TECHNICAL REPORT NO. 594

Architectural Principles for Enterprise Frameworks

by
Richard A. Martin, Tinwisle Corp., Bloomington IN
Edward L. Robertson
John A. Springer

April 2004

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
Bloomington, Indiana 47405-4101

Architectural Principles for Enterprise Frameworks

Richard Martin
Tinwisle Corp.
205 N College Ave
Bloomington IN 47404
richardm@tinwisle.com

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

John Springer
Computer Science Dept.
Indiana University
Bloomington IN 47405
jospring@cs.indiana.edu

Abstract

This paper continues our work on the analysis and formalization of enterprise architecture frameworks, proposing a number of principles related to the construction and use of these frameworks. These principles are intended to guide the development of a formal foundation for frameworks. Enterprise architecture frameworks organize, manage, and interrelate the wide variety of models used to structure and operate an enterprise. The principles are drawn from analyses of a number of existing frameworks and from observation of and participation in framework development. Since these frameworks involve modeling, some of the principles apply to broader aspects of modeling; other principles apply only to frameworks. As the goal of this work is a requirements specification for formalization of frameworks, the paper ends with a sketch of how the identified principles might guide this formalization.

1. Introduction

An *enterprise architecture framework* is a means to understand an enterprise or class of enterprises by organizing and presenting artifacts that conceptualize and describe the enterprise. An *enterprise*[1] is a collective activity in a particular domain, with actors sharing a common purpose; an enterprise can be a business, a collection of businesses with a common market, a government agency, *etc.* *Architecture* is a metaphor to the realm of office towers and bridges, intended to capture the use-oriented, as opposed to construction-oriented, aspects of the design of those structures. A *framework* is a structured container for holding and interconnecting things[2] – in the remainder of this document those things

† Supported by NSF grant ISS-82407.
1. The word "organization" is a common synonym for enterprise, but we must often use "organization" to denote the way things are organized and thus restrict it to that use.
2. As another metaphor, think of a framework for electronic components which both holds circuit boards and provides for wiring between those boards.

are *artifacts* that comprise the enterprise architecture. In framework contexts, artifacts are almost always models of some kind, which we sometimes call "components" to indicate that they are pieces of the entire framework. In the following, "framework" will always be a shorthand for "enterprise architecture framework".

Frameworks have been widely used. The Information Technology Management Reform Act of 1997 led to the US Government's Federal Enterprise Architecture Framework (FEAF), which "describes an approach, including models and definitions, for developing and documenting architecture descriptions"[22]. It is being deployed in all non-military agencies of the US Government. The annual ZIFA Forums[33, register as a "friend"] have included nearly 100 case studies highlighting the benefits of frameworks. Bernus *et al.*[3] give several thorough case studies (along with an extensive discussion of enterprise architecture issues).

Whether the frameworks address manufacturing operations, process control, information systems, or government bureaucracy, the artifacts produced to describe the enterprise comprise a valuable asset requiring its own distinct management. Managing and gaining full value from that asset is the reason enterprise architecture frameworks are conceived, built, and used. Professional practice has taught us about the fragility of isolated application silos on islands of automation and about the difficulty in achieving interoperability under such circumstances. While these are typically called "data silos," the significant problem is that they are in fact model silos. That is, the mismatch of underlying models is the greatest impediment to interoperability.

In spite of their wide use and importance, frameworks have all been defined only descriptively. This means that it is currently impossible to formally relate different frameworks, to say nothing of implementing tools that properly support these frameworks.[3] This paper works toward correcting that deficiency, as part of a larger project which seeks to characterize and formalize frameworks.

This work is about frameworks in general and not about any one particular framework. Although our original motivation was the Zachman Framework for Enterprise Architecture[32,33], we examined and incorporated several other frameworks, which are itemized in section 2. Moreover, this work is about structure and not about contents. Thus "frame-

3. There are software packages that purport to implement various frameworks, but these packages only implement the "holding" aspect of frameworks. That is, they are tools for editing and managing representations which populate a framework instance, without respect to the semantics that the framework provides.

work” by itself indicates a collection of descriptions and principles for organizing framework contents while “framework instance” indicates the use of a framework describing one particular enterprise. In the latter context, the prefix “meta-” is used to explicitly indicate some non-instance aspect.[4]

The primary goal of this paper is the development of principles for guidance in understanding frameworks and for formalizing the use of frameworks to support organization and interaction of the many models associated with an enterprise. This work therefore continues our effort to formalize the ways in which these particular frameworks manifest the architecture of an enterprise[18], with an eye toward (*i*) connecting a framework instance’s contents, (*ii*) manipulating those contents and connections, and hence (*iii*) relating different frameworks and recasting instances from one framework standard to another. While our primary goal in developing these principles is to use them to guide our formalization activities, we hope that many are directly useful in the development of individual frameworks.

Section 2 begins this paper with a discussion of the origin and (to the extent possible) validation of the principles. Section 3 introduces a few principles that are general in nature, applicable to any modeling and analysis endeavor,[5] while section 4 discusses principles especially pertinent to frameworks. We then conclude this document by considering how these principles guide the formalization of frameworks.

2. Origins of the Principles

The principles described below come from (*i*) evaluation and comparison of different frameworks, (*ii*) observation of the process of defining frameworks, and (*iii*) participation in this same process.

Principles are largely based on analysis of the framework architectures: Zachman[33], a revision to the European pre-standard ENV 40003:1990 *Computer Integrated Manufacturing: Systems Architecture Framework for Modeling*[27], ISO Standard 15288 *Systems Engineering – System Life Cycle Processes*[14], and the US Defense Department’s C4ISR

4. This terminology is borrowed from the world of relational databases, where “relation” typically indicates the general structure, or schema, while “relation instance” indicates such a structure containing particular instance values and where names of columns are called “meta-data” while the values in those columns are “data”.
5. We are still using “framework” as a shorthand for “enterprise architecture framework”, but it would be a valuable exercise to see which of these principles hold for other classes of frameworks.

Architecture Framework[5], an analysis which we reported in [19,20].

Principles are also based on professional observation and participation – often experience of the difficulties which arise when these principles are not followed. Draft working notes from ISO efforts illustrate such difficulties, as in the statement "Something is not very clear - the distinction between the interoperability of process models and the interoperability of processes"[16, Nov. 2003], which reflects principle 4 about meta-levels. Our own professional experience includes constructing and analyzing models in an enterprise context, teaching modeling, and participating in the development of international standards for enterprise architectures.[6]

We do not claim to have originated all these principles. Several are simply our statements of well-established suggestions (*e.g.* 6, "Do not hide architecture in methodology", which is a rephrasing of the data independence principle[4]). Principles reflecting some of the same concerns as ours have been identified elsewhere [7,15,28], although these other principles are largely directed at insuring the fidelity of the modeling process; intersections with our principles will be mentioned as they occur. We include them all because we intend this compendium as a basis for the formalization that we will briefly sketch in section 5.

Occasionally specific facts are given in evidence. Only a few principles can be supported so concisely. One such principle (11), that states the independence of three commonly correlated scales, is supported by examples high in one scale but low in another. Unfortunately, principles that describe general behavior do not admit such concise support. This is very loosely similar to the difference between existential and universal propositions, in that one instance proves the former.

Perhaps the most insightful principle is principle 10, which recognizes that decomposition uses both grids and trees. We first observed this duality in the context of adding detail within a Zachman framework[9], necessitating the use of recursion within a frame. This principle has been validated by its use in comparing frameworks[20] and its value in the development of international standards, particularly ISO 15704:2000 *Industrial Automation Systems - Requirements for Enterprise Architecture Methodologies*[13].

Many principles focus on highlighting and refining distinctions (such as principle 5, which distinguishes dependency and temporal order). They arise from observation of the

6. Richard Martin is convener of TC 184/SC 5/WG 1, "Modeling and architecture", of the International Standards Organization.

ways in which people model, and the successes and the difficulties encountered therein.

Principles may be descriptive, describing the way that model artifacts *are* constructed and organized, or prescriptive, recommending how they *should be.* However, prescriptive principles all began as observations of the form "People have trouble with ...." Prescriptive principles of course guide practice; but they also guide the formalization effort, indicating what should be facilitated or discouraged.

3. General Principles of Modeling

Modeling as we mean it is a conceptual exercise, only analogously related to physical modeling as in, say, model railroads.[7] Conceptual modeling does yield representations in a particular *medium*, not necessarily a medium with physical manifestations, but these are representations of the modeled concepts. Thus principles apply to both concepts and representations.

Each of the following principles begins with a <u>short phrase</u> (indicated in that manner) which identifies and hopefully summarizes the principle.

1 <u>Communication is a goal of modeling.</u> Models (including frameworks) are formal artifacts but they are developed and used by people. Therefore any modeling formalism must be robust and tractable in interaction with non-formal components - people. This principle is discussed at great length in [28] and related psychological factors are discussed in [25].

2 <u>Complexity tradeoff.</u> There is typically a tradeoff between complexity in the modeling medium and complexity in model instances constructed using that medium. That is, if the underlying mechanism is too simple, then instances become complex to compensate; if the mechanism is too complex, it becomes the plaything of a very few specialists. Modeling mechanisms therefore should be defined with an attempt to find a "sweet spot" where these complexities are in balance. The success of Entity-Relationship (ER) models is attributable to this balance[2]. Of course we must remember that different modeling efforts have different sweet spots. The Unified Modeling Language[21] is an aggregation of several modeling mechanisms, each of which seeks to establish a "sweet spot" with respect to the representational needs

7. We draw this distinction because, for most people, the first connotation of "make a model" is to construct a model railroad or something similar. Model railroads diminish function but primarily reduce physical scale; indeed, the first descriptor applied to a model railroad is its "gauge", or physical scale.

of the content being modeled.

3 Naming matters. Naming, *i.e.* the assignment of a string[8] to a concept or artifact, serves as the bridge between formal artifacts and human interpretation. That is, there are two sides to naming: "external" (relating to the real world) and "internal" (relating to the mechanism and models of a framework). Said another way, internal naming involves formal meaning while external naming involves human understanding of that meaning.

Because names serve a role in human communication as well as one related to formal structure, naming must be done with great care. Of course the formalism works equally well whether the names used are well understood by human participants or are merely nonsense terms, as long as meaning is unambiguous (This fact is quite beneficial, since it allows the focus to be on the human/ontological aspects of name choice). Naming has important (and sometimes unexpected) consequences because that name typically has other associations. Even professionals often use the same names with different meanings. Thus, the development of ontologies and ontological methods to manage naming is complementary to the study of formalisms for framework expression.

We experienced an interesting example of the impact of name choices at a recent WG 1 meeting[16, April 2004]. In discussions concerning principle 5, the conventional Zachman row identifiers "specify", "design", and "build" were considered too similar to lifecycle stages and thus easily thought of as chronology. The terms "conceptual", "logical", and "physical", equivalent in this context, were easily distinguished as a time-independent dimension.

Concern over naming has broadened the study of ontologies from a few philosophers to many technical, scientific, and management circles[8,23,10]. A particular focus for ontology use is the need to distinguish how term usage is related to context – a concern that must be addressed by formal frameworks[6,26].

4 Use "meta" with great care, because the term is seriously overloaded. This particularly applies when discussing *meta-levels.* This is particularly true because "meta" is a relative term, not an absolute.

One obvious example of the relativeness of "meta" is observable in the realm of ER

8. We do not use "label" because we want to restrict that term to a specific use.

modeling. There, the *meta-model* level decomposes all models into Entities and Relationships; the *model* level may decompose a particular model for corporations into Department, Employee, Project (instances of Entity), Works_For (instance of Relationship), *etc.*; the *model population* level (for a fixed corporation) into Sales, Human Resources, Accounting, *etc.* (instances of Department). Thus the model level is meta with respect to the model population and ER notation is the meta-meta level for the model population. Notice that "instance" is also a relative term, in that it does not indicate an absolute level but only the level below $\mathcal{X}$ when used in the phrase "instance of $\mathcal{X}$".[9] Also, "meta" is roughly the inverse of "instance of", in that the meta of an instance of $\mathcal{X}$ is in fact $\mathcal{X}$. However, since our interest focuses on models and meta-models, henceforth "instance" shall denote artifacts at the model level; that is, Department, Employee, Works_for, *etc* in the above example.

5 Dependency is not chronology. That is, just because $\mathcal{B}$ depends upon $\mathcal{A}$, it is not necessary that $\mathcal{B}$ follows $\mathcal{A}$ in time. For example, there is a dependency in the general activities of SALES $\rightsquigarrow$ SHIPPING $\rightsquigarrow$ PRODUCTION, in that the purpose of SHIPPING is to fulfill SALES and the purpose of PRODUCTION is to enable SHIPPING. While an individual sale (an operation deriving from SALES) is followed by a shipment, in an ongoing enterprise (where sales occur over a long time) it is not the case that SALES as a unit precedes SHIPPING. Moreover, an operation of PRODUCTION must occur before the corresponding SHIPPING and might even occur before the corresponding SALES event, even though the purpose of the first followed from the second. Indeed, this aspect of anticipation - separating timing from dependency - is a central reason for enterprise modeling (whether formal or informal). The widely-used PERT (or Critical Path) algorithm explicitly maps a network of dependencies into a schedule.

While much of the evidence for this principle comes out of difficulties arising when it is not followed, ISO 14258 *Industrial Automation Systems – Concepts and Rules for Enterprise Models*[12] makes this distinction explicit.

6 Do not hide architecture in methodology. It is wrong to bury characterizations of things in methods that are used to construct them. This is not to claim that methods do not constrain results (to claim so would be most foolish) but rather to observe that such constraints must be made explicit and external to the construction process. In particular, the architectural form should survive changes in method and technology.

9. By analogy, the "meta/instance" distinction should be thought of as similar to "up/down" rather than "top/bottom".

Difficulties arising from failure to follow this principle occur when methodology is framed in order to constrain the results produced. Results added from the outside then do not fit those constraints. The explicit "views" of ISO 15704 exemplify this problem; factoring these views out of the methodology motivates current efforts to amend this standard.

4. Principles Specific to Frameworks

7 Frameworks organize artifacts. A framework is a means to facilitate understanding of enterprises and to communicate that understanding, principally by organizing and connecting *artifacts* used to represent a particular enterprise. Frameworks help us to take very richly textured descriptive artifacts and arrange them for practical understanding. Frameworks help to simplify complex presentations which are composed of many inter-related artifacts. The organizational mechanism of a framework is primarily a collection of dimensions along which the artifacts are placed and hence classified. It is in the number and different natures of these dimensions that frameworks vary. Many further principles relate to the characterization of these dimensions. Zachman was the first to identify specific artifacts that belong in frameworks[32], followed soon by the PERA group[31].

8 Distinguish structure from connectivity. Structure and connectivity are distinct aspects of frameworks[10] and a framework formalization (or standard) should distinguish them. The clarity of this distinction directly impacts the quality of a framework; unfortunately many frameworks do not achieve their intended impacts because they do not exhibit this distinction with sufficient clarity. Furthermore, useful reorganizations of a framework (discussed below in terms of view definitions) can be tractably expressed when phrased in structural terms, whereas desired views involving connections may be difficult to specify and expensive to compute.

Modeling that confounds structure and relationship is prone to both inaccuracy and brittleness. Thus we strongly recommend that the models used within a framework exhibit the same distinction; fortunately many common models do, including entity–relationship, process–flow, personnel–reporting line in an organization chart. This

10. We find it helpful to visualize a computer room where frames both hold devices (servers, disk drives, communications interfaces, *etc.*) and provide channels for wiring these devices together. A second metaphor is between bone (structure) and muscle (connection); this emphasizes that operation largely occurs through the connections.

principle harkens back to the "data independence" dictum[11] which was so essential to the maturation of database management systems out of the more *ad hoc* realm of data processing.

9 Separate policy from mechanism. That is, policy should be found in framework contents and not framework structure. As their goal is to facilitate understanding, frameworks provide (structural and connective) mechanism rather than delineate policy concerning enterprise management. Within an instance of a framework, representation of such understanding may constrain the operation of a *particular* enterprise and that framework may of course define policies. This parallels a distinction between mechanism and policy that was popularized in the conceptualization of computer operating systems[17].

10 Two aspects of organization. There are two general ways in which items within a framework are (typically) arranged: (*i*) in an *ordinant* structure (that is, a table, grid, or matrix) or (*ii*) in a *decompositional* structure (that is, a tree). We call either of these *dimensions* of the arrangement. Dimensions of either kind are discrete[12] and ordinant dimensions typically have only a few *coordinate positions*. The coordinate positions of an ordinant dimension may be *ordered* (*e.g.* rank) or *unordered* (*e.g.* gender), while a decompositional dimension is always ordered only by its containment relation.

The significance of this principle is beyond merely that both structures occur – they co-occur regularly. In the enterprise context, we are aware of this first in Inmon *et al.*[9], where the Zachman grid structure appeared at all levels of a refinement tree (see also principle 13).

An important step in organizing artifacts is to identify and characterize (as ordinant or decompositional) the dimensions that define the structure. The definition of an ordinant dimension is the identification of its coordinates and, where relevant, the order of those coordinates. Recall that dimensions only describe the placement of items (in a real or conceptual space) and not the interconnection of these items, which is typically much richer and more complex.

11. Data independence is a content-representation distinction, so the analogy made here is not to the particular distinction but to the great benefit derived from such clean distinctions.
12. This statement necessarily holds for decompositional dimensions but is sometimes relevant to distinguish meta-coordinates from instance coordinates where ordinant dimensions are involved.

Individual artifacts, in turn, are identified within a framework by name. A name can indicate a coordinate position along a particular ordinant dimension or indicate one member of a collection. When a string is used as a name in one of these contexts, we will refer to it as a *label*. Such a label has meaning fixed by formalism within a formal context; but when viewed in isolation that fixed meaning may be lost. An ordered dimension induces an ordering on its coordinate labels and we typically do not distinguish between these orders.

11 Three aspects of scale. There are (at least) three distinct dimensions that reflect conceptual (as opposed to physical) scale: (*i*) abstractness, ranging from abstract to concrete, (*ii*) scope, from general (generic) to special (specific), and (*iii*) refinement, from coarse to fine. Using the terminology of principle 10, abstractness, and scope are ordinant-ordered and refinement is decompositional.[13]

Because it is common to have co-occurrence of the origin or extreme endpoints in all three dimensions (as a module that is concrete, specific, and finely refined), these three dimensions are often confused. But they are in fact independent. Examples validating this independence occur in: (*i*) ER models (fully developed, with all attributes, relationship constraints, *etc*), which are both abstract and finely refined; (*ii*) ISO 19439, where the "Generic" plane includes items across the range from abstract to concrete; and (*iii*) C4ISR, where technically detailed products span a range of operational abstraction.

Understanding (and distinguishing) conceptual scales is essential because they govern the ways in which framework dimensions are conceived, ordered, populated, and constrained.

The fourth principle of Greenspan *et al.*[7, §2] focuses on abstraction and refinement, although without a firm distinction between the two.

The following principles seem less likely to guide practice than those itemized above. That is, they are more purely specifications for our intended formalism development.

12 One dimension manifests purpose within a framework. One, and typically only one, of a framework's ordinant-ordered dimensions reflects the purposive nature expressed within a framework. Note that such a "purposive dimension" does not represent the purpose of the framework (which is to support a particular methodology or standard,

13. In fact, refinement is often the canonical hierarchy.

and all dimensions should support that purpose) but instead represents the fact that artifacts derive their purpose from artifacts earlier in the dimension's order (most often through elaboration). Examples of such purposive dimension are *Role* in the Zachman framework, *Model Phase* in ISO 19439, *Process Group* in ISO 15288, and *Guidance* in C4ISR. Derived dimensions, produced through views (see principle 16 below), may also exhibit a purposive order; the C4ISR's "Force Integration" dimension, derived from a command-structure hierarchy, exhibits the purpose inherent in any chain of command.

The ordering of a purposive dimension often manifests itself as causality, dependency, or chronology. However, it is not merely a time dimension, even though purpose in a framework often leads to temporal ordering in the operations of the enterprise. This indeed follows from general principle 5.

Within framework instances, components at one level of the purposive dimension inform those at the next. Furthermore, connections running along the purposive dimension convey this purposive information. While connections can and do occur arbitrarily within a framework instance, they always occur along the purposive dimension.

13 <u>Refinement</u> <u>is</u> <u>recursive.</u> The decompositional scale dimension, refinement, is fundamentally different in that it works (or at least works best) through decomposition and successive refinement. Thus frameworks *should be* recursive in their application. Inmon *et al.*[9] illustrate a recursive refinement in an enterprise framework.

A major benefit of recursion in framework structure is that it directly supports a "drill-down" approach to framework development and exploration. To manage and comprehend the richness present in a framework, it is necessary to separate the artifacts such that detail is hidden until revealed for consideration. Recursion is the mechanism for providing this layered approach. Furthermore, recursion greatly facilitates building one unified framework out of several here-to-fore independent ones.

Unfortunately, practice often foreshortens the recursion, forcing a fixed (albeit hierarchical) or flattened structure.

14 <u>All</u> <u>context</u> <u>is</u> <u>relevant.</u> It seems necessary, as one moves through a framework along its purposive dimension, from row to row in a Zachman framework for example, that the entire framework structure at one row is *potentially* relevant when describing a

component at the next. This is not a claim that an entire row is in fact materially relevant for each component in the next; it is merely a recognition that all of the models from prior coordinates can be useful in understanding and constructing the next. Moreover, it is sometimes as important to know which concerns are not needed as it is to know which are. Perhaps this principle just reflects the fact that frameworks, and the enterprise domains with which they are concerned, are not suitable for minimally descriptive artifacts. Without considering the entirety of the previous row, redundancy is more difficult to eliminate and comprehensiveness is more difficult to achieve.

Evidence for this principle abounds. In the Zachman model, process designs draw not only from process specifications, but also from the information (ER) model, business rules, and even staffing considerations.

15 Connections can be of arbitrary arity. Connections between framework artifacts can be of arbitrary arity, although binary ones are most common. However, it is sufficient to provide for the construction of arbitrary connections using binary ones. For example, a Relationship in an ER model may be constructed to have any degree, but the basic connections are always between a single Entity and a single Relationship.

16 Views are important in standards and methodologies. A framework formalism should provide a general mechanism for defining views. Views are used in enterprise modeling because the complexity of an enterprise makes it impossible for a single descriptive representation to be humanly comprehensible in its entirety. The notion of view is inherent in any large, complex structure observed and managed by many individuals who neither can nor should attempt to analyze, design, or implement the entire structure. ISO 15704 requires four views: “Function”, “Information”, “Resource”, and “Organization”.[14]

The view mechanism should be general and dynamic. It must be general because there is little commonality of particular views across frameworks. Although 15704 defined four fixed views, recent deliberations on amendment of the standard have included consideration of additional fixed views, along with increasing awareness that views must be dynamically definable. Furthermore, views can be quite simple or very elaborate depending upon the intended use. The view mechanism should facilitate

14. Such views are often described as if they comprise a distinct dimension, but such a collection of views is an artifact of the process rather than part of the underlying framework.

dynamic extraction and restructuring of an enterprise model from various conceptual perspectives.

17 Construction through views. Views are not merely used for viewing; they are often used for constructing and populating frameworks. This is indeed an important reason for views in ISO 19439 *Enterprise Integration - Framework for Enterprise Modeling*[11]. Analogously, entities are placed in the ER model through the "information view" rather than into the complete framework. Thus the "view update" problem from the world of relational database reappears in the context of frameworks.

18 Constraint mechanisms are necessary. Framework standardization, as currently practiced, augments the frameworks themselves with voluminous texts constraining how frameworks are to be constructed or applied. In spite of considerable effort, such texts are inconsistent, ambiguous, and difficult to apply. Such application is of course limited by the degree to which constraints fall wholly within a framework, since a constraint that is even partially outside of the framework is not enforceable within the framework. Framework formalization should provide a foundation upon which unambiguous, concise, and effectively computable constraint mechanisms can and should be built.

   Beyond the simple observations that informal constraints exist and formal ones are highly desirable, at this point we can draw no further principles concerning constraints. Because current frameworks are largely structural, the constraints we observe are also structural. We do caution that constraints are also subject to considerations about meta-levels – in particular, model constraints must be distinguished from instance constraints.

The above principles characterize many of the frameworks that are concerned with domains at the enterprise level, although we have found no framework that exhibits all of these principles. Collectively, these principles constitute the foundation upon which useful enterprise frameworks are constructed.

5. Toward Framework Formalizations

While the previous sections discussed principles obtained from observation and analysis of existing frameworks, this section outlines how these principles guide formalizing enterprise frameworks. Although the individual framework instance is of course the formalized artifact, the following discussion is directed toward "architectural" standards that prescribe how a collection of frameworks is to be formalized.

There are four major aspects of a formalism that follow from the above principles. We itemize these four, justify why they should be treated distinctly, and then delve more deeply into each aspect.

structure: the way that components and sub-components of an enterprise are placed within a framework.

connections: the manner in which components and sub-components of an enterprise are interconnected within a framework. It is through these connections that the operations of an enterprise are manifest.

views: formal mechanisms for restructuring a framework to emphasize features from a particular conceptual or operational perspective.

constraints: formal mechanisms by which the conformance of a particular instance to a standard or architecture may be evaluated.

The deliberate separation of structure and connections is a direct consequence of principle 8. A framework is thus a structure for holding artifacts and a mechanism for connecting them.

The needs for views and constraints are enunciated in principles 16 and 18 respectively. While it is necessary to draw distinctions between structure and connections, it is advantageous to do the opposite, drawing parallels between views and constraints. In particular, the ability to define views immediately enables constraints definable in terms of views, as in "view A is a subset of view B".

A formalism for framework structures provides the foundation upon which formalizable, and therefore precise and coherent, view mechanisms can be built; and, conversely, view mechanisms provide the formalism through which one single overarching structure is coherently and consistently created by these many individuals.

5.1. Structure

A structural formalism is the mechanism through which sub-components of a component are both identified and organized. Because this formalism is symbolic, in the model space, identification and organization are not distinguished within the formalism, but are distinct in uses of the formalism. Specifically, the heart of a structural formalism is a set $\mathcal{C}$ of components and a function s that maps components (artifacts) into sub-components, designated by one or more identifiers.[15] In particular, $s : \mathcal{C} \times identifier^{\alpha} \longrightarrow \mathcal{C}$ (the choice of a value for α is discussed below). Thus the major difference between structural formalisms is in the way that identifiers (names, labels, ...) are used to select sub-components of a given component.

We now return to the ordinant/decompositional distinction of principle 10, recognizing that a complete formalism must provide both kinds of structure. Recall that ordinant dimensions classify components in a tabular structure, while decompositional dimensions indicate sub-components in a hierarchical structure. We consider each in turn.

A ordinant dimension has a fixed set of values and every component must be labeled by exactly one of those values. The number of dimensional identifiers and the label set for each are typically fixed by a framework standard or architecture (although a standard might specify a minimal set of labels, which could be expanded for particular instances). Thus a standard defines a k-dimensional space which components occupy. Note that these dimensions are effectively orthogonal, in that all k identifiers must be specified in order to place an artifact in this space.

A standard may specify that a particular dimension is ordered, so that the set of labels for that identifier is given a meaningful order (meaningful with respect to constraints, which will be discussed later). This order may also reflect the process in which frameworks and framework instances are conceptualized and constructed. Such a shift from dependency to chronology (*cf.* PERT) should be supported but not enforced (per principle 5) by the formalism.

Examples of ordinant dimensions include the Zachman framework's *interrogative* with values {"what", "how", "who", "where", "when", "why" } and the CEN Enterprise Inte-

15. The use of the word "identifier" here picks up two distinct meanings of the word. First, an identifier does indeed help identify subcomponents. Second, an identifier here is similar to an identifier used as a parameter in a program.

gration Framework's *genericity* with values { "generic", "partial", "particular" }.[16] Note that *genericity* is ordered, so the set notation here is inadequate for communicating all that is known about that dimension.

A decompositional dimension allows the decomposition of components into successively smaller sub-components. This decomposition naturally forms a hierarchy of components.[17] For example (at the model-population level), an enterprise typically has an organization chart with levels *enterprise* > *division* > *department* > *office* > *work_group* > *employee.* Since it is highly unlikely that each division will have exactly the same department as other divisions, this hierarchy does not lend itself to the structure of ordinant identifiers.

Although having more than one (independent) decompositional dimension is not explicitly prohibited, there does not seem to be any modeling advantage from multiple decompositions at a single meta-level. In fact, having more than one decompositional dimension is likely to make matters quite confusing. In our organization chart example, it would be very strange to say that a department were broken up, simultaneously and independently, into branches and offices at the same level in the organization chart, although independent parallel hierarchies of organization and location (*who* and *where* columns in a Zachman framework) are common.

The above discussion takes on a different flavor when multiple meta-levels are considered. That is, it is natural and expected that there be meta-level and model-level decompositions. For example, saying that the $\langle conceptual, what \rangle$ cell of a Zachman frame contains Entities and Relationships is a meta-level decomposition of that cell, while saying that Employee and Department are Entities is a model-level decomposition. Consequently, any formalism must provide mechanisms for multiple decompositions at different meta-levels. There are two ways to provide these mechanisms: first having multiple independent decompositional dimensions and second restricting the form of labels in particular cells. As an example of the second, we could provide for ER models by requiring that all labels in the $\langle conceptual, what \rangle$ cell be either of the form `Entity:*` or `Relationship:*`[18], thus achieving the specific model with labels including "`Entity:Employee`", "`Entity:Department`", and "`Relationship:WorksIn`". Because of this choice, there is a second parameter to

16. In an interesting comparison, Armour[1] has eight values along the genericity dimensions.
17. Note that "smaller" here only applies for components entirely within the enterprise; a decomposition may also reveal components in the enterprises scope. For example, decomposition of `computer system` reveals `internet`, an external component much larger than the enterprise being analyzed.
18. As in common convention, `*` indicates an arbitrary character string.

various formalisms: ℓ, the number of decompositional dimensions. Typically ℓ ranges from 1 (second method above) to 3 (first method with independent dimensions for meta-model, model, and model-population). While the name used for a decompositional identifier may vary with level and context ("division", "department", *etc*), it is better for the formalism to treat such identifiers uniformly independent of context.

Because the notion of enterprise is fluid, a collective activity may be considered an entire enterprise at one time and later may be considered just a sub-component of a larger enterprise. For example, individual businesses, once considered enterprises, may combine together in a larger supply-chain enterprise. For this reason, all components must have the same structural identifiers available for their decompositions, although all of them may not be meaningful at lower levels.

In summary, any component in a framework is located through the specification of successive layers of $k + \ell$-vectors of identifiers (that is, α above is $k + \ell$). The identifier vectors are thus edge labels for a tree which successively decomposes the entire enterprise into (typically) smaller and smaller components until the particular component is finally reached. An important notion of any formalism is therefore that of a *path*: a sequence of $(k + \ell)$-vectors that lead to a particular component. The component corresponding to a particular path is of course unique because s is a function, but there is no general requirement that each component has a single unique path that leads to it (although a particular formalism or standard may stipulate this). Thus the set of all paths forms a tree while the components lie in a directed graph (which seems necessarily acyclic in any sensible model).

5.2. Connections

Recall that principle 8 requires distinguishing structure, which reflects organziation of components and is represented using paths, from the connections between these components. Connections are required to express how components interrelate to each other in the actual enterprise being modeled.

For example, an instance of a Zachman framework may have "factory" and "warehouse" location components and a "transfer" process component. Since the location and process components reside in different cells in the framework, an additional mechanism is necessary to formalize the flow from factory to warehouse *via* the transfer process.

For uniformity and simplicity, the formalization of this connection is merely a binary relationship between components. If desired, this relationship may be extended to a third element, holding a label (character string) that is used to characterize particular pairs.[19] This triadic approach occurs in work ranging from C. S. Peirce[24] to the "semantic web"[30] and is supported by principle 15.

The issue of where to connect is still a thorny problem. While it is natural to assume that a binary connection goes from point to point, we have found it advantageous to model connections from one set of points to another set of points[18]. The metaphor of a framework as containing electronic components has already been introduced – now we see that some frameworks metaphorically have direct component to component wiring while other frameworks may, in addition, provide buses.[20]

5.3. Views

A general view mechanism must account for both structure and connections, but this does not mean that a view should be defined in terms of both structure and connections. Indeed, as noted in principle 8, matching view patterns that include connections is, in general, computationally expensive because connections create cycles in the underlying graph and pattern matches may loop around these cycles. Thus view definitions should employ only structural conditions, including conditions based on the existence of connections but excluding following connections, although the results returned may include connections. XSLT[29] makes this distinction clearly: conditions are expressed in patterns, results are returned by templates.

Any structural view can be accomplished by defining path expressions. This allows for simple views, such as projecting out a row or column with a particular identifier, as in the CEN "resource" view[27]. It also allows for substantially more complex views, such as gathering all components which share a common label or even rearranging tree structures. Fortunately there has been substantial work on path-based operations, particularly in the context of XML these days[29].

Incorporating connections in views is more difficult, not in preserving simple connections between components that are carried into the view but in defining new connections constructed through complex expressions involving existing connections.

It seems natural to expect more variety in view mechanisms than in other aspects

19. Said another way, connections are expressed as a graph or as a labeled multi-graph between components.
20. A bus is wiring that connects multiple components.

of formalisms. We certainly suggest that view mechanisms not be programmatic, that is defined by a typical imperative programming language with iteration and conditionals, but this approach cannot be ruled out.

5.4. Constraints

Considerations concerning contraints divide into targets ("what to constrain") and mechanisms ("how to constrain"). On the target side, framework constraints can apply to structure, connections, or even instance contents. On the mechanism side, many formalisms can be envisioned: operational/algebraic (*e.g.* regular expressions), first-order logic, or somewhere in between (*e.g.* PROLOG). Since formal equivalences certainly exist between various mechanisms, the choice of mechanism is governed by ease-of-use, in accordance with principle 2.

Of course the "what" and "how" issues cannot always be disentangled. This particularly applies when views are involved in constraint definition, a likely occurrence. A constraint can either be defined directly involving particular view(s) or through the same path-based operations as used in view definitions.

There are several obvious varieties of constraints: those that hold within a single path, those that hold between paths, and those that hold between sets of paths or the indicated components. The first two of these can be defined directly from path expressions, while the third seems more easily expressed using views. The example restricting $\langle conceptual, what\rangle$ cells to `Entity:*` or `Relationship:*`, given in section 5.1, is expressable by the first sort of constraint.

Constraints can even go into the semantics behind the model, as in a constraint requiring use of consistent clocks at various model levels. Such constraints seem beyond the scope of a general formalization of a framework. However, it seems advantageous to aid the definition of such an external condition, if this is possible. That is, there should be a way to declare and bind external functions and predicates.

6. Conclusion

We have identified 18 principles about the ways in which enterprise frameworks are or should be constructed and used, but this is only one step on a longer path. These principles will guide the formalization of frameworks, as discussed in section 5, but we are early in the work of that formalization. It is evident that the structure of a framework is carried by a tree whose nodes have a tabular, dimensional form, but many details governing the expression of structure and the interaction of this expression with connections, views, and constraints are yet unknown. Because existing frameworks do not treat connections in a disciplined manner, there is less guidance concerning connections from existing practice.

Finally, it is important that the formalization attempts to reach "sweet spots", as discussed in principle 2.

In as much as the principles enunciated herein are the core of a "requirement specication" for analysis and formalization of enterprise frameworks, we welcome all suggestions and comments.

Bibliography

1 Phillip Armour. Closing the learning application gap. *Commun. ACM*, 46(9):27–31, 2003.

2 C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin/Cummings, Redwood City, CA, 1992.

3 Peter Bernus, Laszlo Nemes, and Gunter Schmidt, editors. *Handbook on Enterprise Architecture*. Springer Verlag, 2003.

4 Chris J. Date. *An Introduction to Database Systems*. Addison-Wesley, 1981 ff.

5 Department of Defense – Architecure Working Group. C4ISR Architecture Framework, Version 2.0, 1997.

6 Marlène Gauvin, Anne-Claire Boury-Brisset, and Alain Auger. Context, ontology and portfolio: Key concepts for a situational awareness knowledge portal. In *Proceedings of the 37th Hawaii International Conference on Systems Sciences*, pages 111–120, 2004.

7 Sol J. Greenspan, John Mylopoulos, and Alexander Borgida. On formal requirements modeling languages: RML revisited. In *International Conference on Software Engineering*, pages 135–147, 1994.

8 W3C Web-Ontology Working Group. Owl web ontology language overview, 2004. `http://www.w3.org/TR/owl-features/`.

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

10 Inst. of Electrical and Electronic Engineers Standards Association, SUO Working Group. Standard Upper Ontology. `http://suo.ieee.org/`.

11 International Organization for Standardization. Enterprise Integration - Framework for Enterprise Modeling (ISO FDIS 19439). `www.iso.ch`.

12 International Organization for Standardization. Industrial Automation Systems – Concepts and Rules for Enterprise Models (ISO 14258)). `www.iso.ch`.

13 International Organization for Standardization. Industrial Automation Systems - Requirements for Enterprise Architecture Methodologies (ISO 15704:2000). `www.iso.ch`.

14 International Organization for Standardization. Systems Engineering – System Life Cycle Processes (ISO 15288). `www.iso.ch`.

15 International Organization for Standardization. TR 9007 Concepts and Terminology for the Conceptual Schema, 1987. No long available through `www.iso.ch`.

16 International Organization for Standardization TC 184, SC 5, WG1. Meeting Minutes. `forums.nema.org/~iso_tc184_sc5_wg1`.

17 R. Levin, E. Cohen, W. Corwin, F. Pollack, and W. Wulf. Policy/mechanism separation in Hydra. In *Proceedings of the fifth ACM symposium on Operating systems principles*, pages 132–140. ACM Press, 1975.

18 Richard Martin and Edward Robertson. Formalization of multi-level Zachman frameworks. Technical Report 522, Computer Science Dept., Indiana Univ., 1999. `www.cs.indiana.edu/ftp/techreports/TR522.html`.

19 Richard Martin and Edward Robertson. Frameworks: Comparison and correspondence for three archetypes. In *ZIFA 2002 Enterprise Architecture Forum*, 2002.

20 Richard Martin and Edward Robertson. A comparison of frameworks for enterprise architecture modeling. In *ER2003 - 22nd Intl. Conf. on Conceptual Modeling*, pages 562–564, 2003.

21 Object Management Group. Unified Modeling Language. `www.uml.org`.

22 U. S. General Accounting Office. Gao-03-584g information technology: A framework for assessing and improving enterprise architecture management, 2003.

23 Open Biological Ontologies. `http://obo.sourceforge.net/`.

24 Charles Sanders Peirce. On the algebra of logic. *Amer. J. of Math.*, pages 180–202, 1885.

25 Keng Siau. Information modeling and method engieering: A psychological perspective. *J. of Database Systems*, 10(4):44–50, 1999.

26 John F. Sowa. Laws, facts, and contexts: Foundations for multimodal reasoning, 2003. `www.jfsowa.com/pubs/laws.htm`.

27 The European Committee for Standardization. CEN ENV 40 003, Computer Integrated Manufacturing: Systems Architecture Framework for Modeling , 1990.

28 Terje Totland. *Enterprise Modeling as a Means to Support Human Sense-making and Communication in Organizations.* PhD thesis, Norwegian University of Science and Technology, Department of Computer and Information Science, August 1997.

29 W3C. XSL transformations (XSLT), 1999. `www.w3.org/TR/xslt/`.

30 W3C. RDF Model Theory, 2002. `www.w3.org/TR/rdf-mt/`.

31 T. J. Williams. Reference model for computer integrated manufacturing, a description from the viewpoint of industrial automation. Technical report, CIM Reference Model Committee, International Purdue Workshop on Industrial Computer Systems, Research Triangle Park, NC., 1989.

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

33 Zachman Institute for Framework Advancement. *The Zachman Framework.* Various pages at `www.zifa.com`; the "Quickstart" is particularly relevant.