

Fatal Distractions

By

John A. Zachman

Ó Copyright 2001 Zachman International

I am discovering four corruptions of architectural principles being inadvertently executed by well-meaning folks, who are thinking they are doing “Enterprise Architecture,” using the “Zachman Framework,” but that inevitably will lead them unsuspectingly into a cataclysmic disaster.

First, they are renaming the Rows or Columns of the Framework in an effort to accommodate the current inventory of artifacts created during the Industrial Age.

Second, they are taking model-based approaches to implementations and presuming that the models constitute the Enterprise Architecture.

Third, they are “integrating” Frameworks by a process of aggregation.

Fourth, they are separating the services of the Enterprise from the Enterprise on the basis that the Service is the equivalent of a Product.

Each of the above is a kind-of Architectural malapropism.¹ They all sound like Architecture but they actually are not Architecture and invariably will result in the classic “Silver Bullet” syndrome: “The Zachman Framework? Well, we tried that and that didn’t work!!” Or, even worse, “Enterprise Architecture? Well, we tried that and that didn’t work!!” Then it will be a decade or more before that Enterprise will be able to re-address this incredibly critical issue so vital to its survival in the Information Age.

These are the “Fatal Distractions!”

Denormalizing the Framework

First: Changing the names of the Rows and/or Columns and/or Cells of the Framework.

Although this does not appear to be a very significant issue, it has a profound impact on the integrity of the logic of the Framework. In fact, it denormalizes the Framework. It creates redundancies and discontinuities and leads to drawing erroneous conclusions.

For example, I have seen where some folks have renamed the Process Column of the Framework to “Services.” Here is the problem: “Services” do not equal “Processes.” It is NOT “Input – Service – Output.” It is Input – PROCESS – Output. Process is a

¹ A “malapropism” is the misuse of a word, especially by confusion with one of similar sound.

“transformation”. You take something in, do something to it (transform it) and send something different out. Service is not a transformation and therefore is not answering the primitive question “How” does the Enterprise work. The “How” question is the basis for all of the Column 2 “Process” models. Service is a “view,” a collection of some components of the Enterprise that are intended to provide value to a Customer. Clearly, “Service” does not equal “Process.” It is NOT a “transformation”. It is NOT the answer to the question “How?”

Also, I have seen folks who rename Column 1 to “Information.” This is a throwback to the old Industrial Age days when the end object of I/S was to build and run (computer) systems. The idea was to replace people with machines because machines (computers) were “better, faster and cheaper” than people. The objective was to automate the process and get to implementation as soon as possible and therefore improve the per unit productivity of the Enterprise. Therefore, data was always defined in the context of a process. We SUB-optimized the Enterprise data by optimizing the implementation.

Data in the context of its use (a process) is “Information.” Information is a “view” of data. This is the fundamental problem with the “legacy” today. The “data” that served as input or output to processes were collections of data elements defined relative to the process, that is “information” or a “view” of the total set of Enterprise data. Since the total set of Enterprise data was never defined, the implemented “views” were overlapping, redundant, inconsistently defined, named and formatted. That is, in the current inventory of systems, the same data is in there “n” different times with “x” different names, “y” different formats and worst of all, “z” different meanings. This is not only costing an arm and a leg to maintain but it is causing a major amount of confusion and frustration for Management because they keep getting different answers to the same question and don’t have a warm feeling that they know what is going on!

Clearly, you don’t want to try to classify data by its usage (Information). From a finite set of data, you can create an INFINITE amount of information. Information is a “beauty in the eye of the beholder” problem. Everyone has their own peculiar variation. There is no standard. Therefore, you have to classify data by its intrinsic meaning, independently of its usage if you ever hope to achieve “reusability” and semantic continuity, that is, if you ever want to get rid of the “Tower of Babel” problem and “interfacing” (“spaghetti”) in the Enterprise. The data has to be “normalized.” In fact, anything you want to actually “manage” has to be “normalized,” that is, defined in its non-redundant, coherent, integrated, reusable, Enterprise-wide, natural, “primitive,” state BEFORE you create composite views of it ...think of money (chart of accounts), think of people (organization), think of parts (bills-of-materials), think of chemicals (periodic table), think of anything you want to manage ... think of data (semantic models)!

This is the factor that led to the demise of IBM’s BSP (Business Systems Planning) methodology. Literally thousands of BSP Studies were conducted throughout the world, most of which were never successful. In the ‘70’s, people simply did not understand the difference between data and information and therefore attempted to classify the data based on its usage, not on its intrinsic semantic structure. The “data versus process”

matrix (which is at the heart of any/all formal information strategy analyses) was actually an “*information* versus process” matrix, which was meaningless because the same data was showing up in bunches of “informations” on the “Thing” (data) axis of the matrix. Therefore, BSP was simply more of the same. The data was classified by usage ... redundant, inconsistent, incoherent, “information,” “views,” related to processes and implemented. The “data versus process” matrix was actually a “process versus process” matrix. Nothing had changed! And, as you might expect, BSP fell prey to the Silver Bullet problem. “Well, we tried that and that didn’t work!” So, 10 years later BSP was renamed ISP (Information Strategy Plan) which fell prey to the same Silver Bullet problem again. So now, 10 years later, we are calling the information strategy analysis EAP (Enterprise Architecture Planning), which will also fall prey to the Silver Bullet problem unless we can see the data as representative of the Things of the Enterprise, classified by their intrinsic, “primitive,” semantic structure rather than information “views” classified by its process usage.

In any case, “Information” does not equal “Things.” “Things” constitutes the basis for the Column 1 models as “Things” answers the primitive question “What” “Things” does the Enterprise care about enough to manage? The structure of the Enterprise “Things” is the “semantic structure” of the Enterprise, the basis for “classifying” the data by its intrinsic meaning quite independent from its usage (processes). “Information” is infinite, defined in the context of its use and therefore not standard, not integrated, not reusable, not normalized and not “architected.”

Now you can see the effect of renaming Column 1 to “Information” and Column 2 to “Service:”

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
NOT What	NOT How	Where	Who	When	Why

This is NOT the “Zachman Framework.” Even if you diligently populate all of the resultant cells with models, you will not answer the 6 primitive questions about the Enterprise from which you will be able to derive answers to any question anyone asks about the Enterprise. You will not have the total “knowledge base” for the Enterprise. You will have a mixture of redundant, inconsistent, disintegrated stuff ... not a repository of unique, coherent, non-redundant, normalized, primitive components. All you would have is a matrix, not a logical, normalized structure. In fact, in the first two columns you would likely have “composite” models which leads me to the next point, but before I address that, let me observe: you may THINK you would be doing Architecture by populating the cells of the altered Zachman Framework but actually it is not the Zachman Framework at all. It is just a denormalized matrix and actually, you would only be doing more of the same, the same we have been doing for the last fifty years that resulted in the legacy. You would simply end up with more legacy!

It is inevitable, at some point the Enterprise is going to wake up and say ... “Well, the Zachman Framework. We tried that and that didn’t work!!” Or worse, “Enterprise

Architecture. We tried that and that didn't work." Then you will have to wait for 10 more years to have another crack at solving the problem.^{2,3}

Composites Versus Primitives

This is an insidious problem because composite models are the work products of "application development."⁴ We know how to do these. These are the kinds of models that we have been building for the last 50 years (that is, if we have built any models, they have tended to be composite models.) Anyone who's intent is to build and run systems KNOWS that Enterprise-wide, primitive models (especially Enterprise-wide ones) take too long, cost too much and are difficult to produce. They are absolutely right. Composite models are all you need to get to implementation as soon as possible. You don't need primitive, normalized models.

Primitive models take longer and cost more and are harder to produce because you are trying to "engineer" them such that they are "normalized," you only have one primitive concept in there one time, so that the primitive concepts can be shared or "reused," so that you have Enterprise-wide integrity, so that every time you need some concept, you reuse the same concept, so that you can reduce the time for ensuing implementations to virtually zero, that is, so you can "ASSEMBLE the Enterprise TO ORDER," so you can change the Enterprise with minimum time, disruption and cost, so you have a knowledgebase for the Enterprise from which you can answer any question that anyone needs to ask, etc., etc., etc.

The word "primitive" comes from the idea of the 6 "primitive" interrogatives: What, How, Where, Who, When, and Why. Each of these interrogatives is an independent variable and the total of the 6 constitutes a complete, "normalized," primitive set. For an Enterprise, this translates into Things, Processes, Locations, People, Time and Motivation, the Columns of the "Zachman Framework." For the complete Enterprise description, you would have to build each of these models Conceptually (from the perspective of the business, "Owners"), Logically (from the perspective of the systems, "Designers") and Physically (from the perspective of the technology "Builders".) You

² Note: There are similar problems with renaming either the Rows or the Cells but this article is becoming too long to elaborate those issues. Let it be sufficient to say, if you get Conceptual, Logical and Physical things in the same model (a vertical composite), now you have confusion! Also, if you rename a cell to accommodate a "composite" (like rename Column 4, Row 2 from "Work Flow" to "Swim Lane") now you have real confusion ... see the next section, "Composites Versus Primitives."

³ Note on renaming Rows: There are people who want to rename Row 3 to "Architect's Perspective" and Row 4 to "Designer's Perspective." I know this is more consistent with common usage in the Information Technology community today, but the Architecture problem is an ENTERPRISE problem, and we have to communicate it to Enterprise people, and "Owner, Designer and Builder" sings to the Enterprise. Furthermore, you would have a very hard time convincing an Architect (who builds buildings - doing Row 3 work) that he was NOT "Designing" ... or an Engineer that was designing airplanes that he was not "Designing." Likewise, you would have an equally difficult time convincing a General Contractor who was building buildings that he was NOT a "Builder" and a Manufacturing Engineer that was figuring out how to build the airplane that he was not a "Builder." I would not change the names to conform to the information community's current usage. It will just add confusion!

⁴ I wrote a whole article about this, "Architecture Artifacts versus Application Development Artifacts" that can be found at www.brcommunity.com.

would have to add the Scope and Out of Context perspectives to complete the Framework but this article does not intend to be a tutorial on the Framework. You can find that in other references.⁵

The point is, each cell of the Framework constitutes a “Primitive Model,” a single variable from a single perspective. A “Primitive Model” is composed of “Primitive Components.” For example, in the Enterprise’s Semantic (“Primitive”) Model, it is composed of “Things” (“Primitive Components”) and Relationships between the “Things”. If you are not describing “Primitive Components” that constitute “Primitive Models”, you are not doing “Architecture.” Furthermore, if you are not defining the Primitive Model across the scope of the entire Enterprise, that is, ‘Enterprise-wide’, then you are only going to realize the reuse of the Primitive Components” WITHIN the scope that you have defined the Primitive Model. If you want reuse (“normalization,” integration, etc.) across the scope of the ENTIRE Enterprise, then the Primitive Model must be defined Enterprise-wide. Reuse (“Integration, “Normalization, “Standard Interchangeable Parts” etc.) does NOT happen by accident. It has to be ENGINEERED.⁶ (Please read the footnote below!!!)

Primitives are ARCHITECTURE which are engineered to be used in more than one implementation. Composites are simply application development artifacts comprised of components from more than one Primitive Cell Model which are necessary for a single implementation. Architects produce primitives. Systems designers produce composites.

There is nothing the matter with composites. You need composites for implementation. All you need for implementation are composites. Composites are good. But, if you create the composite ad hoc, that is, for the purpose of one implementation, then the composite is only good for that implementation, that is, it is good as long as nothing changes. You have created (and “locked in concrete”) a “point-in-time” solution, a “view.” This is the problem with the legacy today. It is a conglomeration of composite, point-in-time solution “stovepipes” that are costing an arm and a leg to maintain and ultimately will have to be replaced, if (when) anything in the Enterprise changes. We optimized the implementation and sub-optimized the Enterprise. (Remember however, there were substantial SHORT TERM benefits to this! We got to implementation, system by system, as soon as possible ... but we DIS-integrated, SUB-optimized, DE-normalized the Enterprise in the process.)

⁵ See my bibliography which can be found at www.zifa.com.

⁶ Notice!!! I am not saying you have to build out all 30 cells Enterprise-wide before you can implement anything. You can compromise the overall Enterprise integrity, but you should clearly understand it is a COMPROMISE and understand the implications of the compromise. Some cells of the Framework you can compromise with impunity (e.g. Columns 2 and 4). Other cells you should hold the line determinedly (e.g. Columns 1, 3 and 6). Sometimes you can mitigate the downstream impacts of a compromise (e.g. “promote slivers to Enterprise-wide Quality”). Sometimes there are ways to segment the cell (break it up into parts) without causing disintegration (sub-optimization) (e.g. determine build sequence based on entity-dependency). These are decisions and choices that balance the long term and the short term and should be made as a part of your Enterprise Architecture Strategy.

On the other hand, if you have engineered primitives and assemble the composites from the primitives, then you can assemble virtually any composite you need dynamically, reduce time-to-market for implementation immeasurably and you can change the Enterprise at will with minimum time, disruption and cost. (These are even greater LONG TERM benefits.)

You can see the problem. If you are building models and you think you are doing Architecture and you are not building Primitive Models, you are just building systems ... implementing, NOT architecting. Worse, if you have renamed the first two Columns of the Framework to Information and Services, you are justifying the use of the composites and convincing yourself that you are doing Architecture. You THINK you are okay, but in reality, you are right back where we have been for the last 50 years. You are just building more legacy.

It is only a matter of time before the Enterprise wakes up as says, "Well, Enterprise Architecture and the Zachman Framework! We tried that and that didn't work!! All we got is more of what we already have, legacy!!"

Accidental Integration by Magic

This is another insidious problem.

Some Enterprises are so big and so complex that it is intimidating to try to think about applying the concept of Architecture to the entire Enterprise, that is, one Framework for the whole Enterprise. In fact, many times the organization of a large Enterprise that produces many diverse products or services will evolve such that stand-alone business units are created for decentralized management of the product or service (which tends to be unique) and centralized organizations are created to manage those aspects of the Enterprise that tend to be "common" or "shared," like "infrastructure." In fact, in really big Enterprises they may group the business units into "Groups" or "Clusters" where there is at least some "sharing" potential around the products or services. So, you could have "Business Units" (or "Divisions" or "Departments") grouped into "Groups" or "Clusters" (where there was potential leverage for sharing, typically around the products or services or customers ... wherever there was "sameness") and then some central "Enterprise" group for managing what is common to all the "Groups." Some things are common to all ("Enterprise"), some things are shared by some ("Group/Cluster") and some things are unique ("Strategic Business Units," or "Divisions," or "Profit Centers," or "Departments," etc.)⁷

If you are going to use the Framework to help you think about this kind of a complex Enterprise, you could apply the Framework logic to every single business unit on a stand-alone basis. "Enterprise-wide" would mean Business Unit-wide and the analysis would be containable. The General Manager of that Business Unit would have jurisdictional control over the models within one Business Unit and could unilaterally declare their structure and description. However, in this case, you would not be able to realize any

⁷ This is typical in a large Public Sector organization.

benefits of sharing, commonalities, reuse, etc. To compensate, you could group Business Units into Clusters where there is likely some potential sharing and deliberately search for it and then engineer for it. The whole Enterprise would be comprised of the set of Clusters which, in turn, could be examined and engineered for commonality.

The Business Unit Frameworks are “Peer” Frameworks because they are all Frameworks for Enterprises. Wherever you have “Peer” Frameworks all in the same Enterprise, they all could potentially be “integrated” into one Enterprise Framework. It is convenient to consider the Business Units separately for analytical purposes (above), but where you want sharing to take place, or where you want to leverage commonality, those slivers of those cells of the Business Unit Frameworks that are to be shared or common have to be made THE SAME. Otherwise, they are not going to be “shared” or “common” or “INTEGRATED.”

You don’t get “integration” by accident. If you WANT integration, then you have to ENGINEER integration. That means that you will have to DICTATE to the Business Units that you want such and such slivers of such and such Cells of their Frameworks to all be THE SAME. Alternatively, you could require that the Business Units all get together and NEGOTIATE the common slivers of cells where there is leverage for sharing or commonality. The former approach (dictate) is faster. The latter approach (negotiate) produces more commitment. If you don’t either dictate or negotiate (one or the other), you are not going to get integration, that is, you are not going to get sharing or commonality or reuse. And ... if you are trying to address sharing or commonality with composites ... GOOD LUCK!! There are an infinite variety of composites! The only way you are ever going to get there (reuse) is with PRIMITIVES. (There are an infinite number of composites and everybody has their own peculiar twist on how to compose them, whereas there are a finite number of primitives and at the primitive component level you can see, or agree to, their sameness.)

Furthermore, you have to define what is to be shared or common BEFORE you get to implementation, that is, before you get any systems built. After you get the systems built, there is NO WAY to post integrate them. You are into “throw those things away and start over again (‘scrap and re-work’).”⁸ This is precisely where we are with the legacy today!! It is not integrated and it is frustrating and costing a lot of money to keep running and no amount of more technology is going to fix it and nobody wants to hear that it has to be scrapped and re-worked and the stress levels are out-of-sight!! (Note: if it was easy to post-integrate systems after they are built, you could just post-integrate the legacy and you wouldn’t have a legacy!)

Could you use the Framework to help you think about sharing at the Cluster level? Sure! This is what I have always referred to as “Sameness Templates.” You could define a Framework “template” either through declaration (dictation) or negotiation that would specify which slivers (or slices) of which Cells you want to make the same across a set of Business Unit Frameworks. You could actually define a Cluster Framework in its own right and then you would have to figure out a way to ensure that the Business units

⁸ See my article “Architecture Artifacts versus Application Development Artifacts.” Ibid.

actually used (or reused) the primitive components that were defined by the Cluster Framework in their own (Business Unit) implementations. It would actually be to the Business Unit's advantage to reuse components that were already defined and that were in fact THE SAME as they needed. It would be a LOT cheaper and faster than building them all themselves!!

Likewise, you could define a "Sameness Template" at the Enterprise level for defining what you want to be the same across all Clusters. The same logic holds. The Enterprise Framework (Sameness Template) is likely related to "infrastructure management," what some people are calling "internal services."

Here is the insidious problem. If you think you can collapse (or merge) several peer Frameworks into a single Framework and then by some magic, they are automatically integrated, you are in for a rude awakening. Worse ... if you think you can define composite models, map them against the Business Unit Framework and then collapse several Business Unit Frameworks into a single Cluster Framework and by some magic the models will be factored into primitive models and mystically integrated with all the other Business Unit models without doing any "engineering" (that is, Architecture), you are in for an even bigger surprise!!

This was at the heart of the tragic demise of the IBM Repository a decade or more ago!! They thought they could take the Bachman metamodel, the KnowledgeWare metamodel and the Eccelerator metamodel, (which were all different metamodels) merge them (collapse them) into a single metamodel and by some magic and mystical process they became automatically "integrated." If that wasn't bad enough, then they thought they could dump CASE tool COMPOSITE models from any CASE tool known to mankind (all with their own proprietary metamodels) into the data base derived from the merged (not architected) IBM Repository metamodel and the CASE tool COMPOSITE models would somehow be magically and mystically factored into primitive components and integrated (shared, reused, normalized, common, etc.)!!

Please be careful on this one!! Before we criticize the spec in IBM's eye, we better take careful look at the BOARD in our own eye!! Are we making an assumption that integration (sharing, reusability, commonality, etc., etc.) can be achieved with no engineering (Architecture) work, simply by collapsing (or merging) Frameworks together?

Integration (reuse, sharing, etc.) does not happen by accident and it is not magic. It has to be ENGINEERED ... or maybe I should say, ARCHITECTED!

Need I point out once again where this collapsing or merging peer Frameworks with no primitive engineering and no edicted or negotiated standards will lead? It is only a matter of time before it will be, "Well, all we have is more legacy! Enterprise Architecture, the Zachman Framework, we tried that. That didn't work!!"

Architects Frameworks and Service Frameworks

Here is the fourth problem.

There are two (and only 2) possible relationships between Frameworks that exist within an Enterprise. Either the Framework metamodels are the same, in which case, they are “Peer” Frameworks or the Framework metamodels are different, in which case they are “Meta” Frameworks. Within an Enterprise (or within an Enterprise Cluster) if two Frameworks have the same metamodel, then they are candidates for integration. That is, it is extremely likely that there are at least some components of the primitive models of those two Business Units’ Frameworks that can be shared between them. That is, between them, you can leverage reusability, integration, interoperability, integration, etc. to their advantage.

On the other hand, if the metamodels of two Frameworks within the same Enterprise are different, the high probability is that the Row 2 models of one of the Frameworks have a meta relationship with all the Cells of the other Framework. For example, one Framework may be the set of descriptive representations for the Product of the Enterprise and the other may be the set of descriptive representations for the Enterprise itself, in which case the Row 2 models of the Enterprise Framework are going to be models of the models of the Product Framework (metamodels), because the Enterprise is manufacturing the Product and has to produce the Product Framework full of models in the process of manufacturing it.

For example, see Figure 1, sample lists of things you would expect to find in the Row 2 models of a Product Framework.

<u>Column 1</u>	<u>Column 2</u>	<u>Column 3</u>	<u>Column 4</u>	<u>Column 5</u>	<u>Column 6</u>
“What?”	“How?”	“Where?”	“Who?”	“When?”	“Why?”
Screws	Glide	Curves	Pilot	Maint. Cyc.	Size
Nuts	Climb	Splines	Tail Gunner	Control Cyc.	Speed
Bolts	Dive	Tangents	Navigator	Ignition Cyc.	Range
Radios	Accelerate	Angles	Passenger	Fueling Cyc.	Payload
Wings	Take-off	Switches	Bombardier	Auto-Pilot Cyc.	Flexibility
etc.	etc.	etc.	etc.	etc.	etc.

Figure 1: Sample lists of things you would expect to find in **Row 2 Models** of a **Product Framework**

Figure 2 is a sample list of things you would expect to find in a **metamodel** of the Row 2 models of a **Product Framework**.

Parts
 Functional Specs
 Drawings
 Customers
 Timing Diagrams
 Design Objectives
 etc.

Figure 2: Sample (partial) list of things you would expect to find in a **metamodel** of Row 2 Models of a **Product Framework**

Figure 3 is a sample of lists of things you would expect to find in the Row 2 models of an **Enterprise Framework**.

<u>Column 1</u> “What?”	<u>Column 2</u> “How?”	<u>Column 3</u> “Where?”	<u>Column 4</u> “Who?”	<u>Column 5</u> “When?”	<u>Column 6</u> “Why?”
Parts	Research	Los Angeles	R & D	Economic Cyc.	Growth
Functional Specs	Design	Seattle	Engineering	Product Cyc.	Profit
Drawings	Produce	Dallas	Mfg. Engineering	Budget Cyc.	Quality
Customers	Distribute	Long Beach	Production	Acquisition Cyc.	Time
Timing Diagrams	Sell	St. Louis	Product Support	Sell Cyc.	Cost
Design Objectives	Support	Philadelphia	Marketing	Eng. Release Cyc.	Flexibility
etc.	etc.	etc.	etc.	etc.	etc.

Figure 3: Sample lists of things you would expect to find in **Row 2 Models** of an **Enterprise Framework**

Figure 4 is a sample list of things you would expect to find in a **metamodel** of the Row 2 models of an **Enterprise Framework**.

- Business Entities
- Business Processes
- Business Locations
- Organizations
- Business Events/Cycles
- Business Objectives
- etc.

Figure 4: Sample (partial) list of things you would expect to find in a **metamodel** of Row 2 Models of an **Enterprise Framework**

The metamodel of the Enterprise Framework (Figure 4) is **DIFFERENT** from the metamodel of the Product Framework. (See Figure 5.)

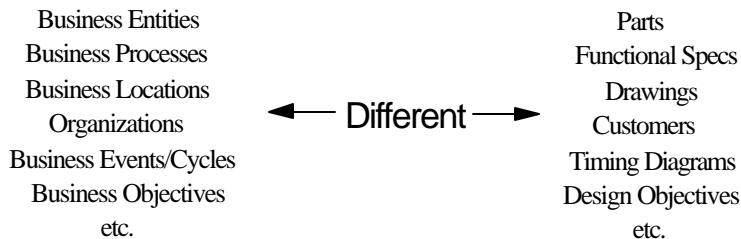


Figure 4: Sample list of things you would expect to find in a **metamodel** of Row 2 Models of an **Enterprise Framework**

Figure 2: Sample list of things you would expect to find in a **metamodel** of Row 2 Models of a **Product Framework**

Figure 5: The metamodel of the Enterprise Framework is **DIFFERENT** from the metamodel of the Product Framework.

The Column 1, Row 2 model of the Enterprise Framework is the metamodel of the Product Framework (See Figure 2 and Figure 3) because in order to manufacture the

Product, the Enterprise has to define the standard manner in which to express each of the Cells of the Product Framework. The Customer of the Enterprise (the “Owner” of the Product) will play a very significant part in specifying the CONTENTS of each of the Row 2 Cells for the Product, but the Enterprise will specify the format and structure, the METAMODEL of the Cells. That is, the CUSTOMER will play a very significant part in defining the CONTENTS of the Row 2 Cells of the Product Framework IF the Enterprise intends to sell very many of the products to the customer! It is the CUSTOMER that defines the CONTENTS. It is the ENTERPRISE that defines the METAMODEL. The Customer doesn’t really care about the metamodel except that the Enterprise defined it and that they knew what they were doing when they defined it, that is, that they were a competent manufacturer.

When the metamodels of two Frameworks in any one Enterprise are DIFFERENT, it is likely that they have a meta relationship with one another and it does not make any sense to try to “integrate” them. There is nothing in one Framework that is sharable with the other Framework ... the metamodels are different. It is like one Framework is describing apples and the other Framework is describing oranges. If you put them together you DON’T have Cantaloupes. You just have apples and oranges together. They are simply different.

For example, you would not expect to find splines and curves and tangents (the geometry of the product) in the database of locations like Los Angeles, Seattle and St Louis (the geometry of the Enterprise). There is no logical relationship. That is why you don’t find the databases of CAD, CAM and CAE applications integrated with the databases of Payroll, General Ledger and Sales Analysis. There is nothing sharable. In fact, if you did try to integrate those two different kinds of databases (apples and oranges) you would have such complexity and performance and maintenance, etc. problems they would be unmanageable and un-maintainable!

Just like the metamodel of the Product Framework is the Column 1, Row 2 model of the Enterprise Framework, so is the metamodel of the Enterprise Framework the Column 1, Row 2 model of an Enterprise who is engineering and manufacturing THE Enterprise. Now ... who is the Enterprise that is engineering and manufacturing your Enterprise (THE Enterprise) for you?? This may be very disconcerting for you, but I would suggest that BY DEFAULT, the INFORMATION SYSTEMS people in your Enterprise are engineering and manufacturing your Enterprise for you!! Are YOU engineering and manufacturing your Enterprise? (Probably not.) Are you even making yourself available to I/S as their CUSTOMER to define the contents of the Row 2 models of YOUR Enterprise Framework? (Probably not again!) If you, the CUSTOMER of I/S, the OWNER of THE Enterprise is not significantly involved in defining the CONTENTS of the Row 2 models of THE Enterprise Framework, how do you expect the people who are engineering and manufacturing your Enterprise for you to produce an Enterprise that is “aligned” with your requirements or expectations?

Let me take a little emotion out of this discussion. Let’s not call the Framework for the Enterprise that is engineering and manufacturing THE Enterprise, the “I/S Framework.”

Let's call it something neutral like the "Architect's Framework," or the "Enterprise Architect's Framework," or simply the "Enterprise Engineering and Manufacturing Framework." Actually, I don't care what you call it. Anyway you look at it, it is the Framework whose Row 2 models are "meta" relative to THE Enterprise Framework because somebody has to define what those models are going to look like in order to build them. When I say "Enterprise Framework," I mean the "Business Unit Frameworks" and/or the "Cluster Frameworks" and/or the "Enterprise Framework" because all of those Frameworks have the SAME METAMODEL. They are PEER Frameworks.

You will not find things that are sharable between any one enterprise Framework and its Meta-Framework. The "Architect's (meta) Framework" will NOT contain a "library" of components that are sharable (or reusable) in the Business Unit, Cluster or Enterprise Frameworks. The "Architect's (meta) Framework" WILL contain a library of standard FORMATS and STRUCTURES (metamodels) (NOT standard CONTENTS) for each cell of a "peer" Framework within the Enterprise.

Sharable CONTENT components will be found in the "Cluster Frameworks" or the "Enterprise Framework" (the "Sameness Templates") if the engineering for reuse has been accomplished.

The CONTENTS of the Enterprise Frameworks should be being dictated by the CUSTOMER, that is, the OWNER of the Enterprise. The OWNER could defer to the Architect to decide the best way to EXPRESS a coherent model of the Things of the Enterprise (a Semantic Model), or a model of the Business Processes of the Enterprise (a Business Process Model) etc., etc., etc. but (the Owner would) be vitally interested in the CONTENTS of the Semantic Model, the Business Process Model, etc., etc., etc. Similarly, the Owner of an Airplane would likely defer to the Airplane Manufacturer's Engineers (Architects) to decide the best way to express coherent Drawings or Functional Specs of the Airplane, etc., etc. although the Owner will be vitally interested in the CONTENTS of the Drawings, Functional Specs, etc., etc.

Now, here is the problem:

There are some folks that would like to use the "Zachman Framework" to describe the Services of an Enterprise on the basis that they are the equivalent of the Product and therefore different than the Enterprise and therefore should have their own Framework.

Figure 6 sample lists of things you would expect to find in the Row 2 models of a Service Framework.

<u>Column 1</u> “What?”	<u>Column 2</u> “How?”	<u>Column 3</u> “Where?”	<u>Column 4</u> “Who?”	<u>Column 5</u> “When?”	<u>Column 6</u> “Why?”
Criminals	Investigate	Los Angeles HQ	Operations	Legislative Cyc.	Coverage
Police Stations	Warn	Rampart Street	Administration	Case Cyc.	Crime Rate
Officers	Arrest	Beats	Dispatch	Budget Cyc.	Human Rights
Citizens	Book	Cars	Homicide	Acquisition Cyc.	Response Time
Arrests	Testify	County Bldg.	Corrections	Maintenance Cyc.	Cost
Courts	Incarcerate	LA Airport	Citizen Liaison	Trial Cyc.	Flexibility
etc.	etc.	etc.	etc.	etc.	etc.

Figure 6: Sample lists of things you would expect to find in **Row 2 Models** of a **Service Framework**

Figure 7 is a sample list of things you would expect to find in a metamodel of the Row 2 models of a **Service Framework**.

Business Entities
 Business Processes
 Business Locations
 Organizations
 Business Events/Cycles
 Business Objectives
 etc.

Figure 7: Sample list of things you would expect to find in a **Meta Model** of Row 2 Models of a **Service Framework**

The metamodel of the Service Framework (Figure 7) is **THE SAME** as the metamodel of the Enterprise Framework. (See Figure 8.)

Business Entities
 Business Processes
 Business Locations
 Organizations
 Business Events/Cycles
 Business Objectives
 etc.

Figure 4: Sample list of things you would expect to find in a **metamodel** of Row 2 Models of an **Enterprise Framework**

← **SAME** →

Business Entities
 Business Processes
 Business Locations
 Organizations
 Business Events/Cycles
 Business Objectives
 etc.

Figure 7: Sample list of things you would expect to find in a **metamodel** of Row 2 Models of a **Service Framework**

Figure 8: The metamodel of the Service Framework is **THE SAME** as the metamodel of the Enterprise Framework.

Since the Service Framework metamodel is the **SAME** as the Enterprise (Cluster and/or Business Unit) metamodel, they are **PEER Frameworks**. There is abundant advantage to be realized from integration, sharing, normalization, standard interchangeable parts, etc. In fact, it becomes obvious that the Services are simply the Customer’s (e.g. Citizen’s) “view” of the Enterprise. The Service is **THE SAME** as the Enterprise.

The Product is NOT the same as the Enterprise. The Product exists independently from the Enterprise. If, after the Product is manufactured, the Enterprise goes out of business, the Product still exists.

The Service is the same as the Enterprise. The Service only exists as the Enterprise exists. If the Enterprise goes out of business, the Service doesn't exist anymore. The Service is simply the customer's view of the Enterprise.

If you separate the Services out from the Enterprise and treat them as META Frameworks, you are going to DIS-integrate the Services from the Enterprise. That is, you will build "stovepipes" around the Services ... and we are right back where we started from again ... focused on implementation ... overlooking integration ... not doing Architecture ... just building more legacy.

I probably don't have to remind you where this leads ... but, it is only a matter of time again before the Enterprise wakes up and says something like, "Well, Enterprise Architecture ... the Zachman Framework ... that didn't work!! All we have is more of the same ... legacy!!"

In Conclusion

In conclusion, here are my recommendations for you:

1. Don't rename anything in the Framework unless you have a lot of time to sit around and think about Theology and Philosophy and stuff like that. Just take the Framework on faith and use it like it is. You will be a lot safer.
2. Start working on primitive models ... you probably have a lot to learn and every day you put it off is just one more day before the Enterprise starts building up an inventory of primitive components it needs to survive the Information Age.
3. Don't set your expectations about integration (reuse, normalization, sharing, commonalities, etc.) out of line with reality. Integration is the result of engineering. If no primitive models are being engineered, there is no integration ... and therefore, no "Architecture."
4. There are two and only two relationships between Frameworks, Peer Frameworks and Meta Frameworks. Where the metamodels are different, don't try to integrate them. Where the Framework metamodels are the same (and in the same Enterprise), don't disintegrate them.
5. Go back and read footnote number 6 found on page 5.

Anyone who would recommend differently is not necessarily a bad person, just a little mis-guided for the Information Age. They would be demonstrating their Industrial Age heritage where the motivation was for implementation, not integration; point-in-time solutions, not reuse; short term, not long term; applications, not Architecture.

For goodness sakes ... don't let yourself get fatally distracted!! If you're just going to go for implementation, then don't say or even think you are doing Architecture. And, if I were you, neither would I tell management I was doing Architecture (long term) things when I was actually doing systems design (short term, implementation) things!! It's okay to do short term things ... just don't expect long term results!! In that case, you'd better say something like: "we are building more legacy ... we're building it as fast as we can ... but, we'll have to pay the price (scrap and re-work the new legacy, just like we are having to scrap and re-work the old legacy right now) at some point later!!" Otherwise, you are going to put yourself in the awkward position of being a fatality from the latest (and greatest) Silver Bullet! And that Silver Bullet it is going to look a lot like Enterprise Architecture ... or the Zachman Framework!!

Then, it will be another 10 years before you will be allowed to try it again ... if the Enterprise can remain viable for another 10 years in the meantime!!

*John A. Zachman
Zachman International
2222 Foothill Blvd. Suite 337
La Cañada, Ca. 91011
818-244-3763 (Phone and Fax)
johnzachman@compuserve.com*