ELSEVIER

Data & Knowledge Engineering 20 (1996) 39–85

DATA & KNOWLEDGE ENGINEERING

Conceptual schemas with abstractions Making flat conceptual schemas more comprehensible

L.J. Campbell[a,*], T.A. Halpin[b], H.A. Proper[c]

[a] *Department of Computer Science, University of Queensland 4072, Australia*
[b] *Asymetrix Corporation, Bellevue WA, USA*
[c] *Cooperative Information Systems Research Centre, Faculty of Information Technology, Queensland University of Technology, GPO Box 2434, Brisbane, 4001, Australia*

Received 30 August 1995; revised 7 November 1995; accepted 5 January 1996

Abstract

Flat graphical, conceptual modeling techniques are widely accepted as visually effective ways in which to specify and communicate the conceptual data requirements of an information system. Conceptual schema diagrams provide modelers with a picture of the salient structures underlying the modeled universe of discourse, in a form that can readily be understood by and communicated to users, programmers and managers. When complexity and size of applications increase, however, the success of these techniques in terms of comprehensibility and communicability deteriorates rapidly.

This paper proposes a method to offset this deterioration, by adding abstraction layers to flat conceptual schemas. We present an algorithm to recursively derive higher levels of abstraction from a given (flat) conceptual schema. The driving force of this algorithm is a hierarchy of conceptual importance among the elements of the universe of discourse.

Keywords: Conceptual data modelling; Schema abstraction; ORM; ER; NIAM

1. Introduction

Conceptual schemas play an important, and recognized role in the development life cycle of an information system [28]. They serve both as a means by which the salient structures of the underlying universe of discourse (UoD) can be captured, and as a communication tool among the designers, programmers, users and managers [32]. Conceptual schema modeling techniques, such as Entity Relationship (ER) modeling [1, 12] and Object Role Modeling (ORM) [17] are widely acknowledged as being visually effective ways in which to specify and communicate the conceptual data requirements of an information system.

* Corresponding author. Email: linda@cs.uq.oz.au

SSDI: 0169-023X(96)00005-5

However, as database application requirements increase in size and complexity, the comprehensibility and maintainability of the specification degrades rapidly [28]. Simsion identified the problem of representing large data models as 'one of the most serious limitations of data modeling in practice' [30]. It is claimed in Feldman and Miller [13] that the 'usefulness of any diagram is inversely proportional to the size of the model depicted'. This problem, which has been referred to as the *Database Comprehension Problem* [8], is shared by all *flat* data models. In the specification of a *flat* conceptual schema, each object type is viewed at only one level of abstraction in a single diagram and all object types are considered to be of equal importance within the application [7]. While this is satisfactory for small, academic examples, when conceptual schemas of a moderate to large size are involved, this feature reduces the rate of comprehension of the application.

The Database Comprehension Problem in flat data models has motivated several authors to try to find successful methods by which to form abstractions on a given flat conceptual schema [8, 10, 13–15, 26, 27–29, 31–33]. In 1983, Vermeir [32] described a number of abstraction techniques including *viewpoint relative abstraction*, which displays the portion of the schema within a particular distance of a focal object type, and the *absolute abstraction hierarchy*, which iteratively removes non-key concepts from successive layers of abstraction.

An abstraction technique which is quite popular in the literature is *Entity-Relationship Model Clustering*. An *ER Model Cluster Diagram* is 'a hierarchy of successively more detailed Entity Relationship diagrams, with a lower-level diagram appearing as a single entity type on the next higher level diagram' [13]. Martin developed an ER clustering procedure based on 1:m relationship between entities, which is simple, but rather arbitrary and judgmental [27]. ER model clustering was then applied to the Whitbread Corporate Data Architecture in June 1983 in order to test the theory on a significant practical application. As an outcome, in 1986, Feldman and Miller proposed a semi-algorithmic approach to ER model clustering – one which still relies heavily on human direction and judgment.

In 1989, Carlson and Ji [8] proposed the Nested Entity Relationship (NER) model as an extension to the multi-level ER clustering techniques of Feldman and Miller. NER supports traditional abstraction techniques such as aggregation, generalization and association, as well as allowing ER diagrams at one level to be abstracted into either complex entities or complex relationships at the next higher level. In that same year, Teorey et al. [31] introduced a set of ER model clustering rules. In this publication, entities were grouped recursively, based on a list of grouping operations prioritized according to the *cohesion* (or internal strength) among the entities involved. Once again, the algorithm proposed is largely based on arbitrary human judgment. This work was taken further by Huffman and Zoeller in 1989, who confirmed the feasibility of using a rule-based system to automate the ER clustering process of Teorey et al. [26].

A number of other abstraction techniques have been introduced in the years since. In 1991, Czejdo and Embley proposed the management of large complex data models using views and a number of functions to manipulate those views [10]. Moody [28] proposed a representation scheme for abstraction based on the organization of a street directory, using various levels of detail and intermap references and overlap between scopes. A new abstraction mechanism for typed graphs, called *Caves*, which allows the designer to selectively 'amplify or diminish' parts of the conceptual model, was presented by Walko [33] in 1992. The techniques available

through Caves include: *Filtration*, which removes extraneous details and constraints; *Perspective* which presents only the local vicinity of a selected focal point; and *Comprehension*, which folds the schema into a smaller version.

The common element of many of the abstraction techniques throughout the literature is the selection of a set of important elements within a conceptual schema. Many different names have been given to these objects which are considered to be of importance within the application domain, including *key concepts* [32], *major entity types* [13], *maximal objects* [8] and *dominant objects* [26, 31]. In this paper (as in [5–7]) we call these the *major object types*. In order to briefly highlight the differences between these similar concepts throughout the literature, we will refer to a small example conceptual schema shown in Fig. 1 ([17, p. 402]) (This diagram uses Object Role Modeling notation which is explained in Section 2).

The abstraction techniques described by Vermeir in [32] are based on the notion of a *key concept*. Key concepts are those objects within a conceptual schema which are considered to be of higher semantic importance because they keep the graph connected. In the example in Fig. 1, therefore, the object types 'Movie', 'Person', MoneyAmt' and 'Country' would be considered, in [32], to be the key concepts. It is quite apparent to a human, however, that 'MoneyAmt' is not one of the most semantically important object types in the example Universe of Discourse. In fact, Vermeir himself observes that the definition of 'key concept' is too simplistic, and many cases arise in which the abstractions produced are not intuitive [32].

Feldman and Miller consider the most important entity types to be those that appear in more than one branch at any particular level of their clustering hierarchy and call these objects *major entity types* [13]. Using Feldman and Miller's algorithm [13], every object type in the schema (Movie, Title, Person, MoneyAmt, Country, Date) would be classified as a candidate

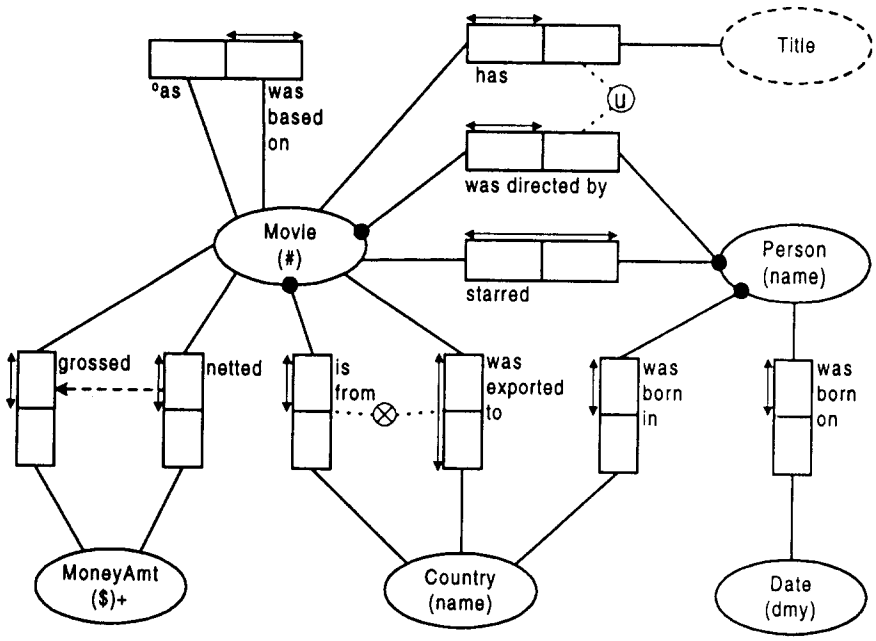


Fig. 1. Example conceptual schema.

to be a *major entity type*, after which human intuition is required to narrow down the choice. Nested Entity Relationship models give a stronger theoretical foundation than previous multi-level ER approaches, though the concept of *maximal objects* [8]. However, no attempt is made to automate the selection of *maximal objects* in a conceptual schema. Teorey et al. [31] and Huffman and Zoeller [26] base their ER model clustering rules around what they call the *dominant objects*. The only *dominant object* that results from the automatic algorithm presented in [26] is 'Movie'. Identifying other dominant objects to, once again, necessitate the use of human judgment. While Huffman and Zoeller's results were loosely comparable to those produced using human intuition, it was acknowledged by the authors that some aspects of the clustering algorithm were too simplistic for complex cases. Referring back to the conceptual schema in Fig. 1, it is most likely that a human would intuitively decide that the *major object types* in this Universe of Discourse are 'Movie' and 'Person'. None of the algorithms reviewed in the literature arrive at this result automatically.

The first goal of this paper, therefore, is to formalize a method for the strictly automatic selection of major object types. What sets our approach apart from others is that our approach considers the detailed conceptual semantics hidden in the constraints and also the manner in which the facts within the domain are verbalized. In particular, our approach utilizes the detailed constraint specifications and verbalizations provided by Object Role Modeling. It is believed that a lot of the human intuition (conceptual semantics) is contained in these constraints and verbalizations. We, therefore, claim that our approach more accurately imitates human intuition than previous methods. As a second goal, this paper also aims to utilize these selected major object types in an algorithm to derive abstractions for a flat conceptual schema.

In Section 2, we begin by introducing a formal description of Object Role Modeling, which will be used as the foundation of the algorithms presented. Section 3 extends the semantics of Object Role Modeling by introducing the notion of conceptual *anchors*, which are required for the detection of major object types. An automated method for selecting anchors is presented. The selection of anchors is based on the semantics of constraints defined on surrounding relationship types. The semantics of these constraints, in terms of populations, allows us to make this selection. The notion of major object types and abstraction levels is then introduced in Section 4, together with a method for automatically determining them. Section 5 illustrates how this automated abstraction process is performed on a small case study; and conclusions are reached in Section 6.

2. Object Role Modeling

Object Role Modeling (ORM) views the world as a collection of objects which play roles and, unlike Entity-Relationship Modeling, makes no initial use of the *attribute* construct. Every elementary type of fact which occurs between an object type in the Universe of Discourse (UoD) is verbalized and displayed on a conceptual schema diagram. Object Role Modeling also allows a wide variety of data constraints to be specified on the conceptual schema, including mandatory role, uniqueness, exclusion, equality, subset and occurrence frequency.

The high level of detail displayed on an ORM diagram allows Object Role Modeling to offer a correspondingly high level of expressiveness. Unfortunately, this high level of detail also tends to promote the degradation in comprehensibility and communicability in large conceptual schemas. An ER diagram, through of its use of attributes, can already be thought of as an abstraction (or summary) of a corresponding ORM diagram. In this way, traditional Entity Relationship modeling can postpone the immediate effects of the *Database Comprehension Problem* until a larger Universe of Discourse is required. It is not uncommon in practice, however, for abstractions (or summaries) of ER diagrams themselves to be required. While the scale of the problem, therefore, differs slightly between ORM and ER, the *Database Comprehension Problem*, nonetheless, is universally shared by all flat modeling techniques.

For the purposes of this paper, we plan to consider the more detailed of the two most common data modeling techniques (Object Role Modeling) and introduce a method to control the schema's visual complexity during the information system development. As argued before in [7, 5, 17], an Entity Relationship model can be considered comparable with the first of the abstraction levels on an ORM model.

The following subsections outline a formalization of some fundamental ORM structures and constraints which will be required in Sections 3 and 4 to describe our abstraction methods. The formalization of ORM as presented in this article inherits a rich and well published history, full of constant refinements and additions. The evolution of this particular ORM formalization started out from the PM/PSM version of ORM [2, 20, 23]. More 'modernized' versions of ORM formalizations can be found in [20, 3] and the most recent developments are discussed in [9]. Alternative formalizations can be found in [11, 16].

While formalizations of ORM have been published before, this paper needs to describe the formalization again in order to be self-contained. In this formalization, we limit ourselves to syntactical issues only. Issues regarding the associated semantics can be found in the referenced publications. Furthermore, the formalization presented in this paper is based on a limited number of basic concepts to provide us with only what is needed for the purposes of abstraction. For a detailed description of the methodology associated with Object Role Modeling, refer to [17].

2.1. Information structure

The cornerstone of a conceptual schema is formed by the so-called 'Information Structure'. This structure is concerned with the object types and their interrelationships in the modeled Universe of Discourse. The information structure of a conceptual schema is described in the following subsection. In doing so, we assume that the reader has some basic working knowledge of the concepts underlying ORM or ER.

2.1.1. Flat conceptual modeling

In [9] an ORM version is proposed which extends ORM with both top-down abstraction mechanisms as well as aspects from object oriented conceptual modeling techniques. The relation between that article and this article is that here we are concerned with an algorithm to 'reverse engineer' the abstraction layers from an existing flat conceptual model, whereas [9] provides the extensions to ORM needed to add abstractions in a top-down way, which is

necessarily a manual process. The output of the algorithm presented in this article can indeed be seen as a 3-dimensional ORM model fitting the top-down abstraction framework.

As a warning to readers of [9], it should be noted that what we call a flat conceptual schema in this article is in fact not a primitive, flat conceptual schema, as described in [9]. In this article our starting point is a schema consisting of object types, relationship types and objectified relationship types. Objectified relationship types are also referred to as *nested object types*, and as its second name suggests, it already introduces depth into a conceptual schema. It is well known that objectifications can be replaced by so-called co-referenced object types. As an example of this, consider Fig. 2(a). This schema fragment is equivalent to the fragment depicted in Fig. 2(b). Depending on the universe of discourse that is being modeled, it may be more natural to use either one of the co-referenced or objectified representations. In the abstraction algorithms, the choice between a co-referenced object type and an objectified relationship type is honored by treating them slightly differently.

2.1.2. Typing scheme

An ORM conceptual schema, $\mathscr{CS}$, is presumed to consist of a set of conceptual types, $\mathscr{TP}$. These types are divided into three main subclasses. The first class is the set of object types, $\mathscr{OB}$. Within this class a subclass of value types, $\mathscr{VL}$, can be distinguished. Instances from value types usually originate from some underlying domain such as strings, natural numbers, audio, video, etc. A separate class of types, the relationship types $\mathscr{RL}$, contain those types used to describe the relationship between one or more object types. Those object types which are not value types are called non-value types: $\mathscr{NV} \triangleq \mathscr{OB} - \mathscr{VL}$. In the formalization used in this paper, we allow types to belong to both the set of object types, and the set of relationship types. We refer to these later types as *nested object types* or *objectified relationship types*. Relationship types which are objectified do not belong to the set of value types. That is:

$$\mathscr{VL} \cap \mathscr{RL} = \varnothing$$

These types also have a number of structural properties which we now consider.

2.1.3. Roles in relationship types

Each relationship type in $\mathscr{RL}$ contains a collection of roles. We refer to the set of all roles, in an RM conceptual schema, as $\mathscr{RO}$. The roles in $\mathscr{RO}$ are distributed among the relationship types by the function Roles: $\mathscr{RL} \rightarrow \wp^+(\mathscr{RO})$, which should provide a partition of the set of roles. (Note that $\wp^+(\mathscr{RO})$ yields all non-empty subsets of $\mathscr{RO}$.) Each role has exactly one

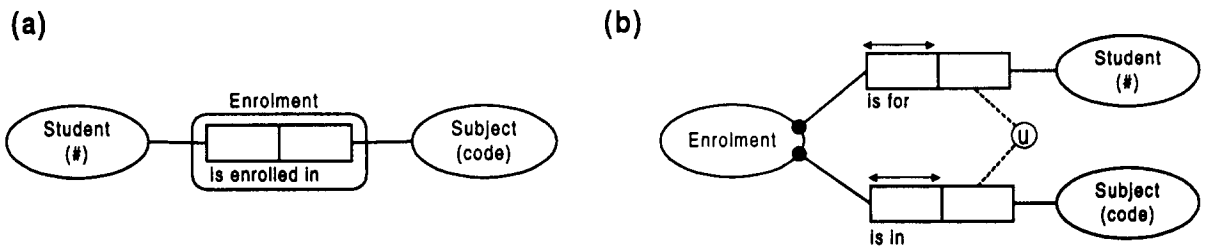


Fig. 2. (a) Nested object type; (b) co-referenced object type.

object type participating in it. This object type can be obtained by applying the function $\mathsf{Player}: \mathscr{RO} \rightarrow \mathscr{OB}$ to the relevant role. If the participants of a set of role is required, we use a generalization of this function, $\mathsf{Players}: \wp(\mathscr{RO}) \rightarrow \wp(\mathscr{RO})$, which is defined as:

$$\mathsf{Players}(v) \triangleq \{\mathsf{Player}(p) \mid p \in v\}$$

To determine the relationship type to which a given role belongs, the inverse of the Roles function ($\mathsf{Rel}: \mathscr{RO} \rightarrow \mathscr{RL}$) is used:

$$\mathsf{Rel}(s) \triangleq p \text{ such that } s \in \mathsf{Roles}(p)$$

In this formalization, the collection of roles contained in a relationship type is considered to have a predefined default order. This order, embedded in the verbalization of the relationship types, is provided by the domain experts during the initial analysis phase. As such, the PosN function is one of the knowledge sources from which we will try to mine the conceptual semantics hidden in the schema. The function $\mathsf{PosN}: \mathscr{RO} \rightarrow \mathbb{N}^+$ is used to assign a position to each role.

The predicate $\mathsf{Rels}: \wp(\mathscr{RO}) \rightarrow \wp(\mathscr{RL})$ is a generalization of the Rel function to sets of roles. It returns all relationship types involved in a given set of roles:

$$\mathsf{Rels}(v) \triangleq \{\mathsf{Rel}(p) \mid p \in v\}$$

Similarly, the Roles function can be extended to accept a set of relationship types and return all roles involved in any of the given relationship types ($\mathsf{Roles}: \wp(\mathscr{RL}) \rightarrow \wp(\mathscr{RO})$):

$$\mathsf{Roles}(v) \triangleq \{\mathsf{Roles}(p) \mid p \in v\}$$

To conveniently access the roles involved in the same relationship type as a given role, we define the function $\mathsf{CoRoles}: \mathscr{RO} \rightarrow \wp(\mathscr{RO})$, as:

$$\mathsf{CoRoles}(v) \triangleq \mathsf{Roles}(\mathsf{Rel}(v))$$

Similarly to Roles and Rel, CoRoles can be extended to perform the same operation on a set of roles:

$$\mathsf{CoRoles}(v) \triangleq \mathsf{Roles}(\mathsf{Rels}(v))$$

If we require those roles in the same relationship type as a given role, excluding the given role, we use the function $\mathsf{OtherRoles}: \mathscr{RO} \rightarrow \wp(\mathscr{RO})$, which is defined as:

$$\mathsf{OtherRoles}(p) \triangleq \mathsf{CoRoles}(p) - \{p\}$$

2.1.4. Subtyping

The specialization relationship between a subtype and a supertype is captured by the relationship $\mathsf{SubOf} \subseteq \mathscr{OB} \times \mathscr{OB}$. The intuition is that when x SubOf y, the population of x is a definable subset of the population of y. Each subtype hierarchy (defined by SubOf) corresponds to a directed acyclic graph which adheres to the laws of transitivity and irreflexivity: The relation $\mathsf{Top}(x, y)$ is defined such that y is a top of x in the associated

subtype hierarchy. The hierarchies we consider must always have one single top; so we can write $\mathsf{Top}(x) = y$ to refer to that unique top.

Given a set of object types in a subtype hierarchy, we can try to find the common supertypes in this hierarchy that are closest to these object types. To this end, we first need to find all common supertypes. This is done using the function $\mathsf{CommonSup}: \wp(\mathcal{OB}) \rightarrow \wp(\mathcal{OB})$, which is defined as:

$$\mathsf{CommonSup}(w) \triangleq \{x \in \mathcal{OB} \mid \forall_{y \in w}[y\ \mathsf{SubOf}\ x]\}$$

The next step is to select those common supertypes that are closest to the given set of object types. We therefore introduce the notion of a lowest common supertype. The lowest common supertypes are those common supertypes which do not have any other common supertype as a subtype. A set of object types can actually have more than one lowest common supertype. The function $\mathsf{LowestCSup}: \wp(\mathcal{OB}) \rightarrow \wp(\mathcal{OB})$ is defined by:

$$\mathsf{LowestCSup}(w) \triangleq \{w \in \mathsf{CommonSup}(w) \mid \neg\exists_{y \in \mathsf{CommonSup}(w)}[y\ \mathsf{SubOf}\ x]\}$$

Given an object type x in a subtype hierarchy, we can determine the set of subtypes of this object type. This is done using the function $\mathsf{SubHierarchy}: \mathcal{OB} \rightarrow \wp(\mathcal{OB})$, which is defined as:

$$\mathsf{SubHierarchy}(x) \triangleq \{y \mid y\ \mathsf{SubOf}\ x\}$$

Consider, for example, the subtype hierarchy defined in Fig. 3. The supertypes of 'Bicycle' are 'Car or Bicycle' and 'Vehicle'. The supertypes of 'Car' are 'Car or Bicycle', 'Motorized Vehicle' and 'Vehicle'. The common supertypes (CommonSup) of 'Car' and 'Bicycle' are 'Car or Bicycle' and 'Vehicle' and the lowest common supertype (LowestCSup) of 'Car' and 'Bicycle' is 'Car or Bicycle'.

2.1.5. Type relatedness

Intuitively, object types may, for several reasons, have values in common in some populations. Two types are considered type related if their populations may share instances. Type relatedness, which we denote by $x \sim y$, is a property held only by object types which are in the same subtype hierarchy. For more detailed rules on type relatedness, refer to [18, 23].

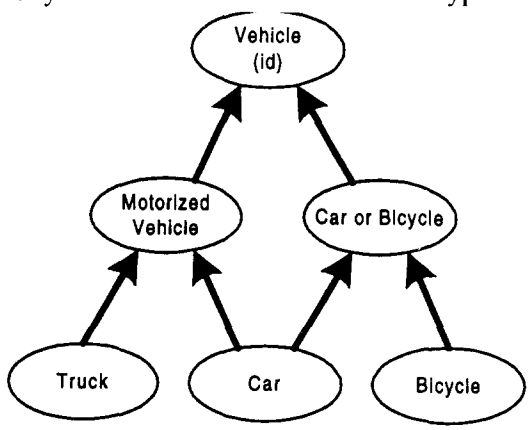


Fig. 3. Example of a specialization hierarchy.

Two roles are type related if their players are type related; so if p, $q \in \mathcal{RO}$, then:

$$p \sim q \stackrel{\Delta}{=} \mathsf{Player}(p) \sim \mathsf{Player}(q)$$

2.1.6. Complete information structure

We can now define the basic information structure $\mathcal{IS}$ of a conceptual schema $\mathcal{CS}$ in terms of the following components:

$$\mathcal{IS} = \langle \mathcal{OB}, \mathcal{VL}, \mathcal{RL}, \mathcal{RO}, \mathsf{Roles}, \mathsf{PosN}, \mathsf{SubOf}, \mathsf{Player} \rangle$$

2.2. Conceptual schema

Besides the information structure, a conceptual schema consists of constraints and derivation rules. For this article, only a limited class of constraints is of interest. The constraint classes we discuss, together with PosN, will be used as a source of information to decide which object types are major.

2.2.1. Mandatory constraint

To specify the requirement that instances of a particular object type must always participate in at least one of some set of roles, we use the mandatory constraint, $\mathsf{Mand} \subseteq \wp(\mathcal{RO})$ (also referred to as 'total role constraint' in [23]). A mandatory constraint specifies that the union of the populations of the constrained set of roles must equal the total population of their player(s). All roles contained in a mandatory constraint must be type related. Therefore, we should have:

$$\mathsf{Mand}(v) \Rightarrow \forall_{p,q \in v}[p \sim q]$$

A basic rule for ORM models (as defined in [17]) states that every instance of an object type must participate in at least one (fact type) role. In Subsection 2.2.6 we will see that the only exception to this rule are the so-called lazy object types. This results in a mandatory role being implied over each set of type related roles. We identify the mandatory constraints which can be inferred in this way, $\mathsf{InferMand} \subseteq \wp(\mathcal{RO})$, with the following derivation rule:

$$\mathsf{InferMand}(v) \stackrel{\Delta}{=} \forall_{p \in V}[p \sim q \Leftrightarrow q \in v \cap \mathcal{FR}] \wedge v \neq \emptyset$$

Note that $\mathcal{FR}$ (the fact type roles) is the subset of $\mathcal{RO}$ which is not used in the identification of any object type in the schema. This is more formally defined later in the paper. For abstraction purposes, we only consider those mandatory constraints which are not directly inferable.

2.2.2. Uniqueness constraint

To introduce the concept of uniqueness, we use the predicate $\mathsf{Unique} \subseteq \wp(\mathcal{RO})$. A uniqueness constraint requires each tuple in the projection of the join of the given roles (based on asserted join conditions) to appear only once. A uniqueness constraint which involves roles from only one predicate is referred to as an internal uniqueness constraint ($\mathcal{IU}$).

$$\mathcal{IU} \triangleq \{\mathsf{Unique}(v) \mid \forall_{p,q \in v}[\mathsf{Rel}(p) = \mathsf{Rel}(q)]\}$$

An internal uniqueness constraint simply specifies that each tuple instance for that predicate will have a unique value combination for the constrained roles. If more than one predicate is involved in the uniqueness constraint, then the constraint is classified as an *external* uniqueness constraint ($\mathcal{EU}$).

$$\mathcal{EU} \triangleq \{\mathsf{Unique}(v) \mid \forall_{p,q \in v}[\mathsf{Rel}(p) \neq \mathsf{Rel}(q)]\}$$

In this case, the predicates involved must be *joinable via common object types* [19]. The general interpretation of a uniqueness constraint is formulated in the *Uniquest Algorithm* provided in [34].

2.2.3. *Primary uniqueness constraint*

For every object type in the data schema, there must be some way in which to uniquely identify each instance of that object type. In other words, we insist that every object-type is *identifiable*.

To identify the instances of non-value types ($\mathcal{NV}$), one uniqueness constraint must be selected to be the primary means of identification for that object type. We call the set of such uniqueness constraints 'PUnique' and require that $\mathsf{PUnique}(v) \Rightarrow \mathsf{Unique}(v)$. If this uniqueness constraint only involves one role, the identification scheme is often collapsed into a reference mode for graphical convenience. The reference mode of an object type is placed in brackets under the object type name. For example, Fig. 4(b) shows the graphical abbreviation for the explicit identification scheme represented in Fig. 4(a).

The algorithms in this paper do not consider this graphical abbreviation. Instead, they presume that all reference schemes are explicitly represented through uniqueness constraints. For more information about primary uniqueness constraints refer to [17]. For more detailed formal requirements on identification in ORM schemas, refer to [19, 23]. Every non-value object type must have exactly one primary identification scheme. That is:

$$\forall_{x \in \mathcal{NV}} \exists!_v[\mathsf{PUnique}(v) \wedge x \in \mathsf{Identifies}(v)]$$

where Identifies is defined as:

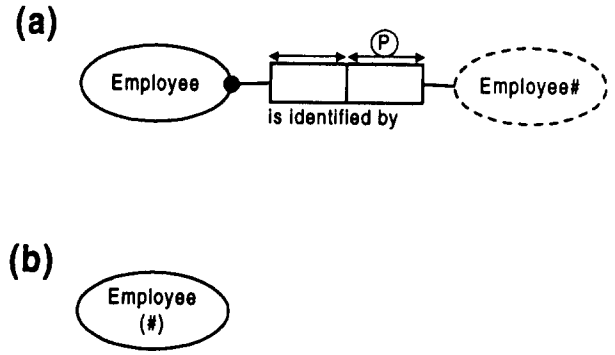


Fig. 4. (a) Explicit identification scheme; (b) implicit identification scheme.

$$\mathsf{Identifies}(v) \triangleq \mathsf{SubHierarchy}(\mathsf{LowestCSup}(\{\mathsf{Player}(p) \mid p \in \mathsf{OtherRoles}(v)\}))$$

The set of roles which are contained in the primary uniqueness constraint of a given non-value type is given by $\mathsf{PIdRoles}: \mathcal{NV} \to \wp(\mathcal{RO})$ such that:

$$\mathsf{PIdRoles}(x) \triangleq \{p \mid \exists v[p \in v \wedge x \in \mathsf{Identifies}(v)]\}$$

The set of predicates which are used to identify a given non-value type is given by the function $\mathsf{PIdRels}: \mathcal{NV} \to \wp(\mathcal{RL})$ such that:

$$\mathsf{PIdRels}(x) \triangleq \{\mathsf{Rel}(r) \mid r \in \mathsf{PIdRoles}(x)\}$$

2.2.4. Occurrence frequency constraint

Uniqueness constraints are used to specify that instances of object types may play a certain combination of roles at most once. Occurrence frequency constraints specify the more general condition that the number of times that object instances may play a combination of roles is restricted to within a fixed range. The condition that the instances of a set of roles σ must occur at least n and at most m times is denoted by $\mathsf{Frequency}(\sigma, n, m)$. The semantics of Frequency are fully defined in [19].

The function MaxFreq returns the maximum number of times an object type instance may participate in the given role. $\mathsf{MaxFreq}: \mathcal{RO} \to \mathbb{N}$ is defined as:

$$\mathsf{MaxFreq}(r) \triangleq \begin{cases} 1 & \text{if } \mathsf{Unique}(\{r\}) \\ m & \text{if } \mathsf{Frequency}(\{r\}, n, m) \\ \infty & \text{otherwise} \end{cases}$$

Note that when taking set types, sequence types, etc. into consideration, $\mathsf{Unique}(\sigma)$ should be replaced by $\mathsf{Unique}(\sigma) \vee \mathsf{ExUnique}(\sigma)$, where $\mathsf{ExUnique}(\sigma)$ is the class of *existensional uniqueness constraints*. This later constraint class is crucial in defining complex types such as set types [25], [24].

2.2.5. Set-comparison constraints

Set-comparison constraints (which we will refer to as 'set constraints') are used to specify conditions which apply between the population sets of two role sequences. If X is a set, then X^+ denotes the set of sequences built from elements of X. For sequences, we presume that the operation $z[i]$ returns the ith element of sequence z. $\mathsf{Set}(z)$ coerces a sequence z into a set of elements, so:

$$\mathsf{Set}(z) \triangleq \{x \mid \exists_i[z[i] = x]\}$$

We use $p \in z$ as an abbreviation for $\exists_i[z[i] = p]$ while $|z|$ denotes the length of sequence z. To determine which position a particular element occupies in a given sequence, we use the function Pos. For sequences where no two elements appear more than once in the sequence, we can define Pos as:

$$\mathsf{Pos}(p, z) \triangleq \quad \text{such that } z[i] = p$$

The relations Subset, Equality, Exclusion each apply to an ordered pair of role sequences (we

do not consider an n-ary form of exclusion constraint in this paper). The subset constraint (defined by relation $\mathsf{Subset} \subseteq \mathcal{RO}^+ \times \mathcal{RO}^+$ species that the population of the first role sequence is necessarily a subset of the population of the second role sequence; the equality constraint ($\mathsf{Equality} \subseteq \mathcal{RO}^+ \times \mathcal{RO}^+$) specifies that the population of the first role sequence must be exactly equal to the population of the second role sequence; while the exclusion constraint ($\mathsf{Exclusion} \subseteq \mathcal{RO}^+ \times \mathcal{RO}^+$) specifies that the population of the first role sequence does not contain any tuple which is in the population of the second role sequence. For a more formal definition of these constraints, please refer to [19, 23].

From their definitions, it is easy to infer an implied subset constraint between every optional role and every mandatory role played by the same object type. Similarly it is possible to infer an *implied* equality constraint between every mandatory role played by the same object type. We will not consider set constraints which are inferable in this manner. That is:

$$\forall_{p,q:\mathsf{Subset}(\langle p\rangle,\langle q\rangle)\vee\mathsf{Equality}(\langle p\rangle,\langle q\rangle)}[\neg\mathsf{Mand}(q) \wedge (\mathsf{Mand}(p) \Rightarrow \mathsf{Player}(p)\mathsf{SubOf}\ \mathsf{Player}(q))]$$

From Subset, Equality and Exclusion, we derive the more generic predicate SetCon using the following rules:

$$\mathsf{SC}(v, w) \triangleq \mathsf{Subset}(v, w) \vee \mathsf{Equality}(v, w) \vee \mathsf{Exclusion}(v, w)$$

$$\mathsf{SetCon}(v, w) \triangleq \mathsf{SC}(v, w) \vee \mathsf{SC}(w, v)$$

The underlying intuition is that if $\mathsf{SetCon}(v, w)$, then some set constraint exists which involves the roles in v and w.

From these rules, we can specify an even more generic definition for SetCon with only a single parameter. If $\mathsf{SetCon}(v)$ then some set constraint exists which involves the roles in v.

$$\mathsf{SetCon}(v) \triangleq \exists_w[\mathsf{SetCon}(v, w)]$$

2.2.6. Refinements to the type classification

Relationship types can now be partitioned into two important subclasses – the *fact types* and the *reference types*. Reference types ($\mathcal{RF}$) are those relationship types which are used within the primary identification scheme of some non-value type:

$$\mathcal{RF} \triangleq \{\mathsf{Rel}(p) \mid \exists_{x \in \mathcal{NV}}[p \in \mathsf{PIdRoles}(x)]\}$$

Fact types ($\mathcal{FT}$) are those relationship types which are not used within the primary identification scheme of a basic entity type or subtype.

$$\mathcal{FT} \triangleq \mathcal{RL} - \mathcal{RF}$$

The set $\mathcal{FR} \subseteq \mathcal{RO}$ is used to refer to those roles which are contained within a fact type. That is:

$$\mathcal{FR} \triangleq \{p \in \mathcal{RO} \mid \mathsf{Rel}(p) \in \mathcal{FT}\}$$

Most object types can only be instantiated by instances which participate in some fact type ($\mathcal{FT}$). Instances of *lazy object types*, however, can exist without participating in any fact type.

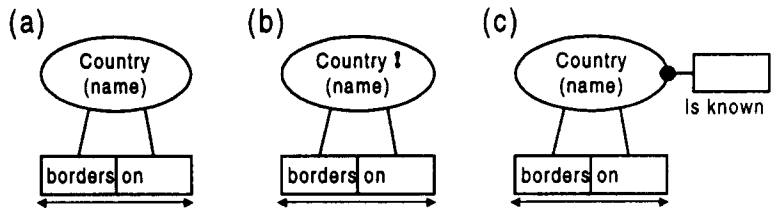


Fig. 5. 'Country' object type represented as: (a) non-lazy; (b) lazy; (c) implied lazy.

We graphically represent a *lazy object type* by concatenating an exclamation mark to the end of the object type name (x!). As an example, consider Fig. 5(a). Only countries which border another country can be recorded. In Fig. 5(b), however, countries may be recorded even if they do not border (by land) any other country (e.g. Australia).

A lazy entity type's behavior can be compared to that of an object type that mandatorily participates in a unary (one roled) fact type which represents its existence (as depicted in Fig. 5(c)). For the purposes of this paper, we consider a lazy entity type to be a graphical simplification to conveniently represent those entity types which participate in a single, mandatory unary role. No special consideration is therefore necessary for lazy entity types in the ensuing algorithms.

As stated before, other complex types like set types and sequence types are not discussed in full detail in this article, however, we will briefly return to this issue.

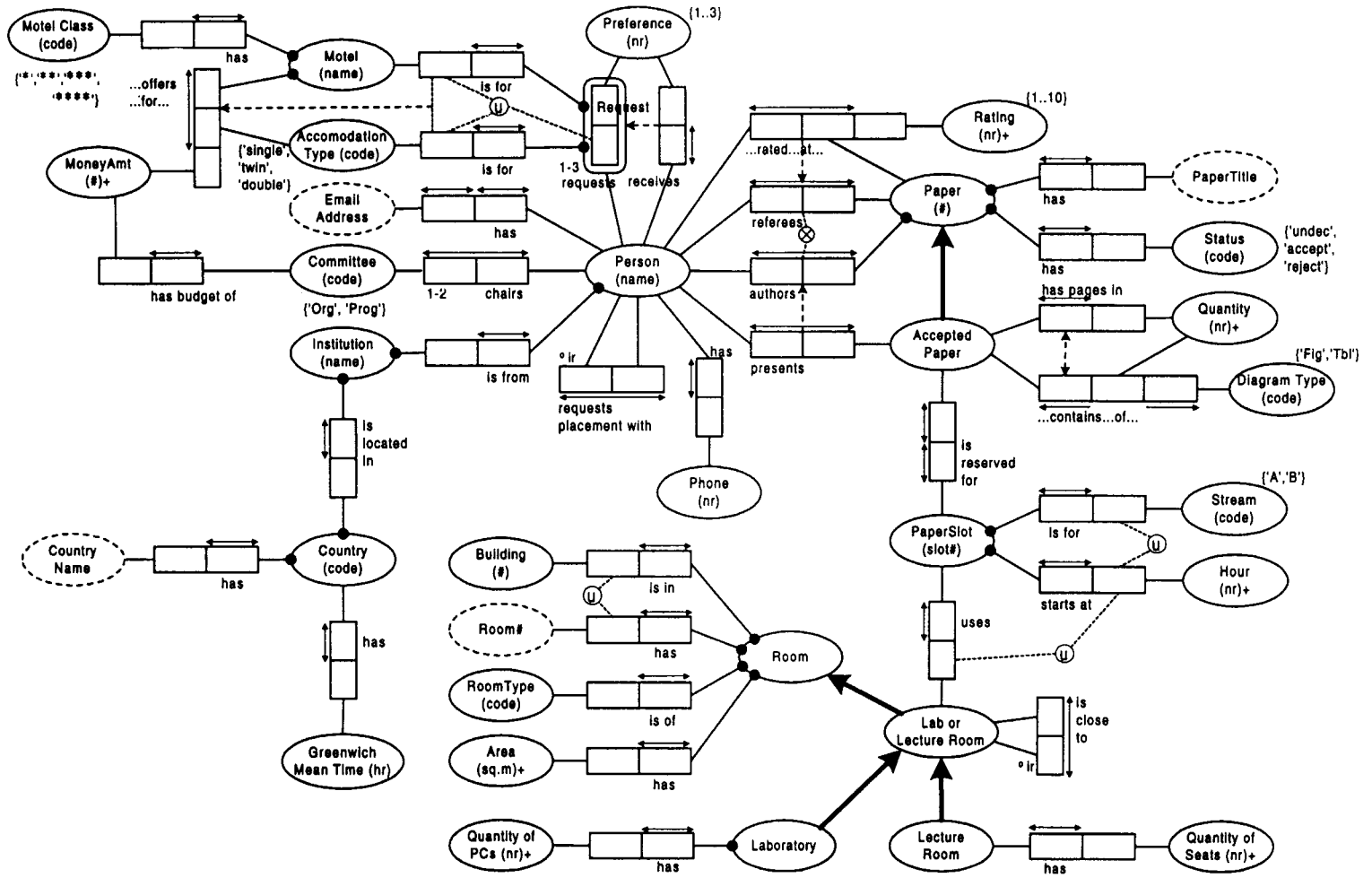
2.3. Summary

A conceptual schema $\mathscr{CS}$ can now be defined in terms of both the information structure $\mathscr{IS}$ and the basic constraints which apply to this information structure.

$$\mathscr{CS} = \langle \mathscr{IS}, \mathsf{Mand}, \mathsf{Unique}, \mathsf{PUnique}, \mathsf{Frequency}, \mathsf{Subset}, \mathsf{Equality}, \mathsf{Exclusion} \rangle$$

Conceptual schemas can have many other components, including ring constraints, subtype definitions, derived fact types and other extraneous constraints. None of these, however, will be considered in this paper, because they do not impact on the abstraction algorithms presented.

An example ORM conceptual schema can be found in Fig. 6. Entity types are depicted as named, solid ellipses. Value types are shown as named, broken ellipses. Predicates are shown as named sequences of role boxes, with the predicate name located in or beside the first role of the predicate. A nested object type is shown as a frame around a predicate (e.g. 'Request'). Arrow-tipped bars over one or more role boxes indicate an internal uniqueness constraint over these roles. A black dot at the base of a connector between an object type and a role indicates a mandatory constraint. Other constraints are represented as defined in [17]. As an example, consider the conceptual schema depicted in Fig. 6. In this schema we have $\mathsf{EmailAddress} \in \mathscr{VL}$, $\mathsf{Preference} \in \mathscr{OB}$, and $\mathsf{requests} \in \mathscr{RL}$. This schema is used as the running example throughout this article.



SUBTYPE DEFINITIONS

each Accepted Paper **is a** Paper **that has** Status 'accept'
each LabOrLecRoom **is a** Room **that is of** RoomType **in** {'lab','lec'}
each Laboratory **is a** Room **that is of** RoomType 'lab'
each Lecture Room **is a** Room **that is of** RoomType 'lec'

Fig. 6. Academic conference example application.

3. Anchoring fact types

In the following sections, we introduce a method by which we can view an ORM conceptual schema at various levels of abstraction. At each subsequently higher level of abstraction, we show only the most conceptually important (major) object types from the previous level, thereby creating a procedure that generates an incremental summary of the schema based on conceptual relevance. To this end, we first provide a mechanism by which the major types in a given schema can be derived.

Throughout this paper, sets of conceptual objects from the current conceptual schema $\mathscr{CS}$ will be denoted as belonging to a particular abstraction view by subscripting the set with the abstraction level. For example, $\mathscr{TP}_i$ refers to the set of types in $\mathscr{CS}$ which appear at abstraction level i.

3.1. Definitions

To define the notion of a *major* object type (relative to a particular abstraction level i) we consider each fact type individually and decide which object type(s) is (are) the most conceptually important participants in this fact type. We say that a role *anchors* a fact type to its player at the current abstraction level, if that player is (one of) the conceptually most important participants in the fact type. Conceptual importance is, to a certain degree subjective. However, a reasonable (and often measureable) indicator of conceptual importance is the proportion of the population of each object type that participates in the fact type.

The conceptual importance of role p played by object type X in fact type predicate F can be indicated by:

$$\begin{cases} \dfrac{|\mathsf{Pop}(\pi_{pF})|}{|\mathsf{Pop}(X)|} & \text{if } |\{q \in \mathscr{FR} \mid \mathsf{Player}(q) = X\}| > 1 \\ 0 & \text{otherwise} \end{cases}$$

where Pop is the population function and $\pi_p F$ indicates the projection on role p of fact type F. Consider, for example, a fact type '*Subject* is lectured by *Academic*'. Suppose we know that a greater percentage of 'Subjects' are lectured by an 'Academic' than the percentage of 'Academics' who lecture a 'Subject'; so:

$$\frac{|\mathsf{Pop}(\pi_1(\text{'}\textit{Subject}\textit{ is}\text{ lectured by }\textit{Academic}\text{'}))|}{|\mathsf{Pop}(\textit{Subject})|} > \frac{|\mathsf{Pop}(\pi_2(\text{'}\textit{Subject}\text{ is lectured by }\textit{Academic}\text{'}))|}{|\mathsf{Pop}(\textit{Academic})|}$$

It is obvious that, as a result, a particular 'Subject' is more likely to be participating in the fact type than a particular 'Academic'. It can also be observed that the fact type '*Subject* is lectured by *Academic*' is more likely to be accessed in relation to a particular 'Subject' than in relation to a particular 'Academic'. We therefore consider 'Subject' to be the more 'conceptually important participant' and consider '*Subject* is lectured by *Academic*' to be anchored

on the role played by 'Subject'. This reasoning will, in general, only be useful when we have access to a typical population of a conceptual schema. When only the conceptual schema is available, we must rely on the conceptual constraints to derive such information from the type level. This is the approach taken in this paper.

The fact that a given role is an anchor, at abstraction level i, captured by the predicate: $\mathsf{Anchor}_i \subseteq \mathcal{FR}_i$, where $\mathcal{FR}_i$ represents the fact type roles which are present at abstraction level i. For convenience, we also introduce the infix predicate $\mathsf{AnchoredTo}_i \subseteq \mathcal{FT}_i \times \mathcal{OB}$, which indicates that the given role is anchored to the given object type (at abstraction level i):

$$r\ \mathsf{AnchoredTo}_i\ x \triangleq \mathsf{Anchor}_i(r) \wedge \mathsf{Player}(r) = x$$

When considering anchors, it is important to do so in their proper context, i.e. at a particular level of abstraction. For example a role which receives one hundred percent participation (i.e. a mandatory role), may become implied mandatory at a higher level of abstraction and consequently lose 'conceptual importance'. As an example, consider the schema fragments in Fig. 7. In Fig. 7(a), 'Subject' mandatorily participates in '*Employee* teaches *Subject*' and 'Department' mandatorily participates in '*Employee* works for *Department*'. Because these object types have one hundred percent participation, the corresponding roles are therefore considered to be anchors in Fig. 7(a). In Fig. 7(b), which shows the next highest level of abstraction, however, the same roles (played by 'Subject' and 'Department') are only mandatory by implication. The roles played by 'Subject' and 'Department', therefore, lose their conceptual importance.

Anchors for fact types are selected by comparing the conceptual importance of the roles involved. To this end, we introduce the notion of the weight of role, to indicate how firmly the role is attached to its player:

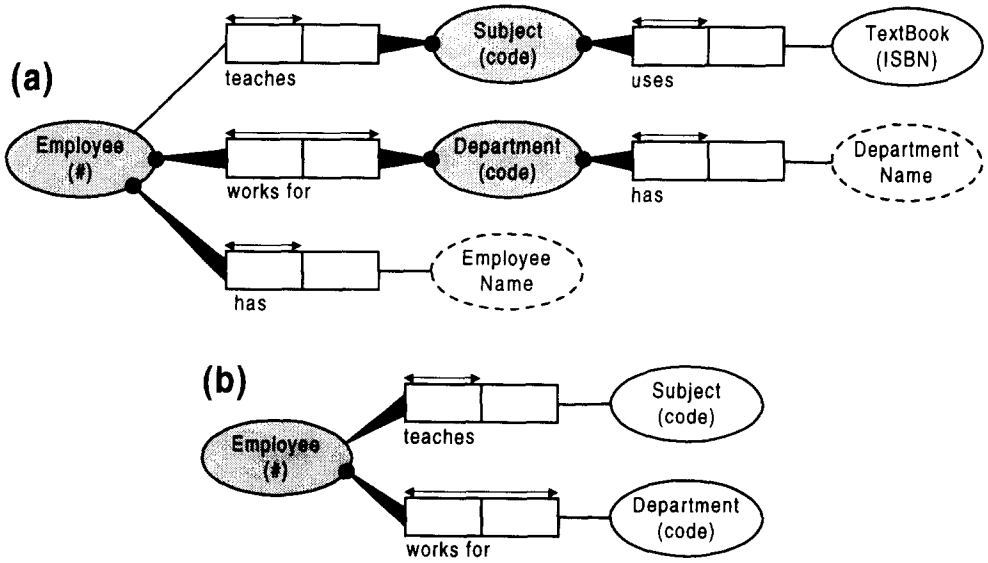


Fig. 7. (a) Abstraction level n; (b) abstraction level $n+1$.

$$\mathsf{Weight}_i : \mathscr{FR}_i \rightarrow \mathbb{N}$$

The weight function is used as a rough indicator of both the importance of each role within a fact type and of the relative importance of anchors between fact types. A role is considered to be an anchor if its weight is the highest (or equal highest) weight of any role in the same fact type. An anchor can therefore be defined in terms of the role weights, as such:

$$\mathsf{Anchor}_i(r) \stackrel{\Delta}{=} \mathsf{Weight}_i(r) > 0 \wedge \mathsf{Weight}_i(r) = \max\{\mathsf{Weight}_i(p) \mid p \in \mathsf{CoRoles}(r)\}$$

A corollary which ensues is:

Corollary 3.1.

$$(\mathsf{Weight}_i(r) < \mathsf{Weight}_i(p) \wedge p \in \mathsf{CoRoles}(r)) \Rightarrow \neg \mathsf{Anchor}_i(r)$$

The particular weight associated with each anchor is defined by the WeightSchema procedure, which is introduced in the next subsections.

We consider a fact type predicate to be anchored ($\mathsf{Anchored}_i \subseteq \mathscr{FT}_i$) if it contains a role which anchors that fact type predicate to an object type:

$$\mathsf{Anchored}_i(s) \stackrel{\Delta}{=} \exists_{r \in \mathsf{Roles}(s)}[\mathsf{Anchor}_i(r)]$$

or in other words, if the sum of the weights of each of its roles is greater than zero:

Corollary 3.2.

$$\mathsf{Anchored}_i(s) \stackrel{\Delta}{=} \sum_{r \in \mathsf{Roles}(s)} \mathsf{Weight}_i(r) > 0$$

We refer to an object type, which has at least one fact type anchored to it, as an $\mathsf{AnchorPoint}_i \subseteq \mathscr{OB}_i$:

$$\mathsf{AnchorPoint}_i(x) \stackrel{\Delta}{=} \exists_{r \in \mathscr{FR}_i}[\mathsf{Player}(r) \sim x \wedge \mathsf{Anchor}_i(r)]$$

For an abstraction level i to be completely anchored, every fact type within the schema must be anchored:

$$\mathsf{AnchoredSchema}(i) \stackrel{\Delta}{=} \forall_{s \in \mathscr{FT}_i}[\mathsf{Anchored}_i(s)]$$

This is the goal for this section.

3.2. Weighting a schema

The procedure called WeightSchema, shown below, automatically assigns default weights to each fact-type role, based on the given semantic constraints within the associated conceptual domain. Weight is a total function. Since there will always be some subjective qualities that cannot be captured by such an automatic procedure, it is important that the user has the ability to override some automatic weighting decisions that may be questionable. For this reason, and because the user will usually only want to express such alternative preferences

once, we allow our automatic abstraction procedure to take previous user-driven weightings into account.

The automatic weighting is defined by a set of weighting rules that associate a weight to each fact type role, based on the context of these roles. The weighting algorithm works by continuously trying to increase the weight of the roles. This is a repetitive process, as increases of weights in one part of the schema may lead to further increases in other parts of the schema.

We will refer to the weightings, which were explicitly generated by a user decision, through the function UserWeight: $\mathscr{FR} \rightarrow \mathbb{N}$. The weights that are derived automatically from the underlying schema are provided by the function AutoWeight: $\mathbb{N} \times \mathscr{FR} \rightarrow \mathbb{N}$. This function is introduced in the next subsection by a set of derivation rules. We employ the notation $\mathsf{AutoWeight}_i(p)$ rather than $\mathsf{AutoWeight}(i, p)$ because of the fact the index i is used as a label rather than carrying any specific semantics.

Once WeightSchema has been automatically performed, the user would have a further opportunity to alter the UserWeights by modifying the set of anchors produced in accordance with an appropriate set of modification rules.

```
WeightSchema: 𝒞𝒮 → (ℱℛ → ℕ)
WeightSchema(𝒞𝒮) ≜
VAR
  Weight: ℱℛ → ℕ ;
  p: ℱℛ ;
BEGIN
  {Initialize anchors}
  FOR EACH p ∈ ℱℛ DO
  Weight(p) := UserWeight(p) ;
  END ;
  WHILE ∃i,p[AutoWeighti(p) > Weight(p)]
  Weight(p) := max{AutoWeighti(p) | ⟨i, p⟩ ∈ dom(AutoWeight)} ;
  END ;
  RETURN Weight ;
END WeightSchema ;
```

This algorithm will always terminate. From the condition on the WHILE loop we can see it terminates if $\neg\exists_{i,p}\,[\mathsf{AutoWeight}_i(p) > \mathsf{Weight}(p)]$. In the next subsection we will also see that the maximum value returned by AutoWeight is fixed to 10. From the body of the WHILE loop follows that we never reduce the Weight of a role. This means that for any role p once $\mathsf{Weight}(p) \geq 10$ we cannot find a rule labeled i such that $\mathsf{AutoWeight}_i(p) > \mathsf{Weight}(p)$. Which means the loop must eventually terminate.

3.3. Rules for role weighting

The following paragraphs describe each of the twelve rules that together define AutoWeight. The resulting weightings returned by these rules serve as a comparative guide, and should at some stage be refined based on empirical testing in practical situations. The existing rules have been formulated after studying a number of cases to observe the effect of particular constraints on the associated populations and on the conceptual importance of surrounding object types.

Rule 1 – Mandatory roles

All non-implied mandatory roles have, by definition, full participation by the population of the player(s). Therefore, any fact type role which is involved in a mandatory role constraint (even if this is a disjunctive mandatory constraint) should be weighted, *unless* the mandatory role constraint is implied (as described in InferMind). This is the only rule which can cause a fact type predicate to be anchored more than once.

$$\mathsf{AutoWeight}_1(p) \triangleq \max\left(\left\{\left\lceil \frac{10}{|v|} \right\rceil \middle| \mathsf{Mand}(v) \wedge p \in v \wedge \neg \mathsf{InferMand}(v)\right\} \cup \{0\}\right)$$

A role can be involved in a number of mandatory role constraints. The simplest case would be where the role itself is mandatory, which would lead to a weight of 10. However, a role may be involved in a disjunctive mandatory role. This means that the instances of the participating object type must play at least one of the roles involved in the disjunctive mandatory role. In this latter case, the weight of 10 is 'shared' among the involved roles. As one role may be involved in a number of mandatory role constraints, we take the weight to be the maximum of the possible weights that would follow from these involvements.

For example, rule 1 would cause each fact type in Fig. 8 to be anchored towards the non-implied mandatory role played by Employee.

This rule can also be considered in the broader context of complex types like sequence types, bag types, etc. We can now discuss why these complex types to not require special provisions in our algorithm. In Fig. 9(a) we show an example of a set type: namely 'Convoy'. A convoy consists of a set of ships, each of which is commanded by a unique captain. Both a ship and its captain are each individually identified by a name. A convoy, however, is identified by a set of ships.

In Fig. 9(b), this set type is modeled in terms of more elementary relationships using the existensional uniqueness constraint (represented by the encircled EU symbol) [24, 25]. The

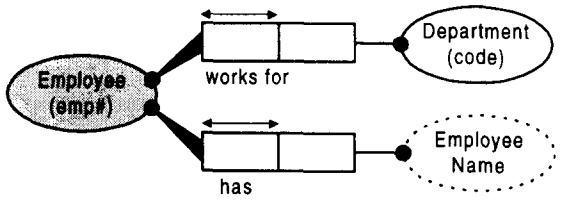


Fig. 8. All non-implied mandatory roles are weighted.

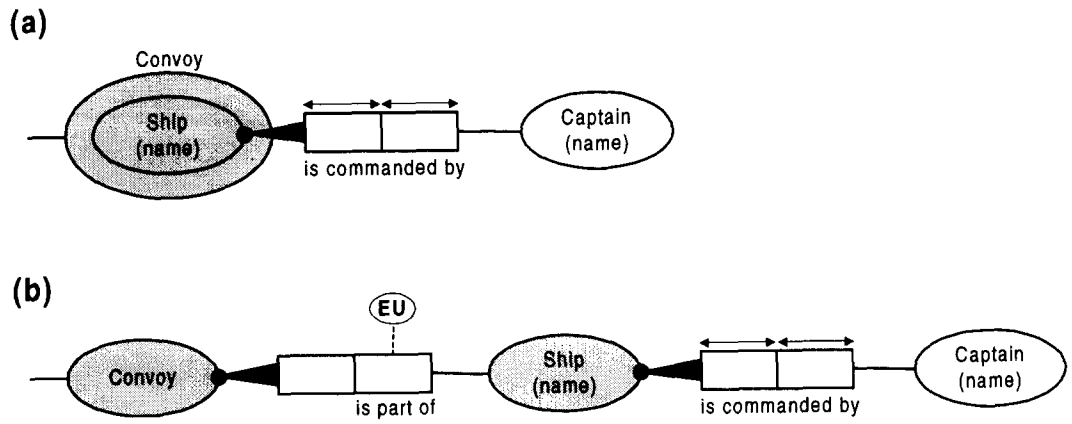


Fig. 9. A convoy of ships modeled using (a) a complex set type; (b) elementary relationships.

AutoWeight rules that we are defining, can therefore be directly applied to the elementary representation of the complex types.

Rule 2 – Unary roles

The player of the only role in a unary predicate must obviously be 'the most important participant' in that predicate. All roles in unary predicates are therefore weighted.

It should be remembered that, for the purposes of this algorithm, lazy object types are treated like non-lazy object types which play a mandatory unary predicate representing the existence of the instances.

$$\mathsf{AutoWeight}_2(p) \triangleq \textbf{if } \mathsf{CoRoles}(p) = \{p\} \textbf{ then } 10 \textbf{ else } 0$$

Fig. 10 shows an example subschema in which every unary predicate is anchored on its one role.

Rule 3 – Non-leaf object types

A leaf facttype role ($\mathsf{Leaf} \subseteq \mathcal{FR}$) is one which has a player that plays only that fact type role. That is:

$$\mathsf{Leaf}(p) \triangleq \neg\exists_{q \in \mathcal{FR}}[\mathsf{Player}(p) = \mathsf{Player}(q) \wedge \mathsf{Rel}(p) \neq \mathsf{Rel}(q)]$$

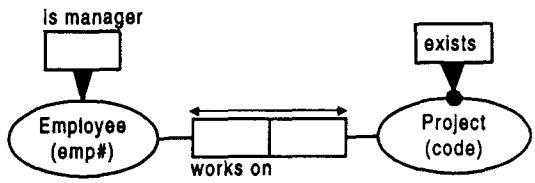


Fig. 10. All unary roles are anchored.

If only one role in a fact type is played by a non-leaf object type, then this role is considered 'conceptually important' enough to be given a strong weighting.

$$\text{AutoWeight}_3(p) \triangleq \textbf{if } \text{Leaf}(p) \wedge \forall_{q \in \text{OtherRoles}(p)}[\text{Leaf}(q)] \textbf{ then } 9 \textbf{ else } 0$$

This rule as such is rather straightforward. The reason to assign only a weight of 9, instead of 10, is that mandatory non-leaf roles are considered to be conceptually more important than optional non-leaf roles.

In the example subschema in Fig. 11, rule 3 would be fired, causing both fact types to be anchored towards the Employee object type. Notice that 'Room' is actually a leaf object type because, while it participates in three roles, it only participates in one fact type role. '*Room* is in *Building*' and '*Room* has *Room#*' are not considered in the weighting procedure as they are both reference types.

Rule 4 – Smallest maximum frequency

The maximum frequency of the population of a role can be determined from one of two constraints. A single role uniqueness constraint indicates that the role has a maximum frequency of one. Alternatively, an occurrence frequency constraint often explicitly specifies the maximum frequency of a role. If exactly one role within a fact type predicate has a smaller maximum frequency than all other roles in that fact type, then this role should be anchored.

$$\text{AutoWeight}_4(p) \triangleq \textbf{if } \forall_{q \in \text{OtherRoles}(p)}[\text{MaxFreq}(p) < \text{MaxFreq}(q)] \textbf{ then } 2 + \left\lceil \frac{6}{\sqrt{\text{MaxFreq}(p)}} \right\rceil \textbf{ else } 0$$

The closer the maximum frequency of a role is to 1, the higher the weighting. The maximum AutoWeight of 8 is applied in those cases in which a uniqueness constraint holds on the role, causing MaxFreq to be 1. If the maximum frequency is higher than 1, the AutoWeight will become lower and lower, down to a minimum of 2. However, because increments in MaxFreq should have less effect if the frequency is already high, we have taken the division of the square root of the MaxFreq value. For example, the increment from a MaxFreq of 8 to one of 9 will have less effect on the AutoWeight than an increment from 1 to 2. The result is a curve that drops down quickly from a maximum Weight of 8, but starts to level out when it gets closer to 2.

The example in Fig. 12 depicts a subschema in which the fact type '*Project* is managed by

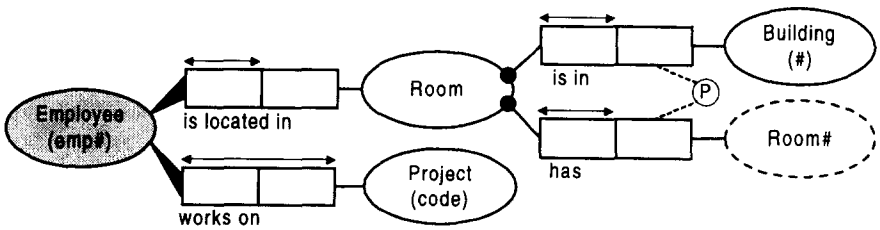


Fig. 11. Non-leaf object types may indicate automatic anchorage.

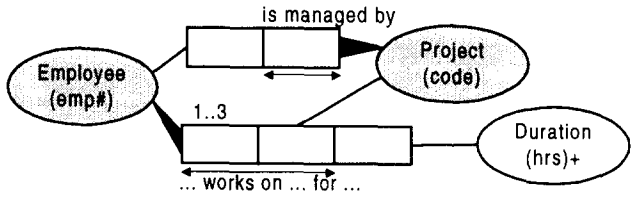


Fig. 12. Roles with the smallest maximum frequency may be anchored.

Employee' is anchored due to a uniqueness constraint, and the fact type '*Employee* works on *Project* for *Duration*' is anchored due to a frequency constraint.

Rule 5 – Non-value types

If exactly one role in a fact type is played by a non-value type, then the fact type should be anchored on this role.

$$\mathsf{AutoWeight}_5(p) \stackrel{\Delta}{=} \textbf{if } \mathsf{Player}(p) \notin \mathcal{VL} \wedge \forall_{q \in \mathsf{OtherRoles}(p)}[\mathsf{Player}(q) \in \mathcal{VL}] \textbf{ then } 7 \textbf{ else } 0$$

The rationale behind this is that value types are by definition conceptually less important than non-value types.

In the example shown in Fig. 13, rules 1 to 4 fail to determine an appropriate anchorage for either fact type. Rule 5, however, triggers the obvious conclusion that both fact types should be anchored towards 'Employee'.

Rule 6 – Anchor points

As we have already discussed, those object types which serve as anchor points to fact types are considered to possess a relatively high conceptual importance. Therefore, if exactly one role in a given fact type is played by an object type which became an anchorpoint via rules 1 to 5, the fact type is anchored on this role. For this purpose we introduce the notion of a 'heavy role' as:

$$\mathsf{HeavyRole}(p) \stackrel{\Delta}{=} \exists_{s:\mathsf{Weight}(s) \geq 7}[p \sim s]$$

The Auto Weight rule then becomes:

$$\mathsf{AutoWeight}_6(p) \stackrel{\Delta}{=} \textbf{if } \mathsf{HeavyRole}(p) \wedge \forall_{q \in \mathsf{OtherRoles}(p)}[\neg \mathsf{HeavyRole}(q)] \textbf{ then } 6 \textbf{ else } 0$$

In the example subschema of Fig. 14, the uniqueness constraint on '*Employee* is managed by *Project*' causes rule 4 to anchor the upper fact type towards 'Project'. Since 'Project' is now

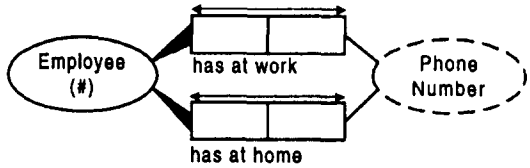


Fig. 13. Roles played by non-value-types may become automatically anchored.

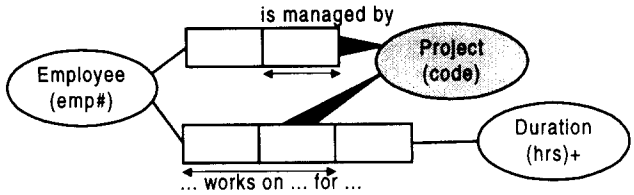


Fig. 14. Roles played by anchorpoints may become automatically anchored.

the only participant in the ternary fact type which is an anchorpoint, the lower fact type is anchored on 'Project'.

Rule 7 – Single-role set constraints

If a fact type is involved in exactly one single-role set constraint (i.e. subset, equality or exclusion constraint), and the role at the other end of the set constraint is anchored, then the constrained role in the given fact type is anchored.

$$\text{AutoWeight}_7(p) \triangleq \textbf{if } \exists_{s:\text{Anchor}(s)}[\text{SetCon}(\langle s\rangle, \langle p\rangle)] \wedge \forall_{q\in\text{OtherRoles}(p)}[\neg\text{SetCon}(\langle q\rangle)] \textbf{ then } 5 \textbf{ else } 0$$

In Fig. 15, the fact type '*Employee* is a supervisor in *Project*' is anchored to 'Employee' by rule 4, as a result of the simple uniqueness constraint. Since the role played by 'Employee' in the ternary fact type is connected to this anchored role via a single-role subset constraint, this role is consequently anchored by rule 7.

It is important to consider the case in Fig. 16, in which the single role set constraints contradict each other. In this case, rule 7 could not produce a determinant anchorage for the

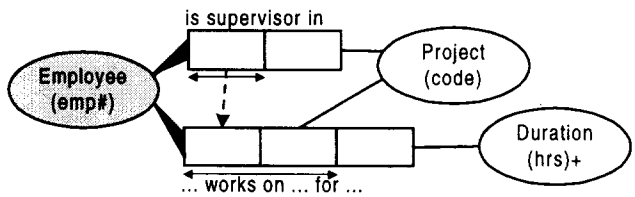


Fig. 15. A role connected to an anchored role by single role set constraints is anchored.

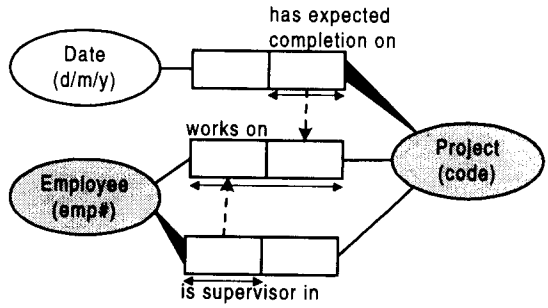


Fig. 16. Rule 7 does not consider cases in which a single-role set constraint contradicts another.

predicate 'works on'. We therefore ensure that this rule only fires on non-anchored fact types which are only involved in one single-role set constraint.

It is also important to realize why we chose not to require the contradicting set constraint to necessarily have an anchor assigned. It would have been inadequate to require the following condition on the other roles in p's fact type:

$$\forall_{q \in \mathsf{OtherRoles}(p)} \neg \exists_{s:\mathsf{Anchor}(s)}[\mathsf{SetCon}(\langle s \rangle, \langle q \rangle)]$$

We illustrate this, by considering the case shown in Fig. 17. If we assume that the above condition is adequate, (i.e. that rule 7 is fired as long as no other role in the fact type participates in an anchored single role subset constraint), then two possible scenarios are possible for the schema fragment below. Firstly, rule 7 could cause '*Person* owns *Car*' to be anchored towards 'Person'; which would then cause '*Person* has driven *Car*' to also be anchored towards 'Person'. Alternatively, rule 7 could first cause '*Person* caused crash of *Car*' to be anchored towards 'Car'; which would then cause '*Person* has driven *Car*' to also be anchored towards 'Car'. As a result, '*Person* has driven *Car*' could be anchored in either direction, depending on the order in which the rule was fired.

For this reason, we only allow fact types to be anchored on a role, p, if no other role in its fact type is involved in any kind of single-role set constraint (as defined in $\mathsf{AutoWeight}_7(p)$). The definition of rule 7 will, therefore anchor all fact types in Fig. 17, except for '*Person* has driven *Car*'.

Rule 8 – Multi-role set constraints

If a fact type is involved in exactly one (possibly multi-role) set constraint (i.e. subset, equality, or exclusion constraint), and exactly one of the roles in the fact type is in the corresponding position within the set constraint as an anchored role, then this role is itself anchored.

$$\mathsf{AutoWeight}_8(p) \stackrel{\Delta}{=} \textbf{if } \exists_{v,w:\mathsf{SetCon}(v,w)}[p \in v \wedge \mathsf{Anchor}(w[\mathsf{Pos}(p, v)]) \wedge \mathsf{SingleSetCon}(v, p)] \textbf{ then } 4 \textbf{ else } 0$$

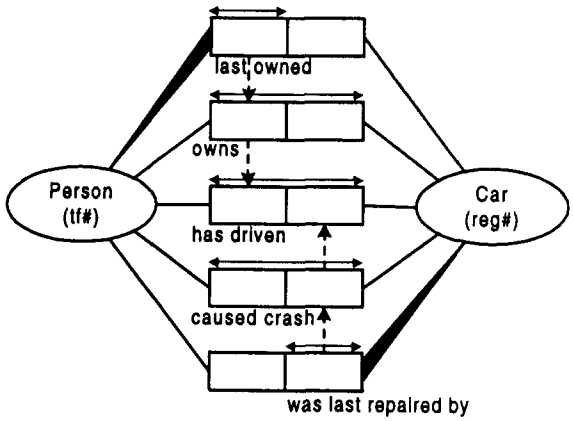


Fig. 17. Rule 7 requires that p's fact type participates in *no* other single-role subset constraint.

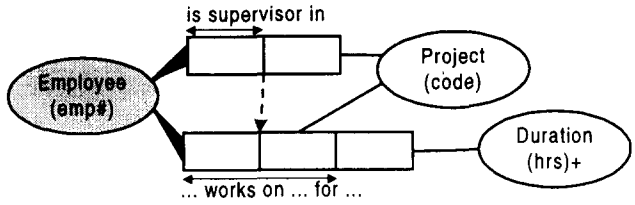


Fig. 18. Roles connected by multi-role set constraints are anchored.

where SingleSetCon(v, p) enforces the singularity of the set constraint v with respect to role p:

$$\mathsf{SingleSetCon}(v, p) \triangleq \forall_{x:\mathsf{SetCon}(x)}[v \neq x \Rightarrow \mathsf{OtherRoles}(p) \cap \mathsf{Set}(x) = \varnothing]$$

In the example in Fig. 18, the fact type '*Employee* is supervisor in *Project*' is anchored to 'Employee' by rule 4, as a result of the simple uniqueness constraint. Since the role played by 'Employee' in the ternary fact type is connected to this anchored role via a multi-role subset constraint, this role is consequently anchored by rule 8.

Similarly to rule 7, it is important to consider the case in Fig. 19, in which the multi-role set constraints contradict each other. In this case, rule 8 would not produce a determinant anchorage for the predicate 'works on'. We therefore only use this rule on non-anchored fact types which are only involved in one multi-role set constraint.

Rule 9 – Set constraints and anchor points

If there exists a non-implied set constraint in which one of the roles involved in the constraint is the only involved role in its fact type to be played by an anchorpoint and the fact type of the role corresponding to it in the other role sequence is not anchored, then this role should become an anchor. If a set constraint v anchors a role p in this sense, then we refer to this as Anchors(p, v):

$$\mathsf{Anchors}(v, p) \triangleq p \in v \wedge \mathsf{AnchorPoint}_i(\mathsf{Player}(p))$$

The resulting AutoWeight rule is then:

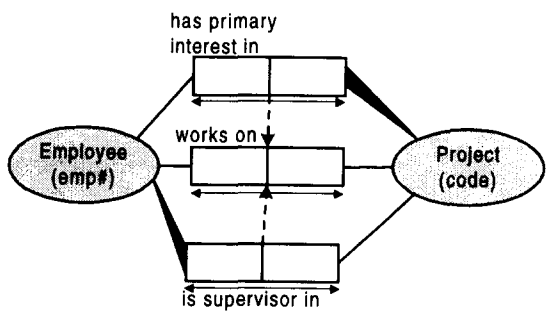


Fig. 19. Rule 8 does not consider cases in which a multi-role set constraint contradicts another.

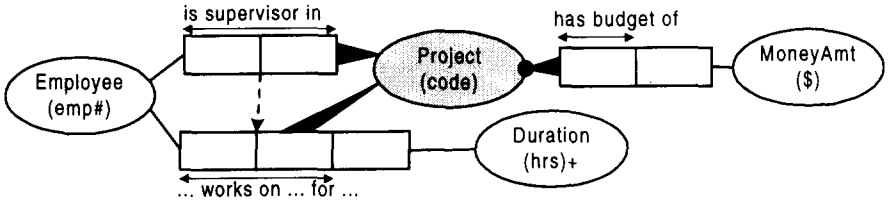


Fig. 20. Roles played by major object types and involved in multi-role set constraints are anchored.

$$\text{AutoWeight}_9(p) \triangleq \textbf{if } \exists_{v:\text{SetCon}(v)}[\text{Anchors}(v, p) \wedge \forall_{r \in \text{OtherRoles}(p)}[\neg\text{Anchors}(r, v)] \wedge \text{SingleSetCon}(v, p)] \textbf{ then } 3 \textbf{ else } 0$$

For example, in Fig. 20, the firing of rule 1 causes '*Project* has budget of *MoneyAmt*' to be anchored towards 'Project'. Since 'Project', consequently, becomes the only player of a role involved in the subset constraint to be an anchorpoint, rule 9 causes both of the other fact types to also be anchored towards 'Project'.

Rule 10 – Joining roles of set constraints

For this rule, we consider each role sequence which is involved in a set constraint and which spans more than one fact type. In these cases, a join condition must be specified (or inferred) to define the manner by which the populations of the involved fact types are related. We call those roles which are involved in the join condition of such a role sequence, the join roles for that role sequence, and define them through the function: $\text{JoinRoles}: \mathcal{RO}^+ \rightarrow \wp(\mathcal{RO})$, such that:

$$\text{JoinRoles}(v) \triangleq \{p \in \text{OtherRoles}(v) \mid \exists_{q,r \in v, s \in \text{CoRoles}(r)}[\text{Rel}(q) \neq \text{Rel}(r) \wedge \text{Rel}(p) = \text{Rel}(q) \wedge p \sim s]\}$$

AutoWeight Rule 10 anchors those unanchored fact types, in which only one role is the join role for some set constraint role sequence:

$$\text{JoinSingleSetCon}(p) \triangleq \exists_{v:\text{SetCon}(v)}[p \in \text{JoinRoles}(v)]$$

This join role becomes the anchor:

$$\text{AutoWeight}_{10}(p) \triangleq \textbf{if } \text{JoinSingleSetCon}(p) \wedge \forall_{q \in \text{OtherRoles}(p)}[\neg\text{JoinSingleSetCon}(q)] \textbf{ then } 2 \textbf{ else } 0$$

Fig. 21 shows an example to which this rule is applicable. The 'works for' predicate is first anchored to 'Employee' when Rule 1 (mandatory roles) is fired. The 'involved in' predicate is then anchored to 'Employee' by the activation of Rule 8 (multi-set constraint with single anchor). Lastly, rule 10 causes the 'sponsors' predicate to be anchored to 'Department', since the role played by 'Department' is the one which is used to join together the *target* role sequence of the subset constraint.

Rule 11 – First role of set constraints

If there is a multi-role, non-implied set constraint (i.e. subset, equality or exclusion

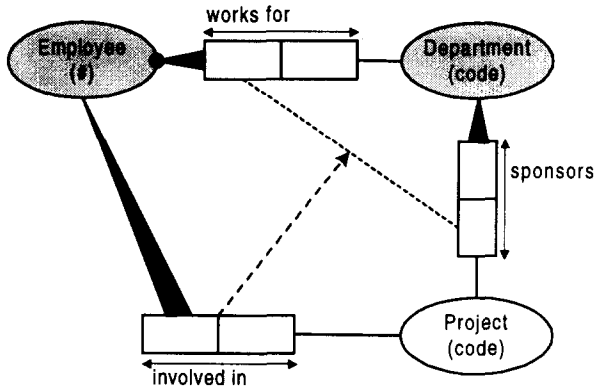


Fig. 21. Roles involved in the join condition of a multi-predicate set constraint role sequence are anchored.

constraint) and one of the involved roles has the lowest sequence position within one of the constraint's role sequence (for its predicate), then this role should become an anchor.

$$\text{AutoWeight}_{11}(p) \triangleq \textbf{if } \exists_{v:\text{SetCon}(v)} \forall_{s \in \text{CoRoles}(p) \cap \text{Set}(v)} [\text{AnchorPoint}_i(\text{Player}(s)) \Rightarrow \text{Pos}(p, v) \leqslant \text{Pos}(s, v)] \textbf{ then } 1 \textbf{ else } 0$$

This choice is based on the semantics which is derivable from the order in which the modeler chose to initially verbalize the fact type. An example of such a situation is depicted in Fig. 22.

Similarly to rules 7 and 8, it is important to consider the case in Fig. 23, in which the multi-role set constraints contradict each other. In this case, rule 11 would not produce a

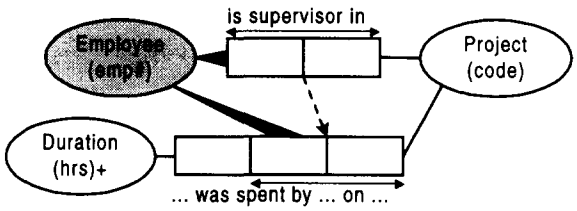


Fig. 22. Roles which appear first in set-constraint role sequences are anchored.

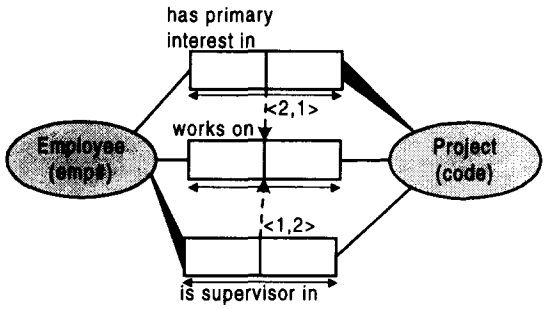


Fig. 23. Rule 11 does not consider cases in which a multi-role set constraint contradicts another.

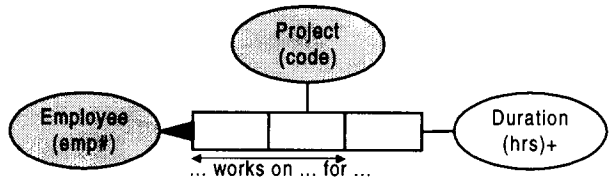


Fig. 24. Roles which are positioned in the first 'keyed' position are anchored.

determinant anchorage for the predicate 'works on'. We therefore only use this rule on non-anchored fact types which are only involved in one multi-role set constraint. Note, that in Fig. 23 we denote the role sequence order by specifying the sequence of role numbers that are involved at the 'source' of the set constraint.

Rule 12 – First role of internal uniqueness constraint

Any fact type, which is not already anchored, should be anchored, by default, on the first role that is involved in an internal uniqueness constraint. This choice is based on the semantics which is derived from the order in which the modeler chose to initially verbalize the fact type.

$$\mathsf{AutoWeight}_{12}(p) \triangleq \textbf{if } \mathsf{PosN}(p) = \min\{\mathsf{PosN}(q) \mid \exists_v[\mathsf{Unique}(v) \in \mathcal{JU} \wedge q \in v]\} \textbf{ then } 1 \textbf{ else } 0$$

Fig. 24 shows an example in which rule 12 is triggered.

4. Deriving abstraction levels

When a conceptual schema is abstracted, each progressively higher level of abstraction includes all the most conceptually important components from the previous level. To define an abstraction level, we must therefore first select the major object types and major fact types. We refer to those major types which form the foundation for an abstraction view at level i as $\mathcal{KER}_i$ (which will be defined formally below).

Once the kernel of an abstraction level has been calculated, there are still a number of steps which must be performed before the abstraction is complete. Firstly, any fact type predicate in which every role is played be a major object type is included in the abstraction. We do not include these predicates in the kernel itself, because we do no want these fact types to effect the outcome of future abstraction levels. Secondly, we include the identification scheme of all object types which appear in the abstraction. Finally, we restore the connectivity of our abstracted conceptual schema. This involves retaining both the connectivity of subtyping hierarchies, and the connectivity of non-type related object types.

The following subsections formally describe these steps in the abstraction process.

4.1. Major types

In a conceptual schema at a particular level of abstraction (i), the set of object types which do not have the lowest conceptual significance are referred to as the major object types

($\mathsf{MajorOT}_i$). Those object types which are of least conceptual significance are referred to as minor. We identify the major object types as the set of object types which have a higher total object type weight (OTWeight) than the minimum total object type weight for the current schema level.

Each object type also has an object type weight (OTWeight) associated with it at a particular abstraction level. The object type weight represents the sum of the weights of those fact type roles which are anchored to it; i.e.:

$$\mathsf{OTWeight}_i(x) \triangleq \sum_{r : \exists_y[\mathsf{AnchoredRole}(r,y) \wedge y \sim x]} \mathsf{Weight}_i(r)$$

where $\mathsf{AnchoredRole} \subseteq \mathcal{FR} \times \mathcal{OB}$ is true when the given fact type role is in $\mathcal{HER}_i$ and is anchored to an object type which is type related to the given object type; i.e.:

$$\mathsf{AnchoredRole}_i(r, x) \triangleq \mathsf{Player}(r) \sim x \wedge \mathsf{Anchor}_i(r) \wedge \mathsf{Rel}(r) \in \mathcal{HER}_i$$

An object type is considered to be *major* at a particular level of abstraction ($\mathsf{MajorOT} \subseteq \mathcal{OB}$) if its OTWeight is greater than the minimum weight for object types in the kernel at that level. Formally, MajorOT is defined as:

$$\mathsf{MajorOT}_i(x) \triangleq \mathsf{OTWeight}_i(x) > \min\{\mathsf{OTWeight}_i(y) \mid y \in \mathcal{HER}_i\}$$

The *major fact types* at a particular abstraction level ($\mathsf{MajorFT}_i \subseteq \mathcal{FT}$) are defined as those fact types which bridge between more than one subtyping hierarchy and in which every participant is a major object type at that level:

$$\mathsf{MajorFT}_i(x) \triangleq \forall_{s \in \mathsf{Roles}(x)}[\mathsf{MajorOT}_i(\mathsf{Player}(s))] \wedge \exists_{s,t \in \mathsf{Roles}(x)}[s \not\sim t]$$

4.2. Algorithm for determining next abstraction level

We refer to the set of component types and constraints included in the level i abstraction view of conceptual data schema $\mathcal{CS}$ as $\mathcal{CS}_i$. In the level 1 conceptual schema, $\mathcal{CS}_1$ (often abbreviated to $\mathcal{CS}$), all component elements are present. Increasing the level of abstraction will never increase the number of populatable types visible in the conceptual schema:

$$\mathcal{TP}_i \supseteq \mathcal{TP}_{i+1}$$

Of even greater importance, though, increasing the level of abstraction will necessary strictly decrease the number of populatable types within the abstraction kernel:

$$\mathcal{HER}_i \supset \mathcal{HER}_{i+1}$$

As described previously, each progressively higher level of abstraction includes all the most conceptually important components from the kernel of the previous level. For this reason, at each level of abstraction, we include all the major fact types from the previous level, plus all the major object types which participate in at least one of these major fact-types. Formally, we define the abstraction kernel at level $i + 1(\mathcal{HER}_{i+1})$ as:

$$\mathcal{HER}_{i+1} \triangleq \{x \mid \mathsf{MajorFT}_i(x)\} \cup \{\mathsf{Player}(p) \mid \mathsf{MajorFT}_i(\mathsf{Rel}(p))\}$$

Notice that this definition does not necessarily include every major object type of one abstraction level in the kernel of the next level. The kernel at a particular level of abstraction will only include those major object types which participate in some fact type role contained in the kernel. This conforms to the standard rules of conceptual schema design, as defined in [17].

The definition of $\mathcal{KER}$ removes all objectified relationship types which neither participate in a major fact-type, nor are major types themselves (as shown in Fig. 25, below).

It is interesting to compare the differences in the way objectified fact types and co-referenced object types are treated. Consider the examples shown in Fig. 26.

In Fig. 26(a), the objectified fact type (identified by the participating A and B) is included in the next higher level of abstraction, because it is considered to be a *major fact type*. In Fig. 26(b), however, the co-referenced object type (AB) is not included in the next higher level of abstraction, because it does not participate in any fact types at this level. AB may, however, be added to $\mathcal{CS}_{i+1}$ if it is required for connectivity. We justify the difference in treatment of objectified fact types and co-referenced object types by the observation that an objectified fact type can, itself be thought of as a type of abstraction on a co-referenced fact type, which must, itself be 'unwrapped' [9].

4.3. Ring fact types

At this stage, the kernel only contains fact types which bridge between subtyping hierarchies. The kernel does not retain those (ring) fact types for which every participant is a type-related object type because we do not want these fact types to perpetually cause their player to be an anchor point. The user, however, is probably interested in viewing all fact types which are played entirely by major object types at the previous level. This includes the ring fact types. To this end, we therefore apply the procedure **AddRingFTs** to the types in the kernel.

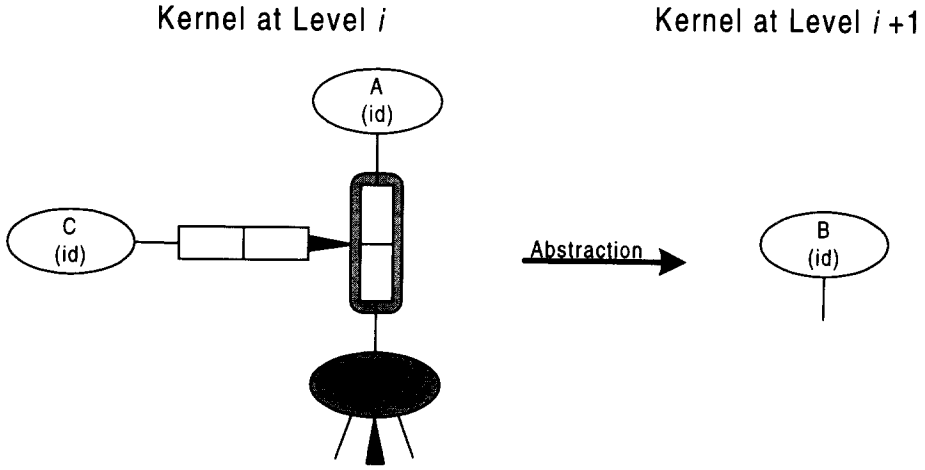


Fig. 25. Removal of objectified relationship types.

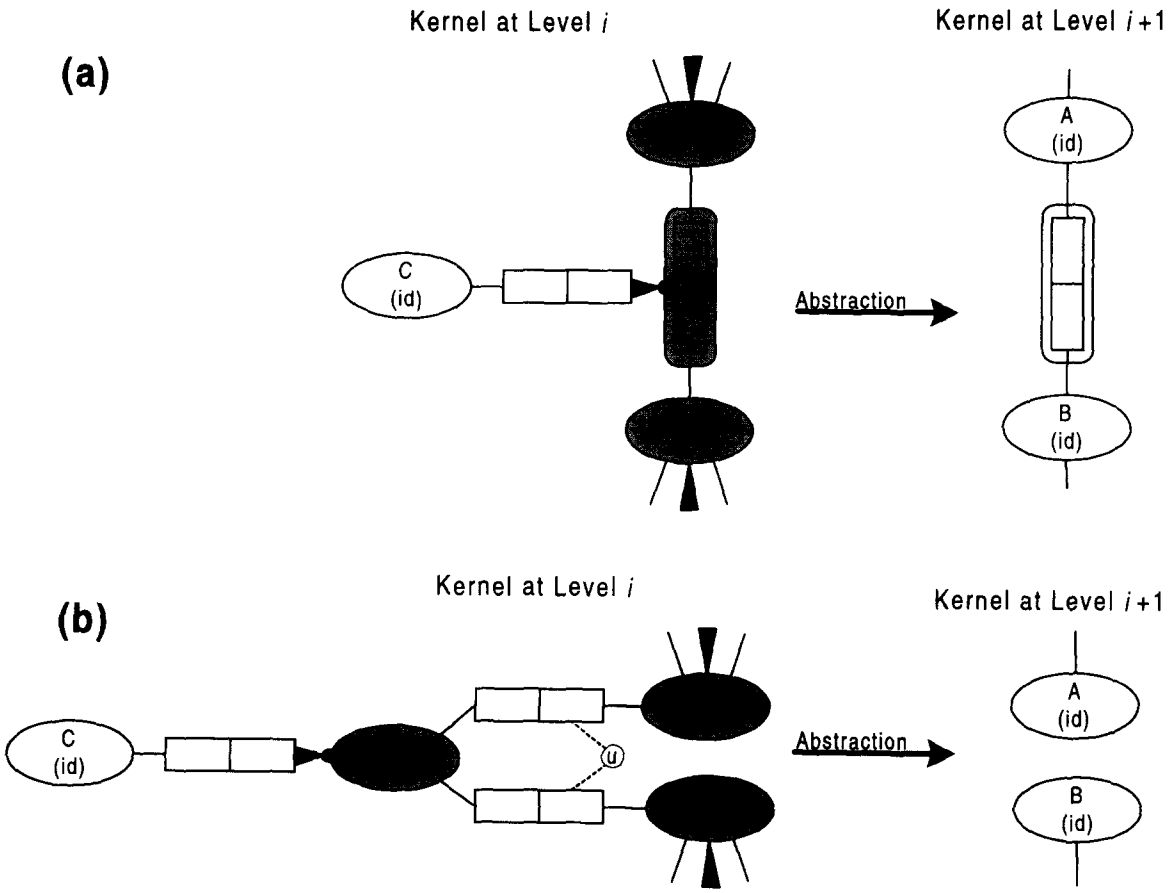


Fig. 26. (a) Objectified fact type; (b) co-referenced object type.

AddRingFTs: $\wp(\mathcal{OB}) \rightarrow \wp(\mathcal{OB})$

AddRingFTs(*Types*)

BEGIN

RETURN $Types \cup \{r \in \mathcal{FT} \mid \forall_{s,t \in \mathsf{Player}(r)}[s, t \in Types \wedge s \sim t]\}$;

ENDAddRingFTs ;

4.4. *Object type identification schemes*

Since the identification scheme of an object type is often important for its understanding, we ensure that the identification scheme of every object type is included in each abstraction level. To this end, we define that relation, $\mathsf{IsIdRel} \subseteq \mathcal{RL} \times \wp(\mathcal{TP})$, which is true when the given relationship type is involved the primary identification of some object type in the given set of types.

$$\mathsf{IsIdRel}(r, z) \triangleq \exists_{x \in z}[r \in \mathsf{PIdRels}(x) - z]$$

The function IdentifiedSchema: $\wp(\mathcal{TP}) \rightarrow \wp(\mathcal{TP})$ takes the types in the current abstraction level and adds to them, those types that are required to identify the input types.

```
IdentifiedSchema(Types)
VAR
  r : FT ;
BEGIN
  WHILE ∃r[r ∉ Types ∧ IsIdRel(r, Types)] DO
    LET r BE SUCH THAT r ∉ Types ∧ IsIdRel(r, Types) ;
    Types += {r} ∪ {Player(s) | s ∈ Roles(r)} ;
  END WHILE ;
  RETURN Types ;
END IdentifiedSchema .
```

It is interesting to consider the effect that this algorithm has on the subtyping hierarchies of the abstracted schema. In Fig. 27, the only major object types in $\mathcal{NER}_i$, which participate in a major fact type, are C and F. C and F are therefore the only object types to appear in $\mathcal{NER}_{i+1}$. Since C inherits its identification from its (indirect) supertype A, however, A is included in IdentifiedSchema($\mathcal{NER}_{i+1}$) as the player of C's identifying relationship type.

Notice that the subtyping arrows in the various schema fragments adapt automatically to the set of object types included in the diagram. This is possible because of the fact that subtyping relationships are inherently transitive, with only the non-implied arrows being displayed on the diagram.

4.5. Connectivity

Since we wish to retain the connectivity of our conceptual schema throughout each level of abstraction, we must define the concept of connectivity. We begin by defining a connected path ($\mathcal{PA}$) through a conceptual schema. A path is a sequence of types in which each element (except the first) is either a relationship type of which the previous conceptual type is a player, or is one of the object types which plays the previous conceptual type. Note that a path does not necessarily define a unique traversal through a relationship type, since an object type may play more than one role in the same relationship type.

$$\mathcal{PA} \triangleq \{x \in \mathcal{TP}^+ \mid \forall_{1 \leq i < |x|}[\mathsf{Connected}(x[i], x[i+1])]\}$$

where Connected identifies whether or not the given conceptual types are connected (in that one participates in the other):

$$\mathsf{Connected}(y, z) \triangleq \exists_{p \in \mathsf{Roles}(y)}[\mathsf{Player}(p) \sim z] \vee \exists_{p \in \mathsf{Roles}(z)}[\mathsf{Player}(p) \sim y]$$

The predicate PathBetween $\subseteq \mathcal{PA} \times \mathcal{TP} \times \mathcal{TP}$, holds exactly when the given path exists, starting at conceptual type x and ending at conceptual type y.

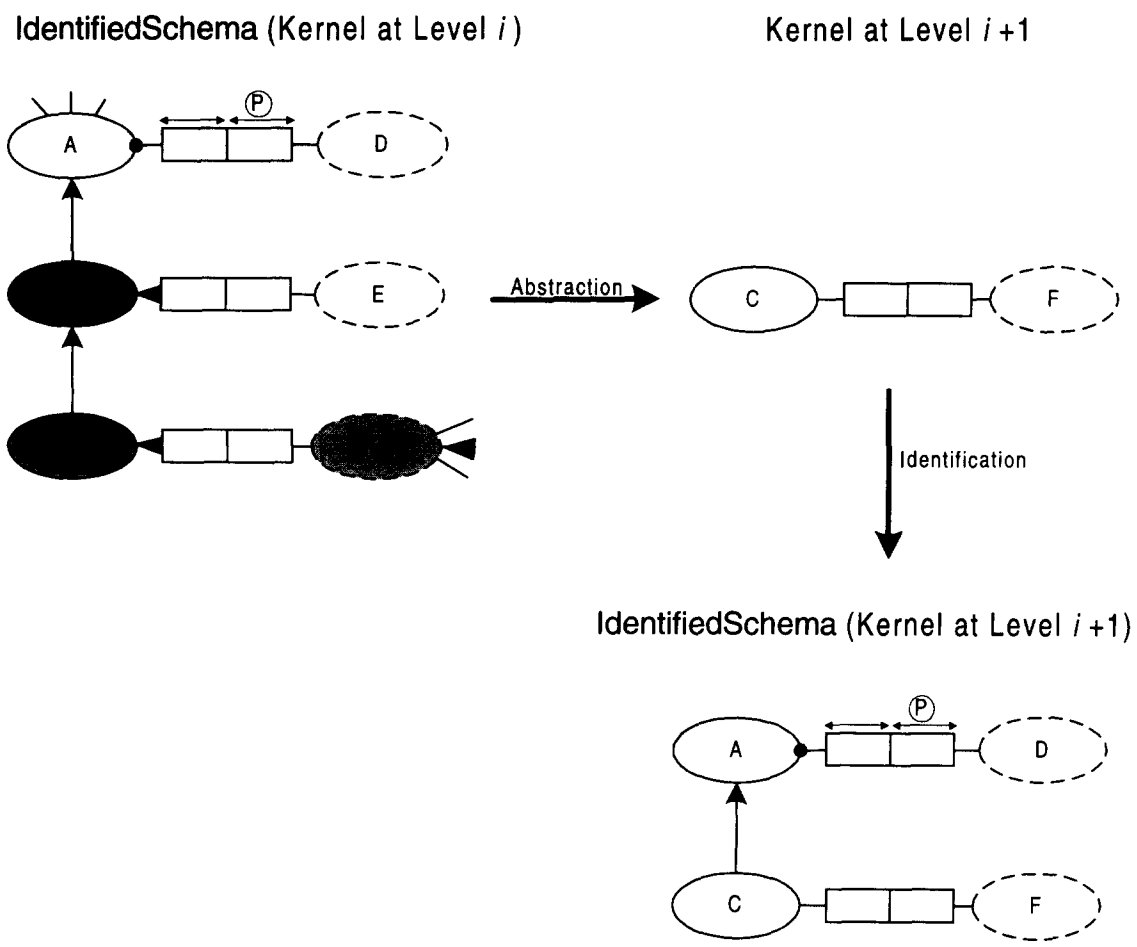


Fig. 27. The primary identification of subtype C is inherited from supertype A.

$$\mathsf{PathBetween}(v, x, y) \stackrel{\Delta}{=} v[1] = x \wedge v[|v|] = y$$

ShortestPaths(x, y) returns the set of paths which start at type x and end at type y, which contain the least number of conceptual types in between (ShortestPaths: $\mathcal{TP} \times \mathcal{TP} \rightarrow \wp(\mathcal{PA})$).

$$\mathsf{ShortestPaths}(x, y) \stackrel{\Delta}{=} \{v \in \mathcal{PA} \mid \mathsf{PathBetween}(v, x, y) \wedge |v| = \min\{|w| \mid \mathsf{PathBetween}(w, x, y)\}\}$$

Another type of connectivity which is useful to maintain is the connectivity of subtyping hierarchies. To this end, each set of unconnected, type-related types in IdentifiedSchema($\mathcal{HCR}_{i+1}$ are reconnected via the lowest common supertype which has its own identification scheme. The notion of a *lowest common identified supertype* is therefore introduced. A set of object types can actually have more than one *lowest common identified*

supertype, in the case in which the subtyping hierarchy forms a lattice. The function LowestCIdObjs: $\wp(\mathcal{TP}) \rightarrow \wp(\mathcal{TP})$ is defined on a set of type-related object types as:

$$\text{LowestCIdObjs}(w) \triangleq \{x \in \text{CommonIDSup}(w) \mid \neg\exists_{y \in \text{CommonIDSup}(w)}[y \text{ SubOf } x]\}$$

where $\text{CommonIDSup}(w) \triangleq \text{CommonSup}(w) \cap \text{Players}(\text{OtherRoles}(\text{PIdRoles}(w)))$. Notice that the only case in which the *lowest common identified supertype* for a set of object types will be the same as the *lowest common supertype*, defined previously, is when the lowest common supertype has an identification scheme directly attached to it.

The function ConnectSchema is used to add those types which are required to connect the given set of types by means of subtyping hierarchies and shortest paths.

ConnectSchema: $\wp(\mathcal{TP}) \rightarrow \wp(\mathcal{TP})$

ConnectSchema(*Types*)

PostCondition: $\forall_{x,y \in Types}[\text{PathBetween}(x, y)]$

BEGIN

1. WHILE $\exists_{x,y \in Types}[x \sim y \wedge \neg\exists_{z \in Types}[z \in \text{CommonSup}(\{x, y\})]]$ DO

Types += LowestCIdObs($\{s \mid s \sim x\}$) ;

2. WHILE $\exists_{x,y \in Types}[\neg\text{PathBetween}(x, y)]$ DO

Types += $\bigcup_{c \in \text{ShortestPath}(x, y)} \text{Set}(c)$;

Types := IdentifiedSchema(*Types*) ;

END 2 ;

END 1 ;

RETURN *Types* ;

END ConnectSchema ;

The first loop (marked 1.) ensures that all type related object types are connected via a subtyping hierarchy in the abstraction schema. The second loop (marked 2.) ensures that the non-type related object types are connected via relationship types in the resulting set of types. Notice that IdentifiedSchema is reapplied during ConnectSchema to ensure that any newly added types are also identifiable in the abstraction schema.

Fig. 28 shows an example in which IdentifiedSchema($\mathcal{HER}_{i+1}$) is disconnected, despite the fact that its components contain type related object types. In order to connect the two schema fragments in IdentifiedSchema($\mathcal{HER}_{i+1}$), ConnectSchema adds A to the set of included types. Object type A represents the lowest common supertype of G and C which has its own identification scheme. In this case, C and G have their own identification scheme. However, in order to retain the notion that the instances of C and G come from a common domain, we include in the abstraction the lowest common supertype which is directly attached to a unique identification scheme for this domain.

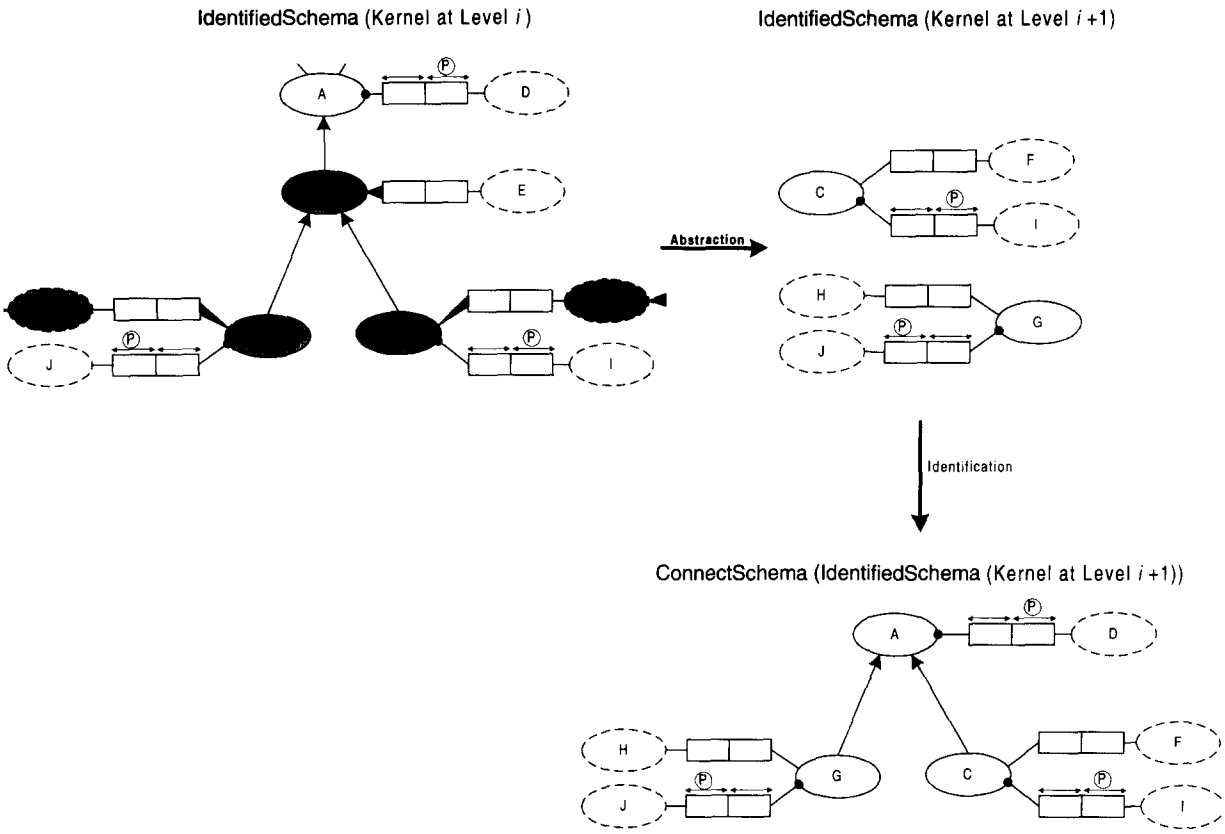


Fig. 28. Connecting the kernel by means of the subtype hierarchy.

4.6. Complete abstraction of conceptual schema

The complete set of abstraction levels for a given conceptual schema, $\mathscr{CS}$, can now be defined. The level i abstraction view, $\mathscr{CS}_i$, of a given conceptual schema, $\mathscr{CS}$, includes the types in $\mathscr{KER}_i$, the types required for connectivity and identification, and all constraints from the original schema in which these included types are involved. We, therefore, formally define $\mathscr{CS}_i$ as follows:

$$\mathscr{CS}_1 \triangleq \langle \mathscr{IS}, \mathsf{Mand}, \mathsf{Unique}, \mathsf{PUnique}, \mathsf{Frequency}, \mathsf{SetCon}, \mathsf{Weight} \rangle$$

$$\mathscr{CS}_i \triangleq \langle \mathscr{IS}_i, \mathsf{Mand}_i, \mathsf{Unique}_i, \mathsf{PUnique}_i, \mathsf{Frequency}_i, \mathsf{SetCop}_i, \mathsf{Weight}_i \rangle \text{for } i > 1$$

where

$$\mathscr{IS}_i \triangleq \langle \mathscr{TP}_i, \mathscr{RO}_i, \mathsf{Roles}_i, \mathsf{PosN}, \mathsf{SubOf}_i, \mathsf{Player} \rangle$$

$$\mathscr{TP}_i \triangleq \mathsf{ConnectSchema}(\mathsf{IdentifiedSchema}(\mathsf{AddRingFTs}(\mathscr{KER}_i)))$$

$$\mathscr{RO}_i \triangleq \{p \in \mathscr{RO} \mid \mathsf{Rel}(p) \in \mathscr{TP}_i\}$$

$$\mathsf{SubOf}_i \triangleq \{\langle x, y\rangle \in \mathcal{TP}_i \times \mathcal{TP}_i \mid x\ \mathsf{SubOf}_{i-1}\ y\}$$

$$\mathsf{Mand}_i \triangleq \{x \in \wp(\mathcal{RC}_i) \mid \mathsf{Mand}(x) \wedge \exists_{p \in \mathcal{RC}_i} \forall_{q \in x}[p \sim q \wedge p \notin x]\}$$

$$\mathsf{Unique}_i \triangleq \{x \in \wp(\mathcal{RC}_i) \mid \mathsf{Unique}(x)\}$$

$$\mathsf{Frequency}_i \triangleq \{x \in \wp(\mathcal{RC}_i) \mid \mathsf{Frequency}(x)\}$$

$$\mathsf{SetCon}_i \triangleq \{\langle v, w\rangle \in \mathcal{RC}_i^+ \times \mathcal{RC}_i^+ \mid \mathsf{SetCon}(v, w)\}$$

4.7. Relation to clusters

The definitions so far allow us to take a flat ORM conceptual schema and derive a number of abstraction layers for this flat ORM schema. Each of these layers is still essentially a flat subsection of the original ORM schema. Therefore, we now introduce the glue that actually holds these layers of abstraction together.

The idea is to view each major object type as becoming the centre for a clustering of surrounding minor object types. As a result, the object types in each $\mathcal{KER}_i$ are clusterings of types from $\mathcal{KER}_{i-1}$. This idea of using clustering as a binding mechanism for abstraction layers for ORM schemas was proposed previously in [5–7]. In [9] a possible formalization of the clustering mechanism is presented.

In this subsection, we show how to derive a clustering of minor object types for each major object type. The presented style of clustering conforms to the requirements given in [9]. This means that when applying the abstraction algorithm discussed in this article, together with the clustering mechanism presented below, a three-dimensional ORM schema results that is in line with the 3-Dimensional Conceptual Modeling Kernel as proposed in [9].

The clustering mechanism is defined as a set of derivation rules. An actual clustering is given as a function $\mathsf{Cluster}: \mathbb{N} \times \mathcal{OB} \rightarrow \wp(\mathcal{TP})$. The intuition is that if $x \in \mathsf{Cluster}(i, y)$, then at abstraction level i type x has been grouped into the cluster surrounding object type y.

The first derivation rule clusters all fact types which have disappeared since the last level of abstraction towards the object type to which they were anchored.

$$\mathsf{Anchor}_{i-1}(r) \wedge \mathsf{Rel}(r) \in \mathcal{KER}'_i \vdash \mathsf{Rel}(r) \in \mathsf{Cluster}(i, \mathsf{Player}(r)) \qquad \textbf{[CL1]}$$

where $\mathcal{KER}'_i \triangleq \mathcal{KER}_{i-1} - \mathcal{KER}_i$ for $i > 1$.

Object types which participate in any relationship type included in a cluster should, obviously, be included in the same cluster:

$$x \in \mathsf{Cluster}(i, c) \wedge y \in \mathsf{Players}(x) \vdash y \in \mathsf{Cluster}(i, c) \qquad \textbf{[CL2]}$$

Please note that an object type could occur in more than one clustering if it is involved in relationship types anchored to different major object types. As a result, the clustering is not a partition of the types.

The following two rules are concerned with subtyping. If an object type in a type hierarchy is removed from the kernel (i.e. it is in $\mathcal{KER}'_i$), we must still cluster those fact types that were anchored to it. We anchor such fact types towards the lowest supertype which remains in the kernel (LowestKernelSup).

$$x \in \mathcal{KER}'_i \wedge y \ \mathsf{LowestKernelSup}\ x \wedge r \ \mathsf{AnchoredTo}_{i-1} x \vdash \mathsf{Rel}(r) \in \mathsf{Cluster}(i, y) \qquad \textbf{[CL3]}$$

where y LowestKernelSup x indicates that y is the lowest supertype of x which remains in the abstraction kernel. That is:

$$y \ \mathsf{LowestKernelSup}\ x \triangleq x \ \mathsf{SubOf}\ y \wedge y \in \mathcal{KER}_i \wedge \neg\exists_{z \in \mathcal{KER}_i}[x \ \mathsf{SubOf} z \ \mathsf{SubOf} y]$$

If no supertype remains in the kernel, however, the relationship types anchored towards a disappearing subtype can be clustered towards the nearest subtype which remains in the kernel.

$$x \in \mathcal{KER}'_i \wedge \neg\exists_z[z \ \mathsf{LowestKernelSup}\ x] \wedge y \ \mathsf{HighestKernelSub}\ x$$
$$\wedge r \ \mathsf{AnchoredTo}_{i-1} x \vdash \mathsf{Rel}(r) \in \mathsf{Cluster}(i, y) \qquad \textbf{[CL4]}$$

where y HighestKernelSub x indicates that y is the highest subtype of x which remains in the abstraction kernel. That is:

$$y \ \mathsf{HighestKernelSub}\ x \triangleq y \ \mathsf{SubOf}\ x \wedge y \in \mathcal{KER}_i \wedge \neg\exists_{z \in \mathcal{KER}_i}[y \ \mathsf{SubOf}\ z \ \mathsf{SubOf}\ x]$$

The remaining derivation rules are completeness rules on clusters. Clustered types are inherited between layers of abstraction. So we have:

$$x \in \mathsf{Cluster}(i, c) \vdash x \in \mathsf{Cluster}(i + 1, c) \qquad \textbf{[CL5]}$$

The reference types needed to identify any of the types in a cluster are also included:

$$x \in \mathsf{Cluster}(i, c) \wedge y \in \mathsf{PIdRels}(x) \vdash y \in \mathsf{Cluster}(i, c) \qquad \textbf{[CL6]}$$

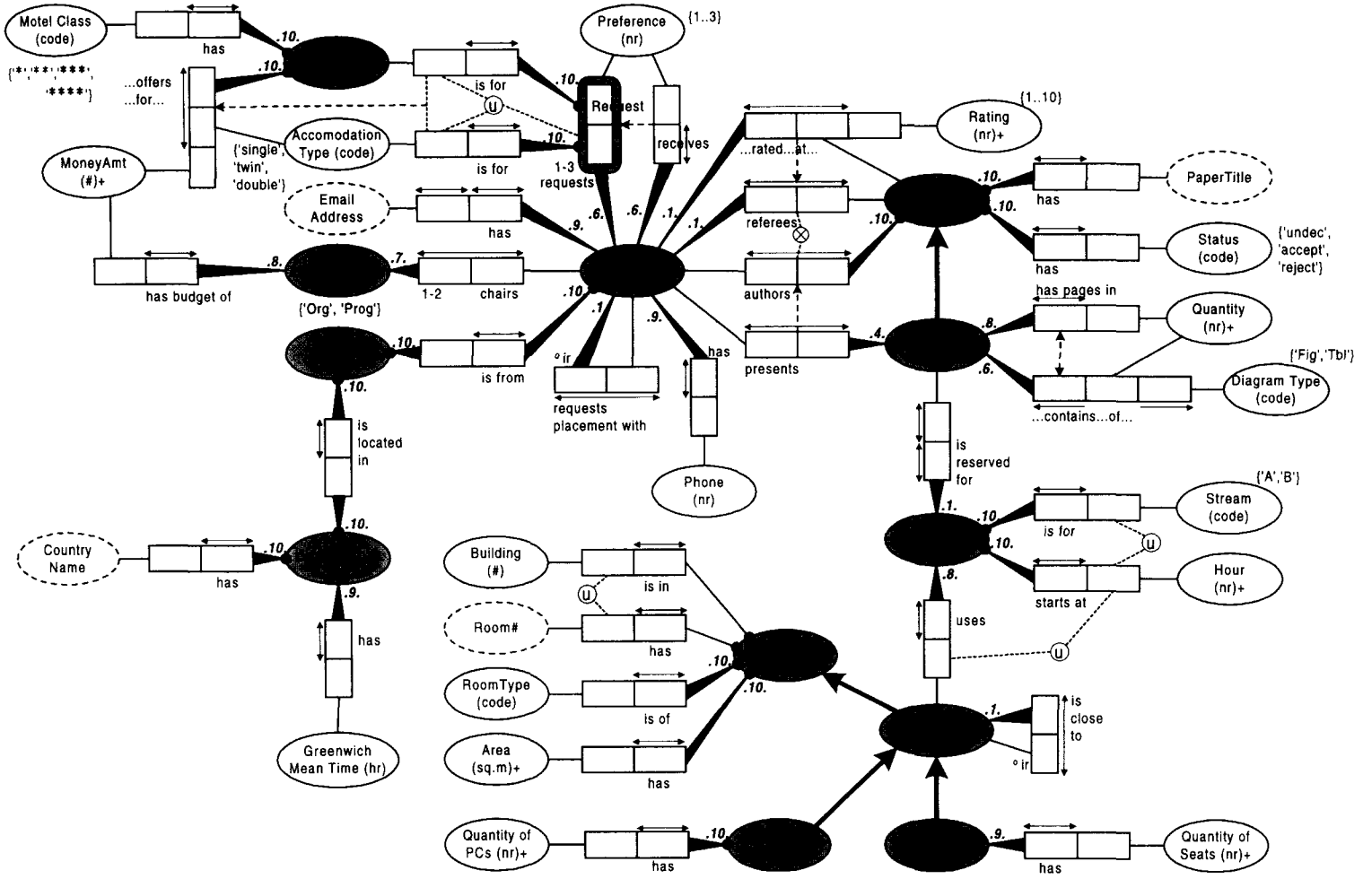
The above definition of cluster is a 'maximally complete' one. However, when displaying clusters to a user, for example, one may choose to only show those clustered types which are part of $\mathcal{KER}'_i$. That is:

$$\mathsf{Cluster}(i, x) - \mathsf{Cluster}(i - 1, x)$$

It may also be decided to only show the clusters for those types which appear in $\mathcal{KER}_i$, and ignore the clusters for those object types which were major at the previous level, but do not participate in a fact types in $\mathcal{KER}_i$. Choices like this are up to the designer of the actual abstraction tool and often depend solely upon the purpose for which the abstraction and clustering was created.

5. Case study

Now that we have developed a theory for the creation of abstractions for a conceptual schema, it is time to study the effect that such a mechanism has on an application example. Applying WeightSchema to the conceptual schema shown in Fig. 6, we achieve the anchored schema shown in Fig. 29. As in previous examples, we have shaded the major object types and indicated the anchors by an arrowed role connector line. For the purposes of our example application, we have also included the Weight assigned to each anchor (e.g. ".9."). This will



SUBTYPE DEFINITIONS

each Accepted Paper **is a** Paper **that** has Status 'accept'
each LabOrLecRoom **is a** Room **that** is of RoomType in {'lab','lec'}
each Laboratory **is a** Room **that** is of RoomType 'lab'
each Lecture Room **is a** Room **that** is of RoomType 'lec'

Fig. 29. WeightSchema($\mathscr{CS}_1$): Abstraction level 1.

hopefully help the reader to retrace the AutoWeight rules that have been fired to achieve this result.

Applying WeightSchema to our application example of Fig. 6 helps us identify the major object types in the Universe of Discourse. In this case, they are 'Motel', 'Committee', 'Institution', 'Country', 'Request', 'Person', 'Paper', 'Accepted Paper', 'Paper Slot', 'Room', 'Lab or Lecture Room', 'Lecture Room' and 'Laboratory'. This follows our own intuition of the most 'conceptually important object types'.

It is important to notice that the relationship types '*Room* is in *Building*' and '*Room* has *Room#*' are not anchored. This is because they are part of the primary identification scheme for 'Room', and are therefore reference types. Only fact types are anchored.

Among the anchored fact types are '*Person* chairs *Committee*', which is anchored towards 'Committee'. The maximum frequency of 2 on the role played by 'Committee' causes AutoWeight rule 4 to assign a Weight of 7 to this role. AutoWeight rule 8 is responsible for anchoring the fact type '*Person* presents *Accepted Paper*' towards 'Accepted Paper', due to the fact that it is associated via a set constraint to a fact type already anchored towards 'Paper'.

Fig. 30 shows the kernel types which form the foundation of the second level of abstraction for our example conceptual schema. Notice that no ring fact types or identification schemas are included in the kernel, and that the kernel is actually disconnected.

When we apply AddRingFTs, IdentifiedSchema and ConnectSchema to the kernel types in Fig. 30, add the constraints that are still relevant and re-apply WeightSchema, we achieve the complete weighted, second level abstraction schema shown in Fig. 31.

Notice that the major object types of $\mathscr{CS}_2$ are 'Request', 'Person', 'Institution', 'Paper', 'Accepted Paper' and 'Paper Slot'. Because these object types are major at both the first and second level of abstraction, we consider them to be more 'conceptually important' than those object types which are only major at the first level of abstraction. In fact, we gauge an object type's degree of majorness (DegreeMajor: $\mathscr{OB} \rightarrow \mathbb{N}$) by calculating the highest level of abstraction at which that object type is major. We define:

$$\mathsf{DegreeMajor}(x) \triangleq \mathsf{max}(\{i \mid \mathsf{MajorOT}_i(x)\} \cup \{0\})$$

and we know that:

$$\mathsf{MajorOT}_i(x) \Rightarrow \mathsf{DegreeMajor}(x) \geq i$$

For example, so far we know that:

$$\mathsf{DegreeMajor}(\text{'}\textit{Rating}\text{'}) = 0\ ;\ \mathsf{DegreeMajor}(\text{'}\textit{Motel}\text{'}) = 1 \text{ and } \mathsf{DegreeMajor}(\text{'}\textit{Person}\text{'}) \geq 2$$

Conceptual importance, or conceptual relevance (as indicated by DegreeMajor) plays a key role in a number of areas. For example, in computer supported query formulation conceptual importance is used to help select between alternative interpretations of queries ([4, 21, 22]).

It is important to understand how the schema abstraction in Fig. 30 was obtained. The object types 'Room', 'Building', 'Room#' and 'Preference' were all added by the IdentifiedSchema procedure, because they are used in the identification of some kernel object type. The fact types '*Lab or Lecture Room* is close to *Lab or Lecture Room*' and '*Person*

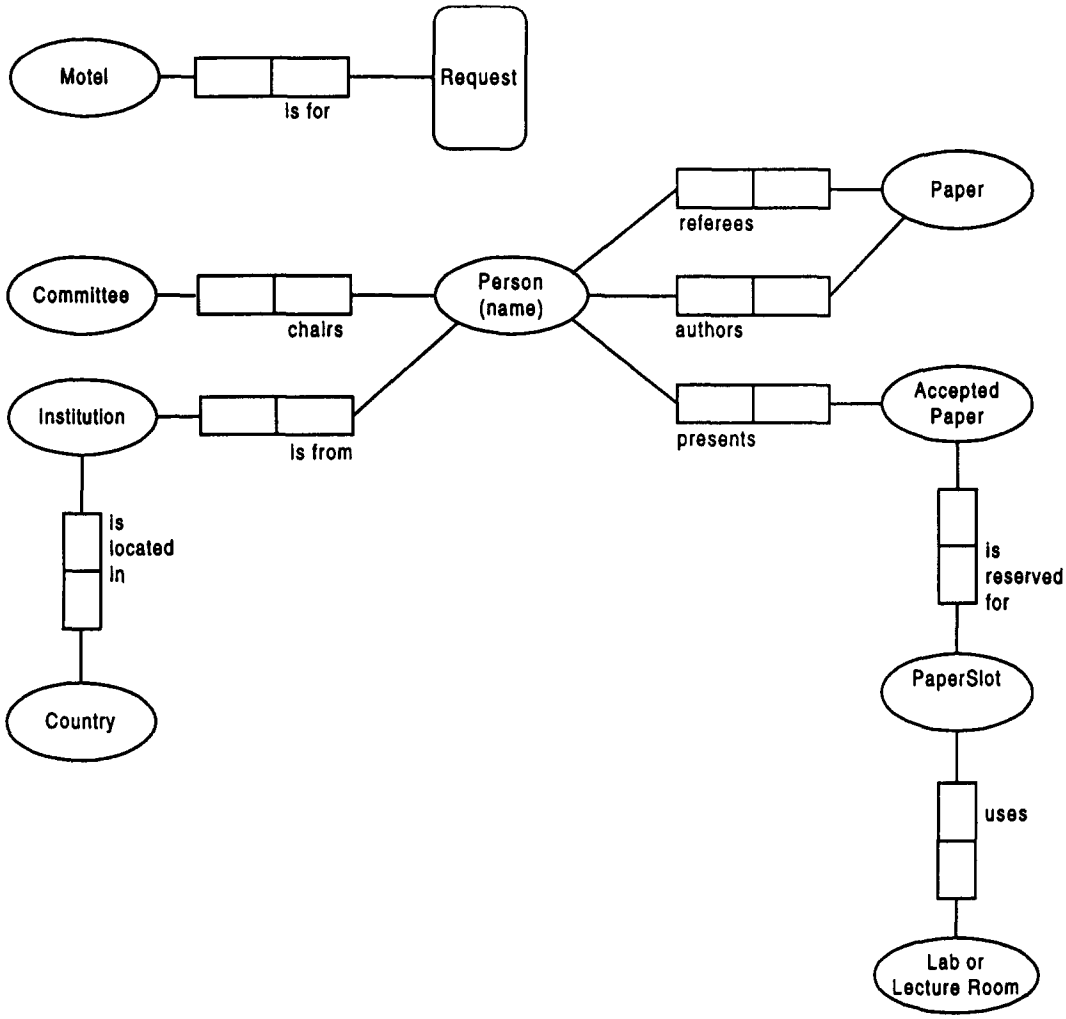


Fig. 30. $\mathcal{KER}_2$: The kernel for abstraction level 2.

requests placement with *Person*' were added during AddRingFTs, and are not actually part of $\mathcal{KER}_2$. This explains why we do not consider these fact types to be anchored.

There are a few interesting things to observe with respect to the anchorage of $\mathcal{CS}_2$. Firstly, notice that '*Institution* is located in *Country*', in contrast to $\mathcal{CS}_1$, is not anchored towards 'Country'. This is because at abstraction level 2, this role becomes implied mandatory. Secondly, the anchor on '*Person* chairs *Committee*' was on the role played by 'Committee' in $\mathcal{CS}_1$, but has now moved to the role played by 'Person'. This is because 'Committee' has become a leaf object type, causing the role played by 'Person' to gain a new weight of .9.. The fact type '*Person* referees *Paper*' also has a change in anchorage. In the previous level of abstraction, it was anchored by Rule 12. Since it is now only associated with a single set constraint, however, Rule 8 now triggers a weight of .4. on the role played by Paper. Lastly, notice that the weight of the anchor on '*PaperSlot* uses *Lab or Lecture Room*' has increased from .8. to .9. because 'Lab or Lecture Room' is now a leaf object type, triggering rule 3.

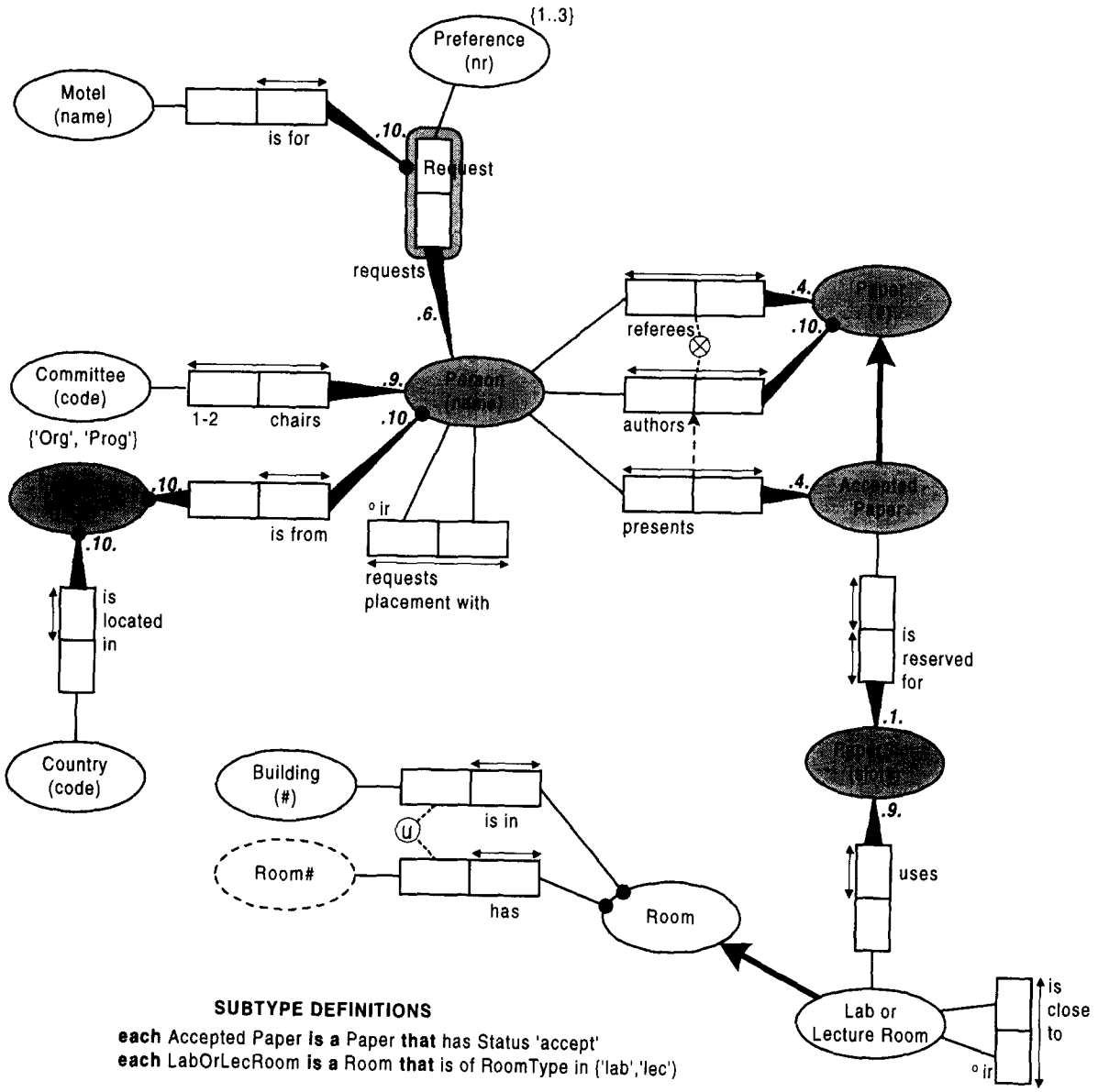


Fig. 31. WeightSchema($\mathscr{GS}_2$): Abstraction level 2.

Fig. 32 depicts a different view of our second level abstraction. In Fig. 32, we explicitly show the clusterings that have occurred during the abstraction process. We have chosen to represent those object types which are repeated in more than one cluster by surrounding them with a second ellipse; and have shown only those constraints which are completely within or completely external to a clustering. It is particularly interesting to observe the subtype clustering that has occurred around the object type 'Lab or Lecture Room'.

Taking things one step further, we can easily extend our results from Figs. 31 and 32 to

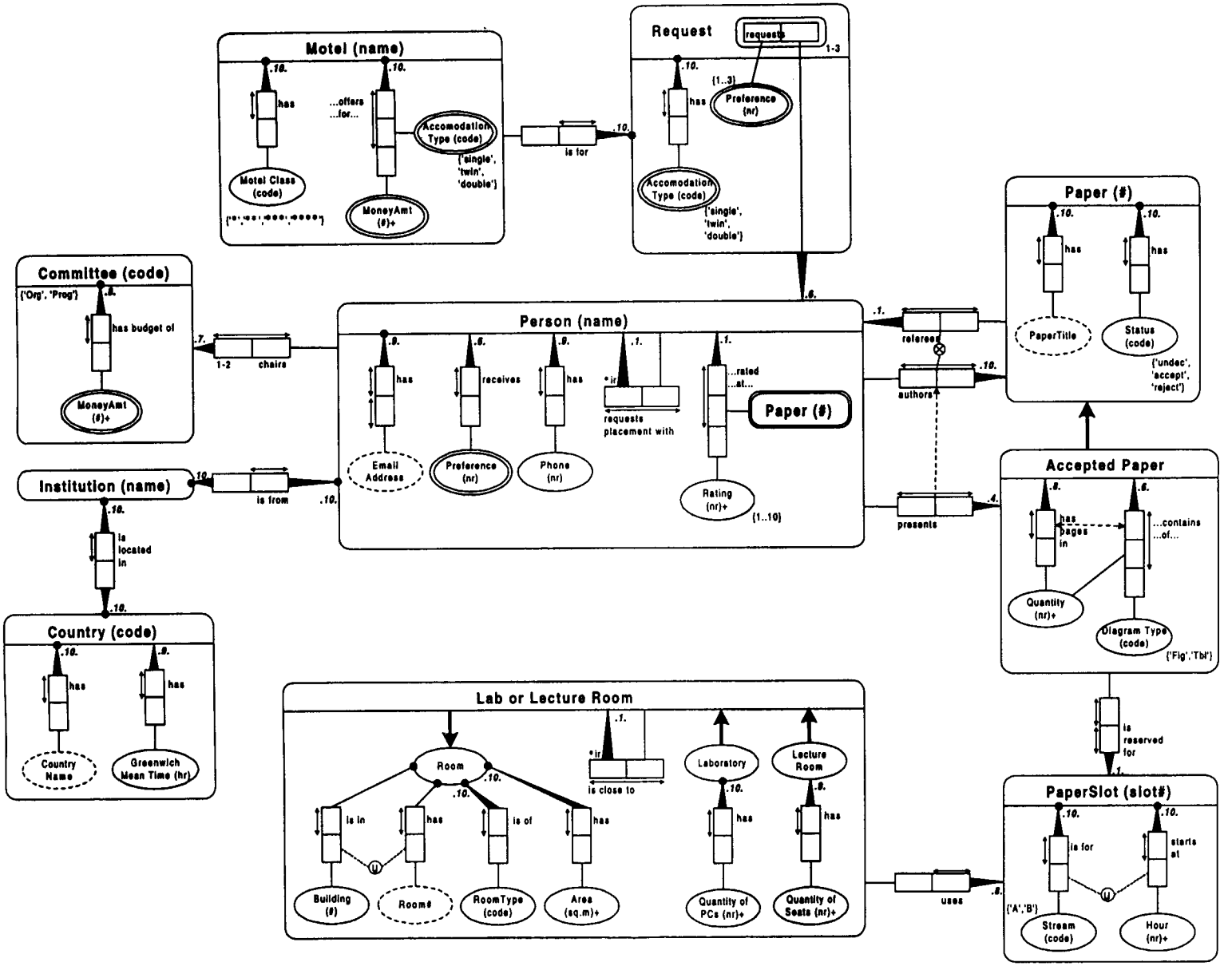


Fig. 32. Abstraction level 2 showing the clusterings that have occurred.

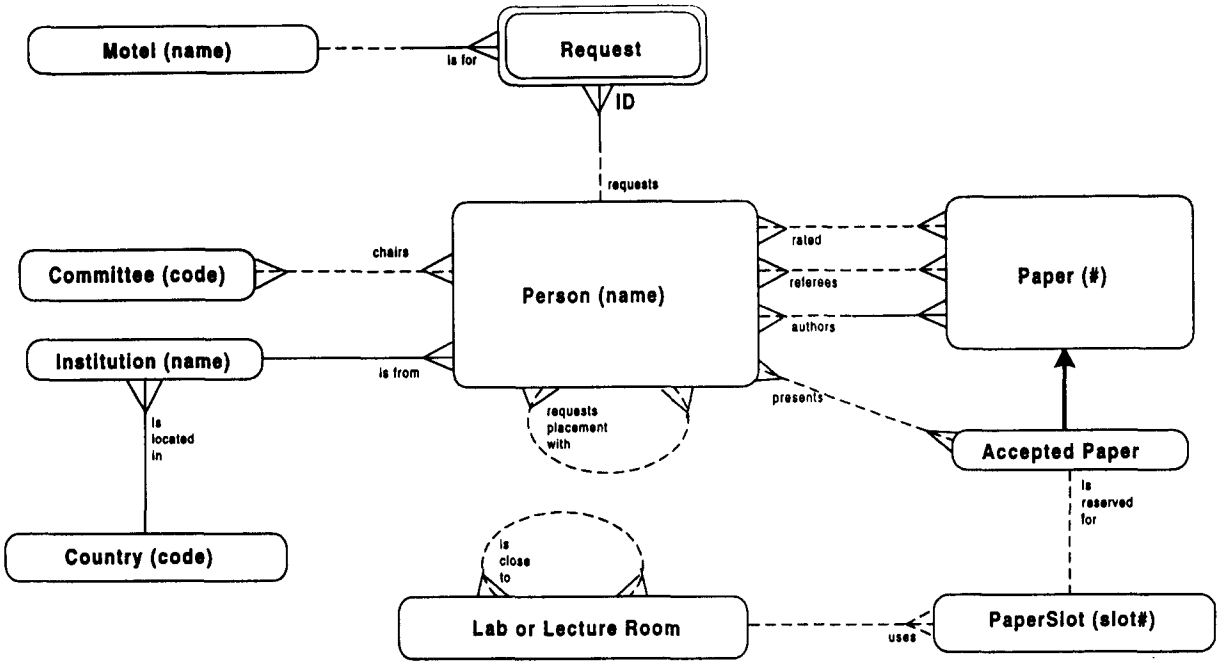


Fig. 33. An ER View of the example application.

show a corresponding Entity Relationship (ER) representation of the application. Fig. 33 shows this ER view. There are many notations used for ER modeling. The one presented here uses rounded rectangles to represent entities, named lines to represent relationship types, crow feet to indicate that the opposite entity can play that role many times, a double rectangle to represent a 'weak entity type', and the letters 'ID' placed on its identifying relationship. Attributes are not shown in this diagram.

It is important to realize that the version of ER used above allows multi-valued and composite attributes. Making this assumption allows us to achieve an intuitive overview of the original ORM diagram using ER notation. We also allow relationships to have attributes. For example, the relationship '*Person* rated *Paper*' has the attribute 'Rating'. Notice that relationships, such as this one, which have attributes do not appear in the ORM abstraction since some of their participants are minor. Also note that 'Request' is represented as a weak entity because its identification scheme involves both an attribute ('Preference') and a relationship to an entity ('Person').

We now take the second level abstraction shown in Fig. 31 and abstract again. Fig. 34 illustrates the third level of abstraction, $\mathscr{CS}_3$. When we apply WeightSchema to $\mathscr{CS}_3$, the only changes in anchorage that can be seen from the previous abstraction are in '*Paper Slot* is reserved for *Accepted Paper*' (as 'Paper Slot' becomes a leaf) and '*Person* is from *Institution*' (as the role played by 'Institution' becomes implied mandatory). We can now determine that the degree of majorness for 'Institution' and 'Request' is 2, and for 'Person', 'Paper', 'Accepted Paper' and 'Paper Slot' is greater than, or equal to three.

The highest level of abstraction that can be reached for our example application is four. Fig. 35 shows $\mathscr{CS}_4$. No higher level of abstraction can be reached, because there are no major fact

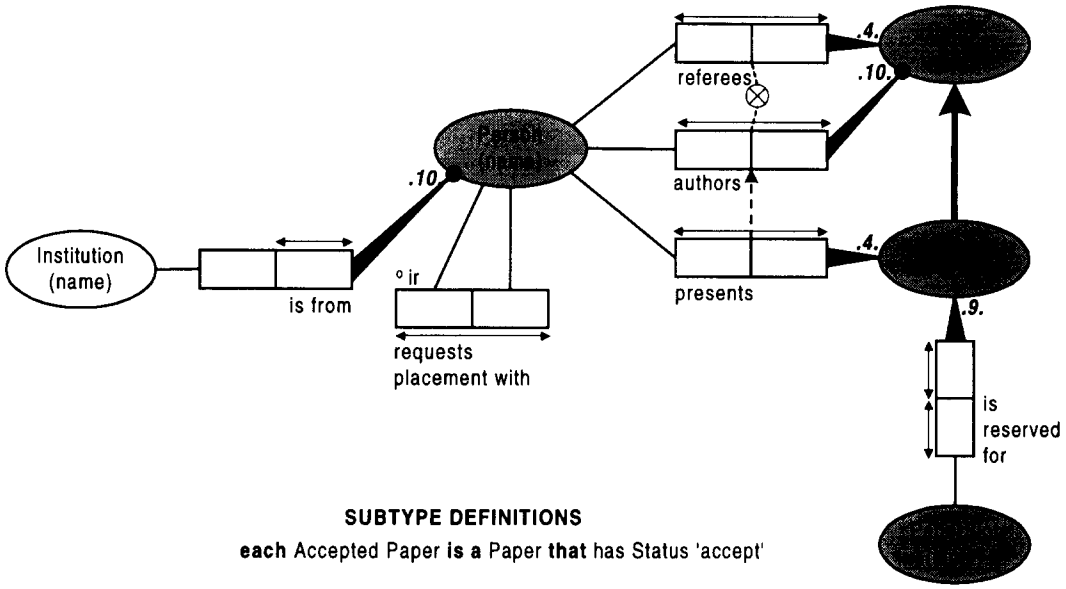


Fig. 34. WeightSchema($\mathscr{CS}_3$): Abstraction level 3.

types in $\mathscr{CS}_4$, and therefore no conceptual types would be present at a higher level of abstraction. We can now conclude that the most conceptually important object type in our Universe of Discourse is 'Paper' (and 'Accepted Paper'), with a degree of majorness equal to four.

6. Conclusions

In this article, we have presented an algorithm to derive layers of abstraction for a given flat conceptual schema. The cornerstone of this abstraction algorithm is the notion of a major

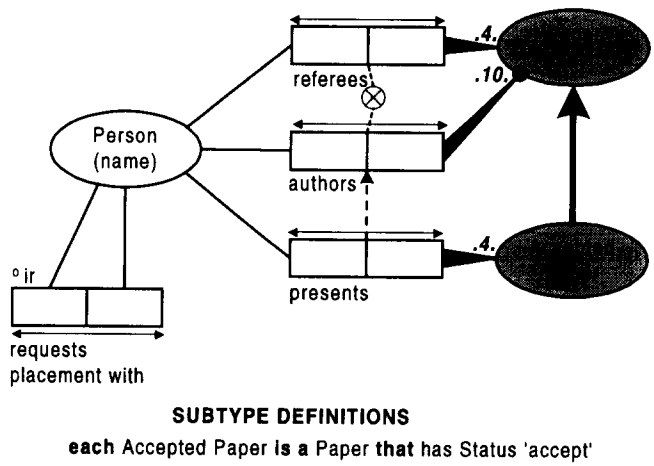


Fig. 35. WeightSchema($\mathscr{CS}_4$): Abstraction level 4.

object type. We have defined a prioritized set of derivation rules to assist in the selection of these major object types at each level of abstraction. In comparison to other approaches which determine the major object types of a conceptual schema, our approach considers more of the semantics that are hidden in the constraints and verbalizations. Alternative approaches have instead relied more heavily on user input.

The paper presents an iterative method for using the major object types to determine the kernel types in each subsequent abstraction level. This kernel is then embellished with additional conceptual types to enhance its comprehensibility, by incorporating identification schemes, ring predicates and connectivity. The result of these processes, together with the constraints applicable to the included types, form the resulting abstraction schema, which is demonstrated on a concrete case study in Section 5.

In addition to this, we have also shown how the resulting layers of abstraction provide a three dimensional view of the underlying flat conceptual schema, where object types at each level of abstraction can be regarded as clusterings of object types from a lower level of abstraction.

Future plans include the tuning of priorities and weights in the derivation rules, which currently reflect the intuition of the authors, to be based on empirical evidence gained through concrete testing. Because the derivation rule approach taken in this paper allows a very modular approach to be taken in its implementation, the tuning of the priority and weighting values can be performed locally.

Other topics for future research include investigating how bottom up abstraction (as described in this paper) and top down abstraction mechanisms can complement each other in a single conceptual schema design procedure; and investigating how the algorithms presented in this paper impact on other types of conceptual schema abstraction.

Acknowledgments

We would like to thank the anonymous referees for their comments and suggestions, which have led to improvements of the original article. Furthermore, we would particularly like to thank A.H.M. ter Hofstede for his comments on the first drafts of this article.

References

[1] C. Batini, S. Ceri and S.B. Navathe, *Conceptual Database Design – An Entity-Relationship Approach* (Benjamin Cummings, Redwood City, California, 1992).

[2] P. van Bommel, A.H.M. ter Hofstede and Th.P. van der Weide. Semantics and verification of object-role models, *Information Systems* 16(5) (October 1991) 471–495.

[3] G.H.W.M. Bronts, S.J. Brouwer, C.L.J. Martens and H.A. Proper, A unifying object role modelling approach, *Information Systems* 20(3) (1995) 213–235.

[4] C.A.J. Burgers, H.A. Proper and Th.P. van der Weide, An information system organized as stratified hypermedia, in: N. Prakash, ed., *CISMOD94, Int. Conf. on Information Systems and Management of Data*, Madras, India (October 1994) 159–183.

[5] L.J. Campbell, Adding a new dimension to flat conceptual modelling, in: T.A. Halpin and R. Meersman, eds., *Proc. First Int. Conf. on Object-Role Modelling (ORM-1)*, Magnetic Island, Australia (July 1994) 294–309.

[6] L.J. Campbell and T.A. Halpin, Automated support for conceptual to external mapping, in: S. Brinkkemper and F. Harmsen, eds., *Proc. Fourth Workshop on the Next Generation of CASE Tools*, Paris, France (June 1993) 35–51.

[7] L.J. Campbell and T.A. Halpin, Abstraction techniques for conceptual schemas, in: R. Sacks-Davis, ed., *Proc. 5th Australasian Database Conference*, 16 (Global Publications Services, Christchurch, New Zealand, January 1994) 376–388.

[8] C.R. Carlson, W. Ji and A.K. Arora, The nested Entity-Relationship model, in: F.H. Lochovsky, ed., *Proc. Eighth Int. Conf. on Entity-Relationship Approach*, Entity-Relationship Approach to Database Design and Querying (Elsevier Science Publishers, Toronto, Canada, 1990) 43–57.

[9] P.N. Creasy and H.A. Proper, A generic model for 3-dimensional conceptual modelling, Technical Report 342, Submitted for publication, Department of Computer Science, University of Queensland, Australia, July 1995. Electronically available as: http://www.icis.qut.edu.au/~erikp/articles/ConDaMo.ps.Z.

[10] B. Czejdo and D.W. Embley, View specification and manipulation for a semantic data model, *Information Systems* 16(4) (1991) 28–44.

[11] O.M.F. De Troyer, A logical formalization of the binary Object-Role model, in: T.A. Halpin and R. Meersman, eds., *Proc. First Int. Conf. on Object-Role Modelling (ORM-1)* Magnetic Island, Australia (July 1994) 28–44.

[12] R. Elmasri and S.B. Navathe, *Fundamentals of Database Systems*, 2nd edition (Benjamin Cummings, Redwood City, California, 1994).

[13] P. Feldman and D. Miller, Entity model clustering: Structuring a data model by abstractions, *Computer J.* 29(4) (1986) 348–360.

[14] C. Francalanci and B. Pernici, Abstraction levels for Entity-Relationship schemas, in: P. Loucopoulos, eds., *Proc. Fourth Int. Conf. CAiSE'92 on Advanced Information Systems Engineering*, *Lecture Notes in Computer Science*, 593 (Springer-Verlag, UK, 1992) 456–473.

[15] M. Ghandi, E.L. Robertson and D.V. Gucht, Leveled Entity Relationship model, in: P. Loucopoulos, ed., *Proc. Fourth Int. Conf. CAiSE'92 on Advanced Information Systems Engineering*, *Lecture Notes in Computer Science* 593 593 (Springer-Verlag, UK, May 1992) 420–436.

[16] T.A. Halpin, A logical analysis of information systems: Static aspects of the data-oriented perspective, Ph.D. thesis, University of Queensland, Brisbane, Australia, 1989.

[17] T.A. Halpin, *Conceptual Schema and Relational Database Design*, 2nd edition (Prentice-Hall, Sydney, Australia, 1995).

[18] T.A. Halpin and H.A. Proper, Subtyping and polymorphism in Object-Role modelling, *Data & Knowledge Eng.*, 15 (1995) 251–281.

[19] A.H.M. ter Hofstede, Information modelling in data intensive domains. Ph.D. thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[20] A.H.M. ter Hofstede, H.A. Proper and Th.P. van der Weide, Formal definition of a conceptual language for the description and manipulation of information models, *Information Systems* 18(7) (October 1993) 489–523.

[21] A.H.M. ter Hofstede, H.A. Proper and Th.P. van der Weide, Computer supported query formulation in an evolving context, in: R. Sacks-Davis and J. Zobel, eds., *Proc. Sixth Australasian Database Conference, ADC'95*, Volume 17(2) of *Australian Computer Science Communications*, Adelaide, Australia (January 1995) 188–202.

[22] A.H.M. ter Hofstede, H.A. Proper and Th.P. van der Weide, Query formulation as an information retrieval problem, Joint Technical Report UQ-321 and CSI-9502, Submitted for publication, Department of Computer Science, University of Queensland, Australia, and Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, January 1995. Electronically available as: http://www.icis.qut.edu.au/~erikp/articles/CSQF.ps.Z.

[23] A.H.M. ter Hofstede and Th.P. van der Weide, Expressiveness in conceptual data modelling, *Data & Knowledge Eng.* 10(1) (February 1993) 65–100.

[24] A.H.M. ter Hofstede and Th.P. van der Weide, Deriving identity from extensionality, .Technical Report CSI-R9416, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, December 1994. Electronically available as: ftp://ftp.cs.kun.nl/pub/SoftwEng.InfSyst/articles/IdentityExt.ps.Z.

[25] A.H.M. ter Hofstede and Th.P. van der Weide, Fact orientation in complex object role modelling techniques, in: T.A. Halpin and R. Meersman, eds., *Proc. First Int. Conf. on Object-Role Modelling (ORM-1)*, Townsville, Australia (July 1994) 45–59.
[26] S. Huffman and R.V. Zoeller, A rule-based system tool for automated ER model clustering, in: F.H. Lochovsky, ed., *Proc. Eighth Int. Conf. on Entity-Relationship Approach*, Entity-Relationship Approach to Database Design and Querying (Elsevier Science Publishers, Toronto, Canada, 1990) 221–236.
[27] J. Martin, *Strategic Data Planning Methodologies* (Prentice-Hall, Englewood Cliffs, NJ, 1982).
[28] D. Moody, A practical methodology for the representation of enterprise data models, in: *Proc. 2nd Annual Conf. on Information Systems and Database Special Interest Group*, Sydney, Australia, 1991.
[29] A.H. Seltviet, An abstraction-based approach to large-scale information system development, in: C. Rolland, F. Bodart and C. Cauvet, eds., *Proc. Fifth Int. Conf. CAiSE'93 on Advanced Information Systems Engineering*, *Lecture Notes in Computer Science* 685 (Springer-Verlag, 1993, Paris, France).
[30] G.C. Simsion, A structured approach to data modelling, *The Australian Computer J.* 21(3) (August 1989).
[31] 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) (August 1989) 975–987.
[32] D. Vermeir, Semantic hierarchies and abstractions in conceptual schemata, *Information Systems* 8(2) (1983) 117–124.
[33] L.A. Walko, Caves: Visualization and abstraction mechanism for object-oriented databases, in: *Proc. 3rd Australian Database Conference* (World Scientific, Melbourne, Australia, 1992) 10–35.
[34] Th.P. van der Weide, A.H.M. ter Hofstede and P. van Bommel, Uniquest: Determining the semantics of complex uniqueness constraints, *Computer J.*, 35(2) (April 1992) 148–156.

Linda J. Campbell is a Ph.D. student, and member of the Asymetrix Research Laboratory in the Computer Science Department of the University of Queensland, Brisbane, Australia.
She received her Bachelor of Information Technology degree with first class Honours and a University Medal at the University of Queensland in 1992. Miss Campbell is currently finalizing her doctoral thesis entitled "Reverse Engineering from a Relational Database System to a 3 Dimensional Conceptual Schema". She is also involved in both lecturing and tutoring roles at the University.
Her main research interests include data abstraction, reverse engineering, the automation of forward engineering, CASE-tool technology and conceptual modelling.

Terry A. Halpin is a Senior Lecturer, and Director of the Asymetrix Research Laboratory, in the Department of Computer Science at The University of Queensland, Australia. He holds the following degrees from this University: BSc, DipEd, BA, MLitStud, Ph.D. His Masters thesis dealt with computer assisted and automated reasoning in formal logic, and his doctoral thesis provided a formalization of Object-Role Modeling within the context of NIAM. His major research interests include conceptual modeling of information systems, conceptual query languages, schema transformation and mapping, and CASE tool support for information systems engineering.
Dr. Halpin is a member of several technical committees dealing with information systems, and has an extensive publication record. His latest book is the second edition of Conceptual Schema and Relational Database Design, Prentice Hall, Sydney. Currently he is on extended leave from his University position, and is working as Head of Research, Database Products Division, Asymetrix Corporation, Bellevue WA, USA.

H.A. (Erik) Proper is currently a lecturer at the School of Information Systems from the Queensland University of Technology, Brisbane, Australia. He is a member of the Cooperative Information Systems Research Centre from that University. He is also a member of the Distributed Systems Technology Centre (DSTC); one of the Cooperative Research Centres funded by the Australian government, Australian Universities, and a number of multi-national companies.
Dr. Proper received his Master's degree from the University of Nijmegen, the Netherlands in May of 1990, and received his Ph.D. from the same University in April 1994. In his Doctoral thesis he developed a theory for conceptual modelling of evolving application domains, yielding a formal specification of evolving information systems. From May 1994 to September 1995 he worked as a Research Fellow at the Computer Science Department of the University of Queensland, Brisbane, Australia. During that period he also conducted research in the Asymetrix Research Lab at that University for Asymetrix Corp, Bellevue, Washington.
Dr. Proper has co-authored several journal papers and conference publications. His main research interests include resource discovery, conceptual modelling, linguistics, and conceptual query languages.