An Architecture Framework: From Business Strategies to Implementation

William F. Hertha, Jim E. Bennett, Frank J. Post, Ian M. Page

Architecture Services,
Canadian Imperial Bank of Commerce
901 King St. W. 7th Floor, Commerce Court Postal Station 'A'
Toronto, Ontario, Canada
E-mail: bhertha@mtnlake.com Fax: 1 416 980 3099

ABSTRACT. *Business systems architects and their clients increasingly suffer from information overload. To help businesses to partition and relate the kinds of architecture information they must build and share, we propose a framework consisting of three related models, each incorporating four tiers of subject matter connected by use cases.*

KEY WORDS: *Architecture Framework Strategy Process Component*

1.0 Introduction

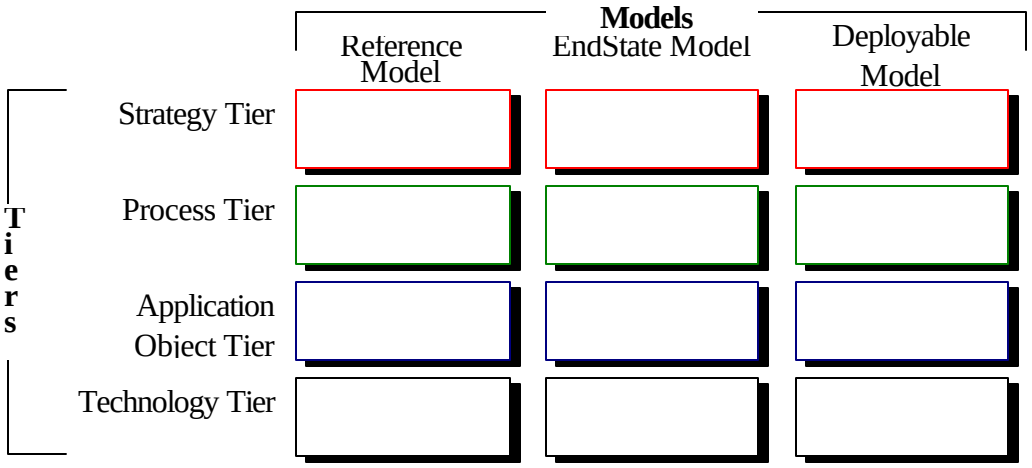


Figure 1 - Architecture Framework

Preparing an architecture for large business systems is never an easy task, and in today's business climate it is made more difficult than ever. The expectations are higher, and the alternatives are wider. There is a great amount and variety of information to absorb.

As Figure 1 illustrates, this paper will explore a four-tier three-model framework which we feel can help to organise and communicate this information.

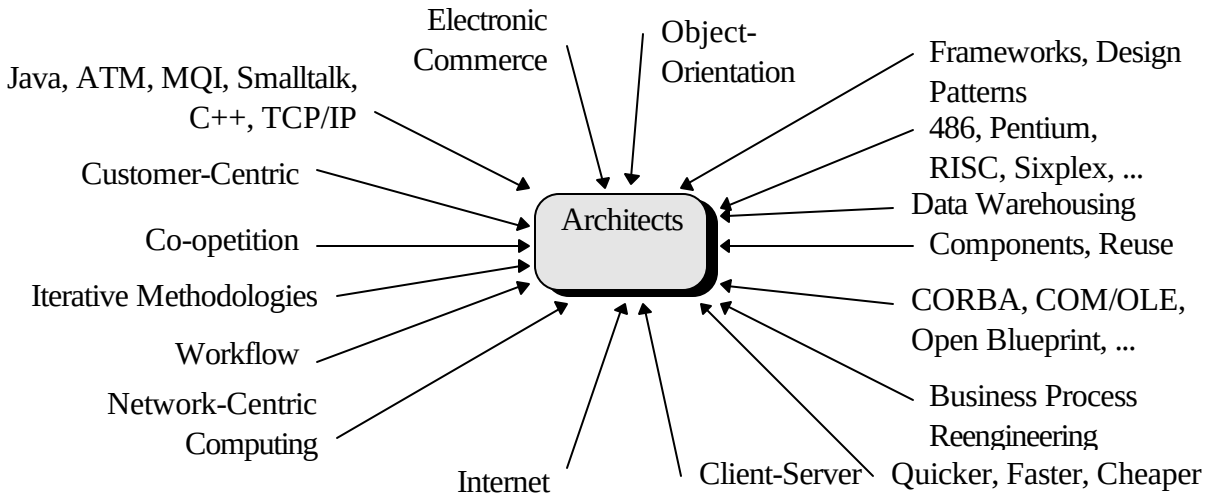


Figure 2 - Challenges for Architects

As Figure 2 shows, the problem of preparing an architecture has become large and complex. Even its scope is not universally agreed. An architecture cannot be created by a small group of people isolated from the business that needs the technology nor from the people that have to apply it to solve business problems. It requires the involvement of many people, from several disciplines.

Business clients need to be involved so that technology is applied to the areas where it can provide the most value. At the other end of the spectrum, technologists need to be involved because they understand what works, and what does not.

Mark Fox [Fox95] says "*Many artifacts, including those produced by the project's participating corporations, are so complex that designs require the efforts of many engineers. This means that the artifact must be functionally and/or physically decomposed, and the pieces divided amongst groups of engineers. The major issue in 'Design-in-the-Large' is how to achieve high levels of collaboration among engineers. That is, how each engineer's design task can be managed so that it integrates well with the results of others.*"

The experience of many business systems architects suggests that we need ...

- .. A way to accumulate, organise and communicate information. A project of this magnitude and duration will result in a vast amount of information and knowledge being generated. Participants will come and go. The information has to grow and evolve even though the contributors may change.

- .. A common vision and language. If we are to muster an army of individuals to realise an objective, they all have to be clear what that objective is, the more so because of the different backgrounds, different skill sets, and different understandings of terms and concepts.

- .. The ability to address sufficient scope. Many of the architectures currently proposed focus only on a subset of the domain, mostly the technology infrastructure -- the hardware, software, tools -- and only to a lesser degree are applications or processes considered. Rarely do we see any discussion on the broader systems problem it relates to the business processes that must employ the technology and the business strategies that caused those business processes to come into being.

- .. To partition the problem domain, especially if we are enlarging the scope.

- .. To assign clear responsibility for each task resulting from our partitioning.

- .. To match the right skills to each task resulting from our partitioning. If we are to expand our view to integrate technical solutions into the business solutions, we need to distinguish these tasks, and apply business skills, business process engineering skills, analytical skills and pure technology skills, where appropriate. Applying the wrong skills to a task can lead to disastrous results.

- .. To relate the solutions to the business need. The solutions that are prepared must be shown to be tied to the problem they are intended to resolve. Too frequently we hear the phrase “a solution looking for a problem.”

- .. To be selective about what we make flexible, as building flexible solutions costs time and money. Too often we blindly overpay for flexibility in the hope that the solution will later b tied to some unknown or unclearly specified needs, rather than understanding the needs in the first place.

- .. To be able to accommodate our existing systems. We cannot assume that the business can afford to replace all the existing systems, and even if they could, the implied risks and challenges makes such considerations infeasible.

- .. To be able to adapt to change, whether it comes from the business or from the availability of new technologies. The architecture we create cannot impose a static framework, never able to accommodate change. We must be able to incorporate new business strategies and new technologies with minimal impact.

2.0 Three Models, Four Tiers

What we propose is an approach which relates three models:

1. Thefirst defines an architecture Reference Model (RM): the Reference Model is actually a meta-architecture which identifies and relates the *kinds* of information we expect to include in our EndState architecture. We start with a Reference Model because there is no universal definition of an architecture, and we need common concepts and language which multiple groups of people can work with.

2. The second model is the EndState Model (EM): the model that we would like to implement in a specific business. We propose a component-based approach where the EndState defines specific components within each tier, based on the kinds suggested in the Reference Model.

3. The third model is the Deployable Model (DM): the actual set of production systems which satisfy the EndState component specifications, to a greater or lesser degree.

Each model has the same four tiers:

1. Strategies Tier (ST). The business prepares its strategies: its principles, its objectives for quality, cost, scale, and performance, the events it wants to respond to, the results it wants to deliver.

2. Processes Tier (PT). The strategies are implemented as business processes --workflows and procedures people follow in responding to events.

3. Application Objects Tier (OT). Applications, which are assembled from Object-based Components, to supply the required process support .

4. Technology Tier (TT). The hardware, technical software, and systems management put in place to run the applications.

3.0 Tier Contents and Relationships

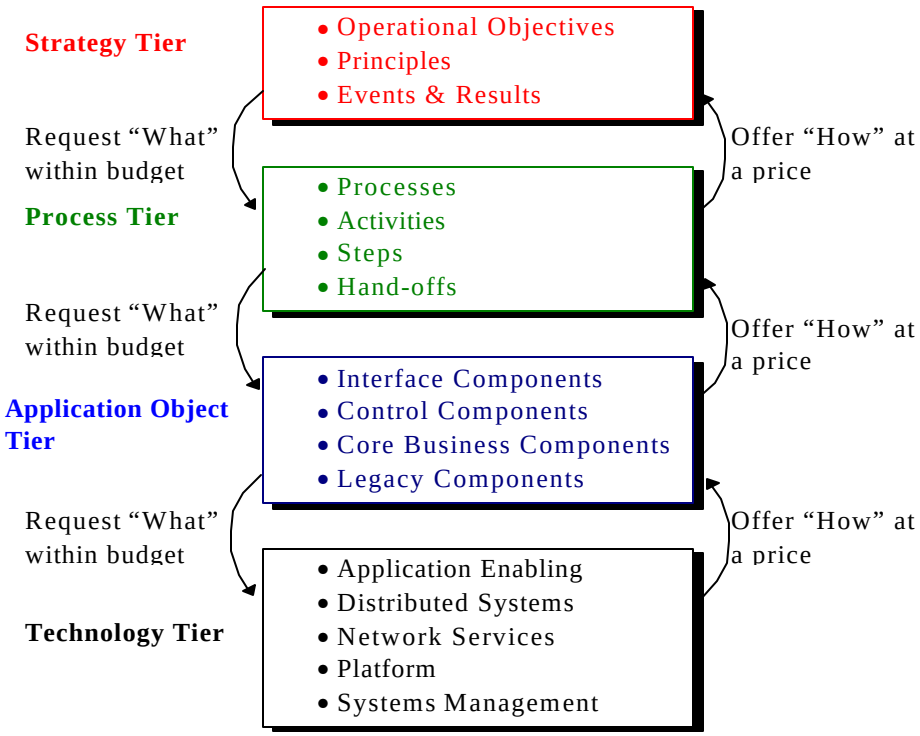


Figure 3 - The Reference Model

Strategies define what is to be accomplished, and how well.

Processes define how the work is distributed and controlled.

Applications define how technology is applied to process activities.

Technologies define how infrastructure components support applications. Our approach differs from many discussions on architecture [Zac87] in the tighte coupling of business strategies and business processes to technology solutions. If these links ai not shown, we cannot show that we are satisfying a business need. If we cannot show traceability to value, then we are left with the impression of doing technology for its own sake.

3.1 Relationships Between Tiers

Figure 3 shows between each tier there is an explicit "what-how" relationship. The strategies state "what" business processes must accomplish, and conversely, a business process represents "how" a business strategy is implemented. This "what/how" relationship continues through the complete model, enabling clear communication between the tiers of "what" requirements and of "how" counter-offers to service those requirements, as shown in Figure 4 - Relationship between Tiers.

We extend Jacobson's notion of use cases [Jac92] as the mechanism for communicating between the tiers. Each inter-tier use case specification identifies a set of inputs, expected results, relevant principles, operational objectives and size/volume metrics. Each such specification identifies specific actions on one or more components in the adjacent tier, and is the basis for accountability and traceability.

As for the components, we combine the notions promoted by Wirfs-Brock et al [Wir90] and the Object Management Group [OMG.93.12.29] to say a component represents a distinct set of responsibilities, which are visible through a set of interfaces, enabling developers to design t the interface, not the implementation. In this way, each component will express what it is capable of doing, in the same terms as the component expressing the need. This allows us to communicate on a common "what" footing among components, regardless of whether they are resident in the same or adjacent tiers.

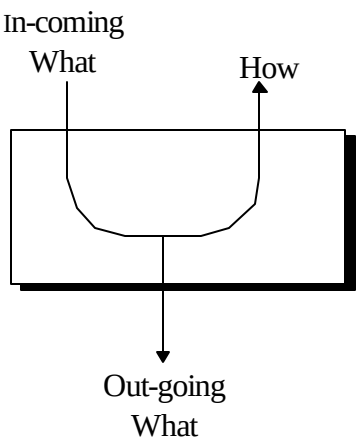


Figure 4 - Relationship between Tiers

However the incoming "what" expressed to a component may not be the same as the "out-going what" it will express to other components.

For example, it may be possible to implement a business strategy completely with manual processes. But if parts of the solution are to be automated, then we must refine the "what" incoming to the Process Tier to a "what" outgoing to the Application Object Tier. For example, Operational Objectives stated in the strategy may define end-to-end timing requirements of the implementing business process, while Operational Objectives stated to supporting application tier are focused to the activity being supported such as specific response time requirements.

3.2 Strategy Tier

Within the Strategy Tier, high-level business goals are formed into the specific components of interest to other tiers: events and their corresponding results, operational objectives, and principles.

Events define what the business is expected to respond to. Results state the expected outcome. Operational Objectives state quality, quantity, time, and cost objectives for the way that business processes will respond to specific events. Principles provide overriding factors, such a company policy, legislation, etc. Together, this information defines a use case for a business process.

3.3 Process Tier

As shown in Figure 5, processes are defined by partitioning them into activities and steps and showing hand-offs between activities. We define *Activity* as the unit of accountability within a process. Activities may be manually executed and hand-offs may be manually forwarded, or there may be automated tools. Activities and Hand-offs are among the components of this tier.

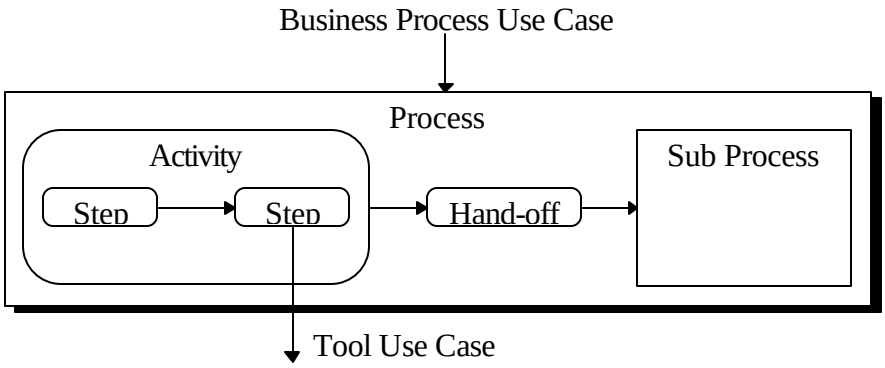


Figure 5 - Process Tier

3.4 Application Object Tier

As shown in Figure 6, one useful way of classifying the components of this Tier is the ICE model [Jac92], made up of Interface, Control (integrity glue), and Entity (core business and legacy system) component types. Another way might be by source: general, industry-specific, vendor, or custom. The Object Management Group, through some of its special interest groups, is addressing the problem from this perspective [BOM95].

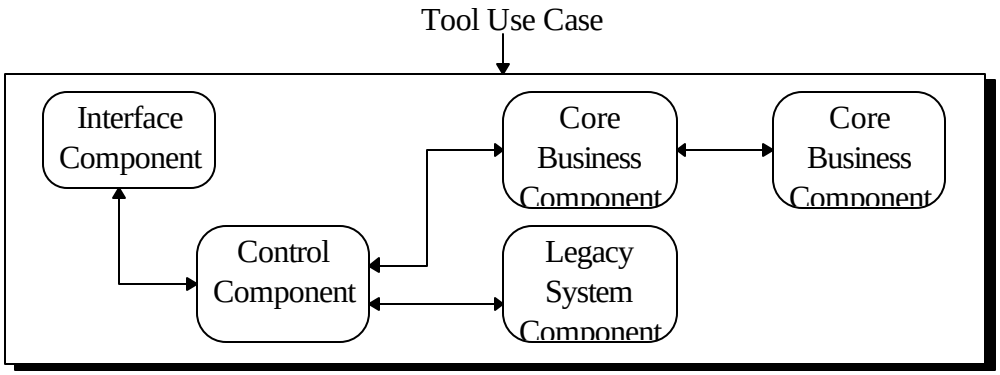
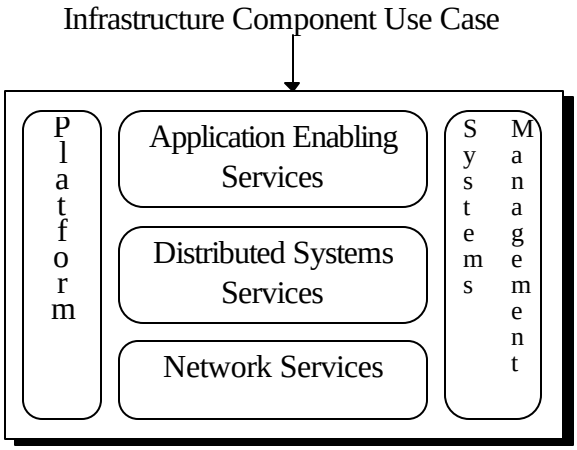


Figure 6 - Application Object Tier

The Tool Use Cases (Figure 5) define the requirements of responsibility-based Application Components. In fact, a use case from any one activity represents a subset of the requirements of an application component. The full set of requirements of a particular component is realised by combining the requirements of all use cases which employ it. Collaboration among components defines further "interior" use cases.

3.5 Technology Tier

The Components of this tier might be classified (following IBM's Open Blueprint [IBM95]) into the following component types: Application Enabling Services, Distributed Systems Services, Network Services, Platform Services and Systems Management, as shown in Figure 7. The Common Object Request Broker Architecture (CORBA) [OMG.93.12.29] and other sources might have different schemes.



Application Enabling Services support "applications" with regard to presentation, persistence, transaction monitoring, etc. Distributed Systems Services enable the distribution of function and data, while Network Services provide the communications infrastructure. The Platform Component includes operating systems and hardware, while Systems Management provides the necessary tools to manage the infrastructure.

Figure 7 - Technology Tier

Interactions with the Technology can occur explicitly (an application or technology componen may request services), or implicitly (resources, such as CPU time and bandwidth are consumed in passing). At least in the explicit case we can again employ the use case approach, specifying the inputs and expected results, operational objectives, and principles.

4.0 Model Contents and Relationships

4.1 Relationship Between Models

Figure 8 shows bi-directional relationships between the three models. Going from left to right, the Reference Model suggests the kinds of information to be collected and who might best define them. This in turn can suggest the definition of some specific EndState components that particular business will need. And this in turn can suggest how this can be Deployed, now or in some future phase, to make the business operate as intended.

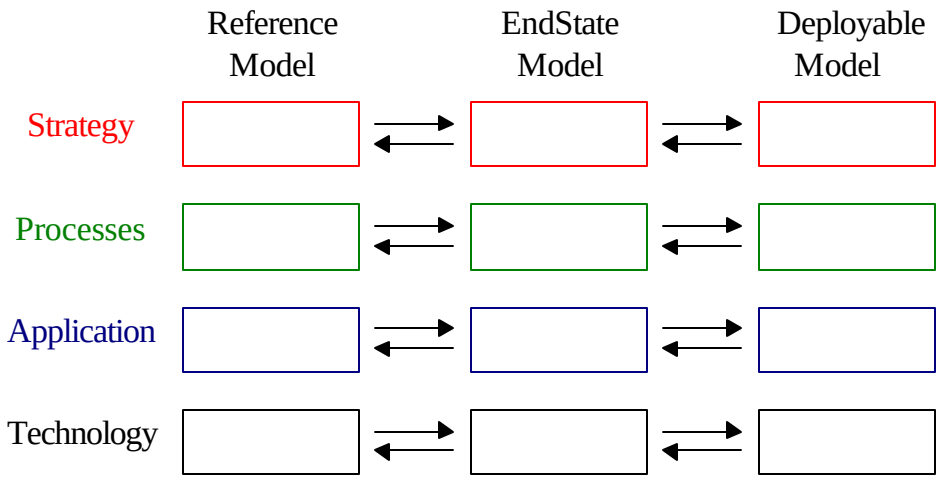


Figure 8 - Relationship Between Models

Conversely, going from right to left, the functionality of existing implemented systems tells us what needs are currently being met, which implies generic EndState components to model the capability. The Reference Model might usefully frame the thinking about what sorts of components should contain that functionality, or conversely the need for new components may suggest ways in which the Reference Model is deficient.

4.2 The Reference Model

The Reference Model defines the kinds of components that should be found in each of the four tiers. For example, we should include event definitions and result requirements in the Strategy Tier, activity and hand-off definitions in the Process Tier, methods and parts in the Application Object Tier, and operational objectives statements in every tier.

The Reference Model can be used in analysing a wide range of businesses; financial services, manufacturing, etc. It does not say what actual components apply to the analysis of a given business.

When a reference model is being prepared for a specific business, one could identify specific standards or guidelines to be used at each tier to guide the kinds of components to be included, such as OMG's CORBA or IBM's Open Blueprint.

4.3 The EndState Model (EM)

With a defined Reference Model, we are now in a position to state what kinds of component should be included in each tier of the EndState Model of a specific business. The Reference Model might tell us that the Process Tier includes Activity components, and we might decide that we want a TakeCustomerOrder activity in our EM. Or the Reference Model may state the need for Core Business Components, but the definition of a Customer or Person or Organisation component is left to the EM.

4.4 The Deployable Model (DM)

We can now map EM components to deployable targets in each tier. If the EndState Model tells us that we want a TakeCustomerOrder activity, then in the DM we must deliver one or more standard implementations of that activity. Similarly, having defined a customer Core Business Component in the EM, we have to buy or build one or more objects or wrapped legacy systems to meet the specification. Likewise, having decided that the EM needs a Relational DBMS component, we have to pick deployable products.

5.0 Managing Change

Deployable components will often be deficient with respect to the EndState Model. For example, wrapped legacy application components may lack the inheritance required of true objects as well as some required functionality. Deployed strategies may fail to live up to some EndState principles. The so-called EndState may need to change. Therefore we need migration plans to more closely align the DM to the EM, or vice-versa.

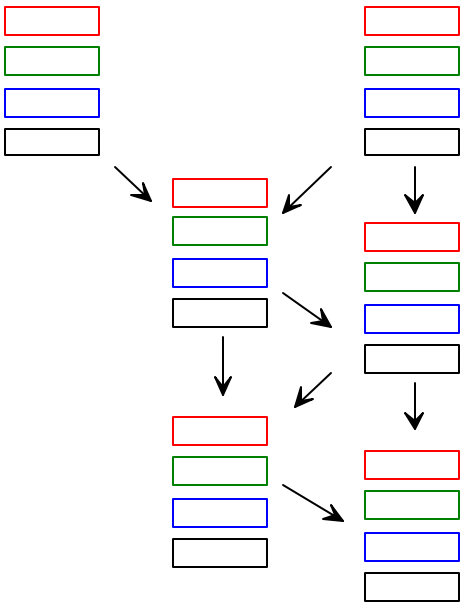


Figure 9 - Change and Realignment

Figure 9 shows a possible sequence of change and realignment of models.

We may begin by forming an EndState Model based on our Reference Model and the current Deployed Model. Then we may change Deployable Model to more closely align it with the EndState. At some point, we may change our EndState based on deployment experience, and then change the Deployable Model to match.

Change can be introduced for reasons other than alignment, ranging from changes in the Strategy to changes in the technology used to implement components. We can generalise this type of change as either a change in implementation (how) or a change in requirements (what).

While changes in implementation may be isolated by encapsulation, changes in requirements may affect the boundaries of our components, thus affecting the EndState Model. If we have built our components around natural and stable responsibilities, the functional model should be quite long-lived. But as Figu 10 shows, we may want component variants based on different operational objectives and possibly different operational interfaces. A new technology may sometimes suggest new

components by highlighting EndState operational deficiencies. Clients of the component who a

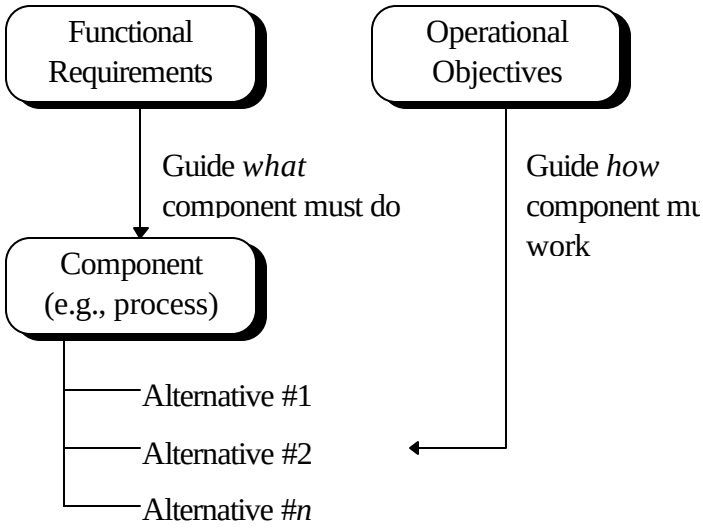


Figure 10 - Functional Requirements versus Operational Objectives

concerned only with functionality will see no difference.

If we must realign component boundaries, we prefer to create new components for new functions, rather than redefining existing components for existing users.

6.0 Packages

Deployable implementations, whether bought or home-grown, may not map easily to the components defined in the EndState Model as a result of their packaging. We view a package as a set of component functions and parts and responsibilities, which may be grouped on some basis other than the EndState Architecture's component partitioning, such as pricing or installation cycle or buyer or builder.

Packaging may therefore cause us some problems. A new package may include an already implemented component. Or the package may contain a component only partially matching our definition (either functional requirements or operational objectives).

7.0 Areas of Further Research

7.1 Accommodating Flexibility and Risk

To handle an uncertain future, we often want to describe a family of requirements scenarios at each tier. We would probably pay a premium for solutions which require minimum time and co to adapt when we change the request to one of the other scenarios. Should this be handled simply by developing a "how" offer for each scenario and comparing them, or can we devise more powerful techniques for handling families ?

7.2 Concepts and Terminology

Some industry groups are using terms such as Business Process and Business Component to refer only to application software elements. Our additional tiers extend these concepts. For example, we feel that Activities and Hand-offs are Business Components, though they are not software. We would like to find a common reference model over all four tiers.

7.3 Separating Functional Requirements from Operational Objectives

We need to further explore the difference between requirements and objectives. Our current approach is to distinguish Functional Requirements from Operating Objectives in each tier.

- Functional requirements directly *what* a component must do, what it must accomplish.
- Operational Objectives influence *how* the component is actually implemented.

Functional Requirements state the business needs in non technical, implementation-free terms. Operational Objectives, on the other hand, state goals which solutions are expected to meet or surpass, which are generally related to cost, time, skills necessary to execute a task, legislative requirements, etc. We use the term Business Requirement to encompass both Functional Requirements and Operational Objectives.

7.4 Identifying Events vs Processes

When is it best to recognise different events at the strategy level, and when is it better to merely distinguish different variants of "how" the event is handled ?

For example, in banking we could identify a withdraw-cash event, and state appropriate functional requirements. Objectives that might influence how the event is implemented could include: it must cost less than $0.25, it must be do-able directly by our customers, it must not take longer than 1 minute for the complete transaction, etc. However the same event could have a second set of objectives. For example, the above description may be appropriate for customers who use Automated Banking Machines, but a different set of objectives may be appropriate for those that use tellers.

From the Strategy perspective, should this be seen as one event or as different events? We can see the case for both, and therefore we need to better understand when the "how" should be "visible" and when it should be hidden as an implementation detail.

7.5 How the Framework Facilitates Development Processes

Some development issues we hope to address with the proposed framework include:

- **Application design reviews** - We believe it may be easier to separate process and workflow logic from business component logic, and technology issues from business issues

- **Package acquisition decisions** - We hope to get a clearer picture of both the fit with current requirements and the prospects for later reuse and recombination.

- **Business requirements gathering** - We would like to frame the discussion in terms of producing satisfaction for business events, in the belief that both the business and the three "how" tiers can find common ground there.

- **Technology upgrade business cases** - By following the suggested linkages, it should be easier to show how better technology positively affects the "good, quick, cheap" operational objectives of the strategy, process and application tiers.

- **Definition of use cases of various kinds** - Clarifying the relevant "client" tier should make it more straightforward to get a solid specification for a component in any "how" tier. For example, a component can be better specified by knowing the set of processes in which it will be invoked; a technology component can be better specified by knowing how component collaborations will use it. In fact, a component may be specified by a collection of such use cases, though we should never claim to know all the potential use cases.

- **Patterns and Frameworks** - The reference model needs to accommodate types of patterns and frameworks in each of the tiers of the EndState Model, but we have not yet worked out how best to do that. Our belief is that patterns will provide a standard way of documenting alternative general implementations. Such an inventory would help developer to plan, to find solutions with known costs and risks, and to communicate available solution to their clients. This latter point would help us make clearer counter-offers as suggested earlier (se**Error! Reference source not found.**).

- **Reuse** - We believe that the kinds of reuse that deliver the highest leverage occur at the highest levels of design. But of course we would like to encourage developers to build for potential reuse at all levels, and we hope to do this by aiming components at the TA rather than just at the current project. We also want developers to be able to find reusable patterns or components easily, and we hope that the use of the Framework will make that easier through categorisation by tier and model.

- **New Function by Recombination** - We don't want to place functional limitations on the solutions that can be offered in support of business strategies, but we also don't want to specify or implement excessively flexible components. We hope to find a better balance by placing more emphasis on delivering new function by new collaborations between existing components.

- **Flexibility** - We are interested in the notion of being able to accommodate the necessary amount of change through the flexible configuration of relatively inflexible components. T

analogy we use is that of bricks, which are in themselves highly inflexible, but a very flexib building material because of the variety of ways they can be combined.

- **Complexity**- While a given level of functionality probably implies some minimum level of complexity, we feel that most business systems greatly exceed such minimums. We hope to use the tiers and models to encapsulate components of all types, and to reduce the inter-connections to those which have demonstrable value.

- **The Framework Applied to the Development "Business"**- The Reference Model is not limited to specific line-of-business architectures. The same concepts may be applied to other domains, such as development. There is a systems development "business" within most businesses, and it has its own strategies and goals, processes and hand-offs, application components, and technology infrastructure. The framework may help to see which of these may be shareable with the parent business.

- **Resource Utilisation**- We would like to work out how to tie the actual and expected utilisation of resources (people, facilities, and technology) into both process design and process instrumentation.

8.0 Experience with the Framework

8.1 Building the Processes Around the Framework

We are deploying processes to apply the framework to real projects. This will ensure that the business gets increased sharing and reduced complexity benefits from its architecture, and that the EndState architecture remains fresh and responsive to real and current needs.

A second necessary process is related to research. We would like to place high-risk elements in an explicitly experimental context, and keep them out of production projects whose definitio of "success" is quite different. Research should investigate how new technologies, strategies, processes might be incorporated into the EndState architecture.

Applying the framework is reactive: it responds to specific needs of projects. Research is proactive: it tries to anticipate future project needs and do the groundwork beforehand.

8.2 Sources of components

To acquire the right set of EndState Model components, we can buy, build, or adapt.

For the Technology Tier, we can usually buy, or at least adapt. This area is relatively mature, and industry is becoming fairly consistent in their view as to what the component types should be. As CORBA[OMG 93.12.29] is exploited, we would expect to be able to buy deployable Common Facilities, Common Services, and Object Request Broker components for this tier, as well as some Application Objects.

This architecture is more focused on the middleware, however. If we want to dig deeper into the infrastructure to determine the right set of components "under the skin" we have to look to other sources. For example, IBM Open Blueprint [IBM95] defines a number of component types that would typically lie beneath the middleware.

The Application Object Tier offers as yet few standard component definitions. If we can afford the time, we can wait for industry or vendor standards to emerge.

Or we could define our own. This approach requires both domain and architectural skill. We prefer to employ standards in order to have the option to purchase components, so if we must prepare our own component definitions, then we should make a 'best guess' as to what the standards will be, and try to be ready to converge to standard components when and if they become available. It may be useful to develop target components by classifying empirically derived information in the way that the reference model suggests:

- Use domain expertise to define a starting set of components
- Review existing systems using their descriptions to refine the definitions of the corresponding components. Analysing several implementations of the same component will enhance the final definition.

In this process, we can expect to find previously unidentified components, as well as those that are not really well formed in current implementations.

We do have some experience purchasing components for the Process and Strategy Tiers; however these have been less industry standards and more 'best-of-breed' components.

8.3 Promulgation of the Architecture Framework

The need for architectural missionary work cannot be overstated. Each area involved in each tier has to understand the relevant models: the content, use, and value. We need to learn how to communicate an architecture, and how each stakeholder sees the value.

8.4 Accumulation of Information and Understanding

The Reference Model allows us to categorise the information we are dealing with. By being able to slot a new piece of information into the appropriate tier and model, we can see more readily how it relates to other information. For example, we can see how to relate a set of operational objectives to the strengths of a particular technology.

EndState Architecture components represent a convenient "hub" around which information car be collected. And, as components become more complicated, we can further sub-divide them into sub-components to manage the complexity.

We are experimenting with various supporting technologies for the development activities whi use and contribute to such information libraries. Hypertext seems promising as a way to suppor the multi-dimensional links called for by the Framework.

9.0 Conclusions

We propose a 4-tier 3-model framework for managing architectural information, and we believ that it provides us with an opportunity to address the issues stated at the outset.

The Reference Model gives us a general framework in which to articulate a EndState Model for a given line of business, facilitating a common vision based on common language and concepts. We can then tie the EndState Architecture to an Deployable Model by mapping EM components to actual packages.

By bringing both Strategies and Processes into the scope of the Framework we have allowed clear linkage to testable statements of business issues and needs. Making technology contribut to realised business value is the key measure of IT.

Through the tiers of the models and their components, we can partition the problem space. We can better apply the right skills to the right tasks and assign responsibilities. We can handle mo parallelism. We can separate issues of "what" from issues of "how" at each tier. We can further separate functional requirements from operational objectives. People are better able to see where their contribution fits, as well as those of others.

We can more clearly see where legacy systems can be recast as deployable (in fact deployed) implementations of desirable EndState components.

Future changes seem more manageable when the Framework allows us to see clearer limits to the ramifications of the proposed change, and to see which changes require new components and which require only new collaborations.

By examining our existing and future systems for useful patterns, we hope to build a repository of familiar general solutions which will help us reduce project risk and delay.

We have had some early success in attempting to use this approach to:

- store architectural information so as to help people navigate and cover the material.
- facilitate discussions with business users and IT people at various levels.
- expand project scope from IT only to include strategy and process development, and conversely to relate business initiatives to process engineering and to IT projects .
- partition discussions of requirements and solutions along lines of skills.
- attempt to relate legacy systems to target IT and workflow components.

Obviously a great deal of research and practical effort is required to flesh out this framework and to make it the normal mode of development thought and work. We expect this framework to shape our agenda for several years, but we also expect that some important benefits will be quickly realisable.

10.0 References

[BOM95] Business Object Management Special Interest Group, Object Management Group, Inc. *OMG Business Application Architecture: White Paper,* Draft 2, October 1995

[Fox95] Mark S. Fox, *Enterprise Integration Laboratory; Research Synopsis,* Department of Industrial Engineering, University of Toronto, July 25, 1995

[IBM95] *Open Blueprint Technical Overview,* International Business Machine Corporation, 1995

[Jac92] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach,* Addison-Wesley Publishing Company, 1992

[OMG 93.12.29] Object Management Group, *The Common Object Request Broker: Architecture and Specification,* Revision 1.2, 29 December 1993

[Wir90] Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener, *Designing Object-Oriented Software,* Prentice Hall, Englewood Cliffs, New Jersey, 1990

[Zac87] J.A. Zachman, *A framework for information systems architecture,* IBM Systems Journal Vol. 26 No. 3 1987

Abbreviations

BOMSIG	Business Object Management Special Interest Group (OMG)
CORBA	Common Object Request Broker Architecture
IT	Information Technology
OMG	Object Management Group
RM	Reference Model
EM	EndState Model
DM	Deployable Model