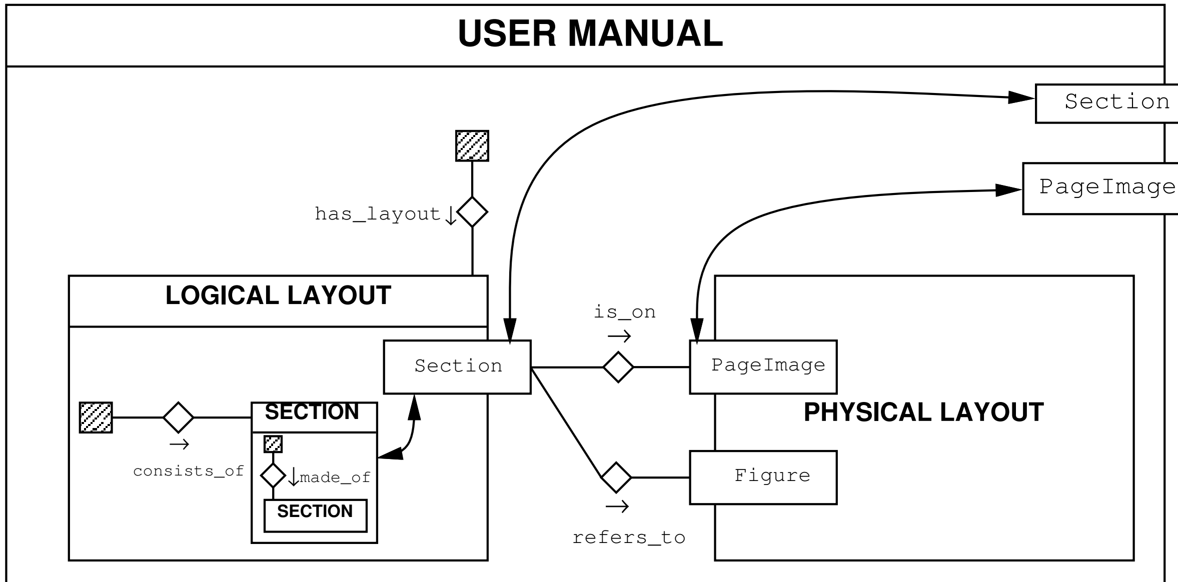

Figure 12: Glass-box view: USER MANUAL

tween the classes.) A CLASS has a Name and declares a set of Methods. For each METHOD, the body of the method may be found in its DEFINITION. The schema also allows DEFECTs in the product to be tracked. DEFECTs may be associated with either a CLASS or a DEFINITION of its method. For example, a method declaration may not have followed the organizational standards. Such a DEFECT is associated with a CLASS. Or, an end condition of a loop may be causing a problem. In this case, the DEFECT is associated with the DEFINITION of the method. The use of DEFECT illustrates the non-locality achievable through a global name space.

The user manual has two manifestations – its LOGICAL LAYOUT and its PHYSICAL LAYOUT (Figure 12). The LOGICAL LAYOUT consists of SECTIONs in the user manual. Each section in turn consists of other (sub)Sections. The

PHYSICAL LAYOUT is the output of processing by some typesetting program. At its interface is the set of Figures and Page Images. These are ordered but we do not show the ordering since it could be done by simply associating each page with its page number. Corresponding to each Section is the set of PageImages on which the section occurs and the set of Figures referred to in the text of that section.

Note that we have only shown the black-box view of the PHYSICAL LAYOUT. By specifying only the black-box view we can isolate the internal structural details which is dependent on the typesetter used. When the typesetter is finalized one can specify the internal structure. In fact, the isolation also allows the internal structure to be specified by a more appropriate formalism – say a grammar.

5.2 Mapping the concepts

A modeling enterprise focuses on representing objects or concepts in the world, their properties and their associations. In an LER model, an entity is used to represent concepts and objects. Properties are represented using aspects and – when they seem to have an importance or structure of their own – using entities. If two entities are associated with one another, an LER relationship may be used to represent the association. More complex associations are represented using entities.

5.2.1 Mapping to entities and sub-entities

It is not always clear whether a concept is an entity in its own right, or just a property of another entity. A database designer could use the following rule of thumb to decide. If the concept has an importance *independent* of other

concepts or a *structure* of its own, then it is more than a property – it is an entity. The entity may then be reflected as a property of the containing entity using the correspondence mechanism.

Let's turn our attention to entities within another entity. Picture a database schema represented using the (flat) ER formalism. Some entities in the schema stand out in importance relative to other entities. Such entities may be considered to be the "central" entities, each of which is associated with many "satellite" entities. The satellites are candidates for consideration as subentities of the center. However, not all satellites are potential subentities. The internalization decision should be based on the nature of the relationship between the center and a satellite.

- The satellite may be in a is-part-of relationship with the center. Furthermore, the satellite may be existentially dependent on the center. In other words, the satellite seizes to exist when the center seizes to exist. For example, the parts of a car existentially depend on the car itself. The is-part-of relationship may also involve a satellite whose instances are shared. For example, consider the classes in a product. A class may be used to build many products. In such cases, while the deletion of a product does not imply the deletion of its classes, the strength of the relationship justifies internalizing class into a product.

- The satellite may not be a sub-part of the center, yet the satellite may be semantically dependent on the center. That is, the satellite may derive all its semantic value due to its association with the center. Of course, the satellite also adds meaning to the central entity. For example, a class defect may be regarded as a subentity of a class,

because the defect is associated only with the class. Also, the defect is existentially dependent on the class.

Consider an example where there is no existential dependency. The logical layout and physical layout are meaningful only in the context of a central entity, the user manual. These layouts are different ways of viewing a user manual, and hence it makes sense to subsume them within the user manual.

- The satellites and the center may not be dependent on each other in any other way other than that they belong to the same functional area [TWBK89]. [TWBK89] also provides examples for clustering using this criterion. One may internalize the sub-entities in this case to reduce entity clutter.

5.2.2 Mapping to relationships

The LER relationship models binary relationship between entities. Such a relationship may be used to relate internals of entities. The previous example (sec 5.1) illustrates such a relationship, hence we do not go into it in any more detail.

We have not included structural (cardinality, membership, ...) constraints in our formalism because we are developing a more comprehensive notion of constraints. However, we do not foresee any special problem in specifying constraints using any of the commonly used techniques [Che76, RBB$^+$85, EN89].

The need for relationships of degree higher than three or for relationships to have associated properties is one of user communication. Such relation-

ships are best handled in theory and implementation through the creation of an entity representing that relationship. But since the communication role of a model is important, a user interface based on LER should have a mechanism which permits modeling higher degree relationships.

5.2.3 Mapping to aspects

Properties of entities which would be considered attributes in the ER approach would be modeled as aspects in LER. Therefore, multi-valued attributes and single-valued attributes could be viewed as multi-valued and single-valued aspects, respectively.

The ER approach has a clear separation of properties and entity. We blur the distinction in the belief that it is more realistic. For example, the address of a person is sometimes modeled as an attribute and at other times as an entity associated with person. LER reconciles the two viewpoints. The black-box view of a person may display an address as an aspect. However, the glass-box view may reveal that the address aspect is in correspondence with a sub-entity of person.

The correspondence mechanism not only exports internal entities to the surface, but may also be used to export properties of internal entities. Continuing our example, the address of a person may be the actual value of the address or an entity which represents the address.

One may also use correspondence to detail a finer structure for what may only be a property of a higher level entity. For example, the classes in a product have a structure that is detailed when we zoom in to the corresponding entity.

6 Related Work and Comparisons

The *Clustered Entity Model* [FM86] is the one of the early attempts [2] at layering ER diagrams. In this approach, an ER diagram at a lower level appears as an entity on the next level. The *Nested Entity-Relationship Model* [CJA90] improves on the above by allowing both entities and relationships to have a deeper structure.

Harel [Har88] uses *higraphs* to layer ER diagrams. A higraph combines topological properties of venn diagrams with edge specifications to specify relationships between sets. Its topological nature, however, interprets leveling as a subset relationship between an entity and its subentity. Thus, higraph based layering is more suited for clustering the specializations of an entity, rather than a general collection of functionally related entities.

[TWBK89] proposes an ER clustering technique guided by grades of *cohesion*. The most cohesive structure being that of strong entities surrounded by weak entities. These are followed by entities related by generalization link, entities related by a constraint relationships, entities with unconstrained relationships between them, and finally, the least cohesive, entities within the same functional area. Since the grade of cohesion is precisely definable, an automated tool can produce a bottom-up clustering of an ER diagram [HZ90]. In contrast, LER is a top-down approach in which clustering is completely specified by a designer. Informally, though, our approach suggests the highest importance to grouping based on the enterprises functional area, followed by grouping based on criterion similar to

[2] We do not intend this to be a comprehensive survey of the work in this area. We consider works which inspired this work and resonate most closely with it.

[TWBK89].

6.1 Non-encroaching relationships

In each of the above, relating an entity E to a subentity S' in entity E' either requires relationships to go across the boundary of E', or E is replicated (and marked as such) inside E' and then related to S'. In either case, one must *encroach* the boundaries of E' to model the relationship. In our approach, if S' needs to be related to E, then it is related via an aspect. Thus, there is a controlled mechanism for modeling relationships between E and S' without encroaching E'.

The difference may also be explained in programming language terms. If we make the analogy that an entity is a procedure, then the above models have only global variables and parameterless procedures. An understanding of the procedure requires knowledge of internals of the procedure. Of course, such a notion of parameterless procedures would be considered ineffective as an abstraction mechanism. Our approach may be considered as adding the notion of a parameter to a procedure. In effect, we hope that the aspect mechanism will permit a cleaner notion of abstraction which helps understanding by localizing the information structure.

6.2 Consistent refinements

Aspects also permit a stricter notion of refinement of the LER diagrams than are permitted by other models. In a data flow diagram[GS79], for example, process decomposition maintains input and output data flows. That is, if two processes, A and B, have a data flow between them, the decomposition of A shows up no additional data flows between B and any subprocess of A.

Thus, the decomposition is consistent with the abstraction.

One may define a similar notion for diagrams which use the ER approach. A refinement of an entity is *consistent* with its abstraction only if the refinement induces no change in the rest of the ER diagram. This makes successive refinement a process which does not cascade changes throughout the enterprise. The modeling activity of an enterprise may begin by a high level design done at suitably high level in the enterprise. Successive refinements refine different portions of the high level design, each of which is constrained to satisfy the higher level design, but is otherwise free to add, change, or delete elements at the lower levels. Note that this also means that the data design has a greater chance of reflecting the design of the enterprise.

6.3 Well-defined refinement

The exact nature of the association between an entity at a higher level and a sub-entity at a lower level remains poorly defined. Thus, such models are used only as communication tools – the database corresponds to only the lowermost layer and can not take advantage of the layering information in the design. The poorly defined correspondence also leads to a mistaken notion of layering as nothing more than aggregation of [SS77] and EER of [EN89].

An LER entity at a higher level is necessarily present at a level below as self. This specifies the relationship between the higher level entity in its elemental form as the self and the cluster in the lower layers. Moreover, the database reflects this relationship.

Surprisingly, notions of well-defined correspondence, consistent refine-

ment, and non-encroaching relationships, though intuitively desirable are not found in the above models. We believe this to be a fundamental contribution of this paper. [LV89], however, does hint at a notion similar to aspects but there is no notion of self. In [LV89] an ER diagram at a lower level *exports* some of its entities so that they may be used as roles in relationships in a diagram at a higher level.

7 Conclusion

A data model consists of three components – object types, operators and integrity rules [Dat83]. In this paper, we have introduced aspects, entities, and relationships as the basic object types in LER. We also placed general integrity constraints on LER schema and instances. We are working towards defining a transaction language for operating on LER databases and incorporating generalization into the LER model [SS77, EWH85].

We introduced three notions which we feel make a layering formalism useful for enterprise modeling: a well-defined correspondence between an entity at a higher level and the entity cluster at a lower level, non-encroaching relationships between entities, and consistency between the refinement and abstraction steps. The primary contribution of this paper is demonstrating a layering formalism which has the above properties.

References

[Che76] P. P. Chen. The Entity-Relationship Model - Toward a unified view of data. *ACM Transactions in Database Systems*, 1(1):9–36, March 1976.

[CJA90] C. R. Carlson, W. Ji, and A. K. Arora. The Nested Entity-Relationship Model. In F.H. Lochovsky, editor, *Entity-Relationship Approach to Database Design and Querying*, pages 221–236, North-Holland, 1990. Elsevier Science Publishers B. V.

[Cod79] E. F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 4(4), December 1979.

[Dat83] C. J. Date. *An Introduction to Database Systems*, volume 2 of *The Systems Programming Series*. Addison-Wesley, Reading, Massachusetts, July 1983.

[EN89] R. Elmasri and S. B. Navathe. *Fundamentals of database systems*. Benjamin/Cummings Publishing Company, Inc., Red-Wood City, California, 1989.

[EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The Category Concept: An Extension to the Entity-Relationship Model. *International Journal on Data and Knowledge Engineering*, 1(1), May 1985.

[FM86] P. Feldman and D. Miller. Entity Model Clustering: Structuring a Data Model by Abstraction. *Computer Journal*, 29(4):348–360, August 1986.

[GS79] C. Gane and T. Sarson. *Structured System Analysis*. Prentice-Hall, 1979.

[Har88] D. Harel. On Visual Formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.

[HZ90] S. Huffman and R. V. Zoeller. A Rule-Based System Tool for Automated ER Model Clustering. In F.H. Lochovsky, editor, *Entity-Relationship Approach to Database Design and Querying*, pages 221–236, North-Holland, 1990. Elsevier Science Publishers B. V.

[LV89] M. Lacroix and M. Vanhoedenaghe. Tool Integration in an Open Environment. In C. Ghezzi and J.A. McDermid, editors, *2nd European Software Engineering Conference, Proceedings*, pages 311–324, Berlin, Hiedelberg, September 1989. Springer-Verlag.

[RBB+85] D. Reiner, M. Brodie, G. Brown, M. Friedell, D. Kramlich, J. Lehman, and A. Rosenthal. The Database Design and Evaluation Workbench (DDEW) Project at CCA. *Database Engineering*, 7(4):10–15, 1985.

[SS77] J. Smith and D. Smith. Database Abstractions: Aggregation and Generalization. *TODS*, 2(2), June 1977.

[TWBK89] T. J. Teorey, G. Wei, D. L. Bolton, and J. A. Koenig. ER Model Clustering as an Aid for User Communication and Documentation in Database Design. *Communications of the ACM*, 32(8):975–987, August 1989.