Object Services: Working in Concert

A White Paper

SunSoft
A Sun Microsystems Company

Table of Contents

Object Services: Working in Concert

The Project DOE Object Services are a core set of services designed to support common interactions between objects. Developers can use the Object Services to integrate objects into a common framework and leverage a set of predefined DOE interfaces and implementations to support these interactions.

This paper describes the individual Object Services, the concept of how objects are collected into "graphs," and how the Object Services work together to support graphs.

What are Object Services?

The Object Services include:

- Associations
- Events
- Naming
- Properties
- LifeCycle
- Transactions
- Persistent Storage Manager (PSM)

Each of these services can be used individually but they are also designed to be used *together*.

Associations

Associations connect objects together. Object references also connect objects, but associations make this connection explicit. When you use object references to connect objects, objects connect because one object holds a reference to the other object — the object is "associated" with the other object. However, the object being referenced is not necessarily aware of the *association* between it and other objects.

In Figure 1, the document holds one object reference to the graphic and another object reference to the logo. However, the logo has no information about the graphic or its connection to the document. Similarly, the graphic has no information about the logo or its connection to the document. Furthermore, these object references cannot be manipulated externally, or extended with standard behavior.

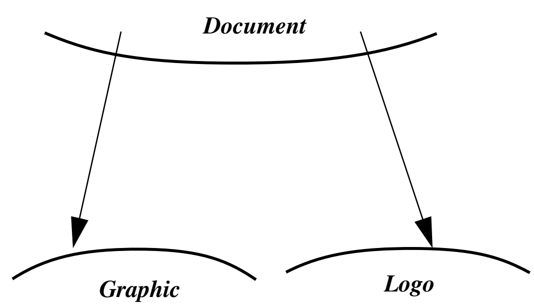

Figure 1 The Document References the Graphic and the Logo

The Association service supports associations between objects explicitly. *Associates* represent objects in associations; they record the connections between objects. Since associates are *themselves* objects, they have well-defined interfaces that can be manipulated externally, and they can be extended with rich behavior and attributes.

Figure 2 shows the association between three objects: a document, a graphic, and a logo. Each object, called an *associated object*, has an *associate* (represented by the gray ovals in the figure).

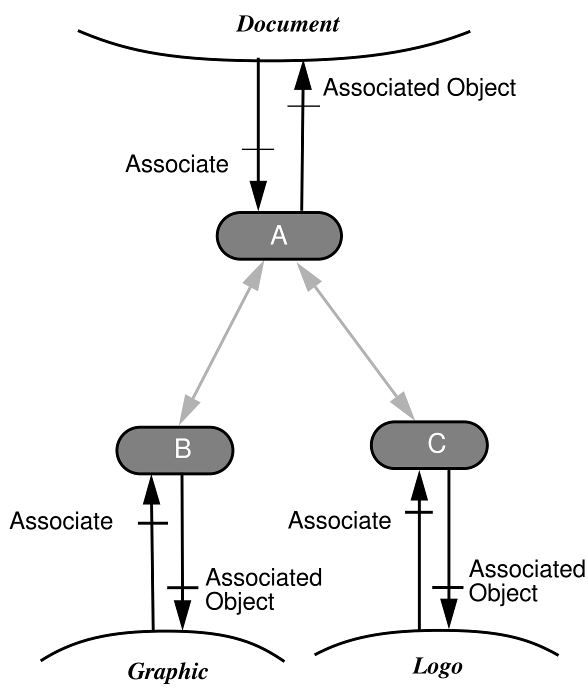

Figure 2 An Association between a Document, a Graphic and a Logo

Bidirectional connections are maintained between associates. The Association service supports *referential integrit*y for the bidirectional connections. If *A* is connected to *B*, then *B* is connected to *A*.

Events

The Event service provides asynchronous communication between objects; it extends the synchronous communication facilities defined by the CORBA[1]. The Event service decouples the communication between objects to provide more flexible communication.

The basic elements of the Event service are the *consumer*, the *supplier*, and the *event channel*. *Suppliers* produce events and *consumers* process event data, which are communicated between suppliers and consumers using standard CORBA requests. Most suppliers and consumers communicate through an event channel, although direct communication is also supported. Figure 3 shows the basic elements of the Event service.

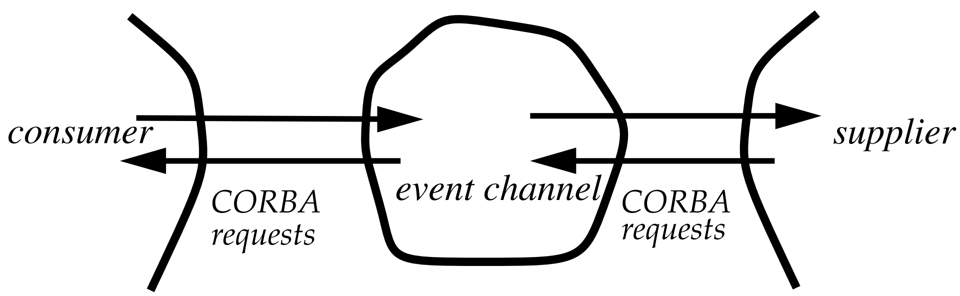

Figure 3 Event Channels Support Asynchronous Communication Between Objects

The Event service supports the *push* model and the *pull* model as two basic models for communication.

In the push model, the supplier controls event delivery; in the pull model, the consumer controls event delivery. In Figure 3, the supplier pushes events to the event channel, which asynchronously pushes the event data to the consumer.

1. The Common Object Request Broker specification (COBRA) describes the Object Management Group's (OMG) industry standard architecture for the way objects transparently make requests and receive responses from other objects.

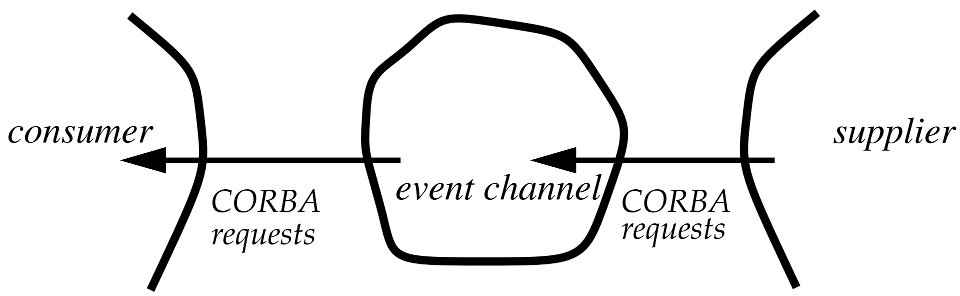

Figure 4 Pushing Event Data

For example, if the title of a document changes, the document (the supplier) could push the title change information to the event channel. The event channel would then push the information to interested objects (consumers); for example, to an index object or to a table of contents object.

In the *push* model, suppliers "push" event data to consumers; that is, consumers ask the supplier for events. If an application needs to regenerate a table of contents, for example, the table of contents would request title change events from the document and receive the new titles. The consumer essentially *polls* the supplier.

The supplier can use a style that is different from the style the consumer uses to communicate with the event channel. For example, a consumer can *pull* event data from the event channel that the supplier has *pushed* into the event channel.

In a distributed system, the asynchronous communication provided by the Event service is important because the consumer might not be available to receive the event data. The event channel acts as a go-between for suppliers and consumers. The event channel holds data until the consumer is ready to receive it.

Naming

The Naming service binds names to objects. The relationship between a name and an object reference is called a *name binding*. Name bindings are always relative to a *naming context*, which holds collections of name bindings. Naming contexts can be named within other naming contexts.

Note – Not all objects need to be named.

Figure 5 shows a naming context containing two city names: "Chicago" and "San Diego." Associated with each name is an object reference to the object that actually represents each city.

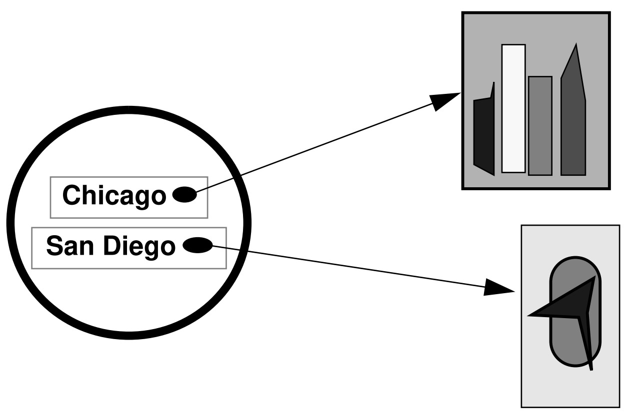

Figure 5 Name Bindings within a Naming Context

Names are *resolved* by determining the object associated with a particular name. You can *bind* a name by creating a name binding within a given naming context. You can bind different names to the same object; for example, "Chicago" in a cities naming context and "CHI" in an airport naming context.

Properties

An object supports an interface which is statically defined in IDL) and consists of *operations* and *attributes*. Two objects are of the same *type* if they support the same interface.

Sometimes it is useful to associate information with an object that is dynamically defined and outside of the type system. Such information is called a *property*. A property is a key-value pair. The keys are strings and the values are of type *any* — in DOE, type *any* enables the value to assume any type, but enables you to determine the value's type and use the value in a typesafe manner.

The Property service implements objects whose interface supports a set of properties. The Property service supports the *PropertySet* interface, which defines operations to create properties, get and set property values, delete properties, enumerate the properties in the set, and destroy the property set.

Figure 6 shows two documents and their property sets.

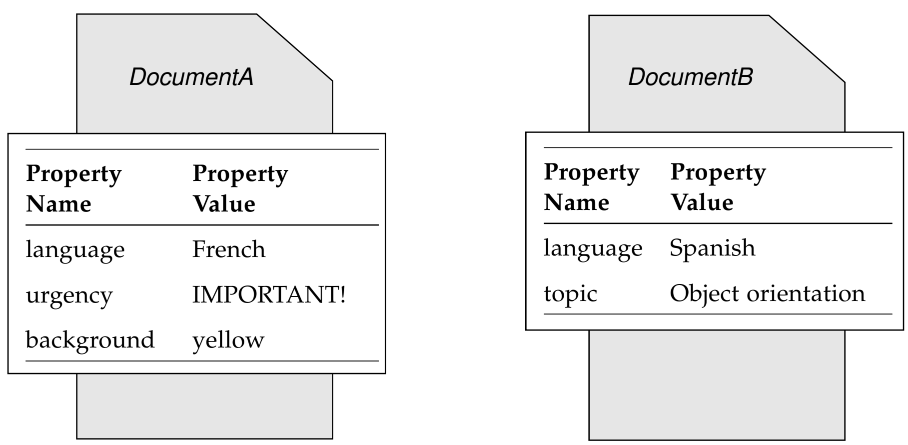

Figure 6 Two Documents and their Property Sets

The property set for *DocumentA* has three properties named “language”, “urgency” and “background” with values “French”, “IMPORTANT!” and “yellow”, respectively. The property set for *DocumentB* has two properties named “language” and “topic” with values “Spanish” and “Object orientation”, respectively. The properties are associated with their respective documents without being part of the documents’ types.

Transactions

The Transaction service guarantees the integrity of operations on objects in a distributive environment. Either a transaction executes completely, or it aborts. For example, if an application attempts to transfer money from a checking account to savings account, the Transaction service guarantees that either the money is transferred successfully, or that there will be no effect on either account. The accounts *cannot* be left in a state where the money is deducted from one account but not credited to another account.

Figure 7 shows an example of a transaction.

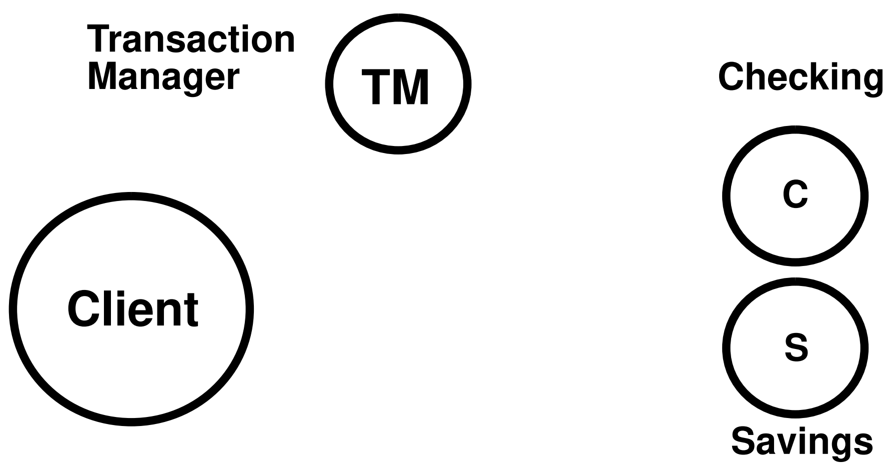

Figure 7 A Banking Transaction

In Figure 7, a client requests the services of two different distributed objects, the checking object (C) and the savings object (S). The Transaction Manager (TM) manages the interaction between the client and the objects. The Transaction Manager, in cooperation with the objects, guarantees that:

1. If the transaction fails, it leaves the objects in a consistent state.
   If the client issues a sequence of operations on the two objects, the Transaction Manager guarantees to the client that either all of the operations will finish completely or the transaction will abort.

2. A *sequence* of operations occurs in isolation.
   Isolating operations prevents interference from other concurrent clients. Without isolation, it is difficult to program a client when the client itself must track other clients and their effects on the target objects.

Persistent Storage Manager (PSM)

When an application implements a persistent object, the implementation of the object must provide a way to store the state of the object when the process running that object is inactive.

Object Services provides a Persistent Storage Manager (PSM) as a solution to the persistent storage problem. The PSM supports *persistent data objects*. Data objects are objects that only have attributes. DOE object implementations can use PSM data objects to persistently store their state.

The PSM provides a local application programming interface (API) and supports efficient caching of data. It provides the basic features of an object database, and is designed to allow upgrade to a full object database when multi-user concurrency control, versioning, complex queries, or other more sophisticated features are required.

To store an object's persistent state, the designer of a CORBA object implementation first uses the PSM Data Definition Language (DDL) to describe the object's persistent state as one or more *data object interfaces*. These data object interfaces are grouped into units called *schemas*. Each application typically uses a single schema. The syntax of DDL is a subset of the IDL language.

The PSM maps its persistent storage to disk in units called *Data stores*. Since a data store can only be accessed by one process at a time, typically a single CORBA server process encapsulates one or more data stores.

Using the PSM, CORBA objects can be implemented by code organized as *servant*s. A servant is composed of skeleton code generated by the IDL compiler, user written code, and code generated by the PSM DDL compiler. In C++, a servant would typically be a single C++ object.

ODF Object Implementation Components

The Object Development Framework (ODF) supports the implementation of DOE objects. ODF provides an API at a higher level of abstraction than DOE's basic object adapter (BOA)[1], which simplifies object construction and provides a connection between the BOA and the PSM.

The ODF model simplifies object construction by providing a set of overall policies for building objects. The ODF model is supported by software components that ease many of the details and complexities of BOA. As a result, developers can more easily implement objects which do not need all of the flexibility that the BOA offers. Developers needing to take advantage of the BOA's rich, flexible set of features have the option of implementing objects using the BOA directly.

Without ODF, you must write code to perform these BOA interactions:

- Activating and deactivating objects
- Object lookup (locating an object's implementation)
- Connecting an object to its persistent state

ODF handles these three tasks for you, and also supports the following object implementation tasks:

- Creating and deleting objects
- Clustering objects into groups
- Activating and deactivating groups of objects
- Deleting groups of objects
- Providing an administration interface
- Providing debugging support

Figure 8 shows the relationship between ODF, the BOA, and the PSM.

1. The BOA is an interface that supports a wide variety of common object implementations. It includes interfaces to generate object references, activate implementations, and establish various kinds of relationships between objects. The BOA also provides a limited amount of persistent storage for objects that can be used to connect to a larger or more general storage facility, to store access control information, or other purposes.

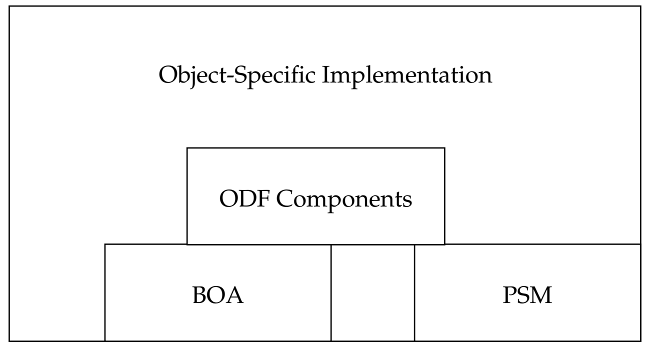

Figure 8 Object Implementation Components

LifeCycle

The LifeCycle service defines services and conventions for creating, removing, copying, moving, externalizing, and internalizing objects. Because CORBA-based environments support distributed objects, the LifeCycle service defines services and conventions that allow clients to perform life cycle operations on objects in different locations.

When a client copies an object, it is assumed that the object and the newly-created object can both communicate through the ORB. With externalization and internalization, this assumption may not be true. The externalized form of the object can exist outside of the DOE and can be internalized in a different, disconnected system.

The LifeCycle service generally operates on groups of connected objects, called *graphs*, rather than on a single object. One LifeCycle service, the Traversal service, coordinates the operation on a graph. Graphs are described in detail in the "The Graph Model" section.

The Graph Model

Objects do not float in space; they are often connected to each other. A set of objects and their connections are called a *graph*[1]. Figure 9 shows an example of a graph.

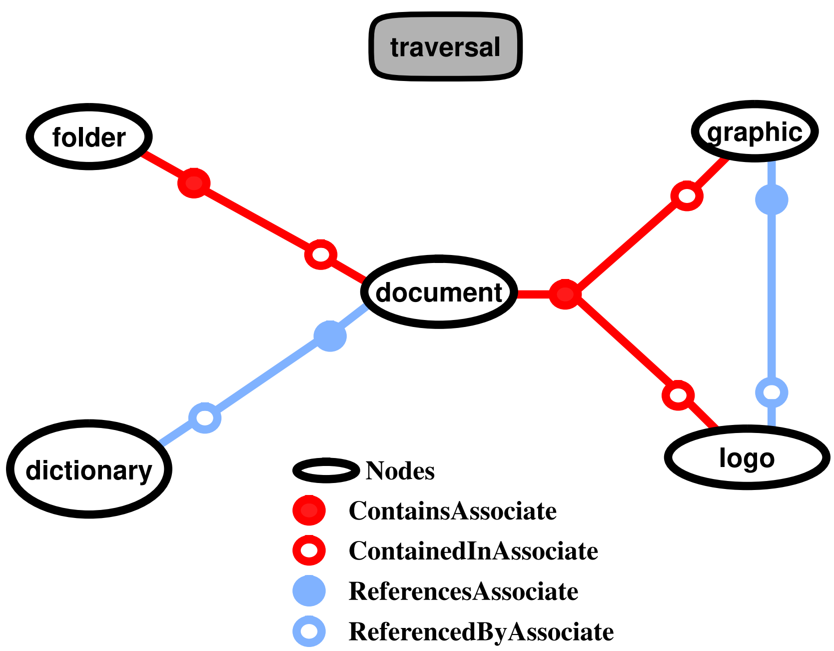

Figure 9 Detailed Graph Example

The document, folder, graphic, logo, and dictionary are known as *nodes*. Each node has a set of *associates*. The associates serve as representatives for the nodes in the associations. The associations between the objects are the *edges*. Different edges can have different semantics. For example, the logo is *contained*

1. In this paper, the term "graph" refers to an object graph and does not indicate a means of plotting data, which is the more common usage of the word graph.

in the document and *referenced* by the graphic. The document *contains* the graphic and the logo, and *references* a dictionary. The document is *contained in* the folder. The graph can contain any number of different types of edges.

We use this example to describe the DOE graph model in the following subsections.

Nodes

A *node* is an object that supports the *Node interface*, as depicted by Figure 10.

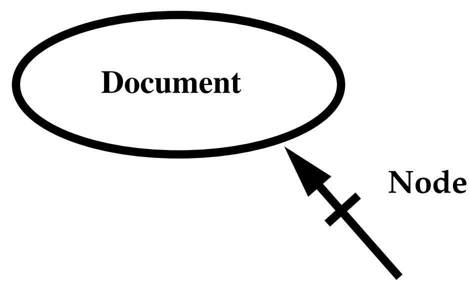

Figure 10 A Node Supports the Node Interface

In the example in Figure 9 on page 12, the document, folder, graphic, logo, and dictionary are all nodes. The Node interface prescribes a well-defined way for the node to behave within the graph. The Node interface supports the following LifeCycle operations for the node itself: copy, move, remove, externalize, and internalize.

Containment

Containment is a *one-to-many* association. A single object can contain many objects, and objects can be contained in no more than one object. The *containment* association is defined by pairs of associates:

- the Contains associate
- the ContainedIn associate

The associations between the folder and the document, and between the document and the graphic and the logo, are containment associations. The folder *contains* the document. The document is *contained in* the folder. The document *contains* the graphic and the logo. The graphic and logo are *contained in* the document. Figure 11 shows the containment associations of the example in Figure 9 on page 12.

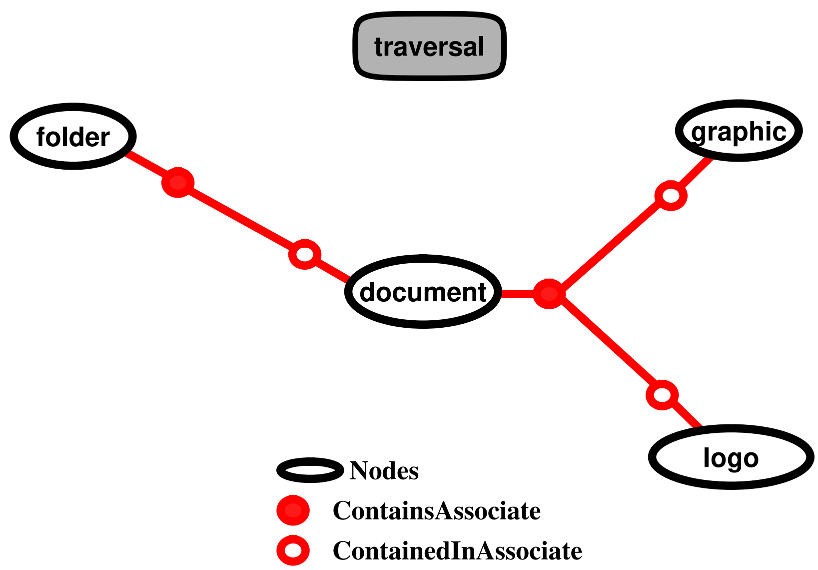

Figure 11 Containment Associations

Reference

Reference is a *many-to-many* association. An object can reference many objects, and objects can be referenced by many objects. For example, a dictionary could be referenced by more than one document, and a document could reference a dictionary and an encyclopedia.

The associations between the graphic and logo, and between the dictionary and the document in Figure 9 on page 12 are *reference associations*.

Figure 12 shows the association between the document and the dictionary. The reference association has two associates: the ReferencesAssociate and the ReferencedByAssociate. The document *references* the dictionary; the dictionary is *referenced by* the document.

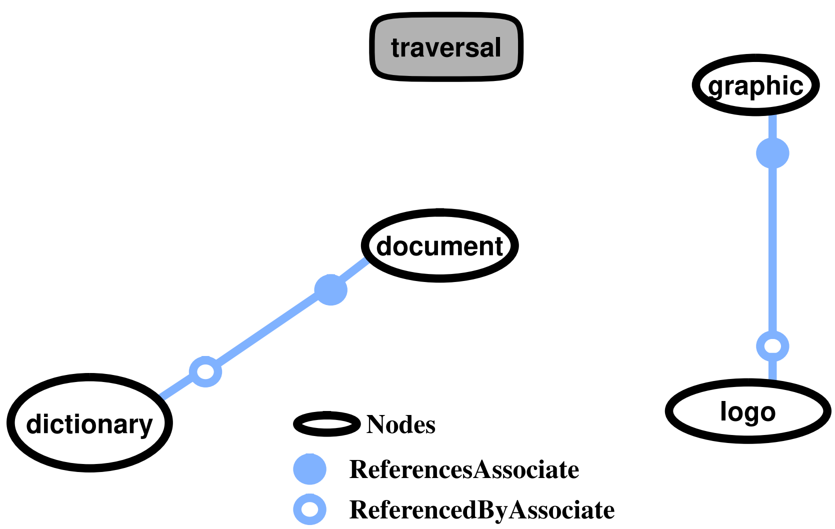

Figure 12 The Reference Associates

Applications that Use Graphs

Graphs are prevalent in DOE-based applications.

Distributed Desktop

A distributed desktop is an environment for distributed folders and documents. Folders contain and reference other folders and documents in multiple locations A distributed desktop is a graph.

Composed Applications

You *compose* an application by first developing the application from existing components, and then adding any other new components you need. An application builder (tool) creates graphs of the objects that represent the application components.

Compound Documents

A compound document appears as if it is one continuous document, yet it actually contains embedded objects or references to other objects. The document and its subdocuments are graphs.

Object Services Benefits for Applications

DOE defines a consistent graph model. The benefit of using the DOE model is that standard services and utilities can take advantage of the consistent graph model when operating on graphs.

Object Services Support Graphs

While individual object services provide powerful benefits individually, they have been designed to work in concert to support graphs.

Associations: The Graph's Glue

As illustrated in Figure 9 on page 12, associations are the glue that connects nodes together. That is, the associations are the edges of the graph. They can be extended with additional attributes and behavior. Containment and reference are two important examples of associations. Other kinds of associations are also possible.

Events: To Report Changes in Graphs

The Association service defines the *ReportingAssociate* interface. Reporting associates support the registration of consumers interested in events related to changes to the graph. An object can register interest in changes to the graph. For example, an interested object could be notified whenever an edge is added to the graph or deleted from it.

LifeCycle: Supports Graph Operations

The Traversal service, an essential component of the LifeCycle service, supports the following life cycle operations on the graph itself:

- copy
- move
- remove
- externalize
- internalize

These operations begin at a starting node. The Traversal service, in cooperation with the associates and the nodes, propagates the operation from one node to another node according to the *propagation semantics* of the association's connecting the nodes. Table 1 describes the three ways that operations propagate from one node to another node within a graph.

Table 1 Ways Operations Propagate within a Graph

Method	**Description**
None	When an operation is applied to a node, the operation is applied only to the node itself.
Shallow	Operations are applied to the node and to the *association* between nodes but *not* to the associated node.
Deep	Operations are applied to the node, to the association, and to the neighboring node.

Figure 13 shows these propagation semantics graphically.

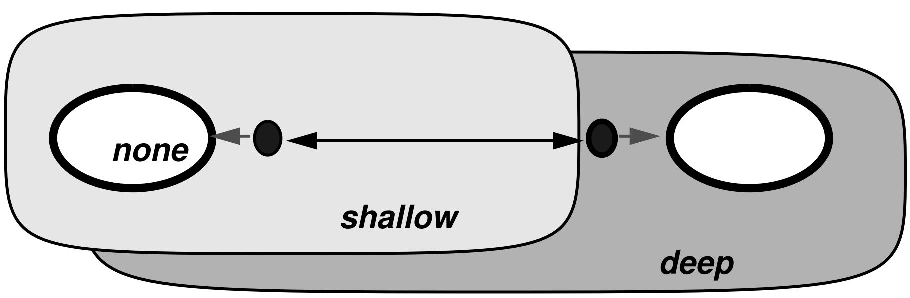

Figure 13 Three Ways of Propagating Operations

For example, the Traversal service makes a copy of the folder in the graph shown in Figure 9 on page 12.

1. The Traversal service starts at a single node: in this case, the folder. The folder makes a copy of itself.
2. The folder then informs the traversal service of all its associates — the folder's representatives in the associations in which it participates. By this protocol, the Traversal service learns about the folder's Contains associate.
3. The Traversal service then asks the ContainsAssociate for its propagation semantics with respect to the copy operation.
4. The ContainsAssociate replies that copy is *deep*, so the copy operation propagates to the document.
5. The Traversal service then asks the document to copy itself and reveal its associates to the Traversal service.
6. The document replies with its own ContainedInAssociate, its ReferencesAssociate, and its ContainsAssociate.
7. The copy process continues until the copy operation is complete. The resulting graph is shown in Figure 14.

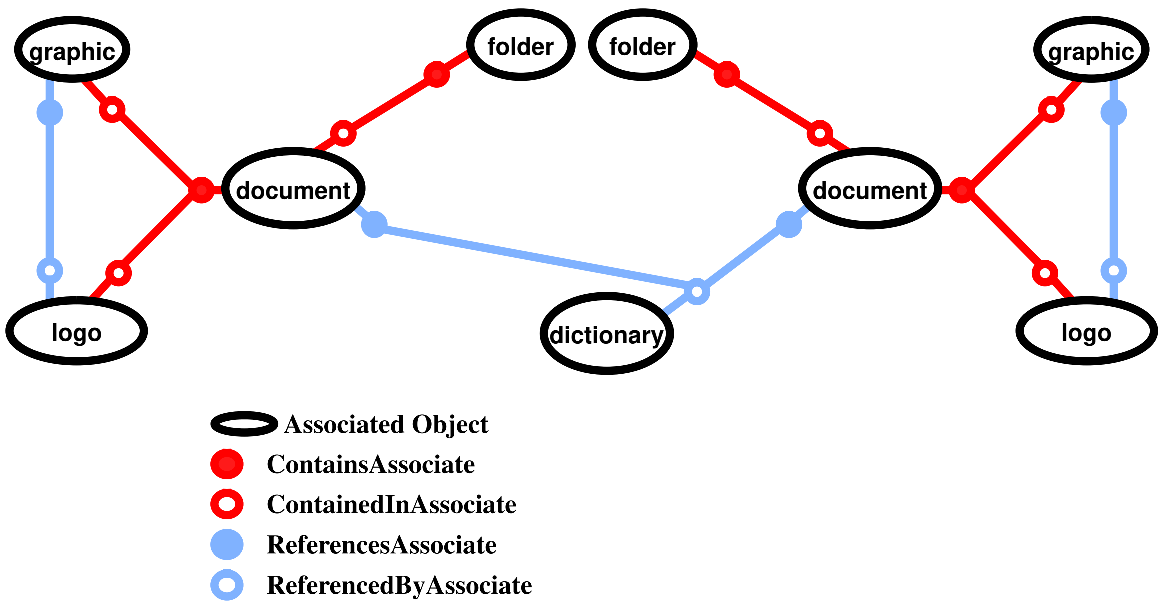

Figure 14 The Traversal Service Supports a Copy Operation

The resulting graph shows a copied folder which contains a copied document. When the document was copied, it revealed that it references the dictionary. The propagation semantics of reference with respect to copy is *shallow*; in this case, the original document and the copied document both share the same dictionary. Copies of the graphic and logo are also made since they are contained in the document.

Transaction Service Ensures Integrity

The Transaction service coordinates operations such as move and copy, and ensures that the operations either complete or abort. Transactions are "atomic," meaning indivisible and non-interfering. Transactions control concurrent operations on the graph in a concurrent environment. This control protects the graph against having two different concurrent activities interfere with each other.

Property Service Extends Nodes and Edges

The nodes and edges of the graph can be dynamically extended with property sets. This extension allows additional information to be dynamically added to the nodes and edges.

Naming Contexts Represent Locations

A naming context is used to represent a *location* in the distributed system. In particular, the Naming service functions as a finding service for *object factories.* An object factory is any object that creates another object.

Figure 15 shows a document being copied from one location to another. A naming context is given as an argument to the copy operation. The naming context represents the location. Using the naming context, the document finds a factory that is compatible with the original document. The factory creates a new document, then the original document and the factory engage in a private protocol to communicate the state of the original document to the new document.

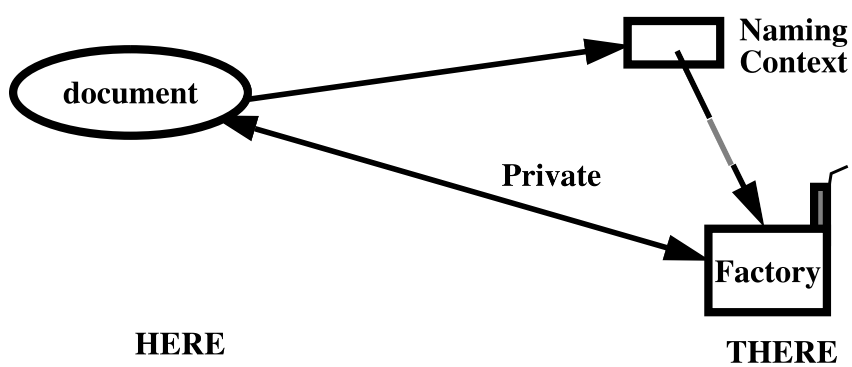

Figure 15 Locating Factories Using the Naming Service

The Application Objects Role in Graphs

Application objects also participate in the DOE graph model. In particular, the application object supports the Node interface; and the Node interface defines lifecycle operations. Figure 10 on page 13 shows an example.

PSM and ODF Provide Node Implementations

If an application is developed using the PSM and ODF, a standard implementation of the Node interface is provided. Application developers do not have to implement the remove, copy, move, externalize, and internalize operations. The standard implementation operates on the application's persistent state stored in the PSM.

Applications can always provide their own specialized implementation of the Node interface.

Object Services: Working in Concert

In summary, Project DOE provides a wide selection of Object Services, each providing features to support the integration of objects. Each service can be used individually according to application needs. However, the services have been designed to work in concert to support graphs of objects. In particular,

- The Association service is used to define the edges of the graph.
- The Event service is used to report changes in the graph.
- The Property service is used to dynamically add information to the nodes and edges of the graph.
- The Traversal service is used to coordinate the lifecycle operations (remove, copy, move, externalize and internalize) on the graph itself.
- The Transaction service is used to guarantee the integrity of operating on the graph.
- The Naming service is used to find object factories needed for the move and copy operations.
- The PSM and ODF support the implementation of DOE objects. If used by an application, default implementations of the lifecycle operations are provided automatically.

References

Object Management Group, *The Common Object Services Specification, Volume 1*, OMG Document Number 94.1.1, January, 1994.

SunSoft submission to Object Management Group, *Transaction Service Specification*, OMG Document Number 93.11.12, November 1993.

SunSoft submission to Object Management Group, *Externalization Service Specification*, OMG Document Number 93.11.11, November 1993.

SunSoft submission to Object Management Group, *Association Service Specification*, OMG Document Number 93.11.10, November 1993.

Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG Document Number 91.12.1, December, 1991.

SunSoft, Inc.
2550 Garcia Avenue
Mountain View, CA 94043

For more information, call 1 800 227-9227.

Printed in USA 4/94 93384-002