We Don't Need No Stinkin' PDM!

Larry L. Johnson & Doug Lawson
TEXAS INSTRUMENTS, INC.

Abstract: This paper contains two different perspectives that arrive at the same conclusion. "PDM" is an outmoded term and is actually counterproductive. One perspective is oriented toward simply communicating more concretely, and not forgetting what we are about as IT providers. The other perspective is oriented toward a better way to design and build better solutions, leveraging modern technology. Over time, some people have included such a broad range of applications under the banner of "PDM" that the term has lost whatever meaning it once had. In stating, "We don't need no stinkin' PDM!", we mean the term and the mind set it tends to encourage, as well as the confusion it creates for the users and IT providers. Existing and emerging technologies are examined in the context of designing and providing IT Infrastructure services in the terminology of the business.

1. Introduction

Our title sounds radical... true heresy at a PDM conference attended by the industry's leading vendors and users.

The title is intended to get your attention, and was our attempt at a little humor. You may, or may not, find humor in it. If not, that is OK by us because the topic is serious.

Our critique of the term PDM also applies to PIM (Product Information Management).

We've been working in the "PDM" arena for well over a decade. During that time, we have had difficulty understanding and communicating among ourselves, with our management, and with potential users. This has been true in TI as well as industry. Some of us understand better than others; however, there is usually variation in understanding that becomes problematic for actual implementation. (Is it) difficult material? Yes, it can be. Do we make it more difficult than necessary? We contend that we do. Is there a better way? We contend there is.

2. What's the Problem?

Let's examine the term "PDM" (Product Data Management). A PDM system must, by its name, "manage" "product data". Well, what is product data -- specifically? In the early 1980's we conducted "JADs" in TI where the IT people and users came together to de-

termine the requirements for a "PDM" system. The users represented a broad cross section of the business -- all those people who have a stake in PDM. The question was always raised that if we were to manage something, we need to know pretty concretely what is was. So, off we went to define, or specify, what we meant by product data in terms of the things we were managing. One session produced a list some 60 pages long of "items" that were considered by somebody to be "product data," which was interpreted to mean data about the product. Included were items such as: production yield data, production cost data, brochures, technical manuals, engineering reports, tooling, assembly instructions, quality standards, "who" made the product, "when" it was made, the drawing it was made from, who bought the product, what parts were used to build the product, what they cost, and so on and so on. We then were faced with trying to decide what was NOT product data, in a manufacturing company. We had grown the definition of "Product Data" to the point that it lost its utility for communication and system design. If we were to explore the word "management", we would find a similar phenomenon. Do we mean manage the configuration of the product? Do we mean manage the process of creating the product? Do we mean controlling access to the data? That which is everything cannot be anything in particular.

A lot of this is the result of the maturity of the idea. We are bound to explore many aspects of the "new thing" (e.g., PDM in the '80's) to gain deep understanding. However, we are troubled by the trend of adding more to the PDM plate. Rather than clarifying and reducing, the term seems to be becoming more vague.

One example of this trend is the decomposition of PDM into its key components. One such decomposition includes the following components:

1. Electronic Vault
2. Workflow
3. Configuration Management
4. Design Retrieval/Component Libraries
5. View, Mark-up and Edit (VME)
6. Project Management
7. Electronic Collaboration
8. Tools and Integrationware
9. Scanning and Imaging
10. Document Management

Does this list imply that if you don't have all these components, you don't have PDM? If you're evaluating PDM systems for purchase, does one lose points because it doesn't have project management or electronic collaboration? How much confusion has been induced by Electronic Document Management versus Product Data/Information Management? In our travels, it appears to be a lot. Do we care? We, the authors, care about the confusion but really don't care about the grouping of functions.

We, as IT providers exist to provide the means to improve business performance. This improvement only occurs with the ***implementation*** of capability; be it an "end-

application" or an infrastructure service. One "best practice" is "IT managers to partner in the development of a business strategy". If the best we can do is provide vague terms, we won't be very effective. Many business managers think of IT as "mumbo-jumbo", they know they need some of it, but don't understand it.

What's the problem? We might refer to it as the "Abstraction Gap". Figure 1 illustrates this gap. We conveniently divide the universe into a Business World and an IT World. Few would argue that they are "different worlds" (i.e., "Those IT folks live in a different world".) The two worlds come together, however, in the area known as "where the rubber meets the road". At this interface, things need to be specific and concrete, because this is normally where "business" applications exist. In the "business" application area of the IT world we have several well-understood applications that for the most part are presented in terms familiar to the business user. The names conjure up meaningful images to the business community. The IT folks and business folks know what each other means (for the most part) when the terms "general ledger", "accounts payable", "shop floor" control, etc. are used. In our experience, this isn't the case when the term "PDM" is used. When "PDM" is uttered, glazed eyes follow; heads bounce off the conference table, as the IT folks ask the imponderable questions surrounding PDM. The business managers ask, "why should I care". Shrewd IT folks answer by saying, because your competitor, company X, is doing this and is ahead of you. This may be fine for inducing paranoia to obtain funding, but it is not fine for making significant improvements in business execution.

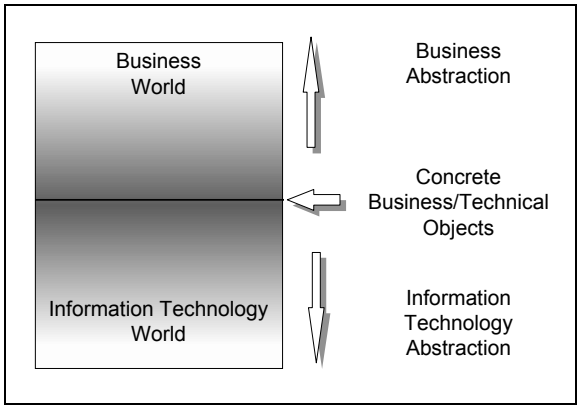

Figure 1. Abstraction Gap: The Collision of Two Worlds.

As one moves away from the boundary of the two worlds, terms become more abstract. This may be useful to those living in particular regions of those worlds but it becomes increasing difficult to communicate in meaningful ways between the two. The IT folks living in the abstract regions of the IT world, say CORBA or "LU6.2", laugh at the abstract vision and corporate mission statement developed by top level business management. The business folks just avoid the "techno-weenies", including IT managers incapable of communicating in business terms. Even within one of the worlds the abstraction gap creates barriers in communication which in turn create barriers to effective business execution. We are reminded of the Dilbert cartoon where the engineers say of the competition, "We're in trouble now, they're gonna use synergy!" Seriously, though, in the abstract a business is a system for generating money (e.g., profits). Out of the context of the particular business this abstraction isn't very meaningful. IT abstractions out of context aren't meaningful either. Later in this paper, we will explore how to bridge the gap in the IT world and provide aid to the business users.

Perhaps the problem here is that PDM is not an "end-application". The tasks being performed at the boundary are task oriented. PDM, in the whole, is a collection of applications as described previously with the "key components". If we understand the components and can, and do, implement them individually, or in various combinations, what value does the term PDM have?

Indeed, what problems are created by viewing this beast as an end-application?

As currently delivered by the suppliers, PDM systems, as such, do not match the processes and tasks that they must support. Many times, they are built to support the entire business process in one application... a single application to support the needs of:

- system, electronic, mechanical, and software engineers,
- product development
- configuration management,
- factory floor operations,
- enterprise information management
- [fill in the blank and repeat ad nauseum].

As in most "one size fits none" situations, each business task is equally ill-served. PDM users need to navigate Byzantine menu structures to find and string together the functions they need to execute their tasks. They are presented with arcane terminology and must translate the function label to the job they are trying to perform; unless, of course, someone customizes the system at additional cost. When this customization is performed it is usually for a specific purpose - in the context of the business, a specific application. Is it still "PDM"?

2.1 Lack of progress in PDM industry

While there are success stories in this area, there are also articles that lament the lack of any large scale implementations. Some "rate" the systems, and suppliers, by their ability to execute, meaning what is the extent of actual implementations. One recent matrix the implementation quadrant was extremely sparse. While it is a "hot" area with increasing activity and interest, there is a lot of money on the table for business users and PDM suppliers alike.

We have been at this collectively for a long time. The question we should be asking is, "how can we accelerate implementation of this technology?" One suggestion is to stop using the term PDM and focus on the business effect we are seeking (in more concrete terms than let's make a bunch of money).

PDM supplier revenues would be enhanced (greatly) if the suppliers presented their wares in the context of the business problem they were solving, better still if it was translated to dollars. The revenues would soar if the "systems" were targeted at specific solutions. Sometimes they are, but still fall short because they don't consider the overall envi-

ronment in which they must operate. This is the trick. How do you solve a specific problem while providing more potential for more specific applications that do not collide with one another.

The so-called "engine" and/or toolkit providers must provide solutions to real problems, without customization, and end-application providers must provide solutions that gracefully operate within the customers IT environment.

The suppliers that crack this code will do their shareholders proud.

2.2 Lack of Progress by PDM Users

There are success stories, but not as many as there could be. The successes are generally characterized by a clear target expressed in terms of the business. Perhaps there was a specific "scenario" for some segment of the overall product development process. Or, perhaps there was some other target that wasn't expressed as a process scenario but as some business effect. We don't remember hearing or reading of a success story of implementing "PDM" per se. It is always couched in terms of the effect. Without that, how can the success be described? "We implemented PDM and it was good, saves us a bunch of time and money". How? We <u>really</u> mean we don't need no stinkin' PDM.

We suggest that if the progress of PDM in your company is moving forward at glacial speeds, perhaps the terminology is to blame. We contend there is no such thing as a PDM system. PDM is not a task that should appear in any business process. None of us are in business to "do PDM"... at best, it is a support function, i.e., a function of the infrastructure. As we will discuss later, the term PDM isn't needed when describing the infrastructure either.

The words we use are a very important consideration in mobilizing people to achieve a goal. The more people involved the more important it becomes. It is imperative that the goal is expressed in meaningful terms to those who must perform the implementation.

At this point the reader may ask, 'well what to you call <u>*it*</u> if not PDM?' If you're asking this we have done a poor job communicating (entirely possible) or you are filtering the message with the "PDM" paradigm. The answer is - there is no "PDM". There are specific task oriented applications and there are specific services (referred to herein as "IT infrastructure services") used by these applications and one another. As we will see, these services can also be expressed in terms of the "business tasks."

Focus on the business problem/opportunity in terms of the business and the effect desired. State clearly and directly what is desired.

3. A Change in Strategy

Within the Systems Group at Texas Instruments we have had a "PDM" Strategy for several years. No more! The "Strategy" is now expressed in terms of the business applications being addressed. We, the IT providers, know we must implement these applications in ways that "Blend and Build". They must blend into the environment while building a base that allows for rapid implementation of new applications. We still have our abstract terms such as ORB's, TCP, etc.; but these are removed from the end-users and management. We know if we cannot relate an abstract service to the business effect, we probably don't need it (i.e., understanding the "food chain" from application to service). We try to focus on where the business needs exist and what the relative priorities are in our environment (where the opportunities always exceed the funds available).

The following story is true and is presented to illustrate the power of a clear business objective.

We have a system in Texas Instrument's Systems Group (SG) called Electronic Drawing Distribution (EDD) which became fully operational this year. Advocates of this capability had tried unsuccessfully to obtain funding since the late 1980's. Many "intangible" benefits were presented to management. It would save time and money, but the savings were often calculated based on some percentage of time people would save in getting a copy of a drawing (usually from a microfilm card or the copy center). Saving an hour a week just wasn't compelling, particularly so when compared to the million dollars+ of "real" money needed to develop and implement the system.

The advocates were annually disappointed by the lack of management vision. Until one day someone said, "Hey, what if we simply looked at replacing the microfilm libraries with an electronic version, what would that be worth to the company? What are we currently spending on creating the microfilm, making copies, distributing to the sites, creating paper copies, and keeping the card files in order? If we eliminated this practice, how much MONEY would we not spend (i.e., savings)? Would it (alone) be large enough to justify the project?". An analyst was assigned to gather up the costs of this activity. If the cost wasn't in somebody's current budget, it didn't count. The amount spent was in several different budgets and amounted to some real money.

The potential savings were large enough to work the cost side of the analysis. What would replacing the microfilm media with an electronic form cost? In the early days the costs included several "infrastructure" items, that over time, came into being for other reasons. Network (LAN/WAN) access had become pervasive. Desktop computers had proliferated. A CAD Drawing Repository had been implemented (to replace the paper "Drawing Control" process). So, as the years went by, the net cost of implementing continued to decrease, and by 1994 the project didn't need to fund infrastructure elements to get up and running. The cost side of the equation included view and print stations (capital), service bureau charges for creating the initial critical mass of scanned images, scanners for "on-demand" scanning, and development of the operational procedures for get-

ting drawings into this system. It turned out the needed IT elements already existed; we simply needed to put them into operation, so there were no appreciable IT development costs. This is the "build" portion of the "Blend and Build" approach, i.e., because the infrastructure and other elements already existed, the time and cost to implement this specific application collapsed. The risk was also very low because we were using things that already existed in the environment.

The project was justified on one simple goal: eliminate the microfilm libraries. This was something management could understand!

During the development of the approach and system design, the business case and goal was continually referenced as a means to perform trade-offs and contain the system requirements. It is not unusual for system requirements to grow during detailed design, and this is especially true if the objective is not clear and understandable. This was the case here. Along the way, people tried to rename the project. "EDD means Electronic DOCUMENT distribution!". "EDD means Electronic DATABASE distribution!" "This system will enable reuse!" "We'll ship databases to Manufacturing!" NO! We are simply eliminating the microfilm libraries. That is where the savings are coming from and if the 'feature' isn't directly aimed at that, we aren't doing it! All these extra "features" or "requirements" increase the cost and stretch out the implementation. Consequently, they are actually barriers to realizing the benefits. We would design the system so we wouldn't prevent adding these at a later date, but they weren't part of this project. Those other features will need to be justified on their own merit.

In retrospect, we believe we would still be designing the system without the clear business case and goal, even if by some miracle management funded it. The parallel to the broader area of PDM seems obvious to us. There hasn't been a clear business target that had meaning in the context of the business. As a result, requirements and features have grown and grown to the point that some people include project management as a feature, consequently, the implementation times have stretched and stretched.

What about the other benefits? Well, we're getting them. The number of actual prints requested is starting to decline. People with desktop machines can access the drawings from their desks and are saving time. The cycle time from release to availability has been significantly reduced. These things provide benefit but weren't used in the justification. They weren't needed. You only need one good reason to do something; the rest of the reasons are side benefits. The other, truly intangible benefit is that we have introduced a new medium to the business users and they are getting comfortable with it. This will make our next steps (e.g., Electronic Document/Database Distribution, On-line Approval and Release, Reuse, etc.) easier to take by the business users, again each step builds toward another.

It is worth pointing out that at no time during this story or the actual project was the term PDM used. The fact that we use a PDM repository (e.g., electronic vault) is not even known by many people, and nobody cares. Even people in the IT organization not inti-

mately familiar with system do not equate it with "PDM". If we express what we are trying to accomplish in terms of the net effect, the term PDM isn't necessary.

At the risk of running on, the business target was *not* "to build a system", i.e., EDD was not the target. The business target was to eliminate the costs associated generation and distribution of Microfilm. Information Technology provided the means to obtain the savings.

If you don't know where you are going, you probably won't get there. If you just want to look at the scenery it doesn't much matter if you get anywhere in particular. This seems to be the approach taken by some PDM practitioners. The technology provides the beautiful scenery; it is an end in itself ("I love this stuff"). This might be fine for a science project, but it doesn't do much for a business.

The next chapter decries how to move into the services "layer" using business terms where the terminology has been abstract. It also illustrates the "Blend and Build" approach. We believe companies that crack this code will be able to keep pace with the accelerating change we all experience while those that can't will be overcome by the glaciers.

4. Technical Aspects of the Business Oriented Approach

4.1 Spanning Layers of Abstraction

In applying technology to solve business problems, it is important to remember that both the business problem and the technology operate at various levels of abstraction. (Fig. 1).

In the business process we have concrete objects such as customers, sales orders, parts, and machines. In managing the business, more abstract objects are needed such vision, objectives, process and other such concepts.

In information technology we often operate at an abstract level of objects such as files, records, databases, etc. Much of our programming and development effort is spent spanning the chasm of abstraction between our information systems and concrete business concepts (Fig. 2). Information models are often used to map business items to our data abstractions. At best these models are re-used, at worst they are inconsistent and always out of date. When

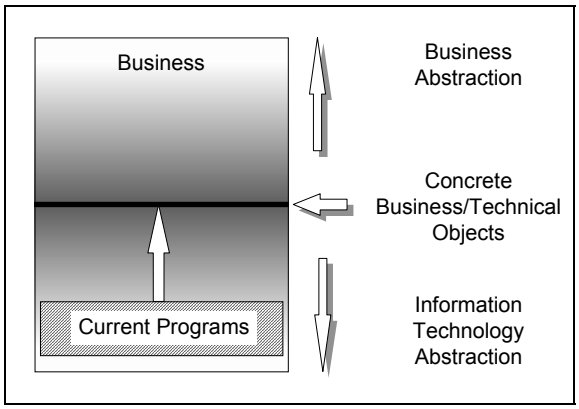

Figure 2. Gulf of Abstraction. Current Software Applications are customarily written at an abstract level, attempting to span the chasm of abstraction to concrete business concepts.

the models are up-to-date and reusable, any changes in the model require applications to be changed. Additionally, programming at this level is labor intensive. Consequently, desired changes in the business process can be made intractable due to the cost of modifying the support programs, if the changes can be made at all!

What we would like is to have business support applications written in terms of the business items themselves, lowering the amount of code necessary. The applications should ideally support a business task directly, in the terms of the task being served. Users should not be required to "translate" abstract concepts into terms natural to the task.

With the advent of object technology we have the tools we need to create information analogues to our business objects. We have the capability to create an almost 1:1 co-existence of concrete business objects with "concrete" information objects. In doing so, the analogue is complete. In terms of business process and information process, we are able to provide parallel realities. The consequence of this is that programming to support the business process no longer occurs at the low-levels previously required. Programming is done directly in terms of the objects themselves (Fig. 3).

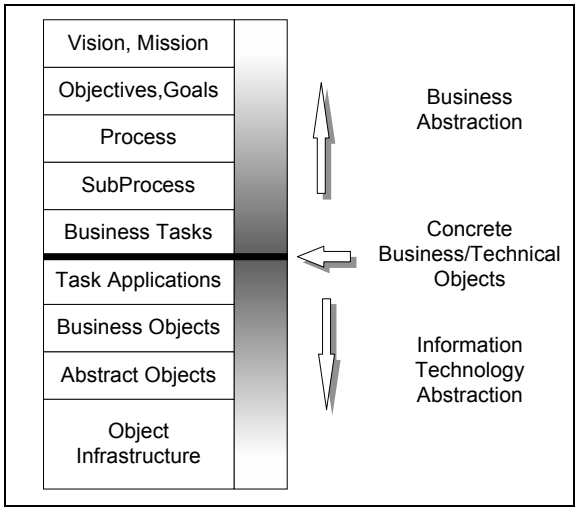

Figure 3. Task Applications Directly Support Business Tasks. Higher levels of abstraction, such as Object Infrastructure, are invisible from the point of view of the user.

One of the distinctive features of object programming is that the object consists of methods, or behavior, as well as data. This means that our applications programs do not need to contain all of the code, much of it is contained in the object itself where it can be employed by many applications.

Object programming by itself won't accomplish our objectives. We will describe an architecture for agile software that is based on current and emerging standards and capabilities.

4.2 What is Architecture?

Before we proceed, we need to provide some simple definitions of Architecture:

- The documentation of inter-relationships of "parts" and their relationship to the whole.
- Architecture can have many "aspects" that are represented differently, but must be brought together.

Let's take for example, the architecture of a house... something all of us can relate to. Figure 4 shows various aspects of the architecture, and the inter-relationships of those aspects. For example, the Plumbing and Electrical Diagrams are usually documented in the context of the framing diagrams.

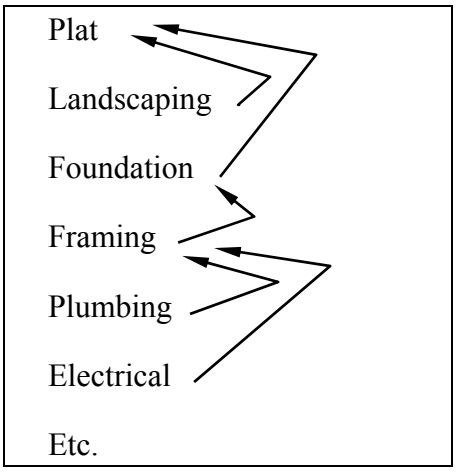

Figure 4. Aspects of the Architecture of a House. The various aspects of the architecture are related to one another.

Why do we bother with architecture? Imagine a house wherein....

- Every frame corner is built differently
- Plumbing is non-standard parts
- Electrical Wiring is ad-hoc
- Landscaped with Michigan plants for a Texas house

What would be the impact on Cost to Build, Cost to Own, Usability, Maintainability? Imagine trying to remodel, or build an addition!

In software, we have a similar situation. Imagine a system wherein there are...

- Inconsistent Interfaces (No standard corners, framing.)
- Monolithic Applications (Walls cannot be moved.)
- Multiple Server Protocols (No standard plumbing.)
- No Information Exchange Standards (Rooms with no doors.)

What would be the impact on Cost to Build, Cost to Own, Usability, Maintainability? Imagine trying to remodel, or build an addition!

4.3 Goals of the Architecture

We are attempting to support enterprises with the following characteristics:

- *Dynamic Business Process* - Business Process Re-engineering is a continuous undertaking. Supporting software must be able to adapt to the evolving processes.

- *Virtual Enterprises* - The ability to team with other enterprises has become essential in the current economic environment. The business process of the virtual enterprise must be able to execute seamlessly across organizational and corporate boundaries.

Additionally, we want the architecture to support the provision of software that has the following characteristics.

- *Business Process Oriented*. We want the software to directly support the execution of business tasks. Each worker should be able to deal in terms that are natural and customary to the task that is being executed. He should not have to deal with special terminology imposed by some software program.

- *Facilitates Process Engineering & Execution.* Processes can't be built around software capability, rather the software must support the process. The software must evolve in step with the evolution of the business process.

- *Cost Effective*. The software must be affordable to acquire. It must also be affordable to own and use, including aspects of training, maintenance, and ease-of-use.

- *Integrateable*. The software must be able to offer consistent behavior and views across multiple applications. It must not require redundant data entry or other operations. It must accommodate legacy systems.

- *Best of Class*. Applications should use best-of-class capabilities. This means that there must be a capability to change from one software product to another without undue burden of cost.

5. Problems in Software Provisioning

It is often easier to examine things from the point of view of specific recurring problems that occur in many of our environments. Let's examine some of the problems that the architecture must address. As we examine the capabilities of the architecture, we will assess them through a scorecard of the problems it helps address. The following sections rephrase our goals and objectives in terms of barriers or problems that prevent us from achieving them.

5.1 Distributed Data

Information for a given part is spread among many systems. A part definition can involve mechanical, electrical, and software definitions. Additional descriptions are found in functional specifications. As it stands today, each aspect of the part definition is found in a separate system specialized for that aspect of the part (Electrical CAD, Mechanical CAD, CASE tools, etc.).

5.2 Redundant Data

Product Structure is an Excellent Example. Almost any tool that is involved in product definition or description requires Product Structure and contains it for its own use. We want to manipulate product structure in the context of the business task, and yet, maintain consistency across business tasks.

5.3 Monolithic Applications

We have spoken of the problems of "One size fits None" applications. PDM applications are particularly good examples of this problem. Different views of product data are required by Mechanical Engineers, Electrical Engineers, Configuration Managers, etc. Every user is required to navigate the Monolith through a maze of menu picks. Functions are not offered in the context of the business task.

Do we really want to divert our engineers to be trained as experts in PDM systems?

5.4 Closed Applications

When functionality is not accessible from other applications, the application holds the business process hostage. It severely limits the degree to which you can re-arrange your tasks, since the software is built around some pre-existing idea of a business process.

5.5 Application Development Cycle Time

In order to provide evolving software to match the needs of the evolving business environment, development cycle time must be much lower than it currently is. In the present situation:

- Production Quality Development or Adaptation requires too much time.
- Rapid Prototyping does not produce production quality applications.
- Expansion of Functionality usually means redoing Applications
- Modification of User Interface usually means re-writing Algorithms

5.6 Intractable Integration

Much of what is needed already exists and is done on some legacy system. Making applications that bring together functions on different systems is costly. When functions move to different systems or applications, the integration needs to be redone.

5.7 Vendor Entrapment

Even with an "open" API, if the new vendor's API doesn't match the old one, then all using applications and integrations need to be redone. This creates a severe impediment to selecting "Best-in-Class" capabilities. Vendor changes are costly.

6. An Infrastructure Services Architecture for Agile Software

We will focus on two of three aspects of technical architecture taken from work done in CALS and the Rapid Response Manufacturing Consortium. The three aspects are:

1. *Standard Semantics*. We take the Standard for Exchange of Product Data (STEP, ISO 10303) as the vocabulary of product definition. We will not spend further time on this, since it is covered in other papers in this symposium.

2. *Three-tier Client/Server Architecture*. We will focus on and the Common Object Request Broker Architecture (CORBA) developed by the Object Management Group (OMG). Much of our discussion will focus on this aspect of the architecture.

3. *Portable Graphical User Interfaces*. We will address the synergy of combining PGUI technology with three-tier client server architecture.

6.1 Portable Graphical User Interfaces (PGUI's)

Portable Graphical User Interfaces provide pre-programmed, tailorable "widgets" (dialog boxes, buttons, tables, ...) complete with behavior "triggers" (click, double click, ...). PGUI's provide a higher level interface (less programming) than Motif or MS Windows. Like CORBA, PGUI's offer platform independence, using the underlying windowing services of the platform.

PGUI technology is very mature. The market place offers more than a dozen commercial offerings (e.g., Neuron Data, Web Browsers, ...) as well as "shareware" (Tcl/Tk, more Web Browsers, ...)

6.2 Three-tier Client Server Architecture (CORBA)

Based on object technology, the idea of three-tier Client Server Architecture is to place a formal object request broker between the application and the object service (or method). The idea is to remove the real work from the application. The application's responsibility is to manage the interface with the user. An important effect of this is to allow all fundamental functionality to be shared among multiple applications, implementing consistency through the use of common services. The object request broker isolates the application from the network protocols, server protocols, platform characteristics, and language characteristics. (See Figure 6) This removes a great deal of burden from the application programmer, allowing focus on the business objects. (See Figure 5)

In addition to providing standardization of the brokering environment (CORBA), the Object Management Group provides standardization of services. Two basic categories of services are defined:

1. *CORBAservices* - These services provide abstract object services required by many objects regardless of their semantics, such as naming, time, security, persistence, etc.

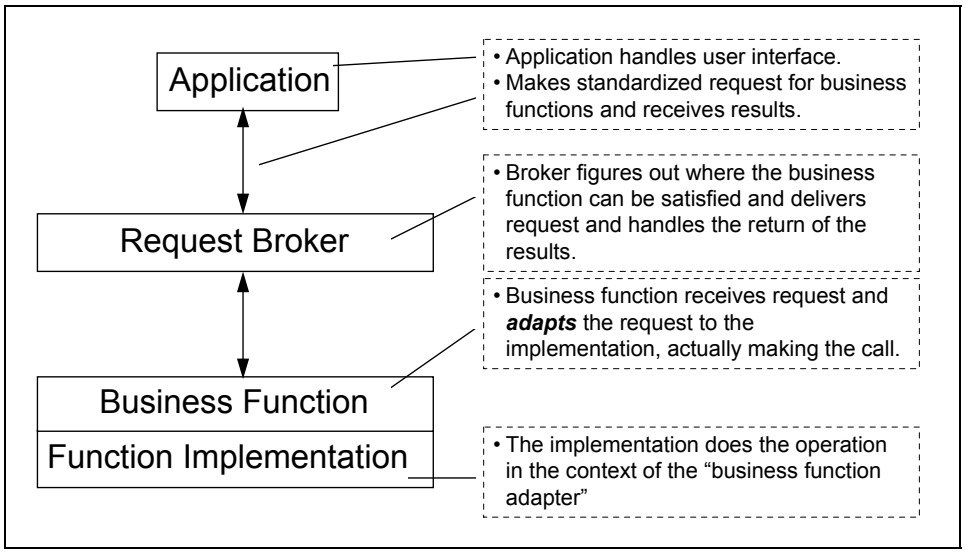

Figure 6. The Basic Idea of Three-Tier Client/Server Architecture.

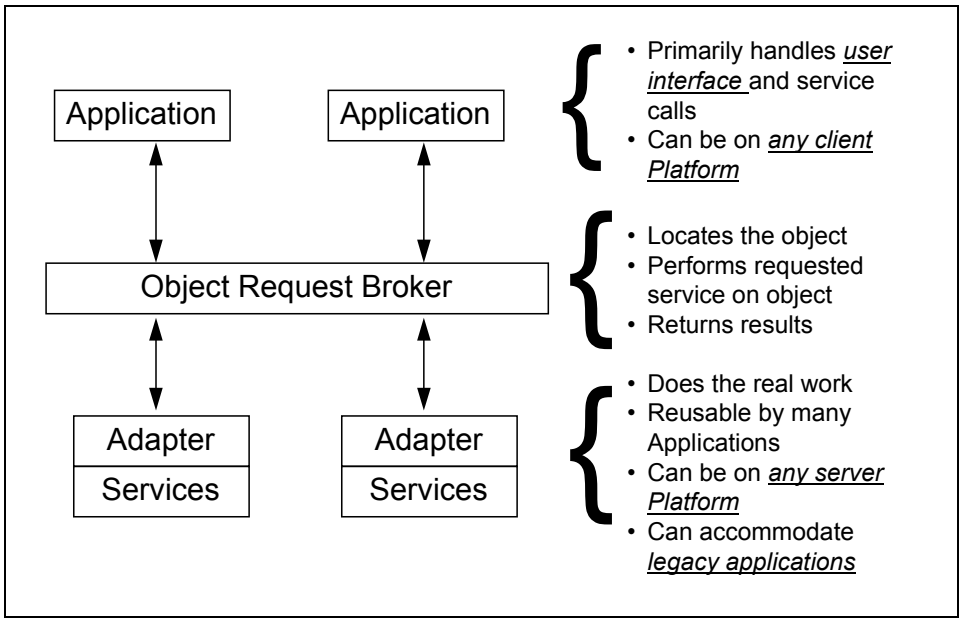

Figure 5. Benefits of Brokered Architecture.

2. *Domain Specific Services* - These are services that pertain to classes of objects that have specific, business-oriented semantics. Domains have been defined such as "Business Objects" and "Manufacturing". It is interesting to note that the Manufacturing Domain Task Force is in the process of standardizing PDM services. But remember, "we don't need no stinkin' PDM". Of interest is their

specific enumeration of eight enablers to be accommodated, which are specific (product structure, effectivity, etc.).

The domain specific services are of particular interest to us. Augmenting the semantics provided by STEP, these service definitions will provide a "common speak" of functionality that addresses the business in terms of the business. Exactly what we are looking for.

7. Benefits of Architectural Features

We have enumerated a set of problems that the architecture needs to address. Having delineated the architecture, we will now examine the features and characteristics of it and then "score" each as to what specific problems it addresses.

☐ Example Score Card ☑

- ☐ Distributed Data----------A Problem not Addressed
- ☑ Redundant Data ----------A Problem Addressed by the Feature
- ☑ ***Monolithic Applications*** -------A Problem Heavily Serviced by the Feature
- ☐ Closed Applications
- ☐ Application Development Cycle Time
- ☐ Intractable Integration
- ☐ Vendor Entrapment

7.1 Service Partitioning

In the parlance of brokered services, a "service" is actually a suite of closely related services. For example, a Part Structure service would contain all the functionality needed to manipulate part structure.

Services, the code that does the real work, are resident in the server and are independent from the applications that use them. The services may use one another (e.g., deletion of Part must result in deletion of its documents and participation in a Structure)

Though services use one another, they are distinct from one another (e.g., Part Structure services are not tangled with document services).

Services can be replaced independently.

Additionally, object adapters can be written over legacy applications to allow them to carry out the services. This allows multiple applications to be synchronized through common services of legacy software, not requiring wholesale reprogramming of existing systems.

☐ Score Card for Service Partitioning ☑

- ☐ Distributed Data
- ☐ Redundant Data
- ☑ ***Monolithic Applications***
- ☑ ***Closed Applications***
- ☑ Application Development Cycle Time
- ☑ ***Intractable Integration***
- ☑ Vendor Entrapment

7.2 Application/Service Independence

Service implementations are independent of applications. We can re-implement a business function without changing or re-deploying applications that use it. This offers a number of benefits; among them:

- Provides intrinsic mitigation of risk in the selection of hosting facility (e.g., is the service to be implemented in a Metaphase? Sherpa? Oracle? Something Else? ... If we need to change we can do it with reasonable cost)

- We also achieve vendor independence in the sense that the service can be replaced with an implementation from a different vendor, transparent to the application.

Let's look at the example of Product Structure Attribution Services. These are services that allow a user to add an attribute (from a catalog, or ad-hoc) to a part in a structure or to the structure relationship itself. (Some call these "loose attributes".) In this scheme of things the Attribute is an object in its own right and has its own attributes; among them are value, who set the value, when the value was set. It supports a method by which the value can be retrieved. See Figure 7.

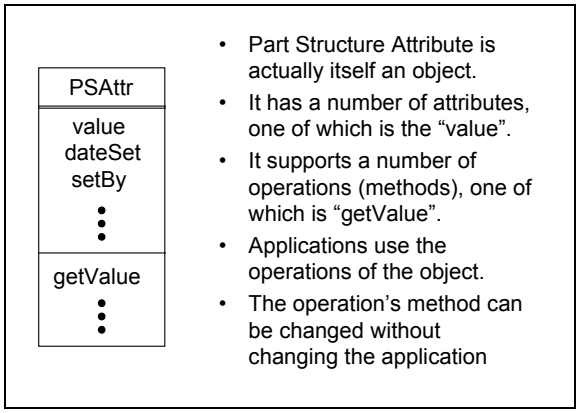

Figure 7. The Basic Part Structure Attribute.

Applications use the operations of the object (such as getValue).

The operation's method can be changed without changing the application (Fig. 8)

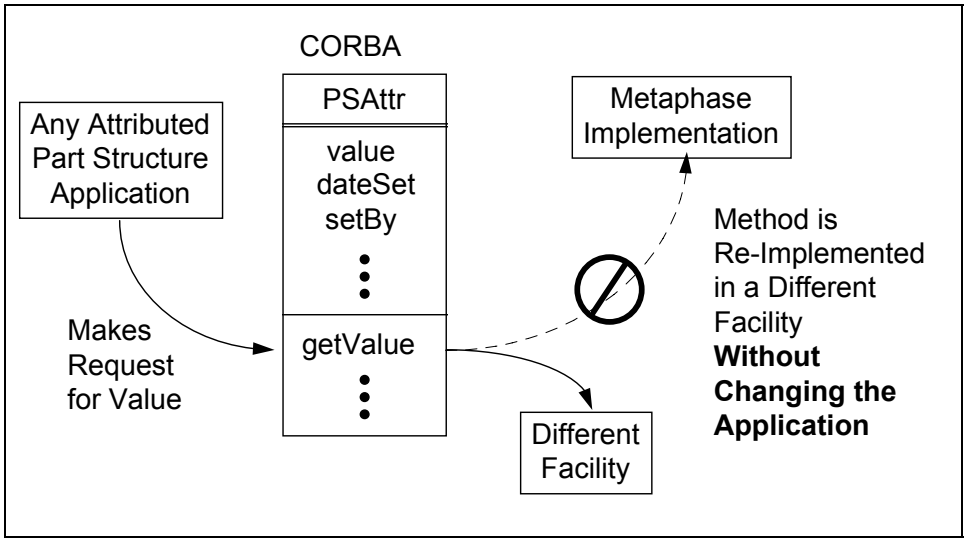

Figure 8. Binding a Service to an Implementation. The application talks to CORBA and lets it dispatch the service to its implementation, which in this example is Metaphase. If it is decided to implement the service in a Different Facility, it can be done without affecting the applications that use it.

☐ Score Card of Application/Service Independence ☑

- ☐ Distributed Data
- ☐ Redundant Data
- ☑ Monolithic Applications
- ☐ Closed Applications
- ☑ ***Application Development Cycle Time***
- ☐ Intractable Integration
- ☑ ***Vendor Entrapment***

7.3 Portable Graphical User Interfaces (PGUI)

Portable Graphical User Interfaces provide pre-programmed, tailorable "widgets" (dialog boxes, buttons, tables, ...) complete with behavior "triggers" (click, double click, ...). PGUI's provide a higher level interface (less programming) than Motif or MS Windows. Like CORBA, PGUI's offer platform independence, using the underlying windowing services of the platform. Portable GUI + Portable CORBA = Portable Application

There is a powerful synergy in combining a PGUI with CORBA services. The application manages user interfaces via the PGUI while the real work is performed via the server environment. Applications are extremely easy to write allowing task specific interfaces to be written easily. We do not need "one size fits none" monolithic applications. We can construct many "mini-apps" that fit the perspective and the function of the user.

These applications can be written in a rapid prototyping environment. The applications can be constructed quickly using pre-programmed GUI widgets, and pre-programmed business services. If the services are written using rigorous methodology, then the rapidly prototyped applications are production quality.

☐ Score Card for Portable Graphical User Interface + Request Broker ☑

- ☐ Distributed Data
- ☐ Redundant Data
- ☑ Monolithic Applications
- ☑ Closed Applications
- ☑ ***Application Development Cycle Time***
- ☑ Intractable Integration
- ☐ Vendor Entrapment

7.4 Extensible Object Model

Functionality can be expanded by extending the object model. Functionality can be extended without breaking existing applications.

Let's extend our example. Figure 9 shows an extension of the Product Structure Attribute, PSAttr, to accommodate externally acquired attributes, PSExtAttr, meaning that the value will be obtained from another system rather than be manually set through an editing application. The external attribute adds characteristics such as the external systems, the update frequency, and the maximum interval allowed between updates (in case we can't get a hold of the system). It adds the method autoUpdate, which is used to get the value. Note that the PSExtAttr IS A PSAttr, which means it inherits the attributes and methods of PSAttr as well (shown in Figure 10)

The applications written against the PSAttr object still work. Through object technology we have been able to expand our functionality without re-writing our previously deployed applications.

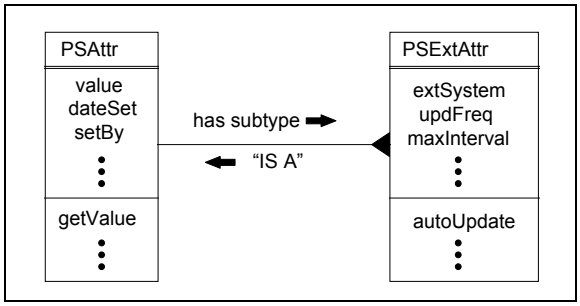

Figure 9. Extending the Object Model. A subtype is declared for the Part Structure Attribute called Part Structure External Attribute.

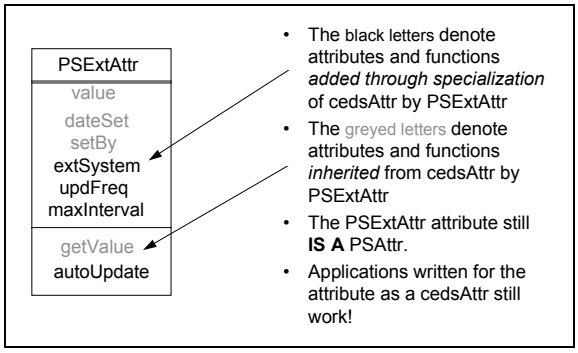

Figure 10. The Effect of Inheritance in Extending the Model.

☐ Score Card for Extensible Object Model ☑

- ☐ Distributed Data
- ☐ Redundant Data
- ☐ Monolithic Applications
- ☐ Closed Applications
- ☑ ***Application Development Cycle Time***
- ☐ Intractable Integration
- ☑ Vendor Entrapment

7.5 Conceptual View of Objects

The object model of the objects being manipulated is expressed in CORBA's Interface Description Language (IDL). Essentially, this makes CORBA aware of the conceptual schema. All service requests are made in the context of this IDL conceptual model. Consequently distributed objects and redundant data stores can be made to appear as single entities to the application. The application does not need to worry about what characteristics of an object are where. An object's characteristics can be distributed among applications and repositories however it makes sense. The distribution of characteristics can be changed transparently to the application. (See Figure 11)

The ability to synchronize redundant data does not come "free" with three tier architecture, but it provides a tool that can be used to address the issue. (See Figure 12)

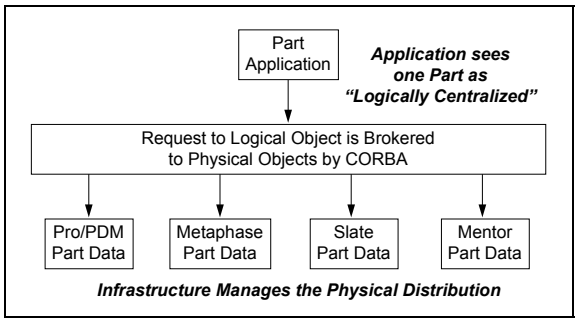

Figure 11. An Application Sees a Single Conceptual Part. The location of the data is transparent.

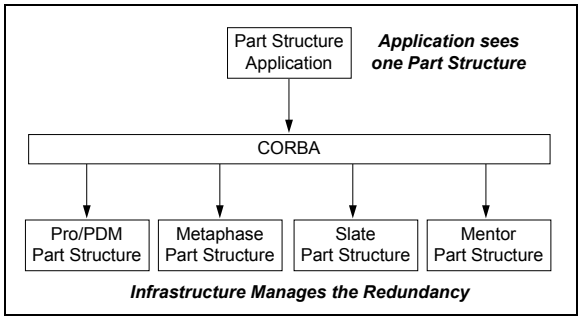

Figure 12. An Application Sees a Single Conceptual Part Structure. The redundant copies are synchronized by the infrastructure.

☐ Score Card for Conceptual View of Objects ☑

- ☑ ***Distributed Data***
- ☑ ***Redundant Data***
- ☐ Monolithic Applications
- ☐ Closed Applications
- ☑ Application Development Cycle Time
- ☑ ***Intractable Integration***
- ☑ Vendor Entrapment

8. References

Johnson, Larry, "The Operational Context and Concepts of Reference Architecture within the Rapid Response Manufacturing Consortium", Rapid Response Manufacturing Consortium Internal Document, 1994.

Johnson, Larry, "Infrastructure Services Architecture, A Reference Architecture for RRM Engineering Environments" Rapid Response Manufacturing Consortium Internal Document, 1994.

Johnson, Judson, et al., "Infrastructure Services Architecture of the Profile for Enterprise Integration", CALS Industry Working Group, 1994.

Judson, Johnson, et al., "Profile for Enterprise Integration" CALS Industry Working Group, 1994.