TECHNICAL REPORT NO. 522

Formalization of Multi-level Zachman Frameworks

by

Richard Martin, Tinwisle Corp., Bloomington IN

Edward L. Robertson

April 1999

COMPUTER SCIENCE DEPARTMENT

INDIANA UNIVERSITY

Bloomington, Indiana 47405-4101

Formalization of Multi-level Zachman Frameworks

Richard Martin
Tinwisle Corp.
205 N College Ave
Bloomington IN 47404
tinwisle@cs.indiana.edu

Edward Robertson
Computer Science Dept.
Indiana University
Bloomington IN 47405
robertson@cs.indiana.edu

I keep six honest serving men
(They taught me all I knew):
Their names are What and Why and When
And How and Where and Who.
Rudyard Kipling, 1902

Abstract

This paper presents a formal specification that structures a framework in Zachman's Enterprise Architecture Model as a tree of frames. Each frame is the familiar *role* $\times$ *interrogative* grid. Each cell of the grid contains, recursively, addition frames down to leaves. The formalism also provides mechanisms for connecting framework components down the recursive levels of detail and down the grid categories of roles. While the tree of frames expresses the structure of a framework model, these connections express the framework's meaning.

On top of the basic formalism, additional mechanisms facilitate customizing of framework meta-models, viewing frameworks from different perspectives, and imposing constraints on the framework model. The paper also discusses implications of the formalism on the ways in which Zachman framework models are discovered.

1. Introduction

This paper explores the formalism inherent in the structure and semantics of Zachman frameworks. The ultimate goal is, of course, to formalize the framework development process; formalizing the artifacts produced by such development is an essential precursor to formalizing the process. The Zachman Enterprise Architecture Model has matured to a point where formalization can enable further significant development and has been applied in ways which suggest natural formalisms.

In his *IBM Systems Journal* articles, John Zachman describes a Framework for Information Systems which defines an architectural structure for organizing institutional artifacts[18,12]. The expression of these artifacts reveals the interconnections and abstractions among structural elements and thereby facilitates the understanding of operational systems. In its latest form, "The Zachman Framework for Enterprise Architecture" presents an architectural model that is currently seeing more frequent use. The extensiveness and flexibility of the Zachman approach is demonstrated in a variety of applications from the design and construction of data warehouses[6],to use in the aerospace industry[9],to a GIS used in a regional planning context[2]. The role of Zachman in academics is shown in the development of course syllabi[14]and text books[15]. Descriptions of several reference architectures are influenced by Zachman, *e.g.* [17], and others seek to address similar enterprise issues in a structural manner as well[16].

While we view the growth of explicit reference architectures as a positive trend, we are disturbed by the deployment of implicit architecture in systems that target manufacturing processes and enterprise systems. A presumptive, implicit architecture underlies each of the ERP systems which are bringing large scale homogeneity to automated applications. However, the understanding of complex systems often becomes lost in the detail of implementation while important levels of enterprise management derived from more abstract concepts of enterprise operation stagnate[7]. The presumptive ERP architecture becomes the enterprise architecture even when it does not fit the enterprise. A complete enterprise architecture must align business practices with business goals.

Since our audience is likely to have a programming background we also feel it is necessary to distinguish the Zachman Framework approach from the application programming frameworks discussed in recent literature. In [8], Johnson characterizes the latter, programming-oriented frameworks:

> Frameworks are firmly in the middle of reuse technology. They are more abstract and flexible than components, but more concrete and easier to use than a pure design (but less flexible and less likely to be applicable). Although they can be thought of as a more concrete form of a pattern, frameworks are more like techniques that reuse both design and code, such as application generators and templates. Patterns are illustrated by programs, but a framework is a program.

While frameworks in the sense described by Johnson might be found in a particular Zachman Framework instance, the Zachman Framework abstraction operates at a level higher than Johnson's design in describing an enterprise. For both uses of the term "framework", the analogy to a structural metaphor is central. A framework represents the underlying structural members that support a realization. In the Zachman case, that realization is a set of artifacts that describe an enterprise at various levels of abstraction and that links those abstractions in a coherent manner.

Frameworks are models of some underlying reality constrained by our points of view. As such, models are merely a tool employed to objectively understand that reality. The purpose of our paper is to add a measure of definition for a tool being used increasingly often in business management endeavors.

Those of us who develop models and must communicate those models with our clientele are well aware of the importance and the difficulty of distinguishing model and instance: abstracting Sue and Joe into *Employee* is not too hard but abstracting "employees work for a single department" as a *business rule* is often difficult. The step from model to metamodel, for example from *Employee* to *entity*, is more difficult in that it introduces two levels of abstraction; one of the strengths of the Entity-Relationship meta-model is that its extreme conciseness limits the complexity of climbing down two levels to the instance level. This paper takes that abstraction one level further, from *entities* and *business rules* to framework components, dealing with meta-meta-models. Thus this paper must deal with four levels of abstraction, from ground instance to meta-meta-models. In an attempt to simplify this situation, our formalization jumps directly from meta-meta-model to model. The meta-model is not omitted; it is expressed in conventions for naming (and hence accessing) frame components rather than actual objects within a framework. That is, the modeling process we propose (with our fixed meta-meta-model) first describes the meta-model by specifying a set of names and then constructs a set of model components obeying these meta-model naming conventions.[1]

One characteristic of the framework concept set forth by Zachman is recursion, reflecting a top-down analysis of the world being modeled. Yet, like many situations in which recursive constructs could be of use, the tendency is to avoid that approach and the "excruciating levels of detail", as Zachman would say, that must be managed. As described in [6], recursion is an important part of Zachman's approach and we believe an examination of its formal properties to be prescriptive.

Therefore a major tenet – and challenge – in our formalization is that a framework is a recursive artifact. In particular, a framework is defined as a structure composed of multiple frames. These frames are labeled in a top-down manner, but the semantics reflected by the framework are revealed from the bottom up. In particular, we examine recursion with respect to model structure and not to the application of frameworks.

The formalism we propose in this paper is more general than that initially proposed by Zachman, in that we do not limit the row and column names or even specify the number of rows or columns. This follows a growing trend in which other modelers modify the rows or columns from Zachman's original proposal[1,3,5,11]. Our goal is not to limit the uses of Zachman's idea but rather it is to explore and enhance its expressive power.

This paper begins with a "bare bones" formalization of frames and frameworks – first merely the structural aspects of frameworks (syntax) and then the meaning (semantics). It then describes how this formalism provides tools for building a framework reflecting the real world with constraints and views. This material is brought together in a detailed, specific example formalizing a slice of an enterprise architecture. Next, we discuss using entire frameworks as situated in the world and manipulations of these frameworks. Finally, we summarize what we have achieved and what next steps that indicates.

2. Framework Structure

1. Since we are defining a methodology for meta-meta-models, this methodology could itself be considered as specifying a meta-meta-meta-model. Although meta^3 seems confusing, it is much more confusing to mix-up levels of abstraction. Because clients in general do not understand levels of abstraction, modelers must.

A *framework* is constructed from *frames*. This section first introduces the components used to describe frames. Then it discusses how a framework is composed of individual frames. Finally, it gives a formal definition of these frames. Once we have a good understanding of the structure, the next section will then discuss how to represent meaning within such a structure.

A word about fonts used in this paper: the vocabulary used in describing a framework will be written in a small sans-serif font. Later, when we talk about populating a framework with particular frame components, those components will be identified using a `small fixed-width font`. Frames and frameworks are denoted in gothic font, as in $\mathfrak{E}$ or $\mathfrak{F}$, while sets of various kinds are typically written in calligraphic font, as in $\mathcal{D}$ or $\mathcal{EIC}$.

component	abstraction level	font	examples
framework	meta-meta-model	gothic	$\mathfrak{E}$, $\mathfrak{F}$, $\mathfrak{G}$
classes of frames, abbreviations, members of **R**, **I**	meta-model	small sans-serif	Entity, Business_Rule, Relationship, owner, who
frames	model	small courier	`Employee`, `works_in`
data	representation of the "world"	roman, with quotes	"Sue", "\$345" "July 4, 1776"
mutable sets	all	calligraphic	$\mathcal{D}$, $\mathcal{EIC}$, $\mathcal{P}$ATHS
fixed sets		bold	**R**, **I**
variables		italic	*r, i, d*
path labels		lower case Greek	α, β, γ

Table 1: Notations and Fonts

2.1. Component Vocabulary

A framework's structure is always defined in terms of three sets: **R** and **I**, which are fixed, and $\mathcal{D}$, which varies with the use context. Names constructed from the three sets, called *path labels*, identify the individual frames of a framework.

R $\{\mathsf{r}_1, \cdots, \mathsf{r}_n\}$, a fixed set of *roles*

R is ordered and, for $r \in \mathbf{R}$, r' indicates the successor of r in the order on **R**. That is, $\mathsf{r}_i{}'$ is r_{i+1}.

Typically n for **R** is 3 and a meaningful vocabulary is used for members of **R**. Common vocabularies for **R** (in their typical order) are {"conceptual", "logical", "physical"}, {"owner", "designer", "builder"}, and {"enterprise", "system", "technology"}. The vocabulary {"equity", "ecology", "economy"} has also been proposed.

Understanding that **R** is ordered is perhaps Zachman's most valuable contribution to enterprise modeling. His understanding was based on observation that complex engineering products are described in artifacts which are successively less abstract, each structuring its successors via constraints.

The differences among the perspectives are driven by the constraints and the cumulative nature of these constraints dictates the sequence of the perspectives.[6]

One common convention for **R** places design (the row of the role "designer") after ownership ("owner") and before implementation ("builder"). The role of design then is to mediate between the goals of ownership and the reality of implementation by applying the constraints of rational judgment and expectation. An ownership goal that cannot be met is no more useful than is achieving an implementation objective that is not an ownership goal. The business alignment within an enterprise achieved by using the Zachman approach is the consequence of the ordering imposed on **R**. No other ordering provides such opportunity for mediation based upon rules of design and logic. While the three elements of **R** as typically used are sometimes linked by three connections to indicate contention between the roles[4], Zachman's insight properly breaks the longest of these linkages to require design in a context and thereby provides architectural structure capable of supporting the conceptual, logical, and physical modeling requirements of the enterprise.

R is extended with new elements $\ominus$ and $\oplus$, respectively before and after all elements of **R**. In Zachman terminology, $\ominus$ corresponds to "context" and $\oplus$ to "out of context". Separating **R** from $\ominus$ and $\oplus$ recognizes that the top and bottom rows of a typical Zachman frame picture are boundaries, which are important for passing information, rather than sites for analysis and further decomposition. Many proposed framework variations do not make this distinction and in fact give $\ominus$ and $\oplus$ names such as "planner" and "sub-contractor" respectively.

The second important set is

I a fixed set of *interrogatives*.

Typical vocabularies for **I** are {"what", "how", "where", "who", "when", "why"}, {"data", "function", "network", "people", "time", "motivation"}, or { "things", "activities", "places", "roles", "events", "rules"}.

The set **I** provides a classification of content into universal categories long ago recognized as essential elements of the enterprise story. Each element of **I** is an important part of the whole that a framework makes explicit.

When a frame is presented graphically, it is almost always as a table with columns corresponding to elements of **I** and rows corresponding to elements of **R**. Thus, for $i \in \mathbf{I}$ or $r \in \mathbf{R}$, we will write of "column i" or "row r", respectively. Figure 1 is the typical way we draw an individual column, using a traditional set of role names; the distinction between cells that merely communicate and those that can be

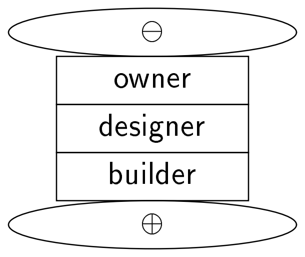

Figure 1: A Column

further decomposed is emphasized by drawing the former as ellipses and the later as rectangles. The fact that the ellipses are wider than the rectangles indicates that they are shared across all columns. This shared characteristic of $\ominus$ and $\oplus$ is critical to the operation of our model because it assures the lexical scoping necessary to both identify and resolve information content among artifacts residing in a framework.

The third set, as with any textually constructed model, is a set of names.

$\mathcal{D}$	a finite set, the *descriptors*

Mathematical usage would call this set the "domain", but we have discovered this usage of "domain" is too easily confused with the use of the word in relational databases. Typically members of $\mathcal{D}$ would have a printable representation as character strings.

2.2. Labels and Paths

Labels bind frames to subframes and thus characterize local structure. Sequences of labels bind to paths followed while traversing through successive subframes and thus characterize the global structure.

When we discern the structure of a frame, we first form a grid of cells indexed by $\mathbf{R} \times \mathbf{I}$ and then designate components of a cell according to names found in $\mathcal{D}$. Thus each frame is connected to its subframes by edges labeled with triples of the form $\langle \mathsf{r},\ \mathsf{i},\ \mathsf{d} \rangle$, naturally called *edge labels*. A succession of edges forms a *path* and thus a sequence of edge labels is a *path label*. Because an edge is also a path of length 1, we use $\mathcal{L}_1$ to denote the set of edge labels.

$\mathcal{L}_1$	$\{\langle r, i, d \rangle : r \in \mathbf{R},\ i \in \mathbf{I},\ \&\ d \in \mathcal{D}\}$; paths of length 1
$\mathcal{L}$	a set of *path labels* defined by the BNF $\mathcal{L} := \epsilon \mid \mathcal{L}\,\mathcal{L}_1$, where ϵ denotes the empty (or root) path.

In this paper, path labels are typically denoted by lower-case Greek letters – α, β, γ, *etc.* Looking ahead, a crucial definition will be $\mathfrak{F}_\alpha$, the frame reached by the path labeled α.

In addition to path labels (sequences of edge labels) that label frames, we will also need sequences of the form $\alpha\, d$, where $\alpha \in \mathcal{L}$ and $d \in \mathcal{D}$. This does not designate a (sub)frame and hence it is not a "label" in our nomenclature. Instead, $\alpha\, d$ makes specific the use of d in the frame labeled by α (that is, frame $\mathfrak{F}_\alpha$).

2.3. Frames and Frameworks

A framework is constructed recursively from two kinds of frames. A frame's descriptors, which constrain the scope of consideration and allow a consistent treatment of artifacts within that frame, provide links to subframes.

$\mathfrak{F}$	a *framework* is a finite set of *frames* $\{\mathfrak{F}_\alpha : \alpha \in \mathcal{L}\}$, where a frame $\mathfrak{F}_\alpha$ is either a *branch* frame or a *leaf* frame, both of which are characterized below.
$\mathcal{PATHS}_{\mathfrak{F}}$	$\{\alpha : \mathfrak{F}_\alpha$ is defined$\}$ for a framework $\mathfrak{F}$. This set is typically written as only "$\mathcal{PATHS}$", with $\mathfrak{F}$ understood from context.

The labels of $\mathfrak{F}$ (*i.e.* $\mathcal{PATHS}$) must correspond to a tree structure, in which branch frames are internal nodes and leaf frames are terminal nodes. That is, all intervening nodes between a leaf and the root must be labeled by elements of $\mathcal{PATHS}$. Formally this is stated: if $\alpha\ell \in \mathcal{PATHS}$, for $\alpha \in \mathcal{L}$ and $\ell \in \mathcal{L}_1$, then also $\alpha \in \mathcal{PATHS}$. Thus the actual structure of $\mathfrak{F}$ is a labeled graph with nodes in $\{\mathfrak{F}_\alpha : \alpha \in \mathcal{L}\}$ and edges labeled by $\mathcal{L}_1$. There is no constraint that the actual frame $\mathfrak{F}_\alpha$ is distinct from $\mathfrak{F}_\beta$. That is, paths α and β may lead to the same frame instance. However, we require that the graph be acyclic; formally: $\mathfrak{F}_{\alpha\beta}$ is distinct from $\mathfrak{F}_\alpha$ unless β is the empty label ϵ.

With this notation, $\mathfrak{F}_\epsilon$ is the "root" frame, that is the broad, top-level frame that presents the entire framework to the outside world.

We now define the structure (syntax) of the two types of frames. The semantics are defined later, when details are given for $\mathcal{IC}_\alpha$, Φ_α, *etc.* Observe that both kinds of frames have $\mathcal{IC}$ and $\mathcal{OC}$ components. This is necessary for the definition of Φ.

A *branch frame* $\mathfrak{F}_\alpha$ is characterized by the 4-tuple:

$\mathcal{IC}_\alpha \subseteq \mathcal{D}$	input ("context")
$\mathcal{OC}_\alpha \subseteq \mathcal{D}$	output ("out of context")
$\mathcal{SF}_\alpha : \mathbf{R} \times \mathbf{I} \times \mathcal{D} \rightarrow \mathfrak{F} \cup \boldsymbol{\mathcal{VF}}$	subframe mapping. $\boldsymbol{\mathcal{VF}}$ indicates *virtual frames* which are discussed below
$\Phi_\alpha \subseteq$ *outputs* $\times$ *inputs* (details below)	parameter passing relationship

A structure generated with two levels of frames is illustrated in Figure 2. The entire framework is $\mathfrak{F}$. The top frame (the artifact drawn with long lines) is $\mathfrak{F}_\epsilon$. $\mathfrak{F}_\epsilon$has several subframes, three of which appear larger and are shown with their descriptors; these are $\mathfrak{F}_{\langle \mathsf{r}_1,\mathsf{i}_1,\texttt{Department}\rangle}$, $\mathfrak{F}_{\langle \mathsf{r}_1,\mathsf{i}_1,\texttt{Employee}\rangle}$, and $\mathfrak{F}_{\langle \mathsf{r}_2,\mathsf{i}_2,\texttt{Hire}\rangle}$. With one conventional interpretation of $\mathbf{R}$ and $\mathbf{I}$, this nomenclature is shorthand for $\mathfrak{F}_{\langle \mathsf{owner},\mathsf{data},\texttt{Department}\rangle}$, $\mathfrak{F}_{\langle \mathsf{owner},\mathsf{data},\texttt{Employee}\rangle}$, and $\mathfrak{F}_{\langle \mathsf{designer},\mathsf{function},\texttt{Hire}\rangle}$ respectively.

Syntactically $\boldsymbol{\mathcal{VF}}$ is just $\mathcal{L}$, but actually we want it to be $\mathcal{PATHS}_{\mathfrak{F}}$.[2] Using paths in $\boldsymbol{\mathcal{VF}}$ connects framework components together with a form of indirection. That is, when $\mathcal{SF}_\alpha(r, i, d) = \beta \in Paths$, this signifies a connection to $\mathfrak{F}_\beta$. The benefit of this can be seen

2. Defining $\boldsymbol{\mathcal{VF}}$ to be $\mathcal{PATHS}$ would introduce a circularity in the definition of $\mathfrak{F}$, since $\mathcal{PATHS}$ is defined in terms of $\mathfrak{F}$. Thus $\boldsymbol{\mathcal{VF}}$ comprises "symbolic links" in the Unix sense.

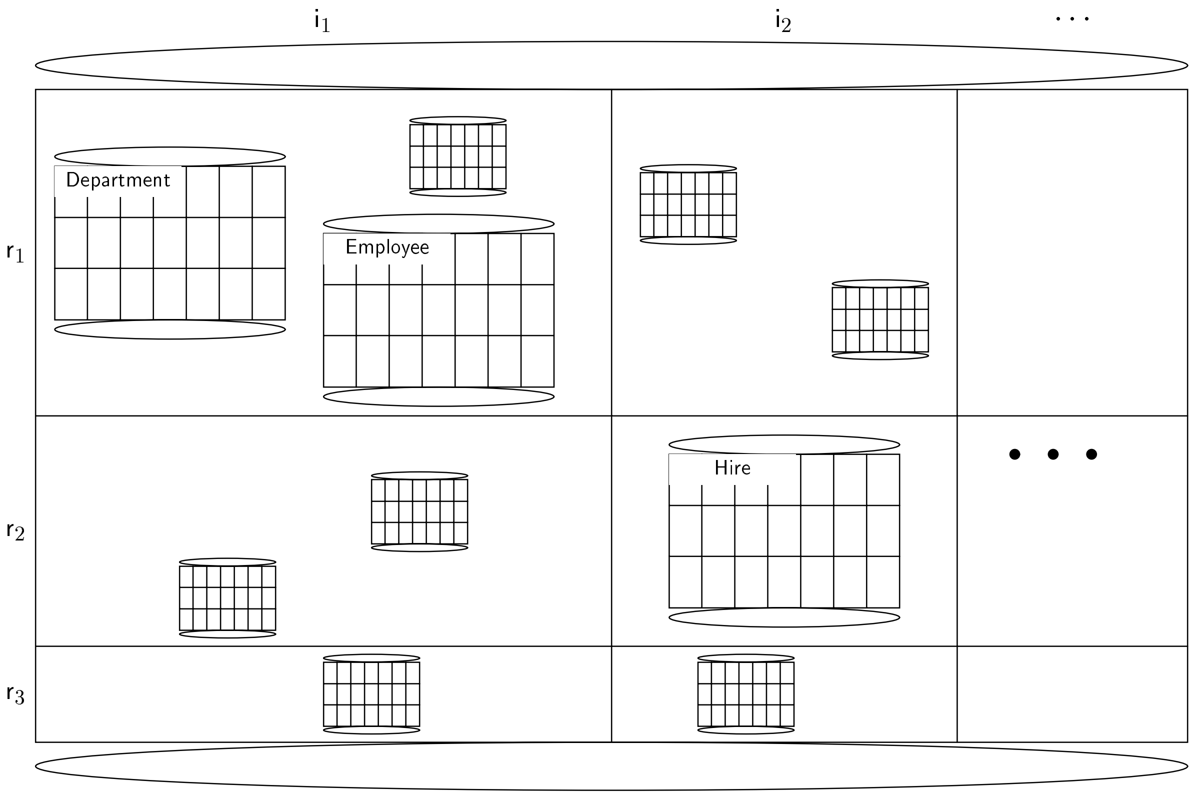

Figure 2: Two Levels of a Framework

in the example of Figure 2: Department and Employee correspond to Entities of an ER model but a Relationship works_for would require some way of explicitly linking to these Entities – $\mathfrak{VF}$ supplies that link. Virtual frames are not absolutely necessary; equivalent results can be obtained without them, albeit with some additional effort (both in developing and in applying the formalism). Moreover, connecting via a virtual frame link indicates a use of, rather than a definition of, a frame component.

A *leaf frame* $\mathfrak{F}_\alpha$ is characterized by the 3-tuple:

$\mathcal{IC}_\alpha \subseteq \mathcal{D}$	input (context)
$\mathcal{OC}_\alpha \subseteq \mathcal{D}$	output (out of context)
$\mathfrak{S}_\alpha$ (details below)	operations of the system modeled by the framework

One leaf frame which commonly occurs at the end of many paths is the completely empty frame $\langle\emptyset,\ \emptyset,\ \emptyset\rangle$, denoted by $\mathfrak{E}$. There is no technical requirement that a frame be present and hence no specific need for $\mathfrak{E}$. However, $\mathfrak{E}$ can be used to indicate that a particular subframe is specifically known to be empty rather than merely overlooked.

A "dot" notation for subcomponents may also be used. Thus, for $r \in \mathbf{R}$, $i \in \mathbf{I}$, and $d \in \mathcal{D}$, the notations "$\mathfrak{F}_{\alpha\langle r,i,d\rangle}$", "$\mathcal{SF}_\alpha(r,i,d)$", and "$\mathfrak{F}_\alpha.\mathcal{SF}(r,i,d)$" are equivalent, provided $\alpha\langle r,i,d\rangle \in \mathcal{P}\mathcal{ATHS}$. We will often use $\mathcal{IC}$, $\mathcal{OC}$, Φ, *etc.* generically for $\mathcal{IC}_\alpha$, $\mathcal{OC}_\alpha$,

Φ_α, *etc.* respectively when α is in fact not specified.

The set of descriptors (names) $\mathcal{D}$ is global to an entire framework $\mathfrak{F}$. Because it is sometimes useful to be able to talk about only those descriptors actually used in a specific frame, we define

$$\mathcal{D}_\alpha \quad = \{d \in \mathcal{D} : \exists r \in \mathbf{R}, \exists i \in \mathbf{I} \quad (\mathcal{SF}_\alpha(r,i,d) \text{ is defined})\}$$

It is expected that $\mathcal{IC}_\alpha \cup \mathcal{OC}_\alpha \supseteq \mathcal{D}_\alpha$, that is, the decomposition defined by $\mathfrak{F}_\alpha$ is entirely accessible at the boundaries of that frame.

3. Adding Meaning to a Framework Structure

Although both kinds of frames present similar interfaces to the outside, internally they are substantially different. This difference is even more apparent when the meanings of the various internal components is considered.

3.1. Branch Frame Details

The $\mathcal{IC}$, $\mathcal{OC}$, and $\mathcal{SF}$ components reflect the recursive structure of a framework and as such are relatively simple and straightforward. In particular, $\mathcal{IC}$ and $\mathcal{OC}$ together correspond to the formal parameters of a procedure declaration. $\mathcal{SF}$ maps to either a subframe or a path – recall the latter allows a form of indirection. The burden of describing how information passes from subframe to subframe falls on the relationship Φ. Because this information passing occurs in complex ways, the definition of Φ is more difficult. Note that Φ defines only interconnections; all decompositions are expressed in subframes and actual real-world facts appear only in leaf frames (motivation for this comes later). Think of Φ as describing a plugboard or wiring patch panel.

In order to set the stage for the definition of Φ, $\mathcal{IC}$ and $\mathcal{OC}$ must be "extended". The $\mathcal{IC}$'s and $\mathcal{OC}$'s of a frame's subframes are collected across each entire row; this happens on a row-by-row basis because the information "flow" through a frame is also row-by-row. Formally, for $r \in \mathbf{R}$,

$$\mathcal{EIC}_{\alpha,r} \quad = \quad \{e : \exists i \in \mathbf{I} \ \exists d \in \mathcal{D} \ (e \in \mathcal{IC}_{\alpha\langle r,i,d\rangle})\}$$

$$\mathcal{EOC}_{\alpha,r} \quad = \quad \{e : \exists i \in \mathbf{I}, \exists d \in \mathcal{D}(e \in \mathcal{OC}_{\alpha\langle r,i,d\rangle})\}$$

Furthermore, as a notational convenience, which allows us to treat boundary transitions (from "context" and to "out of context") with the same mechanism used for internal transitions, define

$$\mathcal{EOC}_{\alpha,\ominus} \quad = \mathcal{IC}_\alpha$$

$$\mathcal{EIC}_{\alpha,\oplus} \quad = \mathcal{OC}_\alpha$$

Finally we may formally define Φ (recall r' is the successor of r in $\mathbf{R}$):

$$\Phi_\alpha \qquad \subseteq \bigcup_{r \in \{\ominus\} \cup \mathbf{R}} \left(\mathcal{EOC}_{\alpha,r} \times \mathcal{EIC}_{\alpha,r'} \right)$$

Note that the definition of Φ is a union over products, not a product of unions. Each component of this union is the interface between two successive roles. The notation $r \in \{\ominus\} \cup \mathbf{R}$ selects a single role interface, between r and r', and constrains Φ to work across that interface. There are two boundary conditions at the first and last elements of the order on $\mathbf{R}$: (1) when r (in the expression defining Φ_α) is $\ominus$ and r' is r_1, the first element in $\mathbf{R}$; (2) when r is r_n, the last element in $\mathbf{R}$, and r' is $\oplus$. In the first case, the above notation allows us to substitute $\mathcal{IC}_\alpha$ for $\mathcal{EOC}_{\alpha,\ominus}$, so that $\mathcal{EOC}_{\alpha,\ominus} \times \mathcal{EIC}_{\alpha,r_1}$ becomes $\mathcal{IC}_\alpha \times \mathcal{EIC}_{\alpha,r_1}$. Thus the inputs to a frame are connected into its subframes by the same mechanisms that connects outputs from one subframe to be inputs to the next. In the second case, the product is $\mathcal{EOC}_{\alpha,r_n} \times \mathcal{OC}_\alpha$. Figure 3 illustrates how Φ is built out of its various pieces. The rows are labeled $\mathsf{r}_1, \cdots, \mathsf{r}_n$ to indicate the generality of $\mathbf{R}$.

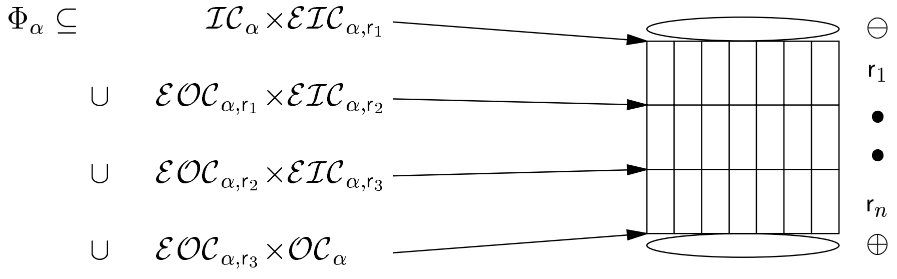
$\Phi_\alpha \subseteq$
$\mathcal{IC}_\alpha \times \mathcal{EIC}_{\alpha,\mathsf{r}_1}$
$\cup$ $\mathcal{EOC}_{\alpha,\mathsf{r}_1} \times \mathcal{EIC}_{\alpha,\mathsf{r}_2}$
$\cup$ $\mathcal{EOC}_{\alpha,\mathsf{r}_2} \times \mathcal{EIC}_{\alpha,\mathsf{r}_3}$
$\cup$ $\mathcal{EOC}_{\alpha,\mathsf{r}_3} \times \mathcal{OC}_\alpha$
$\ominus$
r_1
r_n
$\oplus$

Figure 3: How Φ Relates to a Frame

The net impact of the definition of Φ as discussed above is that Φ is defined so that it operates across columns but only from one row to the next. Figure 4 extends the example of Figure 2 with *just two* of many potential connections defined by Φ.

The question naturally arises about how Φ differs from explicit connections (via $\mathcal{SF}$) or virtual frame links. Explicit connections are very structured, yielding the tree labeled by $\mathcal{P}\textsc{aths}$. $\boldsymbol{\mathcal{VF}}$ is oblivious of structure, connecting arbitrary components (recall, $\boldsymbol{\mathcal{VF}}$ provides a form of indirection). Both explicit and virtual links connect to a single frame. Φ connections, on the other hand, are rich yet modularly structured. The connections are rich because, since any member d of $\mathcal{IC}_\beta$ or $\mathcal{OC}_\beta$ can occur in many ways in links to subcomponents of $\mathfrak{F}_\beta$, using d in Φ establishes a connection that "fans out" to all those subcomponents. The connections reflect modularity because this "fan out" is entirely within the control of the single frame $\mathfrak{F}_\beta$.

3.2. Leaf Frame Details

Frameworks allow us to model the structure of a world of discourse. It is still necessary

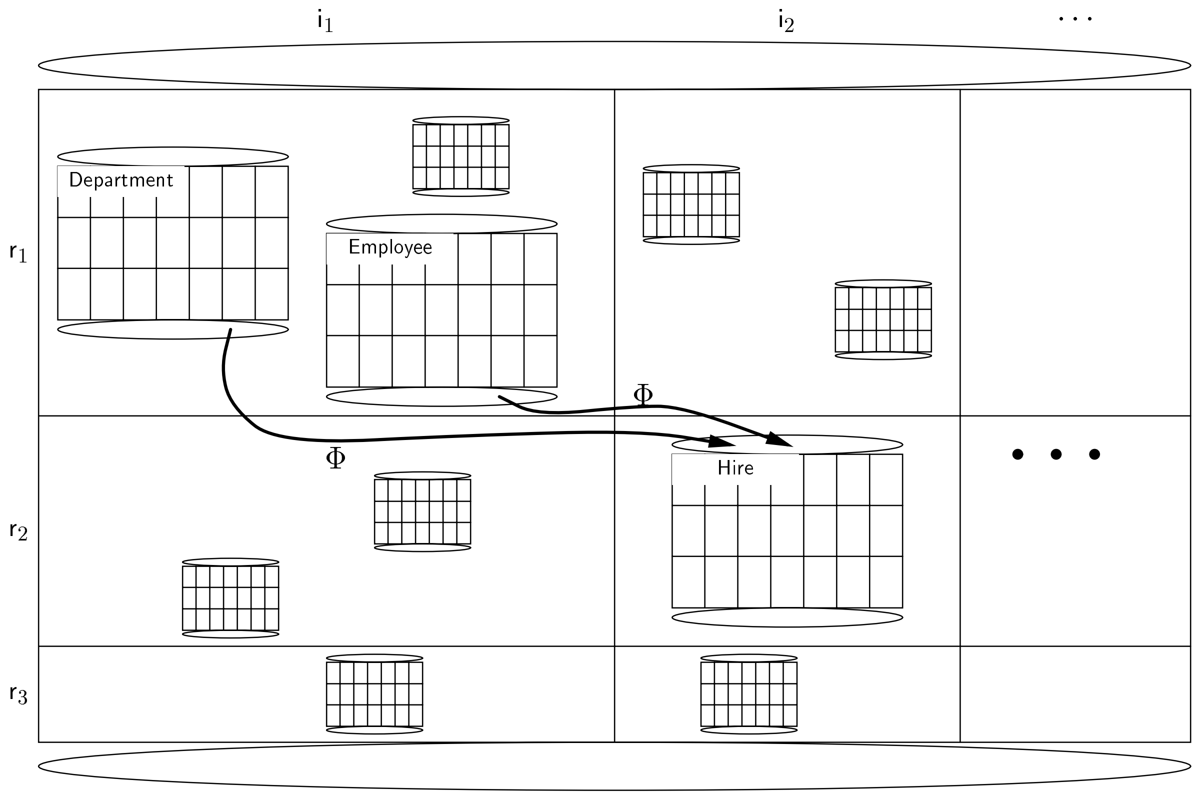

Figure 4: Illustration of Φ

to populate this structure with the items from the world. Thus we distinguish between the *model* and the *instance*. This distinction is exactly paralleled in such areas as relational databases, which commonly appear in the "data" column of a frame. The model, or "schema", of a relation is a structure, such as

```
Employee:  SSN, name, birthdate, ...
```

An instance of that relation is a set of tuples (records) which the relation contains

"123-45-6789"	"Bob Smith"	"7 Aug 1954"	
"478-22-1836"	"Sue Davis"	"13 May 1962"	···

Note that a given relation has one fixed schema but infinitely many possible instances. The dichotomy exists between `TYPE` and `VAR` declarations in some programming languages or between an object class and objects of that class in object modeling.

When connecting to the real world, (such connections occur at leaf frames), the things that can be managed are named by members of $\mathcal{IC}$ and $\mathcal{OC}$. Because we may wish to deal with individual items or with sets of items, we use the notation *Types* to specify how we deal with instances.

$$Types_\alpha \;= \mathcal{IC}_\alpha \cup \{\text{“ SET OF } \lambda\text{”} : \lambda \in \mathcal{IC}_\alpha\} \cup \mathcal{OC}_\alpha \cup \{\text{“ SET OF } \lambda\text{”} : \lambda \in \mathcal{OC}_\alpha\}$$

Recall that $\mathcal{S}_\alpha$ was said to indicate the "operation of the system modeled by the framework"; its definition completes the definition of leaf frames.

$$\mathcal{S}_\alpha \qquad \subseteq \bigcup_{n \in \mathbb{N}} Types_\alpha{}^n$$

The $Types$ are the primary linkage between a framework as a model and an instance of that framework. When a framework is instantiated to reveal the underlying "real world", the signatures[3] in the various $\mathcal{S}_\alpha$ become the operations which do the work of the system. The $Types$ are thus analogous to table schema in a relational database system, as opposed to the records in those tables which are the instances.

For example, a purchase order will have a number of item_price values which must be summed to obtain the total_price. This requires a relationship with signature (SET OF item_price) $\times$total_price.

It is common to talk about the transformation that occurs between two cells of a single column in a Zachman framework. In our characterization, transformation (and other operations) occur only in leaf frames. A "within-cell transformation" is the aggregate of leaf frames encountered via recursion within a cell for which the context is taken from the cells above. Some might find it helpful to think of the within-cell transformation in terms of a process moving down recursive levels and returning back to the top-level frame where the artifact is presented. The transformation occurs as the result of specific leaf frames encountered in the traversal down and back up the paths in the framework. Traversal back up aggregates artifact parts and resolves identities. Our model it thus consistent with more conventional use.

We use $\mathfrak{I}$ for any leaf frame which merely connects its input to its output – that is, $\mathcal{IC} = \mathcal{OC}$ and instantiations of $\mathcal{S}$ are always the identity relation on the relevant set. Formally, such usage is ambiguous because there is an identity frame for each distinct $\mathcal{IC}$, $\mathcal{OC}$ pair but practically it is easy to handle.

4. Building on the Frameworks

4.1. Annotations

It is highly desirable to be able to associate arbitrary descriptive information with frames. This is most easily accomplished by adding to each frame $\mathcal{F}_\alpha$ an *annotation* function

3. A signature is an ordered list of the types that participate in a function or relation. Thus one signature describes the "input/output format" of a family of relations. The types of a procedure's formal parameters (including returned value) are the signature of that procedure. For example, addition, subtraction, multiplication and division have the same signature, namely two numbers in and one returned. In a relational database, the signature of a relation is just its schema.

$$\mathcal{A} \quad : \mathcal{D} \rightarrow \mathcal{STRINGS}$$

where $\mathcal{STRINGS}$ is obviously a finite set of strings over some alphabet. This function is added to both kinds of frames. That is, a branch frame $\mathfrak{F}_\alpha$ is now the 5-tuple $\langle \mathcal{IC}_\alpha, \mathcal{OC}_\alpha, \mathcal{SF}_\alpha, \Phi_\alpha, \mathcal{A}_\alpha \rangle$ and a corresponding leaf frame is $\langle \mathcal{IC}_\alpha, \mathcal{OC}_\alpha, \mathfrak{S}_\alpha, \mathcal{A}_\alpha \rangle$.

Annotations work best when some particular member of $\mathcal{D}$ is used consistently across all frames of a framework. For example, require that $\mathcal{A}_\alpha(\texttt{framename})$ be defined for any non-empty frame $\mathfrak{F}_\alpha$. In the example of Figure 2, saying that $\mathcal{A}_{\langle \mathsf{r}_1, \mathsf{i}_1, \texttt{Department} \rangle}(\texttt{framename})$ is "Department Entity" provides a humanly intelligible name to a component whose technical name is the path $\langle \mathsf{r}_1, \mathsf{i}_1, \texttt{Department} \rangle$.

4.2. Abbreviations

We have already noted that a path may be abbreviated by a single name. There are several other conventions that help the presentation and comprehension of a framework.

First, unique and meaningful names may be given to each cell of the $\mathbf{R} \times \mathbf{I}$ grid that structures a frame. For example, in Zachman's "Enterprise Architecture", the conceptual,data cell is called the "Semantic Model" and the conceptual,motivation cell is the "Business Plan".

Second, there are names associated with the components of a cell labeled by an $\mathbf{R}$, $\mathbf{I}$ pair. Thus an object within the conceptual,data cell (that is, any object with path name $\langle$conceptual, data, —$\rangle$) is given the name "Entity" or "Relationship", while "Semantic Model" designates the entire collection of objects in that cell. This formalizes the discussion of section 2.3 in conjunction with Figure 2; that is, an Entity of the ER model is labeled "Entity" in the framework.

Contrasting these first two points, a name in the first case abbreviates an entire frame, while in the second case it abbreviates only part of a frame's component's name. Summarizing the example, the "Semantic Model" frame contains "Entities" and "Relationships". Each of these "Entities" and "Relationships" is modeled by a (sub)frame at the next level down.

Third, the $\mathbf{R}$ and $\mathbf{I}$ components of a collection of paths are often separated from the $\mathcal{D}$ components. This is especially powerful when combined with the naming of cells in the $\mathbf{R} \times \mathbf{I}$ grid and is particularly useful when there are many paths whose labels all have the same form, as is true with "$\langle$conceptual, data, —$\rangle\langle$conceptual, data, —$\rangle$". We call the conceptual,data cell "Entity" at the first level and "attribute" at the second. Thus the statement "Relation.attribute includes Employee.SSN and Employee.name" means that there is a frame with label "$\langle$conceptual, data, Employee$\rangle\langle$conceptual, data, SSN$\rangle$" and one with "$\langle$conceptual, data, Employee$\rangle\langle$conceptual, data, name$\rangle$". This example is further elaborated in section 5.

Fourth, if a name in $\mathcal{D}$ appears in only one frame, then entire paths are abbreviated. Say that d belongs uniquely to $\mathcal{D}_\alpha$, then $\langle r, i, d \rangle$ can be used in place of $\alpha\langle r, i, d \rangle$. Again this works well with other abbreviations. For example, if Employee appears only at the top

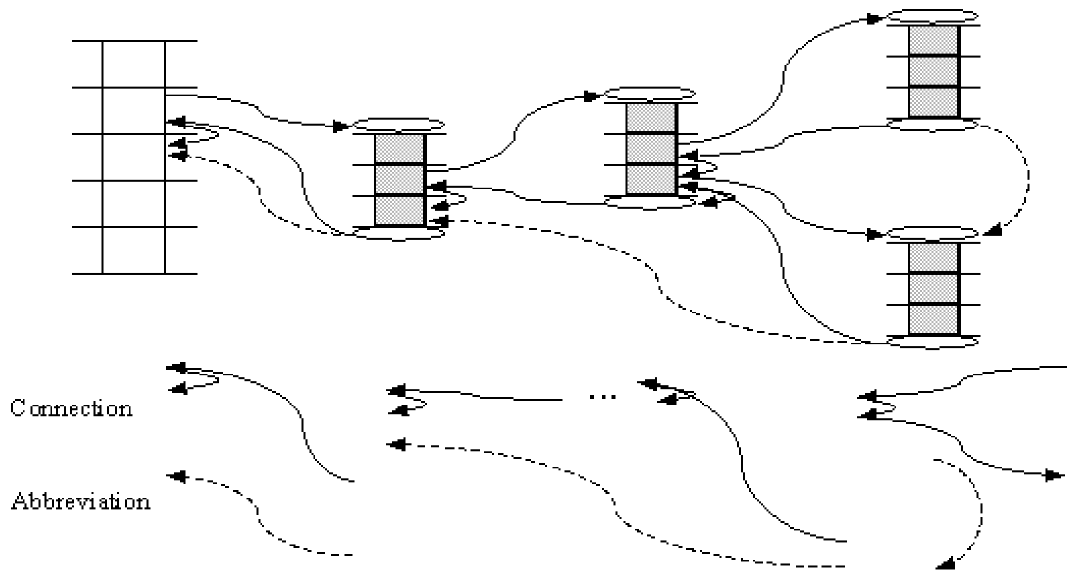

Figure 5: Successive Refinements of Frames with Abbreviations

level, then `Entity Employee` indicates a unique subframe.

Fifth, a succession of $\mathcal{SF}$ and Φ connections is often abbreviated by a single connection. Figure 5 illustrates this kind of abbreviation, where solid arrows represent connections from the underlying $\mathcal{SF}$ and Φ relations (the later as $\curvearrowleft$ only) and dashed lines represent abbreviations. The bottom of the figure breaks out each abbreviation and the underlying Φ relations that it abbreviates. This is especially important "coming back up" since no explicit paths exist in that direction.

The entirety of the abbreviations used in a framework effectively define the architecture of that framework. Zachman's original work was aptly called an architecture because it provided names ("Semantic Model", "Business Process Model", "Logistics Network", $\cdots$) for each cell. Providing an abbreviation "Entity" for objects in the Semantic Model cell, for example, clearly directs our analysis. This abbreviation structure is "meta" in the sense that it characterizes how we define and work with a framework structure, not how the framework describes the structure and functionality of the real world. If a framework is limited to only paths for which there are abbreviations (or initial segments of such paths), then these abbreviations do indeed characterize the abstract structure of the framework.

4.3. Framework Constraints

One of the major benefits of having a formal model is the ability to posit explicit, precise constraints. At this point we only wish to suggest the rich variety and powerful capabilities of constraints. It is not practical to mandate any universal constraints because there are so many possibilities for deploying frameworks, although any framework not obeying Ξ (discussed below) seems at best implausible.

There are two ways in which constraints arise in developing a framework model: *a priori* and derived. An *a priori* constraint is discovered and explicitly declared during the modeling process. This is similar to the way foreign key constraints are handled in a relational database implementation. Derived constraints appear after the model has been developed as the result of some constraint discovery process.

The most important *a priori* constraint at the framework structure level insures that an input connected to multiple sources receive only consistent information. Formally, it first requires the definition of

$$\Xi_\alpha \qquad \subseteq \bigcup_{r \in \mathbf{R}} \mathcal{EOC}_{\alpha,r} \times \mathcal{EOC}_{\alpha,r}$$

which specifies identities between outputs. It is further required that Ξ_α be a true equivalence relation.[4] The constraint is completed with the requirement that, for all x, y, and z, if $x\,\Phi_\alpha\,z$ and $y\,\Phi_\alpha\,z$, then $x\,\Xi_\alpha\,y$. Figure 6 illustrates this situation.

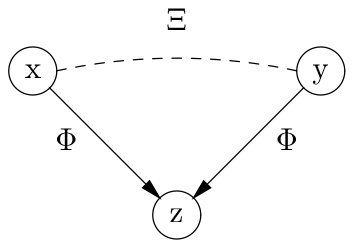

Figure 6: Ξ and Φ

Another approach is to treat Ξ as a derived constraint, as the transitive closure of the base equivalence $x\,\Xi_\alpha\,y$ iff $x\,\Phi_\alpha\,z$ & $y\,\Phi_\alpha\,z$, for some z. Although the mathematics for this is easy to write, the actual calculation of Ξ is obscured by the presence of abbreviations. This suggests that an automated tool for framework manipulation should provide for, perhaps even require, explicit declaration of abbreviations. Such declarations would also make manifest the framework's architecture.

The equivalence relation Ξ and its interaction with Φ illustrate one important family of constraints – those which govern the way in which Φ connects components of the framework. Another family governs the structure of frames within a framework. We have already discussed, in section 4.2, the use of abbreviations to regularize the construction of path names. A constraint that requires that all path names be constructed (or constructable) according to a specific set of abbreviations makes those abbreviations the meta-model of the framework (as discussed in the Introduction).

In the same family, the constraint that a framework $\mathfrak{F}$ is *thorough* holds provided that, whenever a branch frame $\mathfrak{F}_\alpha$ is defined, $\mathcal{SF}_\alpha(r,i,d)$ is defined for all $r \in \mathbf{R}$, $i \in \mathbf{I}$, and $d \in \mathcal{D}_\alpha$.[5]

In order to fully capture the world being modeled, it is expected that there will be additional *ad hoc* constraints. Such constraints often arise in the context of views.

4.4. Views

4. An equivalence relation is reflexive ($x\,\Xi\,x$), symmetric ($x\,\Xi\,y$ implies $y\,\Xi\,x$), and transitive ($x\,\Xi\,y$ and $y\,\Xi\,z$ imply $x\,\Xi\,z$).
5. This is Zachman's "excruciating detail."

Views embody the expressive power of our formalism similar to the power they provide in relational databases. Since our frames are fundamental and constrained information, it must be the views derived from a framework that provide semantic content for broader consumption. Once again the analogy is with relational database systems, which have a well-defined view mechanism.

The motivation for "viewing", that is for rearranging the structure of a framework, has three parts. First is the desire to focus on particular elements of a framework; this is especially important in explaining a design. The second is that different domain knowledge implies different structural prespectives. The third has to do with developing and validating a framework structure and it contents. This third motivation is discussed is Section 7.

Since the structure of a framework is provided by the labels in $\mathcal{PATHS}$, a view is specified in terms of path label rewriting. That is, members of $\mathcal{PATHS}$ are "pattern matched" against sequences of edge-label-like triples, with variables (r, i, d, r_i, *etc*) and constants ("Owner", "Who", "Department", *etc*). The values of variables and (other) constants then define other path labels.

For example, the rewriting rule

$$\langle r, i, \text{“A”}\rangle\langle\text{“Owner”}, i, d\rangle \Longrightarrow \langle r, i, d\rangle$$

takes a framework $\mathfrak{G}$ into a view framework $\check{\mathfrak{G}}$ such that $\check{\mathfrak{G}}_\beta = \mathfrak{G}_\alpha$ when α is of the form $\langle r, i, \text{“A”}\rangle\langle\text{“Owner”}, i, d\rangle$ (the left-hand side of the above rewriting) and β is of the form $\langle r, i, d\rangle$ (the right-hand side). We will extend this $\check{}$ notation as intitively appropriate, as in $\check{\mathcal{D}}$, $\check{\Phi}_{\check{\beta}}$, *etc.* Figure 7 shows the effect of this rewriting. On the left, the figure shows a "column slice" through two-plus levels of framework $\mathfrak{G}$ and on the right, a comparable slice through $\check{\mathfrak{G}}$. This example illustrates how a view may be used to conceptually rearrange a framework.

Unfortunately, a mapping such as $\mathfrak{G}$ into $\check{\mathfrak{G}}$ above can lead to ambiguities. Say we had another rule with "A" replaced by "B" and that two distinct paths $\langle \text{Owner, i, A}\rangle\langle\text{Owner, i, a}\rangle$ and $\langle \text{Owner, i, B}\rangle\langle\text{Owner, i, a}\rangle$ in $\mathfrak{G}$ map to distinct frames. Then the image path $\langle\text{Owner, i, a}\rangle$ in $\check{\mathfrak{G}}$ would be ambiguous in refering to both of these frames.

Another serious difficulty is the fact that the rules may map one single path ambigously. That is, a single ζ may get mapped into $\check{\zeta}_1$, $\check{\zeta}_2$, *etc.* by different rules. This latter ambiguity may be simply accepted, delt with in an *ad hoc* manner, or resolved by ordering the rules and accepting only the first rewrite of any given path.

An example of the use of this framework-view technique occured at Butler International, which provided its sales force with a "Role Classification" framework showing, for each cell within the Framework, the job title of the person responsible for developing that cell [10]. In the corporate framework view, all of these people, their job titles, and the associated work descriptions would be found in the who column of the top-level frame. Recall that sub-frames are not necessarily unique even though their path names are unique.

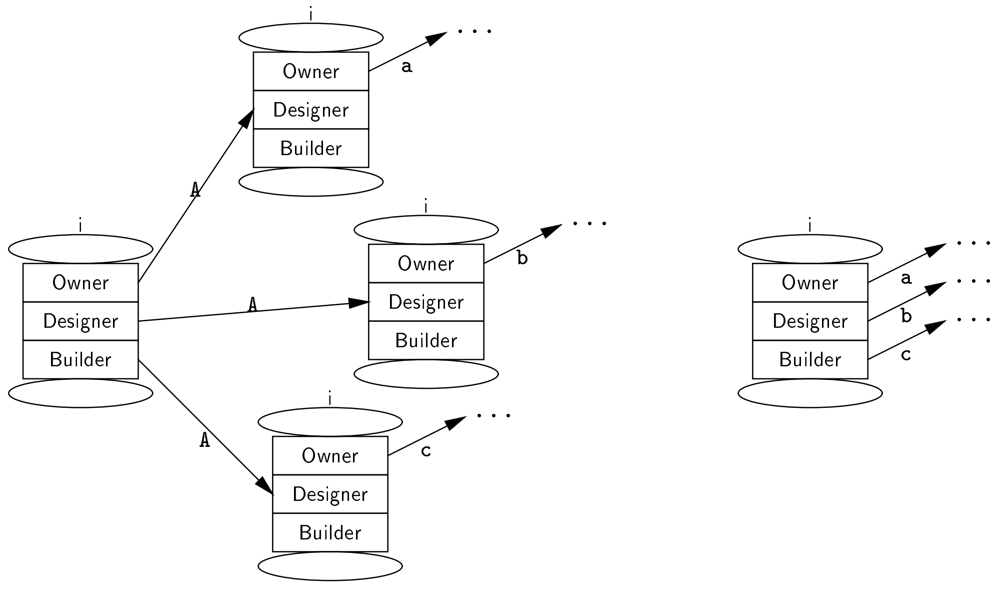

Figure 7: A slice of $\mathcal{G}$ and its view $\check{\mathcal{G}}$. Arrows are labeled by values in $\mathcal{D}$.

Also recall that Φ can provide mappings across $\mathbf{I}$. This kind of view re-arranges the leaves along existing paths to focus on the expression of particular structural patterns. Figure 8 illustrates the re-arrangement used at Butler International; where resulting view on the right is formed from the framework on the left by the rewriting rule

$$\langle r, \textsf{Where}, d_1\rangle\langle\textsf{Owner}, \textsf{Who}, d_2\rangle \Longrightarrow \langle r, \text{``}\textsf{Who}\text{''}, d_1\, d_2\rangle$$

The same figure also indicates, in gray, the existance of other possible views obtained by replacing Owner in the above rule by Designer and Builder. These three views could be called "HR Title Structure" (conceptual), "HR Job Description" (logical), and "HR Staffing Plan" (physical).

A view such as that used by Butler International can also facilitate the specification of constraints. With all job titles collected in one place, simply define a set $\mathcal{ValidJobTitles}$ and constrain $\check{\mathcal{D}}_\epsilon \subseteq \mathcal{ValidJobTitles}$.

Another kind of label rewriting reorders the edge components of paths. For example, edges of the form $\ell_1\ell_2$ could be rewritten as $\ell_2\ell_1$. While the rewritings discussed earlier are conservative, in the sense that no *new* frames are introduced, reordering almost certainly requires the creation of additonal frames not present in the original framework. Consider, for example, a framework is $\mathcal{K}$ and its view $\check{\mathcal{K}}$. For the above rewriting to apply, $\mathcal{K}$ must have frames $\mathcal{K}_{\ell_1}$ and $\mathcal{K}_{\ell_1\ell_2}$ but $\check{\mathcal{K}}$ is unlikely to have a frame labeled ℓ_2. The three distinct views in Figure 8 could be combined into a single framework with such a reordering.

The rewriting rules discussed above (implementing projection and reordering) all take one path into another path. But it is also conceivable to have rules which combine several

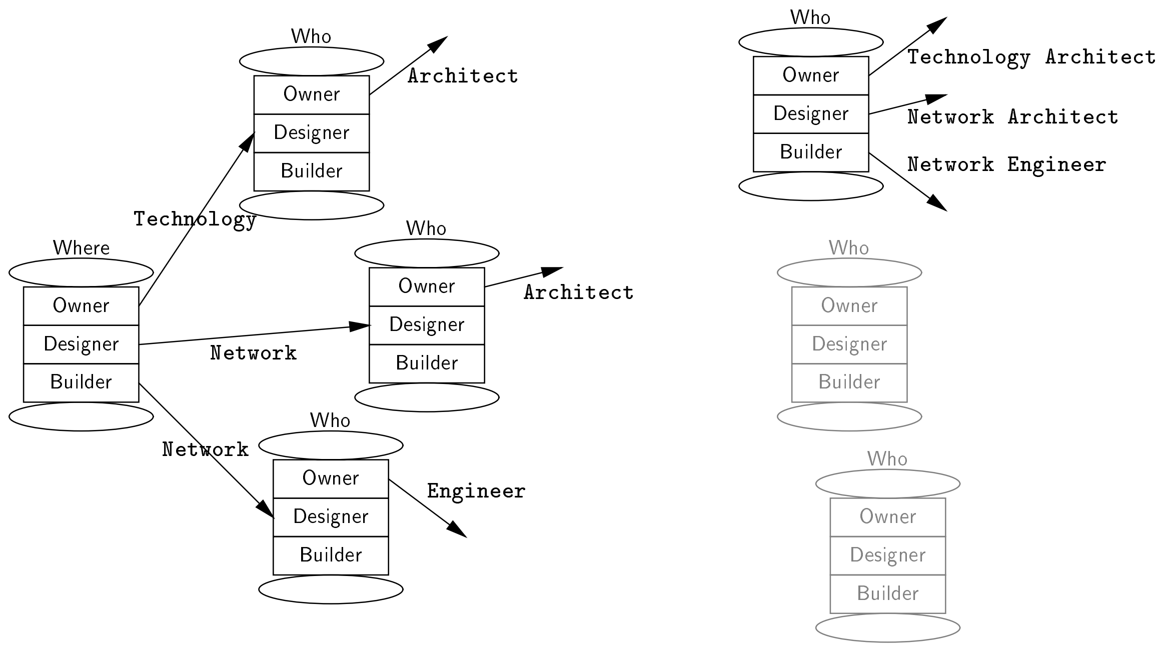

Figure 8: Use of a view at Butler International

paths into a new path. The semantics of such rules use the “unification” mechanism that appear in Horn clauses of logic programming (or related database languages such as DATALOG[13]). Such unification semantics also allow the definition of transitive closure and hence provide a way to completely flatten a framework structure. Since the entire framework consists of many branches and leaves and intertwined paths, it is sometimes advantageous to “flatten” the framework for presentation and discussion. We observe that most prior use of the Zachman approach has been rather “flat”.

While rewriting rules (and certain caution) provide for a generally straightforward restructuring of a framework into a view, the effect on semantics is more awkward to specify. First, the definition of the Φ_α, which was deliberately kept local to facilitate modularity and reuse of framework components, must first be cast into absolute, full-path names. That is, define:

$$\Phi_{\text{abs}} = \bigcup_{\alpha \in \mathcal{PATHS}_{\mathfrak{F}}} \{\langle \alpha x, \alpha y \rangle : \langle x, y \rangle \in \Phi_\alpha\}$$

Second, for each $\langle \zeta, \rho \rangle \in \Phi_{\text{abs}}$ for which there exist rewritings $\check{\zeta}$ and $\check{\rho}$, let $\langle \check{\zeta}, \check{\rho} \rangle \in \check{\Phi}$. The resulting $\check{\Phi}$ may require yet further manipulation, in that the pairs $\langle \check{\zeta}, \check{\rho} \rangle$ may violate the levelwise constraint imposed on the original Φ_α. This may be accomodated by restricting the rewriting rules (or their application) or simply accepted as a consequence of the fact that views sometimes distort.

Handling virtual frames is much easier. A virtual frame link β appearing in $\mathfrak{G}$ is simply replaced in each corresponding occurrance in $\check{\mathfrak{G}}$ by $\check{\beta}$. However, as noted above, the rewrite rules may yield ambiguous results.

As with relational views, complexly defined views may have update anomalies.

The view should preserve the relative ordering on $\mathbf{R}$ determined by the underlying frames and their inter-connections.[6]

Obviously, identities of the Ξ form and differences in the level of abstraction among elements of the result must be carefully managed. This caution becomes an imperative when using views to develop and validate framework structures.

A view is *coherent* if any collapsing happens consistently across all paths. That is, when a path $\langle r_1, i_1, d_1\rangle, \ldots, \langle r_n, i_n, d_n\rangle$ collapses, then so do all paths $\langle s_1, j_1, d_1\rangle, \ldots, \langle s_n, j_n, d_n\rangle$. For example, in the context of a shipping system, only the weight of an item may be significant, so `Item.weight` is always collapsed to `Weight`.

5. Example

In this section, we show how the framework formalism can be used to express the Entity–Relationship model and carry this model forward to the logical design of the database tables. We are intentionally basing this example on aspects of the Enterprise Architecture that already have their own well-defined and widely-accepted formal specifications. For example, there is difficulty formalizing columns such as motivation because the target lacks such a specification.

We already noted above that the conceptual,data cell in the top-level frame provides the enterprise's Semantic Model and that we may identify any frame labeled ⟨conceptual, data, ---⟩ as an entity or relationship.[7] In fact, we have abbreviated any such triple as ⟨Entity, ---⟩ or ⟨Relationship, ---⟩. One such entity is named `Employee`; that is, the frame $\mathfrak{F}_{\langle \text{conceptual,data,Employee}\rangle}$ represents the employee entity, containing (at least) the attributes of that entity. Formally, $\mathfrak{F}_{\langle \text{conceptual,data,Employee}\rangle}.\mathcal{IC} \supseteq \{\text{SSN, name, hiredate, skill}, \cdots\}$. Or, in abbreviation, Entity `Employee` includes `SSN`, `name`, `hiredate`, `skill`, *etc.*

At its simplest manifestation, the frame for entity named `Employee` could be just an identity frame $\mathfrak{I}$,[8] so all it does is define the decomposition of `Employee` into `Employee.SSN`, `Employee.name`, *etc.*

Each attribute of an entity in the Semantic Model is then carried forward to be an attribute of a relation in the Logical Data Model. That is `SSN` as it appears in $\mathfrak{F}_{\langle \text{conceptual,data,Employee}\rangle}$

6. The categorization on $\mathbf{I}$ is sometimes preserved and sometimes deliberately changed.
7. That we can't say for certain whether a conceptual,data subframe is an entity or relationship reflects a fact well-known to data modelers: it is sometimes unclear whether something is a relationship or an entity.
8. Using $\mathfrak{I}$ restricted to the domain { "`SSN`", "`name`", "`birthdate`", "`skill`", $\cdots$} would be more understandable although formally equivalent to using $\mathfrak{I}$.

is mapped by $\mathfrak{F}_\epsilon.\Phi$ to SSN in $\mathfrak{F}_{\langle\text{logical,data},\texttt{Employee}\rangle}$, and similarly with name, hiredate, *etc.* Precisely, the relation

$$\Phi\,(\,\langle\text{conceptual, data, }\texttt{Employee}\rangle\texttt{SSN},\ \ \langle\text{logical, data, }\texttt{Employee}\rangle\texttt{SSN}\,)$$

holds. Observe that the way Φ was defined removes any ambiguity from the occurrences of "SSN": the first occurrence comes from the $\mathcal{OC}$ of $\mathfrak{F}_{\langle\text{conceptual,data},\texttt{Employee}\rangle}$ and the second from the $\mathcal{IC}$ of $\mathfrak{F}_{\langle\text{logical,data},\texttt{Employee}\rangle}$.

As with "conceptual,data", we abbreviate "logical,data" as "Relation" at the first level and "attribute" at the second. Thus the frame labeled ⟨Relation, Employee⟩ has subframes labeled ⟨Relation, Employee⟩⟨attribute, SSN⟩, ⟨Relation, Employee⟩⟨attribute, name⟩, and so on.

Because the attribute named skill is in fact multi-valued, we must create a new relation Emp_Skill. Thus $\mathfrak{F}_{\langle\text{Relation},\texttt{Emp\_Skill}\rangle}.\mathcal{IC} = \{\texttt{SSN, skill\_code}\}$. In this manner information content is enhanced as we move from conceptual through logical to physical.

6. Merging Frameworks

It is fairly common that two frameworks must be merged — for example, when the two organizations modeled by the frameworks merge. Formally it is very easy to merge the frameworks by taking the union of their respective components; in fact, we use the union symbol to signify such a merger. That is, $\mathfrak{H}$ is the merger of frameworks $\mathfrak{F}$ and $\mathfrak{G}$, written $\mathfrak{F} \cup \mathfrak{G}$, if, for all $\alpha \in \mathcal{P}\text{ATHS}_\mathfrak{F} \cup \mathcal{P}\text{ATHS}_\mathfrak{G}$,

$$\mathfrak{H}_\alpha.\Diamond \;=\; \mathfrak{F}_\alpha.\Diamond \cup \mathfrak{G}_\alpha.\Diamond$$

where $\Diamond$ is any of $\mathcal{IC}$, $\mathcal{OC}$, $\mathcal{SF}$, Φ, or $\mathfrak{S}$.[9] If, as is so often the case in the real world, the names are not consistently used in $\mathfrak{F}$ and $\mathfrak{G}$, then this union, while being formally correct, is a modeling mishmash. The next section discusses ways to handle this problem as often encountered in the real world.

Of course any constraints must be merged as well. In the case of *a priori* Ξ, this means first unioning the two relationships and then reconstructing the transitive closure. This will detect any violations of the constraint that may have arisen during the merging.

7. Constructing a Framework

Having presented a formal specification of the Zachman framework, the question arises as to its application. As indicated in Section 1, we have examined recursion with respect to model structure and not to the application of frameworks. In this section we sketch how a framework may be constructed through an analysis process. This of course involves the interaction of an entire framework with its surroundings, which is not only outside our formalism but inherently unformalizable. Our formalization does provide some indication

9. If, say, α is in $\mathcal{P}\text{ATHS}_\mathfrak{F}$ but not $\mathcal{P}\text{ATHS}_\mathfrak{G}$, then each $\mathfrak{G}_\alpha.\Diamond$ is interpreted as the empty set $\emptyset$.

of the organization structures that we consider robust with respect to enterprise analysis conduced with Zachman frameworks.

Ordering and sorting elements of $\mathcal{D}$ has always been difficult. Our formalism supports an architectural approach for those many processes by defining an organizational framework structure upon which to conduct analysis, specification, and implementation.

Suppose that you have a collection of independently created frameworks and frame components – essentially "proto-frames". How can you arrange these parts into a more meaningful whole? The remainder of this section presents a methodology (with two variations) to accomplish this arrangement.

To begin, the various **R** and **I** vocabularies need to be correlated; it is expected that different symbols are used in different contexts but these are consistent with each other.[10] Correlating the **R**'s requires only that they all have the same number of elements, since the ordering implies the equivalence of the specific elements. Since it is common for modelers to give names to $\ominus$ and $\oplus$, such as Zachman's "context" and "out of context", those correspondences must be distinguished as well. The **I**'s must be explicitly correlated.

Because of the importance of the order on **R**, it is more critical that components have the correct **R** classification. That is, misclassification of a component along the **I** dimension is more easily recovered from than misclassification along **R**.

The first variation of the methodology first flattens the framework and then builds a structure. Project all leaves of the proto-frames with the same r and i onto a single new branch frame. The domain of this new frame is thus the entirety of the descriptors of the original artifacts. Assign a temporary path name to each of the original leaves for referential use. At this stage, the new frame resembles Figure 2 but with a lot more subframes. Using this new branch frame, arrange the components into subframes and recursively construct new branches (defining $\mathcal{SF}$) and new leaves from the original collection of descriptors.

A second variation attempts to retain whatever structure is present in the proto-frameworks resulting from the original analysis. First, place these proto-frames in appropriate levels of scope and abstraction. Then iterate through these levels, beginning with the most general, adding new $\mathcal{SF}$ elements to link in frames. As necessary, merge frames as discussed above.

Under either approach, continue by adding new annotations as necessary. Remember to consider the entirety of each $\mathcal{IC}$. Resolve identities Ξ, build Φ, identify occurrences of $\mathcal{E}$, being as *thorough* as possible. Follow branches until leaves emerge and appropriate signatures appear. Elements that do not fit should be set aside for later consideration. Work toward a consistent set of annotations, abbreviations, and path labels. If you are an

10. The more abstract notion that $\mathbf{R} = \{ \mathsf{r}_1, \mathsf{r}_2, \cdots\}$ and $\mathbf{I} = \{ \mathsf{i}_1, \mathsf{i}_2, \cdots\}$ but that many different names may correspond to the r_j's and i_k's reflect this correlation.

experienced modeler, then this process might seem familiar. Initially the edge labels, $\mathcal{L}_1$, are most significant.

Keep the levels of abstraction consistent with a frame. That can be very difficult to achieve at first. Remember that frames can be shared between paths but that paths need to be acyclic. Use the viewing mechanism first presented to validate frames and identify possible new frames. Views that are not *consistent* can indicate an inappropriate ordering or level of abstraction among view elements. We expect that many of our readers are already performing operations similar to these in the course of their analysis. While a satisfactory signature indicates that you have reached a leaf node, the criteria for satisfaction will depend upon the level of operational detail required for modeling the system.

Since the framework architectural structure is consistent, the resulting new frames and framework should be well formed. Test the objective understanding of the model against the reality it represents by viewing it from many perspectives. Find out early if it does not fit too well. The degree of fit is likely reflective of the quality embodied in the original collection.

The new framework is manifest as the path labels, signatures, annotations, and abbreviations that occur – a richly textured mosaic that describes the context with a structure composed of recursively structured frames.

8. Conclusion

8.1. Contributions

This paper presents a formal foundation for Zachman Frameworks using recursion on the basic model structure. We detail an explicit differentiation between the ordering derived from hierarchical decomposition and ordering related to roles (**R**). Distinguishing these two ordering constructs focuses on roles within the enterprise context and on decomposition within the architectural context.

Model structure is separated from the application of the model and thereby provides a common reference for many diverse applications of Zachman's framework concepts. The underlying formalism is consistent with the current uses of the Zachman approach and suggests opportunities for improvement in framework application.

We show that a rather simple recursive structure can, through use of appropriate viewing mechanisms, provide the foundation for significant artifact repositories spanning a wide range of enterprise needs. Those engaged in the development of models know informally that complexity is greater in viewing and synthesizing, not designing; the formalism makes this quite explicit.

The discussion of viewing mechanisms identifies several pitfalls that are the origins of incompatibilities when operational outcomes are based solely on views; this clearly indicates the need for separation of model structure from model-derived views. It is in this context that our concern for presumptive architectures as used in ERP systems arises. Without an explicit definition of the structural architecture supporting these highly complex systems, it is very difficult, if not impossible, to identify the the ambiguities present in operational views used to design an implementation. While a particular view may appear to support the enterprise objective, the supportive structure may lack the consistency required for linkage to other operational contexts. Our approach provides a means for identifying and resolving structural inconsistencies, although such resolution may require returning to the full model.

Our hope is that a formal foundation will encourage a wide range of professionals to pursue the benefits of enterprise architectural modeling in their daily activities.

8.2. Future Work

The formalization that we have presented suggests a great deal more work to be done concerning Zachman frameworks. As is always the case, one answer raises ten more questions and issues. This work falls roughly into three broad classes: refine the concepts and improve their perspicuity, extend and fill in the formal treatment, and apply the concepts to framework modeling.

Concepts:

- ⋆ Explore an equivalent formalism with cells in addition to frames. In this variation, there are two primary classes of objects: frames $\mathfrak{F}$ as before and cells $\mathfrak{C}$. The subframe mappings $\mathcal{SF}_\alpha$ is replaced by two mappings: one from $\mathbf{R} \times \mathbf{I}$ to $\mathfrak{C}$ and one from $\mathfrak{C} \times \mathcal{D}$ to $\mathfrak{F}$.

- ⋆ Extend the notion of meta-model constraints (*i.e.* constraints on abbreviations) to support the specification of different flavors of frameworks.

- ⋆ The specification of model semantics using Φ is sometimes awkward and stilted. It is desirable that this be made more natural and fluid.

- ⋆ Other parts of the $\mathbf{R} \times \mathbf{I}$ grid need the same well-defined and widely accepted formal specifications that the ER model and its relational implementation provide for the what (data) column. This is true independent of our formalism.

Formalisms:

- ⋆ A more complete and rigorous characterization of constraints should yield mathematical results on their limits and implications.

- ⋆ Tools and techniques for view definition need to be made precise and their properties investigated. In particular, the characteristics of view ambiguities need to be investigated and ways of handling these ambiguities need to be fully specified.

Applications:

- ⋆ The methodology sketched in section 7 needs to be expanded.
- ⋆ We intend a trial run of methodology – expressing a real-world architecture using our recursive framework.
- ⋆ The ultimate goal of for developing a formalism to facilitate the specifications for toolkit. For example, the transformation from ER model to relational design discussed in section 5 happens consistently across an entire frame. This suggests the need for tools to support this transformation.

Bibliography

1 M. A. Beedle, *Pattern Based Reengineering*, Framework Technologies Inc., Schaumburg, IL. 1997.
http://www.fti-consulting.com/users/beedlem/pbr.html

2 R. Briggs (Principal Investigator), North Texas GIS Consortium Spatial Data Warehouse NSDI Demonstration Project Final Report, The Bruton Center for Development Studies, The University of Texas at Dallas, 1997.

3 B. E. Burgess and T. A. Hokel, *A Brief Introduction to the Zachman Framework*, Framework Software, Inc., Redondo Beach, CA. August 1994.

4 S. Campbell, Green cities, growing cities, just cities? Urban planning and the contradictions of sustainable development, *Journal of the American Planning Association*, **62**,3, Summer 1996.

5 R. Evernden, The Information Framework, *IBM Systems Journal*, *35*,1 1996.

6 W. H. Inmon, John Zachman, and Jonathan Geiger, *Data Stores, Data Warehousing, and the Zachman Framework*, McGraw-Hill 1997.

7 F. R. Jacobs, personal communication, Kelley School Of Business, Indiana University, 1998.

8 R. A. Johnson, Framework = (components + patterns), *Comm. ACM*, *40*,10, Oct. 1997. 39-42.

9 J. Karlin and D. Welch, A Framework of Mission Operations System Models. 1996.
http://www.esoc.esa.de/external/mso/SpaceOps/1_10/1_10.htm

10 H. Molina Noriega and E. Kopko, A consulting firm and its clients: many uses of the

framework, *ZIFA 1997 Annual Forum*, Chicago, May 7-9, 1997.

11 W. Selkow, Something old, something new, *ZIFA 1997 Annual Forum*, Chicago, May 7-9, 1997.

12 J. F. Sowa and J. A. Zachman, Extending and formalizing the framework for information systems architecture, *IBM Systems Journal*, **31**,3, 1992. IBM Publication G321-5488.

13 J. D. Ullman, *Principles of Database and Knowledge-Base Systems*, Volume I – Fundamental Concepts, Computer Science Press, New York, 1988.

14 P. J.A. van Vliet, Information Systems Architecture & Organization Syllabus, Information Systems & Quantitative Analysis Department, College of Information Science and Technology, University of Nebraska at Omaha. 1997.
http://www.istis.unomaha/isqa/vanvliet/arch/syllabus.htm

15 J. L. Whitten and L. D. Bentley, *Systems Analysis and Design Methods* 4th Ed., Irwin McGraw-Hill. 1998.

16 T. J. Williams (ed), (May 1988), *Reference Model for Computer Integrated Manufacturing, A Description from the Viewpoint of Industrial Automation*, CIM Reference Model Committee, International Purdue Workshop on Industrial Computer Systems, Purdue Laboratory for Applied Industrial Control, Purdue University, West Lafayette, IN; Instrument Society of America (1989), Research Triangle Park, NC.

17 J. Wyns, H. Van Brussel, P. Valckenaers, and L. Bongaerts, Workstation Architecture in holonic manufacturing systems, *28th CIRP International Seminar on Manufacturing Systems*, 1996
http://www.mech.kuleuvan.ac.be/pma/project/goa/extracts/architec.htm

18 J. A. Zachman, A framework for information systems architecture, *IBM Systems Journal*, **26**,3, 1987. IBM Publication G321-5298.

Please note that the URL's used in the above bibliography existed when this article was drafted but web-based citations are notoriously volatile. As information professionals we should be more responsible in our maintenance of information resources.