Business Object Facility

Prepared by: Business Object Architecture team

Contact:
Tom Digre
email: digre@ti.com
phone: 214 575-5272
fax: 214 575-2866
mail address: Texas Instruments Incorporated
6620 Chase Oaks Blvd, MS 8417
Plano, Texas 75023

Table of Contents

1. Foreword

This document was prepared by the Business Object Architecture (BOA) team as part of the TI SC Enterprise Framework strategy. BOA is the information technology architecture used to implement the TI SC Enterprise Framework Business Object Model. A key element of BOA is the Business Object Facility, which is the focus of this document.

OMG has issued an RFP for “Common Business Objects and Business Object Facility”. BOA has adopted this RFP as a statement of requirement for a key element of the architecture. The remainder of this document has been prepared and formatted as a response to the RFP. The BOA team, which includes members from many internal TI development organizations, is not authorized to develop a commercial product and consequently can not directly respond to the RFP. However, this document will facilitate:

- Identifying, structuring, and aligning our specifications consistent with the OMG roadmap.
- Exerting influence on vendors responding to the RFP, particularly with respect to ensuring that TI requirements are being satisfied.
- Evaluating responses to the RFP. Our experience in transforming requirements to OMG specifications, and representing those specifications in the format required for the RFP response, will enable us to quickly evaluate and articulate relative merits and shortcomings of RFP responses.

Commercial products based on the OMG-adopted Business Object Facilty specification will not be available until at least 1998. Furthermore, it may not be feasible for our internal development organizations to build functionality completely conformant with this specification. Consequently, this specification will be considered “abstract”, a well defined technical objective which can not be completely satisfied with current product offerings. To address short term requirements, “concrete” specifications will be written, guided by this abstract specification, but based on readily available purchased or in-house technology.

2. SPECIFICATION DESCRIPTION

2.1 Rationale

This specification defines a **Business Object Facility** which satisfies requirements enumerated in OMG's "Common Business Objects and Business Object Facility" RFP**.** "Figure 2-1 Business Application Architecture" illustrates how that RFP depicted the architectural relationship between the Business Object Facility, other OMG infrastructure components, and Business Objects.

The approach taken by this specification is to leverage existing (and anticipated) OMG specifications through selective composition of defined CORBAservices, with minimal specialization.

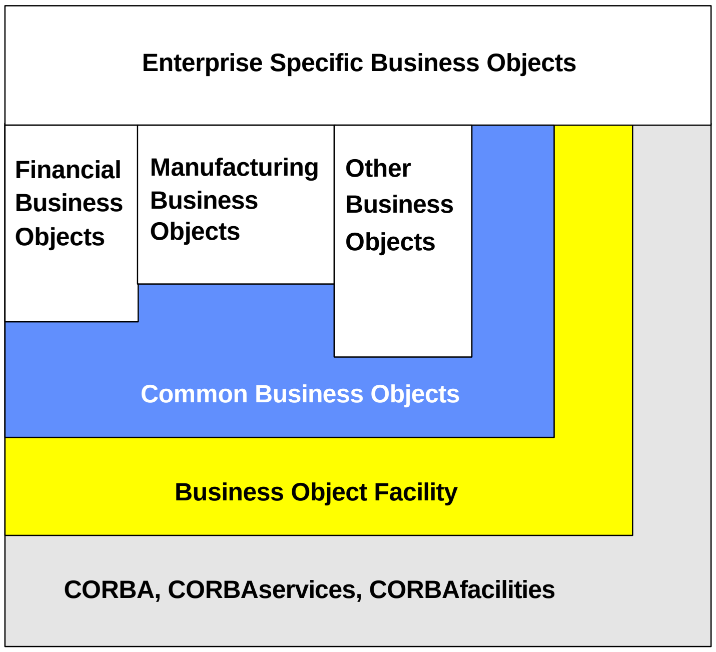
Enterprise Specific Business Objects
Financial Business Objects
Manufacturing Business Objects
Other Business Objects
Common Business Objects
Business Object Facility
CORBA, CORBAservices, CORBAfacilities

Figure 2-1 Business Application Architecture

2.1.1 Goals and Objectives

The primary objectives of this specification are to facilitate:

- Simplicity in the development, deployment, change and use of business objects for application users and developers.
- Interoperability of independently developed business objects.
- Implementation of business object configurations as "plug and play business components" of the information system.
- A direct correspondence between the business object model, in understandable business terms, and the business components of the information system.
- Isolation of infrastructure and business objects from tool or presentation technologies.
- Isolation of technology from business logic.
- Transactional integrity across distributed business objects.
- Direct coupling between a Business Object and its defining meta business object.
- Consistency of business object specification across the development lifecycle.
- Flexible and dynamic business object models.
- Enforced business semantics.

It is anticipated that implementations derived from this specification will have a substantial impact on ability of business systems to adapt to rapidly changing business requirements and on the cost and effectiveness of information systems that are increasingly critical to business priorities.

The Business Object Facility is just one element of an overall component-based architecture, and just one stage in the transition from legacy to plug and play business components, as shown in "Figure 2-2 Component Based Architecture Reference Model". This reference model is with respect to an architecture built upon layers of technology successively enabling capability on the path towards plug and play business components. The architectural structure is motivated by OMG-defined architectures **[OMG 95.01.02, OMG 93.12.29, OMG 95.01.12]**.

The technology layers of the architecture include:

- **Platforms**, operating systems, database management systems, networks, and other fundamental software infrastructure components. Applications built directly on this layer are typically characterized as monolithic and centric.
- **ORB**(Object Request Broker). Using any of a variety of middleware communications mechanisms, this layer of technology typically enables construction of client/server applications founded on syntactic interoperability. CORBA is an example implementation **[OMG 93.12.29]**.
- **Object Services**. A set of fundamental services necessary to implement distributed applications. The services encompass distributed concepts of object relationships, concurrency, persistence, transactions, etc. **[OMG 95.01.12]**
- **Facilities**. A set of tools, common business objects, conventions, and component interoperability services necessary to support integrity of business semantics from time of business requirement specification to implementation. More specifically, this architectural layer includes the Business Object Facility, Meta Object Facility, and Analysis & Design Facility.

- **Vertical Domains**. For each business domain, a set of plug and play Business Application Components compliant with industry-standard specifications. Industry standards are expected to evolve from activities of OMG Special Interest Groups and industrial consortia such as SEMATECH and OAG.

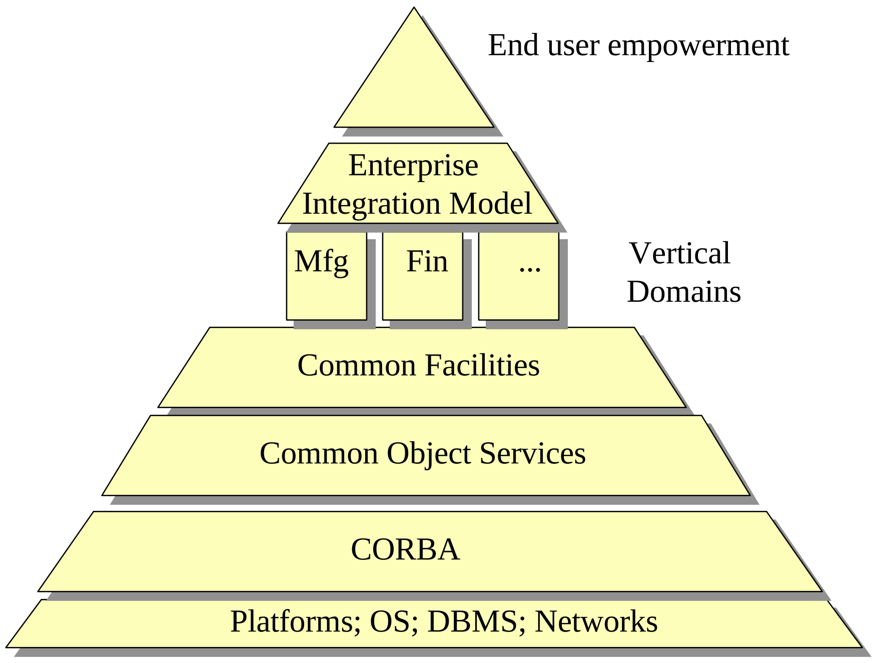

Figure 2-2 Component Based Architecture Reference Model

Each major element of the Component Based Architecture Reference Model represents a level of achievement towards the objective of a component based architecture. The baseline level of comparison is that of “monolithic applications”, which are characterized by applications tightly coupled with a platform (including DBMS, Operating System, Transaction Processing Monitor), driven by an exposed data model (as opposed to encapsulated information), having limited enforcement of consistency with business requirements, and having limited interoperability with other applications. With reference to this baseline, the levels of achievement, or objectives, are (also see “Figure 2-3 Architectural Objectives”):

- **Client/server applications.**   (ORB). The following features characterize components attaining this technology objective:
    - Implementation and location transparency. Clients are unaware of the location of server software components or their implementation details. Component implementation and location can change dynamically, without any modification to the client.
    - Universally accessible on federated enterprise ORBs.
    - Physical black-box encapsulation of related services.

    - Programming language independent. All component public interfaces are defined in IDL(Interface Definition Language).
    - Extensible, reusable specifications. Interface specifications can be inherited.
- **Distributed Applications.** (Object Services). The following features characterize components attaining this technology objective (in addition to characteristics of client/server applications):
    - Information Management technology transparency. Business application components are unaware of the location of data storage components or their implementation details. DBMS implementation and location can change dynamically (plug and play), without any modification to business application components.
    - Heterogenous Information management services are universally and transparently accessible.
    - Applications are fully peer to peer; they can be distributed across multiple heterogeneous platforms and underlying information management systems.
    - Application transactions (units of work) can be distributed and nested across multiple heterogenous platforms.
    - Asynchronous events can be communicated between arbitrary combinations of business application components.
    - Enterprise resources can be concurrently accessed and controlled by arbitrary combinations of business application components.
    - Business Application components can be dynamically related to any other business application component while maintaining full and transparent referential integrity, support for cascaded object deletion, arbitrary query, and without breaking encapsulation.
    - Dynamic, transparent addition and maintenance of properties to business application components.
    - Transparent Query and navigation across dynamic relationships and properties.
- **Semantic Interoperability.** (Common Facilities). The following features characterize components attaining this technology objective (in addition to characteristics of distributed applications):
    - Rigorous Semantics which enable plug-and-play, extensibility integrity, comprehension and control by business user.
    - Fully traceable life cycle, from business process engineering through deployment and continuing to final disposition.
- **Plug & Play Business Application Components.** (Vertical Domains). The following features characterize business application components attaining this technology objective (in addition to characteristics of Semantically Interoperable Components):
    - Fully interchangeable business application components available from multiple vendors.
    - Highly competitive business application component market; wide range of price/quality/features; niche market business application component providers.

- Standard, formal Business Application Component certification.

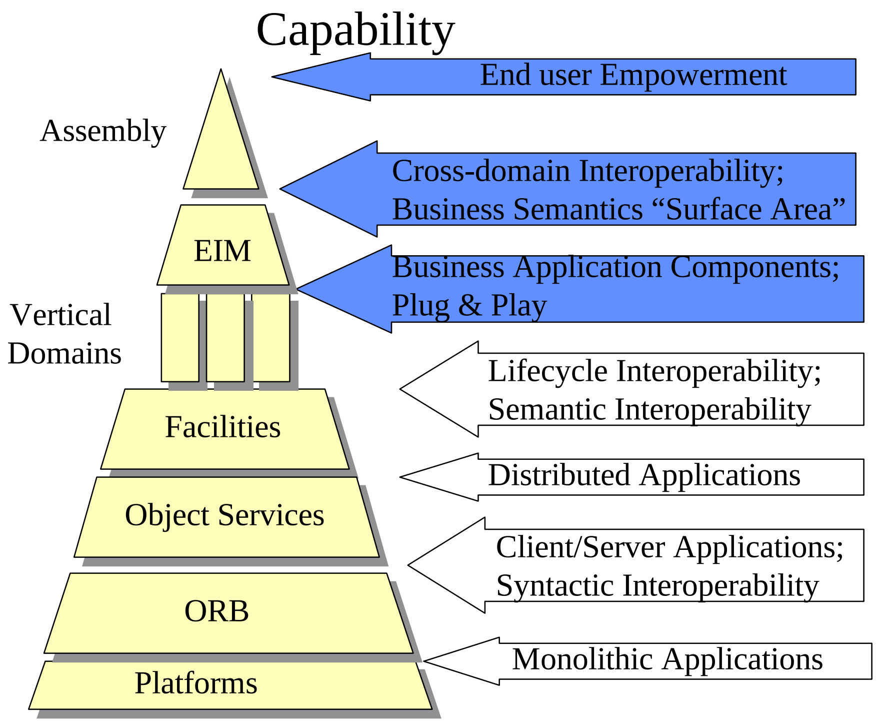
Capability
End user Empowerment
Assembly
Cross-domain Interoperability;
Business Semantics “Surface Area”
EIM
Business Application Components;
Plug & Play
Vertical
Domains
Lifecycle Interoperability;
Semantic Interoperability
Facilities
Distributed Applications
Object Services
Client/Server Applications;
Syntactic Interoperability
ORB
Monolithic Applications
Platforms

Figure 2-3 Architectural Objectives

2.1.2 Scope

This specification addresses the “Business Object Facility” described in OMG’s “Common Facilities RFP-4, Common Business Objects and Business Object Facility”. The Business Object Facility specifies the architecture required to support interoperable business objects configured as plug and play business components.

This specification does not address “Common Business Objects”, which are defined as those business objects representing business semantics that can be shown to be common across most businesses. This specification does, however, define the set of interfaces and semantics which all business objects must have in order to achieve the stated goals and objectives.

Fundamental characteristics of Business Objects limit the scope of the BOF. Business Objects:

- Represent an instantiated thing active in the business domain.
- Persistently store state.   Business Objects maintain persistence transparent to the client, and recover their state to be consistent with the rest of the system if any failure should occur. Persistence may be implemented using Relational Databases, Object

Databases, the OMG Persistence Service, or other mechanisms, as long as the mechanism adheres to transaction semantics.

- Frequently access state shared between many users.
- Are subject to transactional semantics. Transactions provide both recovery and concurrency control. A transaction is a unit of work that has the following (ACID) characteristics:
    - Atomic; if interrupted by failure, all effects are undone (rolled back). Every transaction a user intiates will run to completion (that is, the updates will be made permanently on disk), or that entire transaction will be backed out, as if the updates the transaction made never occurred.
    - Consistent; the effect of a transaction preserves invariant properties.
    - Isolated; its intermediate states are not visible to other transactions. Transactions appear to execute serially, even if they are performed concurrently.
    - Durable; the effect of a completed transaction is persistent; it is never lost (except in a catastrophic failure).
- Manage concurrency and serialization. When concurrent transactions share a business object (or underlying data store), serialization is implemented to assure that the actions of competing transactions are controlled in such a way that the result is equivalent to the transactions being executed one after another instead of concurrently.

Examples of objects which do not match the profile for Business Objects, and consequently are outside the scope of the BOF, include:

- Graphical user interface objects.
- Documents and Office automation objects, including objects embedded in spreadsheets, word processors, presentation tools, and other personal productivity tools. Office automation applications are concerned with routing, processing, designing, using, and finding documents. Characteristics of these types of objects include: document sharing between many users (electronic bulletin boards, shared notes, electronic mail); support for graphics and complex relationships among drawing objects; hypertext links; and dynamic data interconnection (hot links).
- Design applications, including software engineering (e.g., Compilers, CASE tools, Repositories, and other development environment tools) as well as Computer Automated Design of physical and electronic products. Design applications are characterized by: many interdependent subsystem designs; designs evolving over time (complex versioning and management of multiple design alternatives); and concurrent engineering on multiple versions of large designs. These types of applications require the ability to lock a large group of data, such as a composite object or an entire design, for long periods of time.
- Operating systems, TP monitors, communications, DBMS, and other parts of the system platform infrastructure. These infrastructure objects may be used by Business objects, but, individually, they do not have all the characteristics of a business object.

- Transaction Services, Naming services, Concurrency Services, Relationship Services, and other CORBAservices comprising the infrastructure necessary to support Business Objects.

Many of the above application types, and their application objects, may interoperate with business objects. "Figure 2-4 Object Architecture" illustrates the relationship of Business Objects to other types of objects.

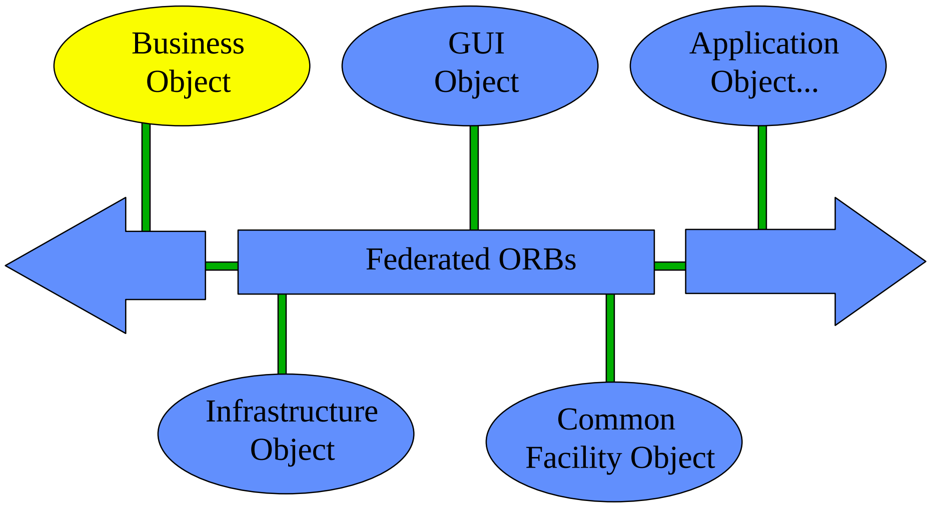

Figure 2-4 Object Architecture

This specification defers much of the definition and integrity of business semantics to the Meta Object Facility and the meta models anticipated from the Object Analysis & Design Facility. Architectural objectives related to tight coupling of business objects across their lifecycle, business semantic interoperability, etc., are dependent upon these facilities. However, neither of these facilities have been specified. Consequently, this version of the Business Object Facility narrows its (IDL specification) scope to the runtime environment, with admittedly inadequate coverage of lifecycle and semantic interoperability. As specifications for the Meta Object Facility and OA&D metamodels become available, the Business Object Facility IDL specification must evolve to ensure the original architectural objectives are satisfied.

2.1.3 Interoperability

It is a primary goal of this specification that business application components configured from business objects be interoperable regardless of the implementation of the framework, implementation of business semantics, engineering methodology, source language, operating system, human language or business domain of the business application components.

Interoperability is maintained at a high semantic level between independently developed business objects through the following mechanisms:

- All business objects are derived from a set of interfaces and semantics which address requirements for interoperability, including:
    - Enforcement of ACID properties through consistent use of persistence, concurrency, and transaction mechanisms.
    - Consistent use of object relationship mechanisms. Includes ability to enforce model-based relationship semantics dynamically, independent of, and transparent to, business object implementation.
    - Consistent handling of events and notifications. Includes ability to enforce model-based event semantics dynamically, independent of, and transparent to, business object implementation.
    - Consistent handling of compound services across graphs of business objects, including lifecycle and externalization services.
- Semantics for all business objects are rigorously specified within a business object model. Each business object is directly linked with its defining business meta-object. Enforcement of many semantic definitions are performed directly by the specified BusinessObject Interface. Additionally, the business object model supports rigorous formal semantic specifications of complete structural and behavioral models, as well as pre-conditions, post-conditions, invariants, etc. Consistency between business meta-objects is enforced by the Meta Object Facility. Within the scope of these specifications, interoperability is assured between business objects.

2.1.4 Separation of technology issues

The Business Object Facility isolates Business Objects and business application components from the technology required to implement or use them.

- The Business Object Facility does not impose implementation constraints for persistent data store. The implementor may use the Persistence Service to help isolate data store technology from business logic. Use of the Persistence Service would enable replacement of data store technology independent of, and transparent to, business application components.
- In order to ensure business application component interoperability when configuring enterprise-specific solutions, the Business Object Facility requires presentation/user interfaces to be isolated from implementations of business semantics. Consequently, visualization/presentation/user interface technology can be replaced independent of, and transparent to, business application components.

- The Business Object Facility defers management of business object semantic specifications to the Meta Object Facility. The Business Object Facility uses published interfaces to interoperate with the Meta Object Facility. There are no technology or implementation dependencies between these facilities.

  As with all OMG specifications, business object specifications are independent from technical implementation. Furthermore, although business object specifications typically incorporate more semantic details than other OMG specifications, there are no defined technology dependencies between business object specifications and any particular form of development life cycle, development tools, or artefacts produced during analysis or other life cycle stages. The Business Object Facility, business objects, and business application components are thus isolated from the technology of development lifecycle artefacts, tools, process, and models. Issues related to interoperability between these technologies will ultimately be addressed by Object Analysis & Design and/or Meta Object Facility specifications.

The location and implementation of storage systems and of presentation and rendering mechanisms are implementation decisions which have no impact on business object model semantics. The business object model is managed by the Meta Object Facility, in conjunction with a variety of tools, methodologies, and processes. The Business Object Facility, along with all business objects, are defined and controlled by the business object model. *The business object model is the driving force of the information system, not the technology.*

2.1.5 Extensibility of business objects

Business objects, and configurations of business objects, will need to be specialized, tailored, extended, and configured for use in specific vertical business application domains, enterprises, or even workgroups within the enterprise. Extensibility requirements are addressed as follows:

- **Replacability**. The unit of implementation is defined by “module” (which is the basis for defining “business application component”). It must be possible to relocate or replace a business application component with another implementation of the business application component, transparent to all clients. The replacement must result in replacement of all business objects which are included in the business application component. This implies that business objects can not, for example, be statically linked to business objects of a different business application component. (Note that implementation of business application components may still utilize a variety of technical approaches, including multiple physical packages distributed across heterogeneous platforms). This requirement supports implementation of plug and play business application components.
- **Inheritance**. Business Objects must support the interface and semantics specified for their parent Business Objects. For example, each Business Object operation must specify support for the same type, or supertype, for all corresponding “in” arguments; and must deliver the same type, or subtype, for all corresponding “out” arguments.

  Furthermore, adherance must be made to the extended semantic definitions specified within the business object model, including provision for “looser” pre-conditions, “tighter” post-conditions, compatible invariants, etc. Likewise, tailored relationship, event, and lifecycle specifications must ensure adherance to parent business object specifications. It is assumed that the Meta Object Facility enforces consistency between all Meta Business Objects. It is also assumed that the Object Analysis and Design specification will define mechanisms for consistently and rigorously expressing business semantics.

  A new Business Object which inherits its specification from an existing Business Object is required to physically inherit the implementation of that parent Business Object, unless the operations are explicitly overriden. This requirement enforces modularity, ensures adherance to principles of plug and play business application components, increases reusability, promotes an open competitive market for business application components, provides uniform inheritance semantics across interoperating ORBs, and reduces the risk of implementaitons becoming monolithic. OMG does not take a position on implementation inheritance, only interface definition inheritance. The requirement for plug and play business application components, and the other requirements listed above, necessitate implementation inheritance. The technical implementation of inheritance will depend upon the native ORB capability. One of the following scenarios will exist:
  - The ORB supports implementation inheritance. In this scenario, the native ORB capabilities are used to implement the mandated implementation inheritance.
  - The ORB does not support implementation inheritance. In this scenario, all inherited operations (which are not overridden) must use a delegation mechanism to ensure that they are physically implemented by the specified module (business application component).

- Context. The Meta Object Facility supports views, or contexts, of the business object model. Contexts may be used to reduce complexity of the composite model by exposing only that detail which is relevant for the particular modeling task. It is assumed that the Meta Object Facility enforces consistency between all contexts. The implementation of a Business Object must encompass the composite of all context specifications.
- Composition. Extensions often include new relationships between business objects or new properties within objects. New relationships and properties can be modeled and implemented dynamically, transparent to business object implementations.
- Configuration/Tailor.   There are several mechanisms for dynamically altering the behavior or characteristics of business objects:
  - For many types of semantic specifications, business objects will dynamically and transparently adopt changes made to the business meta-object specification. Some of these specifications will become initialization policy or validation policy. Some of these specifications will become implemented as roles and relationships with embedded semantics.

    - Some business objects may be designed to dynamically adopt other changes to their business meta-object specification, such as changes to operation pre-condition, post-condition. One mechanism for implementing these types of changes is through the validation policy.
    - Some business objects may be configurable through a defined operation interface.
    - Some business objects may implement a special-purpose business meta-object which enables configuration tailoring through a defined operation interface.
    - Some business objects may be associated with a tailorable "factory" which has configuration options at business object create time. One mechanism for tailoring factory semantics is through the initialization policy.
- Version.

2.1.6 Reusability
The Business Object Facility lays the foundation for specifying and implementing interoperable business objects. Based on this foundation, it is anticipated that OMG business domain task forces, and other industrial consortia such as SEMATECH and OAG, will accelerate consensus building activities and begin issuing industry-specific standards on business objects. The Business Object Facility will provide the industry standard definition of business object infrastructure and will provide the architectural layer which isolates technical complexity from the business semantic.

It will ultimately be the business domain consortia which will, through adopted business object standards, create an open market for business objects and achieve large scale business application component reuse.

2.1.7 Scalability
The Business Object Facility is a composition of OMG specifications. Scalability guidelines accompanying each referenced OMG specification has guided the definition of the Business Object Facility, providing confidence that the facility itself is scalable.

The scalability of a business object model may be influenced by the granularity of business objects, distribution of business objects, and complexity of interdependency between business objects. The issue of business object scalability must be evaluated in conjunction with future business object specifications.

2.1.8 Ease of development and deployment

A primary business driver for Information Technology includes the profit-oriented objective to decrease time-to-market for products and services within an environment of increasing business complexity resulting from accelerating changes to products, processes, customers, partners, and Information Technology. A successful information strategy will accommodate these business drivers by provisioning business solutions at a rate commensurate with the increasing rate of business structural change.

Rapid solution delivery in response to continuous business process change requires direct involvement of empowered users to dynamically change their business processes, workflows, rules, policies, presentation, and other aspects of their environment. The "ease of development" objective means that business users must be able to directly and rapidly provision their solutions in an environment of increasing complexity.

The Business Object/Meta Object/Analysis & Design facilities intentionally ensure isolation of infrastructure from tool or presentation technologies. It is the tool and presentation technologies which will ultimately provide "user-friendly" interfaces for business object development and deployment.

However, the Business Object/Meta Object/Analysis & Design facilities establish the environment in which the underlying business objective can be attained: rapid provisioning of increasingly complex business solutions.

Software productivity and ease of development depend not only on sheer bulk but also on *surface area*, the number of things that must be understood and properly dealt with in order to successfully enable interoperability between business application components. Factors influencing surface area include:

- Amount of visible information. Surface area increases with the number of names that are exposed through the software component interface, including data element names, data types, and function names.
- Sequence dependencies. Surface area increases with each requirement that the business application component user must perform operations in a particular order.
- Environment and responsibility scope dependencies. Surface area increases whenever the software component user is responsible for managing lifecycle, persistence, location, or environmental aspects of software components within a more global application context.
- Technology dependencies. Surface area increases with exposure to each technical domain and form of interface, including middleware communications, data storage.
- Concurrency. Surface area increases when concurrency issues are exposed to the software component user.

The Business Object Facility supports surface area/complexity reduction by standardizing on the way in which CORBAservices are composed, configured, and specialized. These measures reduce the *surface area* exposed to the user or developer due to environment, responsibility scope, technology, concurrency, and other factors which are not directly related to the business problem domain.

The Meta Object Facility will support surface area/complexity reduction through the mechanism of views and contexts. More generally, the Meta Object Facility provides for tailoring, configuration, and integration of business objects within an enterprise.

The Meta Object Facility in conjunction with the Analysis & Design facility ensure integrity and consistency of business object models. They ensure integrity and consistency of business object models across the development lifecycle. They enable tools to be implemented which automates progression of the business object model between stages of the development lifecycle.

The Business Object Facility and Business Objects enforce adherance, often dynamically, to business object specifications. The Business Object Facility enforces the principle of plug and play business application components, thus enabling widespread reuse. It is envisaged that industrial consortia such as SEMATECH and OAG will defined vertical domain frameworks based on the Business Object Facility structure. These frameworks will specify business application component semantics in sufficient detail to enable plug and play between products provisioned from multiple vendors, transparent to any clients.

It is the realization of tailorable, replacable, reusable, interoperable, off-the-shelf business application components which consummates the cycle time objective.   The business user will be able to:

- Select, acquire, replace business application components based on factors of performance, price, quality, technical implementation, etc., confident that the business specification is followed.
- Specify problems in the business language (assuming external tool), confident that the underlying implementation has a common, formal, rigorous, and consistent, semantic basis which ensures interoperability with any other business object.
- Customize in unplanned ways. Many specifications can be dynamically added/removed/customized for any business object, such as properties, relationships, events. Domain-specific specifications will have extended customization options. These forms of customization do not require any coding or reimplementation of the business objects.

Business Application Components are the units of implementation. This specification does not put any packaging or technology constraints on the deployment of business application components.

There is a direct correspondence, maintained at run time, between the business object model and business object instances. Business object instances are directly and dynamically controlled by business object specifications.

2.1.9 Application integration

This specification defines *business application component* as a configuration of business objects. Within the context of OMG IDL, a business application component corresponds to “module”.   The Business Object Facility specifies implementation inheritance, ensuring a common implementation semantic across all ORBs and enabling plug and play replacability between different implementations of a business application component.

Thus, a business application component has boundaries which are rigidly defined by the OMG IDL “module” construct and contains an arbitrary number of objects, including business objects and meta business objects. Business objects have the same basic characteristics as other objects in the OMA, but are differentiated from other objects as follows:

- Business objects inherit specifications from a well defined BusinessObject interface. This interface supports the objectives of the Business Object Facility, including business object interoperability, isolation of technology from business logic, transactional integrity across distributed business objects, implementation inheritance consistency across interoperable ORBs, plug and play business application components, etc.
- A Business Object is directly coupled to its defining meta business object. This permits flexible and dynamic business object models, enforced business semantics, consistency of business object specification across the development lifecycle, etc.

Thus, the Business Object Facility requires:

- All business objects to inherit from the BusinessObject interface.
- All business objects to be associated with a unique Business Meta Object which inherits from the BusinessMetaObject interface.
- All Business Meta Objects must have their business semantics expressed in a form compliant with the Analysis & Design Facility.
- All business application components must have implementation inheritance.

Business objects are not, in and of themselves, business solutions. Part of a business solution requires coupling business objects with a user (in the generic sense, a user can be a machine on the factory floor, an external application, a person working through a user interface, etc.). Business objects are intentionally specified to be isolated from external interfaces. The mechanics of coupling business objects to external interfaces is beyond the scope of this specification, but it is envisaged that “assembly” tools will enable the business user to browse the business object model, select business objects, and “wire” them to visualization components. One form of assembly tool might be workflow. The Workflow Management Coalition is in the process of defining interoperability specifications for workflow products. It is anticipated that these interoperability specifications will be expressed in OMG IDL and implementations will support access to CORBA objects. The relationship between workflow models and business object models is not clear, there is some overlap in objectives. This specification does not attempt to resolve or reconcile potential redundancies between the models.

On the other hand, a business solution also requires specifying business semantics, setting business policy and rules, defining relationships between business objects, etc. This aspect of business solution provisioning is isolated from visualization, storage, and implementation technology, preserving the investment across changes in technology. These semantics are defined on any configuration of Business Meta Objects. In addition to tailoring individual Business Meta Objects, it is possible to define relationships between Business Meta Objects, event semantics between Business Meta Objects, rules and policies spanning Business Meta Objects, etc. These specifications are made as part of a business solution within some business context. The context-specific specifications are reflected in the underlying aggregate business object model. The business objects implementing the business object model reflect specifications made in all business contexts.

One of the implied requirements of business object interoperability is that there are no “boundaries” which delineate one business solution from another. The term “application” often implies a “boundary” which distinguishes and isolates its specification and implementation from other application specifications. To avoid potential negative and restrictive implications associated with “application”, it may be prefereable to use the phrase “contextual business solution”.

A business application component is defined with clearly defined boundaries, per the definition of “module” in OMG IDL. The relationship between business objects and business application components is also clearly defined, using definitions of module and interface in OMG IDL. Contextual business solutions are specified using the business object model. These contextual business solutions:

- Impact an arbitrary number of business objects. It is possible to identify business object specifications which are directly or indirectly impacted by any given contextual business solution.
- Overlap, interoperate, impact, and depend upon other contextual business solutions in arbitrary ways.

To implement a contextual business solution, each altered Business Meta Object must be reflected in the implemented Business Object. Some specification changes can be reflected dynamically in the Business Objects. If the specification change can not be reflected dynamically, the Business Object must be reimplemented. The mechanism for reimplementing a Business Object is beyond the scope of this specification, but it is envisaged that the process will be aided by tools which automate the generation of Business Objects based on the Business Meta Object and Business Object Model.

A BusinessObject is a composition of CORBAservices, including Transaction services. All business objects are transactional objects. This definition ensures that units of work spanning distributed business objects, business application components, processes, and platforms maintain transactional integrity. Transactional boundaries are defined in the business object model and are (transparently) enforced by business objects.

In addition to transactions, business objects collaborate on relationship integrity, generation and consumption of events, lifecycle on graphs of business objects, externalization on graphs of business objects, queries, etc. In some cases, these reflect explicit business object model constructs. In other cases, the features are implied. The consistent use and semantic of CORBAservices across all business objects ensures interoperability for all forms of business object collaboration.

2.1.10 Security

The Business Object Facility does not explicitly address Security. Security is provided by CORBA Security, which implements security features for all applications, whether they are security aware or not. If required, domain or enterprise-specific Business Objects may be implemented as security aware.

CORBA Security provides a level of security functionality which is available to applications which are unaware of security. This includes security of the invocation between client and target object, ORB-enforced access control check and auditing of security relevant system events. Security aware applications can use security mechanisms to exert finer level of access control and auditing for encapsulated functions and data.

2.1.11 How business objects implement the business model

The Business Object Facility defines a tight coupling between Business Objects and the corresponding Business Meta Object. The set of all Business Meta Objects and related semantics constitute the Business Object Model. The Business Object implements, often dynamically, the business semantics defined by the Business Meta Object.

2.1.12 Legacy applications

Until the objective of plug & play business application components has been achieved, we can view all forms of application software as “legacy”. Legacy software will be in many different forms, will be purchased or developed in-house, will be in different stages of maturity with respect to the “Figure 2-3 Architectural Objectives” on page 12, and will utilize many different forms of technology infrastructure. A detailed legacy transition plan will have to be devised by each enterprise to represent their unique configuration of legacy software. This specification can not address all possible scenarios. However, as a guide, it will be useful to consider desirable characteristics at each successive architectural layer.

There are many paths which could be followed on the transition from our legacy applications towards achieving plug & play business application components. Sometimes the choices are arbitrary and multiple alternatives would achieve similar results. Unfortunately, we do not often have the luxury of pursuing multiple paths concurrently. The following criteria will serve as a guide for determining the path to be followed:

- Closest Conformance with the vision of plug and play Business Application Components.
- Lowest Solutions Provisioning Cycle Time.
- Least complex business application components (least exposed “surface area”).
- Loosest runtime coupling (plug and play business application components).
- Highest business semantic integrity (tightest coupling across business application component lifecycle).
- Longest business application component life (longevity: greatest potential for insulating business application components from change).

For each architectural layer, a “reference point” is established based on the definition of services and capabilities for that layer. The following guidelines are with respect to transitioning legacy from one reference point to the next:

- **Plug & Play Business Application Components.** This reference point constitutes the objective, the point at which legacy transition concerns end.
- **Semantic Interoperability.** (Common Facilities). It is risky to speculate on future industry standards for business application components. It is unlikely that those future standards will align exactly with any given enterprise model. To protect software investment and minimize potential disruption both with external components (e.g., visualization components, etc.) as well as interface between business application components, it is recommended to define business application components and business objects with large granularity. These large grained business objects can later be implemented as “containers” for standardized business objects, thus preserving existing interfaces and semantics during the transition to plug & play business application components. The Business Object Facility capability to define relationships, events, etc., will enable most large grained business objects to emulate their “legacy” behavior using smaller grained plug & play business application components. Since the Business Object Facility is model-based and semantics are rigorously specified, it is very likely that the transition to plug & play business application components could be automated. Most difficulties associated with this transition are likely to stem from overlapping boundaries of standard versus enterprise-defined business application components. If a standard business application component has functionality split across multiple enteprise-defined business application components, migration will be more difficult. Larger granularity of enterprise-defined business application components will help reduce the number of conflicts. The other advantage with large granularity, at this stage, is reduction of exposed complexity (surface area).
- **Distributed Applications.** (Object Services). In consideration of transitioning to “semantic interoperability”, business objects and business application components should be large grained. Lack of the Business Object Facility will result in extra effort to incorporate and efficiently use CORBAservices. All business objects should inherit from a single enterprise-specific BusinessObject. This will ease the transition

to the Business Object Facility: the enterprise-specific BusinessObject can be modified to inherit from the facility's BusinessObject. Dynamic specification of relationships, events, and other semantics are impractical at this stage. Nevertheless, static implementations of relationships, events, persistence, transaction, etc. should be developed and should be consistent with corresponding Business Object Facility specifications. Consistency in these static implementations will facilitate replacement with their dynamic, specification-driven counterparts when transitioning to the "semantic interoperability" stage. In place of business semantics defined in the Business Meta Object, descriptions must be written in (english) language and implemented manually. The semantics will need to be entered into the business object model, so they should be written with rigor and formalism which aligns with the Meta Object Facility and Analysis & Design Facility. Business Application Components should use implementation inheritance.

This stage is technically challenging from an implementors viewpoint. Without the tools, formal semantics, and enforced conformity of implementation, there are risk factors and productivity concerns which will deter many from transitioning into this stage. Other than proof-of-concept and technology prototypes, consideration should be given to skipping this level.

- **Client/server applications.** (ORB). In consideration of transitioning to higher levels of capability, business objects and business application components should be large grained. Due to lack of CORBAservices at this stage, there are severe distribution constraints on business application components and underlying technology implementations. To ensure system (transaction, concurrence, persistence, referential, etc.) integrity, each business object operation must be executed within implementation constraints of a single (logical) DBMS. Amongst other things, this normally means execution must occur within the same thread/process/address space (transaction/unit of work/concurrency implementation restriction). In order to ensure adherance to this constraint, each operation is implemented as a transaction. Since there are no concurrency or nested transaction mechanisms, there is no ensured system integrity if an operation were in turn to execute other OMG IDL defined operations. Furthermore, in order to implement relationship semantics and referential integrity, there must exist a common database which is used by one or more business application components. In effect, the client/server application stage has the same characteristics as monolithic applications except that the visualization technology (and other external interfaces) are isolated from the business logic. Relationship semantics and implementations are not exposed through the OMG IDL, but may be independently specified in some data definition language. Due to lack of any general interoperability between business objects at this stage, there is little need for formalism, rigor, or consistency in semantics, and the semantics would be limited to operation semantics (e.g., pre-conditions, post-conditions). In a practical sense, business application components at this stage are wrappered "stove pipe" applications which expose their interface using a consistent, programming language-independent, implementation-independent, location transparent mechanism. Other stove pipe applications can use the interface to execute operations, but there are no mechanisms to ensure system transaction/ concurrency integrity between the operations. This restriction also implies that implementation inheritance should not be used. Thus, all business objects must be defined independently - inheritance should not be specified.

  Transitioning from the client/server application stage to higher architectural levels follow these guidelines:
  - Retain the defined interface. Re-implement the interface as a "router" which delegates operation execution to the new distributed business object implementation. This will preserve all existing presentation and other external interfaces.
  - Use transaction/concurrency service in the distributed business object implementation.
  - Isolate the data store technology. Reimplement the data store interface using persistence service.
  - Reimplement relationships using relationship service.
  - Implement other services for business object per guidelines for "distributed applications".
- **Monolithic applications.** The primary result of transitioning from "monolithic applications" to "client/server applications" is the isolation of user and external interfaces from business logic. ORBs are implemented on virtually every platform and support most of the programming languages used to implement enterprise legacy applications. Typically, interactions with enterprise legacy applications are defined in terms of transaction semantics and consequently map to the characteristics of a "client/server application" operation as described above. Once separation of business logic and user interface is complete, business logic can be "wrapped" for ORB access. It should not be necessary, at this stage, to alter business logic. Business logic wrapping is typically an automated process, there will generally be one business object operation per transaction. One monolithic application may correspond to one or a few business objects (to maintain large granularity at this stage).

It should be noted that the above guidelines assume that unrestricted modifications can be made to the "legacy". Often, the legacy is a purchased product and can not be modified. In some cases, "wrappering" technology, product-provided APIs and exits, and other tailoring options may enable partial transition of the legacy towards the objective of plug and play components. Ultimately, the best course of action is to influence the product vendor to migrate his software to open standards.

2.1.13 Flexibility and longevity

Business objects and/or application components enhance the flexibility, maintainability and longevity of application implementation in the following ways:

- Technology isolation. Business objects are independent of both visualization technology and data store technology. Changes to technology implementation can

occur without modification to business object implementation. Business objects will outlive the underlying technology.
- Industry standard components. Many applications will be implemented through tailoring and combining industry standard plug and play components. The need for enterprises to write and maintain their own code will be greatly reduced.
- Extensibility, reusability, ease of development. All the arguments provided on these topics in previous sections apply to flexibility, maintainability, and longevity of application implementations.

2.1.14 Generality and desktop integration

Business Objects can interoperate with other forms of application components such as user interface, agents, world wide web, palmtop computers or any other system. Based on existing and proposed CORBA interoperability standards, Business Objects can interoperate with application components implemented on Microsoft OLE/COM, Internet/JAVA, or other CORBA implementations.

Furthermore, Business Objects themselves could be implemented on a non-CORBA infrastructure if:
- There is an implementation of CORBA interoperability supported for that non-CORBA infrastructure, and
- The CORBAservices required by the Business Object Facility and all Business Objects are accessible and maintain integrity across the interoperability interface.

2.1.15 Proof of commonality

This Business Object Facility is a composition of the fundamental services necessary to implement and ensure enterprise-wide interoperability between distributed business objects. These services are the distributed environment equivalent of the mainframe services required to support the high-integrity, high-reliability, high-availability business applications of global enterprises.

2.1.16 Specification of business objects and metadata

OMG IDL is not sufficient to represent the semantics of business objects. True interoperability requires exposing information as to what the interface means and requires. The Business Object Facility relies upon the Meta Object Facility to specify business object semantics, including constraints, rules, roles, policies, relationships, states, attributes, visibility, dependencies, protocols, pre- and post- conditions, error conditions, warning conditions or events. A subset of this specification is IDL. The Meta Object Facility will maintain synchronization between the full business object semantic definition and the interface repository (IDL).

2.1.17 Multilingual use

The Business Object Facility is a composition of lower level services. There has been no attributes or operations added which are subject to internationalization.

Internationalization facilities can be freely used in any derived common business object or business domain object.

2.2 Conceptual Model

2.2.1 Business Semantic Model

A *meta-object* is an object that represents aspects or features of another object. A *meta-object* creates and manages representation concepts, along with the actual object instances they represent. Typical examples of *meta-objects* are model, class, operation, constraint, and association. A collection of *meta-object types* will provide the primitive building blocks needed to represent the complex semantics of object models such as the business object model. The term *object schema* is defined as a composition of related meta-objects that together represent semantics of the objects being modeled.

A *business object* is an object which represents an instantiated thing active in the business domain. A *business meta-object* is a meta-object which specifies the interface and semantics for such a business object, including at least its business name and definition, attributes, behavior, relationships, rules, policies and constraints. A *Business Object Model* is an object schema which represents the complete and rigorous semantic specification of interoperating business objects, including their definition, attributes, behavior, relationships, events, rules, policies, constraints, and any other aspect or feature required to be associated with business objects. The Business Meta-Objects and the Business Object Model are implemented within an infrastructure specified by the *Meta Object Facility*.

A *business application component* is a configuration of objects, including business objects, which specifies the unit of implementation. This specification does not impose any specific implementation technology. Implementations can utilize technology which enables replication, federation, distributed packaging, etc. The OMG IDL *module* construct defines the objects and interfaces constituting a business application component.

Business Objects are implementations of Business Meta Object specifications. Business Objects are implemented within an infrastructure specified by the **Business Object Facility**. The **Business Object Facility** constrains Business Object implementations to adhere to their Business Meta Object semantic specification, ensures implementation integrity for the Business Object Model, and enforces architectural principles enabling plug and play business application components.

The **Meta Object Facility**, in conjunction with appropriate tools, promotes ease of construction, use, maintenance, and assembly of business solutions from business object specifications. The Meta Object Facility enables the business solution provider to focus on business semantics while minimizing his exposure to underlying technical complexity.

The **Business Object Facility** is the architectural layer which couples the business semantic (Business Meta Object) to a technical implementation (Business Object). The **Business Object Facility** is not viewed as a facility which is used directly by an end user. It is envisaged that various automated mechanisms will be used to transform semantics defined in Business Meta Objects into Business Objects compliant with this Business Object Facility specification.

As shown in “Figure 2-5 Business Semantic Model”, the Business Object Facility standardizes usage, configuration, and composition of CORBA specifications to ensure business object interoperability. The Business Object Facility is the infrastructure upon which all business objects are implemented. The Business Object Facility consists of the following:

- Business Object. The interface specification for Business Object is inherited by all business objects. It is the specification of the minimum required level of functionality necessary to implement an interoperable business object. The Business Object specification is a composition of CORBAservices designed to ensure interoperability. Additionally, the Business Object is created from, and controlled by, an associated Business Meta Object. The Business Meta Object contains the rigorous business semantic specification for the Business Object.
- Business Services. A collection of services (interfaces and objects) that support basic functions for using and implementing interoperable business objects. Business services are necessary to construct any distributed business application and are always independent of specific business application domains.

The Business Object Facility has critical dependencies upon existing and developing OMG specifications. The Business Object Facility specification is, in particular, dependent upon the Meta Object Facility and the Object Analysis & Design specifications which are being developed independently and concurrently. This specification is predicated on the existence of certain capabilities within those independent specifications. The exact form of interface to those capabilities is speculative and will need to be reconciled with the specifications ultimately adopted. For the purposes of this specification, the reference model depicted in “Figure 2-6 OA&D Object Semantic Specification Reference Model” will be used to describe capabilities of OA&D relevant to the Business Object Facility.

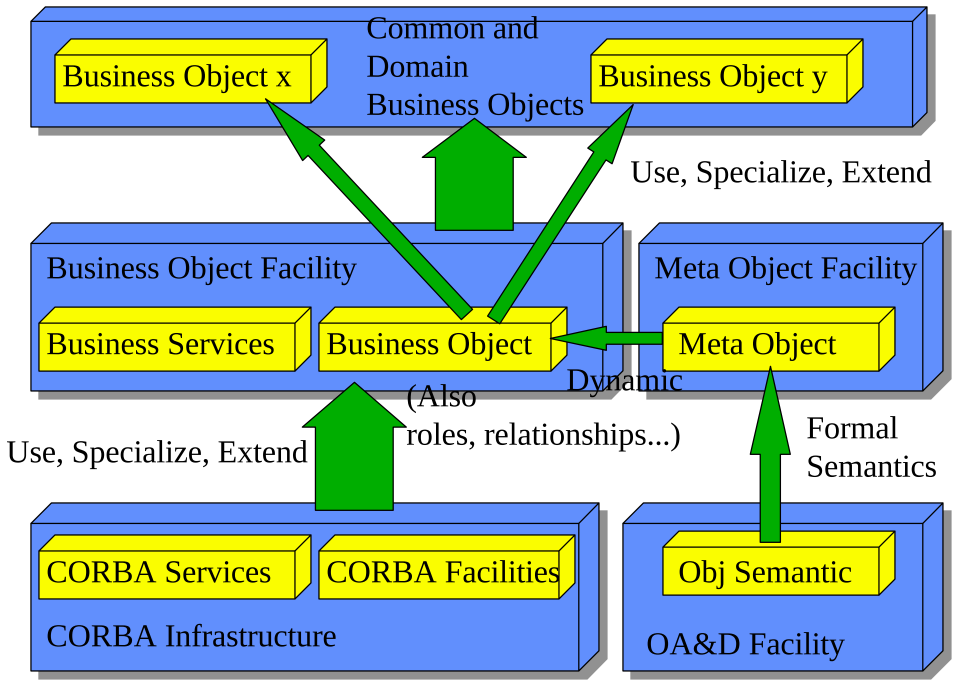

Figure 2-5 Business Semantic Model

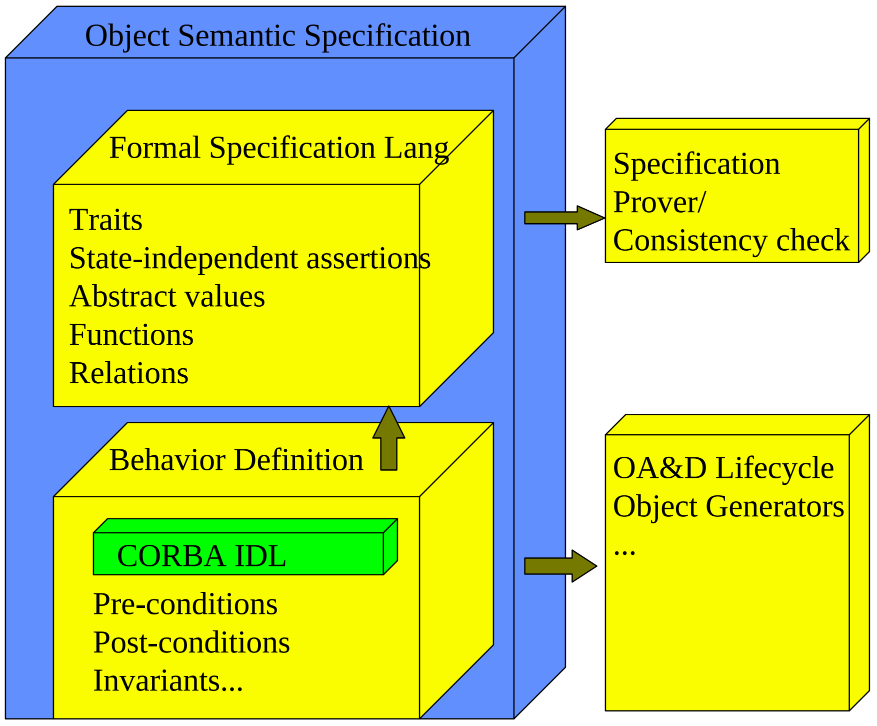

Figure 2-6 OA&D Object Semantic Specification Reference Model

2.2.2 Business Transaction Model

Business objects are persistent objects subject to transactional semantics. Transactions provide both recovery and concurrency control. A transaction is a unit of work that has the following (ACID) characteristics:

- Atomic; if interrupted by failure, all effects are undone (rolled back). Every transaction a user intiates will run to completion (that is, the updates will be made permanently on disk), or that entire transaction will be backed out, as if the updates the transaction made never occurred.
- Consistent; the effect of a transaction preserves invariant properties.
- Isolated; its intermediate states are not visible to other transactions. Transactions appear to execute serially, even if they are performed concurrently.
- Durable; the effect of a completed transaction is persistent; it is never lost (except in a catastrophic failure).

A transaction can be terminated in two ways: the transaction is either committed or rolled back. When a transaction is committed, all changes made by the associated requests are made permanent. When a transaction is rolled back, all changes made by the associated request are undone. The Transaction Service defines transaction semantics in a distributed environment.

To ensure transactional integrity, business objects must be executed within a transactional context. Some operations on Business Objects are defined as TransactionalClients. A TransactionalClient establishes the transactional context, controls the usage and sequencing of Business Object operations, enables event monitoring, and provides an easy to use and stable interface for the user and other external resources (see “Figure 2-7 Non-Transactional Objects Must use Transactional Client Operations”).

Every business object is a transactional object. This ensures propagation of transactional context to other business objects.

Business objects are characterized by frequent access to data shared between many users. Issues related to long transactions or conversational transactions (which are required, for example, in CAD applications) are generally not applicable to business objects. The concept of nested transaction is traditionally applied within the context of long transactions, a nested transaction being one possible mechanism to implement long transactions. The semantics of nested transactions include provision for subtransactions to independently commit or roll back. Commiting a subtransaction is provisional, durability depends upon commiting the top-level transaction. Thus, the ultimate control comes from the top-level “transaction” which makes the final decision on commit versus roll back. The net effect on the integrity of the system, from the users viewpoint, is the same as if it were a flat transaction.

A nested transaction capability is necessary in an environment which must embed transactions, such as those containing wrappered legacy applications. The legacy application is likely to have its transaction logic buried in inaccessible code. When a new business requirement changes the boundaries of "unit of work" (for example, a new requirement to ensure transfer of control of some entity between two independent applications), a new high-level transaction must be implemented. Assuming the legacy transaction can not be changed, the only way to implement the new requirement is to use the mechanism of nested transaction.

Nested transactions require transaction service implementations and resources which are subtransaction aware. It is a capability which business object implementations can utilize for wrapping legacy applications. This specification does not explicitly require nested transaction capability. The interface specification and semantics of business objects do not reference or expose any aspects of nested transactions.

This specification is based on implicit transaction contexts. Transactional clients can use Transaction Services to support multi-threaded implementations of implicit transaction contexts. Explict transactions contexts could be supported by specific business objects if special circumstances warrent their use.

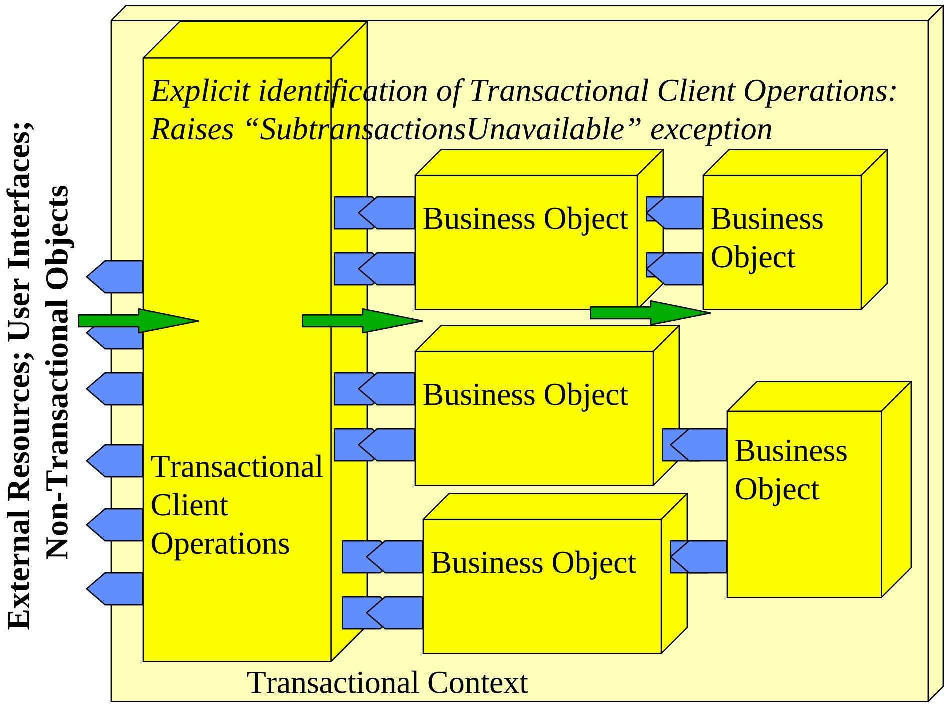
External Resources; User Interfaces;
Non-Transactional Objects
Explicit identification of Transactional Client Operations:
Raises "SubtransactionsUnavailable" exception
Business Object
Business Object
Business Object
Business Object
Business Object
Transactional Client Operations
Transactional Context

Figure 2-7 Non-Transactional Objects Must use Transactional Client Operations

2.2.3 Business Object Dynamic Relationship Model

The Business Object Architecture utilizes the Relationship service to implement dynamic relationships (see “Figure 2-8 Business Object Dynamic Relationship Model”). In this architecture, Business Objects are “nodes”. The roles and relationships can be managed by external objects. In particular, the semantics of relationships are represented in the Business Object Model and enforced through the Business Meta Object and Business Object implementations.

This relationship model is not only a standard implementation of defined relationship services, it is a standard implementation of both externalization and life cycle services. Consequently, relationships have well defined syntax, semantics, and implementations supporting:

- Full referential integrity.
- Compound operation propagation semantics.
- Graph and bounded traversal semantics.
- Life cycle propagation semantics.
- Externalization/internalization propagation semantics.

In order to more closely conform to the ODMG object model, and to directly utilize defined life cycle and externalization services for compound operations, the relationship service has been specialized to support only binary (instead of n-ary) many-to-many relations.

The specific semantics of any relationship is defined in the Business Object Model. These semantics must, of course, be consistent with the relationship service semantics. Semantic implications of the relationship service and business object model include:

- Each “role” within a business object node instance must be unique and can not be a descendent of another role associated with that instance. This restriction further requires uniqueness amongst all roles “inherited” by the business object node instance.
- Each “role” within a relationship must be unique and can not be a descendent of another role associated with that relationship.

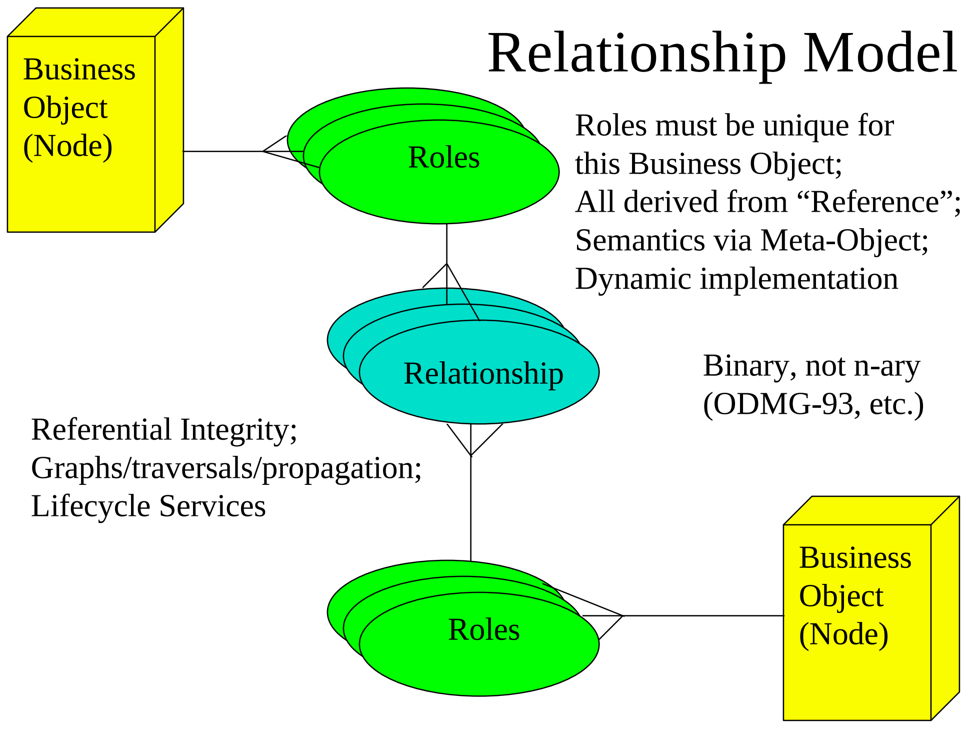
Relationship Model
Business Object (Node)
Roles
Roles must be unique for this Business Object;
All derived from “Reference”;
Semantics via Meta-Object;
Dynamic implementation
Relationship
Binary, not n-ary (ODMG-93, etc.)
Referential Integrity;
Graphs/traversals/propagation;
Lifecycle Services
Roles
Business Object (Node)

Figure 2-8 Business Object Dynamic Relationship Model

2.2.4 Business Object Dynamic Event Model

The Business Object Facility utilizes the Event service to dynamically implement associations between event suppliers and consumers. The architecture enables arbitrary associations between event suppliers, event consumers, and event channels (see “Figure 2-9 Business Object Event Model”). Event suppliers deliver events to an event channel, event consumers retrieve events from an event channel.

Business objects are transactional objects. By default, business object operations expect to be invoked within a transactional context. Some business object operations may serve the role of transactional client. Transactional client operations do not need or expect to be invoked within a transactional context.   Since the asynchronous nature of event management does not permit propagation integrity for the transactional context, only transactional client operations can be event consumers.

Business Objects can be event suppliers and/or event consumers. An event consumer must be a transactional client operation (i.e., there is no transactional context established when the event is delivered). In addition to business objects, user/external objects (e.g., resources, GUIs) can be event consumers. A default configuration of business objects, event channels, and transactional clients are defined in the business object model (and can be dynamically changed). A user/external resource can dynamically add itself to an event channel and consequently receive (filtered) events from any business object.

All Business Objects, as event suppliers, use the Push Model. Use of the push model greatly simplifies the responsibility of business objects. It ensures that the event channel mechanism is immediately and effectively used to implement complex queuing, quality of service, and other event semantics.

All business object transactional clients, as event consumers, also use the Push Model. Use of the push model greatly simplifies the responsibility of transactional clients. It ensures that the event channel mechanism immediately and effectively communicates events to the transactional client.

A user/external resource can choose to use either the Push Model, or the Pull Model, at its discretion.

The event service supports both generic and typed interfaces. The generic interface, which is a specific named operation supporting a single argument of *any* type, facilitates dynamic implementation of events between any business object and any transactional client (and by extension to any user/external resource). Typed interfaces support any form of *oneway* operation. Typed events are useful for explicitly identifying and declaring named events and their arguments. For each business object, the Business Object Facility supports the definition of EventsSupplied by that business object and EventsConsumed by that business object. The use of typed events also facilitates event filtering, the routing of events to the actual event consumer logic, isolation and independence of EventsConsumed logic from the Business Object logic, tight coupling between event specifications within the business object model and business object implementation, and the enablement of support for dynamic assembly of business solutions from interoperable business objects and other components.

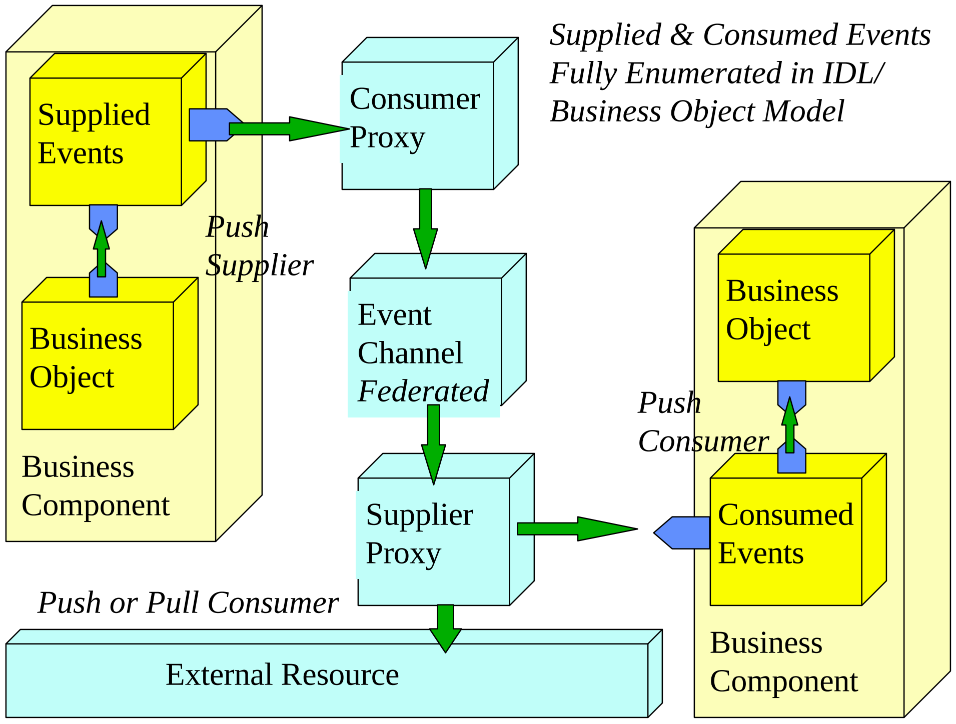

Figure 2-9 Business Object Event Model

2.2.5 Business Object Dynamic Property Model

The Business Object Architecture utilizes the Properties service to implement dynamic attributes for business objects. Property value initialization can be specified as part of the business object initialization policy. Validation of property values can be specified as part of the business object validation policy.

Properties defined on the Business Meta Object will propagate to initialization and validation policies.

2.2.6 Business Object Externalization Model

The Business Object Architecture utilizes the Externalization service to implement externalization/internalization. Each Business Object is a CosCompoundExternalization::Node and has the full semantics of compound object externalization, including operations on graphs of related objects. The semantics of externalize, internalize are controlled by the Business Object Model and ensure preservation of referential integrity.

2.2.7 Business Object System Management Model

The System Management Facility administers “managed objects”.   It provides a consistent foundation for finding object factories, implementing object factories, implementing policy on business object instances, and incorporating System Management functionality such as distribution, deployment, monitoring, recovery, etc., with the rest of the Business Object Architecture (see “Figure 2-10 Business Object System Management Model”).

The System Management Facility provides an enterprise factory find capability based on object type, desired location, and kinds of policy objects registered to it. The factory find capability is implemented with the System Management *Library* object. Factories are implemented from interfaces derived from *InstanceManager*. Instance managers, in addition to being business object factories, provide the mechanism for implementing initialization and validation policies, independent of the business object instances. In order to maintain consistency and integrity, these policies are intended to be driven from the business object model. Each business object is created by, and associated with, an instance manager. Business objects are created as policy-driven objects which utilize the Systems Management framework in support of the requirements for custom rules and policies.

Within the context of the Business Object Architecture, a Business Application Component is the unit of replacable functionality (plug and play). The Business Application Component is a primary unit to be administered by the System Management Facility. Each Business Application Component has a Component Manager which coordinates System Management functionality such as distribution, deployment, monitoring, recovery, etc., with the configuration of business objects contained in the component.

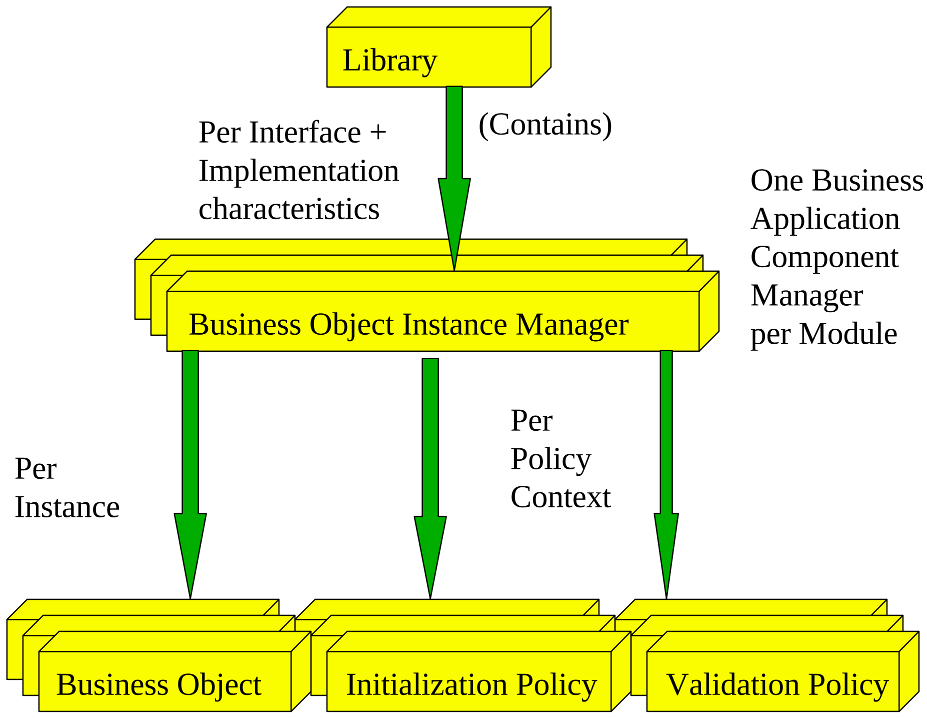
Library
(Contains)
Per Interface + Implementation characteristics
One Business Application Component Manager per Module
Business Object Instance Manager
Per Instance
Per Policy Context
Business Object
Initialization Policy
Validation Policy

Figure 2-10 Business Object System Management Model

2.2.8 Life Cycle Model

Each Business Object is a LifeCycleObject and has the full semantics of compound life cycles, including operations on graphs of related objects. The semantics of copy, move, delete are controlled by the Business Object Model and ensure preservation of referential integrity.

Creation of a Business Object (as well as roles, relationships, etc.) is performed by an Instance Manager, which is a Factory. Finding Business Object factories is done via the *factory_finder* operation defined for *Library* within the System Management Facility. Instances of Business Objects can be found via the Instance Manager *lookup_object* operation. All Business Objects have labels which are unique amongst the contained object instances of the Instance Manager. Other Business Objects, roles, and relationships can be found via navigation from some Business Object.

Instance Managers themselves are created from the System Management *Library* object, which is the factory for instance managers.

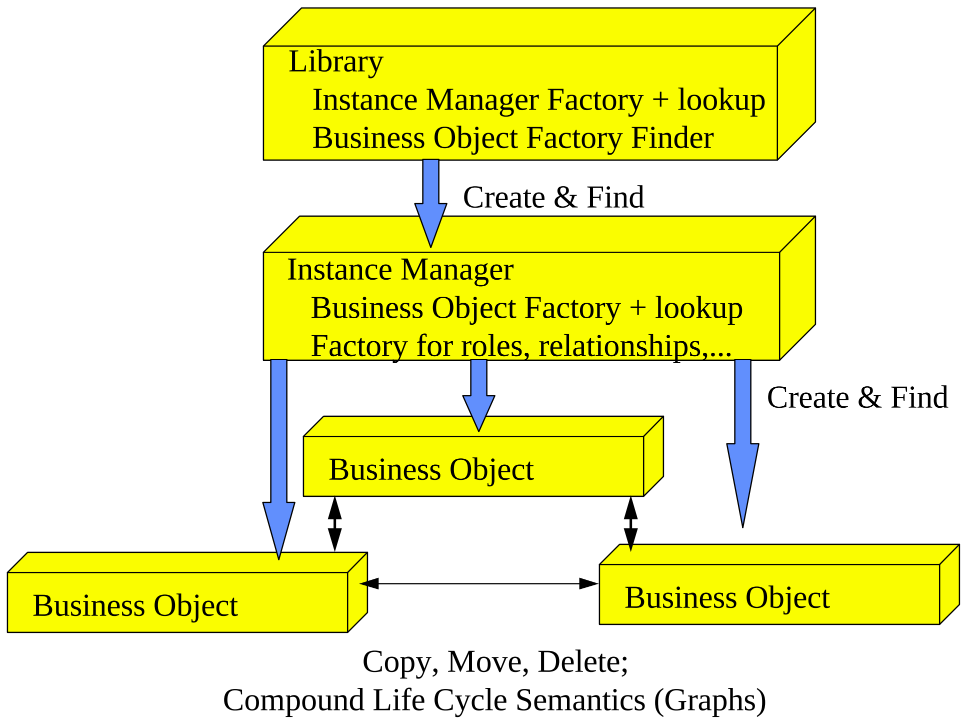
Library
Instance Manager Factory + lookup
Business Object Factory Finder
Create & Find
Instance Manager
Business Object Factory + lookup
Factory for roles, relationships,...
Create & Find
Business Object
Business Object
Business Object
Copy, Move, Delete;
Compound Life Cycle Semantics (Graphs)

Figure 2-8 Business Object Life Cycle Model

2.2.9 Contracted Roles

To achieve the objective of plug & play business components, encapsulation must occur in two directions. The external world knows *what* a business object does, but does not know *how* it does it. From the inside looking out, the business object requires (contracts) a set of service roles. The business object specification must describe *what* contracted service roles are required, but should not specify *which* business objects implement them or *how* they are implemented.

Each contracted role has one interface definition. The contracted role interface defines operations which may be implemented in any arbitrary (semantic-compliant) manner, and may utilize techniques such as delegation, transformation, aggregate propagation, or complex sequences of operations to multiple business objects.

As a possible scenario for the evolution of a contracted role, consider a business object which is initially defined within a narrow domain context. In this case, the contracted role may simply “pass through” operations to a business object which is related via a defined relationship “role”. As system contexts merge into larger contexts, vertical domain frameworks standards evolve, or new common business objects are standardized, the contracted role implementation will change. The change will reflect a mapping between the new configuration of business objects and the original contracted role semantic.

The mix of business objects, legacy systems, purchased systems, and standards-compliance will be different, and evolve differently, for each enterprise. The contracted role mechanism helps preserve the integrity and longevity of individual business objects within these dynamic and enterprise-specific environments.

As shown in “Figure 2-11 Contracted Roles: Excapsulation”, the contracted role object is implemented as a plug & play subcomponent, separate from the business object subcomponent. Thus, enterprise-specific tailoring of the contracted role implementation is possible without altering the business object.

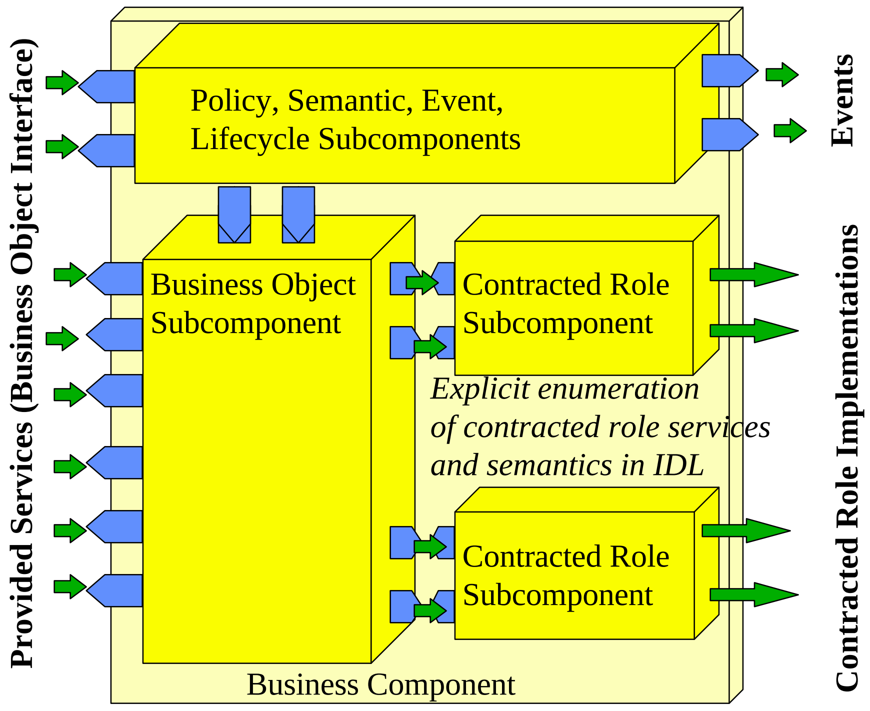
Provided Services (Business Object Interface)
Policy, Semantic, Event, Lifecycle Subcomponents
Events
Business Object Subcomponent
Contracted Role Subcomponent
Explicit enumeration of contracted role services and semantics in IDL
Contracted Role Subcomponent
Contracted Role Implementations
Business Component

Figure 2-11 Contracted Roles: Excapsulation

2.2.10 Other Considerations

- Startup Service is assumed to encompass the Transaction Service, thus ensuring reliable recovery of all business objects to their last committed state.
- Transaction Service is assumed to have capabilities which include load balancing, queueing, recovery, scheduling, redundancy, fault tolerance.
- Version control of the Business Object Model and other aspects of the development environment are assumed to be in the scope of the Meta Object Facility, not the Business Object Facility.

- Some Change Management considerations, particularly run-time versioning issues, are considered to be within the long-term scope of the Business Object Facility. OMG specifications related to versioning and change management should be incorporated into the Business Object Facility specification as they become available. However, it is premature and speculative to incorporate versioning and change management in this version of the Business Object Facility specification.
- The deployment of configurations of components, including the management of implementation inter-dependencies, is considered to be within the scope of Systems Management, not the Business Object Facility.
- Implementation of Business Object Replication is considered to be within the scope of the Replication Service, not the Business Object Facility. OMG specifications related to replication should be incorporated into the Business Object Facility specification as they become available. It is premature and speculative to incorporate replication in this version of the Business Object Facility specification.
- Persistence implementation is out of scope. The persistence mechanism is intentionally not exposed in the interface definitions. As a practical matter, conformity on persistence implementation is not likely to occur soon. There is too much variance in technical approaches and constraints, from wrappered legacy applications to transparent object data base store. The defined Persistent Object Service will not satisfy the requirement for plug & play data store technology.
- Event Services are assumed to have reliability, guaranteed message sequencing, persistence of events, persistence of event subscription, and assurance that events are not lost on failure/recovery of subscriber. Mobile agents will require persistent events, surivival through intermittent connectivity, etc. All of these issues are in the scope of Event Services and appropriate Quality of Service implementations. These issues are considered outside the scope of the Business Object Facility.
- Monitoring and recovery. The Business Application Component Manager has responsibility for monitoring and recovery associated with business objects. The BAC Manager will utilize features of the System Management Facility to implement monitoring and recovery. System Management specifications related to monitoring and recovery should be incorporated into the Business Object Facility specification as they become available. It is premature and speculative to define specific monitoring and recovery measures in this version of the Business Object Facility specification.
- Logging services. Logging is considered to be within the long-term scope of the Business Object Facility. OMG specifications related to logging services should be incorporated into the Business Object Facility specification as they become available. It is premature and speculative to incorporate logging in this version of the Business Object Facility specification.
- Notification/message service. Mechanisms for issuing notifications via email, beeper, etc. are assumed to be addressed in a future OMG specification. It is further assumed that such a service will be based on the Event Services. Since business objects are enabled to supply events, and those events can be routed to a Notification/message service, explicit considerations for Notification/messaging are not necessary withinthe Business Object Facility specifications.

2.3 Interface Description: OMG IDL

```
module CfBusiness {

// TransactionalObject
	interface TransactionalObject: :: CosTransactions::TransactionalObject {};

// Instance
	interface Instance: TransactionalObject,
		:: PolicyRegions::PolicyDrivenBase {};

	interface InstanceManager: :: ManagedInstances::InstanceManager {
			readonly attribute ::CfMetaObject::MetaObject meta_object,
	};
	interface InstanceInitialization: ::PolicyObjectAdmin::InitializationPolicy{};
	interface InstanceValidation: :: PolicyObjectAdmin::ValidationPolicy {
			attribute ::CosLifeCycle::Criteria validation;
	};

// Module Manager
	interface ModuleManager: InstanceManager {
		readonly attribute ReqId startup_id;
		readonly attribute ReqId shutdown_id;
		startup(in NVList arg_list);
		shutdown(in NVList arg_list);
	};
	interface ModuleValidation: InstanceValidation {};

// Operations
	interface Operations: ::CosCompoundLifeCycle::Operations,
		TransactionalObject{};

	interface OperationsManager: InstanceManager,
		::CosCompoundLifeCycle::OperationsFactory {};

// Node
	interface Node: Instance,
		::CosCompoundLifeCycle::Node,
		::CosPropertyService::PropertySetDef,
		::CosCompoundExternalization::Node {};

	interface NodeManager: InstanceManager,
		::CosStream::StreamableFactory,
		::CosPropertyService::PropertySetDefFactory {};
```

```
        interface NodeInitialization: InstanceInitialization,
                ::CosGraphs::NodeFactory {};
        interface NodeValidation: InstanceValidation {};

// ReferencesRole
        interface ReferencesRole: TransactionalObject,
                ::CosLifeCycleReference::ReferencesRole,
                ::CosExternalizationReference::ReferencesRole {};

        interface ReferencesRoleManager: InstanceManager,
                ::CosRelationships::RoleFactory {
                const ScopedName role_type_value = “CfBusiness::ReferencesRole”;
                const maximum_cardinality maximum_cardinality _value =  unbounded;
                const minimum_cardinality minimum_cardinality _value =  0;
                const ScopedName related_object_types_value =  “CfBusiness::Node”;
        };

// ReferencedByRole
        interface ReferencedByRole: TransactionalObject,
                ::CosLifeCycleReference::ReferencedByRole,
                ::CosExternalizationReference::ReferencedByRole {};

        interface ReferencedByRoleManager: ReferencesRoleManager {
                const ScopedName role_type_value = “CfBusiness::ReferencedByRole”;
        };

// Relationship
        interface Relationship: TransactionalObject,
                ::CosLifeCycleReference::Relationship,
                ::CosExternalizationReference::Relationship {};

        interface RelationshipManager: InstanceManager,
                ::CosRelationships::RelationshipFactory {
                const ScopedName relationship_type_value = “CfBusiness::Relationship”;
                const ScopedName named_role1 = “CfBusiness::ReferencesRole”;
                const ScopedName named_role2 = “CfBusiness::ReferencedByRole”;
        };

// Entity
        module Entity {
                interface Entity: :: Node,
                ::CosTypedEventComm::TypedPushSupplier ,
                ::CosTypedEventComm::TypedPushConsumer {
                        readonly attribute
                        CosTypedEventChannelAdmin::TypedEventChannel
                        event_channel;
                    readonly attribute
                       CosTypedEventChannelAdmin::TypedProxyPushConsumer
                        consumer;
                    readonly attribute CosEventChannelAdmin::ProxyPushSupplier
                        supplier;
                    CfBusiness::Contract::Role ContractRole(ScopedName Role);
                };

                interface EntityManager: NodeManager {
                    attribute
                        CosTypedEventChannelAdmin::TypedEventChannel
                        event_channel;
                    attribute
                       CosTypedEventChannelAdmin::Key supported_interface;
                    attribute CosEventChannelAdmin::Key uses_interface;
                };
                interface EntityInitialization: NodeInitialization {};
                interface EntityValidation: NodeValidation {};
                interface EntityEventsSupplied{
                    define_property_with_mode(in Entity entity,
                        in PropertyName property_name, in any property_value,
                        in PropertyModeType property_mode);
                    delete_property (in Entity entity, in PropertyName
property_name);
                    add_role (in Entity entity, in Role a_role);
                    remove_role(in Entity entity, in ::CORBA::InterfaceDef of_type);
                };
        };  /* Entity module */

        module Event {
                interface EntityEventsConsumed{
                    readonly attribute Entity entity;
                    void destroy();
                    /* any event operations */
                };
        };  /* Entity module */

        module Contract {
                interface Role {
                    const ScopedName role_name = "CfBusiness::ReferencesRole";
                    readonly attribute Entity contracted_by;
                    /* following operations are being contracted */
                    }
          };  /* Contract module */

};   /* CfBusiness module */

// Relationship macros

// General Relationship
#define CfRelates\
(module,node1,role1,role1min,role1max,relationship,role2,role2min,role2max,node2)\
          interface role1 : ::CfBusiness::ReferencesRole {};\
          interface role1##Manager: ::CfBusiness::ReferencesRoleManager {\
                    const ScopedName role_type_value = #module "::" #role1;\
                    const maximum_cardinality maximum_cardinality _value =  role1max;\
                    const minimum_cardinality minimum_cardinality _value =  role1min;\
                    const ScopedName related_object_types_value =  #module "::" #node1;\
          };\
          interface relationship: ::CfBusiness::Relationship {};\
          interface relationship##Manager: ::CfBusiness::RelationshipManager {\
                    const ScopedName relationship_type_value = #module "::" relationship;\
                    const ScopedName named_role1 = #module "::" #role1;\
                    const ScopedName named_role2 = #module "::" #role2;\
          };\
          interface role2 : ::CfBusiness::ReferencedByRole {};\
          interface role2##Manager: ::CfBusiness::ReferencedByRoleManager {\
                    const ScopedName role_type_value = #module "::" #role2;\
                    const maximum_cardinality maximum_cardinality _value =  role2max;\
                    const minimum_cardinality minimum_cardinality _value =  role2min;\
                    const ScopedName related_object_types_value =  #module "::" #node2;\
          };\

// Many to Many References
#define CfReferences(module, node1,role1, relationship,role2,node2)\
CfRelates(module,node1,role1,0,unbounded,relationship,role2,0,unbounded,node2)

// Containment
#define CfContains (module,node1,role1, relationship,role2,node2)\
CfRelates(module,node1,role1,0,unbounded,relationship,role2,1,1,node2)
```

2.4 Interface Description: Behavior

2.4.1 Business module

2.4.1.1 TransactionalObject

interface TransactionalObject: :: CosTransactions::TransactionalObject {};

This is an abstract specification for all nodes, roles, and relationships within the Business Object Architecture. Each object is persistent and has transactional semantics. A business object typically contains or indirectly refers to persistent data that can be modified by requests.

2.4.1.1.1 Inherited Interfaces

2.4.1.1.1.1 ::CosTransactions::TransactionalObject

The semantics of ::CosTransactions::TransactionalObject are inherited by TransactionalObject. These semantics permit an operation within TransactionalObject to be invoked with or without an established transaction context. Transaction Service operations have behavior which depends upon whether or not a transaction context has been defined. Within the Business Object Facility, the existence (or lack) of a transaction context is an explicit pre-condition of virtually every operation on TransactionalObjects. Each operation can be seen to operate in one of the following invariant roles:

- **TransactionRequired**. The operation requires a transaction context upon entry. Will raise a *TransactionRequired* exception if invoked outside of a transaction context.
- **TransactionalClient**. The operation does not require a transaction context upon entry. The operation may begin a transaction, in which case it must complete the transaction prior to returning control to the client (i.e., the post condition of any operation must ensure that the transaction context is the same as the pre condition transaction context). A *SubtransactionsUnavailable* exception is raised if the client thread already has an associated transaction and the Transaction Service implementation does not support nested transactions.

IDL will be used to express these behavioral semantics as exceptions. An operation will be defined to raise *TransactionRequired* if the operation requires a transaction context upon entry. An operation will be defined to raise *SubtransactionsUnavailable* if the operation does not require a transaction context upon entry.

Unless otherwise stated, all operations either inherited or explicitly stated in any interface derived from TransactionalObject will, in addition to defined exceptions, raise the *TransactionRequired* exception.

2.4.1.1.1.2 Persistence

Persistence must be implemented for all objects derived from TransactionalObject. Any object derived from TransactionalObject which has multiple inheritance requires persistence on all inherited interfaces and implementations, unless otherwise stated in this specification.

Persistence is implementation dependent, but must adhere to transaction semantics. Persistence may be, but is not required to be, implemented using the Persistent Object Service.

2.4.1.2 Instance

```
        interface Instance: TransactionalObject,
                :: PolicyRegions::PolicyDrivenBase {};
```

This interface defines an object which combines the semantics of a persistent TransactionalObject and policy driven managed instances of System Management.

2.4.1.2.1 Inherited Interfaces

2.4.1.2.1.1 TransactionalObject

The semantics of TransactionalObject are inherited by Instance, with no additional specializations.

2.4.1.2.1.2 :: PolicyRegions::PolicyDrivenBase

The semantics and interface definitions for :: PolicyRegions::PolicyDrivenBase are inherited by Node, with no additional specializations.

2.4.1.2.2 InstanceManager

```
        interface InstanceManager: :: ManagedInstances::InstanceManager {
                        readonly attribute ::CfMetaObject::MetaObject meta_object;
        };
```

2.4.1.2.2.1 readonly attribute ::CfMetaObject::MetaObject meta_object

This attribute is an object reference to the ::CfMetaObject::MetaObject which defines the Business Object within the Business Object Model.

2.4.1.2.2.2 ::ManagedInstances:: InstanceManager

Instance managers provide operations for the registration of initialization policy objects and validation policy objects. These policy objects are specifically associated with the type of policy-driven based managed object supported by the instance manager. Initialization policy objects support operations to define the intial (default) values of the policy-drive object attributes. Validation policy objects support methods that can validate initial values or changes to object attributes, and also methods that can be used to control object behaviours.
The semantics of ::ManagedInstances:: InstanceManager are inherited by InstanceManager, with the following specializations and additions:

- get_instances_interface returns InterfaceDef for Instance.
- create_object operation returns a Instance.

- create_object operation will call initialize_policy_driven_object for each InstanceInitialization related to Instance.

2.4.1.2.3 InstanceInitialization

```
interface InstanceInitialization: ::
        PolicyObjectAdmin::InitializationPolicy{
                attribute ::CosLifeCycle::Criteria initialization;
};
```

attribute ::CosLifeCycle::Criteria initialization

This attribute contains a sequence of name-value pairs which are to be used to initialize the new object. The criteria defined for InstanceInitialization are:

criterion name	type of criterion value	interpretation

This list will be extended by specializations of InstanceInitialization.

2.4.1.2.3.1 PolicyObjectAdmin::InitializationPolicy

The semantics and interface definitions for PolicyObjectAdmin::InitializationPolicy are inherited by InstanceInitialization , with the following additional specializations:

- get_policy_driven_object_type returns "::CfBusiness::Instance".
- initialize_policy_driven_object will perform additional initialization of a Instance based on *initialization*.

2.4.1.2.4 InstanceValidation

```
interface InstanceValidation: :: PolicyObjectAdmin::ValidationPolicy {
                attribute ::CosLifeCycle::Criteria validation;
};
```

attribute ::CosLifeCycle::Criteria validation

This attribute contains a sequence of name-value pairs which are to be used to validate the object. The criteria defined for InstanceValidation are:

criterion name	type of criterion value	interpretation

This list will be extended by specializations of InstanceValidation.

2.4.1.2.4.1 PolicyObjectAdmin::ValidationPolicy

The semantics and interface definitions for PolicyObjectAdmin::ValidationPolicy are inherited by InstanceValidation , with the following additional specializations:

- get_policy_driven_object_type returns "::CfBusiness::Instance".

- validate_policy_driven_object will perform additional validation of a Instance, as specified in *validation*.

2.4.1.3 Module

2.4.1.3.1 ModuleManager

```
interface ModuleManager: InstanceManager {
        readonly attribute ReqId startup_id;
        readonly attribute ReqId shutdown_id;
        startup(in NVList arg_list);
        shutdown(in NVList arg_list);
};
```

The responsibility of the ModuleManager is to perform interaction between the System Management Facility and the configuration of objects embodied by a business application component (i.e., a module). The ModuleManager is governed by a set of validation policies. Although an InstanceManager, the ModuleManager does not create object instances.

When the ModuleManager is created, it registers itself with the ORB shutdown and startup services. The request identifiers are recorded in the shutdown_id and startup_id attributes, respectively. The shutdown and startup registration will cause startup() to be invoked when the ORB starts and shutdown() when the ORB shuts down.

The details of the startup and shutdown operations are implementation-dependent. Note that the Startup Service is assumed to be used by the Transaction Service, thus ensuring reliable recovery of all business objects to their last committed state. The startup and shutdown operations are thus intended to ensure restoration, as required, of other dependent resources.

The semantics of InstanceManager are inherited by ModuleManager, with the following specializations and additions:
- create_object(...) will always raise “NoFactory” exception.

2.4.1.3.2 ModuleValidation

```
interface ModuleValidation: InstanceValidation {};
```

The semantics and interface definitions for InstanceValidation are inherited by ModuleValidation, with the following additions and specializations:
- *validate_policy_driven_object*(...) is passed the ModuleManager as the *object_to_validate*.

2.4.1.4 Operations

```
interface Operations: ::CosCompoundLifeCycle::Operations,
          TransactionalObject{};
```

2.4.1.4.1 Inherited Interfaces

2.4.1.4.1.1 TransactionalObject

The semantics and interface definitions for TransactionalObject are inherited by Operations , with no additions and specializations.

2.4.1.4.1.2 ::CosCompoundLifeCycle::Operations

The semantics of :: CosCompoundLifeCycle::Operations are inherited by Operations, with the following specializations and additions:

- The argument “there”
  in both operations copy and move should be set to a
  ::ManagedInstances::Library.
- The argument “starting_node”
  in operations copy, move, and remove should be set to a
  Node.
- The operations copy, move, and remove should be performed in a transaction context which preserves ACID properties. In particular, it should be possible to rollback the operation.

2.4.1.4.2 OperationsManager

```
interface OperationsManager: InstanceManager,
          ::CosCompoundLifeCycle::OperationsFactory {};
```

The semantics of InstanceManager are inherited by OperationsManager, with the following specializations and additions:

- get_instances_interface returns InterfaceDef for Operations.
- create_object(...) will always raise “NoFactory” exception.

The semantics of ::CosCompoundLifeCycle::OperationsFactory are inherited by OperationsManager, with the following specializations and additions:

- create_compound_operations() returns a Operations.
- create_compound_operations()will call initialize_policy_driven_object for each OperationsInitialization related to new Operations.
- create_compound_operations() implements semantics of ::CosCompoundLifeCycle::OperationsFactory::create_compound_operations().

2.4.1.5 Node

```
interface Node: Instance,
          ::CosCompoundLifeCycle::Node,
          ::CosCompoundExternalization::Node,
          ::CosPropertyService::PropertySetDef {
```

}

This interface defines an object which has the semantics of the persistent Instance, a relationship node, compound life cycle objects, externalizable compound objects, dynamic property management, policy semantics, and managed instances of System Management. A "Node" supports the life cycle interfaces (copy, move, remove) for graphs of business objects. A Node supports full referential integrity semantics, including cascaded propagation of remove semantics through arbitrary graphs of business objects. Node also supports externalization semantics for graphs of objects, as defined in the Externalization service.

2.4.1.5.1 Inherited Interfaces

2.4.1.5.1.1 Instance

The semantics and interface definitions for Instance are inherited by Node , with the following additions and specializations:

- The argument "there"
  in both LifeCycle operations copy and move should be set to a
  ::ManagedInstances::Library.
- The LifeCycle copy, move, and remove operations should be implemented as indicated in section A.3 of LifeCycle services:
  - Node creates an object that supports the Operations interface.
  - Node issues corresponding life cycle requests to the Operations, using itself as the starting node.
  - Node issues destroy request to destroy the compound operation.

2.4.1.5.1.2 ::CosCompoundLifeCycle::Node

The semantics and interface definitions for ::CosCompoundLifeCycle::Node, ::CosObjectIdentify::IdentifiableObject and ::CosGraphs::Node are inherited by Node, with the following additions and specializations:

- The operation
  get_life_cycle_object();
  will return itself.
- The argument "there"
  in both operations copy_node and move_node should be set to a
  ::ManagedInstances::Library.
- The attribute "related_object" will return itself.

2.4.1.5.1.3 ::CosCompoundExternalization::Node

The semantics and interface definitions for ::CosCompoundExternalization::Node and ::CosStream::Streamable are inherited by Node, with the following additions and specializations:

- The argument "there"
  in operations "internalize_node" and "internalize_from_stream" should be set to a
  ::ManagedInstances::Library.
- operation externalize_to _stream should follow semantics for a Node, namely:

    - Delegate the externalize_to_stream() to the StreamIO by invoking write_graph().
    - The Streamable (Node) object would subsequently receive an externalize_node_to_stream() and write out its internal state.
    - write_object() should not be called by this object for other nodes in the graph.
- operation internalize_from_stream should follow semantics for a Node, namely:
    - Delegate the internalize_from_stream() to the sourceStreamIO by invoking read_graph() with the same FactoryFinder argument as the there parameter passed in to the internalize_from_stream() operation.
    - The Streamable (Node) object would subsequently receive an internalize_node_from_stream() and read in its internal state.
    - read_object() should not be called by this object for other nodes in the graph.

2.4.1.5.1.4 ::CosPropertyService::PropertySetDef

The semantics and interface definitions for ::CosPropertyService::PropertySetDef are inherited by Node , with no additional specializations.

2.4.1.5.2 NodeManager

```
interface NodeInitialization: InstanceInitialization,
        ::CosGraphs::NodeFactory,
        ::CosStream::StreamableFactory,
        ::CosPropertyService::PropertySetDefFactory {};
```

The semantics of InstanceManager are inherited by NodeManager, with the following specializations and additions:

- get_instances_interface returns InterfaceDef for Node.
- create_object operation returns a Node.
- create_object operation will call initialize_policy_driven_object for each NodeInitialization related to new Node.
- create_object implements semantics of ::CosGraphs::NodeFactory::create_node and ::CosStream::StreamableFactory::create_uninitialized().
- create_object implements semantics of ::CosPropertyService::PropertySetDefFactory ::create_propertysetdef().

2.4.1.5.2.1 ::CosGraphs::NodeFactory

The semantics of ::CosGraphs::NodeFactory are inherited by NodeManager, with the following specializations and additions:

- create_node(in Object related_object) is implemented as create_object(...). “related_object” is the new Node.

2.4.1.5.2.2 ::CosStream::StreamableFactory

The semantics of ::CosStream::StreamableFactory are inherited by NodeManager, with the following specializations and additions:

- create_uninitialized() is implemented as create_object(...).

2.4.1.5.2.3 ::CosPropertyService::PropertySetDefFactory

The semantics of ::CosPropertyService::PropertySetDefFactory are inherited by NodeManager, with the following specializations and additions:

- create_propertysetdef, create_constrained_propertysetdef, create_initial_propertysetdef are implemented as create_object, with initialization criteria set to satisfy semantics of operation defined in ::CosPropertyService::PropertySetDefFactory.

2.4.1.5.3 NodeInitialization

interface NodeInitialization: InstanceInitialization {};

The semantics and interface definitions for InstanceInitialization are inherited by NodeInitialization , with the following additional specializations:

- get_policy_driven_object_type returns "::CfBusiness::Node".
- initialize_policy_driven_object will perform following additional initialization of a Node:
    - The set of role types specified in "node" of attribute initializationCriteria will be added to the created Node.
    - Any "property" criteria in initialization attribute will be used to initialize properties for the created Node.
- **attribute initialization**. This attribute is extended as follows:

criterion name	type of criterion value	interpretation
"property"	::CosPropertyService::PropertyDefs	initialization parameters for Property Service.
"node"	sequence <::CORBA::InterfaceDef>	Set of role types to add to new node at initialization.

2.4.1.5.4 NodeValidation

interface Node Validation: InstanceValidation {};

The semantics and interface definitions for InstanceValidation are inherited by NodeValidation , with the following additional specializations:

- get_policy_driven_object_type returns "::CfBusiness::Node".
- validate_policy_driven_object will perform following additional validation of a Node:
    - The set of role types specified in "node" of attribute validationCriteria must be defined for the Node. A subtype of role type will satisfy the criteria.
    - The "filter" constraint for "property" within the criteriaValidation attribute is tested for satisfaction of the constraint.
- **attribute validation**. This attribute is extended as follows:

criterion name	type of criterion value	interpretation
"property"	Criteria property	criteria for properties.

“node”	sequence <::CORBA::InterfaceDef>	Set of role types which must exist for node.

- The “Criteria property” definition is as follows:

criterion name	**type of criterion value**	**interpretation**
“filter”	string	constraint on object properties, expressed in the Constraint Language specified in B.2 of LifeCycle specification.

2.4.1.6 ReferencesRole

```
interface ReferencesRole: TransactionalObject,
        ::CosLifeCycleReference::ReferencesRole,
        ::CosExternalizationReference::ReferencesRole {
};
```

2.4.1.6.1 Inherited Interfaces

2.4.1.6.1.1 TransactionalObject

The semantics and interface definitions for TransactionalObject are inherited by ReferencesRole , with no additional specializations.

2.4.1.6.1.2 ::CosLifeCycleReference::ReferencesRole

The semantics and interface definitions for ::CosLifeCycleReference::ReferencesRole are inherited by ReferencesRole, with no specializations.

2.4.1.6.1.3 ::CosExternalizationReference::ReferencesRole

The semantics and interface definitions for ::CosExternalizationReference::ReferencesRole are inherited by ReferencesRole, with no specializations.

2.4.1.6.2 ReferencesRoleManager

```
interface ReferencesRoleManager: InstanceManager,
        ::CosRelationships::RoleFactory {
        const ScopedName role_type_value = “CfBusiness::ReferencesRole”;
        const maximum_cardinality maximum_cardinality _value =  unbounded;
        const minimum_cardinality minimum_cardinality _value =  0;
        const ScopedName related_object_types_value =  “CfBusiness::Node”;
};
```

This interface enables the factory_finder capabilities of the System Management Facility to be used to find the factory for ReferencesRole. The ReferencesRole is created using the create_role operation, which does not add objects to the set, nor require uniquely labeled member objects. Thus, the advantage of flexible, consistent factory finding is combined with the elimination of substantial System Management overhead. This approach does, however, have the restriction that there are no policies associated with roles. This approach has been taken for all roles and relationships.

Constants are used to incorporate semantics into the interface definition. This approach enables simplification of interface definition for specialized roles and relationships.

The semantics of InstanceManager are inherited by ReferencesRoleManager, with the following specializations and additions:

- get_instances_interface returns *role_type*.
- create_object(...) will always raise "NoFactory" exception.

The semantics of ::CosRelationships::RoleFactory are inherited by ReferencesRoleManager, with the following specializations and additions:

- create_role(...) operation returns an object of type *role_type*.
- create_role(...) implements semantics of ::Relationships::RoleFactory::create_role(...).
- the value of attribute *role_type* is the ::CORBA::InterfaceDef found by ::CORBA::lookup(*role_type_value*).
- the value of attribute *maximum_cardinality* is the constant *maximum_cardinality_value*.
- the value of attribute *minimum_cardinality* is the constant *minimum_cardinality_value*.
- the value of attribute *related_object_types* is a single value sequence containing the ::CORBA::InterfaceDef found by ::CORBA::lookup(*related_object_types _value*).

2.4.1.7 ReferencedByRole

```
interface ReferencedByRole: TransactionalObject,
        ::CosLifeCycleReference::ReferencedByRole,
        ::CosExternalizationReference::ReferencedByRole {
}
```

The semantics and interface definitions for TransactionalObject, ::CosLifeCycleReference::ReferencedByRole, and ::CosExternalizationReference::ReferencedByRole are inherited by ReferencedByRole, with no additional specializations.

2.4.1.7.1 ReferencedByRoleManager

```
interface ReferencedByRoleManager: ReferencesRoleManager {
        const ScopedName role_type_value = "CfBusiness::ReferencedByRole";
};
```

The semantics of ReferencesRoleManager are inherited by ReferencedByRoleManager, with a *role_type_value* override.

2.4.1.8 Relationship

```
interface Relationship: TransactionalObject,
        ::CosLifeCycleReference::Relationship,
        ::CosExternalizationReference::Relationship {
}
```

The semantics and interface definitions for TransactionalObject, ::CosLifeCycleReference:: Relationship, and ::CosExternalizationReference:: Relationship are inherited by Relationship, with no additional specializations.

2.4.1.8.1 RelationshipManager

```
interface RelationshipManager: InstanceManager,
        ::CosRelationships::RelationshipFactory {
        const ScopedName relationship_type_value = “CfBusiness::Relationship”;
        const ScopedName named_role1 = “CfBusiness::ReferencesRole”;
        const ScopedName named_role2 = “CfBusiness::ReferencedByRole”;
        };
```

The semantics of InstanceManager are inherited by RelationshipManager, with the following specializations and additions:

- get_instances_interface returns *relationship_type*.
- create_object(...) will always raise “NoFactory” exception.

The semantics of ::Relationships::RelationshipFactory are inherited by RelationshipManager, with the following specializations and additions:

- create(...) operation returns an object of type *relationship_type*.
- create(...) implements semantics of ::Relationships::RelationshipFactory::create (...).
- the value of attribute *relationship_type* is the ::CORBA::InterfaceDef found by ::CORBA::lookup(*relationship_type_value*).
- the value of attribute *named_role_types* is a two value sequence:
    - {named_role1,::CORBA::InterfaceDef found by ::CORBA::lookup(*named_role1*)}.
    - {named_role2,::CORBA::InterfaceDef found by ::CORBA::lookup(*named_role2*)}.

2.4.1.9 Entity

```
interface Entity: :: Node,
        ::CosTypedEventComm::TypedPushSupplier ,
        ::CosTypedEventComm::TypedPushConsumer {
                readonly attribute
                        CosTypedEventChannelAdmin::TypedEventChannel
                        event_channel;
                readonly attribute
                        CosTypedEventChannelAdmin::TypedProxyPushConsumer
                        consumer;
                readonly attribute CosEventChannelAdmin::ProxyPushSupplier
                        supplier;
        };
```

An Entity is a Node which also supports event generation and consumption. As an event supplier, Entity supplies the typed events defined in the interface EntityEventsSupplied. As an event consumer, Entity consumes the typed events defined in the interface EntityEventsConsumed.

When Entity is created, it is associated with an event channel. A TypedProxyPushConsumer is created and connected with the event channel and the Entity. This TypedProxyPushConsumer supports typed events defined in EntityEventsSupplied. The EntityEventsSupplied interface documents the events generated by Entity. To generate a typed event <I>, the following pseudo-code is executed:

```
        ((EntityEventsSupplied *)(consumer->get_typed_consumer()))->operation<I>
```

A TypedProxyPushSupplier is created and connected with the event channel and the Entity. This TypedProxyPushSupplier supports typed events defined in EntityEventsConsumed. When a typed event <I> is received, the following pseudo-code is executed:

```
        ((EntityEventsConsumed *)get_typed_consumer())->operation<I>
```

EntityEventsConsumed is intended to be a mechanism, independent of Entity itself, which catches events and can perform operations on Entity based on those events. EntityEventsConsumed can be replaced, tailored, and specialized to capture events from any combination of events and event suppliers.

It is assumed that:

- The TypedEventChannel supports a federation of TypedEventChannels.
- The TypedEventChannel supports administrative functions which specify event filtering. Filtering should include:
    - Selection of suppliers for a consumer.
    - Selection of event types for a consumer. This is largely satisfied through the TypedEvent mechanisms.
    - Selection based on message content.
- The EventServices have transactional integrity.

readonly attribute CosTypedEventChannelAdmin::TypedEventChannel event_channel;
The TypedEventChannel associated with the Entity at create time.

readonly attribute CosTypedEventChannelAdmin::TypedProxyPushConsumer consumer;
The TypedProxyPushConsumer associated with the Entity at create time.

readonly attribute CosEventChannelAdmin::ProxyPushSupplier supplier;
The ProxyPushSupplier associated with the Entity at create time.

2.4.1.9.1 Inherited Interfaces

2.4.1.9.1.1 Node

The semantics and interface definitions for Node are inherited by Entity, with the following specializations:

- *remove*() will also perform *disconnect_push_consumer*() on *consumer*, *disconnect_push_supplier*() on *supplier*, and *destroy*() on *events_consumed*.

2.4.1.9.1.2 ::CosTypedEventComm::TypedPushSupplier

The semantics and interface definitions for ::CosTypedEventComm::TypedPushSupplier are inherited by Entity, with no specializations.

2.4.1.9.1.3 ::CosTypedEventComm::TypedPushConsumer

The semantics and interface definitions for ::CosTypedEventComm::TypedPushConsumer are inherited by Entity, with the following specializations:

- *get_typed_consumer*() returns the EntityEventsConsumed associated with the Entity at create time.

2.4.1.9.2 EntityManager

```
interface EntityManager: NodeManager {
                attribute
                        CosTypedEventChannelAdmin::TypedEventChannel
                        event_channel;
                attribute
                       CosTypedEventChannelAdmin::Key supported_interface;
                attribute CosEventChannelAdmin::Key uses_interface;
        };
```

The semantics of NodeManager are inherited by EntityManager, with the following specializations and additions:

- *get_instances_interface* returns InterfaceDef for Entity.
- *create_object* operation returns a Entity.

- *create_object* operation will call *initialize_policy_driven_object* for each EntityInitialization related to new Entity.
- *create_object* operation sets Entity *event_channel* attribute to the EntityManager *event_channel* attribute. A TypedProxyPushConsumer is created, based on the *supported_interface* attribute, and a reference is placed in the Entity *consumer* attribute. A ProxyPushSupplier is created, based on the *uses_interface* attribute and a reference is placed in the Entity *supplier* attribute. A EntityEventsConsumed is created corresponding to the *uses_interface*, and a reference is placed in the Entity *events_consumed* attribute.

attribute CosTypedEventChannelAdmin::TypedEventChannel event_channel;
The *event_channel* attribute contains the object reference to the TypedEventChannel to be used when creating objects.

attribute CosTypedEventChannelAdmin::Key supported_interface;
The *supported_interface* attribute represents the EntityEventsSupplied interface.

attribute CosEventChannelAdmin::Key uses_interface;
The *uses_interface* attribute represents the EntityEventsConsumed interface.

2.4.1.9.3 EntityInitialization

```
interface EntityInitialization: NodeInitialization {};
```

The semantics and interface definitions for NodeInitialization are inherited by EntityInitialization , with the following additional specializations:
- get_policy_driven_object_type returns “::CfBusiness:: Entity”.

2.4.1.9.4 EntityValidation

```
interface EntityValidation: NodeValidation {};
```

The semantics and interface definitions for NodeValidation are inherited by EntityValidation , with the following additional specializations:
- get_policy_driven_object_type returns “::CfBusiness:: Entity”.

2.4.1.9.5 EntityEventsSupplied

```
interface EntityEventsSupplied{
        define_property_with_mode(in Entity entity,
                in PropertyName property_name, in any property_value,
                in PropertyModeType property_mode);
        delete_property (in Entity entity, in PropertyName property_name);
        add_role (in Entity entity, in Role a_role);
        remove_role(in Entity entity, in ::CORBA::InterfaceDef of_type);
};
```

This interface defines the events supplied by the Entity object. Each event is specified as an operation, subject to the following restrictions:

- All parameters must be *in* parameters only.
- No return values are permitted.
- The operations may optionally be declared *oneway*.

This interface is primarily informational. Depending upon Event Service implementation, there may or may not be an actual implementation of the operations. The interface definition is referenced by the EntityManager *supported_interface* attribute.

As examples of potential supplied events:

**define_property_with_mode(in Entity entity, in PropertyName property_name,
in any property_value, in PropertyModeType property_mode);**

(Optional). Event supplied whenever a property is defined, the mode is changed, or the value is changed. The event identifies the Entity for which a property characteristic changed, along with the property name, the new value, and the new mode.

delete_property (in Entity entity, in PropertyName property_name);

(Optional). Event supplied whenever a property is deleted. The event identifies the Entity for which a property has been deleted, along with the property name.

add_role (in Entity entity, in Role a_role);

(Optional). Event supplied whenever a role is added to an entity. The event identifies the Entity for which a role has been added, along with the role.

remove_role(in Entity entity, in ::CORBA::InterfaceDef of_type);

(Optional). Event supplied whenever a role is removed. The event identifies the Entity for which a role has been removed, along with the type of role.

2.4.1.9.6 EntityEventsConsumed

```
interface EntityEventsConsumed{
        readonly attribute Entity entity;
        void destroy();
        /* any event operations */
}
```

This interface defines the events consumed by the Entity object. Each event is specified as an operation, subject to the following restrictions:

- All parameters must be *in* parameters only.
- No return values are permitted.
- The operations may optionally be declared *oneway*.

This interface definition serves two purposes:

- defines the interface used to implement typed events for the consumer. It is the interface returned for the get_typed_consumer() operation on Entity.
- defines the interface which must be called from the ProxyPushSupplier. The interface definition is referenced by the EntityManager *uses_interface* attribute.

readonly attribute Entity entity;
Contains the object reference to Entity which is linked to this EntityEventsConsumed.

void destroy();
Destroys the object. Intended for use only via Entity *remove*().

2.4.2 Relationships

A set of macro definitions which simplify the specification of role and relationship interfaces. The Business Object Facility requires each relationship between business objects x and y to be defined in terms of an explicit association between business object x, role x, relationship, role y, business object y.

2.5 Glossary

abort	rollback. **[OMG 95-03-31]**
active	The state of a transaction when processing is in progress and completion of the transaction has not yet commenced. **[OMG 95-03-31]**
atomicity	A transaction property that ensure that if work is interrupted by failure, any partially completed results will be undone. A transaction whose work completes is said to commit. A transaction whose work is completely undone is said to rollback (abort). **[OMG 95-03-31]**
begin	An operation on the Transaction Service which establishes the initial boundary of a transaction. **[OMG 95-03-31]**
behavior	The observable effects of an object performing the requested operation including its results binding. **[OMG CORBA]**
business application component	A configuration of objects, including business objects, which specifies the unit of implementation. The OMG IDL module construct defines the objects and interfaces constituting a business application component.
business meta-object	A meta-object which specifies the interface and semantics for a business object, including at least its business name and definition, attributes, behavior, relationships, rules, policies and constraints.
business object	An object which represents an instantiated thing active in the business domain. A Business Object implements a Business Meta Object specification.
Business Object	Infrastructure used to manage Business Objects. The **Business**

Facility	**Object Facility** constrains Business Object implementations to adhere to their Business Meta Object semantic specification, ensures implementation integrity for the Business Object Model, and enforces architectural principles enabling plug and play business application components.
Business Object Model	An object schema which represents the complete and rigorous semantic specification of interoperating business objects, including their definition, attributes, behavior, relationships, events, rules, policies, constraints, and any other aspect or feature required to be associated with business objects.
client	The code or process that invokes an operation on an object. **[OMG CORBA]**
committed	The property of a transaction or a transactional object, when it has successfully performed the commit protocol. **[OMG 95-03-31]**
completion	The processing required (either by commit or abort) to obtain the durable outcome of a transaction. **[OMG 95-03-31]**
consistency	A property of a transaction that ensures that the transaction's actions, taken as a group, do not violate any of the integrity constraints associated with the state of its associated objects. This requires that the application program is implemented correctly: the Transaction Service provides the functionality to support application data consistency. **[OMG 95-03-31]**
CORBA	Common Object Request Broker Architecture. **[OMG CORBA]**
durability	A transaction property that ensures the results of a successfully completed transaction will never be lost, except in the event of catastrophe. It is generally implemented by a combination of persistent storage and a logging service that provides a backup copy of permanent changes. **[OMG 95-03-31]**
encapsulation	The process of hiding all of the details of an object that do not contribute to its essential characteristics. Typically, the structure of an object and the implementation of its methods are hidden. The terms encapsulation and information hiding are usually interchangeable. **[SEMATECH CIM]**
event	A state change of an object that causes the behaviour of an object. **[OMG 95-12-05]**
factory object	An object that creates another object. **[OMG 95-12-05]**
federation	The principle whereby each comonent retains its autonomy rather than becoming subordinate to another. **[OMG 95-12-05]**
flat transaction	A transaction that has no subtransactions - and that cannot have subtransactions. **[OMG 95-03-31]**
implementation	A definition that provides the information needed to create an object and allow the object to participate in providing an appropriate set of services. An implementation typically

	includes a description of the data structure used to represent the core state associated with an object, as well as definitions of the methods that access that data structure. It will also typically include information about the intended interface of the object. **[OMG CORBA]**
implementation definition language	A notation for describing implementations. The implementation definition language is currently beyond the scope of the ORB standard. It may contain vendor-specific and adapter-specific notations. **[OMG CORBA]**
implementation inheritance	The construction of an implementation by incremental modification of other implementations. The ORB does not provide implementation inheritance. Implementation inheritance may be provided by higher level tools. **[OMG CORBA]**
implementation object	An object that serves as an implementation definition. Implementation objects reside in an implementation repository. **[OMG CORBA]**
implementation repository	A storage place for object implementation information. **[OMG CORBA]**
in-doubt	The state of a transaction if it is controlled by a transaction manager that can not be contacted, so the commit decision is in doubt. **[OMG 95-03-31]**
inheritance	The construction of a definition by incremental modification of other definitions. See interface and implementation inheritance. **[OMG CORBA]**
instance	An object is an instance of an interface if it provides the operations, signatures and semantics specified by that interface. An object is an instance of an implementation if its behavior is provided by that implementation. **[OMG CORBA]**
interface	A listing of the operations and attributes that an object provides. This includes the signatures of the operations, and the types of the attributes. An interface definition ideally includes the semantics as well. An object satisfies an interface if it can be specified as the target object in each potential request described by the interface. **[OMG CORBA]**
interface inheritance	The construction of an interface by incremental modification of other interfaces. The IDL language provides interface inheritance. **[OMG CORBA]**
interface object	An object that serves to describe an interface. Interface objects reside in an interface repository. **[OMG CORBA]**
interface repository	A storage place for interface information. **[OMG CORBA]**
interoperability	The ability for two or more ORBs to cooperate to deliver requests to the proper object. Interoperating ORBs appear to a client to be a single ORB. **[OMG CORBA]**
isolation	A transaction property that allows concurrent execution, but the

	results will be the same as if execution was serialized. Isolation ensures that concurrently executing transactions cannot observe inconsistencies in shared data. **[OMG 95-03-31]**
language binding or mapping	The means and conventions by which a programmer writing in a specific programming language accesses ORB capabilities. **[OMG CORBA]**
life cycle object	An object whose interfaces are defined by the life cycle services, specifically remove, copy, and move. **[OMG 95-12-05]**
Meta Object Facility	The infrastructure used to manage meta-objects and object schemas.
meta-object	An object that represents aspects or features of another object. A meta-object creates and manages representation concepts, along with the actual object instances they represent. Typical examples of meta-objects are model, class, operation, constraint, and association. A collection of meta-object types will provide the primitive building blocks needed to represent the complex semantics of object models such as the business object model.
method	An implementation of an operation. Code that may be executed to perform a requested service. Methods associated with an object may be structured into some or more programs. **[OMG CORBA]**
multiple inheritance	The construction of a definition by incremental modification of more than one other definition. **[OMG CORBA]**
name binding	A name-to-object association. A name binding is always defined relative to a naming context. **[OMG 95-12-05]**
nested transaction	A transaction that either has subtransaction or is a subtransaction on some other transaction. **[OMG 95-03-31]**
object	A combination of state and a set of methods that explicitly embodies an abstraction characterized by the behavior of relevant requets. An object is an instance of an implementation and an interface. An object models a real-world entity, and it is implemented as a computational entity that encapsulates state and operations (internally implemented as data and methods) and responds to requestor services. **[OMG CORBA]**
object creation	An event that causes the existence of an object that is distinct from any other object. **[OMG CORBA]**
object destruction	An event that causes an object to cease to exist. **[OMG CORBA]**
object reference	A value that unambiguously identfies an object. Object references are never reused to identify another object. **[OMG CORBA]**
object schema	A composition of related meta-objects that together represent semantics of the objects being modeled.
OMA	Object Management Architecture. **[OMG 95-12-05]**

OMG	Object Management Group. **[OMG 95-12-05]**
operation	A service that can be requested. An operation has an associated signature, which may restrict which actual parameters are valid. **[OMG CORBA]**
operation name	A name used in a request to identify an operation. **[OMG CORBA]**
ORB	Object Request Broker. Provides the means by which clients make and receive requests and responses. **[OMG CORBA]**
persistent object	An object that can survive the process or thread that created it. A persistent object exists until it is explictly deleted. **[OMG CORBA]**
propagation	A function of the transaction service that allows the Transaction context of a client to be associated with a transactional operation on a server object. The Transaction Service supports both implicit and explicit propagation of transaction context. **[OMG 95-03-31]**
recoverable object	An object whose data is affected by committing or rolling back a transaction. **[OMG 95-03-31]**
recoverable server	An object that registers a resource (not necessarily itself) with a transaction coordinator to participate in transaction completion. **[OMG 95-03-31]**
referential integrity	The property ensuring that an object reference that exists in the state associated with an object reliably identifes a single object. **[OMG CORBA]**
request	A client issues a request to cause a service to be performed. A request consists of an operation and zero or more actual parameters. **[OMG CORBA]**
resource	An object in the Transaction Service that is registered for involvement in two-phase commit. Corresponds to a resource manager. **[OMG 95-03-31]**
resource manager	An X/Open term for a component which manages the integrity of the state of a set of related resources. **[OMG 95-03-31]**
results	The information returned to the client, which may include values as well as status information indicating that exceptional conditions were raised in attempting to perform the requested service. **[OMG CORBA]**
security service	An object service which provides identifications of users (authentication), controls access to resources (authorization), and provides auditing of resource access. **[OMG 95-03-31]**
server	A process implementing one or more operations on one or more objects. **[OMG CORBA]**
server object	An object providing response to a request for a service. A given object may be a client for some requests and a server for other requests. **[OMG CORBA]**

signature	Defines the parameters of a given operation including their number order, data types, and passing mode; the results if any; and the possible outcomes (normal vs. exceptional) that might occur. **[OMG CORBA]**
single inheritance	The construction of a definition by incremental modification of one definition. Contrast with multiple inheritance. **[OMG CORBA]**
state	The time-varying properites of an object that affect that object's behavior. **[OMG CORBA]**
synchronous request	A request where the client pauses to wait for completion of the request. Contrast with deferred synchronous request and one-way request. **[OMG CORBA]**
thread	The entity that is currently in control of the processor. **[OMG 95-03-31]**
transaction	A collection of operations on the physical and abstract application state. **[OMG 95-03-31]**
transactional client	An arbitrary program that can invoked operations of many transactional objects in a single transaction. Not necessarily the transaction originator. **[OMG 95-03-31]**
transaction context	The transaction information associated with a specific thread. **[OMG 95-03-31]**
transactional operation	An operation on an object that participates in the propagation of the current transaction. **[OMG 95-03-31]**
transaction originator	An arbitrary program - typically, a transactional client, but not necessarily an object - that begins a transaction. **[OMG 95-03-31]**
transaction manager	A system comonent that implements the protocol engine for 2-phase commit protocol. **[OMG 95-03-31]**
transactional object	Strictly speaking, an object that offers at least one transactional operation, and thus requiring the ORB and the Transaction Service to propagate transaction context - but usually used to refer to an object none of whose operations are affected by being invoked within the scope of a transaction. **[OMG 95-03-31]**
transactional server	A collection of one or more objects whose behavior is affected by the transaction, but have no recoverable states of their own. **[OMG 95-03-31]**
transaction service	An Object Service that implements the protocols required to guarantee the ACID (atomcity, consistency, isolation, and durability) properties of transactions. **[OMG 95-03-31]**
transient object	An object whose existence is limited by the lifetime of the process or thread that created it. **[OMG CORBA]**
two-phase commit	A transaction manager protocol for ensuring that all changes to recoverable resources occur atomically and furthermore, the failure of any resource to complete will cause all other resource

	to undo changes. **[OMG 95-03-31]**
typed event	An event for which an interface is defined in terms of IDL. **[OMG 95-12-05]**
wrapper	A software implementation which forms a layer surrounding another implementation for the purposes of presenting interface and functional behavior required by other implementations. The need for wrappers often arises when migrating existing applications to embrace a new, more advanced approach. This typically occurs when migration is preferred to wholesale replacement for cost/benefit reasons. **[SEMATECH CIM]**

3. RESOLUTION OF TECHNICAL AND NON-TECHNICAL ISSUES

This section discusses how this submission meets the general technical and non-technical requirements listed in Sections 4.3 and 4.4 of the RFP.

3.1 General Technical Requirements

- **Interfaces shall be object-oriented and shall be expressed in OMG IDL**

The Business Object Facility proposed follows a strictly object-oriented approach. All objects, and all specification requirements placed on general objects, are given an OMG IDL interface description.

Encapsulation is a major design feature of the proposed interfaces. The interfaces as proposed do not expose any implementation details, neither in the interface, nor by describing expected performance profiles as part of the interface description. Special care was taken to achieve a strict separation of interfaces and their possible implementations.

All interfaces are defined in OMG IDL following the IDL naming conventions for interfaces, types, operations, attributes, and so on. The specification uses OMG IDL conventions for exceptions and exception parameters.

Semantics beyond those expressible in OMG IDL are expressed in English. Semantics are also expressed formally using the mechanism defined by the Meta Object Facility.

- **Proposed extensions to OMG IDL, CORBA, Object Services, and/or the OMG Object Model shall be identified**

This specification is consistent with OMG IDL, CORBA, CORBAServices, CORBAFacilities, and the OMG Object Model. It assumes that there exist good implementations of existing CORBA specifications.

Specification of complete business semantics are deferred to the Meta Object Facility and the Object Analysis & Design metamodel.

One area of concern with current OMG specifications is with respect to collections. There are now many forms of collections specifications, including those found in the Collection Service, Query Service, Naming, Properties, Relationships, etc. There is a need to establish uniformity, consistency, and generality among these redundant specifications.

- **Operation sequencing shall be included where applicable**

For each operation a behavior description is given. Where applicable, operation sequencing is included to describe the operation behavior. Behavior, protocol, and sequencing of operations for business objects are further formalized in the Meta Object Facility specification.

- **OMG specifications shall not contain implementation descriptions**

This specification does not contain any implementation descriptions. In a few cases where possible implementations are sketched it is only to specify semantics and operation sequencing.

- **OMG specifications shall be complete**

The specification to the best of our knowledge is complete. No “magic” is required on the part of clients.

The creation of objects is defined in the specification, and is defined in terms of the Lifecycle Service.

- **OMG Specifications should have precise descriptions**

This specification describes all service behaviors, and the functions required from other services.

- **Independence and modularity of Specifications**

This specification explicitly requires modularity when implementing business object specifications. However, the enforcement of implementation modularity means there are dependencies between specifications and their implementations. Dependencies include, for example, the requirement imposed on all business objects to inherit the specification and implementation from the BusinessObject specification. This form of dependency is inherent in OMG IDL, the dependency is practiced with CORBAServices, and is required to realize the objectives of the general technical requirements which follow.

Furthermore, there is the issue of context. Business object models for a vertical domain will specify semantics for common business objects within their domain. Another domain will specify additional semantics with respect to its domain. When these contexts are combined, the semantics of both must be realized in the implementation. Although the principle of independently specifying business objects and their semantics is supported through the Meta Object Facility, business object implementations must reflect the aggregate semantic requirement.

In summary, the principle of independent context-dependent specifications are supported via the Meta Object Facility. Business Object implementations, of necessity, are dependent upon multiple context-dependent specifications.

- **Minimize duplication of functionality (the Bauhaus principle)**

This specification is primarily a composition of other CORBAservices and CORBAfacilities. The Business Object Facility minimizes duplication of functionality by enforcing uniform use of existing specifications. Very little new functionality is introduced by the Business Object Facility, the emphasis is on specifying semantic constraints with respect to existing specifications in order to achieve the objectives of the facility.

- **No hidden interfaces among specifications**

A specific objective of the Business Object Facility is that business object interfaces and behaviors be specified sufficiently well so that implementations can be replaced and the parts will still work together. The enforcement of implementation inheritance, modularity, and the rigorous formal specifications of Business Objects help ensure the principle of plug and play components are achieved.

- **Consistency among OMG specifications**

Implementation inheritance, modularity, and rigorous formal specifications enforce consistency between Business Object specifications.

- **Extensibility of individual specifications**

Extensibility mechanisms were examined in section “2.1.5 Extensibility of business objects”, on page 14.

- **Extending the collection of OMG specifications**

Through enforced modularization and implementation inheritance, this specification should enable accomodation of new OMG specifications without impacting existing business object design or implementation.

- **Configurability**

It should be possible to arbitrarily combine the Business Object Facility specification with other OMG specifications.

- **Integrity, Reliability, and Safety of OMG specifications.**

The Business Object Facility ensure the integrity, reliability, and safety of business objects through incorporation of the OMG persistence, transaction, and concurrency services.

- **Performance.**

The Business Object Facility is a composition of OMG specifications. Performance guidelines accompanying each referenced OMG specification has guided the definition of the Business Object Facility, ensuring that the facility itself is consistent with known infrastructure performance trade-offs.

The performance of a business object model may depend upon the granularity of business objects, distribution of business objects, and complexity of interdependency between business objects. The issue of business object performance must be evaluated in conjunction with proposed business objects and models.

- **Scalability.**

This issue was detailed in section "2.1.7 Scalability", on page 16.

- **Portability.**

The Business Object Facility, specified in OMG IDL, accommodates portability of implementations across a wide range of platforms. IDL is programming language independent. Furthermore, the specification may accomodate implementations on other forms of infrastructure under some cases (see section "2.1.14 Generality and desktop integration" on page 25).

- **Consistency with Common Object Services Specifications (COSS)**

The Business Object Facility is a composition of CORBAservices and CORBAfacilities. It is fully consistent, compliant, and non-redundant with defined CORBAservices and CORBAfacilities. It utilizes the Lifecycle Object's Factory Service as needed to create objects.

- **Conventions and guidelines**

Conventions and guidelines defined, used or implied in this specification are consistent with the information contained in Appendix C of the RFP.

- **Mandatory versus optional interfaces**

The Business Object Facility specification includes only mandatory interfaces.

- **Constraints on object behavior**

Business Objects have their semantics formally specified in the Meta Object Facility. Business Object implementations must take into account these extended (and possibly dynamic) semantics.

• **Integration with future OMG specifications**
Through enforced modularization and implementation inheritance, this specification should enable accomodation of new OMG specifications without impacting existing business object design or implementation.

3.2 Technical Criteria

3.2.1 Change and Event Notification

Many forms of change notifications will be available from any business object without explicit programming by the application developer. Business objects will be able to request notification of events or changes in any other business object or class of object and later cancel the request using standard business object facility services. Notification messages will be in a standard form for all business objects. Notification will be provided for many forms of changes that occur in any business object. This service will be available in a consistent fashion across heterogeneous environments.

The OMG Event Service Specification is the basis for change and event notification within the Business Object Facility. The Event Service is a general purpose service which supports a variety of event communication requirements but is characterized by multiple approaches such as *Pull* versus *Push* models and *generic* versus *typed* models. In order to ensure interoperability between business objects, the Event Service must be used consistently by all business objects. The following approach has been selected for Business Objects:

- The push model is used for suppliers.
- The pull or push model is available for consumers.
- Typed event models are used. This also supports the generic interface.
- A BusinessObject is defined with both supplier and consumer interfaces.

Examples of event services which may be defined for each BusinessObjectinclude:

- Parameter Change notification. Every addition, deletion, or value change to parameters could result in an event.
- Role change notification. Every addition or deletion of a role could result in an event.

3.2.2 Active Views

As models are developed and implemented, their size and complexity will grow. There is a need to be able to implement solutions to particular problems using simplified abstractions of the enterprise model. Views, as intended here, are much like traditional database views where a view is a simplified representation that excludes some attributes, relationships and operations, and also flattens a complex structure. In addition, views can provide aliases and signature conversions for specific methods. Consequently, views may be used to insulate individual applications from evolutionary changes in the enterprise model. An active view is linked to the underlying objects such that attributes and relationships represented in the view are kept current with the states of the underlying objects.

The Object Query Service provides the mechanism for implementing the concept of views with respect to arbitrarily complex configurations of business objects. More specifically, the Query object provides the capability for composing, containing, and executing query specifications (views). The execution of the Query will return a result reflecting current attributes, states, and relationships of underlying business objects.

3.2.3 Transparent Persistence

The implementation of persistence, in conjunction with standardized transaction, concurrency, data recovery, and other services, ensures integrity, consistency, and recoverability of configurations of business objects across heterogeneous environments.

All Business Objects are Persistent Objects. Implementation of persistence is not specified for the Business Object Facility.

The OMG Persistence Service is one possible approach to implementation of business object persistence. The Persistence Service helps to isolate data store technology from business objects, enabling the end user to implement data storage technology according to his evaluation of tradeoffs between availability, data integrity, resource consumption, performance and cost. The selection criteria and acquisition of business objects, in these cases, would be independent of data store technologies. There may be alternative mechanisms for business object providers to isolate themselves from data store technologies.

The Business Object Facility recommends, but does not enforce, isolation of data store implementation from business object implementation. Following this recommendation would facilitate the use and replacement of data store technology without reprogramming any business objects.

3.2.4 Search mechanism

The Object Query Service provides the mechanism for implementing flexible predicate-based searches over arbitrarily complex configurations of business objects.

3.2.5 Backout

Backout is the ability to restore an earlier state, e.g., as in termination of transactions. It operates across heterogeneous environments so that related or dependent changes will be backed out in a consistent fashion.

The Business Object Facility uses the OMG Transaction Service and associated mechanisms to implement backout (rollback operation). Each BusinessObject is a TransactionalObject.

3.2.6 Concurrency and Serialization

When concurrent transactions share objects, changes applied by one transaction process may cause the processing of another transaction to be invalid. Serialization assures that the actions of competing transactions are controlled in such a way that the result is equivalent to the transactions being executed one after another instead of concurrently. Serialization is achieved across heterogeneous environments to the extent that transactions traverse multiple environments.

The Business Object Facility indirectly uses the OMG Concurrency Control Service (as a result of using the Transaction Service) to implement concurrency control, including serialization of client requests, detection of deadlocks, suspension and resumption of processes, and transaction contexts. Each BusinessObject is a TransactionalObject.

3.2.7 Nested Transactions

It should be possible to implement business transactions that are invoked by other business transactions and be assured that the serialization, recoverability, backout and commitment of the associated transactions are handled properly. To achieve this, the management of transactions must be consistent and coordinated across heterogeneous environments.

The Business Object Facility uses the OMG Transaction Service to implement transaction requirements, including nested transactions. Each BusinessObject is a TransactionalObject. Recoverable Resources used by the BusinessObject are subtransaction aware.

3.2.8 Referential Integrity and Garbage Collection

Referential integrity within the Business Object Facility is maintained through a combination of Relationship, Persistence, Transaction, and Concurrency Services. The Business Object Facility enforces referential integrity across all business objects, there are never any “dangling” references to deleted business objects, and there is no need to explicitly perform garbage collection. The referential integrity mechanism does not require any explicit action by the programmer or user, it is implemented automatically when a business object is removed, according to the defined semantics of business objects and their relationships.

3.2.9 Encapsulated attributes and relationships

A standard protocol is used to provide consistent interfaces, across all business objects, to attributes and relationships. Attributes are specified and implemented according to the CORBA IDL specification for “attribute” and the defined language mappings.

Relationships are specified and implemented according to the syntax specified for the Relationship service, with semantics specialized according to the specifications detailed in this document. The Business Object Facility defines the mechanism for persistence, transaction integrity, and concurrency to maintain the integrity of attributes and relationships for all business objects.

3.2.10 Constraints, rules and policies

The Meta Object Facility supports user specification of constraints, rules, and policies on the business object model. Once these specifications have been verified for internal consistency they are transformed into initialization policies, validation policies, roles, and relationships. Initialization policies influence the creation of new business objects. Validation policies dynamically control behavior, state, and relationships of all business objects.

Many of the constraint/rule/policy specifications are enforced by the BusinessObject itself, in conjunction with defined role and relationship semantics. For example, BusinessObject enforces referential integrity constraints, relationship constraints/invariants, and various forms of lifecycle dependency propagation. Each Business Object uses persistence, transaction, and concurrency services to ensure system integrity, that the system never commits an invalid state.

3.2.11 Relationship Management

Relationships between objects are complementary, each has access to the identity of the other. They are consistently maintained without explicit programming by the application developer.

The Business Object Facility manages relationships using the OMG Relationship Service. Each Business Object inherits the Node Interface. Business Objects, in conjunction with the Business Object Facility, enforce relationship semantics defined in the Meta Object Facility. Features of the capability include:

- Explicit business object model (Meta Object Facility) representation of relationships and roles.
- Type and cardinality constraints are explicitly expressed and validated.
- The capability is extensible.

A BusinessObject is a Node (defined in Relationship Service). When a BusinessObject is created, each explicitly defined Role of the Node is created. When Business Object creation is complete, each Node is checked to ensure that its minimum cardinality constraint satisfied (an exception is raised if cardinality constraint violated during creation). In addition to the Node itself, external objects may alter, add, or remove relationships. The Relationship service will automatically maintain the following relationship integrity rules:

- Following creation of a BusinessObject, minimum cardinality constraint of all of the BusinessObject's Roles will be ensured. An exception will be raised if attempt is made to reduce cardinality below minimum.
- Maximum cardinality constraint of all BusinessObject's Roles will be ensured. An exception will be raised if attempt is made to increase cardinality above maximum.
- Cascade delete. Request to delete a business object will result in propagation of delete to each BusinessObject which would otherwise have minimum cardinality constraint violated.
- Referential inegrity. Each BusinessObject which refers to another related BusinessObject, via a relationship, will always have the related BusinessObject also referring to it.

3.2.12 External name management

External names, as used here, are identifiers used in the real world. A single business concept may have several names which apply in different contexts. The Business Object Facility defines a mechanism for accessing objects in a distributed environment using names. The mechanism can be used to model the contexts and classifications used in the real world.

The following mechanism may be used to implement the requirements for external name management based on the *Root* interface:

- A set of roles may be associated with *Root*, each role representing a "context" or classification. Each of these role definitions must be ultimately derived from one of the Role interfaces.
- The specification for *Root* is completed by specifying the remaining semantics of the relationship, including relationship type and the reletant Business Object *Node* type.
- A Business Object *Node* may be associated with multiple naming contexts (roles).
- Any Business Object *Node* may be related to the *Root* via a role. Implementation of the capability is transparent to the Business Object *Node*.
- As with any relationship supported by the Business Object Facility, full referential integrity is ensured, relationship semantics are flexible, etc.

3.2.13 Exception/Fault Resolution

The Business Object Facility specifies the coupling between Business Application Components (and consequently contained Business Objects) and the System Management Facility. Recommendations from the System Management Facility with respect to exception specification have been adopted. Based on these guidelines, exceptions generated by servers, at the point of error detection, will include specific context and error information for eventual presentation to the user. As the exception is returned, servers that pass the exception through to clients may add additional information related to the context or nature of the error. This approach allows an administrator to receive more information about an exception than simply "something undesirable happened".

In addition to reporting the exception to the user, a mechanism is needed to support analysis of the state of processes and to determine at what point they might be restarted. Standard mechanisms are required to report, analyze, reconcile states and restart processes when failures occur. These requirements are most appropriately addressed by the Systems Management Facility. The Business Object Facility, through its coupling with the Systems Management Facility, should be able to incorporate relevant specifications addressing these requirements, when they become available.

3.2.14 Configuration Management

In a large, distributed, heterogeneous environment, it will be necessary to upgrade business objects (and other components) without interrupting the operation of the networked system. This will include changes to the implementations of active business objects and relationships which must be coordinated across heterogeneous environments. A business object should be configurable after it has been delivered and deployed and should be able to be subclassed and used by other business objects in unanticipated ways without recourse to development. Business Objects should cooperate with other objects in ways that do not bind them to a specific location or implementation.

As discussed in section “2.1.8 Ease of development and deployment” on page 16, and elsewhere, many of the semantics of the Business Object Model, including relationships and configuration/tailoring options, can be reflected dynamically in deployed business objects. This enables business objects to be upgraded within large, distributed, heterogeneous environments without interrupting the operation of the networked system.

Consistent application of implementation inheritance semantics ensures that any Business Object can be subclassed and used by other Business Objects. Furthermore, new implementations of Business Object supertypes will be immediately inherited by all subtypes. There will be no requirement to redevelop, rebind, or reimplement subordinate Business Objects when a parent Business Object changes. Adherance to CORBA standards ensure that Business Objects are location and implementation transparent to their clients. Implementation inheritance extends location and implementation transparency to subclasses.

Implementation of managed change coordination to Business Meta Objects, Business Objects, implementations, and Business Object instances are issues which are being addressed by the Interface Type Versioning Management specification. Initial submissions of these specifications are due July, 1996. The Interface Type Versioning Management specification will be reflected in this Business Object Facility specification, as required.

3.2.15 Composite Object Bounds

A composite object is a structure that represents a complex application concept with a principal object and associated application component objects. Typically, if the principal object is deleted, the application component objects will no longer be meaningful.

Sometimes, however, a composite object may "contain" other composite objects as components. Generally, the components of a composite object will be saved or moved with the principle object and the principle object may have versions which incorporate different component objects. The problem is to define the bounds of composite objects and facilitate versioning, storage and transport of the composite as a unit consistently across heterogeneous environments.

The Business Object Facility utilizes the OMG Relationship Service to facilitate implementation of arbitrarily complex composite structures. The CosContainment Module and the CosReference Module form the basis for the interface and semantics of common composite business objects. Based on the Graph traversal semantics of the Relationship Service, composite business objects can be manipulated, saved, moved, externalized, versioned, etc., as a unit.

3.2.16 External Resource Representation

An objective of the Business Object Facility is to ensure isolation of Business Objects from any form of presentation/device/external resource technology. Therefore, the Business Object Facility intentionally omits any reference to external resources and recommends that any future Business Object specifications avoid explicit references to external resources.

External resources will, of course, be necessary to implement some business solutions. The recommended architecture for these solutions involves the implementation of a “mediator” between the business object and the external resource. Interaction with the business object will include direct use of business object operations as well as use of the event monitoring capability. The specification of such a mediator is beyond the scope of this specification.

3.2.17 User Attributes and Preferences

Every Business Object supports user-defined properties. Users can add properties to specific business objects or to the corresponding Meta Object (which applies to all Business Objects of that type). These properties can be referenced in queries, policy constraints, or user-specified semantics (Business Meta Objects) which control behavior of the Business Object.

3.2.18 Textual Representation

Business Objects are translatable to and from various external forms, including textual form, for storage, transport and editing. The external forms represent individual business objects, arbitrary sets of multiple business objects, and graphs (as defined by the Relationship Service) of business objects. Graphs comprehend arbitrarily complex structures and traversal criteria as well as handling detection and representation of cyclic structures. The external forms include representations which are language and environment independent.

The Business Object Facility uses the OMG Externalization Service to render objects into external form. Each BusinessObject inherits the Node Interface defined within the CosCompoundExternalization Module.

3.2.19 Executable Object Expressions

There is a need to be able to express operations on objects in the form of objects. Applications need to be able to create and sometimes analyze executable expressions and pass them from one computing environment to another. These may be used for rules or constraints, or they may be used in executable blocks for exception handling, iteration, or event processing. The language used to specify or display these expressions should be independent of the particular programming environment.

This specification does not address this requirement, it is outside the scope of Business Object Facility. The requirement should be addressed by some other facility. There are currently no facilities on the OMG roadmap which address this requirement.

3.2.20 Loose binding

The Business Object Facility explicitly requires implementation inheritance and modularization (plug and play components), which in turn will require dynamic binding between business objects.

Business Objects are fully CORBA compliant, meaning that they are location independent, implementation independent, and programming language independent.

3.2.21 Instance Specialization

Instance specialization is the capability to add methods and state to an object instance to incorporate unique capabilities. The extensions may be temporary for solution of a unique problem, or they may be persistent for a continuing requirement. They may be used to support analysis or reasoning about the object, or to evolve the representation of a concept as it is discovered in the real world.

All Business Objects are based on a composition of CORBAservices. This enables any Business Object *instance* to be dynamically specialized in the following ways:

- New Properties may be added.
- New Relationships may be established.
- Events can be linked to new event channels, new events can be monitored.

The Business Object Facility does not provide a specification for how to dynamically add methods to instances, the concept is fundamentally in conflict with the nature and intent of OMG IDL.

3.2.22 Reflection

Reflection is the ability of a system to analyze and report its state and activities and to alter its state and activities based on this analysis. This requires the ability to examine meta-information, call-paths and executable expressions. The ability should be ad hoc so that active processes can be compiled for performance but switched to interpretation for reflection. Such capabilities are important for such activities as exception resolution, automatic code generation, interactive query, machine learning, performance tuning and adaptation to particular users. Reflection should also support tools for analysis of system performance, debugging, design of tests and failure mode analysis.

"Reflection" is beyond the scope of this specification. However, the Business Object Facility, and its architectural relationship to other CORBA services and facilities, facilitate implementations of the reflection concept in the following ways:

- There is a tight coupling between Business Objects and Business Meta Objects. Consequently, it is possible to navigate from the Business Object directly into the Business Object Model. All semantics can be explored, all contractual obligations between Business Objects can be analysed, interactive query can utilize defined relationships between Business Objects, exception sources can be found.
- The coupling with System Management can provide additional information related to performance analysis and tuning, failure analysis, recovery, and policy.
- Automated code generation, test design, simulation, interpretative mode execution, etc., are capabilities which can be based on the Business Object Model.

3.2.23 External interfaces

External systems, such as user interfaces or desktop programs have a known and consistent interface to any business object. This interface is defined in IDL and is available via the Interface Repository.

All Business Object constraints, dependencies, rules, and other semantics are fully specified in the associated Business Meta Object. The mechanism for constructing, implementing, or changing a Business Object is beyond the scope of this specification and would be in conflict with the fundamental objective to isolate implementation from specification. However, the Business Object Facility specifies that any Business Object must enforce the semantics embodied in the Business Meta Object, including any dynamically specifiable semantics.

3.3 Relationship with other Common Facilities, OMG Object Services, CORBA and OMG Object Model

The **Business Object Facility** specification does not impose any particular or unique implementation restrictions. There are no restrictions on the use of internal, hidden interfaces or protocols with the underlying CORBA infrastructure, as long as the external interface definition and the associated semantics are compliant with the specification.

The following sections explain how the Business Object Facility, Business Application Components, and Business Objects relate to the elements of the OMG architecture.

3.3.1 CORBA

The Business Object Facility, Business Objects, and Business Application Components are defined in terms of the syntax and semantic scope of OMG IDL. The Meta Object Facility enables formal specification of business semantics - this may be viewed as a formalization of the english language semantic specification accompanying OMG IDL.

3.3.2 CORBAServices

Business Objects and the Business Object Facility are derived from, and/or use, the following object services:

- Persistent object service
- Transaction service
- Relationship service
- Lifecycle service
- Query service
- Event service
- Collections service

The Business Object Facility specifies the combination of these services required to meet its objectives. In some cases, there are specializations of operation semantics. These specializations are detailed in section “2.4
Interface Description: Behavior”, starting page 43.

3.3.3 CORBAFacilities

Business Objects and the Business Object Facility directly utilize the following Common Facilities:

- Meta Object Facility. Used to store and maintain the business object model, including the full semantics for each Business Object type.
- Systems management. Used to manage Business Application Components

The Business Object Facility specifies the combination of these facilities required to meet its objectives in section “2.4
Interface Description: Behavior”, starting page 43.

3.4 End User Requirements

The Business Object Facility specification addresses administration of objects in the following ways:

3.4.1 LifeCycle

The Business Object Facility mandates the LifeCycleObject interface, plus a Factory interface, for all Business Objects.

3.4.2 Installation/De-installation

Automated remote installation and deinstallation of features, including Business Application Components, are intended to be provided through the System Management Facility. The Business Object Facility is coupled with the System Management Facility. The System Management Facility has not yet fully described the interfaces necessary to implement remote installation and desinstallation.

3.4.3 Upgrade

Automated remote upgrade of the Business Object Facility, including Business Application Components, is intended to be provided through the System Management Facility. The Business Object Facility is coupled with the System Management Facility. The System Management Facility has not yet fully described the interfaces necessary to implement remote upgrade.

3.4.4 Performance Management

In order to support the management of large, heterogeneous OMA- compliant environments, standard interfaces shall be defined which provide for the querying and tuning of performance-critical resources.

Querying and tuning of performance-critical resources is intended to be provided through the System Management Facility. The Business Object Facility is coupled with the System Management Facility. The System Management Facility has not yet fully described the interfaces necessary to tune performance-critical resources. Additionally, the Business Object Facility architecture facilitates certain aspects of performance tuning by isolating technology and modularizing Business Application Components. Thus:

- Data store technology can (sometimes, depending on implementation) be replaced or tuned, transparent to Business Application Components.
- Visualization and external interface tchnology can be replaced or tuned, transparent to Business Application Components.
- Any Business Application Component can be replaced or tuned, transparent to other Business Application Components. Performance improvements associated with a Business Application Component will be inherited transparently by all dependent Business Objects.

3.4.5 Testing and Problem Determination / Resolution

All interface specifications for the Business Object Facility and Business Objects explicitly include unique exception codes for each possible error condition.

History of Business Object Facility object invocations is intended to be provided through the System Management Facility. The Business Object Facility is coupled with the System Management Facility. The System Management Facility has not yet fully described the interfaces necessary to implement object invocation history.

Each Business Object is coupled with its Business Meta Object. The Business Meta Object fully specifies the semantics of the Business Object, including a description of its states. These state semantics are accessible through Business Meta Object interfaces. The semantics also include formal and rigorous specifications of all pre-conditions, post-conditions, invariants, etc. These semantics can form the basis for a portion of testing. In a broader sense, testing is part of the Business Object development lifecycle, and consequently should be formally specified using the Analysis and Design Facility.

Problem determination/resolution is uniformally handled across all Business Objects, as described in section “3.2.13 Exception/Fault Resolution” on page 73.

4. SPECIFICATION DEPENDENCIES

This specification take advantage of CORBAFacilities and CORBAServices that are available, including:

- Transaction service
- Relationship service
- Lifecycle service
- Query service
- Event service
- Collections service
- System Management Facility
- Meta Object Facility

The Business Object Facility specifies combinations of these services to meet its objectives, as detailed in section “2.4
Interface Description: Behavior”, starting page 43.

Some of the above specifications have not yet been adopted. Inclusion of the “Meta Object Facility”, and other specifications, is speculative since RFPs have not even been issued yet. It is presumed that all OMG specifications have available technical implementations, including the specifications which have not yet been adopted. The following sections describe assumptions concerning specifications. These assumptions are with respect to anticipated future specifications.

4.1 Meta Object Facility

module CfMetaObject{

```
        interface Meta: {};
};
```

4.2 Object Analysis & Design Facility

4.3 System Management Facility

4.4 Notification and Messaging

4.5 Change Management

4.6 Replication

4.7 Logging

5. RELATIONSHIP TO CORBA

The specification is fully CORBA compliant and does not propose any extension of CORBA, including OMG IDL.

6. RELATIONSHIP TO OMG OBJECT MODEL

This specification conforms to the OMG Object Model.

7. STANDARDS CONFORMANCE

This specification is composed from existing OMG specifications, which in turn are based on relevant existing industry standards, including the architecture for system distribution defined in ISO/IEC 10746, Reference Model of Open Distributed Processing (ODP).

8. OTHER INFORMATION

8.1 References

[OMG OMA]	*Object Management Architecture Guide*, Revision 3.0.
[OMG CORBA]	*The Common Object Request Broker: Architecture and Specification*, Revision 2.0, July 1995.

[OMG 95-01-02] *Common Facilities Architecture*, January 1995. OMG TC Document 95-1-2

[OMG 95-01-47] *Object Services Architecture*, Revision 1.1, January 1995. OMG TC Document 95-1-47

[OMG 95-12-05] *Systems Management*, Revision 2.0, December 6, 1995. OMG TC Documents 95-12-02 through 95-12-06.

[OMG 92-08-05] *Object Services Roadmap*, October, 1992. OMG TC Document 92-8-5

[OMG 95-03-31] *CORBAservices: Common Object Services Specification*, March 31,1995. OMG TC Document 95-03-31

[OMG 95-04-01] *BOMSIG white paper*, OMG TC Document 95-4-1

Business Objects, Oliver Sims	McGraw-Hill	0-07-707957-4
Object Advantage, Ivar Jacobson	Addison-Wesley	0-201-42289-1
The Object-Oriented Enterprise	Rob Mattison	McGraw Hill
Business Engineering with object technology, David Taylor		Wiley, 1995

[SEMATECH CIM] *Computer Integrated Manufacturing (CIM) Application Framework Specification 1.2,* March 31,1995. SEMATECH. Technology Transfer 93061697E-ENG