

# A Business Process Management Architecture

By the Staff at Management Strategies

*This paper proposes an architecture for describing, analyzing, designing, and modifying business processes. The architecture provides a conceptual framework for viewing business processes at the highest level. However, the architecture is not limited to a high-level or conceptual view of business. It maps with technical rigor into the detail of business process re-engineering and information engineering. Its role is to supply the missing executive view of these disciplines and to forge the link between the processes of concern to business and the detail of the systems that automate them.*

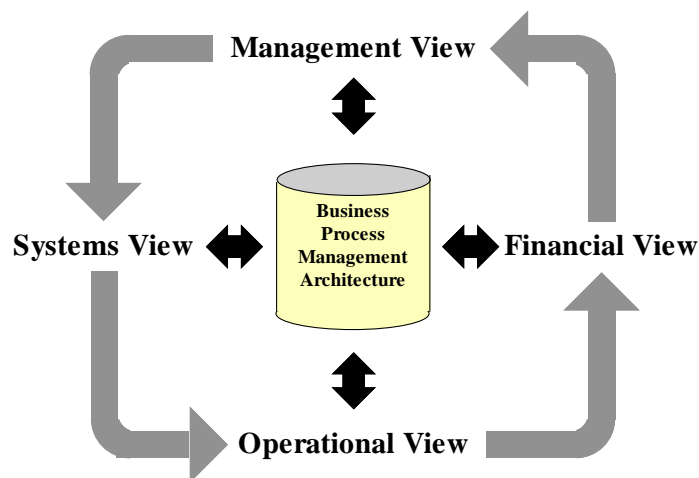
## The Need for a Business Process Management Architecture

If you undertake to build a house, your architect will start by asking you some very general questions about your budget, your family, your lifestyle, and your taste. He (or she) will then make some very rough sketches, perhaps literally on scrap paper, to find out whether you are communicating. When a sketch seems right for you, your architect will follow up with simple plans and illustrations. Gradually the design will emerge in a series even more detailed drawings. The architect will begin to introduce other specialists, first engineers, then contractors, who will produce even more detailed drawings and specifications. But when your house is finally built, it will probably match your original sketch in most particulars.

From your house to the space shuttle, designers everywhere work this way. The designers of information systems do not, and they stand practically alone in this respect. One reason for this is the lack of a good system for "sketching" business requirements at the highest level. Another is the fact that information systems, unlike houses, change frequently as they are built. The route from initial requirements to final application is not a one-way street. The planning system must have built-in "round trip" capabilities to synchronize requirements and design.

Synchronizing business requirements with system design is the essence of business process management. The cycle from the establishment of management requirement to the financial result can take years. The Business Process Management Architecture provides the core information about the requirements, systems, operations, and results that makes business process management possible.

**Figure 1. Business Process Management**

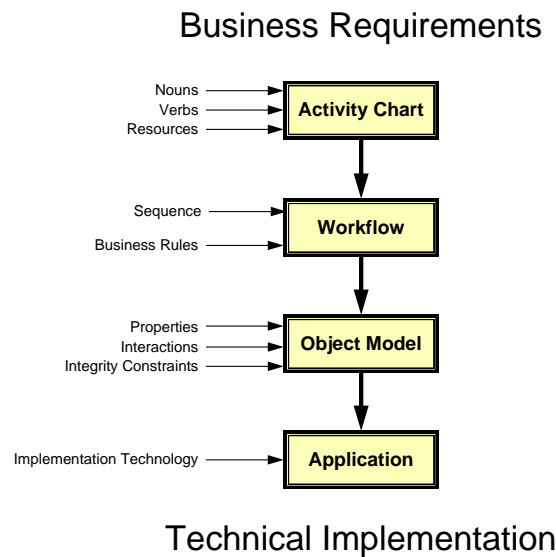


## Describing the Link

The link from business requirements to technical detail has several steps. For our purposes, it has four, each of which represents a concept that is in more or less common use today--by business people at the high end, and by application developers at the low end.

Each step is just a representation of business activity. As you move down the ladder towards technical detail, the steps acquire more and more definition. In other words, the link is forged by a chain of increasing descriptive detail or *semantics*. Here is a picture:

**Figure 2. Forging the Link**



This picture shows the four key links in the chain from business requirement to application code. The semantics that distinguish each link are shown (in non-technical terms) on the left. The dark arrows show the progression from the general semantics of business operation to technical semantics of application development, each link being richer in these semantics than its predecessor. Here is a short tour of the four links:

- The **Activity Chart** is the top-level view of the architecture. All business people understand readily what is meant by a business activity: Product Sales, Order Entry, Order Fulfillment, and so forth. We doubt that many ever bother to frame a formal definition. Our definition of business activity pairs a description of a business object ("Product" or "Order") with a description of a business function ("Sales", "Entry", or "Fulfillment"). Rather than worry about what a business object or business function really is, we simply call them "nouns" and "verbs"--to be fleshed out later. Activities can be very high level ("Customer Service") or more detailed parts of an activity hierarchy ("Return Authorization"), but they can never have sequence. If you think one activity follows another, you have gone too far for the activity chart. Sequential activities belong at the next level down.

Business activities can also be described in terms of the resources (cash, people, raw materials, etc.) they consume or create. The business activity is something that business people can account for.

- **Workflow** describes sequences of business processes. For example, the Order Entry business activity might be described by the following process sequence: Answer Phone, Enter Order, Validate Credit,

## A Business Process Management Architecture

Check Stock, Place Order. This is a high-level or executive view of business process, often called a *value chain*.

Each process in the value chain can be decomposed into many sub-processes. These sub-processes can be further decomposed without theoretical limit. Each process grouping in this hierarchy is a *scenario*, with the value chain being the top-level scenario. We use the term "scenario" as it is normally used in information engineering: *one particular path through a complex sequence of events*.

Describing workflow adds two factors missing from the activity chart: *sequence* and *business rules*. Sequence determines the order of processing. Business rules state the conditions that determine the path. For example, the exact path may depend on the evaluation of the condition "Is the order over \$1,000?" or "Does the customer have good credit?"

- The **object model** goes beyond workflow to focus on the objects affected by business activity. While a business can have workflow paths of great complexity, they can only work on a relatively small range of objects. The object model accounts for all of the nouns that we have used to define our various business processes and activities and gives each a rigorous definition. At this point, we are deep in the realm of business detail and knocking at the door of pure technology.

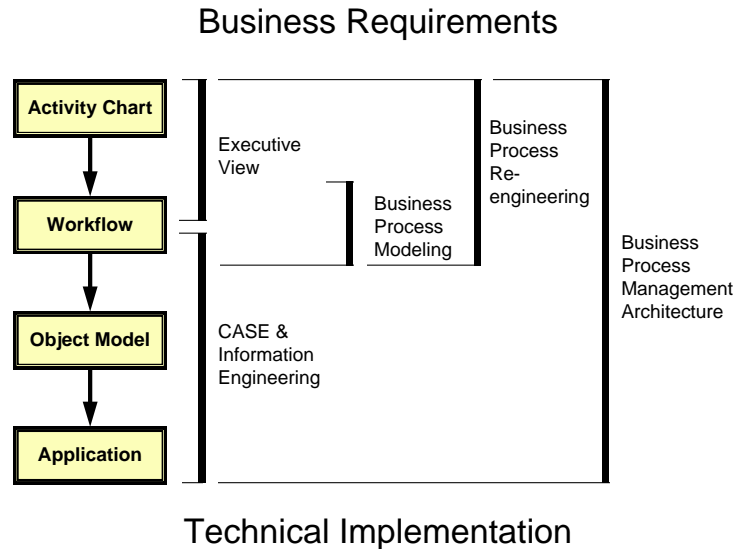
The object model adds three new semantics: *properties*, *integrity constraints*, and *interactions*. A property defines the variables that make each object unique--an Order object must have an OrderNumber, a Date, a Customer, and some LineItems. An integrity constraint (sometimes called a *data integrity constraint*) defines how business objects and their properties relate to one another--an Order object must have at least one LineItem or the Date must be no later than today's date. An interaction defines how the processes that define the object--that is the workflow process and the integrity constraints--work together. In other words, an interaction defines what information must be passed to a process to make it work.

- At the lowest level is the **application** or technical implementation. At this level we are free to add all of the machines, networks, operating systems, programming languages, and other technical specifics that comprise the physical installation of the system. Until this level, our architecture has been strictly business. Nothing in the first four levels has dictated whether an application has a host-centered, client/server, or three-tier technical architecture. Nothing has dictated the choice of programming language, user interface, or database management system. In fact, although we have been following the path of object-oriented design, nothing in the first three levels dictates an object-oriented implementation. This is a purely technical decision!

## The Link in Context

The Business Process Management Architecture is designed to provide the missing link between business requirements and the technical implementation. Since it is a link, it complements existing design methodologies and implementation technologies--it does not compete with them. To illustrate the point, we chart the Business Process Management Architecture against key complementary activities:

**Figure 3. Linking Key Activities**

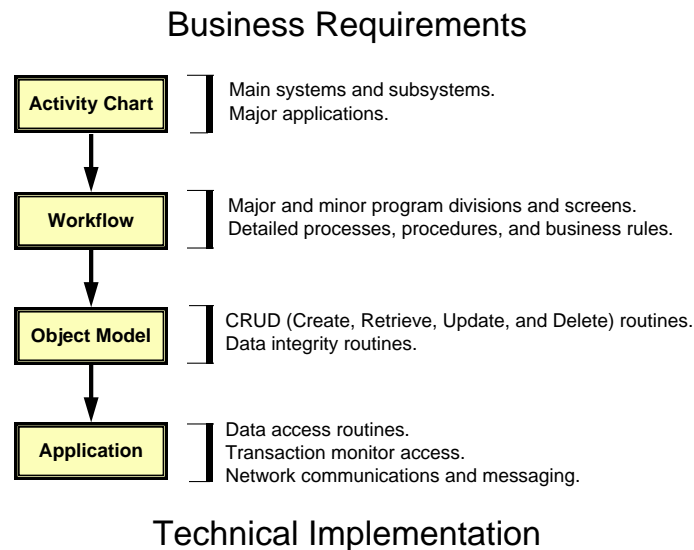


- **The Executive View** supplies the highest level view of business activity: the activity chart and the value chain. This view is missing from most technical approaches to computer automation.
- **CASE and Information Engineering** are the disciplines that the Executive View naturally complements. Usually, these disciplines cover the ground from scenarios (business process modeling) through object or data modeling to the finished application.
- **Business Process Modeling** methodologies like IDEF/0 focus on workflow. They are usually too detailed to communicate well to upper management.
- **Business Process Re-engineering**, in theory, covers the entire gamut from the highest level view of business activities down to the most minute process. In practice, BPR is limited by its tools and must rely on traditional process modeling and CASE technology.

## Systems Management

The journey from business requirements to working software applications is not a trip from the vague to the specific, but from executive concerns to detail. The choice of a business activities in the chart requires as much precision as the composition of a line of C. It is just as likely to be implemented in hard code as any other business requirement or business rule articulated as a design requirement. Just as a sketch of your kitchen on scrap paper leads to an actual built room in your house, an activity or process defined in the business process management architecture leads to a system, sub-system, program, or screen. In fact, the entire application design hierarchy can and should be based on these components. We can illustrate this hierarchy as follows:

**Figure 4. Isolating Application Components**



Each level in the hierarchy contains the information needed for a different level of the actual built application:

- **Major applications, systems, and subsystems** should map to the activity chart. In other words, the activity chart should provide a high-level overview of a company's computer systems.
- **Major program divisions** should map to the highest level of workflow: the value chain.. Often these will be key interfaces to external systems (like downloading point-of-sale data) or major data entry screens (like the order-entry screen).  
**Detailed workflow processes** of all sorts map to scenarios describing workflow detail. These can be lookups, validations, minor screens, reports, and business rules.
- **Routines to create, modify, and destroy data** map to the object model. These routines define objects in computer memory and check to make sure all values are valid.
- **Routines that relate purely to the implementation technology** map to the application level. These routines might define connections to the database, transaction monitor, network, or ORB.

The result of this approach is the antithesis of spaghetti code. It is a technical architecture that compartmentalizes functionality into isolated components. Changes to any component have minimal effect on other components. The application is designed to facilitate the continual refinement of business processes. It can grow and change organically in response to changing business conditions.

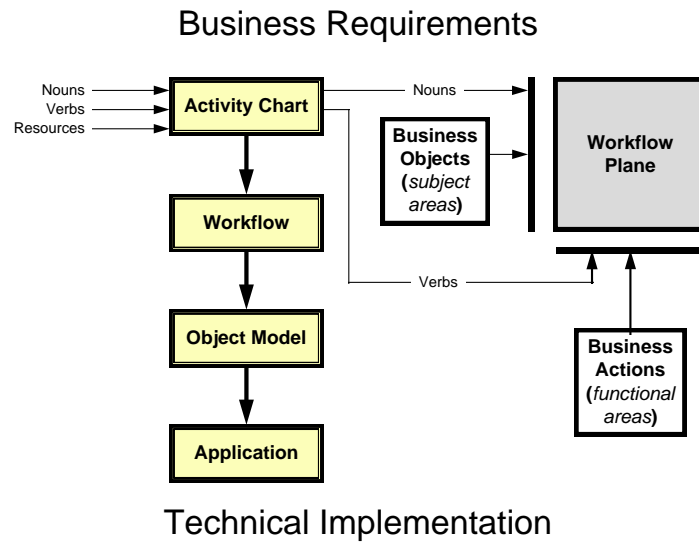
## The Activity Chart

The Activity Chart is the capstone of the entire application architecture. It provides the top view of the Business Process Management Architecture and the more detailed and technical layers that fall below. The Activity Chart depicts core business processes against the background of the workflow plane. The workflow plain is dimensioned by the nouns and verbs that describe the activities. Each dimension is segmented into subject areas and functional areas which constitute the organization.

## The Workflow Plane

*The explanatory power of the Activity Chart arises from a single key idea: separating the nouns and verbs from the description of business activities, then arraying them to form a two-dimensional plane. This plane is the **plane of workflow**. All business activity occurs within its boundaries. We illustrate this idea below:*

**Figure 5. Defining the Workflow Plane**



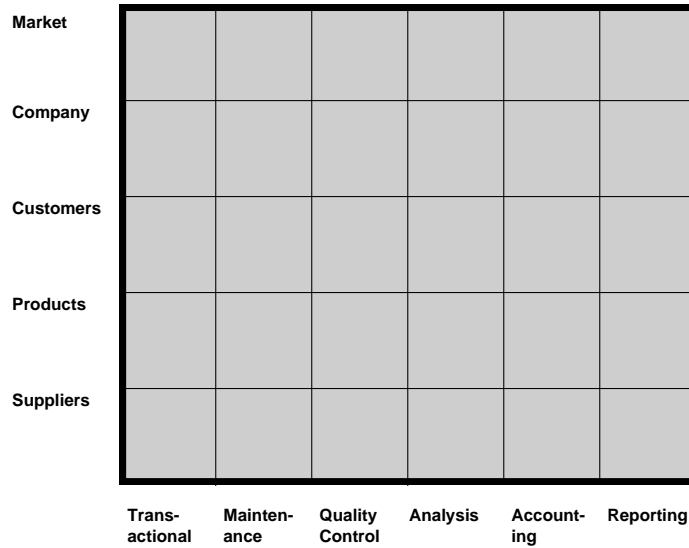
The workflow plane is defined by two dimensions, Business Actions (or functional areas) and Business Objects (or subject areas). We can define these dimensions as follows:

- **Functional Areas** are logical clusters of business actions that comprise value creation in an organization.
- **Subject Areas** are logical clusters of the business objects interactively managed by a company to create value.

## Segmenting the Dimensions

The next step is to segment each dimension of the workflow plane into groupings of business objects (or *subject areas*) and groupings of business actions (or *functional areas*). These groupings are not technical. Rather, they reflect the knowledge and operation of the business. They do not have to be invented or extracted from detailed knowledge. Instead, they exist naturally in the minds of executives, managers, and rank-and-file employees. It is our experience that a set of five or six key or top-level segments can be found for every business. Here is a segmentation that might fit a typical manufacturing business:

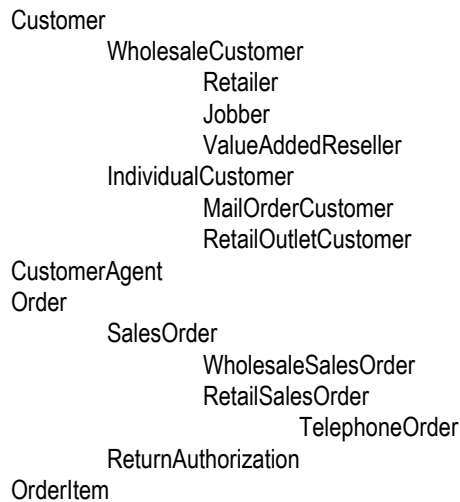
**Figure 6. Segmenting the Workflow Plane**



**Business Objects**

The vertical segmentation of subject areas represents clusters of business objects (to be precise, clusters of business object *classes*). As a rule, we have named each segment after its key object classes: Company, Customers, etc. These clusters should not be confused with the objects (or classes) themselves. They are just groupings of classes that make sense to business users. For example, the Customers segment may or may not contain a class called Customer or Customers. It is very likely to have an array of customer subclasses like Wholesaler, Retailer, or Reseller, or MailOrderCustomer. It is also likely to have customer-directed classes like Order, LineItem, and CustomerRepresentative. However, *within* each segment, the objects form a standard class hierarchy and form a natural sub-segmentation. Here is a class hierarchy that might represent the Customers segment:

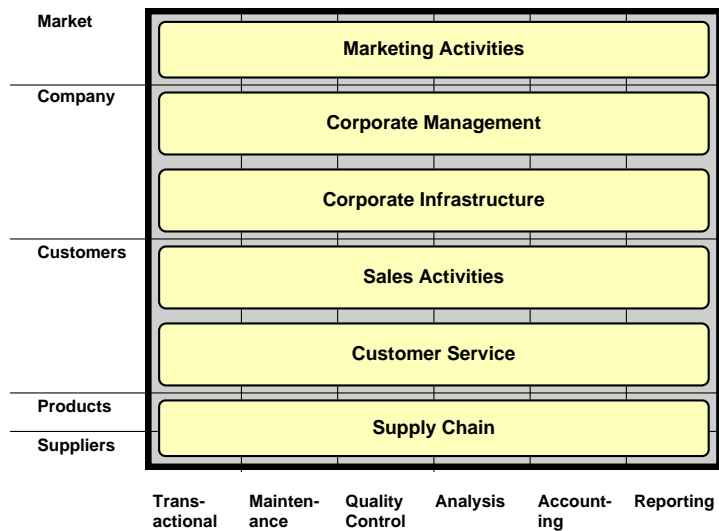
**Figure 7. The Classes of the Customer Segment**



## Charting Major Business Activities

Most major corporate activities--core business processes--will map closely to the highest level of the activity chart. The exact mapping can vary from one company to another. Below we show the major activities, as defined by a Fortune 100 manufacturing company. Each activity is defined as a broad horizontal band comprising all functional areas. Only Marketing Activities exactly fits the confines of a single subject area (Market). Corporate activity (Company) is shared between Corporate Management and Corporate Infrastructure. Customer-oriented activity (Customers) is shared by Sales Activities and Customer Service. On the other hand, the Products and Suppliers subject areas are subsumed by a single activity--Supply Chain.

**Figure 8. Charting Major Business Activities**

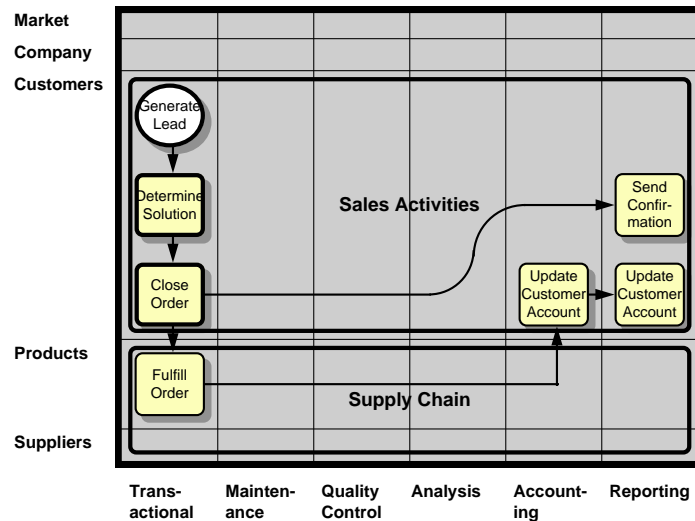


## Value Chains

A value chain represents a high level description of a major business process. A value chain usually focuses on a single major business activity, although it can trigger processes in other activities as well. It usually represents an activity initiated by a single actor in a single role (for example, a buying customer). The value chain can and should be broken down into its constituent processes.

Below, we illustrate the Selling Process value chain. The Selling Process falls within the area of Sales Activities. It begins with a generated lead, then continues to the determination of the sales solution and ends with the closed order. Closing the order generates two main activities, sending the customer an order confirmation, and order fulfillment. Shipping the order results in updating the customer's account and sending an invoice.

**Figure 9. Value Chain: The Selling Process**



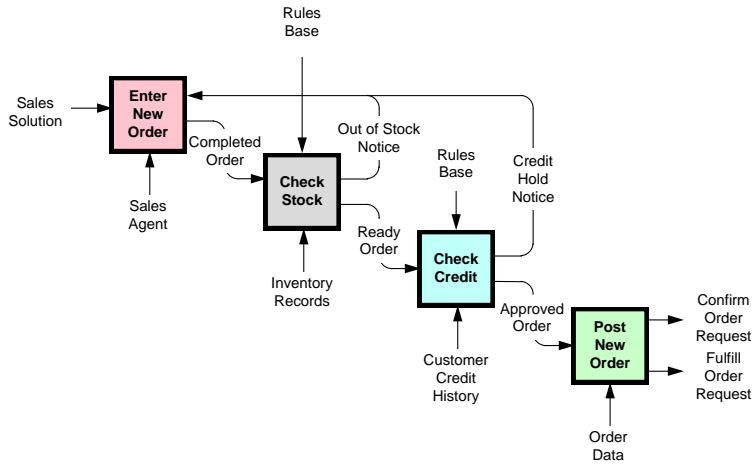
## Scenarios

Each process in the value chain is made up of detailed sub-processes. Each sub-process, in turn, can be decomposed into even more detailed sub-processes if needed. As we move from high-level processes to detailed processes, we abandon the term "value chain" in favor of the more general concept of "scenario".

Scenarios describe workflow--they do not describe how data is stored and retrieved from the database, how data integrity is maintained, or how the technical application interacts with networks, ORBs, transaction monitors or other aspects of the technical implementation.

At the same time, scenarios are usually confined to small areas of business activity. Unlike value chains, there is little reason to display scenarios within the object/action plane of the activity chart. Instead, the user should pick among any of a number of common process modeling tools to illustrate scenarios. Here is the Close Order scenario, illustrated in a modified IDEF/0:

Figure 10. Scenario: Close Order



## Workflow Rules

Note that two of the processes shown here, CheckStock and CheckCredit, are actually business rules. They work by evaluating conditional statements then branching back to the EnterOrder process. This kind of conditional branching is not part of traditional IDEF/0, but is possible in modified versions.

When rules are found in scenarios, they usually take the form:

**ON event IF condition THEN action1 ELSE action2**

In our example, the CheckCredit rule is triggered by the ReadyOrder event, itself an outcome of the CheckStock rule. It evaluates a conditional statement, perhaps comparing the order total with the customer's credit limit and outstanding balance. If credit is good, the rule triggers the posting of the order. Otherwise, the rule sends an error message back to the EnterOrder process, perhaps to reduce the amount of the order.

Such simple rules are often called *ECA* or *Event-Condition-Action* rules. Large applications can contain thousands of these simple rules. Since these rules govern critical business processes, it is important to handle them in a way that makes them easy for the business user to understand and easy for the technologist to change. Failure to do so results in code that is unnecessarily hard to manage.

More complicated rules, like the CheckStock rule which works on each line item, are not ECA rules, but they can call on ECA rules as part of their logic.

## The Object Model

Remember that every activity in the activity chart was associated with a noun and a verb. By implication, every process in a value chain, and every detailed process, sub-process, and business rule in a scenario is associated with a noun and verb as well.

These nouns are actually the names of objects--to be precise *business objects*. However, the activity chart, value chains, and scenarios are not sufficient to fully define a business object, so we have treated this association informally so far. We are now ready to supply the missing semantics to define business objects and complete the non-technical part of the architecture.

## Business Objects

In object-oriented design, there is no such thing as data--only objects. Everything is an object. But some objects are more important than others. For business applications, the objects that count are called *business objects*. Business objects represent things that are important to managers--things like Customers, Employees, Orders, and Products, plus all of the more detailed objects that are needed to fully describe the business: WholesaleCustomers, LineItems (of Orders) Departments, Products, Components, and so on. All told, it takes a few hundred of these objects to run a large business. Business objects add three new semantics to our description of the business. They are *properties*, *integrity constraints*, and *interactions*.

## Properties

Properties define business objects. Different values for the FirstName and LastName properties make John Doe different from Richard Roe. At the same time, the properties themselves are what make the Employee class (containing John Doe, Richard Roe, and other properties) different from the Customer class (containing CompanyName, CreditLimit, and its own set of properties).

Business objects almost always have numerous properties. This makes them different from other, simpler, kinds of objects like Description (only a text field) or Price (only a decimal number) or OrderNumber (a text field with a few special rules for composing its value).

One business object can have another as its property. For example, an Order will have a Customer and a set of LineItems (each a business objects in its own right) as properties. These kinds of properties are called *relationships* (and are precisely what the Entity-Relationship model means by the term). Other, simpler properties like OrderDate and OrderNumber are called *attributes* (once again, following the E-R model)

## Integrity Constraints

Integrity constraints are what are often known as data integrity constraints. We are reticent to use the full term because the word "data" can be confusing in the realm of object oriented analysis. These constraints define relationships that must hold to be true among business objects and their properties. Here are some examples of integrity constraints:

- An Order must have one and only one Customer.
- An Order must have at least one LineItem.
- An Order must have an OrderDate.
- The OrderDate must be prior to today's date, but not more than one week earlier.
- An Order must have an OrderNumber.

## A Business Process Management Architecture

- The OrderNumber is assigned automatically.
- The OrderNumber consists of one alpha, representing the division taking the order, two digits representing the year, and four digits assigned sequentially.

These constraints are actually another kind of business rule. Unlike workflow rules used in describing scenarios, constraints do not directly describe branches or loops. Instead they define statements of truth (equalities or inequalities) that the system must enforce. As it happens, computers are not very good at handling equalities and inequalities. They are much better at branching and looping. This means that, before constraints can be executed, they must be translated into processes.

### Interactions

If every process has an object, it must follow that every object has one or more processes. In fact, every object (business object or otherwise) has several processes. In object terms, the processes that belong to an object are its *methods*. Since processes have sequence, one method must be able to initiate another. To do this, a method must:

- a) be able to call another method and
- b) be able to tell it what to do.

When a method calls another method, an event happens. When a method tells another method what to do, it sends what in the language of objects is called a *message*. A message is really just a list of parameters to be decoded by the receiving method.

### Design Patterns

The interactions among methods, including the workflow processes from the highest level down to the most detailed business rule, the more detailed logic that enforces constraints, plus methods of purely technical significance, form a complex network of interactions. As a practical matter, the complexity of these interactions becomes nearly incomprehensible below the level of workflow.

The way to keep a straightforward process design from degenerating into an unmanageable tangle of code is to begin to enforce a consistent approach to the implementation of integrity constraints then to follow by taking this same approach in the technical design as well.

These consistent approaches to building constraints and technical interfaces represent design patterns. Hewing to consistent design patterns makes code easy to maintain, re-use, and re-engineer. At the level of the object model, design patterns must be established for:

- **Creating and destroying objects.** Usually, all properties of objects are brought into memory and saved all at once--not piecemeal as with the SQL database interface.
- **Messages.** The designer must enforce a consistent convention for how information is passed from one method to another.
- **Triggering constraints.** The designer must create a consistent time and sequence for validating data as it is saved plus a consistent approach to error handling.

## Application Technology

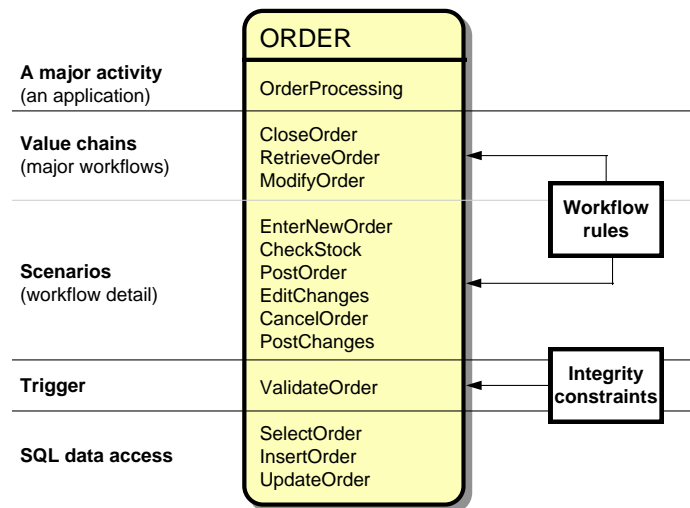
The shape of the technical application begins to emerge as the business requirements are defined. Each process defined as a business requirement--beginning with the most general business activities and ending with the most detailed integrity constraints and business rules--is built as a clearly articulated technical component.

This means that a direct map exists from any business requirement to a corresponding piece of programming code. Each piece of code is defined in terms of its interactions with every other piece of code. This being so, each piece of can be examined and modified in isolation from every other piece of code. This isolation strategy--called *encapsulation* in object terminology--is what makes object-oriented applications easy to maintain. Changes are limited to only a single method or a small group of interacting methods.

Where application components are properly isolated, rules can be encapsulated separately from larger processes. Capturing rules separately makes conditional logic easy to locate and modify. In this approach, rules are handled by a separate engine. Business users (in conjunction with a rules expert) can locate rules, read them in a plain-English form, and modify them without touching critical application code.

Note that pure object-oriented programming techniques are not required to achieve this kind of encapsulation. Here is an illustration of the methods of the Order object, tied to the application hierarchy:

**Figure 11. Articulating an Order**



The final result is an application in which each business requirement--activity, process, rule, or constraint--maps directly to a piece of code that can be isolated for easy modification. Because the application is build on object-oriented design principals, it is easy to build and maintain. Because business requirements and code are closely, wedded it is easy to re-engineer. The application will not become obsolete as soon as it is built, but will adapt to continuing changes in the business requirement.