An Object Model for Business Applications

By
Fred A. Cummins
Electronic Data Systems
Troy, Michigan
cummins@ae.eds.com

EDS

This presentation will focus on defining a model for objects--a generalized representation to be used in the development of business applications. In "business application" I am including any application used in the operation of an enterprise. The enterprise might be commercial business or it might be a government agency or an academic institution. My purpose is to define a level of abstraction to be supported by the Business Object Facility (BOF) that is the subject of the RFP issued by the OMG Business Objects Domain Task Force. My concern is that the BOF should greatly simplify the effort required by application developers to develop solutions to business problems. I am hopeful that this presentation will help potential submitters understand the requirements I see for the BOF from an application developer's perspective, so they can respond accordingly to the RFP.

Business Object Model

A computational abstraction:

- **The conceptual basis for implementation of a class of applications**
- **Incorporates general-purpose computational mechanisms**

EDS

The Business Object Model I will be discussing is a computational abstraction. It is the conceptual basis for implementation of a class of applications. These applications represent problems that would exist with or without computers. They are problems where objects are used to represent business concepts and model problem situations.

The Business Object Model incorporates general purpose computational mechanisms. As such, the model conceals computational problems from the concern of the application developer, allowing him or her to concentrate on modeling the business problem.

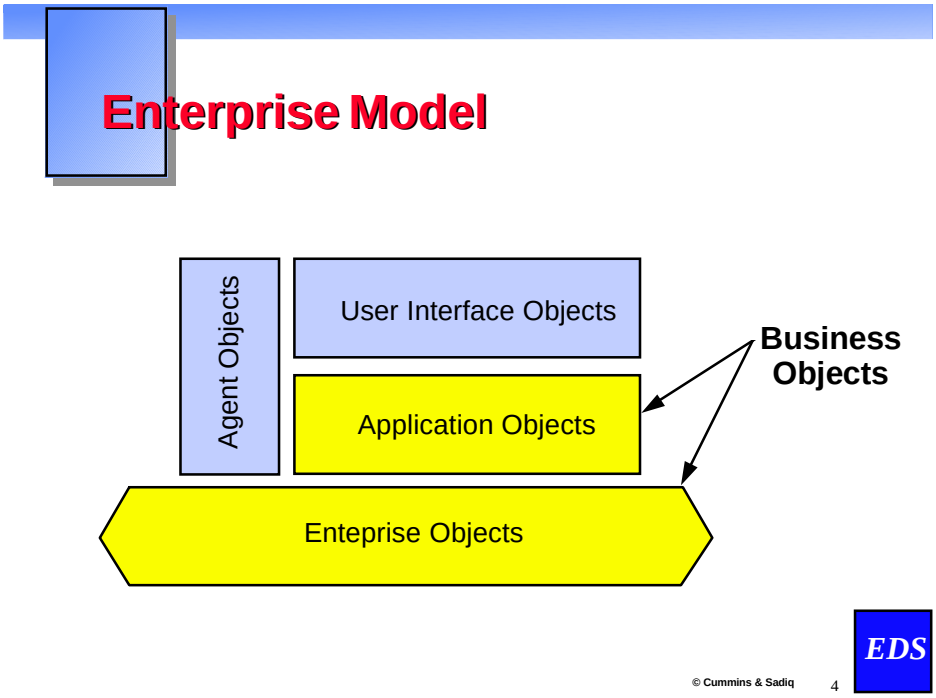
Why do we need a Business Object Model that is different from the model of objects used for other computations?

First, we should minimize the transformation effort required to implement models for business domains. This is important for development of flexible systems that meet user requirements. The more involved the transformation, the greater the difficulty in meeting user requirements.

Second, we should conceal computational mechanisms necessary to support applications in a distributed, heterogeneous environment. OMG technology has introduced several new dimensions of complexity which will increase the effort and risk of application development if these problems must be solved in every project by application developers.

Third, we must assure compatibility of independently developed components. Application developers must ensure the compatibility of the business concepts and methods, but there must also be compatibility of underlying mechanisms such as transaction management, concurrency, relationship management, notification and persistence. Without this compatibility, we cannot hope to develop a marketplace in sharable business application components.

Fourth, with a higher level of abstraction and a stable architecture, we can begin to integrate intelligence into business applications with such mechanisms as constraints, agents, expert rules and adaptive mechanisms. Today, such capabilities are limited in scope and difficult to integrate because of inconsistencies and inaccessibility of information across business applications.

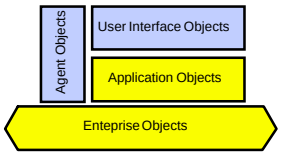


This Enterprise Model illustrates where I see business objects used. This model is similar to the three-tiered model with a couple exceptions. First of all, the bottom layer is not data storage, but rather business objects that are shared across the enterprise. Data storage, or persistence, should be orthogonal to the design of business applications--it should be automatic. Secondly, agents are added as mechanisms for monitoring and directing activities in the other layers. These might be workflow management components, expert systems or intelligent tutors, for example.

Business objects exist in the enterprise objects and application objects layers. As I mentioned, the enterprise objects implement concepts that are shared by many applications. They model the business as a whole. Application objects model a particular problem or segment of the business. Many of the application objects will be views of enterprise objects and there must be an active linkage between these layers.

The use of views to separate the enterprise and application layers allows these models to evolve relatively independently. Because the interface is through methods, when changes occur in the enterprise model the affected view methods may be modified to adapt to the change without otherwise affecting the applications. When applications change, the effects need not extend to the enterprise model unless the extension should be shared across the enterprise. Views also allow applications to work with a simplified representation that is sufficient for the application but maps to a more robust representation for the enterprise-wide model. The same basic mechanism should be used to attach agents and user interfaces.

"Business" Objects



Objects used to represent concepts that occur in an enterprise (business) for performing computations or maintaining information about the enterprise.

Concepts

- **Would exist in a non-computer solution**
- **Range from enterprise-independent to situation-unique**
- **Could be represented with large-grained or fine-grained objects**
- **Represented independent of programming language, operating system, database, or other computing facilities**

EDS

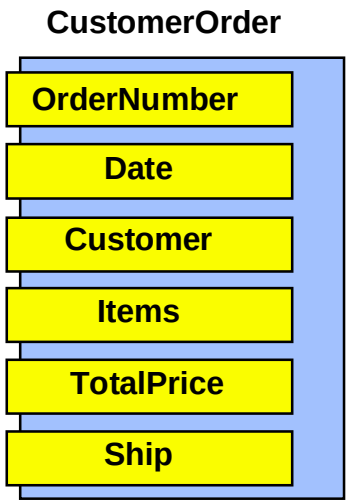
This slide refines the concept of a business object for purposes of this discussion. Business objects are used to represent concepts that occur in an enterprise, or business, for performing computations or maintaining information about the enterprise.

These concepts would exist in a non-computer solution to the business problem--they are not concepts that are necessary to make computers useful. For example, graphical display objects are not business objects.

Business objects include large-grained and fine-grained objects. They implement a wide range of concepts.

Finally, the concepts being represented are independent of programming language, operating system, database, or other computing facilities. The concepts do not depend on computational implementations, but on the nature of the enterprise.

A "Business Object" Example



EDS

Here is an example of a business object: a customer order. The smaller boxes represent functional interfaces to the order which will get or put information or perform operations.

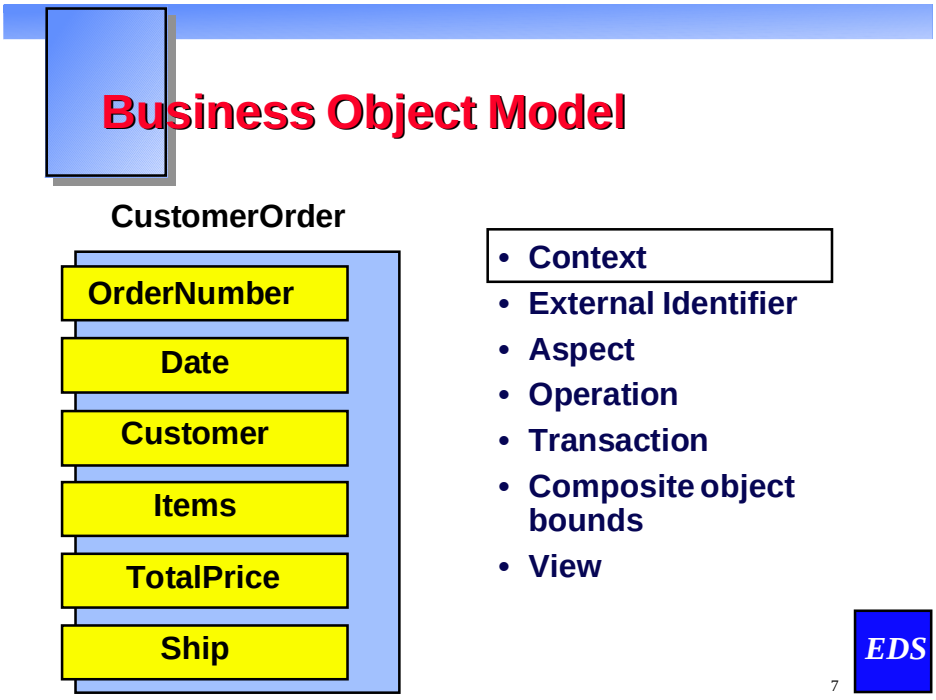
The OrderNumber, Date and TotalPrice interfaces will return elementary values--I refer to these as attributes.

The Customer and Items interfaces return references to other structured objects--I refer to the Customer and Items interfaces as relationships. The objects they reference may have complementary references.

The Ship interface invokes an operation (i.e., ship the order).

I refer to this object as a "composite" object because, semantically, the Items referenced by the order are part of the order. They represent the specification of parts and quantities requested by the customer. Generally, the order and the items will be used as a unit.

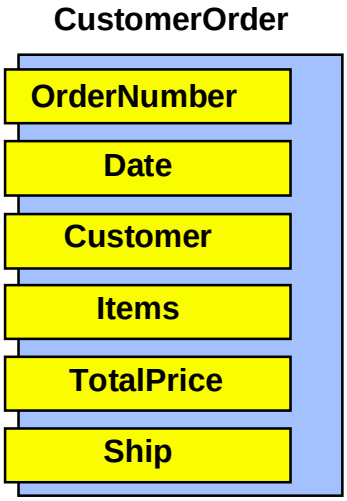
I will use this object example in further discussions.



These are characteristics of business objects that are important for the application developer. They represent the level of abstraction at which application developers should work.

First, business objects are used in a “context” that must be accessible to the application. This context includes such things as user preferences, organizational affiliation, and default I/O devices such as the user's printer. This context must be available even though the processing associated with the application may extend to other computers in a distributed environment.

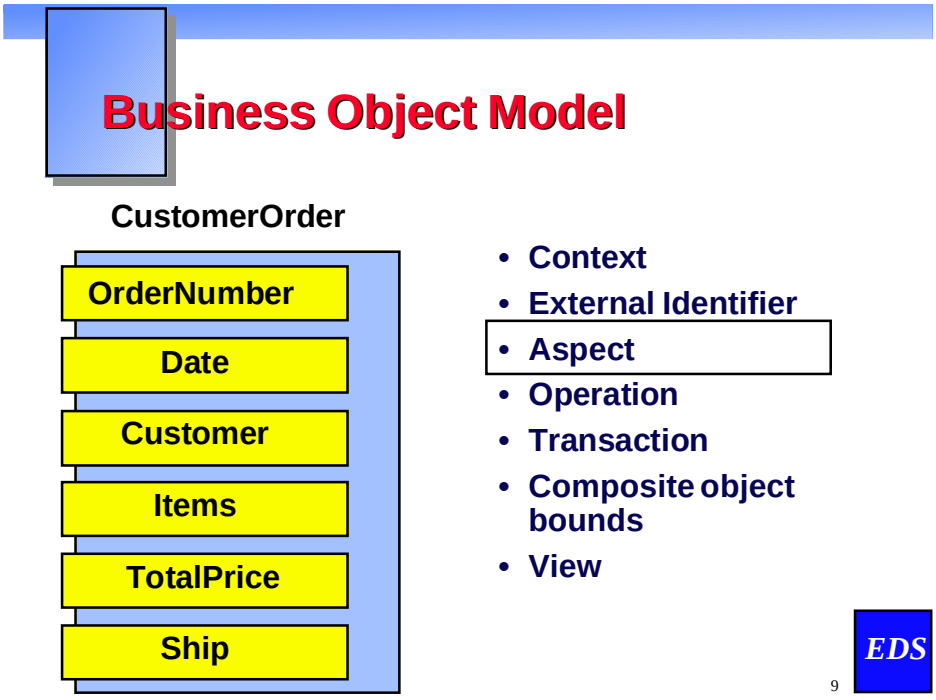
Business Object Model



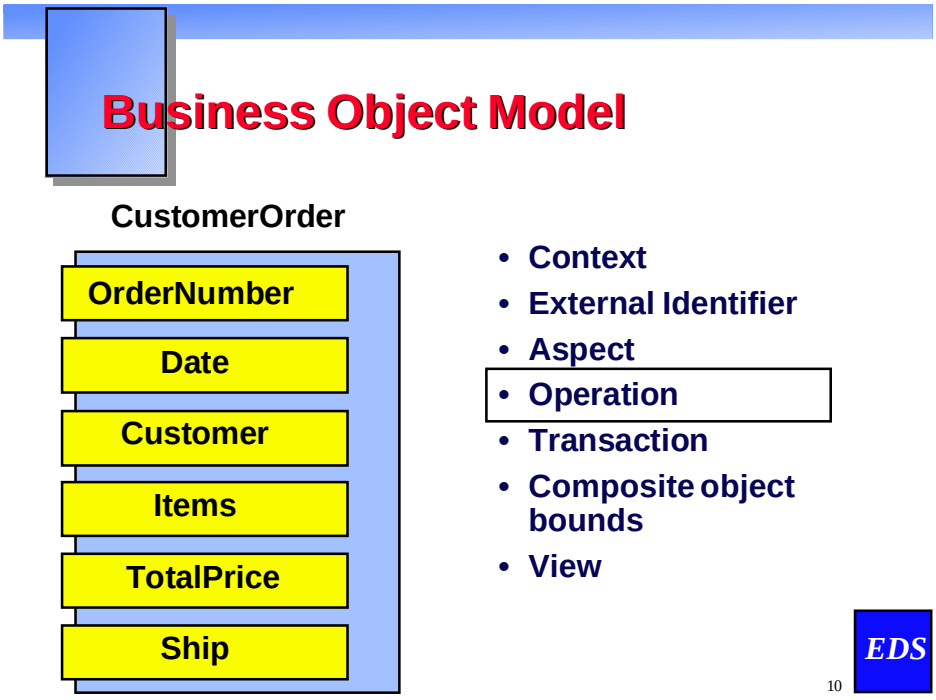
- Context
- External Identifier
- Aspect
- Operation
- Transaction
- Composite object bounds
- View

EDS

Business objects, for the most part, have identifiers that are used by humans to reference the objects. For example, order number as well as social security number, employee number, vehicle ID, part number, customer name, city name, and product name are all external identifiers. These must be interpreted in a context since the identity of an employee or an order, for example, must be determined by reference to the enterprise. Whenever one of these objects becomes the target of processing, these external identifiers must be translated to the identity of an active object even though the object may not currently be active--it may only exist in a database.

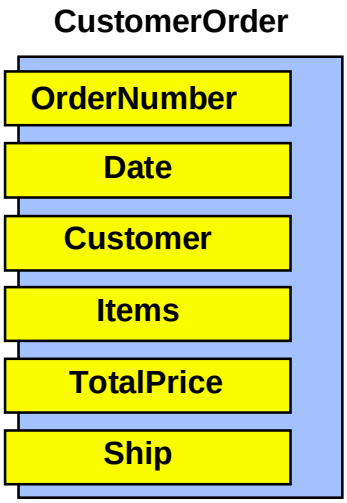


I use the term *aspect* to define interfaces associated with attributes or relationships of an object. At an elementary level, an aspect provides get and put methods for a value associated with the object. However, some aspects may return computed rather than stored values. For example, TotalPrice is probably a computed value. Furthermore, the implementation of aspects must include functionality to support related computational mechanisms which I will discuss later. For example, in order for aspects to support change notification, the put methods must generate appropriate notices when a value is changed.



Operations are methods that perform actions--they implement business processes. These processes should propagate to other computers and objects implemented in other programming languages without application programmer concern about destruction of objects inadvertently created, concurrency conflicts with other processes or retrieval of referenced objects from a database. Programming of these methods should be essentially the same as if all of the objects referenced were in a single computing environment.

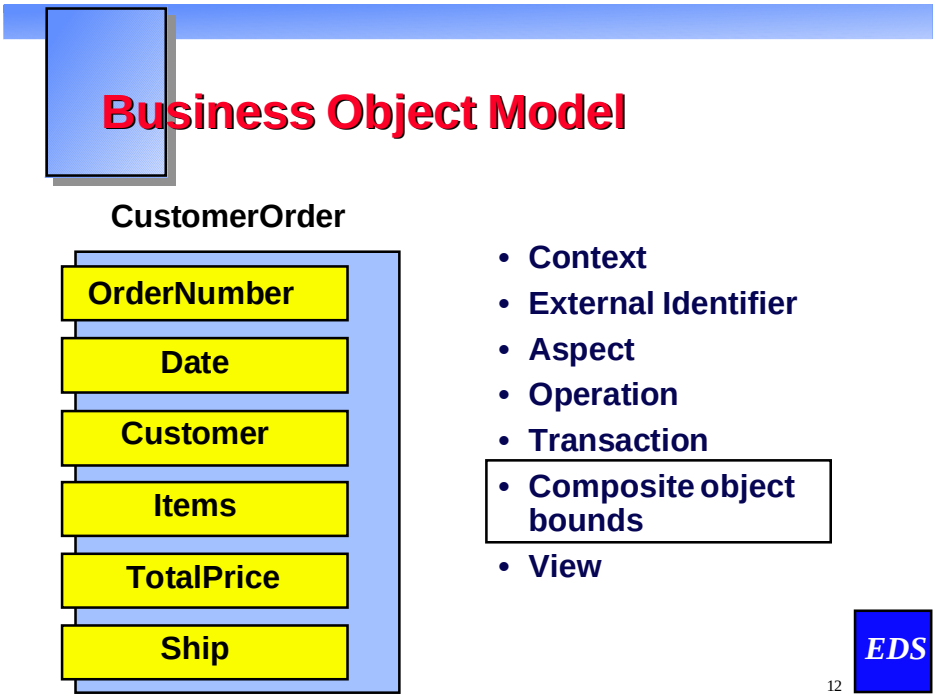
Business Object Model



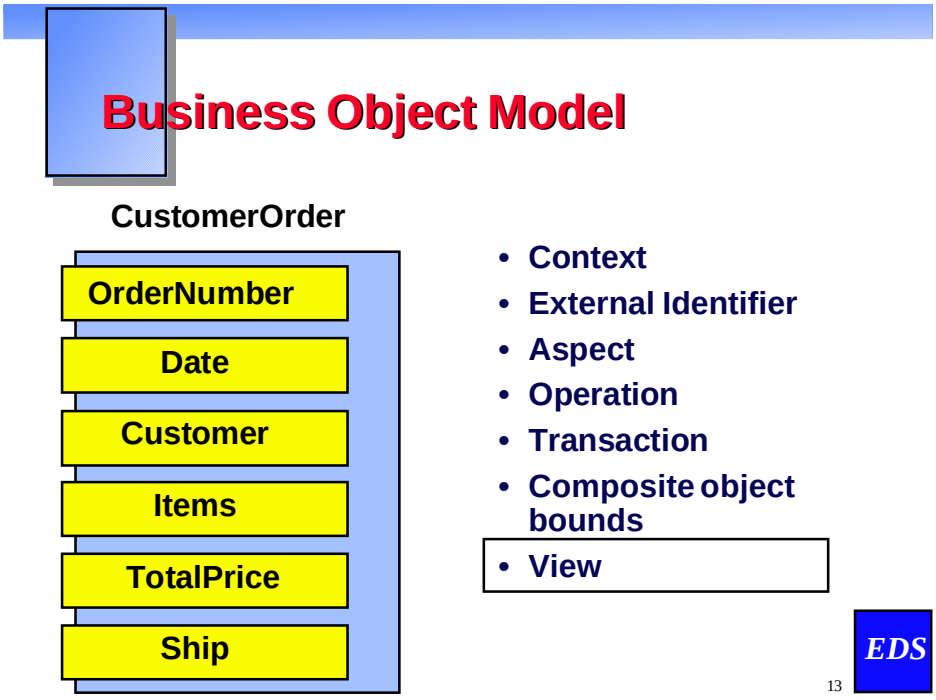
- Context
- External Identifier
- Aspect
- Operation
- Transaction
- Composite object bounds
- View

EDS

Transactions must be visible to the application developer from the standpoint that the developer must define when a transaction starts and terminates, either successfully or unsuccessfully. Transactions are a business concern and must be defined in a manner that is consistent with the way the user does his or her work. At the same time, of course, transactions affect other mechanisms that should be concealed from the developer, such as concurrency control. For example, if a user is going to update information on an order, a transaction should be initiated for that activity. If the update is completed successfully, then the transaction should be committed, and the user should be aware of its acceptance. If the user attempts to enter unacceptable information or decides to abandon the changes, then the transaction should be terminated and any tentative changes should be backed out--the user should also be aware of this action.



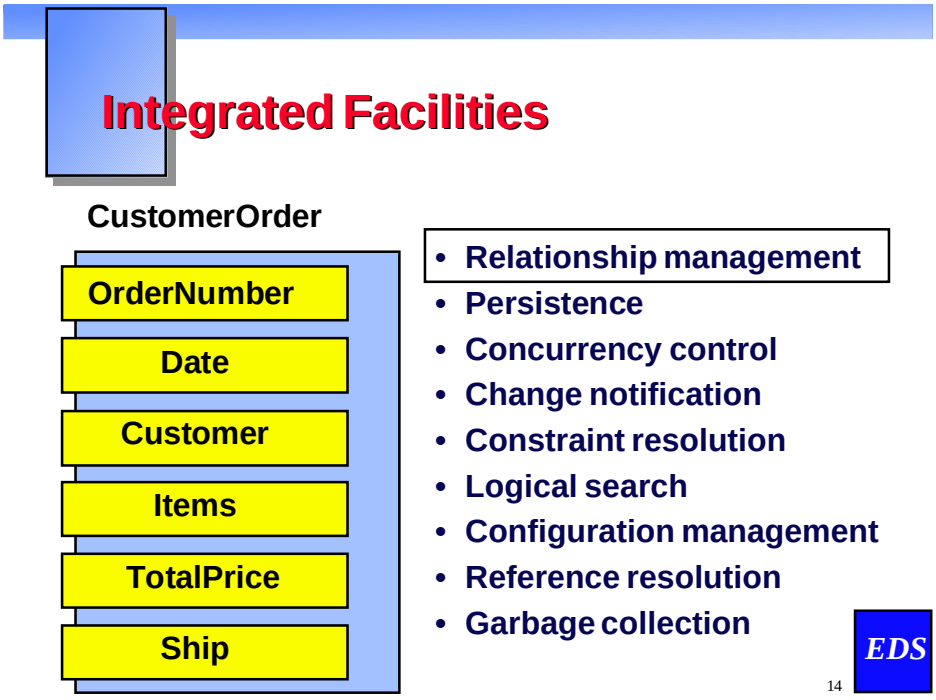
I noted earlier that the customer order is a composite object--it has item objects that are part of the order. These item objects should be included when the order is copied or cancelled, but the parts that are referenced by the items should not be included in such operations. Consequently, the relationships between objects that are part of the order and other objects external to the order--such as parts--must, in some way, be identified as limiting the bounds of the order. Such specifications should be indicated on the aspects that define the boundary relationships, but the imlementation of this designation should be concealed from programmer concern.



Finally, application developers should be given views for their local representation of enterprise concepts. Views are a special form of application business objects. These views must participate in application processing as local objects and may have local aspects and operations. But they must also be linked to the corresponding enterprise objects so that updates to state directed to views are propagated to the enterprise objects, and updates to the enterprise objects are propagated to the views of other applications that are actively working with the shared information.

Views may also simplify the representation of a business concept so that a complex structure in the enterprise model might be simplified to a single view for a particular application. For example, a participant in a project might be represented as a single object for a project management application, while that single object might incorporate information obtained from the full employee object and related person object in the enterprise model

In addition to providing simplified representations and adaptive interfaces for applications, views also provide a mechanism for security control. A view will define those aspects and operations available to the user of the view. For example, a project manager view of an employee would exclude salary information.



Now lets turn to the facilities that should be integrated into the Business Object Facility. The implementation of these facilities should not concern the application developer.

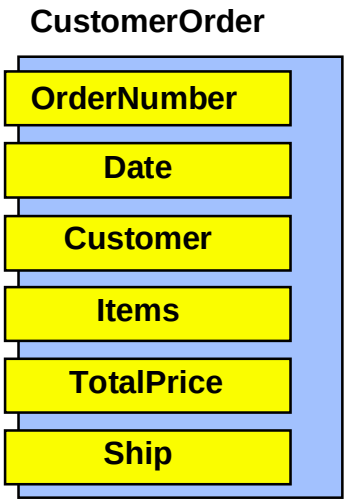
Relationships between objects should be incorporated in the implementation of the “aspects” I mentioned earlier. When an object is added or removed from a relationship, complementary relationships should be updated appropriately without explicit programmer action.

In the example, the addition of an item to the order should update the Items relationship including a reference from the new item object back to the order.

Relationship management must also handle the difference between references to objects that are “part of” a composite object, and objects that are just referenced. When a non-part-of relationship is externalized--moved to a file or database, the references should be converted to the associated external identifiers. The elements of a composite object will generally be stored together and may be linked by local references appropriate to the storage medium.

In the example, items are part of the order, but references between items and parts are not “part of” relationships.

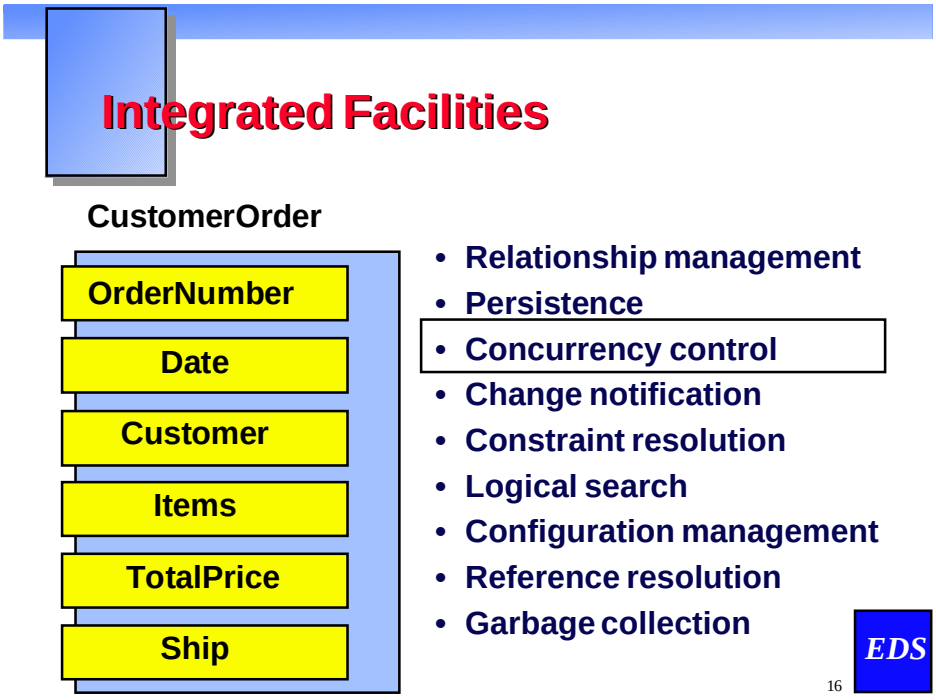
Integrated Facilities



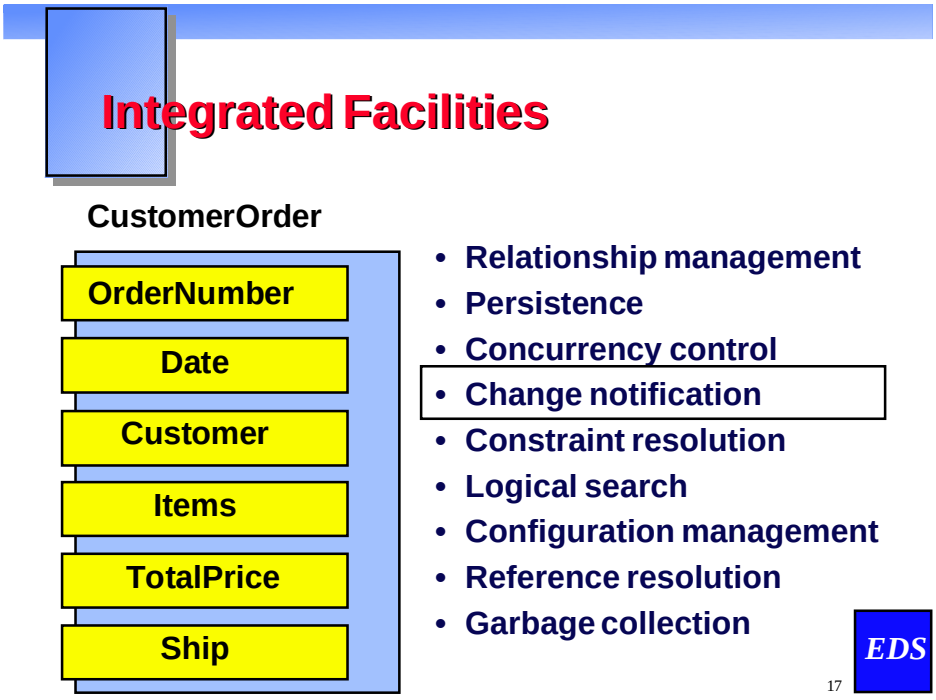
- Relationship management
- Persistence
- Concurrency control
- Change notification
- Constraint resolution
- Logical search
- Configuration management
- Reference resolution
- Garbage collection

EDS

Persistence should be invisible to the application developer except that an object should be designated as persistent or non-persistent in a way that is independent of application logic. When a message is sent to a persistent object, the object should be automatically retrieved from the database if it is not already active. When an update transaction is committed, all persistent objects that were updated by that transaction should automatically be written to the database.

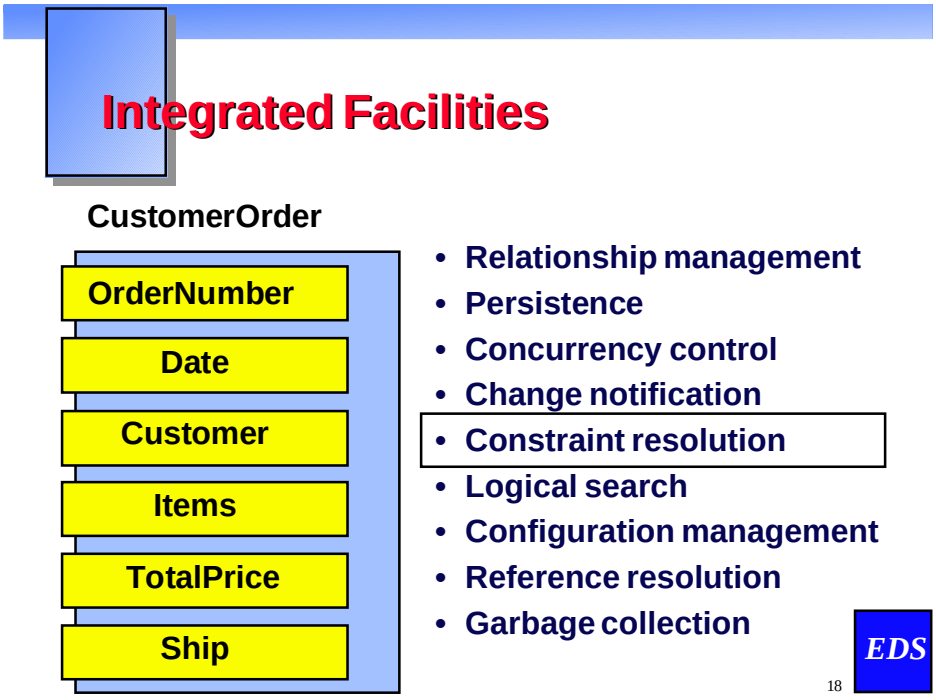


Concurrency control should likewise be invisible to the application developer. Concurrency control, for this purpose, means assuring transaction serialization. The net effect of processing of a group of transactions should be the same as if they were executed serially, one after another. Locking and deadlock detection should be handled automatically.



Change notification is a service that should be available for all business objects. For example, an account manager might be interested in knowing if the total price of an order changes. The monitoring facility would request change notification on the TotalPrice aspect for orders of interest. In general, most orders would not generate these events because nobody is interested.

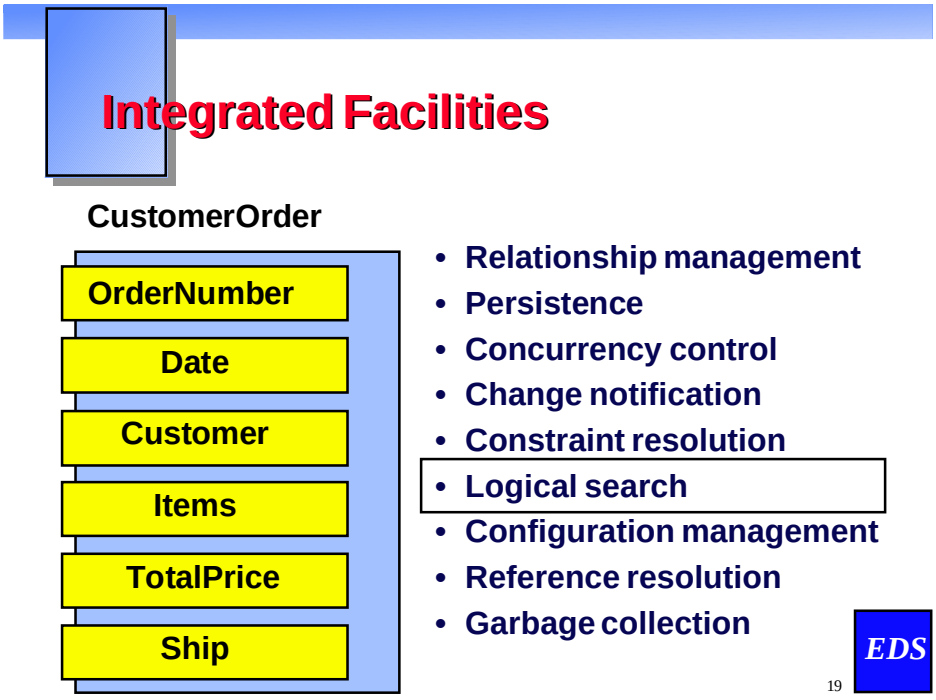
Change notification is not the same as event notification as currently defined in OMG. A request for change notification is directed to a specific object instance for notification of change to a particular aspect. OMG event notification is based on a request to a service for notice of any occurrence of an event type; it requires that all sources of events which may be of interest always notify the service of the occurrence of any such event. The granularity is very different. Change notification is necessary to drive displays, agents and active views. The notification mechanism should be automatically incorporated in object aspects.



Constraints are a very important facility that has not been well developed for object-oriented applications. Constraints provide a declarative means to apply business rules and assure model consistency and integrity.

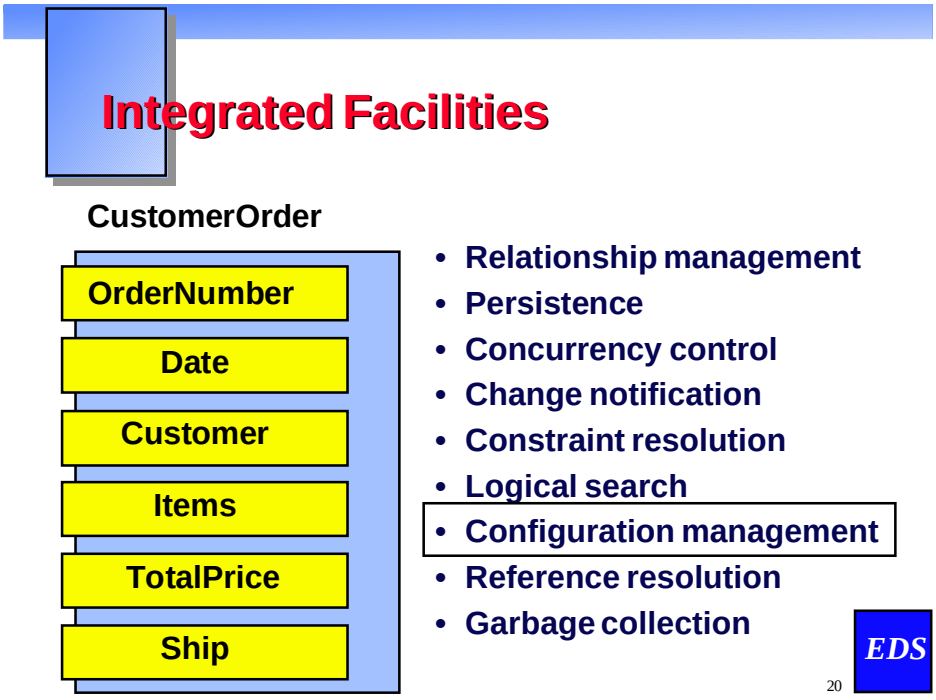
For example, a constraint might limit the TotalPrice of an order based on the customer's credit limit.

Constraint propagation provides a valuable mechanism for manipulation of complex models such as engineering designs. The mechanism for attachment and resolution of constraints should be invisible to the application developer.

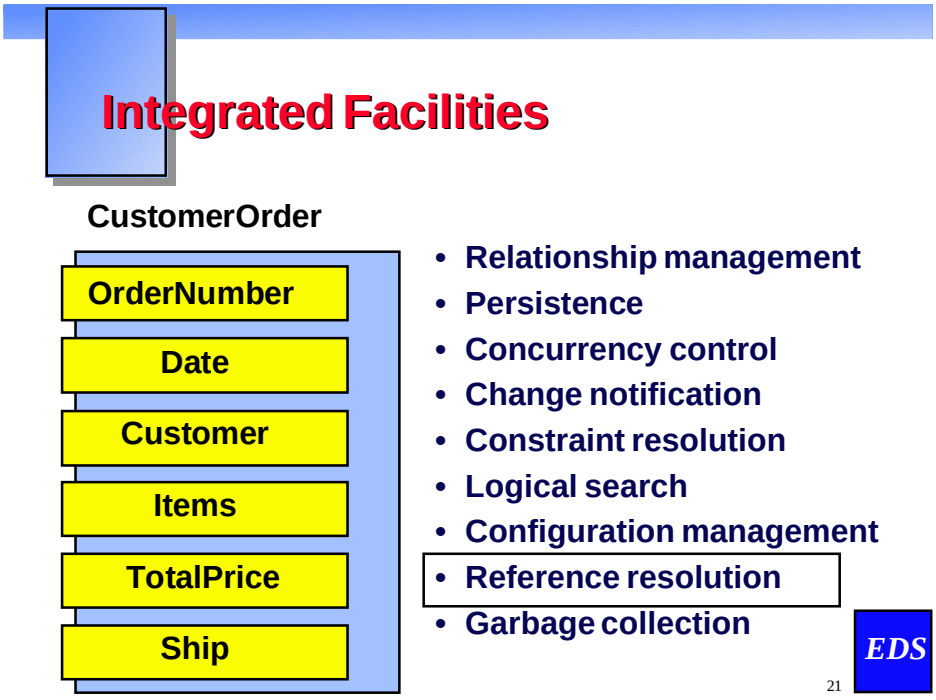


Searches in the business objects environment, i.e., queries, should be logical searches based on the conceptual model, not searches based on physical structures. For example, SQL queries require that the user understand the table structure of the database. The scope of an SQL query is the scope of the database. These are physical characteristics of the implementation which should not be visible to either the application user or the developer.

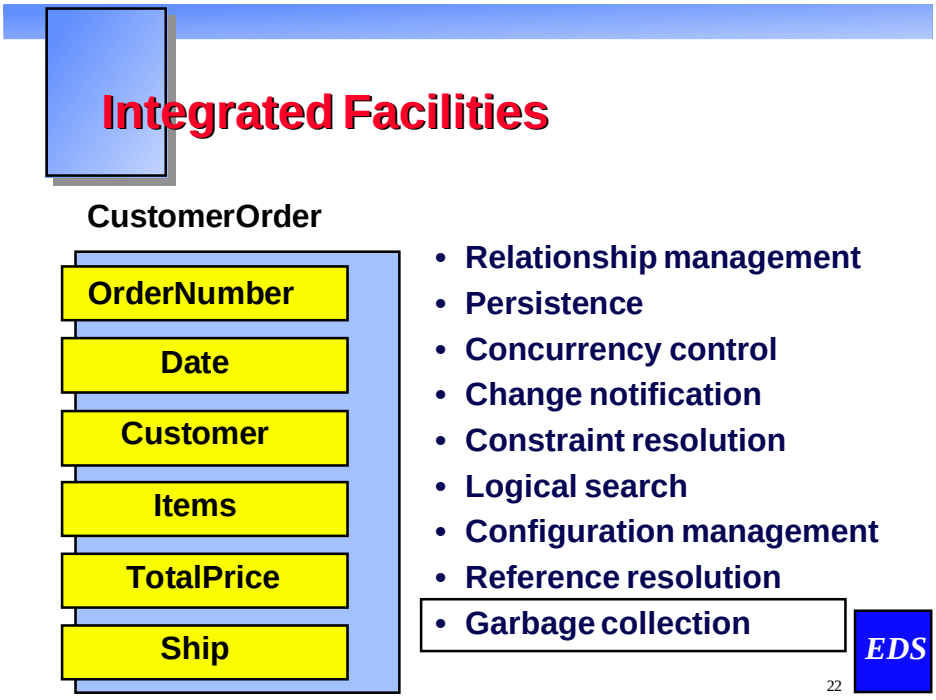
A logical search should be based on the conceptual model, the context of the request and logical definitions of facts. The search should not be restricted to objects that are persistent. And the search mechanism should be integrated into the environment, not restricted by the boundaries of computer and data storage devices.



Application developers also need facilities to support configuration management. Developers must be able to package application components and deploy versions of those components. Components and their versions will have dependencies on other components and versions. This is challenging when configuring a single-computer application; it can be a nightmare when dealing with a distributed computing environment with workstations of varying configurations depending upon user requirements and authorization. Configuration management should also consider security controls and workstation hardware constraints.



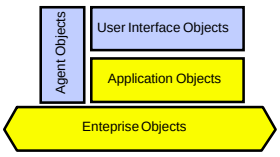
Object references should be resolved without application developer involvement. If an application has a reference to an object, that reference should be guaranteed viable unless an unrecoverable error has occurred. If the object is persistent but is not currently active, then, if a message is sent to it, it should be automatically retrieved from the data base without explicit programming by the application developer.



Finally, objects that are no longer in use should be automatically garbage collected. This applies to objects in computer memory and objects in databases. Application developers should not need to worry about explicit destruction of objects or memory leaks. When an object is no longer referenced in any environment then it should be destroyed. This is particularly critical when objects are shared by multiple applications and users. Automatic garbage collection requires mechanisms to resolve remote references and mechanisms to detect and update accounting for references by remote nodes that have failed.

AGarbage collection and all of the above facilities must function consistently across ORB domains so that application developers need not be concerned with the distribution of their applications or related objects. This is essential for meaningful interoperability of business applications.

Summary



- **Minimal modeling transformation**
- **Easy to develop applications**
- **Large-scale integration**
- **Compatible components**
- **Heterogeneous environments**
- **Build on accomplishments**

EDS

In summary, the implementation of this abstraction with the Business Object Facility will have very important benefits.

The transformation of analysis models to application implementations will be minimal, improving the productivity of developers and the flexibility and effectiveness of applications.

It will be easy to develop applications in distributed, heterogeneous environments because application developers generally will not need to be concerned about the added complexity.

Such facilities will enable large-scale integration. The enterprise model described allows applications and enterprise objects to evolve while the scope of impact of changes is confined (using views as adaptors). This enables sharing across many applications and business functions.

We can reasonably expect that independent developers will be able to produce compatible components because the architecture and computational mechanisms will be consistent, and developers need only concentrate on defining consistent business semantics and protocols.

These systems can be expected to operate in heterogeneous environments because application developers will be shielded from the differences between environments. When new software releases or computing platforms are introduced, there will be no need to redevelop applications.

Finally, we will be able to start building on our accomplishments instead of starting from scratch over and over again to adapt to new technology. We can start to exploit more sophisticated techniques to incorporate intelligence, facilitate collaborative work, and achieve enterprise-level optimizations rather than sub-optimal, local solutions.

Thank you for your attention.