# Enterprise Re-Use

*Distributed object computing and business objects will cause re-use to happen. How will we manage it?*

## Successful re-use requires fundamental change

**Distributed Object Computing will create re-use opportunities**

Distributed object computing systems using message-bus middleware based on Object Management Group's Common Object Request Broker Architecture (CORBA), Open Software Foundation's Distributed Computing Environment remote procedure calls (DCE RPC), NextStep DO and other ORB-like products are coming into commercial use. Larger-scale information systems are being developed in important or mission-critical areas. Examples of such systems include Fidelity Investments fund management and trading desktop, fund management and front office systems at Wells Fargo Bank and subsidiaries, customer service systems being developed at MCI and a division of Pacific Bell, and trading systems at many of the major Wall Street investment houses and banks.

In distributed object systems, components are made available on the enterprise network to offer business services. In a funds management business, an *account* object might offer *balance*, *deposit* and *withdraw*; a *stock* object might offer *opening price* and *closing price*; a *trade* object could be *placed* or *settled*. In a telecom environment, a *premise* might know its *telephone number* and *service address*, a *residence customer* may have a *billing address*, and so on.  Given the requisite security clearance, any application could become a client of these objects, a consumer of their services.

**Sharing is an overlooked form of re-use**

One side-effect of distributed object computing (DOC) is often overlooked: re-use happens. DOC does not create "successful" re-use in the way the term is often used to describe a benefit of object technology. Rather, DOC creates an environment in which software components are inherently shared. Objects initially developed for one application become "re-used" as other applications are developed to use their services. Re-use is encouraged because DOC makes an object's services available in much the same way the RDBMS facilitated ad-hoc and shared access to date.

Sharing is a form of re-use which Data Administration organizations have sought to achieve using relational databases, corporate data models and data warehouses. DOC will supplement large database systems and make business objects, rather than simply data tables, the focus of sharing. Business objects enable the sharing of business process and business rules as well as data. DOC infrastructure is the platform of choice for delivering business objects to an enterprise audience because it masks many of the complexities of network distributed computing, and increasingly provides the services required for a robust distributed computing system (e.g. security, transaction control, adapters for interfacing to legacy data systems).

### Re-use is Changing

With the growing use of DOC and business objects at an enterprise level, the idea of re-use is changing. Initial enthusiasm for re-use focused on technology components to simplify development of technically demanding software such as graphical user interfaces and distributed systems. Re-use could be created, the argument ran, by using libraries of purchased or custom-developed software components. These libraries would replace the tradition of developers reinventing on project after project that functionality which was essentially common. Today, with multiple applications being developed to share a set of common business objects, the re-use challenge becomes one of management rather than pure creation.

In this scenario, however, there lurks an irony: re-use still must be created! As unmanaged sharing proliferates, each new application project is tempted to redefine the data or procedures being re-used. Historically, this behavior has been observed in applications which define and manipulate corporate data in conflicting ways. Thus, these applications “share” the data, but corrupt it through conflicting definitions, edits, and procedures. The same result can be achieved with objects if every application is allowed to redefine fundamental enterprise concepts in conflicting ways (and maintain separate storage as a result). The consequence is that re-use can be defeated in the long-term through corruption of the underlying re-usable assets.

### Values, Organization and Process

Management of the development and sharing of re-usable components must make re-use part of the process of developing systems - and part of the culture which develops those systems. Otherwise, re-use takes on a more anarchic flavor, and becomes a cost driver instead of a cost-saver. Re-use can be achieved on an economically-significant scale only by changing the values, organization structure and business processes of Information Systems organizations.

## Why Re-Use?

Re-use is beneficial because it avoids (eliminates) redundant work. The value of a re-used asset can be measured in part by development and maintenance work (time and cost) avoided. Perhaps more significant, though, are the consequences of re-use:

- Shared definitions of business concepts reduce data and process integrity failures, redundant storage of data (records or instances), ripple effect from propagating business change.
- Shared business processes reduces variation across business units where differences add no competitive or operating advantage.
- Reduced cost of operation through elimination of non-productive variance -- differences in business practices or policy for difference sake is a large, rarely-measured recurring expense in most large businesses.
- Reduced time to market - assembly of applications from pre-designed, pre-fabricated parts

- Easier for developers to come up to speed on a second product/application once they are familiar with the first application -- the parts used to construct both applications will be similar.

## Re-use COTS and Custom Parts

Re-use applies equally to purchased and in-house-developed assets. In lieu of a “development and maintenance cost”, commercial off-the-shelf components have acquisition cost (often a license fee) and a support fees. Based on my experience, the cost to an I.S. organization of acquiring reusable components from external vendors is less than half the cost of developing exact functional-equivalent components in-house (recalling that the external vendor amortizes development costs across multiple customers). The annual cost of maintenance is an even smaller fraction (commonly less than 20% of the license or purchase list price).

Other than cost, there are two crucial advantages to purchasing reusable components: time and reliability. Off the shelf components are available now, while “equivalent” parts developed in house would require (often significant) production lead time. Off the shelf components have other users (unless your organization is the first customer), thus have a known track record and bug list. Buying off the shelf components is the most cost-, time- and quality-conscious way to obtain re-usable components.

## Why Re-use is Hot Again

Re-use is not completely new. Large-scale re-use efforts have been sponsored by the US Department of Defense to promote interoperability of vendor products through Ada component libraries. Organizations such as the Re-use Library Interest Group (“RIG”) provide self-help and information sharing for re-use managers. Data Administration and enterprise data modeling are fundamentally about re-use of data assets. Programming libraries of shared routines are an age-old form of re-use.

Object components, however, offer a variety of characteristics which make them more compelling than procedural code and procedural designs as (pardon me) objects of re-use:

- Well-defined interface protocols -- design-by-contract and constraint specification combine to provide semantic interface integrity (as opposed to syntactic integrity which is generally available with procedural modules in most languages).
- Multi-purpose interface -- an object can support many methods, groups of methods, views on methods (as opposed to a single routine having a single interface).
- Combined data, procedures and rules create semantically and structurally coherent packages.
- Modularity can be enforced through encapsulation (in some languages and environments -- this is notably lacking in C++).
- Specialization facilitates adaptation of data, procedures and rules -- language-supported inheritance, delegation and composition facilitate using objects as parts and the idea of mass customization.

Distributed object computing takes parts which are well-suited to re-use (i.e. objects) and facilitates making them generally available across the enterprise. Object Request Broker middleware usually includes facilities for asking an object about its services.

Once parts are easily accessed by multiple users, and users can query objects about their services, a defacto re-use opportunity exists. The challenge, clearly, becomes one of management and focus, as ORB middleware helps to overcome some of the mechanical access barriers posed by traditional library storage mechanisms.

DOC middleware is not, however, the only driver of renewed interest in re-use. Today, many business managers and executives feel that the cost-benefit balance of information technology is upside down. They perceive that growing expenditures have not rejuvenated stagnant or declining IT leverage. On one level, therefore, re-use (of technology components) is seen as a way to cut the cost of new systems design and development. On a higher level, re-use (of business objects) is seen as a way to correct the data and process integrity problems which have reached crisis point in many large organizations -- and to enable the rapid assembly of new business processes as the business changes ever more quickly each year.

## What Is Re-Use?

The idea of re-use is changing because of the growing use of DOC and business objects at an enterprise level. While initial re-use efforts focused entirely on low-level technology components, organizations are increasingly looking to higher-level re-use opportunities such as building multiple applications that share a set of common business objects, or re-using standard architectures and large-scale design components. Let us examine these different levels at which re-use can occur. In order to do this, though, we will first briefly consider what is meant by “re use”.

**Third-party must use the component**

Re-use is the incorporation of a unit of work (e.g. code, design, models) by at least one project other than the one which developed a unit of work. Using a unit of work again within the same project is not re-use according to this definition. Re-use is about leverage (i.e. getting more from a sound investment) that is shared.

**Retroactive Value Added**

Dr. Adele Goldberg and Kenneth Rubin of ParcPlace Systems provide additional criteria for our definition: re-use involves use of an existing unit of work for a new purpose, and requires that existing "users" of that unit of work be able to expect benefit from improvements made by future re-users. This criteria require that value be delivered to someone other than the original creator, where even future improvements add value “back” to the original consumers.

This definition of re-use is akin to sharing. Something is created, and multiple consumers gain distinct value from it. When that thing is improved, and everyone can benefit. This definition rules out “cut and paste re-use”.

**Return on investment is required**

Economically significant re-use requires that the work's ***total life cycle cost*** (initial development or acquisition plus maintenance costs) be re-gained through re-use, and that some additional value (i.e. profit) be gained beyond cost-recovery. This we will term ***return on investment***: cost recovery plus profit. (Note that cost of operation in any particular situation has been excluded, as that would be weighed against the perceived benefits achieved in applying the product. Thus I define total life cycle cost and consequently ROI differently from ***total cost of ownership***, which would include the cost of operation.) This measure is based on avoided cost - that is, the cost of not designing, producing, testing and maintaining comparable components because the original one can fill future needs. If one assumes that the cost of re-development is equal to the cost of developing the first product, then at least two additional projects would need to use the initial product in order to generate a positive return. The first re-user avoids (and thus "covers") the cost of redevelopment, while the second re-user reaps the "profit".

**Making components re-usable can add up-front cost**

Avoided cost must be balanced with the cost of making a component or product re-usable. There is cost associated with making a product re-usable and managing it as a re-usable asset. From experience, a re-usable component can cost twice as much to design and develop as a non-re-usable one, but about the same amount to maintain. This cost doubling arises from considering (up front) or adapting to (after initial development and deployment) a broader set of requirements than posed by the initial user of the component. (The reader may note that this cost factor is based on informal metrics. The commercial software community does a poor job of tracking and allocating primary costs, not to mention secondary costs due to re-use.)

By this reasoning, re-use is not economically significant until at least three additional projects have used the unit of work: the first "covers" development cost; the second amortizes the cost of adapting the component for re-use; while the third reaps the initial profit by cost-avoidance.

**Two criteria for amortization**

Thus, the definition of re-use can be constrained by economic impact, and a component would not be considered "re-used" until a threshold of use-instances (i.e. 3 projects) or a cost recovery goal (i.e. avoided cost is greater than actual cost) was met. Or it can be based simply on the second-user standard. In each case, the principle in common is amortization of the cost of a component across multiple and separate consumers which delivers a favorable return on investment.

## Levels of Re-use

**Rising above code-level re-use**

Initial interest in re-use focused on code-level components. Over the last several years, many organizations have raised their sights: higher-order components are increasingly the focus of re-use efforts because they promise greater return on investment. Let us consider why.

Consider by way of example a technology component: an object which displays a graphic user interface window on a computer screen. Perhaps this component was developed for the Microsoft Windows environment in the C programming language. This component would have value to other C programmers using Windows, but no value would be recouped by programmers developing in the various Unix environments, under OS/2 or Macintosh, or in languages incompatible with C. A similar component would be needed in each of the other environments in which the organization developed applications. Application programs designed to use this component would, likewise, need to be different in each environment -- although they could all be consistent in using a common object for displaying a GUI window.

**Design-level re-use achieves portability**

This situation need not pose a problem, provided that the organization focuses on design re-use. In the above example, one would likely require slightly (or very) different code-level components in environments as different as Macintosh, Windows and Unix X/Windows simply because the environments themselves are different in their designs and interface specifications. The basic principles of opening a GUI window, however, can be abstracted into a component design that would provide the application programmer with a uniform services interface across these environments. This is the point of design-level re-use. The application programmer can develop an application for multiple environments with design-level re-use, and can also reap benefits from code-level re-use within a given environment.

**Small-grain vs large-grain components**

So far, we have discussed re-use as though the target was a small-grain part such as a GUI window or button component or string and integer classes -- software development nuts and bolts. Greater benefits can be achieved by re-using large-grain components such as business-level design patterns, business objects and application frameworks.

**Larger is Higher Leverage**

Larger-grained components (like business objects and application frameworks) bring greater re-use leverage. Why? Integrating small-grain components into a developer’s thinking process involves more design work, searching for the “right” component, and integration than standardizing on large-grain components. When integrating large-grain components, the logic of component design is reversed: instead of “figuring out” how to integrate a large number of small components into a design conceived by the developer, the developer conceives the solution in terms of the design inherent in the (smaller number of) large-grain components themselves. Each component represents a comparatively larger element of the solution.

**Architecture-driven Re-use**

This is architecture-driven design, which shall be discussed in the context of patterns later in this report. Before continuing this exploration, let us characterize the units of re-use, and the levels at which re-use can profitably occur.

## Categories of Re-Usable Components

**Why Objects are Good Components**

An object is a black-box building block that represents a real-world (business domain) or computer technology concept. An object packages the ***data*** (attributes or variables) together with the ***procedures*** (methods or services) and semantic ***constraints*** (rules and policy) that are relevant to the concept which the object represents. Objects are hermetically sealed packages. An object's data are only accessible to the outside world through the object's methods. The outside world does not even know how the data or methods are implemented. (This is called ***encapsulation***.) An object presents a clearly-defined interface by which outsiders may use its services. This interface makes objects effective units of re-usable work because the semantics of the object can be defined and managed at the interface.

**Viewpoint scopes component re-usability**

Objects are always defined from a point of view. If the object serves a narrow set of needs (i.e. display a GUI feature), this viewpoint can be quite simple. If the object represents a wider viewpoint (i.e. a ***Customer*** object which serves multiple purposes across an organization) it may have a much more complex interface. In fact, not all properties will be relevant to all users of such objects defined from a corporate viewpoint. (In effect, the viewpoint and the object reverse roles here, and the object is externally defined by the collection of viewpoints.) This characteristic makes objects excellent units of modularity for business analysis, systems design and program construction because the boundary of that modularity is defined by some external (business or technology) concept instead of by units of programming (e.g. load modules and compilation units). The more closely an object's definition aligns with the problem or solution space, the more useful it is as a re-usable component.

**Three categories of objects**

There are, in fact, three categories of objects: ***business, technology and application objects***. An object lives in one and only one category. It is thinking about and organizing I.S. activities in terms of these categories of objects that delivers much of the power objects can offer as re-usable components.

## BUSINESS OBJECTS

***Business objects*** are objects that represent a person, place, thing or concept in the business domain. They package business procedures, policy & controls around business data. Business objects serve as a storage place for business policy and data, ensuring that data is only used in a manner semantically consistent with business intent.

One of the greatest costs in business information systems arises from failures to re-use business data, process and rules. We often refer to this failure by its result: data integrity problems. Business objects can resolve this problem: instead of limiting business to sharing only data, business objects allow sharing of business concept definitions, process, policy *and* data. In ROI terms, re-use of business objects is a better investment than re-use of technology objects, even though exactly the opposite emphasis has been seen in most early re-use efforts.

**Viewing business in business terms**

Business people often look at a business in terms of the things which comprise it and the processes which work on those things: for example *customers*, *products*, *orders* and an *order fulfillment* process. Business objects are specialized further into two subcategories to represent the business as we think about it: **business entity objects** and ***business process objects***.

**Business entity objects are actors**

Business entity objects represent people, places and things, in much the same manner as a data-modeling entity. Among the differences between a data entity and a business entity object: (1) the object packages procedure and rules, while the data entity packages only data; (2) the business entity object can engage in a far richer set of structural relationships that is available in ER modeling, such as type classification, roles, composites, aliases and populations. A business entity object represents a tangible or intangible business noun.

The instances of a business entity objects are records of data or facts about the business noun. The constraints and procedures packaged in the object type (or class) represent the actions which can be taken by (or upon) the business noun, and the rules under which this action can occur. The relationships between business entity object types characterize the identity of these objects, and real-world associations in which they engage. (See Illus. 1) The type structures which define the specializations of one object type from another, and the role structures which define the assumable behavior, collectively specify structural patterns of definition and identity which give business entity objects their meaning in the business context.

## Business Entity Objects and their Relationships

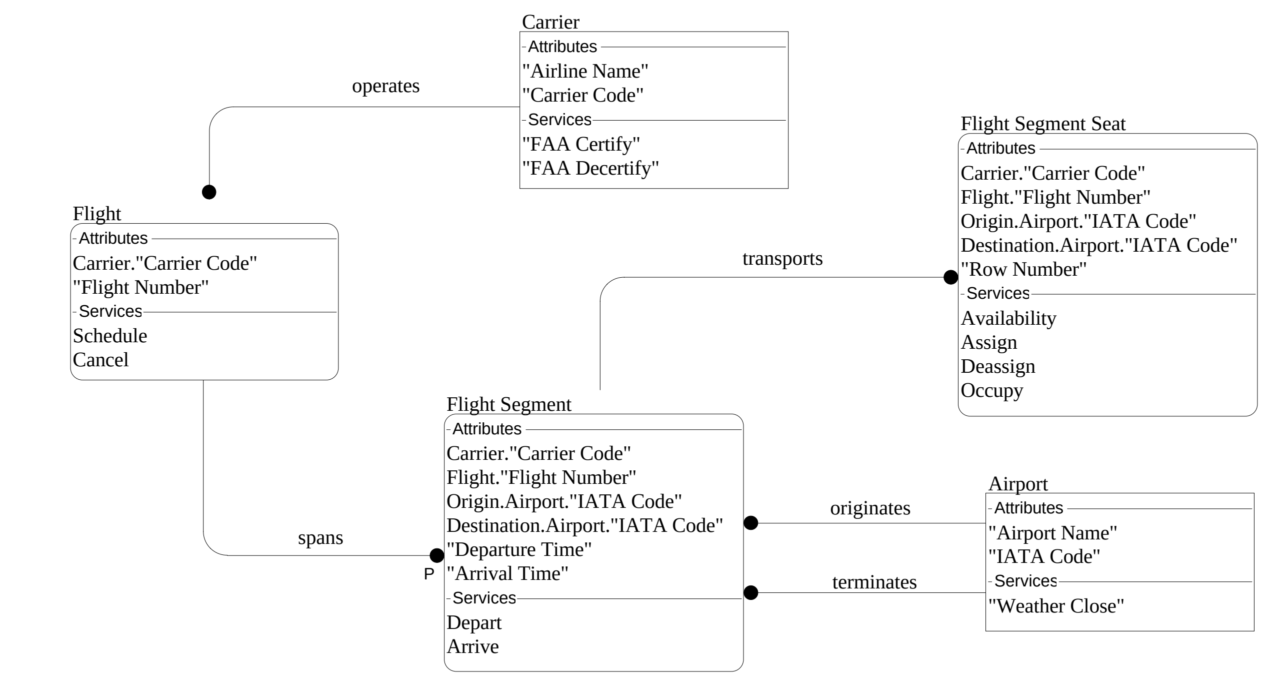

*Illustration 1. : Business entity objects with selected services and attributes displayed. This is also an example of a business structural pattern, to be further discussed below.*

**Business process objects are actions**

Business process objects represent business verbs. They represent business processes (as opposed to procedures), where a process is characterized by the interaction of a set of business objects which interact in a predictable, repeatable manner to produce a recognized business result. Business entity objects are the actors or role-players that carry out the business process. Each interaction between a pair of business entity objects represents one work step in the business process. (See Illus. 2)

Just as a business entity object can have instances, so can a business process object. Its instances represent in-progress units of work. Its attributes reflect the process state or are references to the business entity objects which carry out the process. Its services start, stop, step and status units of work. Business process objects can also be specialized from (that is, subtypes of) more generalized or abstract process objects. One would expect this, as business entity objects and business process objects are both specializations of the concept of business object.

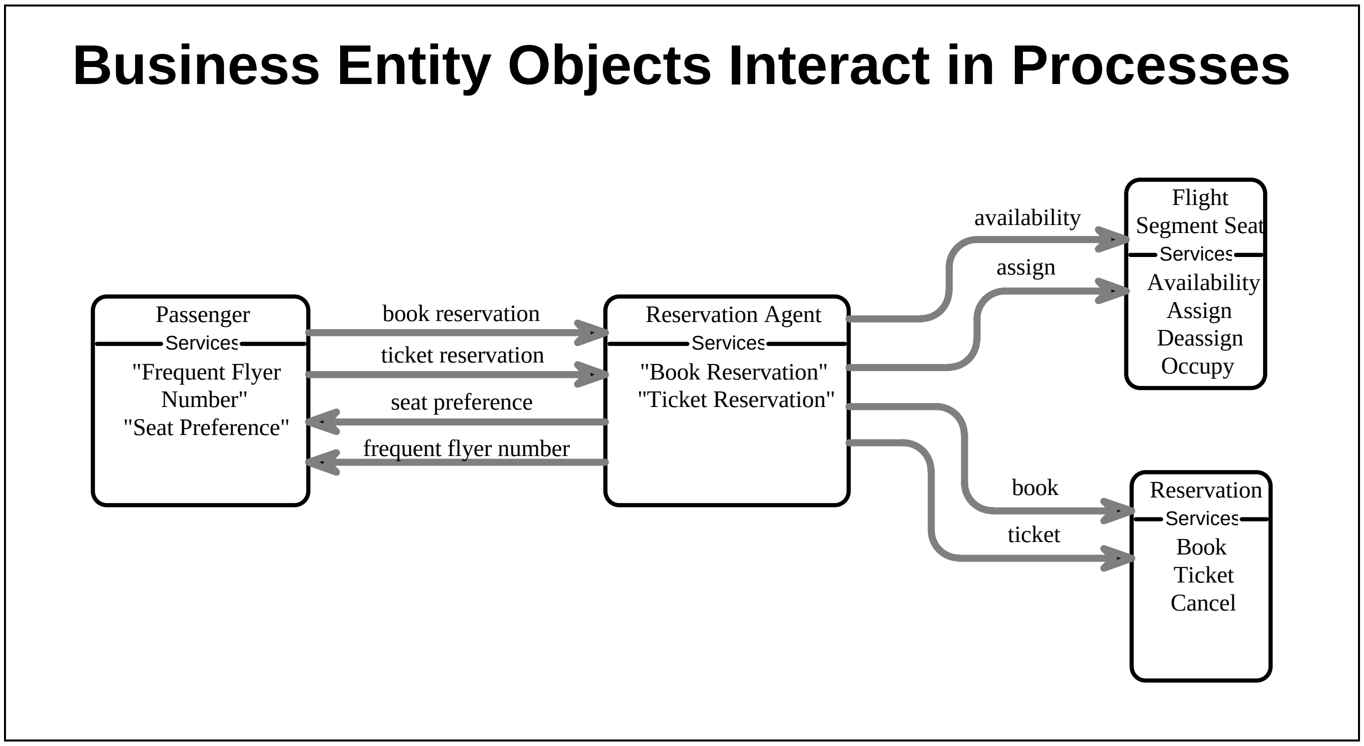

*Illustration 2. Interactions between business entity objects which comprise the worksteps of a business process object are shown in OOBE Object Behavior Diagram notation.*

## Where Business Objects are Re-used

Business objects are candidates for re-use on several levels:

1. Within the organization, to share data, implement standard business policies, and eliminate redundant procedures;
2. Within an industry, so that members of an industry need not reinvent the 80% of their business which is fundamentally like others in the field;
3. Across industries, as a mechanism for integrating vendors and suppliers in a value chain;
4. Generic objects, as a foundation from which to jump-start design of any business model.

Let us consider examples of each.

### Level 1

All efforts (object or otherwise) at data and process standardization in any company have been aimed at the first level. Most organizations that are developing custom re-usable business objects for internal use are also targeting the first level - even where the target may be phrased as standardization across applications.

### Level 2

Several industry consortia and vendors are working toward the second level. Petroleum Open Software Consortium (POSC, Houston, Texas) and Sematech

(Austin, Texas) have developed industry business object models for, respectively, petroleum exploration and extraction, and semiconductor manufacturing. Infinity (Menlo Park, Calif.) offers a financial/trading class library.

**Level 3**

Digital Equipment Corp. (Maynard, Mass) offers a business object model with specialized variants which has been used in a variety of industries: semiconductor (discrete) manufacturing, oil & gas, pharmaceutical (process) manufacturing, mining and banking. Software2000 (Hyannis, Mass) offers business object model products for two business areas which cut across industries: human resources and accounting.

**Level 4**

Digital Equipment Corp. (as part of the product listed above) and Open Engineering Inc. (San Francisco, Calif.) offer what one might call “base” or generic objects which form the basis for developing one’s own business model. In both cases, these re-usable parts are offered as part of a package which includes services and tools. The reader will find that this idea of offering generic objects based on a firm’s experience in modeling or systems development will become increasingly common, and object-experienced consultants will often work from such “strawman” object models.

## TECHNOLOGY OBJECTS

Technology objects represent a programming or technology concept, and thus are the building blocks of applications and implemented business objects. Examples of technology objects include GUI components, object request brokers, databases, base data structure classes and application frameworks. Technology objects comprise the infrastructure from which systems are built. Even more than with business objects, technology objects can be essentially the same across companies.

As with business objects, significant opportunities for re-use exist on several levels:

⇒ Basic software development technology, such as base classes for data structures (numerics, money, strings, etc.), date/time utilities, etc.;
⇒ User interface components which provide an abstract mask over the differences between user interface environments from the software developer’s viewpoint;
⇒ Distributed computing middleware, such as ORBs which provide a virtual computing environment which spans the network;
⇒ Robustness services, such as transaction processing, name/location management, secure messaging, etc. (OMG calls these Object Services);
⇒ Storage, such as components which provide a uniform storage model across different storage management products;
⇒ Workflow management, a cross between storage and transaction services for running business processes;
⇒ Adapters, software components which provide access to legacy data, transactions or applications;

From a ROI perspective, technology objects should be purchased off the shelf. Little (if any) competitive edge can be gained from custom-building one’s own technical infrastructure when commercial off-the-shelf (COTS) products exist to perform such functionality.

### APPLICATION OBJECTS

Application objects are built to solve business problems, as are applications today. They focus on a specific work task or problem, though they may be useful in other situations if well designed. They are assemblies of business objects & foundation objects, glued together with program code. Using client-server terms, application objects are clients of business objects. Examples here would include Order Entry, Quarterly Report, and New Account Setup applications.

Application objects are not high-priority re-use targets. They are (or at least can be) highly personalized. The bulk of re-usable components have been “extracted” into business and technology objects, leaving applications rather pure to form: literally, ways of applying the technology and business components. In any event, applications developed from an application-centric perspective will yield fewer re-usable components to the enterprise because the components were developed to fit a particular purpose. This is not to say that application objects may never be reusable, but their re-use delivers significantly less systematic value because they are inherently smaller in scope, more specific in purpose (viewpoint), and designed to solve a specific (rather than general) solution.

## Scaling Up: Frameworks and Patterns

### TECHNOLOGY FRAMEWORKS AND PATTERNS

**Frameworks enable design re-use**

A ***framework*** is a collection of abstract and concrete classes, and the interfaces between them. Wirffs-Brock, Johnson and Johnson have defined a framework as the design of a subsystem in the same way that an abstract class is the design of a concrete class. In an application, this subsystem might be the mechanism for data retrieval and display which is used throughout an entire application system. A framework provides a model of interaction among several objects belonging to classes defined by the framework. Thus, a framework is a set of classes which are associated by a set of structural relationships. Instances of these classes can interact in various ways which are delimited by these structural relationships. These structures and interactions form what we shall call ***patterns***. The framework is a set of configurable parts (the concrete and abstract classes). The parts can be configured to interact in many different ways with the set of patterns provided by the framework. The framework defines the patterns.

Like the pre-fabricated components used to build many modern "tilt up" office buildings and homes, design frameworks can be rapidly configured and assembled into a wide variety of software applications. Generally, application frameworks serve a particular purpose, such as the construction of on-line screens which store and retrieve data from an SQL DBMS. A framework built for one general purpose might not be useful for a very different purpose -- for example the on-line application framework described above probably would not help us build a real-time telemetry system. In practice, however, the vast majority of applications supported by most I.S. organizations are on-line and reporting systems. Frameworks can provide significant cost and time savings, while delivering more bug-free code per developer day. An application framework is a large-grain re-usable component which can substantially leverage the software development process.

## BUSINESS OBJECT FRAMEWORKS AND PATTERNS

**Business frameworks offer more leverage than business objects**

Experience with BPR suggests that all businesses operate a similar set of fundamental business processes -- although each business tends to implement those processes in different ways. One would expect, then, that the fundamental processes of any business could be represented by a common set of business objects. I call these ***business process patterns***. Furthermore, one finds certain structures repeated across businesses. For example, all businesses have some concept of *customer*, *supplier* and *employee*. These business objects are roles of another (abstract) business object which is commonly called *legal party*. This is an example of what I call a ***business structure pattern***.

Thus, a set of abstract and concrete business object classes with a set of built-in business process patterns and business structure patterns would be a ***business object framework***. One would shape such a framework into processes which could serve many different businesses. Specializations of components would enforce the unique business policies and rules of an organization. Specializations of business process patterns would implement core business processes in a way that reflected the company's unique competitive edge. Specializations of business structure patterns would reflect differences in definition and emphasis of the components within the business processes.

**Contribution to Re-use**

Business object frameworks have significant implications for large-scale re-use. One could envision commercial off-the-shelf generic business models, or others which were targeted to a specific vertical industry. Such products exist today in several industries, and are under development for near-term release in others. Research into the supply of commercial off-the-shelf business object frameworks is being conducted by the Object Management Group's Business Object Management Special Interest Group (OMG BOMSIG) which this writer chairs. The results are expected to unearth many new products in this arena.

Sharing a common set of technology components across a mid- to large-size Information Systems organization requires strong internal management practices. Sharing business objects across an enterprise requires an even more robust management and process structure, as both the business and IS “sides” of the organization must be involved to define, specify, specialize and maintain business entity and process objects which accurately reflect the operations of the enterprise. Thus, the balance of this report will examine an organizational model which focuses IS on its core processes -- providing the robust organization and process structure needed to approach this level of enterprise re-use.

## Organizing for Re-Use

**Most IS organizations are functionally structured**

Today, most I.S. groups are functionally structured. The hierarchy commonly decomposes into operations, data administration, applications development, and (sometimes in a separate group) maintenance. Often, there are also functional units for testing/release control, technology planning, LAN management, telecommunications, and end-user computing. It is common to see I.S. further divided along hardware brands or platform size (mainframe, midrange, desktop). Functional groups represent skill sets delivering like units of work: all programming is in one group, separated from design or analysis groups, because programming skills are presumed interchangeable, and the skill areas are presumed to work best when separated.

**Little IS funding goes to infrastructure**

In today's functional-structured I.S. group, it has been my observation that 90% of development and maintenance funding (e.g. excluding operations, equipment leases, etc.) goes to functional groups responsible for applications. About 5% is allocated to cross-application activities like BPR, data administration, data base administration and business modeling. Another 5% goes to technology planning and strategic architecture.

In many organizations, funding is only available for project-related activities. Cross-project or infrastructure activities are simply not funded, either by policy or because no equitable mechanism is in place for allocating budget to collective or cross-departmental efforts. When technology is purchased, it is too often selected for, funded by and used on a single project.

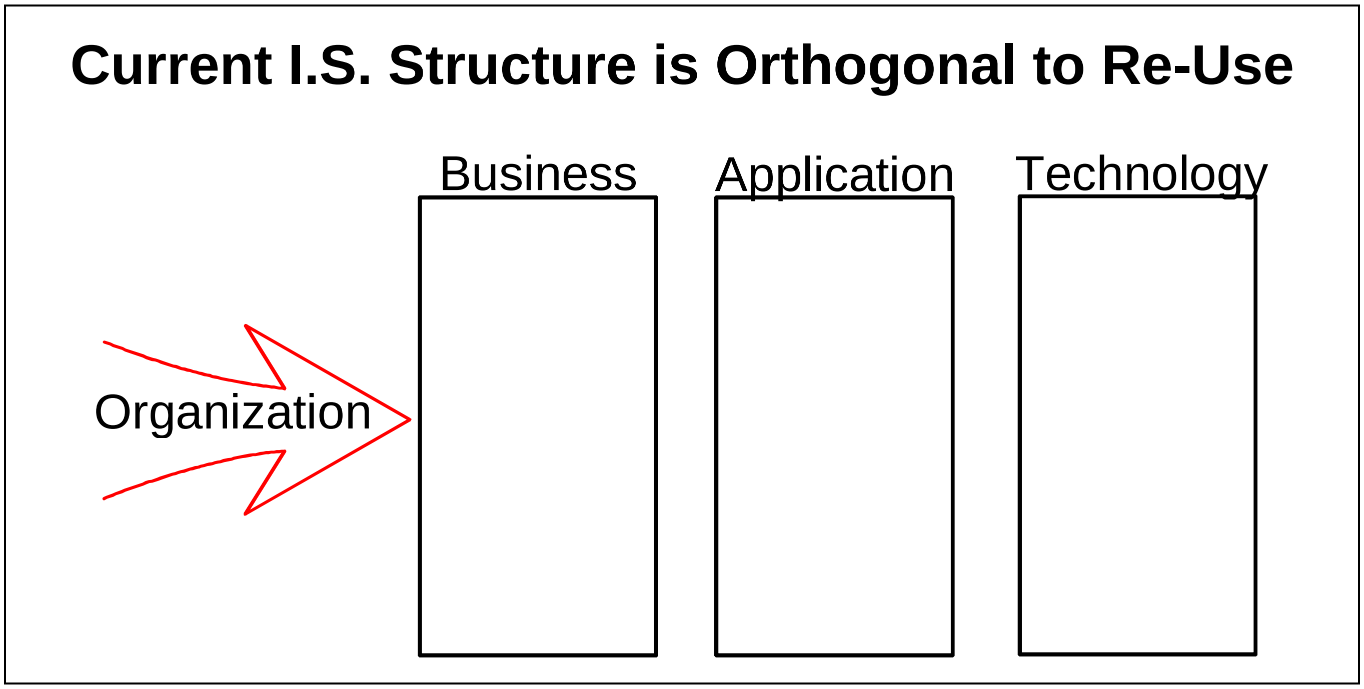

*Illustration 3. Today, most Information Systems organiations have a functional structure, which fragments the group into application-focused teams and is orthogonal to the actual processes which must be implemented. The required processes align with the three categories of objects: deliver business components, deliver technical infrastructure and deliver applications. All three function as interlinked manufacturing lines.*

**IS should be process structured**

To substantially change this picture, IS would organize around its own core processes. (See Illus. 3) The value structure would shift *from* a craft wherein the focus is on the supposed defects in off-the-shelf products which require development of completely custom, non-standard alternatives, *to* a manufacturing process wherein the focus is on value delivered to the business.

To accomplish this, IS would organize as a set of manufacturing processes that delivers three types of components: business, foundation and application objects. Because the business and its processes are captured in the business objects, understanding the business itself is no longer part of the application development process. The technology infrastructure is captured in purchased and integrated foundation objects. Because technology infrastructure is a corporate function shared by all applications, the focus of application design shifts to designing what the application does rather than how technology is assembled.

**Who is Doing This?**

Several sizable firms have started migrating their I.S. capability toward this model. A few are mentioned here in brief. Connecticut Mutual Life Insurance (Hartford, Conn.) redesigned their I.S. organization and had all employees reapply for the newly-designed job positions based on experience and skills. A major Canadian petroleum refining firm aligned their I.S. development capabilities to produce business, technology and application components. Wells Fargo Bank (San Francisco, Calif.) is experimenting with this model in part of their I.S. organization. Another insurance firm, one of the top 10 property and casualty insurers in the U.S., has been examining this approach in the context of an organization with SEI CMM Level 1 characteristics.

Not everyone has been successful. One major casual clothing manufacturer/retailer started the process of redefining organization and job roles, even to the extent that employees were competing for new jobs. The organization backed out of a plan to restructure I.S. around business components due, in part, to the politics around breaking up established application-centered power structures.

Perhaps as a leading indicator (or a warning call, depending on the industry in which the reader happens to work), a number of leading Japanese industrial and utility giants are starting down this path. This organization model for I.S. dovetails nicely with established Japanese manufacturing practices, in particular with the thinking of Deming and such practices as just-in-time manufacturing.

In the balance of this report, I will present a vision for organizing I.S. around a flexible manufacturing process structure. This vision reflects work being done at a few organizations that are pushing the edge of the envelope today. It is, however, identical in principle and concept to work that is already completely proven and in place at hardware manufacturing organizations world wide. And the reader may recall that only a few years ago, every “edge of the performance envelope” computer was built around custom, discrete component processors...

In this analyst’s assessment, supported by the work of groups like the Software Engineering Institute, it is the software manufacturing industry that is presently behind the power curve. This vision proposes a set of steps forward.

## Infrastructure would be Funded more than Applications

As a result of such realignments, one would expect both the I.S. organization's topology and resource allocation to change substantially as the process and organization mature toward this flexible manufacturing model. In particular, most of IS funds and personnel would be allocated to infrastructure development and maintenance -- not to application development and maintenance. The following proportions are based on my own observations of IS projects, and the input of other IS and consulting professionals:

- 60% of the development and maintenance funding will be allocated to the business object delivery process, including tasks as variant as BPR and business object server design.
- 20% of the funds will be allocated to acquisition, integration and deployment of a shared technology infrastructure.
- 20% will be used to develop and maintain "applications".

These figures were developed by estimating the proportion of application project resources which are actually used to develop the “business”, “technology infrastructure” and “application” components of typical application systems today.

Although many IS professionals might assess the “cost” of technology infrastructure in today’s systems as greater than 20% of budget, this analyst believes that the cause is structural: mechanisms to acquire, maintain and enforce a shared technology infrastructure are infrequently in place and effective. In many IS organizations, each application project develops most (if not all) of its own technology infrastructure. Therefore proportionally more budget and time are allocated today to this area than are actually required (individual, legitimate exceptions *notwithstanding*).

On the "business" side, the structural burden created by the traditional application development technology, process and environment frequently prevents projects from meeting business requirements. As a result, large parts of functionality needed to operate or leverage the business are not delivered until years after they are needed -- or simply not delivered at all (Paulk, Weber, Curtis and Chrissis, The Capability Maturity Model, Addison-Wesley, 1995). Given:

- the 1-4 year applications backlog in large I.S. shops, and
- my experience at virtually every I.S. organization with which I have worked or consulted in the last 14 years that the end-user has stopped asking I.S. for many needed systems and functions, therefore making the backlog artificially low,

this analyst believes that the actual demand for business functionality is far greater than today's budget allocation would suggest. If one takes seriously the reengineering ideas of Davenport (Process Innovation, Harvard Business School Press, 1993), Rummler & Brache (Improving Performance, Jossey-Bass, 1995), and Hammer & Champy (Re-Engineering the Corporation, Harper Business, 1993), and the continuous improvement ideas of Deming (Out of the Crisis, MIT Press, 1982) and the Software Engineering Institute, it is reasonable to conclude that the "business" component of I.S. responsibility will consume a clear majority of funding, time and staff resources in a mature I.S. process and organization.

## Experience with re-use exists in Data Administration

Several lessons have been learned by the data community which should be applied to object re-use. Numerous organizations *have* successfully jockeyed conflicting definitions and data structures onto common ground. Applications *have* been successfully developed to share corporate data. We do know how to re-use data assets. We also know from experience that organization, process, skills and funding are required. Organized re-use does not just happen - it requires a well-resourced team with appropriate political and technical skills, tools and sponsorship. Re-use is not a by-product of application development - corporate entities and processes must be defined from a company point of view to provide enterprise-wide perspective (i.e. a perspective which reflects how the whole company, as opposed to a single group or applications, views such things as *customer* or *product*). Tools and methods will not overcome defective values or a counter-productive organization structure. The big problems are human, not technical.

## Business objects resolve data sharing problems

Business objects put a new spin on an old challenge. Traditional data management functions faced conflicts created by the organization's structure itself. Data professionals collided with application developers because the semantics (business rules and processes) intended to work upon and preserved the integrity of shared data were under the control of someone with a different agenda, budget, schedule, outlook and management reporting structure. Business object resolve this conflict by repackaging business data and process. This is why upwards of 60% of an IS organization's development and maintenance resources would be applied to business objects: work now assigned to application developers would now be assigned to business object developers. Hand-in-hand with the re-assignment of work would go a reassignment of workers and management. This change would leave behind only true *application* development and maintenance.

## Business object teams

Business Object Management involves deployment of teams to manage the definition, development and deployment of business entity objects and business process objects. Each team would be completely responsible - from concept through code and operation - for the components it managed. By aligning such teams closely with the core concepts and processes of the business itself, IS is more capable of responding quickly to business change. Moreover, IS is structured around the processes required to deliver high-value re-usable business components.

| ***Generalized Business Entity Object*** | ***Specialized Business Entity Object*** |
|---|---|
| Financial Instrument | Stock |
| | Bond |
| | Managed Fund |
| Account | Fund Account |
| | Customer Account |
| Exchange | Market Trade |
| | Swap |
| | Loan |
| Legal Party | Customer |
| | Traded Company |
| | Bond Issuer |

*Illustration 4. A list of business entity objects from a hypothetical brokerage and investment fund management business.*

Earlier in this report, we identified some candidate business objects from a brokerage business. Here is an expanded list from which we can build an example organization under the Business Object Management concept. (See Illus. 4 and 5) Please note that the sole purpose of these example objects is to illustrate the organization structure proposed in this report, and not to suggest a complete or correct business model.

| ***Business Process Object*** |
|---|
| Manage Fund |
| Broker Exchange |
| Lend Asset |
| Collect Asset |
| Disburse Payment |

*Illustration 5. A list of business process objects for our business example.*

Business entity object teams would be formed around *financial instrument*, *account*, *exchange* and *legal party*. Subteams would be responsible for the specializations of each major abstraction, for example *stock*, *bond*, and *managed fund* subteams would work within the *financial instrument* team. These groups would be responsible for the full life cycle of their component(s) (i.e. business modeling, software and data design, coding, database management, etc.)

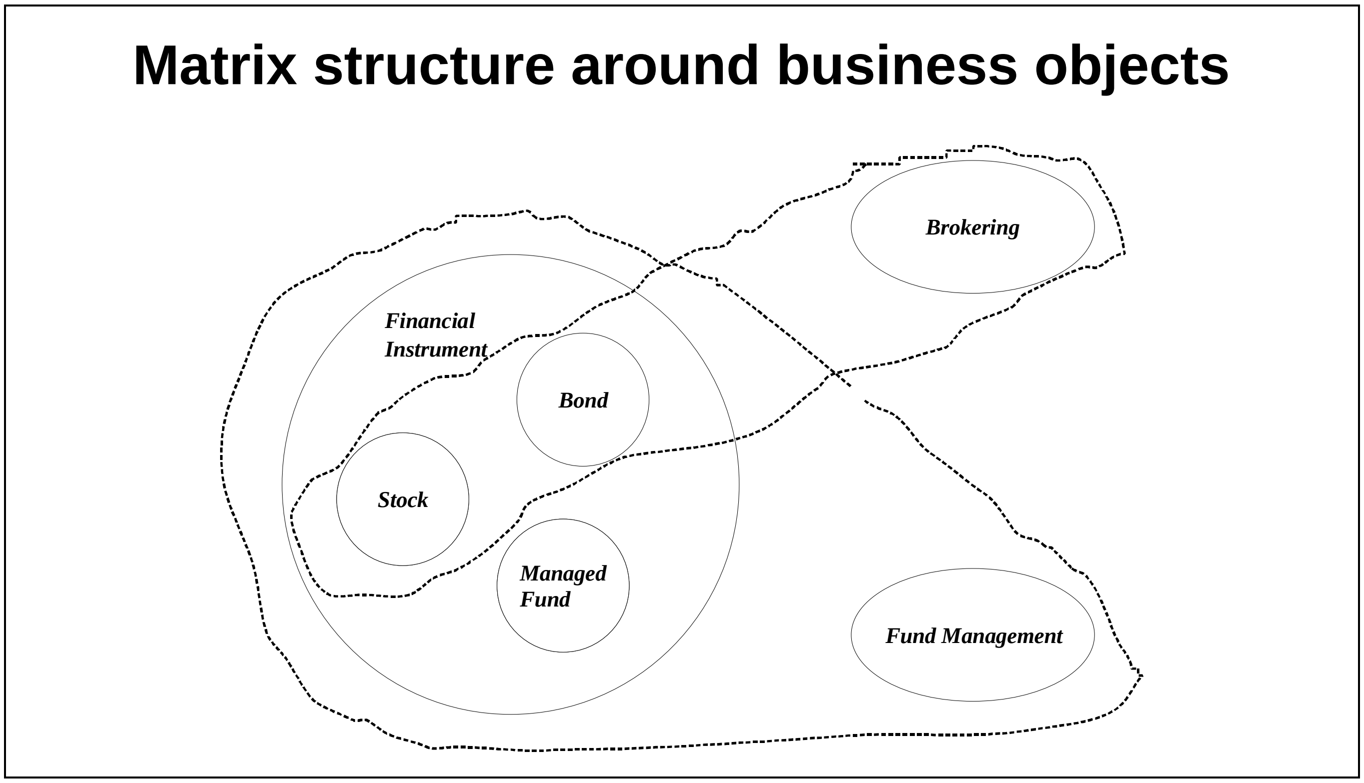

*Illustration 6. Teams would be structured to support the full life cycle of each business entity object and business process object. Sub-teams (like those for* financial instrument*) would be formed around specializations. Matrix reporting (shown as dashed boundaries) would be set up to negotiate between process teams and the teams responsible for the entity objects involved in the process.*

Business process object teams would overlap (or matrix, to use management jargon) the business entity object teams. Process teams would be responsible for BPR activities conducted with the line of business groups. In such activities they would engage the appropriate business entity object teams to represent and maintain their respective components. The business process teams would negotiate required changes with the various business entity object teams, and would be responsible for their own business process objects. From the list above, business process object teams would be formed around *fund management*, broker*ing*, *lending*, *collection* and *disbursement*. (See Illus. 6) Process teams could also matrix other process teams where one process requires another as a sub-process, for example *fund management* may require a *brokering* process, and *brokering* in turn may require *collection* and *disbursement* of funds.

## Conclusions

Business objects are powerful because they package in a sharable unit many parts of our understanding of a business and its processes that heretofore have been independent and free-floating: definitions of key concepts, the data kept about those concepts, the processes that work on those concepts and data, and the business rules and integrity constraints which govern process, data and concept membership. Business objects provide the packages for modeling and implementing one single logical source for data and procedures about each essential business concept in a way that guarantees consistency across applications, departments, the enterprise and even multiple related enterprises. Business entity object and their assembly into processes provide a way to make business systems mirror the business itself by allowing new systems to be built in terms of these objects and their interactions.

Business object frameworks offer a way to package business objects and business processes. Frameworks and patterns of business objects represent the next step in the developing field of business objects. Frameworks and patterns will force significant changes in the way we approach BPR, business application development and business modeling with objects.

Business object frameworks also suggest changes in the I.S. business process. For example, groups which currently manage data would manage data, procedure and constraints. These groups would manage the specialization and operation of a business object framework, instead of only managing data models and databases.

Robert E. Shelton is President and Chief Technologist of Open Engineering, Inc., a San Francisco-based consulting firm specializing in object technology and business engineering. Mr. Shelton is Chairman of the Object Management Group's Business Object Management Special Interest Group (BOMSIG). He is Editor and Columnist for Data Management Review, and past Editor of SIGS Publications Hotline on Object Oriented Technology. If you have questions or comments, he may be reached at 415-989-9050, fax at 415-989-9055, or email at rshelton@openeng.com.

Next month's *Distributed Computing Monitor* will address
**Object Databases in Distributed Environments.**

For reprint information on articles appearing in this issue,
please contact Donald Baillargeon at (617) 742-5200, extension 117.