A Formal Enterprise Architecture Framework to Support Multi-model Analysis

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 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 models with respect to abstraction level and conceptual aspect. The levels of abstraction are typically drawn from the engineering role perspectives: "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 problem as needed. The formalism provides mechanisms for connecting framework components down recursive levels of detail and across abstraction boundaries. While the lattice of frames expresses the structure of a framework, the cross-component connections express the framework's meaning. The basic formalism is extended by adding mechanisms to facilitate the customization of framework meta-models, impose constraints on frameworks, and view frameworks from different organizational perspectives.

1. INTRODUCTION

This paper explores the formalism inherent in the structure and semantics of Enterprise Architecture Frameworks. An Enterprise Architecture Framework organizes the wide variety of models that appear in a well-run enterprise, most often in the context of automated information systems. The ultimate goal is, of course, to support modeling that is more useful and accurate. The proximate goal is to formalize the framework development process; formalizing the artifacts produced by such development is an essential precursor to formalizing the process.

Enterprise Architecture is perceived as the best and perhaps only means to develop flexible yet reliable information systems in the face of rapidly changing software tools and even more rapidly changing business conditions. 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 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.

With such significant resources allocated to enterprise architecture efforts, formalized mechanisms to support those efforts are required. Formalization has several compelling advantages: (1) it exposes commonalities in the variety of modeling methodologies, (2) it challenges us to be precise in our concepts and the use of these concepts, (3) it enhances our understanding of the issues, (4) it promotes coherence among models along the scale from enterprise-wide to detail, (5) it allows communication across various methodologies and approaches, (6) it provides conceptual metrics for evaluating particular methodologies, (7) it facilitates toolkit development, (8) it enables precise specification of constraints and views, and (9) it scales.

We have chosen to begin our formalization with 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[9,7]. 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 architecture that has become a standard of reference for both government and commerce. Figure 1 shows a framework as described by Zachman, populated with a variety of models[10]. The framework will be fully explained in section 2; for now, observe the organization governed by "roles" (row labels) and "interrogatives" (column labels).

	what	how	where	who	when	why
owner	semantic model	business process model	logistics network	work flow model	master schedule	business plan
designer	logical data model	"application architecture"	"distributed systems architecture"	human interface architecture	processing structure	business rules
builder	physical data model	"systems design"	"system architecture"	presentation architecture	control structure	rule design

Figure 1: Models in Zachman's Enterprise Architecture Framework

The approach we formalize is more general than Zachman's. Our goal is to explore and enhance the expressive power inherent in that structure and to broaden its application to issues of enterprise architecture.

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 [3], 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 framework structure and not to the application of frameworks.

The significant principles that govern our formalization are: (1) formalize the framework approach, not one particular framework, (2) abstraction↔concreteness and generality↔detail are distinct dimensions, (3) the framework should provide mechanism, not policy, (4) recursion is decomposition and successive refinement, not reoccurrence, (5) multiple views are inherent and essential in system analysis and design, (6) roles are ordered, interrogatives are not, and (7) entire preceeding role is relevant for each cell of a frame.

This paper begins with a "bare bones" formalization of frames and frameworks: how a collection of frames is structured into a framework and how these frames interact. It then extends this formalism with constraint and view mechanisms, providing tools for building a framework reflecting the real world. This material is brought together in a specific example modeling a slice of an enterprise architecture. A more detailed version of this paper, giving the full formalism and expanding on the issues of views and constraints, is available as a Technical Report from Indiana University[4].

2. FRAMEWORK STRUCTURE

A *framework* is constructed from *frames*. A crucial definition will be $\mathfrak{F}_\alpha$, the frame named by α. Thus this section first characterizes such names. It then discusses how a framework is composed of individual frames. Finally, it describes how frame-to-frame connections construct "meaning".

2.1 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 $\mathbf{I}$ indexing columns and with $\mathcal{D}$ identifying components of a grid cell.

$\mathbf{R} \quad = \{\mathsf{r}_1, \cdots, \mathsf{r}_n\}$, 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}. Using abstract symbols (that is, "r_i") rather than specific terms allows us to be "notationally neutral," recognizing that part of the confusion arises from the vocabularies used in the variety of methodologies. 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.

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 longest linkage (r_1 to r_3) 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 r_0 and r_{n+1}, respectively before and after all elements of $\mathbf{R}$. In Zachman terminology, r_0 corresponds to "context" and r_{n+1} to "out of context". The separation of r_0 and r_{n+1} from $\mathbf{R}$ 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.

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

Also, $\mathcal{D}_\alpha$ is the subset of $\mathcal{D}$ localized to the frame $\mathfrak{F}_\alpha$ named by α.

2.2 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 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.*

2.3 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.

$\mathfrak{F}$ 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.

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

The labels of $\mathfrak{F}$ (*i.e.* $\mathcal{P}\mathcal{ATHS}$) 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 $\mathcal{L}_1$. However, 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. Thus the syntax ($\mathcal{PATHS}$) has a tree structure, although the underlying semantics ($\mathfrak{F}$) is a graph. However, we require that the graph be acyclic. With this notation, $\mathfrak{F}_\epsilon$ is the "root" frame, that is the top-level frame that presents the entire framework.

We now define the structure (syntax) of the two kinds of frames. 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:

$$\begin{array}{lll}
\mathcal{IC}_\alpha & \subseteq \mathcal{D} & \text{input ("context")} \\
\mathcal{OC}_\alpha & \subseteq \mathcal{D} & \text{output ("out of context")} \\
\mathcal{SF}_\alpha & : \mathbf{R} \times \mathbf{I} \times \mathcal{D} \to \mathfrak{F} \cup \mathcal{L} & \text{("subframes"")} \\
\Phi_\alpha & \subseteq \mathit{outputs} \times \mathit{inputs} & \text{(details in Appendix)}
\end{array}$$

The $\mathcal{IC}$, $\mathcal{OC}$, and $\mathcal{SF}$ components reflect the recursive structure of a framework. 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. The relationship Φ describes how information passes from subframe to subframe. Because this information passing occurs in complex ways, the definition of Φ is more difficult than other frame components. Φ is defined so that it operates across columns but only from one row to the next. Note that Φ defines only interconnections; all decompositions are expressed in subframes and actual real-world facts appear only in leaf frames.

A structure generated with two levels of frames is illustrated in Figure 2. The entire framework is $\mathfrak{F}$. The top frame is $\mathfrak{F}_\epsilon$, which has several subframes. Figure 2 shows *just two* of many potential connections defined by Φ.

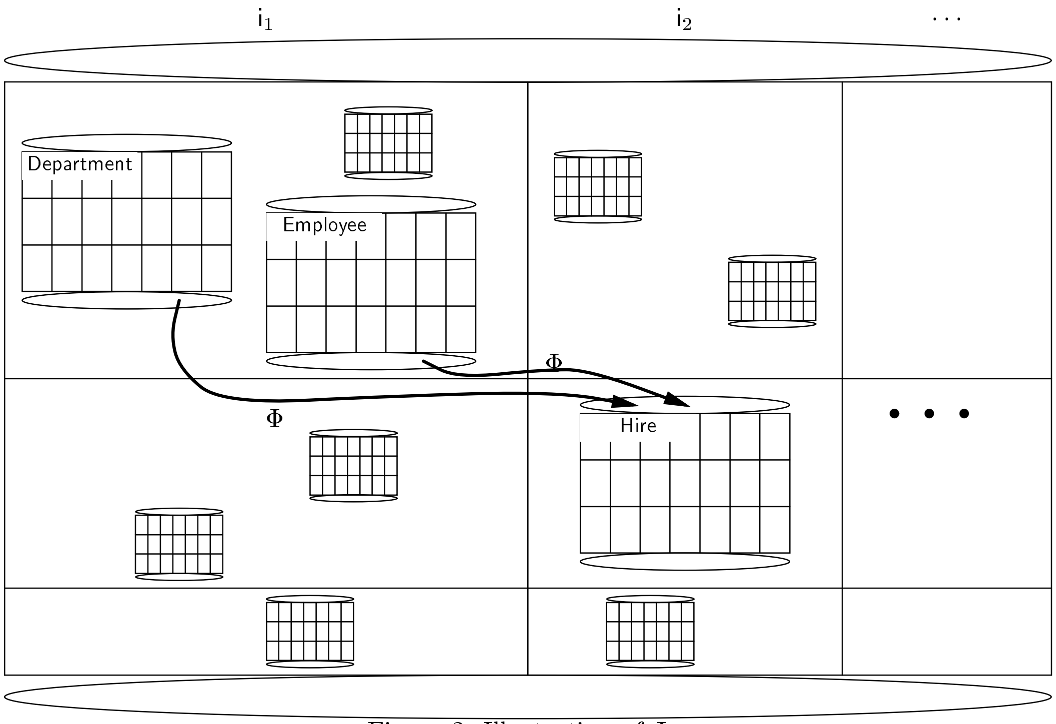

Figure 2: Illustration of Φ

$\mathcal{L}$ 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 $\mathfrak{VF}$ to indicate $\mathcal{L}$ use in this context. $\mathfrak{VF}$ 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 $\boldsymbol{\mathfrak{F}}_\beta$. The benefit of this can be seen 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 – $\boldsymbol{\mathcal{VF}}$ 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.

The question naturally arises concerning the way in which Φ differs from explicit connections (via $\mathcal{SF}$) or virtual frame links. Explicit connections are very structured, yielding the tree labeled by $\mathcal{P}_{\mathcal{ATHS}}$. $\boldsymbol{\mathcal{VF}}$ provides a form of aliasing. 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 $\boldsymbol{\mathfrak{F}}_\beta$, 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 $\boldsymbol{\mathfrak{F}}_\beta$.

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

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

$\boldsymbol{\mathfrak{S}}_\alpha$ indicates the "operation of the system modeled by the framework". Frameworks help us model the structure of a world of discourse. It is still necessary to populate this structure with the items from that world. Thus we distinguish between the *models* and the *instance*. When a framework is instantiated to reveal the underlying "real world", the signatures in the various $\boldsymbol{\mathfrak{S}}_\alpha$ become the operations that do the work of the system.

3. BUILDING ON FRAMEWORKS

This section first describes one minor addition to the base formalism and then describes how manipulations with a framework's formalism may provide abbreviations, constraints, and views.

3.1 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 $\boldsymbol{\mathfrak{F}}_\alpha$ an *annotation* function

$$\mathcal{A}: \qquad \mathcal{D} \rightarrow \mathcal{S}_{\mathcal{TRINGS}}$$

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

Annotations work best when members of $\mathcal{D}$ are used consistently across all frames of a framework. For example, we might require that $\mathcal{A}_\alpha(\texttt{framename})$ be defined for any non-empty frame $\boldsymbol{\mathfrak{F}}_\alpha$. 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 $\langle \mathsf{r}_1, \mathsf{i}_1, \texttt{Department} \rangle$.

3.2 Abbreviations

Other conventions 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 $\langle$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 in conjunction with Figure 2; 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`⟩".

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 meta-structure of the framework.

3.3 Framework Constraints

One of the major benefits of having a formal specification 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 framework models: *a priori* and derived. An *a priori* constraint is identified and explicitly declared during the modeling process. After the models have been developed, derived constraints appear 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 a relationship Ξ that 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.

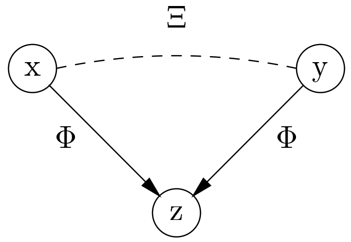

Figure 3: Ξ and Φ

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 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 $\mathfrak{F}$ is said to be *thorough* 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 $d \in \mathcal{D}_\alpha$.

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.

3.4 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.

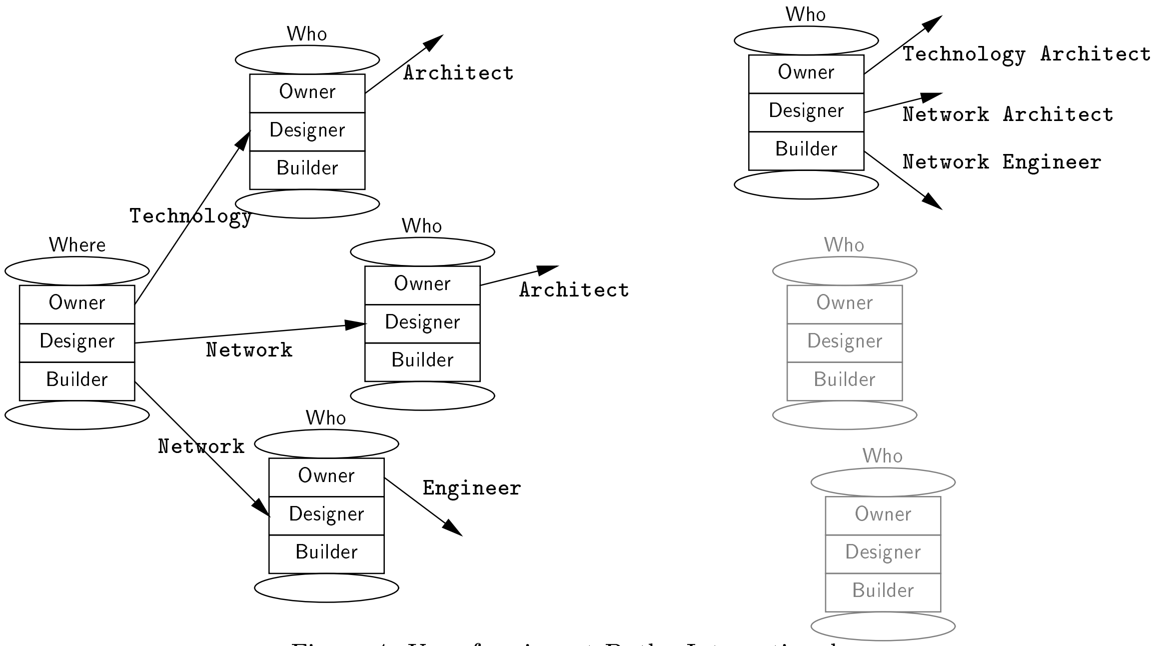

Figure 4: Use of a view at Butler International

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 arising 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 $\mathcal{P}\mathit{aths}$, a view is specified in terms of path label rewriting. That is, members of $\mathcal{P}\mathit{aths}$ 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.

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 [6]. 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

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

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 $\mathcal{VALIDJOBTITLE}$ and constrain $\mathcal{D}_\epsilon \subseteq \mathcal{VALIDJOBTITLE}$ in that view.

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. 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[8]). 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 be cast into a single Φ based on absolute, full-path names. Second, this Φ must be rewritten into a new $\check{\Phi}$. Finally, $\check{\Phi}$ must be projected down to localized $\check{\Phi}_\alpha$. The resulting $\check{\Phi}_\alpha$ may require yet further manipulation, in that they may violate constraints imposed on the original Φ_α. This can either be accommodated by restricting the rewriting rules (or their application) or be accepted as a consequence of the fact that views sometimes distort.

4. EXAMPLE

In this section we show how the framework formalism can be used to express a portion of IBM's Enterprise Solutions Structure (ESS) Project (entire issue of [2]). The set of descriptors ($\mathcal{D}$) is extracted from those articles and the organizational structure is placed into paths. The semantic analysis techniques of McDavid[5] are used to assign the **I** part of the paths. McDavid's Primary Business Concepts correspond to the standard **I** as follows: "what" $\approx$ "resources", "how" $\approx$ "behavior", "where" $\approx$ "location", "who" $\approx$ "delivery", and "why" $\approx$ "drivers". Note that no explicit correspondence is found for "when", although the ESS architecture exhibits many time-related specifications.

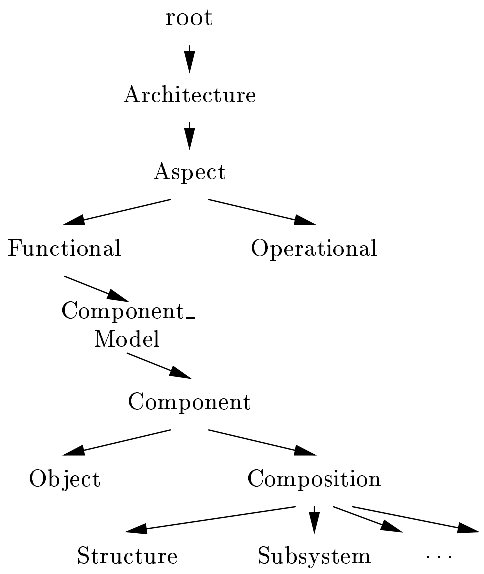

Figure 5: A fragment of the ESS framework

A fragment of the ESS meta-hierarchy is shown in Figure 5 with the **R** and **I** parts omitted. Figure 6 shows an idealized frame corresponding to "Composition" in Figure 5.

Notice the use of UML technology as the means for "building" components through the process of composition. The outputs of this frame, as expressed by the resulting UML models, are passed along as the input to a builder perspective one level up. The frame Composition supplies the design perspective detail for its parent the Component frame.

5. CONCLUSION

This paper examines a formal foundation for Enterprise Architecture Frameworks using recursion on the basic structure. We detail an explicit differentiation between the ordering derived from hierarchical decomposition and ordering related to **R**. Distinguishing these two ordering constructs focuses on roles

what	how	where	who	when	why
Structure	Dynamic_Behavior Enablement	Subsystem	Collaborations Role_Player	Sequence	Use_Case Situation Purpose
Object_State Partitions Allocations	Object_Interface Optimizing Templates	Grouping Overlap Platform_ Span	Performing_ Component Requesting_ Component Workflow	Interactions Operation_ Sequence Task_Sequence	Interface_Use Service_Level Commitment
UML_Class_ Diagram	Component_ Interaction_ Diagram	UML_Deployment_ Diagram	UML_ Collaboration_ Diagram	UML_ Sequence_ Diagram	UML_ Scenario

Figure 6: ESS frame with path ⟨conceptual, what, `Architecture`⟩ ⟨conceptual, what, `Aspect`⟩ ⟨logical, how, `Functional`⟩ ⟨logical, what, `Component_Model`⟩ ⟨physical, what, `Component`⟩ ⟨logical, how, `Composition`⟩

within the enterprise context and on decomposition within the architectural context.

Framework structure is separated from the application of the framework 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 structure from 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 picture.

The formalisms of our recursive-frame approach provide scalability by facilitating (1) focus on subdomains via partitionings of the set of descriptors, (2) analysis and presentation via multiple views, (3) reuse of models via frame naming and aliasing, (4) composing subframes via merging (detailed in [4]), and (5) joining frames at common descriptions to form larger and more comprehensive frame lattices via view mechanisms.

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), 1996.

2 *IBM Systems Journal*, 38(1), 1999.

3 W. Inmon, J. Zachman, and J. Geiger. *Data Stores, Data Warehousing, and the Zachman Framework*. McGraw-Hill, 1997.

4 R. Martin and E. Robertson. Formalization of multi-level Zachman frameworks. Technical report, Computer Science, Indiana Univ., 1999. `www.cs.indiana.edu/ftp/techreports/TR522.html`.

5 D. W. McDavid. A standard for business architecture description. *IBM Systems Journal*, 38(1):12–31, 1999.

6 H. Molina Noriega and E. Kopko. A consulting firm and its clients: many uses of the framework. In *ZIFA 1997 Annual Forum*, May 1997.

7 J. F. Sowa and J. A. Zachman. Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*, 31(3), 1992.

8 J. D. Ullman. *Princ. of Database and KnowledgeBase Systems*, volume I – Fundamental Concepts. Computer Science Press, New York, 1988.

9 J. A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3), 1987.

10 Zachman Institute for Framework Advancement. *The Zachman Framework*. PowerPoint presentation, `www.zifa.com/frmwork2.htm`.

Appendix

This Appendix formalizes the domains of several sets described in the paper. The formalisms define the respective sets only to the extent of specifying the allowable values each may contain.

1. Definition of $\mathcal{D}_\alpha$

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

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

2. Domain of Φ

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

$$\mathcal{EOC}_{\alpha,\mathsf{r}_0} = \mathcal{IC}_\alpha$$
$$\mathcal{EIC}_{\alpha,\mathsf{r}_{n+1}} = \mathcal{OC}_\alpha$$

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

$$\Phi_\alpha \subseteq \bigcup_{r\in\{\mathsf{r}_0\}\cup\mathbf{R}} \left(\mathcal{EOC}_{\alpha,r} \times \mathcal{EIC}_{\alpha,r'}\right)$$

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 $r \in \{\mathsf{r}_0\} \cup \mathbf{R}$ 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 r_0 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 r_{n+1}. In the first case, the above notation allows us to substitute $\mathcal{IC}_\alpha$ for $\mathcal{EOC}_{\alpha,\mathsf{r}_0}$, so that $\mathcal{EOC}_{\alpha,\mathsf{r}_0} \times \mathcal{EIC}_{\alpha,r_1}$ 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 as inputs to the next. In the second case, the product is $\mathcal{EOC}_{\alpha,r_n} \times \mathcal{OC}_\alpha$.

3. Domain of $\mathfrak{S}_\alpha$

Elements of $Types$ are the primary linkage between a framework as a collection of models 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 sets, lists, or bags of items, we use the notation $Types$ to specify how we deal with such instances.

$$Types_\alpha = \mathcal{IC}_\alpha \cup \mathcal{OC}_\alpha \cup \{\text{“COLLECTION OF } \lambda\text{”} : \lambda \in \mathcal{IC}_\alpha \cup \mathcal{OC}_\alpha\}$$
$$\mathfrak{S}_\alpha : \mathcal{D} \to \bigcup_{n\in\mathbb{N}} Types_\alpha^n$$