 Published in Nikkei Computer, Nov. 30, 1995

Business Objects

By Robert E. Shelton
President & CEO
Open Engineering Inc.

50 California Street, Suite 860
San Francisco, California 94111
(415) 989-9050 • www.openeng.com

Business objects provide a powerful mechanism for dynamic business modeling and re-engineering. Business objects can be used for packaging shared business policy, process, data and definitions. In addition, business objects help to manage the architectural complexity of distributed object and three-tier client server computing. Instead of limiting a business to sharing data only, business objects allow sharing of process, policy and data. Using objects to understand and re-engineer business processes can be a breakthrough for companies that want to succeed in this complex and on-going endeavor. This article introduces the fundamental concepts and uses of business objects, and outlines the OOBE method for modeling and designing business objects.

What is an Object?

To understand business objects, first we must establish what is intended by the term *object*. An object is a building block that represents a real-world (business) or computer (technology) concept as a "black box". An object packages the ***data*** (attributes or variables) together with the ***procedures*** (methods or services) and semantic ***constraints*** (rules and policy). All data, procedures and constraints (collectively, these are called *properties*) that are part of an object must be related to the central concept that the object represents. Everything an object knows is represented in its data. Everything an object does is accomplished by its methods.

An object's data are only accessible to the outside world through the object's methods. The outside world does not know how the data or methods themselves are implemented (this is called *encapsulation*). In short, an object presents a clearly-defined interface or protocol by which outsiders may use its services. To get work done by an object, one must know only how to identify it, what to ask for, and how to phrase the service request. An object can obtain services from other objects, thereby extending its own knowledge and range of services. An object can present different facets of its interface in different situations, making available only those properties relevant to the situation in which it is participating.

These characteristics make 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 view instead of by units of programming. Objects are not just software building blocks. They can also take the form of design or analysis abstractions, or a natural-language description -- for example, in describing business processes. Thus objects are a useful alternative to viewing the business world in computer program terms. Objects allow us to see both the business and its underlying information systems in business terms.

Three Categories of Objects

There are three categories of objects: *business objects, technology objects* and *application objects.* An object lives in one and only one category. Different issues must be addressed in identifying and developing objects in each category. Thus, a different development or acquisition process must be in place for each category. The power of this approach is that the Information Systems organization can be

re-engineered to practice just-in-time delivery of objects that are required for specific business or application requirements. The traditional application development paradigm can be replaced with flexible software manufacturing.

Business objects are objects that represent a person, place, thing or concept in the business domain. (See Figure 1) They package business procedures, policy and controls around business data. Business objects serve as a storage place for business policy and data, holding together in a coherent unit the right business policy with the right data and ensuring that data is only used in a manner semantically consistent with business intent. We call this *semantic normalization*: putting the right data with the right procedures in the right place. Common examples include: purchase order, customer, product, invoice, payment, stock swap, flight segment and vehicle.

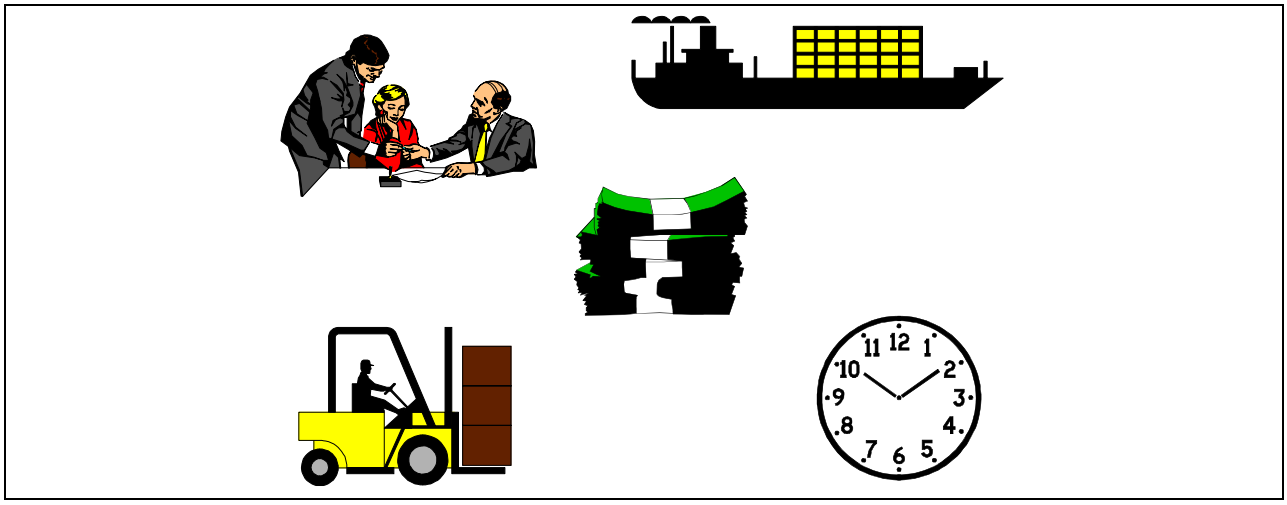

Figure 1. Business objects are the people, places, things, concepts and events which make up our every-day business environment. Examples include vehicles, products, bills, employees, customers, and temporal events like a quarterly earnings period or annual tax cycle.

The OMG Business Object Management Special Interest Group (BOMSIG) has standardized the definition of a business object in a way that usefully summarizes our discussion to this point: "A business object is a representation of a thing active in the business domain, including at least its business name and definition, attributes, behavior, relationships and constraints. A business object may represent, for example, a person, place or concept. The representation may be in a natural language, a modeling language, or a programming language." Open Engineering founded and has been active in OMG BOMSIG to develop industry-wide standard terminology for business objects. The work and terms discussed in this article are, however, those of Open Engineering unless otherwise noted.

Technology objects represent a programming or technology concept, and thus are the building blocks of applications and implemented business objects. They are the components of the information systems and applications environment. Examples of technology objects include GUI components like windows and push-buttons, programming constructs like integer and string classes, object request broker services, databases, base data structure classes and application frameworks. These are (or should be) off-the-shelf technical assets that are selected for their ability to leverage the I.S. designer and developer both in program development and maintenance. Technology objects should be selected as well for conformance to central industry standards, such as OMG Common Object Request Broker Architecture (CORBA), so that third-party products can be integrated intelligently, and eventually interoperate. By standardizing on interface definitions for components of the technology infrastructure, OMG is creating what they call “interoperability out of the box” -- true open systems based on a component architecture.

Application objects are programs which present information and manage interaction with human users, process information, and produce reports. They are solutions to specific business problems, and take the form of control panels which operate a specific set of business objects to perform a specific task. Examples include: Order Entry, Quarterly Report, Reservation and Ticketing, and New Account Setup applications.

Physically, application objects are assembled from business objects and technology objects, glued together with program code as needed to perform the application's specific task. Using client-server terms, application objects can be viewed as clients of business objects.

Entity, Event and Process Business Objects

Business objects actually come in three varieties: *business entity objects*, *business event objects*, and *business process objects*. This further categorization of business objects will help us implement practical solutions to business modeling, BPR and business systems development.

Business entity objects are what usually come to mind in a discussion of business objects. They represent people, places and things, in much the same manner as a data-modeling entity. Key differences between a data entity (that one might find on an ER diagram) and a business entity object include:

- The object packages procedure and rules that are specific to the concept being represented, while the data entity packages only data; and
- 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 business noun, though it can also represent an intangible concept (examples: *employee*, *employer* and the concept of *employment*). The instances of a business entity object are packages of data or facts about the business noun. (See Figure 2)

Business event objects represent business events, such as business time boundaries (end of quarter or fiscal year, elections, etc.), changes in the business environment, product lifecycles, etc. Many business event objects represent boundaries in time, while others recognize that some significant action has taken place. Business event objects seem similar to business entity objects, in that they are a repository for business information and rules about the event; however they primarily serve as an actor to initiate business activity.

Businesses retain data about events, expend much effort trying to trigger (or avoid) events, and manage and measure performance against events. A business event object will have a name and definition, facts kept about it, and procedures and constraints associated with it - just like any business object. Event objects, however, occupy an important place in a business object model: they are found at the initiation and termination of interactions between business entity objects.

Events which trigger an interaction (including those which trigger a business process) are found at the initiation of an interaction between two business entity objects. One or more distinct business events can initiate any given interaction. One or more distinct business events can also result from an interaction between two business entity objects. Part of business modeling with Object-Oriented Business Engineering is determining which events trigger and result from which interactions.

Business process objects represent business verbs. They represent business processes (as opposed to procedures, a distinction to be explained below), where a process is characterized by the interaction of a set of business objects. Business objects 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 Figure 3) The business entity objects package the policy and controls that determine how the process is carried out. Business process objects package the “how” into an object.

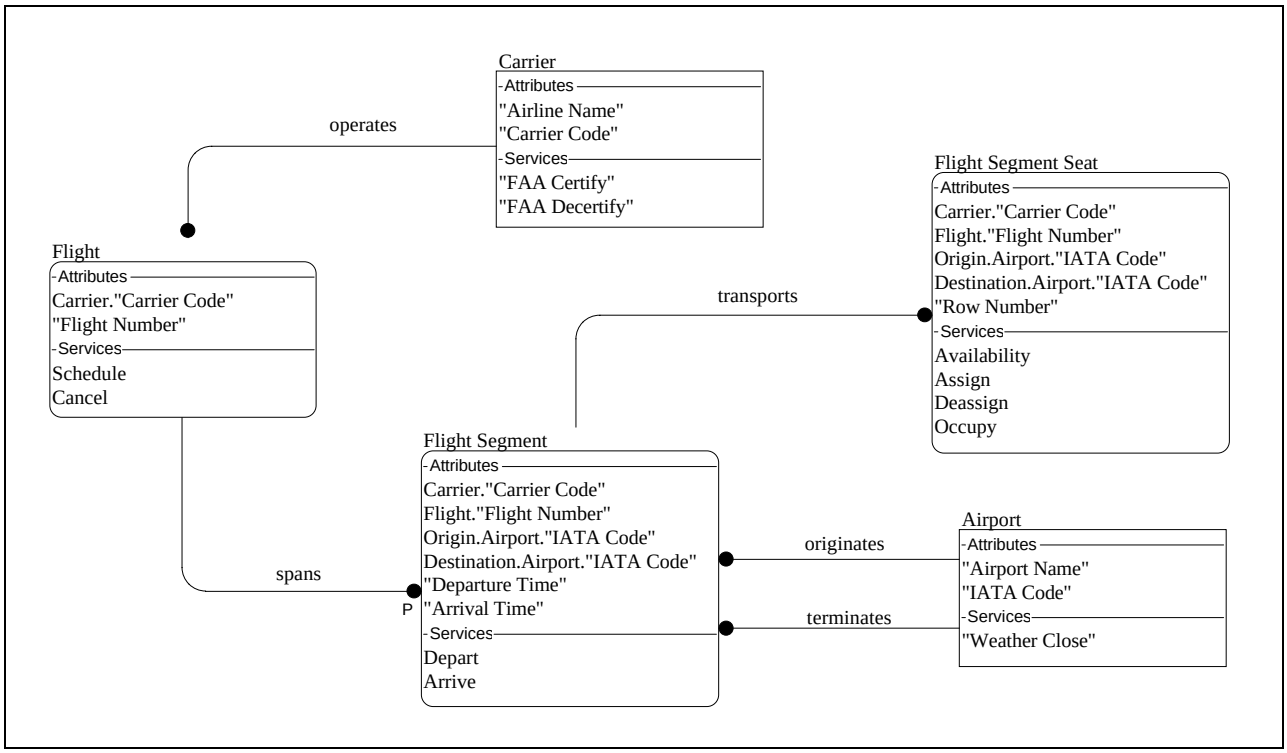

Figure 2. Business entity objects with selected services and attributes displayed. This diagram is in OOBE™ Object Relationship Diagram notation.

For our discussion, the term ***procedure*** references a single behavior (i.e. service or method) of a business object. A procedure is behavior specific to a single business object in much the same way that a properly-normalized attribute is a fact about a single entity. A procedure may delegate part or all of its work by requesting the services of other business objects. However, the action or result of the procedure is specific to the business object that "owns" it. A procedure is never instantiated - that is, we never keep attributes about a procedure. It is simply a bit of business logic or program code that operates on the instance (the data) of a business object.

In contrast, a ***process*** is an action that always involves multiple actors. Its instances represent in-progress units of work, not facts about a business person, place or thing. Its attributes are references to the business entity objects that carry out the process. Its services (procedures) start, stop, step and status units of work. The business process object is similar in concept to the workflow "program", as it organizes and controls the work steps that are required to complete the unit of work. An individual transaction in the workflow environment is similar to the instance of a business process object: both represent a defined unit of work being performed.

The term “instance” deserves clarification. A particular business object type (or class) is ***instantiated*** when it directly represents concrete concepts in the business world. That is, ***instances*** (rows or records in database terminology) can be created for the class. Instances of a *business entity object* represent the data values retained about specific things in the real world -- for example, customers, products, orders, contracts, and suppliers. Each individual customer, for example, would be represented by an instance of the customer *business entity object*. Instances of a *business event object* would represent individual occurrences of an event in the business world -- for example, the hiring of a particular class of freshmen,

the close of a particular fiscal period, and so on. Instances of a *business process object* represent the initiation of a particular business process to deliver a business result -- for example, the process which is initiated to fulfill a customer's order for product or the process of hiring a new employee.

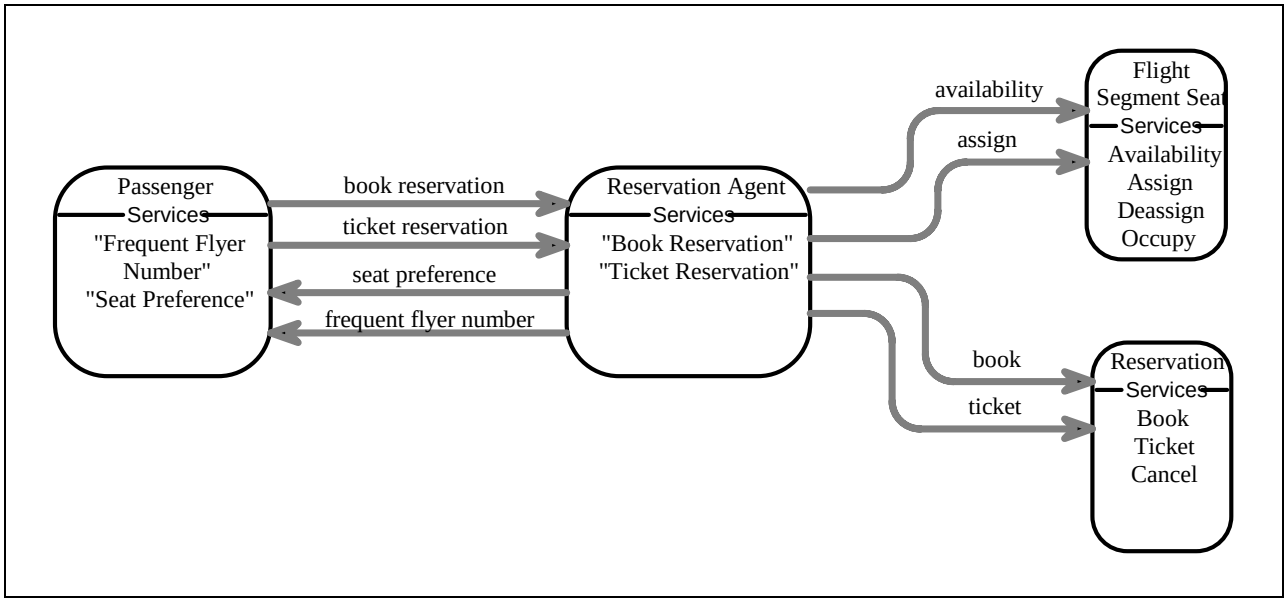

Figure 3. Interactions between business entity objects that comprise the work steps of a business process object are shown in OOBE™ Object Behavior Diagram notation.

A Method for Discovering and Designing Business Objects

Open Engineering developed and has been improving upon Object-Oriented Business Engineering since 1990. OOBE is a framework and discipline for business modeling/re-engineering, business object design, client-server application development and architecting enterprise-distributed infrastructure. In this article, we will examine elements of OOBE that are specific to business modeling and re-engineering with business entity, process and event objects.

One of the driving values behind OOBE is that information systems can and should be assembled from components. Thus OOBE is comprised of a set of processes each of which is tuned to deliver a particular type of component within a structure which makes component assembly possible. Here, one of the operative words is *deliver*: components can be bought, built or specialized. This is the essence of re-use. To describe OOBE in non-computer terms, consider it to be *manufacturing process control for business and systems components* - a way to achieve flexible manufacturing of the business and its systems.

OOBE builds on the notion that there are three categories of objects because the process of delivering systems to support the business (especially one which has undergone business process re-engineering) must address the business, the information technology which will support the business, and the applications which will provide specific information- and process-related services to the business. OOBE is characterized by development of application-independent business object models, and the realization of these models in software components. The role of application development thus shifts to specializing the resulting components to a particular task. Immediately, one can identify fundamental differences here between analysis and design methods and OOBE: (1) the processes create and maintain a linkage from strategy to software, so the software components reflect the strategy; (2) there are multiple processes, so that business and technology are actually considered and developed separately from applications; software is (or more precisely, *can be*) software delivered from each process, meaning that software

business objects can be shared by multiple applications, and can operate on multiple technology platforms.

OOBE applies basic patterns derived from architectural engineering to the structure of data processing and business engineering. Our thinking in this area proceeded from the determination that business process, event and entity objects were required to describe a business. Furthermore, the organization structure and distribution characteristics of an enterprise had to be mapped against the business model to fully describe the enterprise. While all three (business, organization and location) models can technically be handled as one business model, their orderly separation makes distributed business and workflow much clearer. (See Figure 4)

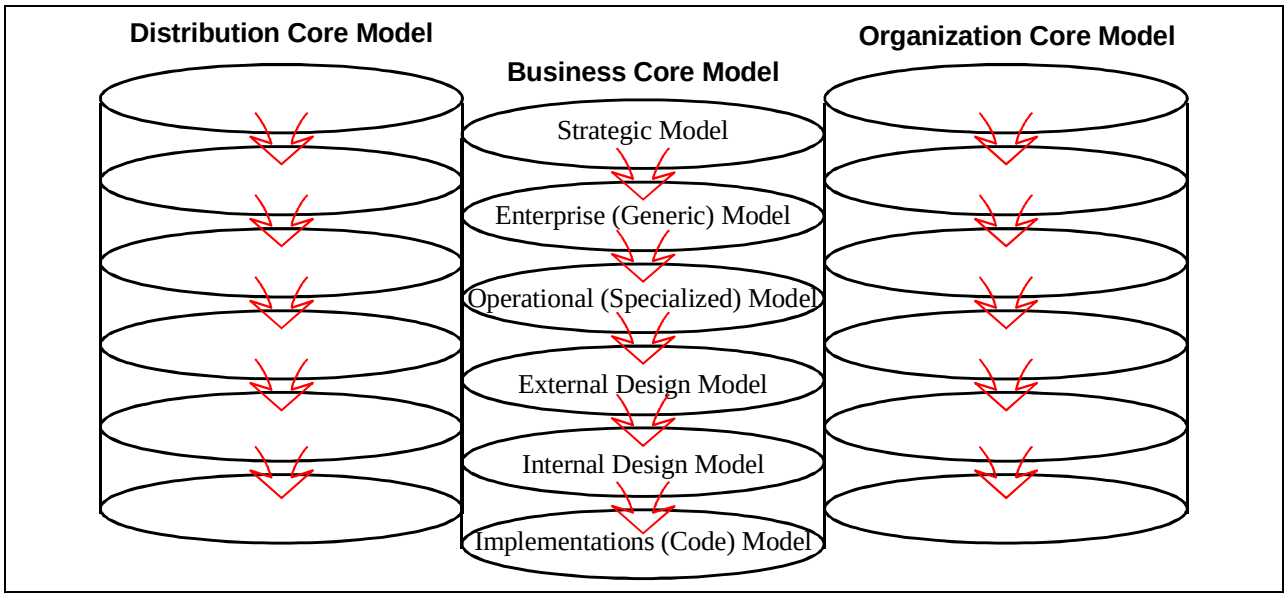

Figure 4. OOBE is build around three core models which represent the business processes and structure, the organization structure and authority, and the location of business components relative to one another..

To be comprehensive, a business model must span from strategic thinking about the business to the implementation of software components which represent the business. Thus, OOBE is based on the idea which we often call “concept to code”: the business representations can be transformed into systems models to run (and track changes in) the business itself. Each “level” in a core model represents a transformation from concept to code.

OOBE is not a “waterfall” approach. Experience with object and non-object methods alike has supplied much evidence that a waterfall approach creates categorical defects in the resulting products (the product is always out of sync with requirements and characteristically slow to adapt to requirements change). OOBE thus advocates an ***iterative process*** applied to each domain, and accomplishes this by scoping tasks within each level. Furthermore, OOBE supports a “re-entrant” strategy -- on may enter the transformations at any one of the top 3 levels. This is called a “top half down” approach, and emphasizes our commitment to understanding the business before developing software to support the business.

The ***Strategic*** level represents the goals, objectives, vision and strategies of the business, what it seeks to achieve with technology, and how applications will leverage key business processes. The ***Enterprise*** level represents best-of-class, generic business solutions for each domain. The enterprise business model, for example, contains best-of-class business components, processes, relationship type and role structures, process-tuned organization templates, and logistics models. At enterprise level, best-of-class can be defined within a specific industry, or across industries. Again, fundamental differences from traditional

methods: (1) enterprise means *generic*, but not necessarily abstract or "high level", and (2) OOBE presumes that an organization will import best-of-class solutions for re-use and specialization, instead of inventing everything from scratch. Many consulting and business professionals have come to realize that business concepts and structures are about 80% common across industries, and up to 95% common between businesses in a given industry. The ***Operational*** level is used to specialize the chosen enterprise model to the needs and strategy of a particular organization. Because it is company-specific, an operational model defines the "unique 5%" - that is, the parts of the organization that are different from the generic in order to create a unique competitive advantage. In very large, complex organizations (generally conglomerates or holding companies), there can even be more than one operational model - by necessity, as different parts of the organization can be substantively different from others.

The ***External*** level is used to design the external interface (or protocol) of each component, and (when used for object-oriented software development) the class hierarchy which will support specialization of business objects to the needs of particular technologies or applications. As with the subsequent levels, External models are environment specific, reflecting such technology interface standards as the OMG's CORBA, Workflow Coalition's interface definitions, or a particular language. The ***Internal*** level provides for design of the internal structure and workings of each object. The internal business model, for example, supports design diagrams and tasks for the following: object data and method structures, storage keying and indexing, data and process distribution mapping, data and process mapping to legacy data structures and on-line transactions, and the mapping of business rules and integrity constraints onto attributes and methods. The ***Implementation*** level is, of course, the level of generated or written code based on the designs, which were based on the customized model, which was based on the generic model, which was selected based on strategy. Traceability is created, end-to-end.

In order to represent the Business Core Model, OOBE uses four integrated diagrams (See Figure 5):

- Business Process Diagram (BPD)
- Object Behavior Diagram (OBD)
- Object Relationship Diagram (ORD)
- Object State Diagram (OSD)

The BPD is used to model high-level business processes, usually between 2-5 levels of depth. This diagram provides scope (in business process terms) for pairs of OBD-ORD diagrams, which are used to represent the structure of the business process itself. OOBE emphasizes using existing, industry standard constructs where possible. In this spirit, we use the IDEF0 style of process diagram. Therefore, existing process diagrams can often be "imported" into OOBE to provide the high-level process picture. What OOBE provides that process-only methods do not are (1) a way to transform processes into less abstract representations of the business (we have elected to use objects and interactions); (2) a method for translating these representations into systems.

The OBD is an object interaction style of diagram with the addition of business events at the start and end of every interaction between any two objects. Figure 3 is an example of an OBD. The OBD treats every business entity object as an actor capable of performing its "part" in a process. Each OBD sits "inside" at least one process on the BPD. Thus, the OBD shows how the process on the BPD is carried out. As with the BPD, the OBD represents business activities in business terms when used in the Strategic, Enterprise or Operational levels of the Business Core Model. When used in the lower levels, the OBD relates to design and component interactions. In business modeling, the OBD is an object-type model: it represents types or classes, not instances. For this reason, OOBE is more effective in identifying patterns of business activity than other methods based on instance diagrams or use cases.

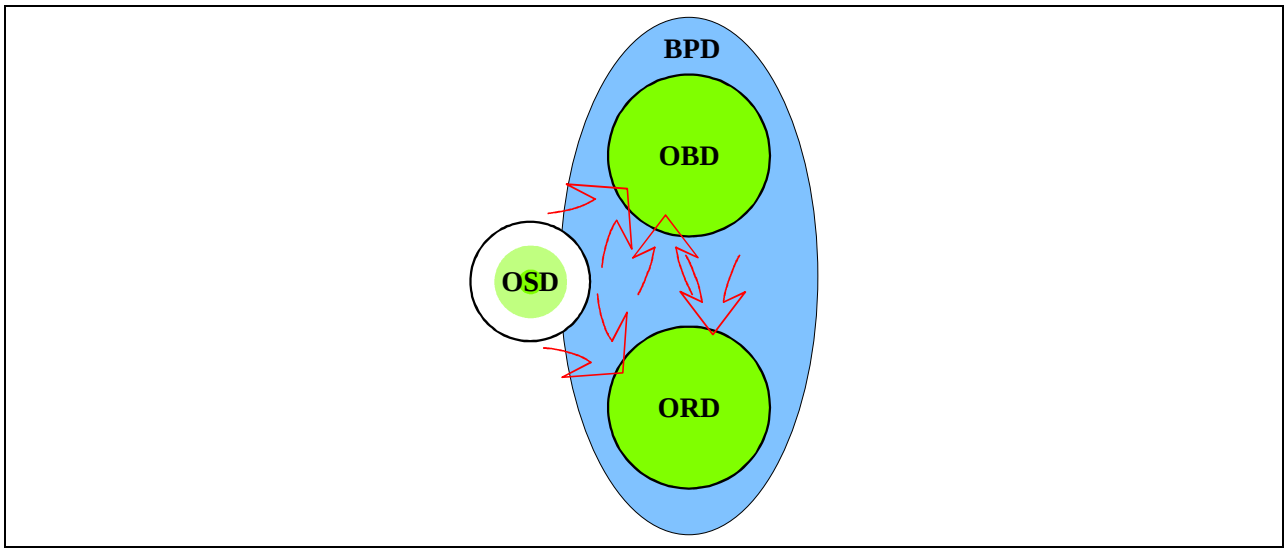

Figure 5. OOBE uses four diagrams to represent the Business Core Model: behavior, relationship, lifecycle and process. Note that an organization diagram is used for the Organization Core Model, and a map or schematic is commonly used to represent the Distribution Core Model.

Each OBD is paired with an ORD. Thus an ORD also sits "inside" a process on the BPD. In keeping with our philosophy on accepted industry standards, the OOBE ORD is based on the IDEF1X relationship model. Figure 2 (above) is an example of an ORD. We use the ORD to represent relationships between business entity objects. Also, the ORD represents such structural relationships as types (inheritance), roles (delegation), composition (aggregation), aliases (synonym/homonym resolutions and vocabulary conflicts) and populations (logical subsets of instances). The ORD is used to represent business semantics, structural business rules (such as cardinality in relationships), and to capture attributes associated with each business entity object. Each object on an OBD will be found on the corresponding "pair" ORD - thus we call these OBD-ORD pairs. This integration is another significant advance by OOBE: unlike many OO, information engineering and structured methods, we tightly integrate structure, relationships and behavior.

The OSD represents the lifecycle of each business entity object. This kind of model is also called state-transition or event-trace. We have elected to use the Harel statechart style of diagram, as it is both standard and easily understood. The OSD is used as a "cross-check" on the OBD, to validate that the object will perform as required in response to business events and the various interactions which occur in all of the processes in which a given business entity object participates.

Business Modeling with Business Objects

With previous approaches to business modeling and systems design, business models were static blueprints of business data and business function. Static models were primarily useful for defining corporate-wide data and designing shared databases. In contrast, business objects provide the ability to create business models out of active representations of the business world, not just static data entities. The components of a business object model are defined using concepts that business people work with every day, with a twist: the focus is on the whole business concept instead of separate data and program modules.

Consider an approach to business modeling and process reengineering that focuses on *actor* (the business entity objects), their interactions (tasks, activities, actions or responsibilities), and their context (the business process and business events). Instead of decomposing business processes into their sub-

processes, OOBE *composes* business processes from collections of interacting actors. The latter approach breaks the decompositional thinking paradigm by asking questions like "Given this new business process, who performs which processes, how do they do it, and how will the information systems support it?" In short, OOBE avoids building rigid, hierarchical representations of business processes and structure. This is a crucial step toward building more flexible information systems, and implementing continuous improvement in business process.

Recall that a business entity object represents a person, place, thing, event or concept in the business. It packages business procedures, policy and controls around data. A business process object is as a set of business entity objects that interact in a predictable, repeatable manner to produce a recognized business result. The business event objects are the triggers in this interaction, the business's way of tracking and scheduling its life. The business entity objects are the actors or role-players that carry out the business process. They package the policy and controls that determine how the process is to be carried out. And they do this in a way that their interactions mirror activities in the business -- that is, the business model translates directly into the actions and actors which comprise the business itself.

Business entity objects can mirror the business because they are *actors*. If a customer and a sales representative negotiate a sale, business objects representing the customer and the sales representative can be designed to support the essential elements of that business transaction. If the result of the sale is removal of product from inventory and the initiation of manufacturing to fill the order balance, business objects that represent inventory, product, various manufacturing roles and machines, shipper, package and invoice (among others) will become involved. The idea is intended to be straightforward: business objects can be used to model (simulate or mimic, if the reader prefers) what is happening in the real world of business. The business object "model" becomes a working analog of the business: the players, the processes and the events.

Reengineering the business then becomes an exercise in creative thinking applied to an actor-based simulation of the business. The business can be changed by disconnecting requests from one object and reconnecting them to another to form a more efficient process, bringing "missing" objects into the picture, and removing unproductive or non-value-adding objects and interactions. The business can be evaluated by testing process flow against business rules and constraints, and assessing the time/cost performance of new processes against existing ones. Further, the proposed new business can be communicated to others in the business by explaining which actors and actions will be needed to conduct the process.

Implementing Systems with Business Objects

Business objects simplify control and testing of business procedure change, database redesign for management or re-modeling, physical data distribution, and synchronization of distributed or multi-database transactions. Why do business objects effect this change? In each case, encapsulation or packaging is the key. Much of the business change we make to our systems would be contained within the business objects themselves, like changes to data structures and formats, data location, business rules, and integrity constraints. Likewise, much of the technical complexity of multi-database transactions, replication or redundancy created by departmental applications can be hidden behind business objects.

While business objects will not themselves fix problems in today's systems, the simple truth is that fixing problems from the viewpoint of the new client-server systems organizations are trying to design would be a major step forward - even a step toward the eventual solution of problems in the very systems we cannot fix today.

On the application development front, business objects simplify the implementation of three-tier client/server by providing the middle "business logic" layer in workable units. (See Figure 6) Instead of

having business logic imbedded in either the presentation layer or the back end (as required by two-tier tools), business objects provide a home for business logic that carries an important advantage over conventional approaches: business objects mirror the business instead of applications.

Finally, business objects make a significant contribution to the integration and migration of legacy systems. Business objects can wrap existing databases, transactions and application screens, thus concealing the underlying legacy implementation. New applications can be built on correct data and process models represented by the business objects. Migration can occur without changing new client applications, because the legacy systems are hidden behind the business objects. Business objects will not inherently correct problems in the legacy systems themselves, and integration requires a substantial knowledge of these systems, their definitions and interactions.

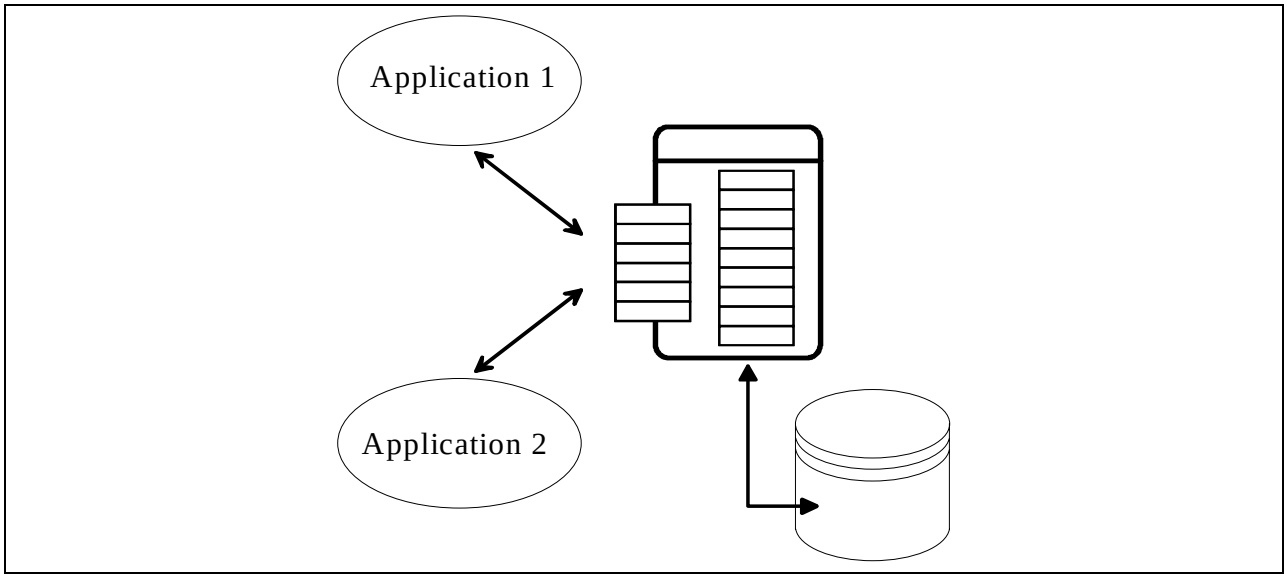

Figure 6. ***Business objects form the "middle tier" in three-tier client-server architecture, providing a logical modular packaging of business logic, rules and data between the front-end presentation layer and back-end storage management and legacy application systems.***

Where are Business Objects Used Today?

The concepts we have discussed in this article are in use today at organizations worldwide. In some organizations, business objects are principally used at the software level to integrate and migrate legacy applications and databases. In organizations with more mature software development and business engineering processes, business objects are also being used to model and rethink the business. Here are some brief examples.

In the banking and financial sector, distributed computing and business objects are frequently used to integrate applications, data feeds, databases, on-line information services and complex analytical data sources used by fund managers, stock and bond traders, and loan officers. Fidelity Investments, one of the largest fund managers in the world, has built a fund manager's desktop using business objects on the Digital ObjectBroker object request broker (ORB) middleware. Wells Fargo Bank, one of the 2 largest regional banks in the Western U.S., is using business objects in a variety of trading and front office systems projects. Many of the large banking and trading houses on Wall Street have, likewise, used business objects on distributed object middleware to build trading, front office and back office systems.

In manufacturing, Texas Instruments and Sematech have developed business objects to control the semiconductor manufacturing process. POSC has developed a business object model to capture and synchronize business concepts that are shared among 76 companies involved in petroleum exploration and extraction. One of the Big 3 U.S. auto manufacturers is using business objects and distributed object middleware in projects that range from automobile design to manufacturing.

In the field of insurance, Connecticut Mutual Life is integrating the idea of business objects into their architecture for future information systems. Another firm, the seventh largest property and casualty insurer in the U.S., is designing business objects into their next-generation architecture and into plans for an advance flexible software manufacturing organization. Another major insurance firm has developed a business object model for the insurance industry, and is evaluating offering the model for sale as a product.

Today, a growing number of companies offer business objects for sale in a variety of industries, including: process and discrete manufacturing, pharmaceuticals, mining, banking, human resources and accounting. Some products are code-level class libraries, while others are more abstract models. This emerging market reflects growing demand that is being created by the benefits that organizations achieve by using business objects.

The principle benefits we have experienced from business objects are:

- the ability to make business changes more quickly, and reflect those changes in systems,
- shorter time to market with new products/services, including time to produce systems that support and leverage the new processes,
- greater flexibility in delivered applications,
- business rules can be located in objects that make sense to business users, instead of locating related rules across many different applications,
- business process can be made visible and more controllable,
- business objects provide a single source for data, process and business rules, reducing redundancy and opportunity for errors,
- easier to design applications that share data, process and business rules.

Business objects are less about software and more about business modeling and design. They make it possible for the software to operate like the business - almost like a direct simulation of the actors and processes. This increases the accuracy with which the software does what the business needs -- therefore directly improving software quality and integrity. Fundamentally, business objects are not about tools or a new way of programming. They represent a much improved way of looking at (modeling, re-engineering) the business, and designing applications that directly support the business processes.

Conclusion

Business objects package many parts of our understanding of a business and its processes into sharable units that heretofore have been independent and free-floating:

- definitions of key concepts,
- the data kept about those concepts,
- the procedures that work on those concepts and data,
- the business rules and integrity constraints that govern process, data and concept membership, and
- business processes and events which govern business operations.

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 enterprises.

Business objects 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. This implies a substantial change in the traditional approach to business analysis and systems development, as business objects are independent of applications in both conception and implementation. Systems design with business objects involves building a working model of the business and its processes in terms of business objects, then building systems to operate that model.

Experience gained from applying Object-Oriented Business Engineering, reveals that the business objects truly mirror the business instead of the computer system. This direct representation of real business concepts breaks down one of the major roadblocks to implementing business process reengineering and improving today's systems. The strength of business objects extends beyond business process reengineering: designers can use business objects to "draw" models of the business during BPR endeavors, then transform this model into a design for implementable objects. Furthermore, developers can implement these business objects on mainframes, as mid-tier servers, or as load modules in a distributed or traditional computing environment.

Business objects are not magic, and major work is required to conceive, design and implement them. But a growing body of industry experience shows that a business object approach to business process reengineering and to information systems can make the challenges of reinventing the business more manageable. Business reengineering does not result in a final product: business processes must be designed for continuous change in a continuously changing business world. Business objects can provide the vehicle for synchronizing continuous improvement in both business processes and their underlying information systems. Progressive companies that recognize the business advantage of robust and adaptive business processes have already taken steps to fuse business process reengineering and object orientation. Those companies that wish to maintain leadership in their industry should place the fusion of business and technology high on their list of priorities.

Robert E. Shelton is President & CEO of Open Engineering, Inc., in San Francisco, California. Open Engineering is known for originating ideas such as business objects, business engineering with objects, and business object management.. Mr. Shelton is founding Chairman of the Object Management Group's Business Object Management Special Interest Group (OMG BOMSIG), and Object Technology Editor and Columnist for Data Management Review magazine. He has written articles for the Distributed Computing Monitor, Database Programming & Design, OMG's FirstClass, Client-Server Developer, and Object Magazine. His degree in Psychology and Computer Science is from Stanford University, California. Mr. Shelton's first book, Understanding Business Objects, will be available in 1997. Mr. Shelton will also edit a book series by Addison-Wesley, which will focus on business objects and related business and technical issues.