A Model of Database Components and their Interconnection Based upon Communicating Views

Stephen J. HEGNER[†]
Umeå University
Department of Computing Science
SE-901 87 Umeå, Sweden
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

Abstract A formalism for constructing database schemata from simple *components* is presented in which the components are coupled to one another via communicating views. The emphasis is upon identifying the conditions under which such components can be interconnected in a conflict-free fashion, and a characterization of such, based upon the acyclicity of an underlying hypergraph, is obtained. The work is furthermore oriented towards an understanding of how updates can be supported within the component-based framework, and initial ideas of so-called *canonical liftings* are presented.

1. Introduction

Large systems typically possess a complex structure; database systems are no exception. Schemata with thousands of relations are not uncommon; the largest have tens of thousands. If not designed properly, such systems are certain to be unmanageable intellectually, leading to redundancy, design errors, and difficulties as the schema evolves over time. In many engineering settings, including in particular computer hardware, this problems associated with complexity are addressed, at least in part, by designing a large system as the interconnection of simpler (often prefabricated) components. In the field of logic design, for example, this approach has become completely standard [16]. For a variety of reasons, it has not seen as much success in software engineering [17], and in database systems in particular.

Although the idea that formal objects which describe computations, such as automata, can be the basic modules of an interconnection calculus dates back to relatively early days of theoretical computer science (see, for example, [2, Sec. 3.2]), corresponding ideas have not found great success in application to more concrete problems. To address some of the shortcomings of more classical approaches, Broy [5] [6] has proposed a formal interconnection calculus for components, including software components in particular. His approach has two flavors. The first is based upon input and output *streams* with otherwise stateless components; the state being effectively recaptured in the stream. The second is based upon a more conventional state-transition approach.

Thalheim [24] [22] [25] has recently forwarded the idea of basing database design upon components. While his formal calculus is based upon the state-transition model of Broy,

[†]Much of the research leading to this paper was completed while the author was a visitor at the Information Systems Engineering Group, Department of Computer Science, Christian-Albrechts University of Kiel, Germany.

the emphasis is solidly upon a less formal approach to the design and synthesis of schemata within the context of the *higher-order entity-relationship model (HERM)* [23], which is in turn based upon the classical entity-relationship (ER) model [8].

The present work was motivated by the desire to understand the various aspects of updates, in particular their propagation, in the the context of the database components of Thalheim. As the investigation evolved, however, it gradually became apparent that a somewhat different model of database component was necessary. Although database systems are a form of software system, and therefore it is not unreasonable to use similar formalisms for both, database systems also have special characteristics not shared by all software systems. In particular, in many database modelling scenarios, the central notions are sequences or streams of data, but rather database schemata and their views. This is particularly true in the context of updates. Furthermore, while the ER model and its extensions are widely used in database modelling and design, few if any actual systems are based upon it. Rather, ER-based designs are invariably converted to operational models, such as the relational model or an object-relational model, for final realization.

That an understanding of database updates is closely tied to database views has been recognized since the seminal work of Bancilhon and Spyratos [3], and has been elaborated greatly in [12] [13]. Thus, in an attempt to understand updates in the context of database components, it seems most natural to seek a model of components in which views play a central rôle. In this work, an alternative notion of database component, based upon schemata and views, is forwarded. Roughly speaking, the components are schemata which are interconnected via common views, called *ports*. Communication is then not via sequences or streams, but rather by applying updates to these ports.

An approach to components which is based upon the notions of database schema and view must begin with a choice for the representation of these notions. It turns out that there is relatively little to be gained by restricting the investigation to a particular data model, such as the relational model and its relatives. Rather, much more abstract models, in which a database schema is represented by a (possibly structured) set which represents its databases, and a database morphism is a structure preserving morphism, suffice completely. The formalizations surrounding the constant-complement strategy [3] [12] [13] adapt naturally to the component framework. In particular, the notions developed in these works for view updates provide, perhaps somewhat surprisingly, precisely the notions which are necessary for the representation of component interconnection.

The organization of this paper is as follows. In Section 2, the fundamental notions of view-based database components and their interconnections are presented. In Section 3, the notion of the hypergraph of a component is presented, and characterizations of "good" components, in terms of properties of their underlying hypergraphs, are presented. In Section 4, the topic of updates to to interconnections of components is explored briefly. Finally, in Section 5, some conclusions and further directions are presented.

2. The Basic Ideas of View-Based Database Components

The component-based approach is compositional, not decompositional. In other words, instead of beginning with a large main schema and breaking it into its constituent parts, the starting point is a collection of smaller schemata, together with information on how they may be combined. Since the decomposition of relational schemata is a topic which should be familiar to most readers, it is helpful to begin with a simple example of such decomposition, and then identify the conditions under which it may be reversed to yield a component-based composition theory.

2.1 Example — Reconstructing simple components from a decomposition Let $\mathbf{E}_0$ be the relational schema comprised of the single relation name $R_0[B_1B_2B_3]$, constrained by the set $\mathcal{F}_0 = \{B_1 \to B_2,\ B_2 \to B_3\}$ of functional dependencies (FDs), and let $\mathsf{LDB}(\mathbf{E}_0)$ denote the (finite) legal databases (*i.e.*, relations) of this schema (relative to appropriately chosen domains). The *view* $\Pi^{\mathbf{E}_0}_{B_1B_2} = (\mathbf{E}_1, \pi^{\mathbf{E}_0}_{B_1B_2})$ of $\mathbf{E}_0$ is the projection of $\mathbf{E}_0$ onto the attributes B_1B_2. More precisely, $\mathbf{E}_1$ denotes the schema whose single relation is $R_1[B_1B_2]$ and whose databases, denoted $\mathsf{LDB}(\mathbf{E}_1)$, are precisely the projections of members of $\mathsf{LDB}(\mathbf{E}_0)$ onto the attributes B_1B_2, with the view mapping $\pi^{\mathbf{E}_0}_{B_1B_2} : r \mapsto \{(\mathrm{b}_1, \mathrm{b}_2) \mid (\exists \mathrm{b}_3)((\mathrm{b}_1, \mathrm{b}_2, \mathrm{b}_3) \in r\}$. By construction, $\pi^{\mathbf{E}_0}_{B_1B_2}$ is surjective. Define the views $\Pi^{\mathbf{E}_0}_{B_2B_3} = (\mathbf{E}_2, \pi^{\mathbf{E}_0}_{B_2B_3})$ and $\Pi^{\mathbf{E}_0}_{B_1B_3} = (\mathbf{E}'_2, \pi^{\mathbf{E}_0}_{B_1B_3})$ similarly, with $R_2[B_2B_3]$ and $R_3[B_1B_3]$, their relation schemes, respectively.

It follows from one of the earliest and most widely known results in relational database theory [10, p. 31] that the decomposition of $\mathbf{E}_0$ into the views $\{\Pi^{\mathbf{E}_0}_{B_1B_2}, \Pi^{\mathbf{E}_0}_{B_2B_3}\}$ is *lossless*, in the sense that the decomposition mapping $\Delta_{(B_1B_2,B_2B_3)} : \mathsf{LDB}(\mathbf{E}_0) \to \mathsf{LDB}(\mathbf{E}_1) \times \mathsf{LDB}(\mathbf{E}_2)$ defined by $r \mapsto (\pi^{\mathbf{E}_0}_{B_1B_2}(r), \pi^{\mathbf{E}_0}_{B_2B_3}(r))$ is injective. Similarly, the decomposition of $\mathbf{E}_0$ into $\{\Pi^{\mathbf{E}_0}_{B_1B_2}, \Pi^{\mathbf{E}_0}_{B_1B_3}\}$ is lossless. However, the decomposition $\{\Pi^{\mathbf{E}_0}_{B_1B_2}, \Pi^{\mathbf{E}_0}_{B_2B_3}\}$ enjoys a second property which $\{\Pi^{\mathbf{E}_0}_{B_1B_2}, \Pi^{\mathbf{E}_0}_{B_1B_3}\}$ lacks. Specifically, define

$$(*) \qquad \mathsf{LDB}(\mathbf{E}_1) \otimes \mathsf{LDB}(\mathbf{E}_2) = \{(r_1, r_2) \in \mathsf{LDB}(\mathbf{E}_1) \times \mathsf{LDB}(\mathbf{E}'_2) \mid \pi^{\mathbf{E}_1}_{B_2}(r_1) = \pi^{\mathbf{E}_2}_{B_2}(r_2)\}$$

with $\pi^{\mathbf{E}_1}_{B_2}$ and $\pi^{\mathbf{E}_2}_{B_2}$ the obvious projections onto attribute B_2. Then $\Delta_{(B_1B_2,B_2B_3)}(\mathsf{LDB}(\mathbf{E}_0)) = \mathsf{LDB}(\mathbf{E}_1) \otimes \mathsf{LDB}(\mathbf{E}_2)$. Viewed another way, there is a bijective correspondence between the legal databases of $\mathbf{E}_0$ and those pairs of databases of $\mathbf{E}_1$ and $\mathbf{E}_2$ which agree on the common column B. Rissanen [21, Sec. 3] calls this property *independence*, and shows that it holds precisely in the case that the decomposition is lossless and *dependency preserving*; the latter meaning that a cover of the FDs of the original schema embeds into the views. In this case, the embedding is particularly simple, with $B_1 \to B_2$ lying in $\mathbf{E}_1$ and $B_2 \to B_3$ lying in $\mathbf{E}_2$. Indeed, $\mathsf{LDB}(\mathbf{E}_1)$ (resp. $\mathsf{LDB}(\mathbf{E}_2)$) is exactly the set of relations which satisfy $B_1 \to B_2$ (resp. $B_2 \to B_3$). On the other hand, there is no embedded cover of $\mathcal{F}_0$ into $\{\Pi^{\mathbf{E}_0}_{B_1B_2}, \Pi^{\mathbf{E}_0}_{B_1B_3}\}$. and so this independence property does not hold for that context. In [12, 2.17], this property is generalized to much more general classes of constraints.

What is remarkable about independence is that the question of whether a pair $(r_1, r_2) \in \mathsf{LDB}(\mathbf{E}_1) \times \mathsf{LDB}(\mathbf{E}_2)$ arises from a database of the main schema may be answered with reference only to a combination of local constraints on the component schemata and a view-based compatibility condition between them; no other knowledge of the main schema is necessary. This leads naturally to the component-based philosophy. Define the *components* $K_1 = (\mathbf{E}_1, \{\Pi^{\mathbf{E}_1}_{B_2}\})$ and $K_2 = (\mathbf{E}_2, \{\Pi^{\mathbf{E}_2}_{B_2}\})$. The set $\{\Pi^{\mathbf{E}_1}_{B_2}\}$ identifies the *ports* of K_1, and $\{\Pi^{\mathbf{E}_2}_{B_2}\}$ likewise for K_2, with $\Pi^{\mathbf{E}_i}_{B_2}$, for $i \in \{1, 2\}$; the obvious views being defined by projection. The underlying schemata of these two ports are identical. Denote it by $\mathbf{G}_1$; it has $T_1[B_2]$ as its sole relational symbol. The interconnection of K_1 and K_2 is depicted graphically in Figure 1 above.

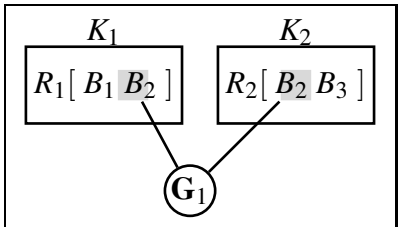


Figure 1: The interconnection of K_1 and K_2

This property — that the view schemata of two ports are identical — is called *star compatibility*, and is central to the interconnection of these components. The *star interconnection* of K_1 and K_2 is in effect a join of K_1 and K_2; its schema is the "union" of the schemata of K_1 and K_2, subject to the constraint that the two relations agree on the views defined by the components. In more detail, define the schema $\mathbf{E}_{12}$ to have the two relation symbols $R_2[B_1B_2]$ and $R_3[B_2B_3]$, constrained by the FDs $B_1 \to B_2$ and $B_2 \to B_3$ respectively, as well as the *port*

constraint which stipulates that for any $(r_1, r_2) \in \mathsf{LDB}(\mathbf{E}_1) \times \mathsf{LDB}(\mathbf{E}_2)$, $\pi^{\mathbf{E}_1}_{B_2}(r_1) = \pi^{\mathbf{E}_2}_{B_2}(r_2)$. Formally, this join of K_1 and K_2, the *compound component* defined by the star interconnection $\{\Pi^{\mathbf{E}_1}_{B_2} \Pi^{\mathbf{E}_2}_{B_2}\}$, is denoted $\mathsf{Cpt}\langle\{K_1, K_2\}, \{\Pi^{\mathbf{E}_1}_{B_2}, \Pi^{\mathbf{E}_2}_{B_2}\}\rangle$, and is given explicitly by $(\mathbf{E}_{12}, \{\Pi^{\mathbf{E}_{12}}_{B_2}\})$, with $\Pi^{\mathbf{E}_{12}}_{B_2}$ the view whose schema is $\mathbf{E}_{12}$ and whose view mapping projects either of the two relations of $\mathbf{E}_{12}$ onto attribute B_2. Note that $\mathbf{E}_{12}$ is the schema which is obtained by decomposing $\mathbf{E}_0$ into $\mathbf{E}_1$ and $\mathbf{E}_2$, and since $\mathbf{E}_{12}$ and $\mathbf{E}_0$ are isomorphic in a natural way, essentially the same information is represented. However, the component-based approach is compositional – the components K_1 and K_2 make no reference whatever to any main schema.

This same construction does not work for $\{\Pi^{\mathbf{E}_0}_{B_1B_2}, \Pi^{\mathbf{E}_0}_{B_1B_3}\}$. Upon defining $\Delta_{(B_1B_2, B_1B_3)}$: $\mathsf{LDB}(\mathbf{E}_0) \to \mathsf{LDB}(\mathbf{E}_1) \times \mathsf{LDB}(\mathbf{E}'_2)$ by $r \mapsto (\pi^{\mathbf{E}_0}_{B_1B_2}(r), \pi^{\mathbf{E}_0}_{B_1B_3}(r))$, with

$$(*') \qquad \mathsf{LDB}(\mathbf{E}_1) \otimes \mathsf{LDB}(\mathbf{E}'_2) = \{(r_1, r_2) \in \mathsf{LDB}(\mathbf{E}_1) \times \mathsf{LDB}(\mathbf{E}'_2) \mid \pi^{\mathbf{E}_1}_{B_1}(r_1) = \pi^{\mathbf{E}'_2}_{B_1}(r_2)\}$$

it is not the case that $\Delta_{(B_1B_2, B_1B_3)}(\mathsf{LDB}(\mathbf{E}_0)) = \mathsf{LDB}(\mathbf{E}_1) \otimes \mathsf{LDB}(\mathbf{E}'_2)$. Rather, to determine whether a pair $(r_1, r'_2) \in \mathsf{LDB}(\mathbf{E}_1) \times \mathsf{LDB}(\mathbf{E}'_2)$ arises as the projection of some $r \in \mathsf{LDB}(\mathbf{E}_0)$, it is necessary first to compute the join of that pair, since the constraint $A_2 \to A_3$ cannot be checked within either of the projections alone. In other words, upon defining the component $K'_2 = (\mathbf{E}'_2, \{\Pi^{\mathbf{E}_0}_{B_1B_3}\})$, it is not the case that the interconnection of K_1 and K'_2 will have a schema which is isomorphic to $\mathbf{E}_0$. Mathematically, the *congruences* of the pair $\{\Pi^{\mathbf{E}_0}_{B_1B_2}, \Pi^{\mathbf{E}_0}_{B_2B_3}\}$ commute, while those of $\{\Pi^{\mathbf{E}_0}_{B_1B_2}, \Pi^{\mathbf{E}_0}_{B_1B_3}\}$ do not. For details of these ideas, as well as their connection to the constant-complement update strategy for views, consult [12].

2.2 Database contexts As noted in the introduction, the the ideas developed here are not limited to a specific data model, such as the relational model or the ER model. Rather, they apply to virtually any database model. Formally, a *database context* is a pair $\mathcal{S} = (\mathcal{D}, \mathsf{LDB})$ in which $\mathcal{D}$ is a class of database schemata and their morphisms, and LDB is a function which associates to each schema $\mathbf{D}$ of $\mathcal{D}$ a set $\mathsf{LDB}(\mathbf{D})$ of legal states, and to each database morphism $f : \mathbf{D}_1 \to \mathbf{D}_2$ of $\mathcal{D}$ a function $\mathsf{LDB}(f) : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$. The idea is that a database schema $\mathbf{D}$ is modelled by its legal states alone, and that a database morphism is modelled by its underlying function. This is precisely the framework which is used in the original work on the constant-complement update strategy [3]. Suitable examples for $\mathcal{D}$ include the context of all relational schemata with morphisms defined by the relational algebra, nested relational models [20, Ch. 7], and the HERM model [23] together with suitably defined morphisms.

This modelling assumption must be faithful to the more structured framework which it represents. In particular, joins, as exemplified in equation $(*)$ of 2.1, must translate back and forth between the full data model and the LDB-based abstraction. These conditions are so natural that it is difficult to envision any reasonable formalization of database schemata and morphisms which would not satisfy them. Therefore, the straightforward but lengthy list of conditions which must be satisfied are not given here. However, for those readers familiar with the basic language of category theory [1] [15], these conditions can be characterized succinctly by requiring that the database schemata and morphisms form a concrete category $\mathcal{D}$ with finite limits over the category Set, with the further condition that the grounding functor $\mathsf{LDB} : \mathcal{D} \to \mathsf{Set}$ both preserve and reflect limits.

As a notation convenience, since the LDB notation on morphisms can become quite cumbersome, for a database morphism $f : \mathbf{D}_1 \to \mathbf{D}_2$ in $\mathcal{D}$, $\mathring{f}$ or $(f)^\circ$ will often be used as shorthand for $\mathsf{LDB}(f)$.

2.3 Notational convention Throughout the rest of this paper, unless stated specifically to the contrary, take $\mathcal{S}$ to be a database context. Unless stated specifically to the contrary (*e.g.*,

in examples), all database schemata and morphisms will be assumed to be based in $\mathcal{S}$.

Since views are central to the approach to components given here, a precise definition within the context $\mathcal{S}$ is essential. The following definition is based in large part upon those found in [12] and [13], to which the reader is referred for more detail.

2.4 Views Let $\mathbf{D}$ be a database schema. A *view* of $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V}, \gamma)$ with $\mathbf{V}$ a database schema and $\gamma : \mathbf{D} \to \mathbf{V}$ a database morphism with the property that $\mathring{\gamma} : \mathsf{LDB}(\mathbf{D}) \to \mathsf{LDB}(\mathbf{V})$ is surjective.

The *zero view* on $\mathbf{D}$ is $\mathsf{ZView}_{\mathbf{D}} = (\mathsf{ZSchema}, \mathsf{ZMor}_{\mathbf{D}})$, with $\mathsf{ZSchema}$ is a database schema with the property that $\mathsf{LDB}(\mathsf{ZSchema})$ consists of exactly one element, and $(\mathsf{ZMor}_{\mathbf{D}})^{\circ} : \mathsf{LDB}(\mathbf{D}) \to \mathsf{LDB}(\mathsf{ZSchema})$ the function which sends every $M \in \mathsf{LDB}(\mathbf{D})$ to the unique element of $\mathsf{LDB}(\mathsf{ZSchema})$. In other words, $\mathsf{ZSchema}$ is a constant schema with exactly one state, and $\mathsf{ZView}_{\mathbf{D}}$ is the view of $\mathbf{D}$ which maps every $M \in \mathsf{LDB}(\mathbf{D})$ to that state. Under the conditions identified in 2.2, a zero view exists for every schema $\mathbf{D}$, and is furthermore unique up to isomorphism.

Views occur very frequently in this work. To avoid the need to spell out the complete definition every time, the following convention will be followed. If a view is named by the Greek letter Γ, then the view morphism will be denote using γ, and $\mathsf{Schema}\langle\Gamma\rangle$ will be used as an alias for the underlying view schema. This convention furthermore extends to all subscripted and superscripted variants. For example, for the view Γ defined above, $\mathsf{Schema}\langle\Gamma\rangle$ is an alias for $\mathbf{V}$. Similarly, for a view named Γ'_i, the full definition is $\Gamma'_i = (\mathsf{Schema}\langle\Gamma'_i\rangle, \gamma'_i)$. Additionally, when $\mathsf{Schema}\langle X\rangle$ appears as the argument of another notation, X will frequently be used in its stead when no confusion can result. In particular, $\mathsf{LDB}(\Gamma_i)$ will be used as an abbreviation for $\mathsf{LDB}(\mathsf{Schema}\langle\Gamma_i\rangle)$.

2.5 Components The formal definition of a component follows the pattern introduced in the example of 2.1, but is based not upon the relational model but rather upon the more general model of database schemata and views in the context $\mathcal{S}$. Specifically, a *component* is an ordered pair $C = (\mathsf{Schema}(C), \mathsf{Ports}(C))$ which satisfies the following two conditions.

(cpt-i) $\mathsf{Schema}(C)$ is a database schema.

(cpt-ii) $\mathsf{Ports}(C)$ is a finite set of views of $\mathsf{Schema}(C)$, called the *ports* of C, with the property that none of the ports are zero views.

$\mathsf{LDB}(C)$ will be used as notational shorthand for $\mathsf{LDB}(\mathsf{Schema}(C))$.

2.6 Name-normalized components In describing interconnections of components, it is essential to be able to recover the identity of the embodying component from the name of a port. While an elaborate tagging formalism could be developed, the solution proposed here is much simpler; namely, that for a given set of components, all port names are *globally* unique. Since this is only a naming convention, there is no loss of generality in such an assumption. Specifically, let X be a finite set of components.

(a) X is called *name normalized* if for distinct $C, C' \in X$, $\mathsf{Ports}(C) \cap \mathsf{Ports}(C') = \emptyset$.

(b) For $Y \subseteq X$, define $\mathsf{Ports}\langle Y\rangle = \bigcup\{\mathsf{Ports}(C) \mid C \in Y\}$. Thus, $\mathsf{Ports}\langle Y\rangle$ is just the set of all ports which are associated with some component in Y.

(c) If X is name normalized and $\Gamma \in \mathsf{Ports}\langle X\rangle$, then $\mathsf{SrcCpt}(\Gamma)$ denotes the *source component* of Γ, which is the unique $C \in X$ for which $\Gamma \in \mathsf{Ports}(C)$.

2.7 Star interconnections and interconnection families Let X be a finite set of components.

(a) For $I \subseteq \mathsf{Ports}\langle X\rangle$, $\mathsf{Components}(I)$ denotes $\{\mathsf{SrcCpt}(\Gamma) \mid \Gamma \in I\}$. Thus, $\mathsf{Components}(I)$ is just the set of all components which are associated with a port in I.

(b) A *star-compatible set* for X is an $I \subseteq \mathsf{Ports}\langle X\rangle$ with the property that for all $\Gamma, \Gamma' \in I$, $\mathsf{Schema}\langle\Gamma\rangle = \mathsf{Schema}\langle\Gamma'\rangle$.

(c) A *star interconnection over X* is a star-compatible set I for X with the property that for distinct $\Gamma, \Gamma' \in I$, $\mathsf{SrcCpt}(\Gamma)$ and $\mathsf{SrcCpt}(\Gamma')$ are distinct as well. In this case, I is called a *star interconnection* of $\mathsf{Components}(I)$.

(d) A *interconnection family* for X is a finite set of star interconnections over X.

The above definitions are critical to the interconnection model. For a set $\{C_1, C_2, \ldots, C_k\}$ of components to be coupled in a single star configuration using ports $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$, respectively, the schemata of the ports must be identical (and not just isomorphic). This is necessary because in the interconnection, the states of these views must always be the same. The resulting star interconnection, as defined formally below, is illustrated for $k = 4$ in Figure 2. This is somewhat distinct from the notion of a *star component*, as defined in [24]. Here it is the interconnection, and not the component itself, which has the star property.

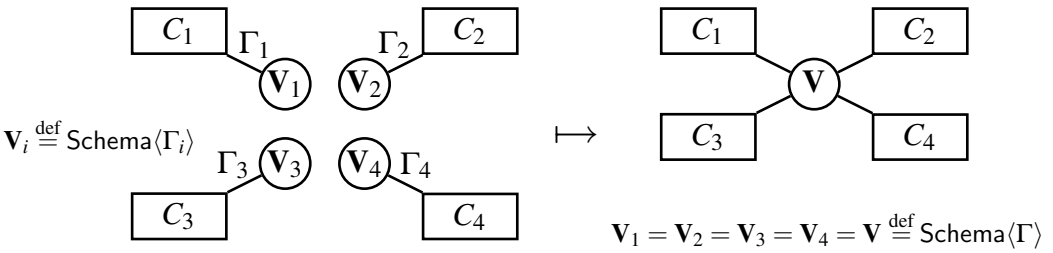


Figure 2: The star-compatibility condition and the interconnection of four components

It is possible to construct a maximal interconnection family, from which all others can be obtained via appropriate subset operations.

(e) For $\Gamma \in \mathsf{Ports}\langle X\rangle$, define $\mathsf{MaxStar}_X(\Gamma) = \{\Gamma' \in \mathsf{Ports}\langle X\rangle \mid \mathsf{Schema}\langle\Gamma\rangle = \mathsf{Schema}\langle\Gamma'\rangle\}$, and define $\mathsf{MaxStar}(X) = \{\mathsf{MaxStar}_X(\Gamma) \mid \Gamma \in \mathsf{Ports}\langle X\rangle\}$.

Each member of $\mathsf{MaxStar}(X)$ is a maximal star-compatible set for X. Thus, J is an interconnection family for X iff J is the union of disjoint subsets of members of $\mathsf{MaxStar}(X)$.

2.8 Notational convention Unless specifically stated to the contrary, for the rest of this paper, take X to be a name-normalized set of components with J an interconnection family for X.

2.9 Annotated examples In 2.12 and 2.13, examples are presented which illustrate many of the ideas which have been developed thus far, as well as those which will be developed in the rest of this section. Rather than distributing fragments of these examples throughout the text, it seems more appropriate to present them in unified fashion. The reader is therefore encouraged to look ahead to these examples for clarification of the concepts which are developed.

2.10 The compound component defined by an interconnection family A central idea surrounding the component philosophy is that simple components may be combined to form more complex ones. While this idea is simple in principle, there are nevertheless some complicating details which mandate that not every possible interconnection family is suitable for the formation of a complex component. These issues are now investigated in more detail.

(a) The schema $\mathsf{Schema}\langle X, J\rangle$ defined by J on X is given as follows.

$$\mathsf{LDB}(\mathsf{Schema}\langle X,J\rangle) = \{\langle M_B\rangle \in \prod_{B\in Y} \mathsf{LDB}(B) \mid (\forall I \in J)(\forall\{\Gamma_1,\Gamma_2\} \subseteq I)(\mathring{\gamma}_1(M_{\mathsf{SrcCpt}(\Gamma_1)}) = \mathring{\gamma}_2(M_{\mathsf{SrcCpt}(\Gamma_2)}))\}$$

(b) For $C \in X$, define the *natural projection morphism* $\mathsf{NatProj}_{\langle J;X\to C\rangle} : \mathsf{Schema}\langle X,J\rangle \to \mathsf{Schema}(C)$ on elements by $\langle M_B\rangle_{B\in X} \mapsto M_C$.

It is clear that $\mathsf{Schema}\langle X,J\rangle$ is the natural schema for the interconnection of the elements of X into a single component, using J as the interconnection family. There is, however, a complication. For this complex component, it is necessary to identify the ports. The obvious solution is to define one common port for each star interconnection. In light of the definition of (a) above, the view of this common port will not depend upon which of the components from X is used to define it. Unfortunately, the combined ports may no longer be views, because the underlying mapping is no longer surjective. In general, the constraints on the schemata of the constituents which arise by combining components may limit the values which may appear on the ports. It is therefore essential to identify conditions under which these problems do not occur. The following definitions lay the framework for studying this question in more detail.

(c) The component $C \in X$ is *free in* X if for any $N \in \mathsf{LDB}(C)$, there is an $\langle M_B\rangle_{B\in X} \in \mathsf{LDB}(\mathsf{Schema}\langle X,J\rangle)$ with the property that $M_C = N$.

(d) Let $C \in Y$ and let $\Gamma \in \mathsf{Ports}(C)$. The port Γ is said to be *free in X with respect to J* if for any $N \in \mathsf{LDB}(\Gamma)$, there is an $\langle M_B\rangle_{B\in X} \in \mathsf{LDB}(\mathsf{Schema}\langle X,J\rangle)$ with $\mathring{\gamma}(M_C) = N$.

(e) X is *free for ports with respect to J* if for every $C \in X$ and $\Gamma \in \mathsf{Ports}(C)$, Γ is free in X with respect to J.

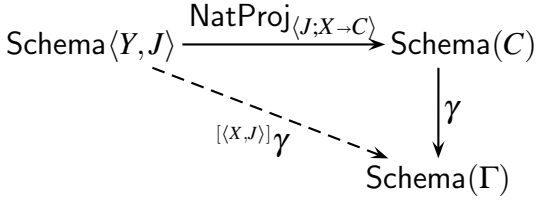
The condition of the component C being free in X is the stronger of the two; it essentially states that the interconnection does not further constrain C. It is very easy to find examples of situations in which constituent components are not free; see 2.12.

The condition of a view Γ being free in X with respect to J is strictly weaker, since the entire component C need not be unconstrained, but rather only that each of its ports must be unconstrained individually. In the example of 2.12, all ports are free with respect to $\mathsf{MaxStar}(L)$. Nevertheless, it is possible to construct relatively simple relational examples in which this condition is not satisfied, as illustrated in 2.13.

The weaker condition of X being free for ports with respect to J is sufficient to admit a consistent definition for the ports of a compound component. By its very nature, it guarantees that the view mapping for the combined port will be surjective — it is both necessary and sufficient. The formal details are as follows.

For parts (f)-(i), assume further that X is free for ports with respect to J.

(f) Let $C \in X$ and let $\Gamma \in \mathsf{Ports}(C)$. Define the *lifting of* Γ to $\mathsf{Schema}\langle X,J\rangle$ to be the pair ${}^{[\langle X,J\rangle]}\Gamma = (\mathsf{Schema}\langle X,J\rangle, {}^{[\langle X,J\rangle]}\gamma)$ with ${}^{[\langle Y,J\rangle]}\gamma : \mathsf{Schema}\langle X,J\rangle \to \mathsf{Schema}\langle\Gamma\rangle$ given as the composition illustrated to the right.



(g) Let $I \subseteq J$, and let $\Gamma,\Gamma' \in \mathsf{Ports}\langle X\rangle$. Define the equivalence relation $\equiv_{\langle X,J\rangle}$ on liftings of such views by ${}^{[\langle X,J\rangle]}\Gamma \equiv_{\langle X,J\rangle} {}^{[\langle X,J\rangle]}\Gamma'$ iff $\Gamma = \Gamma'$ or else there is a $I \in J$ with $\{\Gamma,\Gamma'\} \subseteq X$. Let $\left[{}^{[\langle X,J\rangle]}\Gamma\right]$ denote the equivalence class of ${}^{[\langle X,J\rangle]}\Gamma$ under this equivalence relation.

(h) Define $\mathsf{Ports}\langle X,J\rangle = \{\left[{}^{[\langle X,J\rangle]}\Gamma\right] \mid \Gamma \in \mathsf{Ports}\langle X\rangle\}$.

(i) The *compound component defined by* $\langle X,J\rangle$ is given as follows.

$$\mathsf{Cpt}\langle X,J\rangle = (\mathsf{Schema}\langle X,J\rangle, \mathsf{Ports}\langle X,J\rangle)$$

It is worth repeating that the above notion of compound component is well defined only in the case that X is free for ports with respect to J.

It should perhaps be noted that it is not always necessary (or desirable) to include *all* ports from the constituents in the compound component. However, the choice of which ones to include and which ones to exclude cannot be made on a formal level; rather, it must be a modelling decision. For example, consider forming a compound component from $\{L_1,L_2\}$ of 2.12. If one decides to exclude from the compound component those ports which have already been matched, then it would be impossible to connect L_3 to the compound of L_1 and L_2, since the necessary port has been removed. This must be a design decision, not a mathematical one.

2.11 Subcomponents of compound components Just as simple components may be combined to construct more complex ones, so too may simpler components be extracted from complex ones. For this extraction process to yield well-defined subcomponents, certain conditions must be met, which are now explored in more detail. In the following, let $Y \subseteq X$.

(a) For J an interconnection family for X, the *relativization* of J to Y is $\mathsf{Relt}\langle J,Y\rangle = \{I \cap (\bigcup\{\mathsf{Ports}(C) \mid C \in Y\}) \mid I \in J\}$. Thus, $\mathsf{Relt}\langle J,Y\rangle$ is obtained from J by removing all ports which are not associated with components in Y.

(b) The *relative schema* $\mathsf{Schema}\langle Y,J\rangle$ defined by J on Y is given as follows.

$$\begin{aligned}\mathsf{LDB}(\mathsf{Schema}\langle Y,J\rangle) = \{\langle M_B\rangle \in \prod_{B\in Y} \mathsf{LDB}(B) \mid \\ (\forall I \in \mathsf{Relt}\langle J,Y\rangle)(\forall\{\Gamma_1,\Gamma_2\} \subseteq I)(\mathring{\gamma}_1(M_{\mathsf{SrcCpt}(\Gamma_1)}) = \mathring{\gamma}_2(M_{\mathsf{SrcCpt}(\Gamma_2)}))\}\end{aligned}$$

Thus, $\mathsf{Schema}\langle Y,J\rangle = \mathsf{Schema}\langle Y,\mathsf{Relt}\langle J,Y\rangle\rangle$. In other words, the relative schema $\mathsf{Schema}\langle Y,J\rangle$ is precisely the schema of the compound component $\mathsf{Cpt}\langle Y,\mathsf{Relt}\langle J,Y\rangle\rangle$. On the other hand, one can also consider the schema obtained by projecting from $\mathsf{Schema}\langle X,J\rangle$ the constituent schemata which arise from components in Y.

(c) The *projected schema* $\mathsf{ProjSch}^X\langle Y,J\rangle$ is defined as follows.

$$\begin{aligned}\mathsf{LDB}(\mathsf{ProjSch}^X\langle Y,J\rangle) = \{\langle M_B\rangle \in \prod_{B\in Y} \mathsf{LDB}(B) \mid \\ (\exists\langle N_B\rangle_{B\in X} \in \mathsf{LDB}(\mathsf{Schema}\langle X,J\rangle))(\forall B \in Y)(N_B = M_B)\}\end{aligned}$$

(d) Call Y *closed in* X *with respect to* J if $\mathsf{ProjSch}^X\langle Y,J\rangle = \mathsf{Schema}\langle Y,J\rangle$.

The above closure conditions is very important, because it states that $\mathsf{Cpt}\langle Y, \mathsf{Relt}\langle J,Y\rangle\rangle$ is embedded in $\mathsf{Cpt}\langle X,J\rangle$ without the latter imposing any additional constraints. Clearly $\mathsf{LDB}(\mathsf{ProjSch}^X\langle Y,J\rangle) \subseteq \mathsf{LDB}(\mathsf{Schema}\langle Y,J\rangle)$; the reverse inclusion $\mathsf{LDB}(\mathsf{Schema}\langle Y,J\rangle) \subseteq \mathsf{LDB}(\mathsf{ProjSch}^X\langle Y,J\rangle)$ holds precisely when Y Is closed in X with respect to J.

The definition of a subcomponent now proceeds similarly to that of a compound component, as given in 2.10(i).

(f) Under the condition that Y is closed in X with respect to J and that Y is free for ports with respect to $\mathsf{Relt}\langle J,Y\rangle$, define the *subcomponent of* $\langle X,J\rangle$ *generated by* Y as follows.

$$\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle = (\mathsf{ProjSch}^X\langle Y,J\rangle, \mathsf{Ports}\langle Y, \mathsf{Relt}\langle Y,J\rangle\rangle)$$

An illustration of why the closure condition is essential for this definition is given at the end of 2.12.

As noted at the end of 2.10, the choice of which ports to include and which to exclude in a compound component is a design condition. This is equally true for subcomponents. However, in the latter case, there is a classification which is useful — to partition the ports of $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle$ into those which connect it to other parts of X and those which do not. The formalization is as follows. Again, for this definition to make sense, it must be assumed that Y is free for ports with respect to $\mathsf{Relt}\langle Y,J\rangle$.

(g) Define the set of *all ports*, *external ports*, the *internal ports*, of Y with respect to J, respectively, as follows.

$$\begin{aligned}
\mathsf{AllPorts}\langle Y,J\rangle &= \{[^{[\langle Y,\mathsf{Relt}\langle Y,J\rangle\rangle]}\Gamma] \mid \Gamma \in \mathsf{Ports}\langle Y\rangle\} \\
\mathsf{ExtPorts}\langle Y,J\rangle &= \{[^{[\langle Y,\mathsf{Relt}\langle Y,J\rangle\rangle]}\Gamma] \mid (\exists I \in J)(\exists \Gamma')((\{\Gamma,\Gamma'\} \subseteq I) \wedge \\
&\qquad\qquad (\Gamma \in \mathsf{Ports}\langle Y\rangle) \wedge (\Gamma' \notin \mathsf{Ports}\langle Y\rangle))\} \\
\mathsf{IntPorts}\langle Y,J\rangle &= \mathsf{AllPorts}\langle Y,J\rangle \setminus \mathsf{ExtPorts}\langle Y,J\rangle
\end{aligned}$$

Finally, there is a natural projection morphism, whose underlying function is guaranteed to be surjective, defined as follows.

(h) For $Z \subseteq Y$, define the *natural projection morphism* $\mathsf{NatProj}_{\langle X;J;Y\to Z\rangle} : \mathsf{ProjSch}^X\langle Y,J\rangle \to \mathsf{ProjSch}^X\langle Z,J\rangle$ on elements by $\langle M_B\rangle_{B\in Y} \mapsto \langle M_B\rangle_{B\in Z}$. Define the *natural projection view of* $\mathsf{ProjSch}^X\langle Y,J\rangle$ *to* Z to be $\mathsf{ProjView}\langle J;Y\to Z\rangle = (\mathsf{ProjSch}^X\langle Y,J\rangle, \mathsf{NatProj}_{\langle X;J;Y\to Z\rangle})$.

2.12 Example — An illustrative set of components The purpose of this example is to provide a setting in which many of the concepts which are introduced in this paper may be illustrated. It is not intended to model a "real" database situation, but rather to illustrate a wide variety of possibilities. All components are based upon the relational model, and all ports are defined via projections, although neither of these limitations is inherent to the model. By using the familiar relational model, the key ideas can be illustrated in a relatively compact fashion, and certain modelling pitfalls can be highlighted.

Table 1 summarizes the key information for each atomic component. For the port names, the superscript identifies the component, while the subscript identifies the attributes which are projected. Since each attribute name is used at most once in each component, this convention is unambiguous. For simplicity, it will be assumed that with each attribute A_i is associated a countably infinite domain $\mathsf{dom}(A_i)$, while the legal relations themselves must be finite.

Table 2 summarizes information about the ports, grouped by those which have identical underlying schemata. This is a natural grouping, since ports with identical schemata are precisely those which may be coupled to one another. Figure 3 shows all possible (star) interconnections of these components. That is, it connects all ports with identical underlying schemata. The components are shown as rectangles, while the port schemata are displayed as circles. Letting $L = \{L_1, L_2, L_3, L_4, L_5, L_6, L_7\}$, the associated interconnection family is

$$\begin{aligned}\mathsf{MaxStar}(L) = \{&\{\Pi^{\mathbf{F}_1}_{A_1}, \Pi^{\mathbf{F}_2}_{A_1}, \Pi^{\mathbf{F}_3}_{A_1}\}, \{\Pi^{\mathbf{F}_3}_{A_4A_5}, \Pi^{\mathbf{F}_4}_{A_4A_5}\}, \{\Pi^{\mathbf{F}_4}_{A_7}, \Pi^{\mathbf{F}_5}_{A_7}\},\\ &\{\Pi^{\mathbf{F}_4}_{A_8}, \Pi^{\mathbf{F}_7}_{A_8}\}, \{\Pi^{\mathbf{F}_5}_{A_0}, \Pi^{\mathbf{F}_6}_{A_0}\}, \{\Pi^{\mathbf{F}_6}_{A_9}, \Pi^{\mathbf{F}_7}_{A_9}\}, \{\Pi^{\mathbf{F}_3}_{A_4}, \Pi^{\mathbf{F}_4}_{A_4}\}\}\end{aligned}$$

Of course, there is no requirement that when a set of components is interconnected, all possible connections must be included. Any subset of a member of $\mathsf{MaxStar}(L)$ is a valid star interconnection. Thus, any set consisting of disjoint subsets of members of $\mathsf{MaxStar}(L)$ is a valid star interconnection over L.

Comp. Name	Schema Name	Relations	Schema Constraints	Ports
L_1	$\mathbf{F}_1$	$R_1[A_1A_2]$		$\Pi^{\mathbf{F}_1}_{A_1}$
L_2	$\mathbf{F}_2$	$R_2[A_1A_3]$		$\Pi^{\mathbf{F}_2}_{A_1}$
L_3	$\mathbf{F}_3$	$R_3[A_1A_4A_5]$	$A_4 \to A_5$	$\Pi^{\mathbf{F}_3}_{A_1}$ $\Pi^{\mathbf{F}_3}_{A_4A_5}$
L_4	$\mathbf{F}_4$	$R_{4a}[A_4A_5A_6]$ $R_{4b}[A_7A_8]$	$A_4 \to A_5$ $A_7 \to A_8$	$\Pi^{\mathbf{F}_4}_{A_4A_5}$ $\Pi^{\mathbf{F}_4}_{A_7}$ $\Pi^{\mathbf{F}_4}_{A_8}$
L_5	$\mathbf{F}_5$	$R_5[A_7A_0]$	$A_0 \to A_7$	$\Pi^{\mathbf{F}_5}_{A_7}$ $\Pi^{\mathbf{F}_5}_{A_0}$
L_6	$\mathbf{F}_6$	$R_6[A_9A_0]$	$A_9 \to A_0$	$\Pi^{\mathbf{F}_6}_{A_9}$ $\Pi^{\mathbf{F}_6}_{A_0}$
L_7	$\mathbf{F}_7$	$R_7[A_8A_9]$	$A_8 \to A_9$	$\Pi^{\mathbf{F}_7}_{A_8}$ $\Pi^{\mathbf{F}_7}_{A_9}$

Table 1: The atomic components of the running example

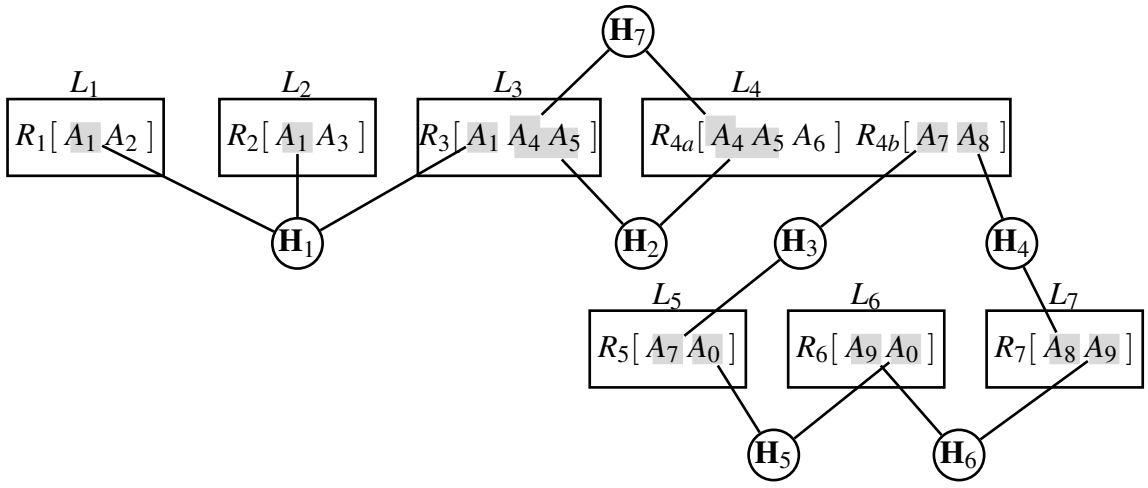


Figure 3: Graphical depiction of all possible interconnections

Observe that the components L_4, L_5, L_6, and L_7 are not free in L with respect to $\mathsf{MaxStar}(L)$. Indeed, the interconnection forces the additional constraints $A_8 \to A_7$, $A_7 \to A_0$, $A_0 \to A_9$, and $A_9 \to A_8$ on L_4, L_5, L_6, and L_7, respectively. On the other hand, L is free for ports with respect to $\mathsf{MaxStar}(L)$.

Finally, an illustration of the need for the closure condition in the definition of subcomponent is given. Let $X_{567} = \{L_5, L_6, L_7\}$, $J_{567} = \{\{\Pi^{\mathbf{F}_5}_{A_0}, \Pi^{\mathbf{F}_6}_{A_0}\}, \{\Pi^{\mathbf{F}_6}_{A_9}, \Pi^{\mathbf{F}_7}_{A_9}\}\}$, and $Y_{57} =$

Schema Name	Schema Relations	Schema Constraints	Associated Ports and View Mappings	
$\mathbf{H}_1$	$T_1[A_1]$		$\Pi^{\mathbf{F}_1}_{A_1}$	$\pi^{\mathbf{F}_1}_{A_1} : \mathbf{F}_1 \to \mathbf{H}_1$
			$\Pi^{\mathbf{F}_2}_{A_1}$	$\pi^{\mathbf{F}_2}_{A_1} : \mathbf{F}_2 \to \mathbf{H}_1$
			$\Pi^{\mathbf{F}_3}_{A_1}$	$\pi^{\mathbf{F}_3}_{A_1} : \mathbf{F}_3 \to \mathbf{H}_1$
$\mathbf{H}_2$	$T_2[A_4A_5]$	$A_4 \to A_5$	$\Pi^{\mathbf{F}_3}_{A_4A_5}$	$\pi^{\mathbf{F}_3}_{A_4A_5} : \mathbf{F}_3 \to \mathbf{H}_2$
			$\Pi^{\mathbf{F}_4}_{A_4A_5}$	$\pi^{\mathbf{F}_4}_{A_4A_5} : \mathbf{F}_4 \to \mathbf{H}_2$
$\mathbf{H}_3$	$T_3[A_7]$		$\Pi^{\mathbf{F}_4}_{A_7}$	$\pi^{\mathbf{F}_4}_{A_7} : \mathbf{F}_4 \to \mathbf{H}_3$
			$\Pi^{\mathbf{F}_5}_{A_7}$	$\pi^{\mathbf{F}_5}_{A_7} : \mathbf{F}_5 \to \mathbf{H}_3$
$\mathbf{H}_4$	$T_4[A_8]$		$\Pi^{\mathbf{F}_4}_{A_8}$	$\pi^{\mathbf{F}_4}_{A_8} : \mathbf{F}_4 \to \mathbf{H}_4$
			$\Pi^{\mathbf{F}_7}_{A_8}$	$\pi^{\mathbf{F}_7}_{A_8} : \mathbf{F}_7 \to \mathbf{H}_4$
$\mathbf{H}_5$	$T_5[A_0]$		$\Pi^{\mathbf{F}_5}_{A_0}$	$\pi^{\mathbf{F}_5}_{A_0} : \mathbf{F}_5 \to \mathbf{H}_5$
			$\Pi^{\mathbf{F}_6}_{A_0}$	$\pi^{\mathbf{F}_6}_{A_0} : \mathbf{F}_6 \to \mathbf{H}_5$
$\mathbf{H}_6$	$T_6[A_9]$		$\Pi^{\mathbf{F}_6}_{A_9}$	$\pi^{\mathbf{F}_6}_{A_9} : \mathbf{F}_6 \to \mathbf{H}_6$
			$\Pi^{\mathbf{F}_7}_{A_9}$	$\pi^{\mathbf{F}_7}_{A_9} : \mathbf{F}_7 \to \mathbf{H}_6$
$\mathbf{H}_7$	$T_7[A_4]$		$\Pi^{\mathbf{F}_3}_{A_4}$	$\pi^{\mathbf{F}_3}_{A_4} : \mathbf{F}_4 \to \mathbf{H}_7$
			$\Pi^{\mathbf{F}_4}_{A_4}$	$\pi^{\mathbf{F}_4}_{A_4} : \mathbf{F}_4 \to \mathbf{H}_7$

Table 2: The port schemata of the running example

$\{L_5, L_7\}$. Then $\mathsf{Relt}\langle Y_{57}, J_{567}\rangle = \{\{\Pi^{\mathbf{F}_5}_{A_0}\}, \{\Pi^{\mathbf{F}_7}_{A_9}\}\}$. Operationally, $\mathsf{Relt}\langle Y_{57}, J_{567}\rangle$ is equivalent to $\emptyset$; that is, it imposes no constraints at all. This implies that the subcomponent $\mathsf{SubCpt}_{\langle X_{567}, J_{57}\rangle}\langle \mathsf{ProjSch}^{X_{567}}\langle Y_{57}, J_{567}\rangle\rangle$ is not well defined. To illustrate this directly, let $M_{L_5} = \{(\mathrm{a}_7, \mathrm{a}_0)\} \in \mathsf{LDB}(\mathbf{F}_5)$ and $M_{L_7} = \{(\mathrm{a}_8, \mathrm{a}_9), (\mathrm{a}'_8, \mathrm{a}_9)\} \in \mathsf{LDB}(\mathbf{F}_7)$ with $\mathrm{a}_8 \neq \mathrm{a}'_8$. In view of the FD $A_9 \to A_0$ on $\mathbf{F}_6$, there can be no $M_{L_6} \in \mathsf{LDB}(\mathbf{F}_6)$ such that $(M_{L_5}, M_{L_6}, M_{L_7}) \in \mathsf{LDB}(\mathsf{ProjSch}^{X_{567}}\langle Y_{57}, J_{567}\rangle)$, since the fact that M_{L_5} contains only one tuple implies that M_{L_7} can consist of only one tuple as well.

2.13 Example — A pair of components whose interconnection is not free for ports It is useful to show how a simple interconnection of relational components can violate the condition 2.10(e) of being free for ports. To this end, let A_1 and A_2 be attributes with the same countably infinite domain; $\mathsf{dom}(A_1) = \mathsf{dom}(A_2)$. There are three components over these domains, as identified in Table 3. Each component has two ports, one for the projection of its relation on A_1, and a second for A_2. For the ports $\Pi^{\mathbf{F}_\alpha}_{A_1}$, $\Pi^{\mathbf{F}_\beta}_{A_1}$, and $\Pi^{\mathbf{F}_\delta}_{A_1}$, let the port schema have the single relation symbol $T_{A_1}[A_1]$, and for the ports $\Pi^{\mathbf{F}_\alpha}_{A_2}$, $\Pi^{\mathbf{F}_\beta}_{A_2}$, and $\Pi^{\mathbf{F}_\delta}_{A_2}$, let the port schema have the single relation symbol $T_{A_2}[A_2]$. There are no constraints associated with these port schemata, other than the domain constraints. However, if L_α and L_β are interconnected via $J_{\alpha\beta} = \{\{\Pi^{\mathbf{F}_\alpha}_{A_1}, \Pi^{\mathbf{F}_\beta}_{A_1}\}, \{\Pi^{\mathbf{F}_\alpha}_{A_2}, \Pi^{\mathbf{F}_\beta}_{A_2}\}\}$, then it is easy to see that $\mathsf{LDB}(\mathsf{Schema}\langle\{L_\alpha, L_\beta\}, J_{\alpha\beta}\rangle) = \emptyset$. Similarly, letting $J_{\alpha\delta} = \{\{\Pi^{\mathbf{F}_\alpha}_{A_1}, \Pi^{\mathbf{F}_\beta}_{A_1}\}, \{\Pi^{\mathbf{F}_\gamma}_{A_2}, \Pi^{\mathbf{F}_\gamma}_{A_2}\}\}$, it follows that $\mathsf{LDB}(\mathsf{Schema}\langle\{L_\alpha, L_\delta\}, J_{\alpha\delta}\rangle) = \{\emptyset\}$. Thus, $\{L_\alpha, L_\beta, L_\delta\}$ is not free for ports

with respect to either $J_{\alpha\beta}$ or $J_{\alpha\delta}$.

Comp. Name	Schema Name	Relations	Schema Constraints	Ports
L_α	$\mathbf{F}_\alpha$	$R_\alpha[A_1A_2]$	$R_\alpha[A_1] \subseteq R_\alpha[A_2]$	$\Pi_{A_1}^{\mathbf{F}_\alpha}$ $\Pi_{A_2}^{\mathbf{F}_\alpha}$
L_β	$\mathbf{F}_\beta$	$R_\beta[A_1A_2]$	$R_\beta[A_1] \not\subseteq R_\beta[A_2]$	$\Pi_{A_1}^{\mathbf{F}_\beta}$ $\Pi_{A_2}^{\mathbf{F}_\beta}$
L_δ	$\mathbf{F}_\delta$	$R_\delta[A_1A_2]$	$R_\delta[A_1] \cap R_\delta[A_2] = \emptyset$	$\Pi_{A_1}^{\mathbf{F}_\delta}$ $\Pi_{A_2}^{\mathbf{F}_\delta}$

Table 3: The atomic components for 2.13

3. Acyclic Interconnections of Components

As observed in 2.10 and 2.11, the property of a set X of components being free for ports with respect to an interconnection family J is critical for the definition of both compound components and subcomponents. Therefore, it is crucial to identify conditions under which this condition is met. Fortunately, by requiring the acyclicity of a hypergraph derived from the interconnection, it is possible to guarantee the even stronger condition that each constituent component of is free in X.

The reader is perhaps familiar with the use of hypergraphs in characterizing the structure of relational decompositions [9]. However, there are very substantial differences between the hypergraph of a compound component, as defined here, and the hypergraph of a relational decomposition. The use of hypergraphs in this paper is closer to that found in more general characterizations of desirable schemata, as studied in [11]. In any case, it seems appropriate to give a self-contained presentation of the ideas.

3.1 Hypergraphs and acyclicity To begin, a very brief summary of the key notions from the theory of hypergraphs is given. The standard reference on this subject is the monograph of Berge [4] , to which the reader is referred for details.

A *hypergraph* is a pair $G = (V,H)$ in which V is a finite set of *vertices* and $H \subseteq \mathscr{P}(V)$ (the set of all subsets of V), with each $h \in H$ containing at least two distinct elements.[1] The members of H are called *hyperedges*. A *path* from v_1 to v_n in G is a sequence $\langle v_1, h_1, v_2, h_2, .., v_{n-1}, h_{n-1}, v_n\rangle$ in which the following conditions hold:

(i) $v_i \in V$ for $1 \leq i \leq n$ with $\{v_i \mid 1 \leq i \leq n-1\}$ all distinct, and $\{v_i \mid 2 \leq i \leq n\}$ all distinct. It may be the case that $v_1 = v_n$, but this is not necessary.

(ii) $h_i \in H$ for $1 \leq i \leq n-1$ with $\{h_i \mid 1 \leq i \leq n-1\}$ all distinct.

(iii) $\{v_i, v_{i+1}\} \in h_i$ for $1 \leq i \leq n-1$.

The number $n-1$ is called the *length* of the path. A *(Berge)*[2] *cycle* in G is a path of length at least two from a vertex v to itself. G is called *(Berge) acyclic* if it does not contain any (Berge) cycles.

For $V' \subseteq V$, the *full subhypergraph of G generated by V'* is $\mathsf{SubHGraph}\langle G, V'\rangle = (V', \{h \cap V' \mid h \in H\})$. $V' \subseteq V$ is *closed* in G if whenever $v_1, v_n \in V'$ and $\langle v_1, h_1, v_2, h_2, .., v_{n-1}, h_{n-1}, v_n\rangle$ is a path from v_1 to v_n in G, then $\langle v_1, h_1 \cap V', v_2, h_2 \cap V', .., v_{n-1}, h_{n-1} \cap V', v_n\rangle$ is a path from v_1 to v_n in $\mathsf{SubHGraph}\langle G, V'\rangle$.

[1] In [4, Ch. 17, §1], hyperedges (*arêtes*) are allowed to have only one member, implying that a hyperedge may connect a vertex to itself. Such edges are not allowed in the formalism presented here.

[2] In [4], such entities are called simply cycles, but in other contexts, such as that of [9], many different types of cycles for hypergraphs are investigated, so the qualifier "Berge" is appended.

3.2 The hypergraph of an interconnection family Interconnection hypergraphs are defined as follows.

(a) The *interconnection hypergraph* of X defined by J, denoted $\mathsf{IntGraph}_X(J)$, has X as its set of vertices and $\{\mathsf{Components}(I) \mid I \in J\}$ as its hyperedges. The interconnection hypergraph for $\mathsf{MaxStar}(L)$ of 2.12 is shown in Figure 4 to the right. Each hyperedge is represented as an ellipse, with its members the component names which it encircles.

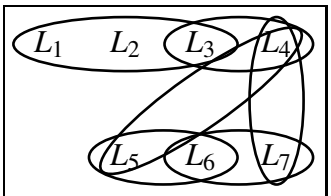


Figure 4: The interconnection hypergraph of $\mathsf{MaxStar}(L)$ of 2.12

(b) For $Y \subseteq X$, the *interconnection hypergraph* of Y defined by J, denoted $\mathsf{IntGraph}_Y(J)$, is the full subhypergraph of $\mathsf{IntGraph}_X(J)$ generated by Y.

The key result is the following.

3.3 Proposition *Assume that* $\mathsf{IntGraph}_X(J)$ *is acyclic.*

(a) *Every* $C \in X$*, and hence every* $\Gamma \in \mathsf{Ports}\langle X\rangle$*, is free in* X *with respect to* J*.*

(b) *If* $Y \subseteq X$ *is a closed set of vertices of* $\mathsf{IntGraph}_X(J)$*, then* Y *is closed in* X *with respect to* J*.*

PROOF OUTLINE: The idea is very simple. Assume that $\mathsf{IntGraph}_X(J)$ is acyclic, and choose any $C \in X$ and $M_C \in \mathsf{LDB}(C)$. For each $I \in J$, $\Gamma \in I \cap \mathsf{Ports}(C)$, and $\Gamma' \in I$ with $\Gamma \neq \Gamma'$, choose any $M_{\mathsf{SrcCpt}(\Gamma')} \in \mathsf{LDB}(\mathsf{SrcCpt}(\Gamma'))$ with $\mathring{\gamma}(M_C) = \mathring{\gamma}'(M_{\mathsf{SrcCpt}(\Gamma')})$. Since $\mathsf{IntGraph}_X(J)$ is acyclic, it is guaranteed that this construction will not result in any conflicts of other port matchings. Now choose another $C \in X$ from those which were included in the previous step, and repeat the process. The formal proof proceeds by induction. This establishes (a). Part (b) is almost identical, except that the starting point is a member of $\mathsf{LDB}(\mathsf{Schema}\langle Y, J\rangle)$ instead of a member of $\mathsf{LDB}(C)$. □

Thus, whenever $\mathsf{IntGraph}_X(J)$ is acyclic, the notion of compound component 2.10(i) is well defined. Furthermore, for all $Y \subseteq X$ with the property that Y is a closed set of vertices in $\mathsf{IntGraph}_X(J)$, the subcomponent $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle$, as given in 2.11(f), is well defined as well.

3.4 Attribute hypergraphs Since Berge acyclicity is not viewed as the appropriate one for the characterization of good schema construction in the relational context [9], it is worthwhile to present a short example to show how the notion of hypergraph for a relational schema differs from that of component interconnection.

Consider just the components L_3 and L_4 of 2.12. The attribute hypergraph for this pair, as well as the component hypergraph, are shown in Figure 5 to the right. It is easy to see that the attribute hypergraph is cyclic. Indeed, $(A_4, \{A_1, A_4, A_5\}, A_5, \{A_4, A_5, A_6, A_7, A_8\}, A_4)$ is a path from A_4 to itself. However, no such corresponding path occurs in the component hypergraph. The key distinction is that in the relational representation, as studied in [9], each attribute is a vertex of the hypergraph, while in the model of this paper, each component is a single vertex, regardless of how many attributes the relations of its schemata may have. In the relational representation, any port

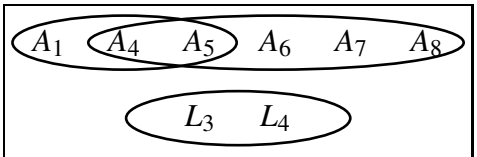


Figure 5: The attribute hypergraph (above) and interconnection hypergraph (below) for L_3 and L_4 of 2.12

view with more than one attribute will result in a Berge-cyclic hypergraph. Thus, the fundamental properties of the underlying hypergraphs in the two representations can be completely different.

4. Updates to components

The initial motivation for developing the ideas reported here was to study how updates to components propagate throughout the interconnection, and in particular to look for canonical ways to extend an update on a subcomponent to the entire network. While a complete presentation of these ideas must be deferred to a separate article, it is nevertheless instructive to illustrate the key ideas.

4.1 Liftings and feasible environments Let $Z \subseteq Y \subseteq X$, and let (M_1, M_2) be an update on $\mathsf{ProjSch}\langle Y, J\rangle$.

(a) (M_1, M_2) is *internal to* $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Z\rangle$ if $\mathring{\gamma}(M_1) = \mathring{\gamma}(M_2)$ for all $\Gamma \in \mathsf{ExtPorts}\langle Z, J\rangle$.

Thus, if (M_1, M_2) is internal, the update can be made without involving any components not in Z. If the desired update is not internal, then it must be *lifted* to a larger subcomponent.

(b) An update (M_1', M_2') on $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle$ is called an *internal lifting* of (M_1, M_2) to $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle$ if (M_1', M_2') is internal to $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle$ and $(\mathsf{NatProj}_{\langle X;J;Y\to Z\rangle})^{\circledcirc}(M_i') = M_i$ for $i \in \{1,2\}$.

(c) Y is called a *uniformly feasible environment for* (M_1, M_2) *relative to* J if for every $M_1' \in \mathsf{LDB}(\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle)$ with $(\mathsf{NatProj}_{\langle X;J;Y\to Z\rangle})^{\circledcirc}(M_1') = M_1$, there is an $M_2' \in \mathsf{LDB}(\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle)$ with the property that (M_1', M_2') is an internal lifting of (M_1, M_2) to $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle$.

Uniform feasibility is crucial because it says that the update (M_1, M_2) on $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Z\rangle$ has an internal lifting to $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle$ regardless of the actual state of that subcomponent.

4.2 Order and canonical updates There is one additional issue regarding the lifting of updates which is not recaptured in the formalism of 4.1. In general, there are many possible liftings of an update (M_1, M_2) from $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Z\rangle$ to $\mathsf{SubCpt}_{\langle X,J\rangle}\langle Y\rangle$, even in the case that Y is a least uniformly feasible environment for it. The further goal is to find the *canonical* such lifting — characterized by that which adds the least amount of additional information to the database. The problem of recapturing such minimality has received significant attention in the context of view updates, particularly in the context of logic programming [19]; however here the databases do not generally consist of sets of clauses, and so a different approach is necessary. For the component context developed here, the approach which is currently being developed is to regard the best lifting to be those defined by *free updates*. The details will be reported in a separate article, but the following example provides a glimpse of the main ideas.

4.3 Example — Canonical liftings of updates to components In this example, there is a total of eight components, including the two which were introduced in 2.1. The overall format is similar to that employed in 2.12; therefore, only significant differences will be elaborated. Information about the atomic components and the ports is given in Tables 4 and

5, respectively. The interconnection family $\mathbf{I}_1$ which will be used is shown below; Figure 6 illustrates this family in graphical form.

$$\mathbf{I}_1 = \{\{\Pi^{\mathbf{E}_1}_{B_2}, \Pi^{\mathbf{E}_2}_{B_2}\}, \{\Pi^{\mathbf{E}_3}_{A_2}, \Pi^{\mathbf{E}_4}_{A_2}\}, \{\Pi^{\mathbf{E}_4}_{A_4A_5}, \Pi^{\mathbf{E}_5}_{A_4A_5}\}, \{\Pi^{\mathbf{E}_5}_{A_6}, \Pi^{\mathbf{E}_6}_{A_6}, \Pi^{\mathbf{E}_7}_{A_6}\}, \{\Pi^{\mathbf{E}_6}_{A_7}, \Pi^{\mathbf{E}_8}_{A_7}\}\}$$

Comp. Name	Schema Name	Relations	Constraints	Ports
K_1	$\mathbf{E}_1$	$R_1[B_1B_2]$	$B_1 \rightarrow B_2$ $\mathsf{ForbidNulls(B_1B_2)}$	$\Pi^{\mathbf{E}_1}_{B_2}$
K_2	$\mathbf{E}_2$	$R_2[B_2B_3]$	$B_2 \rightarrow B_3$ $\mathsf{ForbidNulls(B_2B_3)}$	$\Pi^{\mathbf{E}_2}_{B_2}$
K_3	$\mathbf{E}_3$	$R_3[A_1A_2]$	$A_1 \rightarrow A_2$ $\mathsf{ReqNullTup(A_1A_2)}$ $\mathsf{NoPartNulls(A_1A_2)}$	$\Pi^{\mathbf{E}_3}_{A_2}$
K_4	$\mathbf{E}_4$	$R_4[A_2A_3A_4A_5]$ $S_4[B_1B_2]$	$A_2 \rightarrow A_3A_4A_5$ $R_4[A_3] \subseteq S_4[B_1]$ $\mathsf{NoPartNulls(A_2A_3A_4A_5)}$ $\mathsf{ReqNullTup(A_2A_3A_4A_5)}$ $\mathsf{ForbidNulls(B_2)}$	$\Pi^{\mathbf{E}_4}_{A_2}$ $\Pi^{\mathbf{E}_4}_{A_4A_5}$ $\Pi^{\mathbf{E}_4}_{B_2}$
K_5	$\mathbf{E}_5$	$R_5[A_4A_5A_6]$	$A_4A_5 \twoheadrightarrow A_6$ $\mathsf{NoPartNulls(A_4A_5)}$ $\mathsf{ReqNullTup(A_4A_5A_6)}$	$\Pi^{\mathbf{E}_5}_{A_4A_5}$ $\Pi^{\mathbf{E}_5}_{A_6}$
K_6	$\mathbf{E}_6$	$R_6[A_6A_7]$	$A_6 \twoheadrightarrow A_7$ $\mathsf{ReqNullTup(A_6A_7)}$	$\Pi^{\mathbf{E}_6}_{A_6}$ $\Pi^{\mathbf{E}_6}_{A_7}$
K_7	$\mathbf{E}_7$	$R_7[A_6A_8]$	$A_6 \twoheadrightarrow A_8$ $\mathsf{ReqNullTup(A_6A_8)}$	$\Pi^{\mathbf{E}_7}_{A_6}$
K_8	$\mathbf{E}_8$	$R_8[A_7A_9]$	$A_7 \twoheadrightarrow A_9$ $\mathsf{ReqNullTup(A_7A_9)}$	$\Pi^{\mathbf{E}_8}_{A_7}$ $\Pi^{\mathbf{E}_8}_{A_9}$

Table 4: The atomic components of the example of 4.3

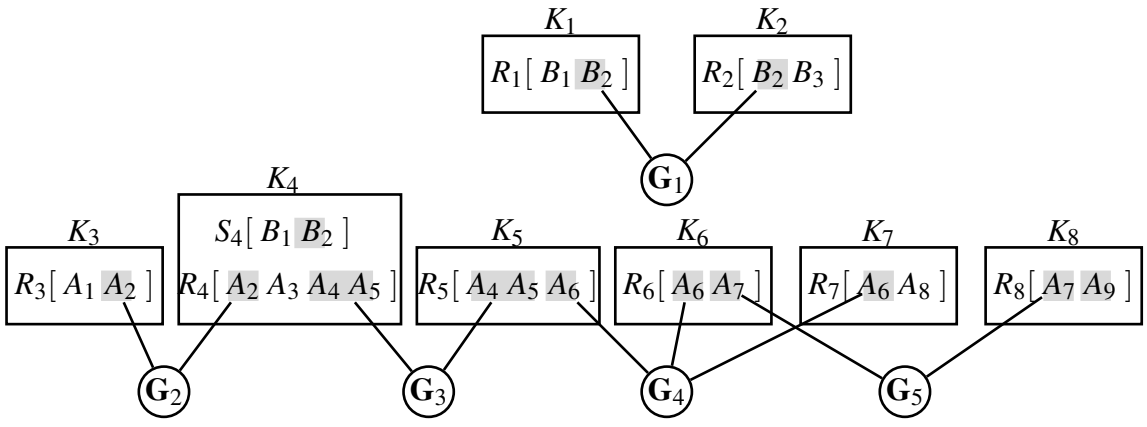


Figure 6: Graphical representation of the interconnection family $\mathbf{I}_1$

For simplicity, this example will be limited to the characterization of canonical least liftings for insertions. For the realization of such liftings to be nontrivial, it must be possible to insert partial information into relations, and to pad out the remainder with nulls. However,

Schema Name	Schema Relations	Schema Constraints	Associated Ports and View Mappings	
$\mathbf{G}_1$	$T_1[B_2]$	$\mathsf{ForbidNulls}(\mathsf{B}_2)$	$\Pi^{\mathbf{E}_1}_{B_2}$	$\pi^{\mathbf{E}_1}_{B_2} : \mathbf{E}_1 \to \mathbf{G}_1$
			$\Pi^{\mathbf{E}_2}_{B_2}$	$\pi^{\mathbf{E}_2}_{B_2} : \mathbf{E}_2 \to \mathbf{G}_1$
			$\Pi^{\mathbf{E}_4}_{B_2}$	$\pi^{\mathbf{E}_4}_{B_2} : \mathbf{E}_4 \to \mathbf{G}_1$
$\mathbf{G}_2$	$T_2[A_2]$		$\Pi^{\mathbf{E}_3}_{A_2}$	$\pi^{\mathbf{E}_3}_{A_2} : \mathbf{E}_3 \to \mathbf{G}_2$
			$\Pi^{\mathbf{E}_4}_{A_2}$	$\pi^{\mathbf{E}_4}_{A_2} : \mathbf{E}_4 \to \mathbf{G}_2$
$\mathbf{G}_3$	$T_3[A_4A_5]$	$\mathsf{NoPartNulls}(\mathsf{A}_4\mathsf{A}_5)$	$\Pi^{\mathbf{E}_4}_{A_4A_5}$	$\pi^{\mathbf{E}_4}_{A_4A_5} : \mathbf{E}_4 \to \mathbf{G}_3$
		$\mathsf{ReqNullTup}(\mathsf{A}_4\mathsf{A}_5)$	$\Pi^{\mathbf{E}_5}_{A_4A_5}$	$\pi^{\mathbf{E}_5}_{A_4A_5} : \mathbf{E}_5 \to \mathbf{G}_3$
$\mathbf{G}_4$	$T_4[A_6]$		$\Pi^{\mathbf{E}_5}_{A_6}$	$\pi^{\mathbf{E}_5}_{A_6} : \mathbf{E}_5 \to \mathbf{G}_4$
			$\Pi^{\mathbf{E}_6}_{A_6}$	$\pi^{\mathbf{E}_6}_{A_6} : \mathbf{E}_6 \to \mathbf{G}_4$
			$\Pi^{\mathbf{E}_7}_{A_6}$	$\pi^{\mathbf{E}_7}_{A_6} : \mathbf{E}_7 \to \mathbf{G}_4$
$\mathbf{G}_5$	$T_5[A_7]$		$\Pi^{\mathbf{E}_6}_{A_7}$	$\pi^{\mathbf{E}_6}_{A_6} : \mathbf{E}_6 \to \mathbf{G}_5$
			$\Pi^{\mathbf{E}_8}_{A_7}$	$\pi^{\mathbf{E}_8}_{A_6} : \mathbf{E}_8 \to \mathbf{G}_5$
$\mathbf{G}_6$	$T_6[A_9]$		$\Pi^{\mathbf{E}_8}_{A_9}$	$\pi^{\mathbf{E}_8}_{A_9} : \mathbf{E}_8 \to \mathbf{G}_6$

Table 5: The port schemata of the example of 4.3

this must be done in a systematic way, paying careful attention to constraints which specify where nulls may appear, and how functional dependencies (FDs) behave in their presence.

Nulls: There is a distinguished null marker, denoted $\mathfrak{n}$, with $\mathfrak{n} \in \mathsf{dom}(A)$ for every attribute A. This null marker is similar to the *placeholder* described in [18, Sec. 12.5.2]. There are three associated constraint types, each of which takes a list of attribute names as its argument. Since attribute names can occur in only one relation name of a schema, the semantics described below are unambiguous.

$\mathsf{ForbidNulls}(-)$: This constraint specifies that no tuple may have the value $\mathfrak{n}$ in *any* of the attribute positions listed.

$\mathsf{NoPartNulls}(-)$: (No partial nulls) This constraint specifies if a tuple has the value $\mathfrak{n}$ in one of the attribute positions listed, then it must have the value $\mathfrak{n}$ in every position listed.

$\mathsf{ReqNullTup}(-)$: (Require null tuple) This constraint specifies that a relation must have at least one tuple which has the value $\mathfrak{n}$ in every position listed in the argument.

Functional dependencies and nulls: In addition to the usual semantics for an FD $\mathbf{A} \to \mathbf{B}$, there are several variations which further specify the special way in which the null marker $\mathfrak{n}$ is handled. In all descriptions below, assume (without loss of generality) that $R[\mathbf{U}]$ is a relation scheme on attribute set $\mathbf{U}$ with $\mathbf{A}, \mathbf{B} \subseteq \mathbf{U}$, and that r is a tuple over $R[\mathbf{U}]$.

$\mathfrak{n}$-FDs: The relation r satisfies the $\mathfrak{n}$-FD $\mathbf{A} \xrightarrow{\mathfrak{n}} \mathbf{B}$ iff the following two conditions are satisfied.

Null extension: For every $t \in r$ there is a $t' \in r$ with $t'[\mathbf{A}] = t[\mathbf{A}]$ and $t'[B] = \mathfrak{n}$ for every $B \in \mathbf{B}$.

Quasi-functionality: If $t, t' \in r$ with the property that $t[\mathbf{A}] = t'[\mathbf{A}]$, then at least one of the following three conditions must hold: (i) $t = t'$; (ii) $t[B] = \mathfrak{n}$ for every $B \in \mathbf{B}$; or (iii) $t'[B] = \mathfrak{n}$ for every $B \in \mathbf{B}$.

In words, for an $\mathfrak{n}$-FD $\mathbf{A} \rightarrow \mathbf{B}$ to be satisfied, there may be at most two tuples over associated with each distinct value for $\mathbf{A}$, one with all nulls in $\mathbf{B}$ (required) and possibly one other which is not all null on $\mathbf{B}$.

There are three extensions of the notion of an $\mathfrak{n}$-FD, which are identified below.

Null preservation: The $\mathfrak{n}$-FD $\mathbf{A} \overset{\mathfrak{n}}{\rightarrow} \mathbf{B}$ is *null preserving* if whenever $t \in r$ with $t[A] = \mathfrak{n}$ for some $A \in \mathbf{A}$, then $t[B] = \mathfrak{n}$ for every $B \in \mathbf{B}$. The notation $\mathbf{A} \overset{\mathfrak{n}\Rightarrow}{\rightarrow} \mathbf{B}$ indicates that the $\mathfrak{n}$-FD $\mathbf{A} \overset{\mathfrak{n}}{\rightarrow} \mathbf{B}$ is null preserving.

Null reflection: The $\mathfrak{n}$-FD $\mathbf{A} \overset{\mathfrak{n}}{\rightarrow} \mathbf{B}$ is *null reflecting* if whenever If $t \in r$ with $t[B] = \mathfrak{n}$ for every $B \in \mathbf{B}$, then $t[A] = \mathfrak{n}$ for every $A \in \mathbf{A}$. The notation $\mathbf{A} \overset{\Leftarrow\mathfrak{n}}{\rightarrow} \mathbf{B}$ indicates that the $\mathfrak{n}$-FD $\mathbf{A} \overset{\mathfrak{n}}{\rightarrow} \mathbf{B}$ is null reflecting.

Simultaneous preservation and reflection: The notation $\mathbf{A} \overset{\mathfrak{n}\Leftrightarrow}{\rightarrow} \mathbf{B}$ indicates that the $\mathfrak{n}$-FD $\mathbf{A} \overset{\mathfrak{n}}{\rightarrow} \mathbf{B}$ is both null preserving and null reflecting.

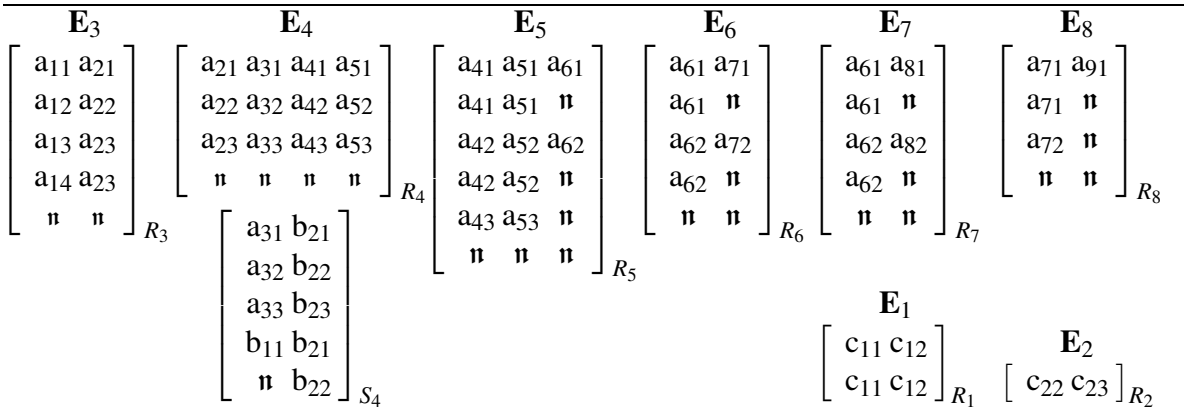


Figure 7: An example database for the interconnection I_1

Figure 7 shows an example of a consistent database for the interconnection $\mathbf{I}_1$. To illustrate the ideas of canonical updates, a few selected examples will now be considered.

Suppose that it is desired to insert the tuple $(\mathrm{a}_{15}, \mathrm{a}_{25})$ into the relation of $R_3[A_1A_2]$ of the schema $\mathbf{E}_3$ of K_3. The goal is to identify the canonical lifting of this update from K_3 to to a compound component under which this update may be supported naturally, without making any arbitrary choices.

First of all, it is important to clarify what is meant by an arbitrary choice. Consider a solution which lifts the update to the compound component $\mathsf{Cpt}\langle\{K_3, K_4\}, \{\Pi^{\mathbf{E}_3}_{A_2}, \Pi^{\mathbf{E}_4}_{A_2}\}\rangle$ by inserting the tuple $(\mathrm{a}_{25}, \mathrm{a}_{31}, \mathrm{a}_{41}, \mathrm{a}_{51})$ into $R_4[A_2A_3A_4A_5]$, leaving the state of all other components unchanged. This lifting makes an arbitrary choice for the values for attributes $A_3A_4A_5$; note that $(\mathrm{a}_{25}, \mathrm{a}_{32}, \mathrm{a}_{42}, \mathrm{a}_{52})$ or $(\mathrm{a}_{25}, \mathrm{a}_{33}, \mathrm{a}_{43}, \mathrm{a}_{53})$ would work just as well, and there is no reason to prefer one over the other. All involve parts of tuples which already occur in the database, and so make arbitrary semantic choices for the information associated with $(\mathrm{a}_{15}, \mathrm{a}_{25})$.

The canonical solution involves using completely new values in these positions, and so is *independent* of the other values in these relations. More precisely, let $\mathrm{a}_{35} \in \mathsf{dom}(A_3) \setminus \{\mathfrak{n}\}$, $\mathrm{a}_{45} \in \mathsf{dom}(A_4) \setminus \{\mathfrak{n}\}$, $\mathrm{a}_{55} \in \mathsf{dom}(A_5) \setminus \{\mathfrak{n}\}$, and $\mathrm{b}_{25} \in \mathsf{dom}(B_2) \setminus \{\mathfrak{n}\}$ be distinct domain values which have not already used in any relation. First insert $(\mathrm{a}_{25}, \mathrm{a}_{35}, \mathrm{a}_{45}, \mathrm{a}_{55})$ into the relation of

$R_4[A_2A_3A_4A_5]$ and $(\mathrm{a}_{35}, \mathrm{b}_{25})$ into the relation of $S_4[B_1B_2]$. Note that the second tuple is mandated by the inclusion dependency $R_4[A_3] \subseteq S_4[B_1]$. This further mandates an insertion into E_5, so the update must be extended to $\mathsf{Cpt}\langle\{K_3, K_4, K_5\}, \{\Pi^{\mathbf{E}_3}_{A_2}, \Pi^{\mathbf{E}_4}_{A_2}\}\{\Pi^{\mathbf{E}_4}_{A_4A_5}, \Pi^{\mathbf{E}_5}_{A_4A_5}\}\rangle$. Insert $(\mathrm{a}_{45}, \mathrm{a}_{55}, \mathfrak{n})$ into the relation of K_5. The use of the null is a recognition that the constraints do not force one to select a specific value for A_6. Note that this insertion does not violate the $\mathfrak{n}$-FD is $A_4A_5 \overset{\mathfrak{n}}{\Rightarrow} A_6$, so that the resulting state is legal. Furthermore, it does not make use of any arbitrary values which already occur in other relations and it is least, in the sense that no subset of the specified insertions will do, and there is no smaller set of components which will support such an insertion using new values.

One issue still remains; namely, that arbitrary choices for a_{35}, a_{45}, a_{55}, and b_{25} were made. Formally, this is reconciled by noting that all other such solutions are isomorphic, up to a renaming of the values. A solution to the update problem is thus not a single update, but rather an equivalence class of isomorphic updates. Replacing a_{35}, a_{45}, a_{55}, and b_{25} by different values which do not occur elsewhere, say a'_{35}, a'_{45}, a'_{55}, and b'_{25}, results in a solution which is structurally indistinguishable. The resulting solution is canonical, and identifies the natural scope for lifting an insertion to K_3 as $\{K_3, K_4, K_5\}$.

A formal justification of the canonicity of this solution is rooted in the construction of *free objects* [15, §31] [1, 8.22] over a suitable category of updates, and is beyond the scope of this paper; the details will appear in forthcoming report. However, it is possible to give an informal justification. The idea is that every other possible lifting of the desired update to K_3 can be obtained from the canonical one via a combination of adding additional tuples and forcing the "free" values in the canonical update to take on specific values. For example, the lifting which inserts the tuple $(\mathrm{a}_{25}, \mathrm{a}_{31}, \mathrm{a}_{41}, \mathrm{a}_{51})$ can be obtained from the canonical one identified above which inserts the tuples $(\mathrm{a}_{25}, \mathrm{a}_{35}, \mathrm{a}_{45}, \mathrm{a}_{55})$, $(\mathrm{a}_{45}, \mathrm{a}_{55}, \mathfrak{n})$, and $(\mathrm{a}_{35}, \mathrm{b}_{25})$ by mapping the "free" values a_{35}, a_{45}, a_{55}, and b_{25} to the existing values a_{31}, a_{41}, a_{51}, and b_{21}, respectively.

The case of deletions is handled similarly, although it turns out to be somewhat simpler, since there is no need to group isomorphic solutions (as no new values need be inserted).

5. Closing Remarks

5.1 Conclusions The foundations of a component-based model of database schemata, with the interconnection of components realized via communicating views, has been presented. Particular attention has been paid to the question of when such interconnections are well behaved, and a characterization in terms of the acyclicity of an underlying hypergraph has been presented. Furthermore, the way in which updates propagate through such components has been illustrated, although not fully formalized.

5.2 Further directions This paper is only a beginning, and many topics remain to be studied. Among them are the following.

Updates in the component-based framework: This work began as a study of updates in the context of components. Consequently, an important future direction is to complete the formalization of canonical liftings, as discussed in 4.3. A related direction is *update via cooperation*, in which the realization of a proposed update requires that other components be updated as well, not as a canonical update but rather as one chosen by a user who has update rights for that component. First results on this topic, including a formal model for the update process, are reported in [14]. Upon following the communication between ports that update by cooperation entails, it is possible to infer much about the

necessary workflow patterns behind such updates. Initial investigations on this latter topic are now being pursued.

Component-based HERM to relational design theory: It is a standard design technique to begin by modelling the enterprise using a flavor of ER, such as the HERM model, and then to translate that design to a relational schema [23, Ch. 10]. A future direction of this research is to extend this design theory to components; that is, to develop systematic tools for the translation of a HERM design based upon components, such as elaborated in [25], to a relational design which preserves the component structure, using the component model developed here for this final schema.

Rapprochement with the behavioral theory: As already noted in the introduction, the work presented here is motivated by the database-component model of Thalheim [25], which is in turn based upon the more general component model of Broy [6]. It is important to pursue an understanding of the degree to which these two models can be unified, and to understand their fundamental differences as well.

5.3 Remarks on the literature Nearly thirty years ago, Weber [26] suggested that modular design techniques could be applied fruitfully to database systems as well, although no detailed formalization was presented. In [7], a software tool for modular database design is presented. In that approach, the emphasis is not so much upon building systems by interconnecting components as it is in refining the design, specifically by combining and even redefining the so-called *conceptual modules* via *subsumption modules*. As such, it does not emphasize basic communication between components as does the framework presented here. Rather, it has much more of a software-engineering flavor. As noted in the introduction, the flavor of component-based modelling of database systems upon which this paper has its roots in the approach of Thalheim [24] [22] [25].

6. Acknowledgments

Much of this research was completed while the author was a visitor at the Information Systems Engineering Group at the University of Kiel during parts of 2005 and 2006. He is indebted to Bernhard Thalheim for suggesting the idea that his ideas of database components and the author's work on views and view updates could have a fruitful intersection, as well as for inviting him to work with his group on this problem. He is furthermore indebted to Bernhard, as well as to the other members of the Information Systems Engineering Group, particularly Hans-Joachim Klein and Peggy Schmidt, for many helpful discussions during the course of this work. Peggy Schmidt also read a preliminary draft of this paper and made numerous suggestions to improve the presentation.

References

[1] ADÁMEK, J., HERRLICH, H., AND STRECKER, G. *Abstract and Concrete Categories*. Wiley-Interscience, 1990.

[2] ARBIB, M. A. *Theories of Abstract Automata*. Prentice-Hall, 1969.

[3] BANCILHON, F., AND SPYRATOS, N. Update semantics of relational views. *ACM Trans. Database Systems 6* (1981), 557–575.

[4] BERGE, C. *Graphes et Hypergraphes*. Dunod, 1970.

[5] BROY, M. A logical basis for modular software and systems engineering. In *SOFSEM* (1998), B. Rovan, Ed., vol. 1521 of *Lecture Notes in Computer Science*, Springer, pp. 19–35.

[6] BROY, M. Model-driven architecture-centric engineering of (embedded) software intensive systems: modeling theories and architectural milestones. *Innovations Syst. Softw. Eng. 6* (2007), in press.

[7] CASANOVA, M. A., FURTADO, A. L., AND TUCHERMAN, L. A software tool for modular database design. *ACM Trans. Database Systems 16*, 2 (1991), 209–234.

[8] CHEN, P. P. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Systems 1*, 1 (1976), 9–36.

[9] FAGIN, R. Degrees of acyclicity for hypergraphs and relational database schemes. *J. Assoc. Comp. Mach. 30*, 3 (1983), 514–550.

[10] HEATH, I. J. Unacceptable file opearations in a relational data base. In *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access, and Control* (1971), pp. 19–33.

[11] HEGNER, S. J. Characterization of desirable properties of general database decompositions. *Ann. Math. Art. Intell. 7* (1993), 129–195.

[12] HEGNER, S. J. An order-based theory of updates for database views. *Ann. Math. Art. Intell. 40* (2004), 63–125.

[13] HEGNER, S. J. The complexity of embedded axiomatization for a class of closed database views. *Ann. Math. Art. Intell. 46* (2006), 38–97.

[14] HEGNER, S. J., AND SCHMIDT, P. Update support for database views via cooperation. submitted for publication, 2007.

[15] HERRLICH, H., AND STRECKER, G. E. *Category Theory*. Allyn and Bacon, 1973.

[16] KATZ, R. H., AND BORRIELLO, G. *Contemporary Logic Design*, second ed. Pearson Education, 2005.

[17] KRUEGER, C. W. Software reuse. *ACM Comput. Surveys 24*, 2 (1992), 131–183.

[18] MAIER, D. *The Theory of Relational Databases*. Computer Science Press, 1983.

[19] MAYOL, E., AND TENIENTE, E. A survey of current methods for integrity constraint maintenance and view updating. In *Proc. ER '99 Workshops, Paris, Nov. 15-18, 1999* (1999), vol. 1727 of *Springer LNCS*, Springer-Verlag.

[20] PAREDAENS, J., DE BRA, P., GYSSENS, M., AND VAN GUCHT, D. *The Structure of the Relational Database Model*. Springer-Verlag, 1989.

[21] RISSANEN, J. Independent components of relations. *ACM Trans. Database Systems 2*, 4 (1977), 317–325.

[22] SCHMIDT, P., AND THALHEIM, B. Component-based modeling of huge databases. In *Advances in Databases and Information Systems: 8th East European Conference, ADBIS 2004, Budapest, Hungary, September 22-25, 2004, Proceedings* (2004), A. Benczúr, J. Demetrovics, and G. Gottlob, Eds., no. 3255 in Lecture Notes in Computer Science, Springer-Verlag, pp. 113–128.

[23] THALHEIM, B. *Entity-Relationship Modeling*. Springer-Verlag, 2000.

[24] THALHEIM, B. Database component ware. In *ADC* (2003), K.-D. Schewe and X. Zhou, Eds., vol. 17 of *CRPIT*, Australian Computer Society, pp. 13–26.

[25] THALHEIM, B. Component development and construction for database design. *Data Knowl. Eng. 54*, 1 (2005), 77–95.

[26] WEBER, H. Modularity in data base system design: A software engineering view of data base systems. In *Issues in Data Base Management, Proceedings of the 4th VLDB, September 13-15, 1978, West Berlin, Germany* (1978), H. Weber and A. I. Wasserman, Eds., North-Holland, pp. 65–91.