Formalization of Enterprise Architecture 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

Abstract

This paper formalizes the structure of an Enterprise Architecture Framework model as described by Zachman. A Framework, in this context, is a schema for organizing information artifacts over a broad scope, ranging up to an entire enterprise. It classifies information with respect to abstraction level and conceptual aspect. The levels of abstraction are typically drawn from the role perspectives of "specifier", "designer", and "implementer" while the conceptual aspects are typically the familiar interrogatives "what", "how", "where", "when", "who", and "why". Each cell of the resulting grid recursively contains additional frames that decompose the abstraction aspect as needed. The formalism provides mechanisms for connecting framework components down the recursive levels of detail and across the abstraction boundaries. While the tree of frames expresses the structure of a framework model, these connections express the framework's meaning.

We extend the basic formalism by adding mechanisms to facilitate the customization of framework metamodels, impose constraints on the framework model, and view frameworks from different perspectives.

1 INTRODUCTION

This paper explores the formalism inherent in the structure and semantics of Enterprise Architecture Frameworks. Enterprise Architecture refers to the structure of information necessary for the alignment of enterprise goals with operational practice, most often in the context of automated information systems. 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.

In 1996, the United States Congress enacted the Information Technology Management Reform Act (ITMRA, also known as the Clinger-Cohen Act) to address a growing frustration with the discontinuity between the goals of Federal agencies and their information technology capability to support those goals. This act and OMB Memorandum 97-16, which implemented the act, mandated the development of Enterprise Architectures in all government agencies. Enterprise Architecture development is therefore a central pillar in the $34 billion non-defense information technology spending for the US government. Consequently, legions of public employees are now assigned responsibilities related to the development and maintenance of enterprise architectures. Similar discontinuities in businesses world-wide are being addressed by comparable legions of private employees. These discontinuities are also responsible for the explosive interest in ERP systems – one of the major benefits of an ERP package is that it supplies an enterprise information architecture. The need to describe complex systems in a way that is thorough, enduring, and useful over time motivates the formalization of a comprehensive approach that offers both insight and opportunity to those tasked with fulfilling that need.

Enterprise Architecture is perceived as the best and perhaps only means to provide rapid response to changing business conditions. With such significant levels of resource allocation to enterprise architecture efforts, we believe the examination of formalized mechanisms to support those efforts are required because such mechanisms will facilitate tool development.

We have chosen to explore the Zachman approach because of its world-wide use as an underlying conceptual framework for enterprise architecture development. In his *IBM Systems Journal* articles, John Zachman describes a Framework for Information Systems which defines an architectural structure for organizing institutional artifacts[6,4]. 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 has become a standard of reference for both government and commerce.

One characteristic of the framework concept set forth by Zachman is recursion, reflecting a top-down analysis

of the world being modeled. As described in [2], 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 present is more general than that initially proposed by Zachman. Our goal is to explore and enhance the expressive power inherent in that earlier model and to broaden its application to issues of enterprise architecture.

This paper begins with a "bare bones" formalization of frames and frameworks: first 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 specific example formalizing a slice of an enterprise architecture. Finally, we summarize what we have achieved.

2 FRAMEWORK STRUCTURE

A *framework* is constructed from *frames*. This section first introduces the components used to describe frames. It then discusses how a framework is composed of individual frames. Finally, it gives a formal definition of these frames. The next section discusses how to represent meaning within such a structure.

Component Vocabulary

A framework's structure is always defined in terms of three sets: $\mathbf{R}$ and $\mathbf{I}$, 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. A frame is therefore considered a grid, with $\mathbf{R}$ indexing rows and $\mathbf{I}$ columns and with $\mathcal{D}$ identifying components of a grid cell.

$\mathbf{R}$ $= \{\mathsf{r}_1, \cdots, \mathsf{r}_n\}$, a fixed set of *abstraction levels* (the *roles*)

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

Typically n for $\mathbf{R}$ is 3 and a meaningful vocabulary is used for members of $\mathbf{R}$. Common vocabularies for $\mathbf{R}$ (in their typical order) are {"conceptual", "logical", "physical"}, {"owner", "designer", "builder"}, and {"enterprise", "system", "technology"}.

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

One common convention for $\mathbf{R}$ places design (the row of the role "designer") between specification ("owner") and implementation ("builder"). The design role mediates between the desire of specification and the reality of implementation by applying the constraints of rational judgment and expectation. A specification that cannot be implemented is no more useful than is achieving an implementation objective that is not a specification. The business alignment within an enterprise achieved by using the Zachman approach is the consequence of the ordering imposed on $\mathbf{R}$. No other ordering provides such opportunity for mediation based upon rules of design and logic. While the three elements of $\mathbf{R}$ as typically used are sometimes linked by three connections to indicate contention or coordination between the roles[1], Zachman's insight properly breaks the unconstrained linkage to require design in a context and thereby provides architectural structure capable of supporting the conceptual, logical, and physical modeling requirements of the enterprise.

$\mathbf{R}$ is extended to $\mathbf{R}^{\odot}$ with new elements $\ominus$ and $\oplus$, respectively before and after all elements of $\mathbf{R}$. In Zachman terminology, $\ominus$ corresponds to "context" and $\oplus$ to "out of context". Separating $\mathbf{R}$ from $\ominus$ and $\oplus$ recognizes that the top and bottom rows of a typical frame are boundaries, which are important for passing information, rather than sites for analysis and further decomposition.

The second important set is

$\mathbf{I}$ $= \{\mathsf{i}_1, \cdots, \mathsf{i}_m\}$, a fixed set of *aspects* (the *interrogatives*)

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

The set $\mathbf{I}$ provides a classification of content into universal categories long ago recognized as essential aspects of the enterprise story. Each element of $\mathbf{I}$ is an important part of the whole that a framework makes explicit.

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

$\mathcal{D}$ $= \{\mathsf{i}_1, \cdots, \mathsf{i}_\ell\}$, the *descriptors*

Labels and Paths

Labels bind frames to subframes and thus characterize local structure. Paths are sequences of labels encountered while traversing through successive subframes.

Thus paths 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 $\mathcal{L} := \epsilon | \mathcal{L}\,\mathcal{L}_1$, 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 $\mathcal{F}_\alpha$, 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 $\mathcal{F}_\alpha$).

Frames and Frameworks

A framework is constructed recursively from two kinds of frames, branch frames and leaf 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.

$\mathcal{F}$ a *framework* is a finite set of *frames* $\{\mathcal{F}_\alpha : \alpha \in \mathcal{L}\}$, where a frame $\mathcal{F}_\alpha$ is either a *branch* frame or a *leaf* frame, both of which are characterized below.

$\mathcal{PATHS}_\mathcal{F}$ = $\{\alpha : \mathcal{F}_\alpha \text{ is defined}\}$ for a framework $\mathcal{F}$. This set is typically written as only "$\mathcal{PATHS}$", with $\mathcal{F}$ understood from context.

The labels of $\mathcal{F}$ (*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 $\mathcal{F}$ is a labeled graph with nodes in $\{\mathcal{F}_\alpha : \alpha \in \mathcal{L}\}$ and edges labeled by $\mathcal{L}_1$. There is no constraint that the actual frame $\mathcal{F}_\alpha$ is distinct from $\mathcal{F}_\beta$. That is, paths α and β may lead to the same frame instance. However, we require that the graph be acyclic.

With this notation, $\mathcal{F}_\epsilon$ 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* $\mathcal{F}_\alpha$ is characterized by the 4-tuple:

$\mathcal{IC}_\alpha$	$\subseteq \mathcal{D}$	input ("context")
$\mathcal{OC}_\alpha$	$\subseteq \mathcal{D}$	output ("out of context")
$\mathcal{SF}_\alpha$	$: \mathbf{R} \times \mathbf{I} \times \mathcal{D} \to \mathcal{F} \cup \mathcal{L}$	("subframes"")
Φ_α	$\subseteq \mathit{outputs} \times \mathit{inputs}$	(details below)

A structure generated with two levels of frames is illustrated in Figure 1 below. The entire framework is $\mathcal{F}$. The top frame is $\mathcal{F}_\epsilon$, which has several subframes. This figure also has two Φ connections which are explained in section 3.

$\mathcal{L}$ in the definition of $\mathcal{SF}$ is a "virtual frame" allowing framework components to be connected together with a form of indirection. To keep this purpose in mind we use $\boldsymbol{\mathcal{VF}}$ to indicate $\mathcal{L}$ use in this context. We actually would like $\boldsymbol{\mathcal{VF}}$ to be $\mathcal{PATHS}_\mathcal{F}$, but defining $\boldsymbol{\mathcal{VF}}$ to be $\mathcal{PATHS}_\mathcal{F}$ would introduce a circularity in the definition of $\mathcal{F}$, since $\mathcal{PATHS}$ is defined in terms of $\mathcal{F}$. $\boldsymbol{\mathcal{VF}}$ comprises "symbolic links" in the Unix sense. That is, when $\mathcal{SF}_\alpha(r, i, d) = \beta \in \mathcal{PATHS}$, this signifies a connection to $\mathcal{F}_\beta$. The benefit of this can be seen in the example of Figure 1: 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 – $\boldsymbol{\mathcal{VF}}$ 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* $\mathcal{F}_\alpha$ is characterized by the 3-tuple:

$\mathcal{IC}_\alpha$	$\subseteq \mathcal{D}$	input (context)
$\mathcal{OC}_\alpha$	$\subseteq \mathcal{D}$	output (out of context)
$\mathcal{S}_\alpha$		(details below)

One leaf frame which commonly occurs at the end of many paths is the completely empty frame $\langle \emptyset, \emptyset, \emptyset \rangle$, denoted by $\mathcal{E}$. There is no technical requirement that a frame be present and hence no specific need for $\mathcal{E}$. However, $\mathcal{E}$ 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 "$\mathcal{F}_{\alpha\langle r,i,d\rangle}$", "$\mathcal{SF}_\alpha(r, i, d)$", and "$\mathcal{F}_\alpha.\mathcal{SF}(r, i, d)$" are equivalent, provided $\alpha\langle r, i, d\rangle \in \mathcal{PATHS}$. 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.

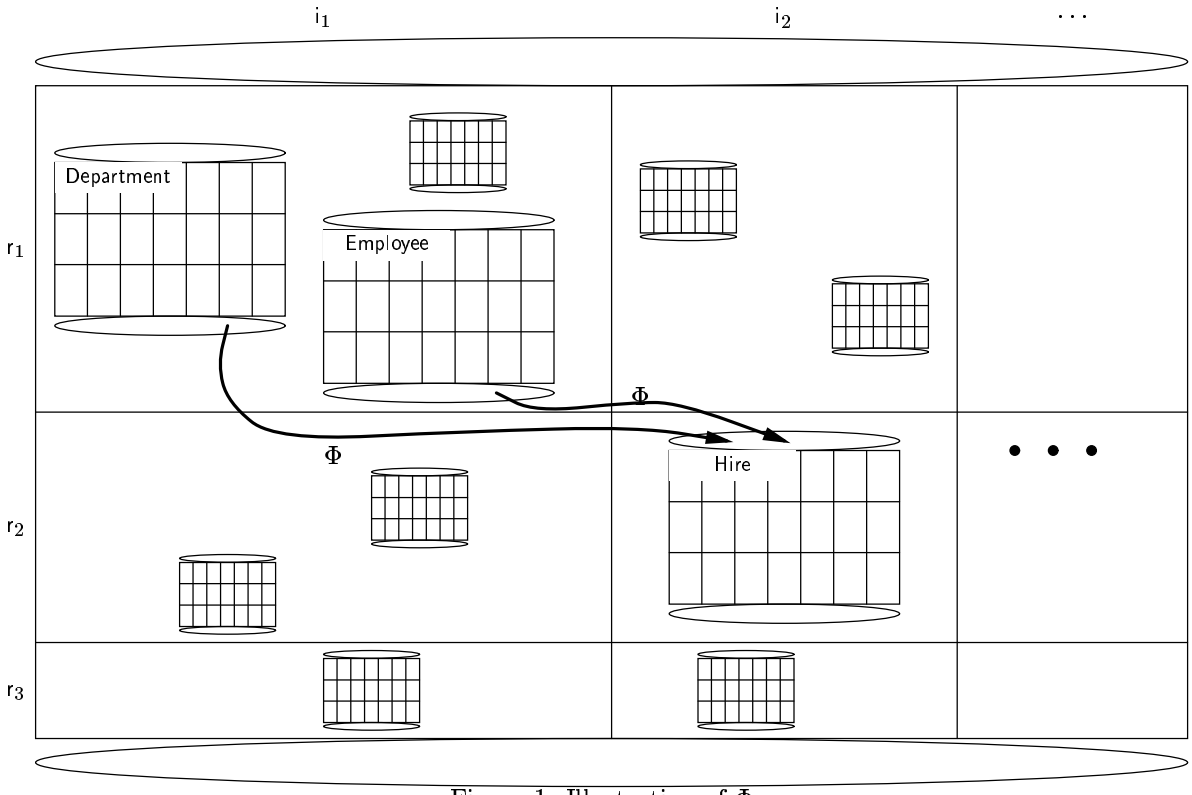

Figure 1: Illustration of Φ

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

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

It is expected that $\mathcal{IC}_\alpha \cup \mathcal{OC}_\alpha \supseteq \mathcal{D}_\alpha$, that is, the decomposition defined by $\mathcal{F}_\alpha$ 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.

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 than other frame components. 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).

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 $r \in \mathbf{R}$,

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

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

$$\mathcal{EOC}_{\alpha,\ominus} = \mathcal{IC}_\alpha$$
$$\mathcal{EIC}_{\alpha,\oplus} = \mathcal{OC}_\alpha$$

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

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

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 2 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 $\mathbf{R}$.

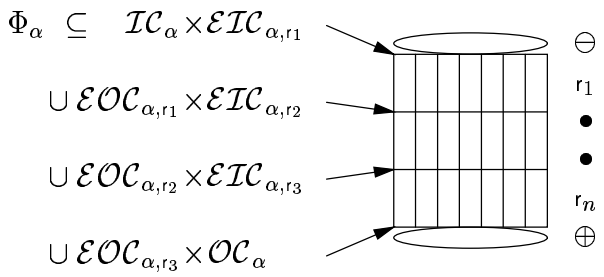

Figure 2: How Φ Relates to a Frame

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 1 shows *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 $\mathcal{PATHS}$. $\mathcal{VF}$ is oblivious of structure, connecting arbitrary components (recall, $\mathcal{VF}$ 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 $\mathcal{F}_\beta$, 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 $\mathcal{F}_\beta$.

Leaf Frame Details

Frameworks allow us to model the structure of a world of discourse. It is still necessary to populate this structure with the items from the world. Thus we distinguish between the *model* and the *instance*.

The definitions of $\mathcal{S}_\alpha$ and its component $Types_\alpha$ complete the definition of frame syntax.

Elements of $Types$ are the primary linkage between a framework as a model and an instance of that framework. When connecting to the real world, 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 bags of items, we use the notation $Types$ to specify how we deal with such instances.

$Types_\alpha = \mathcal{IC}_\alpha \cup \mathcal{OC}_\alpha \cup \{\text{"BAG OF } \lambda\text{"} : \lambda \in \mathcal{IC}_\alpha \cup \mathcal{OC}_\alpha\}$

$\mathcal{S}_\alpha$ indicates the "operation of the system modeled by the framework". That is, when a framework is instantiated to reveal the underlying "real world", the signatures in the various $\mathcal{S}_\alpha$ become the operations that do the work of the system.

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

We use $\mathcal{I}$ for any leaf frame that 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 bag. Formally, such usage is ambiguous because there is an identity frame for each distinct $\mathcal{IC}$, $\mathcal{OC}$ pair but in practice it is easy to handle.

4 BUILDING ON FRAMEWORKS

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

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

where $\mathcal{STRINGS}$ is a finite set of strings over some alphabet. This function is added to both kinds of frames. That is, a branch frame $\mathcal{F}_\alpha$ 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 $\langle \mathcal{IC}_\alpha, \mathcal{OC}_\alpha, \mathcal{S}_\alpha, \mathcal{A}_\alpha \rangle$.

Annotations work best when members of $\mathcal{D}$ are used consistently across all frames of a framework. For example, we might require that $\mathcal{A}_\alpha(\mathsf{framename})$ be defined for any non-empty frame $\mathcal{F}_\alpha$. In the example of Figure 1, saying that $\mathcal{A}_{\langle \mathsf{r}_1, \mathsf{i}_1, \mathsf{Department}\rangle}(\mathsf{framename})$ is "Department Entity" provides a humanly intelligible name to a component whose technical name is the path $\langle \mathsf{r}_1, \mathsf{i}_1, \mathsf{Department}\rangle$.

Abbreviations

There are several other conventions that help the presentation and comprehension of a framework.

First, a path may be abbreviated by a single name.

Second, 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".

Third, 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 ⟨conceptual, data, —⟩) 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 in conjunction with Figure 1; that is, an Entity of the ER model is labeled "Entity" in the framework.

Contrasting these last two points, a name in the first case abbreviates an entire frame, while in the second case it abbreviates only part of the name of a frame's component. 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.

Fourth, 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 $\mathbf{R} \times \mathbf{I}$ cells and is particularly useful when there are many paths whose labels all have the same form, as is true with "⟨conceptual, data, —⟩⟨conceptual, data, —⟩". We call the conceptual,data cell "Entity" at the first frame level and "attribute" at the second. Thus the statement "Relation Employee includes attribute SSN and name" means that there are frames with label "⟨conceptual, data, Employee⟩⟨conceptual, data, SSN⟩" and "⟨conceptual, data, Employee⟩⟨conceptual, data, name⟩". This example is further elaborated in section 5.

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

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.

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.

There are two ways in which constraints arise in developing a framework model: *a priori* and derived. An *a priori* constraint is identified and explicitly declared during the modeling process. 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

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

which specifies identities between outputs. It is further required that Ξ_α be a true equivalence relation. 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 3 illustrates this situation.

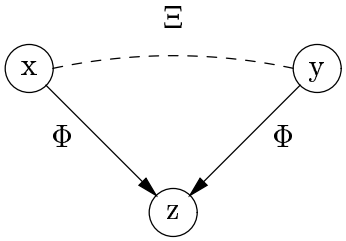

Figure 3: Ξ and Φ

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, the use of abbreviations to regularize the construction of path names. A constraint that requires that all path names be constructed according to a specific set of abbreviations makes those abbreviations the meta-model of the framework.

In the same family of constraints, a framework $\mathcal{F}$ is said to be *thorough* provided that, whenever a branch frame $\mathcal{F}_\alpha$ is defined, $\mathcal{SF}_\alpha(r, i, d)$ is defined for all $r \in \mathbf{R}$, $i \in \mathbf{I}$, and $d \in \mathcal{D}_\alpha$.

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.

Views

Views embody the expressive power of our formalism. The views derived from a framework provide its semantic content in a manner suitable for broader consumption. This is analogous to relational database systems, which have a well-defined view mechanism.

There are three motivations for "viewing", that is for rearranging the structure of a framework. The first is to focus on particular elements of a framework; this is especially important in explaining a design. The second is to apply the structural perspectives arrising from a particular domain knowledge. The third is to facilitate developing and validating a framework structure and its contents.

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 then define new path labels.

For example, the rewriting rule

$$\langle r, i, \mathsf{A}\rangle\langle \mathsf{Owner}, i, d\rangle \Longrightarrow \langle r, i, d\rangle$$

takes a framework $\mathcal{G}$ into a view framework $\check{\mathcal{G}}$ such that $\check{\mathcal{G}}_\beta = \mathcal{G}_\alpha$ 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). Unfortunately, such a mapping can lead to ambiguities. Say we had another rule with "A" replaced by "B" and that two distinct paths $\langle \mathsf{Owner}, \mathsf{i}, \mathsf{A}\rangle\langle \mathsf{Owner}, \mathsf{i}, \mathsf{a}\rangle$ and $\langle \mathsf{Owner}, \mathsf{i}, \mathsf{B}\rangle\langle \mathsf{Owner}, \mathsf{i}, \mathsf{a}\rangle$ in $\mathcal{G}$ map to distinct frames. Then the image path $\langle \mathsf{Owner}, \mathsf{i}, \mathsf{a}\rangle$ in $\check{\mathcal{G}}$ would be ambiguous in referring to both of these frames.

Another serious difficulty is the fact that the rules may map one single path ambiguously. 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, dealt 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 occurred 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 [3]. 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. 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 4 illustrates the re-arrangement used at Butler International, where the framework on the left is rewritten to the view on the right by the rule

$$\langle r, \mathsf{Where}, d_1\rangle\langle \mathsf{Owner}, \mathsf{Who}, d_2\rangle$$

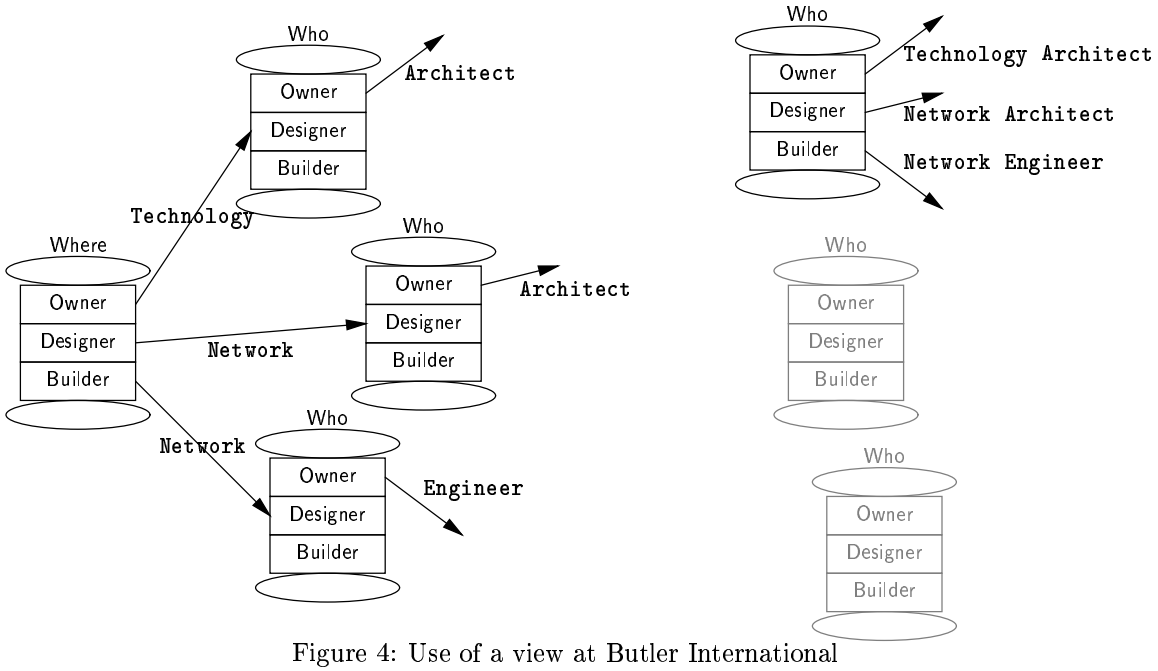

Figure 4: Use of a view at Butler International

$$\Longrightarrow \langle r, \text{“Who”}, d_1\, d_2\rangle$$

The same figure also indicates, in gray, the existence 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 $\mathcal{VALID\,JOBTITLE}$ and constrain $\check{\mathcal{D}}_\epsilon \subseteq \mathcal{VALID\,JOBTITLE}$.

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 additional 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 4 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 that combine several 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[5]). 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, leaves and intertwined paths, it is sometimes advantageous to “flatten” a framework (or portions of a framework) for presentation and discussion.

While rewriting rules 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:

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

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 constraint imposed on the original Φ_α. This may either be accommodated by restricting the rewriting rules (or their application) or accepted as a consequence of the fact that views sometimes distort.

Handling virtual frames is much easier. A virtual frame link β appearing in $\mathcal{G}$ is simply replaced in each corresponding occurrence in $\check{\mathcal{G}}$ by $\check{\beta}$, (if such a rewriting $\check{\beta}$ of β exists). 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 interconnections.[1]

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 aspects (columns) such as motivation because the target lacks such a specification.

We already noted 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.[2] In fact, we have abbreviated any such triple as ⟨Entity, —⟩ or ⟨Relationship, —⟩. One such entity is named Employee; that is, the frame $\mathcal{F}_{\langle\text{conceptual, data, Employee}\rangle}$ represents the employee entity, containing (at least) the attributes of that entity. Formally,

$$\mathcal{F}_{\langle\text{conceptual, data, Employee}\rangle}.\mathcal{IC} \supseteq \{\text{SSN, name, hiredate, skill, } \cdots\}$$

Or, in abbreviation, Entity Employee includes SSN, name, hiredate, skill, *etc.*

Each attribute of an entity in the Semantic Model is then carried forward to be an attribute of a relation

1. The categorization on $\mathbf{I}$ is sometimes preserved and sometimes deliberately changed.
2. That we cannot 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.

in the Logical Data Model. That is SSN as it appears in $\mathcal{F}_{\langle \text{conceptual, data, Employee} \rangle}$ is mapped by $\mathcal{F}_\epsilon.\Phi$ to SSN in $\mathcal{F}_{\langle \text{logical, data, Employee} \rangle}$, and similarly with name, hiredate, *etc.* Precisely, the relation

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

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

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 multivalued, we must create a new relation Emp_Skill. Thus $\mathcal{F}_{\langle \text{Relation,Emp\_Skill} \rangle}.\mathcal{IC} = \{\text{SSN, skill\_code}\}$. 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, $\mathcal{H}$ is the merger of frameworks $\mathcal{F}$ and $\mathcal{G}$, written $\mathcal{F} \cup \mathcal{G}$, if, for all $\alpha \in \mathcal{P}_{\mathcal{ATHS}_\mathcal{F}} \cup \mathcal{P}_{\mathcal{ATHS}_\mathcal{G}}$,

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

where $\Diamond$ is any of $\mathcal{IC}$, $\mathcal{OC}$, $\mathcal{SF}$, Φ, or $\mathcal{S}$.[3] If, as is so often the case in the real world, the names are not consistently used in $\mathcal{F}$ and $\mathcal{G}$, 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.

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

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 of the organization structures that we consider robust with respect to enterprise analysis conduced with architectural 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 we have a collection of independently created frameworks and frame components – essentially "proto-frames". How can these parts be arranged into a more meaningful whole? The remainder of this section presents a methodology (with two variations) to accomplish this arrangement.

To begin with, the various $\mathbf{R}$ and $\mathbf{I}$ vocabularies need to be correlated; it is expected that different symbols are used in different contexts but these are consistent with each other.[4] Correlating the $\mathbf{R}$'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 $\mathbf{I}$'s must be explicitly correlated.

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

One 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. 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, pushing descriptors down the tree as appropriate.

4. The more abstract notion that $r_j \in \mathbf{R}$ or $i_k \in \mathbf{I}$ may correspond to many different names reflects this correlation.

Another 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 $\mathfrak{E}$, being as *thorough* as possible. Follow branches until leaves emerge and appropriate signatures appear. Elements that do not fit should be considered later. Work toward a consistent set of annotations, abbreviations, and path labels. If you are an experienced modeler, then this process might seem familiar. Initially the edge labels, $\mathcal{L}_1$, are most significant.

Keep the levels of abstraction consistent within 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 (acquaintance, familiarity, or mastery).

Since the framework's 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. 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

This paper examines a formal foundation for Enterprise Architecture Frameworks using recursion on the basic model structure. We detail an explicit differentiation between the ordering derived from hierarchical decomposition and ordering related to $\mathbf{R}$. 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 enterprise architecture concepts. The underlying formalism is consistent with the current 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. 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 and to encourage the development of a wide range of tools to support this modeling.

Bibliography

1 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.

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

3 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.

4 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.

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

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