Differences Between Enterprise Objects Framework 1x and 2.0

A New Control Layer

The basic architecture of Enterprise Objects Framework 1x includes the interface layer, the access layer, and an underlying database. Enterprise Objects Framework 2.0 keeps these layers, but it also introduces a new *control* layer between the access and interface layers. In 1x, data flowed from the access layer to the interface layer on its way from the database to a user interface. In 2.0, data follows the same path through the Framework, but also flows through the new control layer on its way from the access layer to the interface layer. Figure 1 shows the difference in data flow between the versions.

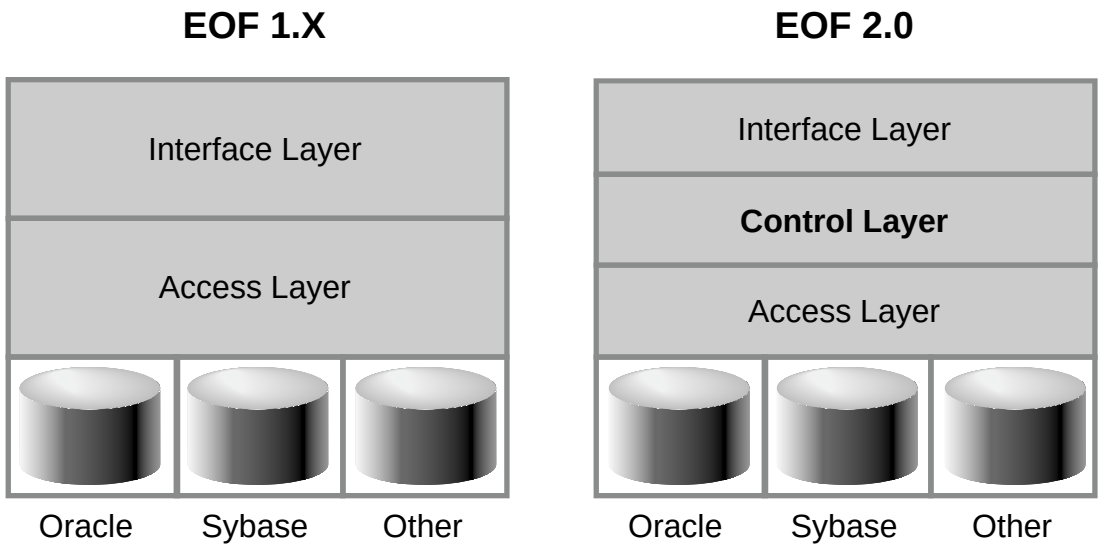

Figure 1. *Control Layer*

Motivation for the Control Layer

In Enterprise Objects Framework 1x, it is the responsibility of the EOController class to keep enterprise objects in sync with the user interface. To do so, EOControllers require you to make changes to enterprise objects by editing values in the user interface (using EOAssociations) or using the EOController method **setValues:forObject:**. If you change an enterprise object by sending it a message, the EOController won't update the user interface or save the change in the database.

To simplify changing enterprise object values programmatically, the control layer of Enterprise Objects Framework 2.0 incorporates the concept of change

notification. In 2.0, objects that need to know about changes to an enterprise object register as observers for change notifications. When an enterprise object changes, it posts a change notification, and registered observers are notified.

In addition, the EOController of Enterprise Objects Framework 1x is not available on PDO platforms. Because the controller is so tightly coupled with user interface, it can't be used in non-UI applications. Because operation buffering and undo are implemented in EOController, these features are unavailable on server platforms.

Enterprise Objects Framework 2.0 divides the functionality of the Enterprise Objects Framework 1x EOController between the interface layer and the control layer. While UI-related functionality remains the responsibility of the interface layer, non-UI functionality is provided by the control layer. As a result, server platforms have access to the non-UI functionality.

EOController does not exist in Enterprise Objects Framework 2.0. The interface layer class EODisplayGroup provides UI-related EOController functionality (such as transporting values between enterprise objects and the user interface), and the control layer class EOEditingContext provides the non-UI functionality (such as undo).

See Figures 11 and 12 at the end of this document to see how changes in Enterprise Objects Framework 2.0 have altered the architecture of database applications.

Change Notification

The change notification introduced by the control layer has the following effects on code you write.

- Enterprise objects should invoke the method **willChange** prior to altering their state. For example, set methods should invoke **willChange** before assigning new values.

```
- (void)setColor:(NSString *)aColor
{
    [self willChange];
    [color autorelease];
    color = [aColor copy];
}
```

- You can make changes to enterprise objects by sending messages to them directly.

For more information on change notification, see the chapter "Architectural Overview" in the Enterprise Objects Framework Developer's Guide, the class specification for EOEditingContext, and the protocol description for EOObserving.

EOEditingContext and EOObjectStore Classes

An EOEditingContext object manages a graph of objects fetched from an external store. EOEditingContexts watch for changes to their objects using change notification, and they record snapshots for object-based undo.

EOObjectStore is an abstract class whose subclasses act as a stores of objects for EOEditingContexts. Object stores are responsible for constructing and registering objects, servicing object faults, and committing changes made in an editing context. The basic relationship between the classes is depicted in Figure 2.

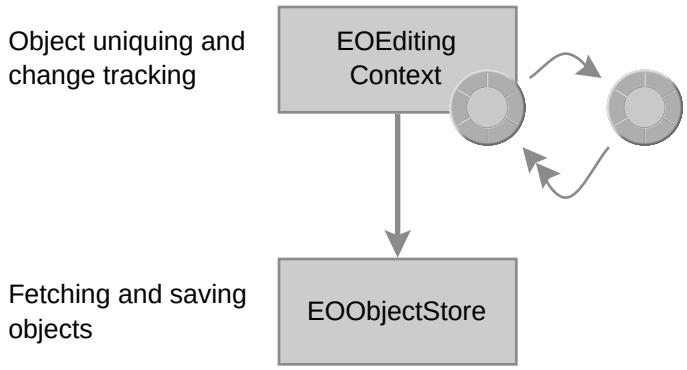

Figure 2. *Relationship Between EOEditingContexts and EOObjectStores*

In database applications, the relationship between the classes is a little more complex. The following scenarios are often combined:

- One or more "peer" EOEditingContexts can share a single object store.
- EOEditingContexts can be nested such that one EOEditingContext acts as an object store for another.
- Using an EOObjectStoreCoordinator, an EOEditingContext can maintain a single object graph consisting of objects from more than one database.

Figure 3 illustrates the general case.

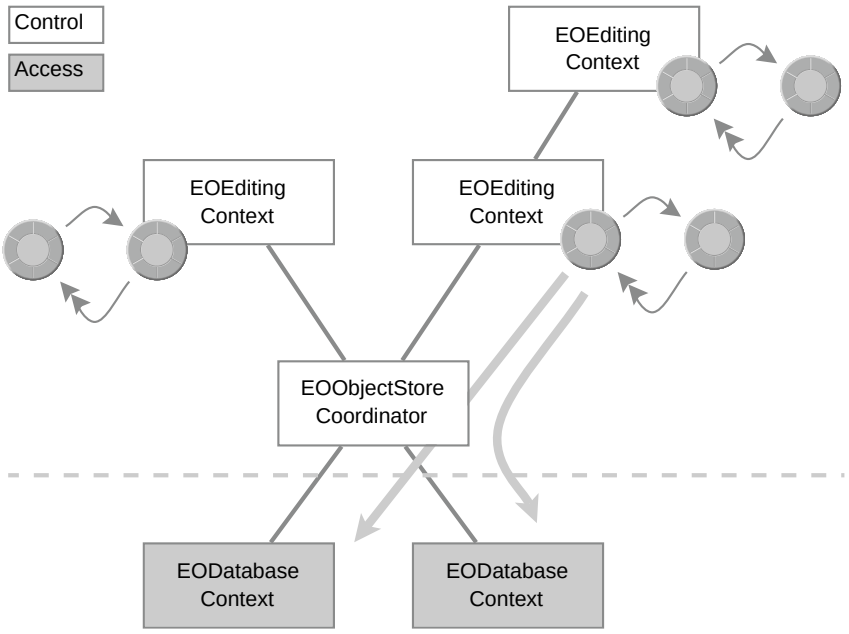

Figure 3. *Combined EOEditingContext Scenarios*

Note: EOObjectStoreCoordinator and EOCooperatingObjectStore are subclasses of EOObjectStore. EODatabaseContext, an access layer class, is a subclass of EOCooperatingObjectStore. As such, instances of all three classes can perform as object stores for EOEditingContexts.

EOQualifier

In Enterprise Objects Framework 1.1, qualifiers were based on SQL. As a result, qualifiers frequently introduced database dependence into application code and their use was restricted to fetching records from an SQL database. In Enterprise Objects Framework 2.0, qualifiers aren't based on SQL and they don't rely upon an EOModel. Thus, a single qualifier can be used to perform in-memory searches and to qualify database fetches.

There are several EOQualifier classes, each representing a different semantic. What you think of as a logical qualifier (for example, "name = 'fred' and age < 20") is represented by a tree of EOQualifier nodes. These trees can be combined using EOAndQualifiers and EOOrQualifiers as illustrated in Figure 4. EOQualifier has a method to parse a textual representation of a qualifier into a tree of EOQualifier nodes.

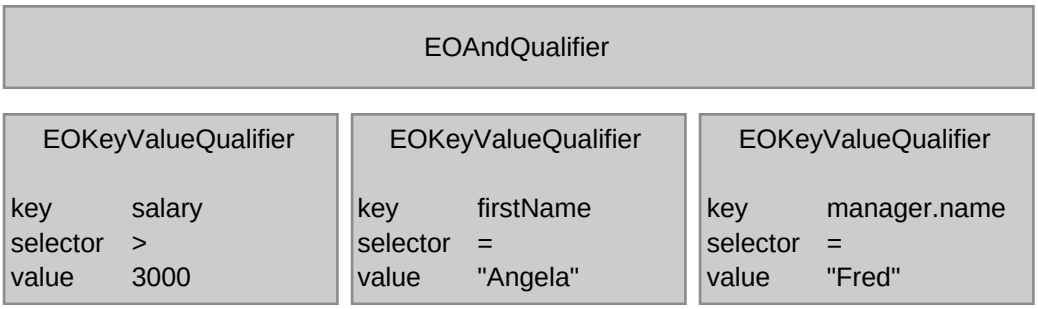

Figure 4. *EOQualifier tree for* **salary > 300 and firstName = "Angela" and manager.name = "Fred"**

All qualifier classes are public and can be extended with categories and subclasses. For more information on qualifiers, see the EOQualifier class specification and the class specifications for its subclasses:

- EOAndQualifier
- EOKeyComparisonQualifier
- EOKeyValueQualifier
- EONotQualifier
- EOOrQualifier
- EOSQLQualifier

Note: All the qualifier classes but EOSQLQualifier are declared in the EOControl framework. Because of its dependence on SQL, EOSQLQualifier is declared in the EOAccess framework.

Summary of Control Layer Classes

Classes from the 1x Access Layer

- EOFault
- EONull
- EOQualifier — Redesigned in 2.0. The 1x EOQualifier is functionally equivalent to the 2.0 EOSQLQualifier.

New Classes in the Control Layer

- EOAndQualifier
- EOClassDescription
- EOCooperatingObjectStore
- EODataSource
- EODelayedObserver
- EODelayedObserverQueue
- EODetailDataSource — Replaces the 1x EODetailDatabaseDataSource
- EOEditingContext
- EOFaultHandler
- EOFetchSpecification
- EOGlobalID
- EOKeyComparisonQualifier
- EOKeyValueQualifier
- EONotQualifier
- EOObjectStore
- EOObjectStoreCoordinator
- EOObserverCenter
- EOObserverProxy
- EOOrQualifier
- EOSortOrdering — Replaces the 1x EOAttributeOrdering and EOKeySortOrdering
- EOTemporaryGlobalID
- EOUndoManager

Access Layer Enhancements

Database Level

The Enterprise Objects Framework 2.0 database level architecture has the same basic design as it does in version 1x. As illustrated in Figure 5, the database level is comprised of three classes: EODatabase, EODatabaseContext, and EODatabaseChannel. However, each of these classes has somewhat different responsibilities and behavior than it does in Enterprise Objects Framework 1x.

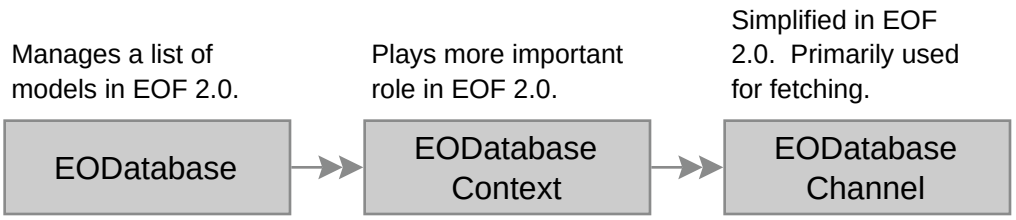

Figure 5. *Database Level Architecture*

EODatabase plays a role very similar to the one it plays in 1x. The most significant difference is that EODatabase objects now manage a list of models. In Enterprise Objects Framework 2.0, you access models through EODatabase objects instead of through EOAdaptors.

On the other hand, EODatabaseContext plays a much more significant role in 2.0 than it does in 1x. EODatabaseContext objects are responsible for analyzing graphs of enterprise objects and recording changes in the database. EODatabaseContext objects save changes by translating changes in the object graph to database operations. It then uses an EOAdaptorChannel to perform the operations.

Accordingly, the role of EODatabaseChannel is smaller in Enterprise Objects Framework 2.0. It no longer provides methods such as **insertObject:**, **updateObject:**, and **deleteObject:**. Rather, EODatabaseChannel objects are primarily used by an EODatabaseContext for fetching enterprise objects. In fact, EODatabaseContexts maintain a list of EODatabaseChannels for fetching. Consequently, fetching conflicts due to "busy channels" are more easily avoided in Enterprise Objects Framework 2.0.

The following sections describe the impact database level enhancements have on development tasks.

Relationship Updating

In Enterprise Objects Framework 1x, relationships are essentially read only. To update a relationship between two rows in the database, developers must write code to track and update foreign keys. In Enterprise Objects Framework 2.0, EODatabaseContext objects recognize changes to relationships and automatically update foreign keys.

For example, assume that a new employee—Jane—is added to the engineering department. As illustrated in Figure 6, an Employee object for Jane is added to the **employees** array in the Department object representing engineering. Similarly, Jane's **department** variable is assigned to the engineering Department object. When an EODatabaseContext analyzes the Department-Employees object graph, it detects the new relationships, translates the changes to database operations, and performs the database operations using an EOAdaptorChannel. As a result, the **DeptID** foreign key in the Employee row for Jane is updated with the **DeptID** value for the engineering department.

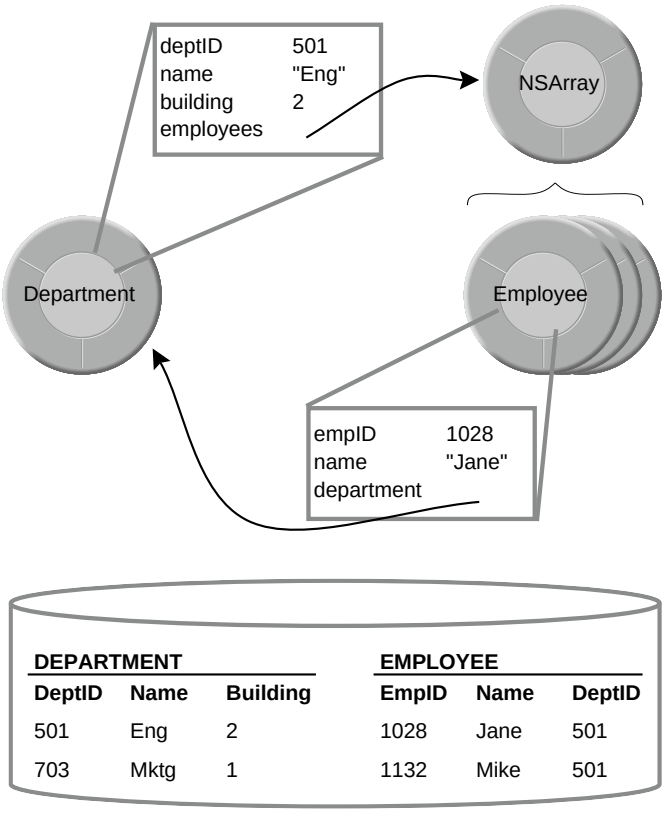

Figure 6. *Pointer-Based Relationship Manipulation.*

Because Enterprise Objects Framework 2.0 automatically updates foreign key values when relationships change, you don't need to make primary and foreign key attributes class properties of enterprise objects. Usually, the only time you make a primary or foreign key attribute a class property is when the value is meaningful to the user and must be displayed in the user interface.

Updating Flattened Attributes

Using flattened attributes, an enterprise object can contain values from more than one table in a database. In Enterprise Objects Framework 1x, flattened attributes are read only. In Enterprise Objects Framework 2.0, flattened attributes can be modified, and changes to them are automatically updated in the database.

For example, the **aCustomer** object in Figure 7 contains flattened attributes from the Address table. Changing **aCustomer**'s street attribute has the effect of updating the corresponding database row in the Address table.

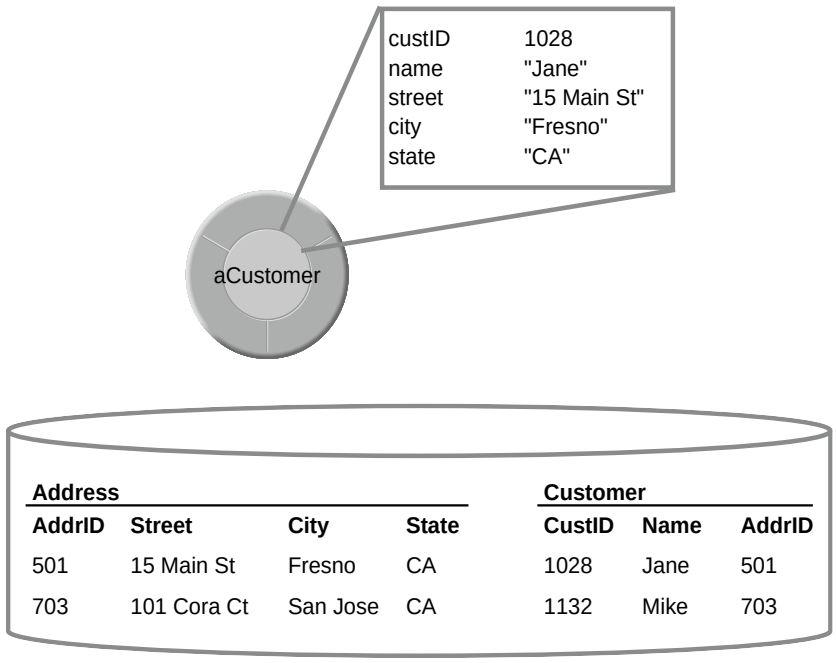

Figure 7. *Updating Flattened Attributes.*

Enterprise Object Inheritance

To improve support for inheritance, Enterprise Objects Framework 2.0 allows you to capture the object hierarchy in a model. Developers can specify how class hierarchies are mapped to entities and how entities are mapped to tables in the database. Enterprise Objects Framework 2.0 automatically manages the selection, insertion, updating, and deletion of objects. For more information, see the chapter "Answers to Common Design Questions" in the Enterprise Objects Framework Developer's Guide.

Accessing Multiple Databases

In Enterprise Objects Framework 2.0, you can create relationships that span multiple databases. For example, you can define a relationship from an entity in a Sybase database to an entity in an Oracle database. The database level in Enterprise Objects Framework 2.0 automatically assembles an object graph with enterprise objects fetched from multiple databases. In addition, the database level saves subsequent changes to the appropriate databases.

Note: You can't flatten attributes, map inheritance hierarchies, or define flattened many-to-many relationships across databases.

The ability to mix objects from multiple databases in a single object graph is enabled by the control layer architecture. The EOObjectStoreCoordinator and EOCooperatingObjectStore classes in the control layer specify the way objects from different object stores can be mixed in the same object graph. As a subclass of EOCooperatingObjectStore, EODatabaseContext provides the ability to mix objects from multiple databases.

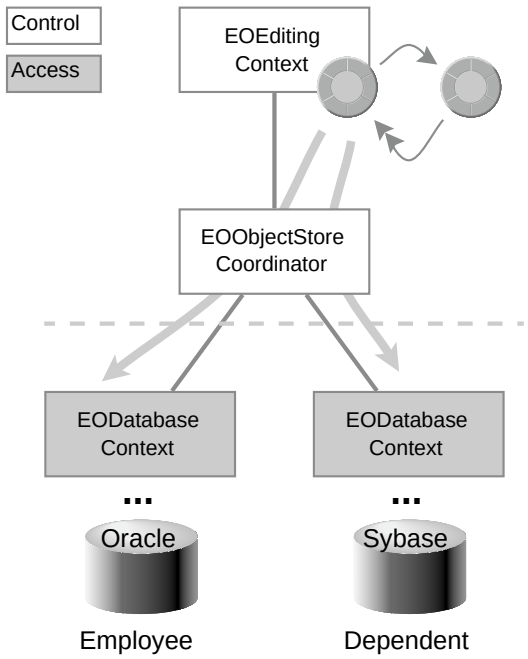

Figure 8. *Accessing Multiple Databases*

EOModeler and the modeling classes have also been updated to support multiple database access. See the chapter "Using EOModeler" in the Enterprise Objects Framework Developer's Guide for more information about creating relationships between entities in different databases.

Referential Integrity

Enterprise Objects Framework 2.0 extends support for relational integrity rules from the database to the object graph. In EOModeler, you can specify optionality and delete rules for relationships. Optionality rules can be enforced when an object graph is validated and/or saved. See the chapter "Using EOModeler" in the Enterprise Objects Framework Developer's Guide for more information.

Stored Procedures

Enterprise Objects Framework 2.0 includes a new EOStoredProcedure class that defines a general API for encapsulating vendor specific stored procedures. It provides a mechanism for mapping keys to input and output parameters. In EOModeler, you can assign stored procedures to entities for the following operations:

- Fetching all the objects for the entity
- Fetching a single object by its primary key
- Inserting a new object
- Deleting an object
- Generating a new primary key

Number Conversion

In Enterprise Objects Framework 1x, numeric database values are stored in NSNumber objects. As a result, values are limited to double precision and operations are inexact. With Enterprise Objects Framework 2.0, you can use the Foundation class NSDecimalNumber to avoid the problem illustrated in Figure 9.

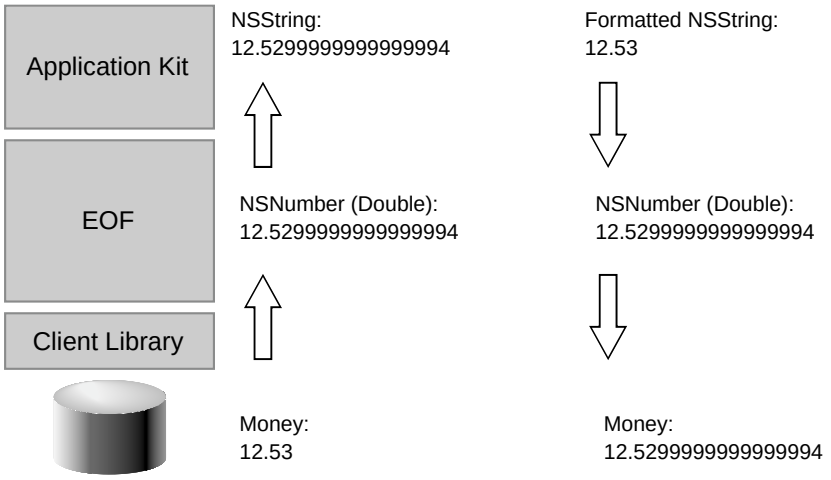

Figure 9. *Using NSNumber*

Adaptor Level

Enterprise Objects Framework 2.0 provides four adaptors: updated Oracle and Sybase adaptors, a new Informix adaptor, and on Windows NT, a new adaptor for ODBC-compliant databases.

The Oracle adaptor includes the following enhancements:

- Based on the most recent production version of the Oracle client libraries.
- Provides shared SQL pool support by using bind variables in generated SQL.
- Constructs relationships from metadata in the database server.
- Supports stored procedure result sets and parameters.

The Sybase adaptor includes the following enhancements:

- Based on CT-Lib client libraries for System 10 servers.
- Uses array fetching for faster access times.
- Supports RPC-based stored procedure invocations.

In addition to improved support for specific database servers, Enterprise Objects Framework 2.0 makes significant improvements to the adaptor level by enhancing the abstract classes that implement the bulk of adaptor functionality. The Enterprise Objects Framework 2.0 adaptor architecture has the same basic design as it does in version 1x. As illustrated in Figure 10, an adaptor is comprised of three classes: EOAdaptor, EOAdaptorContext, and EOAdaptorChannel. Each of these classes has essentially the same responsibilities and behavior as it does in Enterprise Objects Framework 1x.

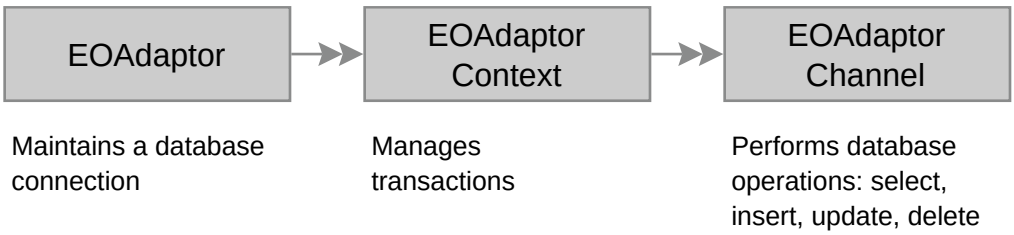

Figure 10. *Adaptor Architecture*

The following sections describe the impact adaptor level enhancements have on development tasks.

Writing Custom Adaptors

Although the adaptor architecture is relatively unchanged, the adaptor level classes take advantage of enhancements in other areas of the framework that make writing custom adaptors easier in Enterprise Objects Framework 2.0.

- The adaptor API uses the new EOQualifier class (now in the Control layer). As a result, it is much easier to write adaptors for non-SQL data sources.
- The EOSQLExpression class has been simplified: it's more heavily based on SQL syntax and the responsibilities of subclasses are more straightforward.
- Abstract adaptor classes use new API in the EOAttribute class for specifying custom values so it's easier to support custom values in custom adaptors.

Accessing Database Metadata

Enterprise Objects Framework 2.0 provides the following enhancements for accessing database metadata:

- The EOSQLExpression class contains new API that supports schema generation.
- The EOAdaptor class provides API for mapping internal (Objective-C) names and types to external (database) names and types, facilitating translation of models from one adaptor to another.
- The EOAdaptorChannel class provides API for constructing stored procedure objects from metadata in the database server.

These enhancements are particularly useful for extending EOModeler. See the EOModeler Enhancements section for more information.

Modeling Classes

Enterprise Objects Framework 2.0 provides the following enhancements to the modeling classes:

- The custom value support in EOAttribute has been extended so you can specify a factory (creation) method, a conversion method, and an intermediate type (NSData, NSString, or bytes). For more information, see the chapter "Using EOModeler" in the Enterprise Objects Framework Developer's Guide and the class specification for EOAttribute.
- The new EOStoredProcedure class extends model support for stored procedures.
- Model files are incrementally loaded.
- The new EOModelGroup class manages all the models used by an application or framework to support cross-database relationships.

Access Layer Gotchas

If you are used to developing applications with Enterprise Objects Framework 1x, watch out for the following gotchas when you begin developing in Enterprise Objects Framework 2.0.

- EOGenericRecord no longer provides the methods **objectForKey:** and **setObject:forKey:**. Use **valueForKey:** and **takeValue:forKey:** instead.
- Model lookup is now project and framework-based. The search path **~/Library/Models**, **/LocalLibrary/Models**, **/NextLibrary/Models** is no longer used. Instead, add all relevant model files to application and framework projects.
- EOAdaptor no longer provides the method **model** for accessing an EOModel. Instead, use the EODatabase method **models** to return a list EOModel objects.
- Many of the 1x methods that return BOOL values have been modified to return void. Instead of indicating failure by returning NO, 2.0 methods often raise exceptions to communicate information about the type of failure.

Summary of Access Layer Additions and Deletions

Updated Access Layer Classes

- EOAdaptor
- EOAdaptorChannel
- EOAdaptorContext
- EOAttribute
- EODatabase
- EODatabaseChannel
- EODatabaseContext
- EODatabaseDataSource
- EOEntity
- EOGenericRecord
- EOJoin
- EOModel
- EORelationship
- EOSQLExpression

Removed Access Layer Classes

• EOAttributeOrdering	Replaced with control layer's EOSortOrdering
• EODetailDatabaseDataSource	Replaced with control layer's EODetailDataSource.
• EOFault	Moved to control layer.
• EOFormatExpression	Removed from 2.0. Superseded by EOSQLExpression.
• EOKeySortOrdering	Replaced with control layer's EOSortOrdering.
• EONull	Moved to control layer.
• EOQualifier	Moved to control layer.
• EOQuotedExpression	Removed from 2.0. Superseded by EOSQLExpression.

New Access Layer Classes

- EOAdaptorOperation
- EODatabaseOperation
- EOEntityClassDescription
- EOModelGroup
- EOSQLQualifier — Functionally equivalent to the 1x EOQualifier.
- EOStoredProcedure

Interface Layer Enhancements

In Enterprise Objects Framework 2.0, the interface layer plays a reduced role. The control layer provides much of the functionality formerly provided by EOController. To reflect its new role, the EOController class has been renamed to EODisplayGroup.

EODisplayGroup

EODisplayGroup provides the following UI-related functionality:

- Maintain a collection of enterprise objects and a selection of zero, one, or more of those objects.
- Notify associations of changes to enterprise objects.
- Invoke validation methods.
- Perform object sorting using the new EOSortOrdering class.
- Perform object filtering using the new EOQualifier class.

EOAssociation

In Enterprise Objects Framework 2.0, EOAssociations incorporate the following enhancements:

- *Support for validation*. An association has the ability to notify its EODisplayGroup when a user begins editing, permit a user to leave an edited field only if its EODisplayGroup approves the change, and pass formatter validation errors to its EODisplayGroup so the EODisplayGroup can handle the error.

- *Support for multi-aspect associations*. For example, EOPopUpAssociation has a **values** aspect and a **selectedObject** aspect. You can bind the **titles** aspect to one enterprise object key and the **selectedObject** aspect to a different enterprise object key.

For more information on associations, see the chapters "Creating an Enterprise Objects Framework Project" and "Overview of the Examples" in the Enterprise Objects Framework Developer's Guide. You can also refer to the class specifications for the following association classes:

- EOAssociation
- EOActionAssociation
- EOActionCellAssociation
- EOActionInsertionAssociation
- EOColumnAssociation
- EOControlAssociation
- EODetailSelectionAssociation
- EOGenericControlAssociation
- EOMasterDetailAssociation
- EOMasterPeerAssociation
- EOPickTextAssociation
- EOPopUpAssociation
- EORadioMatrixAssociation
- EOTableViewAssociation
- EOTextAssociation

Formatters

The interface layer of Enterprise Objects Framework 2.0 takes advantage of formatter objects provided by the Application Kit. You can assign formatters to any NSCell. Cells use formatters to do the following:

- Format object values for display.
- Parse strings into object values.

NSTableView

The interface layer of Enterprise Objects Framework 2.0 also takes advantage of a redesigned table view. The table view class, renamed NSTableView, is now provided by the Application Kit.

- It is implemented in four public classes: NSTableView, NSTableColumn, NSTableHeaderView, and NSTableHeaderCell.
- NSTableView uses cells to perform drawing. You can use any cell class—NSTextCell, NSImageCell, and so on—in a table view.
- The interface between NSTableView and its data source has been simplified.

Interface Layer Gotchas

If you are used to developing applications with Enterprise Objects Framework 1x, watch out for the following gotchas when you begin developing in Enterprise Objects Framework 2.0.

- The **saveToObjects:** and **saveToDataSource** methods provided by EOController aren't provided by EODisplayGroup. The EODisplayGroup class doesn't provide buffering of edits or operations. You can use nested EOEditingContexts to get the effect of buffering edits, and operation buffering is handled automatically by an EOEditingContext.
- When you create an association in Interface Builder in Enterprise Objects Framework 2.0, you drag a connection from the user interface control to the EODisplayGroup.
- The EOAssociation API has changed considerably. Communication between an EODisplayGroup and its EOAssociations is much different than that between an EOController and its EOAssociations. EOAssociations observe their EODisplayGroups using the EOObserver protocol. As a result, associations are updated asynchronously.

Summary of Interface Layer Additions and Deletions

Updated Interface Layer Classes

- EOAssociation
- EOActionCellAssociation
- EOColumnAssociation
- EOControlAssociation
- EOPopUpAssociation
- EOTextAssociation

Removed Interface Layer Classes

• EOBrowserAssociation	See the EOBrowserAssociation example in **NextDeveloper/Examples/EnterpriseObjects**.
• EOButtonAssociation	Replaced with EOActionAssociation.
• EOImageAssociation	Use the EOControlAssociation instead.
• EOMatrixAssociation	
• EOQualifiedAssociation	Replaced with EOMasterDetailAssociation and EOMasterPeerAssociation.
• EOController	Replaced with EODisplayGroup and EOEditingContext.

New Interface Classes

- EOActionAssociation
- EOActionInsertionAssociation
- EODetailSelectionAssociation
- EOGenericControlAssociation
- EOMasterDetailAssociation
- EOMasterPeerAssociation
- EOPickTextAssociation
- EORadioMatrixAssociation
- EOTableViewAssociation
- EODisplayGroup

Enterprise Object Enhancements

Key-Value Coding and Relationship Accessor Methods

Enterprise Objects Framework 2.0 includes the following enhancements to the key-value coding methods:

- The primitive methods—**valueForKey:** and **takeValue:forKey:**—are single-key. The multi-key versions, **valuesForKeys:** and **takeValues:fromDictionary:**, still exist, but their default implementation invokes the single-key methods.
- When enterprise object values are null in the database, **nil** is passed as an argument to set methods instead of EONull.
- Two new methods, **valueForKeyPath:** and **takeValue:forKeyPath:**, have been added for accessing variables using a key path. For example, suppose an employee entity has a relationship to its department. Assuming employees have a **department** instance variable that points to a department object and department objects have a **departmentName** instance variable, you can get and set the name of the employee's department using the following statements:

  ```
  [employee valueForKeyPath:@"department.departmentName"];
  [employee takeValue:newDepartmentName
          forKeyPath:@"department.departmentName"];
  ```

  However, you rarely invoke the key path methods directly. Instead of the statements above, you are more likely to write the following lines of code:

  ```
  [[emp department] deptName];
  [[emp department] setDeptName:newName];
  ```

- The default implementations of **valueForKey:** and **takeValue:forKey:** raise exceptions when you provide a key that doesn't correspond to a method or instance variable in the receiver. To implement a more permissive policy, you can override the methods **handleQueryWithUnboundKey:** and **handleTakeValue: forUnboundKey:.**
- The method **unableToSetNilForKey:** is provided to set policy for an attempt to assign **nil** to an instance variable that requires a C scalar type.

In addition to the key-value coding methods, new relationship accessor methods have been introduced in Enterprise Objects Framework 2.0. As with key-value coding methods, a category of NSObject defines default implementations of the relationship accessor methods for manipulating relationship values:

- addObject:toPropertyWithKey:

- removeObject:fromPropertyWithKey:
- addObject: toBothSidesOfRelationshipWithKey:
- removeObject: fromBothSidesOfRelationshipWithKey:

For more information on key-value coding and relationship accessor methods, see the NSObject Additions specification as well as the class specification for EOClassDescription.

Validation

EOF 2.0 incorporates a validation mechanism for enterprise objects. A category of NSObject uses the EOClassDescription class to provide default implementations of the following validation methods:

- validateValue:forKey:
- validateForSave
- validateForDelete
- validateForInsert
- validateForUpdate

The above enterprise object methods are invoked automatically by framework components such as EODisplayGroup and EOEditingContext. For more information, see the NSObject Additions specification and the class specification for EOClassDescription. For more information on the process of validating enterprise objects, see the chapter “Overview Of The Examples” in the Enterprise Objects Framework Developer’s Guide.

Enterprise Object Initialization

Enterprise Objects Framework 2.0 replaces the optional initialization methods defined in Enterprise Objects Framework 1x

- Enterprise Objects Framework 2.0 replaces **initWithPrimaryKey:entity:** with **initWithEditingContext:classDescription:globalID:.** Like **initWithPrimaryKey:entity:**, **initWithEditingContext:classDescription:globalID:** is used to initialize enterprise objects created by the framework. If the enterprise object class doesn't implement this method, **init** is used instead.

- Enterprise Objects Framework 2.0 replaces **awakeFromDatabaseChannel:** with **awakeFromFetchInEditingContext:**. This method is sent to an enterprise object immediately after the object has been created from a database row.

Enterprise Objects Framework 2.0 includes a new initialization method as well.

- **awakeFromInsertionInEditingContext:** is an optional method sent to newly created enterprise objects. This method is invoked in objects that aren’t being

initialized with data from an object store. This method provides an opportunity to assign default values.

For more information about these methods, see the class specification for EOClassDescription and the NSObject Additions specification.

Enterprise Object Gotchas

If you are used to developing applications with Enterprise Objects Framework 1x, watch out for the following gotchas when you begin developing in Enterprise Objects Framework 2.0.

- The optional enterprise object method **prepareForDataSource** is not supported in Enterprise Objects Framework 2.0. It's replaced by the validation mechanism described above.
- Enterprise objects should invoke their **willChange** method in set methods before making changes.
- When enterprise object values are null in the database, **nil** is passed as an argument to set methods instead of EONull.
- Enterprise objects don't have to declare instance variables for primary key and foreign key values. The framework manages primary and foreign keys automatically. The default mechanism for assigning unique primary keys is provided with the EOAdaptorChannel's **primaryKeyForNewRowWithEntity:**. If you need to provide a custom mechanism for assigning primary keys, you can implement the EODatabaseContext delegate method **databaseContext:newPrimaryKeyForObject:entity:**. If you use either of these two mechanisms, you don't need to store the primary key in your enterprise object.
- The optional enterprise object method **classForEntity:values:** has been replaced with the EOModelGroup's delegate method **entity:classForObjectWithGlobalID:**. If you implemented **classForEntity:values:** in any of your enterprise object classes, you'll need to reimplement it.

EOModeler Enhancements

The new version of EOModeler features several user interface improvements.

- *New table mode*. In addition to the browser mode, EOModeler 2.0 provides a table mode for displaying models. The table mode is capable of displaying more information than the browser mode, and it is editable. For more information, see the chapter "Using EOModeler" in the Enterprise Objects Framework Developer's Guide.
- *Multi-pane inspectors*. The inspectors in EOModeler 2.0 have multiple panes. For example, the Entity Inspector has four panes in which to display entity characteristics: Entity, Advanced Entity, Stored Procedure, and UserInfo.
- *UserInfo inspectors*. EOModeler 2.0 provides user info inspectors for models, entities, attributes, and relationships.

In addition, EOModeler incorporates the following extensions to support Enterprise Objects Framework 2.0 features:

- Simple database creation
- Support for inheritance
- Specification of stored procedures for accessing entities

Extensibility

EOModeler is extensible in Enterprise Objects Framework 2.0. You can add custom inspectors, consistency checking, and menu items.

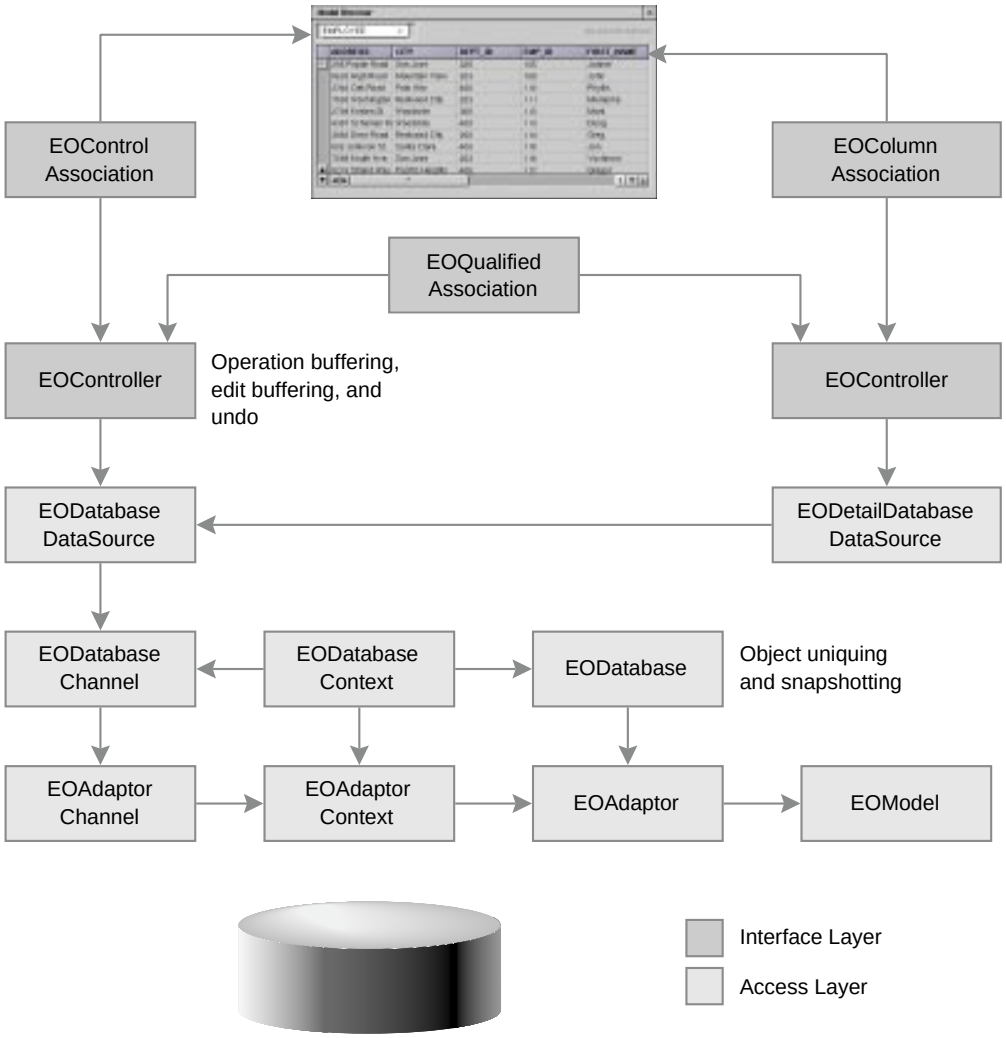

Figure 11. *Architecture of an Enterprise Objects Framework 1x Application*

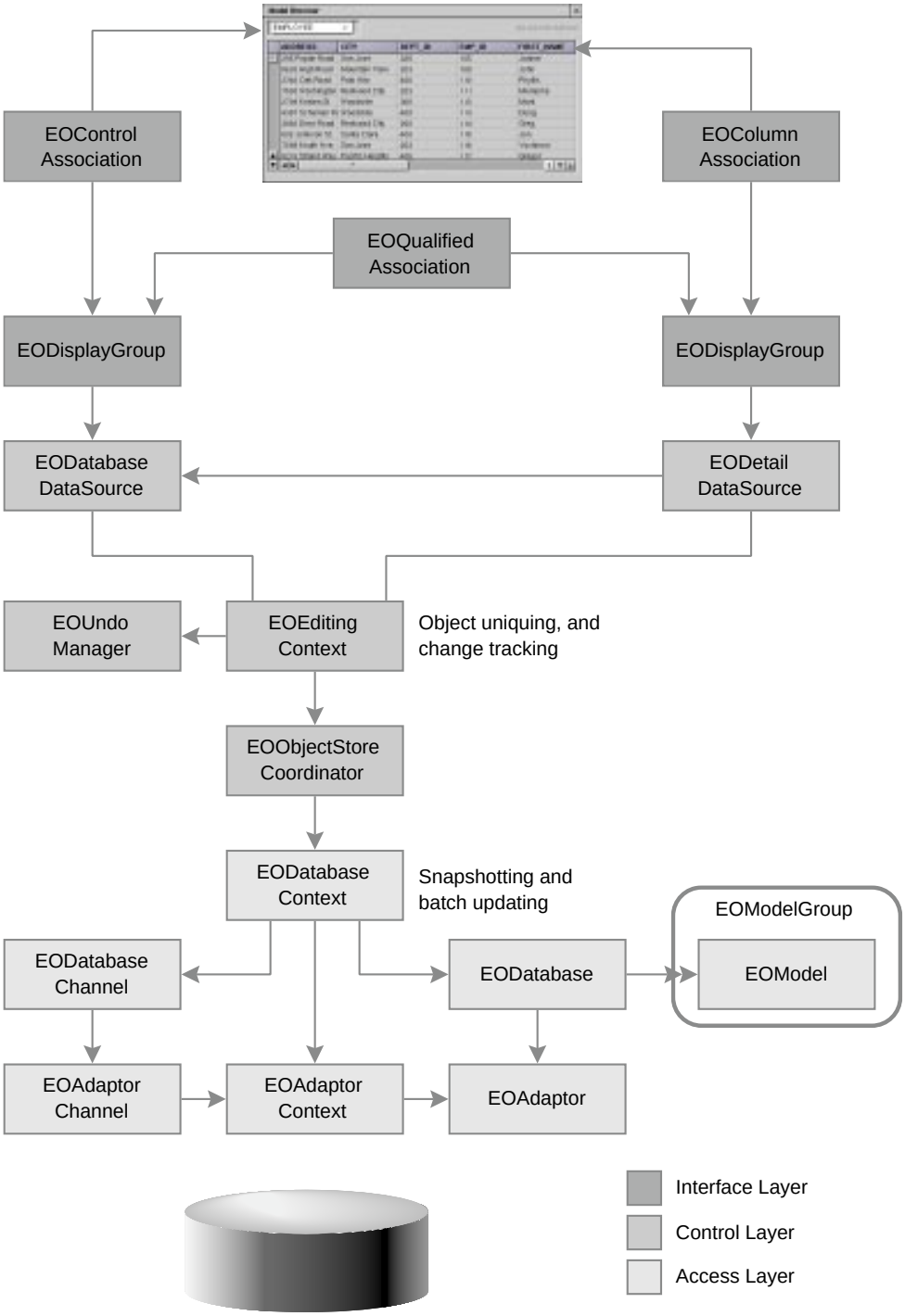

Figure 12. *Architecture of an Enterprise Objects Framework 2.0 Application*