Network Working Group
Internet Draft
Expires April, 2000

L. Bartz
Internal Revenue Service
October, 1999

Composite Objects for the Directory

< draft-bartz-directory-composite-objects-00.txt >

Status of this Memo

Abstract

Composite Objects for the Directory (COD) describes an abstraction layer above the schema, content, and structure facilities of the Directory. COD enables the construction of reusable information components from schema primitives. These components include composite objects, type-signed object references, graphs of object hierarchies, and object-oriented relationships among Directory objects.

COD also provides a repository of templates, or prototypes of the information components. This repository captures component design information which cannot be expressed in the Directory schema. This repository allows the design of the components to be unequivocably described and reused both within and beyond the immediate Directory instance.

Table of Contents

1. Introduction

Composite Objects for the Directory (COD) provides an abstraction layer above the schema, content, and structure facilities of X.500 [1] and LDAP [2], [3] (hereafter collectively called "the Directory"). COD enables the construction of reusable information components from schema primitives. These components include composite objects, type-signed object references, graphs of object hierarchies, and object-oriented (OO) relationships among Directory objects.

COD also provides a repository of templates, or prototypes of the information components. This repository captures component design information which cannot be expressed in the Directory schema. This repository allows the design of the components to be unequivocably described and reused both within and beyond the immediate Directory instance.

Implementation and use of this framework does not require any change to, or extension of the canonical protocols of the Directory. COD uses the Directory just as it is.

COD is not intended to subvert, undermine, or minimalize the role or the use of the Directory's formal schema, DIT content rule, or DIT structure rule mechanisms.

Object-oriented design environments, such as UML [5], and OO implementation environments, such as Java [6], CORBA [7], and XML [8] capitalize upon composite objects as higher order expressions of type. Composite objects are constructed from primitive objects and attributes and other composite objects. Composite objects enable emergent qualities, semantics or behaviors which are not obvious in the individual parts. Composite objects can also serve as vehicles of design reuse, enhancing productivity and facilitating adaptability.

The Directory has no adequately expressive mechanisms in its schema to describe and share composite objects. The Directory thus cannot share in the benefits which are accorded to information models (such as UML, Java, CORBA, XML) which make extensive use of object composition.

This shortcoming is particularly painful in the area of object relationships, or associations. In current Directory practice, the mechanisms for constructing relationships among objects are limited and inhibit design reuse. These all-important relationships, if they have been described in the Directory, have been described in brittle, simplistic, and implementation-specific constructs.

COD facilitiates the representation of complex relationships among

objects in the Directory. COD's directory object relationship model is a reusable implementation of widely accepted object-oriented design.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, reference [4].

2. Composite Objects for the Directory

2.1. Composite Objects

2.1.1. Introduction

Composite objects are the first building blocks in the COD information component framework. A composite object is a structured collection of Directory primitives, such as attributes, objectclasses, and object references. The composite has more meaning than the sum of its individual parts. The COD framework provides for treating a composite as a type.

The Directory should able to satisfy the information requirements of clients which employ object-oriented analysis, design, programming, and implementation strategies and techniques.

These environments require an adaptable information repository in which complex objects can be composed from primitive types, and in which the definition of those information components can be shared and reused as objects, with the full privileges and responsibilities accorded by that first-class status.

COD's framework facilitates the design, construction, and implementation of information components which are more sophisticated and more expressive than the parts of which they are composed.

2.1.2. Composite Object information model

Instead of encoding Directory primitives to match every potential composite object, COD provides a framework from which composites can be constructed, allowing the Directory to adapt to the information description and structure requirements of its clients.

COD's information model is a meta-metaInformation infrastructure, in which the meta information is described in the information terms of some more primitive, more elemental underlying model. See "2.2 Four-Layer Metamodel Architecture" of [9] [UML Semantics]. In COD, the meta information is described in terms of the Directory's basic constructs, which include attributes, objectclasses, and initialized values of attributes. Elaborations, extensions, and instances build upon COD's meta framework.

The base design for these classes and attributes is reused from RFC 2293 [10], with modifications to support the object-oriented design patterns "Composite" [11.b] [composite] of [11] [GHJV95], and "Whole-Part" [12.b] [whole-part] of [12] [POSA]. The objectclasses and

attributes of RFC 2293 are not extended or reused in COD.

2.1.2.1. The Carbon Fiber Directory

Carbon fiber (CF) composite materials are stronger than steel, lighter than aluminum, and are structurally sound in shapes which neither steel nor aluminum can emulate. CF is composed of limp carbon fibers and brittle epoxy resin, neither of which could possibly serve as a structural material on its own.

At the heart of COD's composite model is the same "name=value" construct which is described in RFC 2293. A composite element has a name and it contains or refers to a value. The table/tableEntry design of RFC 2293 is extended to enable:

- composable hierarchy - per [composite]

  Allows creation of tree-like structures of objects which represent whole-part hierarchies

- homogeneous containers - per "collection-members" [POSA] variant of [whole-part]

  Composite objects used as containers can be constrained (within the bounds of this model) to contain a single type of object. The quantity or multiplicity of objects contained can be constrained.

- heterogeneous containers - per "container-contents" [POSA] variant of [whole-part]

  Composite objects used as containers can be constrained (within the bounds of this model) to contain several specific types of objects.

- shared parts - per the "shared parts" [POSA] variant of [whole-part]

  Objects can be "contained" by reference. The referenced object's lifecycle is not dependent upon its relationship to the container.

- fixed structural assemblies - per the "assembly-parts" [POSA] variant of [whole-part]

  Allows design, construction, and use of information components in which all variable parameters (type, multiplicity, order of traversal or evaluation, relationship) are strictly defined and

controlled.

- O sharable attributes

  Attributes encapsulated within information components assume status as objects, for sharability, whereas in the Directory model, only objectclasses can be referenced and shared. This quality is a reuse of RFC 2293 design.

- O type-signed object references

  The elevation of attributes to the status of objects allows for the assignment of supporting attributes which describe their qualities and constraints. In the case of object references, the type of the referenced target is asserted.

2.1.2.2. The Raw Directory

RFC 2293 asserts a few specific types of "tableEntry" objectclasses and suggests that many more are possible.

COD asserts many specific types of "objectElement" objectclasses, which are analogous to RFC 2293's tableEntry.

The attribute-bearing objectElement objectclasses are "raw", in the sense that they do not assert that they represent any thing in particular. Their principal assertion, their Directory-type assertion, is the Directory attribute syntax of the value they contain in the "name=value" construct.

The objectElement objectclasses are "raw" because until they are "cooked" by the information which uses them, they are in an undigestible raw state.

An information realm cooks a raw objectElement by assigning values to the "infoType" attribute. Values assigned to infoType are meaningful in the realm of the information which uses the COD information infrastructure.

By canonicalizing these raw objectclasses and the syntax-specific values they contain, we can allow information to shape itself in the Directory. A Directory implementation can thus become more nimble, more agile; flexible and adaptable.

Where have we heard this before? Which other systems or technologies allow information to describe itself and dynamically shape the media which it inhabits?

XML is a good example. In XML, the information describes itself. Although some information realms will canonicalize structure and content for some types of XML information, this is not a requirement of XML itself. XML canonicalizes just enough rules to provide for consistent syntax and structure.

For another example, consider the Massachusetts Institute of Technology's Raw Computing Architecture [28]. MIT's Raw CPU is a very capable blank computing slate. Software shapes the runtime behavior of the chip, so it can behave as a radio, a cellphone, a computing device, and more.

The Raw computer is shaped by the software.

The Raw Directory is shaped by the information which uses it.

COD asserts that the Directory should be as agile, flexible, and adaptable as XML and Raw. COD provides facilities to make it so.

Appendices A.2 and A.3 of this document describe the objectclasses and attributes of the Composite Object information model of COD.

Appendices A.6 and A.7 describe the "raw" attribute-bearing objectclasses and attributes.

2.2. Object Associations

2.2.1. Introduction - The Learning Directory

COD introduces a reusable design for modeling relationships among Directory objects. By extension, it is a reusable design for modeling relationships among the "real" objects the Directory represents. The object association information model is an elaboration of the composite object framework.

"Information" which is simply a collection of facts is barely useful. Facts do not constitute knowledge. Connections among facts constitute knowledge. Creating connections among facts constitutes learning.

COD's reusable directory object relationship model offers this potential for the Directory in a dynamic information environment: learning, connectionism, relationships, knowledge, power.

With COD, we can create a Directory implementation which stores knowledge, a Directory which can facilitate connectionism (learning) on a massive scale.

We can't achieve this scale if we have to engineer each connection type one-by-one. This is the canonical, common practice strategy. It doesn't scale. It inhibits reuse. It impedes progress. That's why the massive connectionism we seek hasn't been done yet. In common practice, every objectclass which is designed to represent an object relationship is a separate, unique construct. While these are effective in their own limited contexts, neither their design nor their implementation can be reused outside their native information realm.

See Appendix B for a more complete description of how the limitations of Directory common practice provide the context and the motivation for COD's object association model.

This model provides a meta-tool for learning (creating connections among facts) and knowledge (mapping relationships among objects) in the Directory. By capitalizing upon the benefits of reuse, both in design and implementation, COD can extend the power of the Directory.

COD's object relationship model, which is a special type of composite object, can become a focus for learning (connectionism), and a locus of knowledge (relationships), in the Directory.

2.2.2. OO Design Reuse

COD's general model for describing relationships among Directory objects is a Directory-specific embodiment of existing object-oriented relationship designs. This Directory object relationship model is based largely upon UML 1.1 [9] [UML Semantics] object associations. The model also draws from the CORBA Relationship Service [13] [CORBA Relationships], and "Associative Objects" of Shlaer-Mellor object-oriented design methodology [14] [Shlaer-Mellor].

COD's OO object relationship/association design should be a boon to OO developers. By reusing object association design from UML and object relationship design from CORBA, COD facilitiates design and development of Directory software by OO developers. These constructs are familiar and natural for object-oriented analysis, design, development, and implementation.

2.2.3. Composite Type Consistency

2.2.3.1. Design from the bottom, up

Elemental members of the composite object assert their Directory types naturally, in the canonical type mechanisms of the Directory.

COD compositeElements are also capable of asserting another "type" value, their infoType. As described in 2.1.2.2., values assigned to infoType are meaningful in the realm of the information which uses the COD information infrastructure.

2.2.3.2. Implement from the top, down

Superiors in a composite object hierarchy assure the types of their subordinates and their referents by assigning values to their own "targetType" and "targetInfoType" attributes. Examples in 2.2.6 and Appendix F illustrate this principle of the framework.

2.2.4. Object Relationship Schematic

2.2.4.1. Description

The following ASCII representation, Figure 1, of a UML schematic [15] [UML Notation] illustrates the static object relationships.

The model's objectAssociation object is analogous to UML 1.1's "association" object, CORBA Relationship Service's "relationship" object, and Shaler-Mellor's "associative" object. The objectAssociation object is a representation of the relationship itself.

An objectAssociation is a typedObjectContainerCompositeObject. As a

container, it can be constrained by COD to contain objects of a certain type.

As a relationship is composed of the roles which the actors in the relationship play, an objectAssociation is composed of one or more objectAssociationEnd objects, abbreviated "associationEnd" in Figure 1.

An objectAssociation contains objectAssociationEnds.

An objectAssociationEnd plays a role in the relationship. It is the number and type of relationship roles which give the association a unique semantic, makes it a unique type.

The model's objectAssociationEnd is analogous to UML 1.1's "associationEnd" and CORBA Relationship Service's "relationship role".

An objectAssociationEnd is a typedObjectContainerCompositeObject. As a container, it can be constrained by COD to contain objects of a certain type.

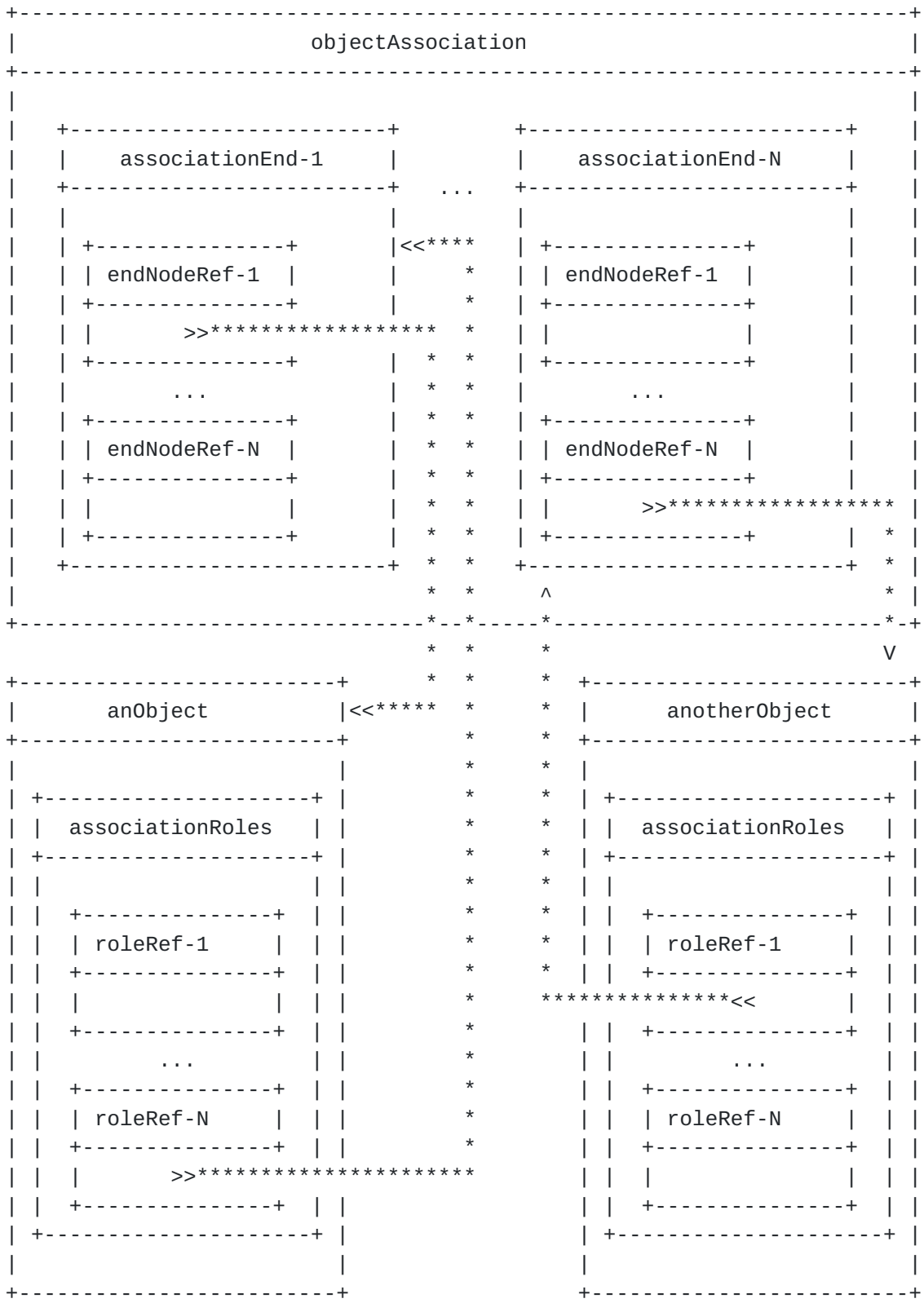
The subordinates of an objectAssociationEnd are objects of type associationEndNodeRef (abbrev "endNodeRef"), a subtype of typedObjRefCompositeElement. The associationEndNodeRef asserts the type of the object to which it must refer.

The "anObject" object in this schematic is a collaborator in the relationship/association. "anObject" contains a composite object "objectAssociationRole" (abbreviated "associationRoles" in Figure 1), which is a container of references to objectAssociationEnd objects. These references are the mechanism by which anObject can determine the roles it plays in associations in which it is a participant.

The subordinates of an objectAssociationRole are objects of type associationRoleRef (abbrev "roleRef"), a subtype of typedObjRefCompositeElement. The associationRoleRef asserts the type of the object to which it must refer, which must be an associationEnd.

The "anotherObject" object in this schematic plays one of the other roles in the relationship.

2.2.4.2. Figure 1
Object Association Schematic



2.2.5. Principal Features and Capabilities

COD's object relationship model provides for:

2.2.5.1. Type Specification

COD's framework introduces a mechanism (via schema) and a semantic (via information model) for effecting and assuring type-signed relationships among objects in the Directory.

The types of all participants, or collaborators, in an object relationship are explicitly specified. The types can be discovered and assured by applications which follow this information model.

Relationship role aggregations are defined to assure that all of their members are of the same type. For example, to create a many-to-many-to-many (M-to-M-to-M) relationship in which the three collaborators are aggregations of apples, oranges, and pears, we can be assure that there will not be a banana hiding in one of the baskets.

2.2.5.2. Arbitrary Complexity

The COD Directory object relationship model is configurable, via the states of its objects (values of their attributes and subordinates), to support relationships of binary, ternary, quaternary orders and beyond.

Each of the collaborators in these relationships can consist of one or more objects, configurable by the states of the objects, without need for extension of the objectclasses described here. The multiplicity of the relationship collaborators can be controlled or ignored.

Given these adjustable parameters, arbitrarily complex graphs of related Directory objects are possible.

As a result, the classic 1-to-1, 1-to-many, many-to-1, and many-to-many relationships are all achievable from this set of general purpose objectclasses.

Moreover, COD can represent virtually unlimited extensions and permutations of classic relationship models. Consider 1-to-M-to-M, M-to-M-to-M-to-1, or 2-to-7-to-9, or more, or less, or whatever.

2.2.5.3. Navigability

Navigability of the entire relationship graph, beginning from any node, is assured.

Graphs of related objects can be navigated from edge to edge. All participants or collaborators in an object relationship can be identified, visited, and examined.

2.2.5.4. Reusability

By using a single, flexible, expressive construct to represent complex Directory object relationships, we can avoid the wasteful effort of creating a new, implementation-specific construct for every new or different object relationship circumstance.

By reusing a model of complex object relationships in the Directory, we also provide an opportunity to reuse the design, construction, behavior, and usage of the Directory client software which must create, retrieve, interpret, and navigate the object relationships.

By reusing the UML 1.1 "association" model for object relationships, we capitalize upon a proven design which is in wide use.

UML's object associations can describe all relationships among objects. The association contains roles. The roles contain references to the actors. The actors keep references to their roles. Tweaks and elaborations define, determine, and assure how many roles are in a relationship, how many actors may act in a role, the required types of the actors, and in what order, if any, the roles and actors are to be evaluated.

Appendices A.4 and A.5 describe the objectclasses and attributes of the Object Association information model of COD.

2.2.6. The play's the thing

Consider the following analogy, which illustrates COD's information model for Directory object relationships.

A play, such as a stage production or a movie, is an object. It is a distinct and identifiable thing. It has qualities and attributes of its own, such as a name, an author, and a script.

Our objectAssociation objectclass is analogous to the play.

The play also represents a relationship. The roles (played by actors, who we'll discuss shortly) are each separate and distinct objects. Each role possesses qualities and attributes of its own, such as name and the part of the script which pertains to the role (role-specific behavior, such as lines and stage directions). The role is contained within the play. It has no meaning or existence outside the context of the play.

Our objectAssociationEnd objectclass is analogous to the role.

A play contains roles. An objectAssociation contains objectAssociationEnds.

A role in a play can be performed by more than one actor. This is often the case in large productions and long-running performances.

The actors exist outside the context of the role they act in the play. The actors of the role each have their own qualities and attributes, such as name, physical characteristics, and talents.

An actor is not a role. A role is not an actor. They are separate objects.

An instance, or production, of the play possesses a list or a manifest of the actors who play each role. This list is likely created and maintained by the director of the play. At the time of the play's performance, the list is published in the play's program. The program includes a name=values(s) listing of role=actor(s)Name(s). Note carefully: the list does not contain actors. It contains names of actors, which are references.

Our objectAssociationEnd objectclass is a container of references to the objects which act in the role.

Actors keep a resume of the roles they play. An actor who is capable, trained, and selected to play the role of King Lear in Shakespeare's play will certainly mention this in his resume. The actor does not

keep the role itself, but a reference to the role, in his resume.

Our objectAssociationRole objectclass is analogous to the actor's resume. It is a container of references to roles the actor plays.

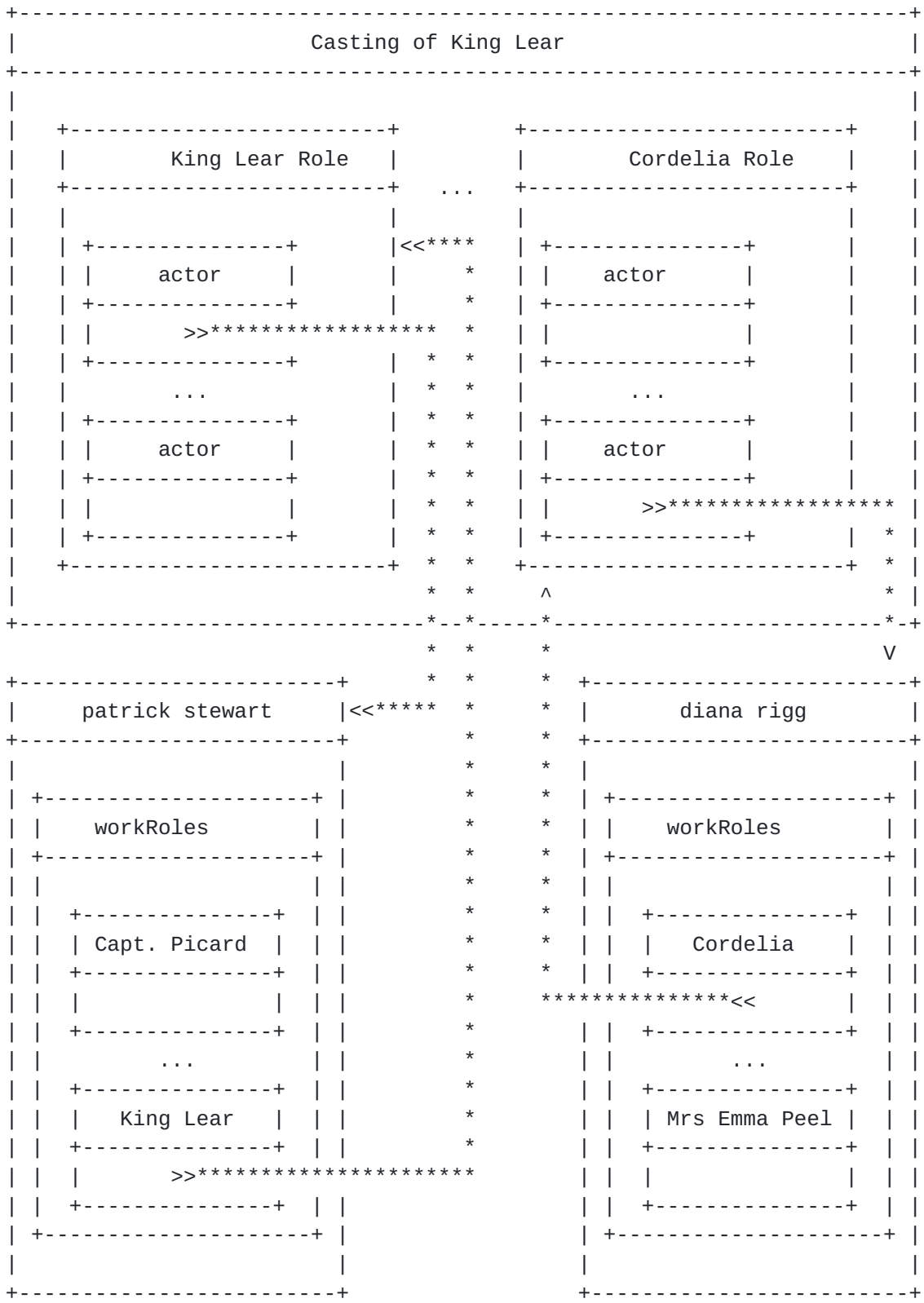
An stage actor keeps a resume. An object actor (in a role) keeps an objectAssociationRole. The objectAssociationRole is a container of references to the roles the object plays in relationships.

This analogy serves to describe all relationships among objects, just as UML's object associations can describe all relationships among objects. The association contains roles. The roles contain references to the actors. The actors keep references to their roles.

An illustration, Figure 2 (2.2.6.1), of the object relationships follows this analogy.

An LDIF [32] representation of Figure 2 follows the figure.

2.2.6.1. Figure 2
Larry's Casting of King Lear



2.2.6.2. LDIF representation of Figure 2

The example is completely described and implemented in terms of the objectclasses and attributes contained in COD. There is no requirement to extend COD for objectclasses such as "casting", "roles", or "actors".

2.2.6.2.1. the relationship

The "Casting of King Lear" object is an instance of objectAssociation.

```
version: 1
dn: compositeElementName=Casting of King Lear, ou=Cast Plays,
 dc=LarryCasting, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
objectclass: typedObjectContainerCompositeObject
objectclass: objectAssociation
compositeElementName: Casting of King Lear
description: This is our dream cast for King Lear
infoType: casting
targetType: objectAssociationEnd
targetInfoType: workRole
# how many roles in King Lear? A guess:
degree: 27
```

2.2.6.2.2. the roles

The "King Lear Role" and "Cordelia Role" objects are instances of objectAssociationEnd.

```
dn: compositeElementName=King Lear Role,
 compositeElementName=Casting of King Lear, ou=Cast Plays,
 dc=LarryCasting, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
objectclass: typedObjectContainerCompositeObject
objectclass: objectAssociationEnd
compositeElementName: King Lear Role
description: references to actors who play Lear are here
infoType: workRole
targetType: associationEndNodeRef
targetInfoType: actorReference
minInstanceMultiplicity: 1
maxInstanceMultiplicity: 3
```

[25 other roles]..

```
dn: compositeElementName=Cordelia Role,
 compositeElementName=Casting of King Lear, ou=Cast Plays,
 dc=LarryCasting, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
objectclass: typedObjectContainerCompositeObject
objectclass: objectAssociationEnd
compositeElementName: Cordelia Role
description: references to actors who play Cordelia are here
infoType: workRole
targetType: associationEndNodeRef
targetInfoType: actorReference
minInstanceMultiplicity: 1
maxInstanceMultiplicity: 3
```

2.2.6.2.3. references to actors of the relationship roles

The references to the actors are instances of associationEndNodeRef.

```
# subordinate of King Lear Role
dn: compositeElementName=actor-1,
 compositeElementName=King Lear Role,
 compositeElementName=Casting of King Lear, ou=Cast Plays,
 dc=LarryCasting, dc=com
objectclass: top
objectclass: compositeElement
objectclass: typedObjRefCompositeElement
objectclass: associationEndNodeRef
compositeElementName: actor-1
description: reference to primary actor who plays Lear
infoType: actorReference
targetType: person
targetInfoType: actor
target: cn=patrick stewart, ou=people, dc=anEnterprisingISP, dc=com
order: 1

# subordinate of Cordelia Role
dn: compositeElementName=actor-1,
 compositeElementName=Cordelia Role,
 compositeElementName=Casting of King Lear, ou=Cast Plays,
 dc=LarryCasting, dc=com
objectclass: top
objectclass: compositeElement
objectclass: typedObjRefCompositeElement
objectclass: associationEndNodeRef
```

```
compositeElementName: actor-1
description: reference to primary actor who plays Cordelia
infoType: actorReference
targetType: person
targetInfoType: actor
target: cn=diana rigg, ou=people, dc=someOtherISP, dc=com
order: 1
```

2.2.6.2.4. people

Person objects, extended to be composite objects, contain workRoles container and references to their roles.

2.2.6.2.4.1. patrick stewart

```
version: 1
# patrick stewart
dn: cn=patrick stewart, ou=people, dc=anEnterprisingISP, dc=com
objectclass: top
objectclass: person
objectclass: compositeObjectAuxClass
description: patrick stewart is a whole person, not just an actor
infoType: actor
infoType: gardener

# container for references to work roles, a resume
dn: compositeElementName=work roles, cn=patrick stewart,
 ou=people, dc=anEnterprisingISP, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
objectclass: typedObjectContainerCompositeObject
objectclass: objectAssociationRole
compositeElementName: work roles
description: this is my work
infoType: workRoles
targetType: associationRoleRef
targetInfoType: workRoleRef

# references to roles

# reference to King Lear Role
dn: compositeElementName=King Lear,
 compositeElementName=work roles, cn=patrick stewart,
 ou=people, dc=anEnterprisingISP, dc=com
objectclass: top
objectclass: compositeElement
objectclass: typedObjRefCompositeElement
```

```
   objectclass: associationRoleRef
   compositeElementName: King Lear
   description: cast by LarryCasting
   infoType: workRoleRef
   targetType: objectAssociationEnd
   targetInfoType: workRole
   target: compositeElementName=King Lear Role,
    compositeElementName=Casting of King Lear, ou=Cast Plays,
    dc=LarryCasting, dc=com

   # reference to Captain Jean Luc Picard Role
   dn: compositeElementName=Captain Jean Luc Picard,
    compositeElementName=work roles, cn=patrick stewart,
    ou=people, dc=anEnterprisingISP, dc=com
   objectclass: top
   objectclass: compositeElement
   objectclass: typedObjRefCompositeElement
   objectclass: associationRoleRef
   compositeElementName: Captain Jean Luc Picard
   description: as seen on TV
   infoType: workRoleRef
   targetType: objectAssociationEnd
   targetInfoType: workRole
   target: compositeElementName=Captain Jean Luc Picard Role,
    compositeElementName=Casting of Star Trek TNG, ou=Star Trek TNG,
    dc=starTrekForever, dc=org
```

2.2.6.2.4.1. diana rigg

```
   version: 1
   # diana rigg
   dn: cn=diana rigg, ou=people, dc=someOtherISP, dc=com
   objectclass: top
   objectclass: person
   objectclass: compositeObjectAuxClass
   description: diana rigg is a whole person, not just an actor
   infoType: actor
   infoType: martial arts expert
   infoType: e-Custodian

   # container for references to work roles, a resume
   dn: compositeElementName=work roles, cn=diana rigg,
    ou=people, dc=someOtherISP, dc=com
   objectclass: top
   objectclass: compositeElement
   objectclass: compositeObject
   objectclass: typedObjectContainerCompositeObject
```

```
   objectclass: objectAssociationRole
   compositeElementName: work roles
   description: this is my work
   infoType: workRoles
   targetType: associationRoleRef
   targetInfoType: workRoleRef

   # references to roles

   # reference to Cordelia Role
   dn: compositeElementName=Cordelia,
    compositeElementName=work roles, cn=diana rigg,
    ou=people, dc=someOtherISP, dc=com
   objectclass: top
   objectclass: compositeElement
   objectclass: typedObjRefCompositeElement
   objectclass: associationRoleRef
   compositeElementName: Cordelia
   description: cast by LarryCasting
   infoType: workRoleRef
   targetType: objectAssociationEnd
   targetInfoType: workRole
   target: compositeElementName=Cordelia Role,
    compositeElementName=Casting of King Lear, ou=Cast Plays,
    dc=LarryCasting, dc=com

   # reference to Mrs Emma Peel Role
   dn: compositeElementName=Mrs Emma Peel,
    compositeElementName=work roles, cn=diana rigg,
    ou=people, dc=someOtherISP, dc=com
   objectclass: top
   objectclass: compositeElement
   objectclass: typedObjRefCompositeElement
   objectclass: associationRoleRef
   compositeElementName: Mrs Emma Peel
   description: as seen on TV
   infoType: workRoleRef
   targetType: objectAssociationEnd
   targetInfoType: workRole
   target: compositeElementName=Mrs Emma Peel Role,
    compositeElementName=Casting of The Avengers, ou=The Avengers,
    dc=theAvengersForever, dc=org

   # reference to Electronic Building Custodian Role
   # see F.3 of this document
   dn: compositeElementName=Electronic Building Custodian,
    compositeElementName=work roles, cn=diana rigg,
    ou=people, dc=someOtherISP, dc=com
```

```
   objectclass: top
   objectclass: compositeElement
   objectclass: typedObjRefCompositeElement
   objectclass: associationRoleRef
   compositeElementName: Electronic Building Custodian
   description: in my spare time, from the comfort of my own home!
   infoType: workRoleRef
   targetType: objectAssociationEnd
   targetInfoType: workRole
   targetInfoType: hyperDRIVE RBAC clients
   target: compositeElementName=authorized e-Custodians,
    compositeElementName=Electronic Building Custodian, ou=RBAC Roles,
    dc=electronicRealtyManagement, dc=com
```

3. Information Component Prototype Repository

The Directory schema is incapable of describing the qualities of composed information and object relationships. A Directory implementation can keep a repository of the designed composite types as Directory information. These prototypes are expressed as "stub" instances, with appropriate values assigned to pertinent attributes, and with enough composed hierarchy to demonstrate the design.

The component prototype repository can assist Directory clients who must create, interpret, and maintain these types. More importantly, the repository facilitates reuse of information component design.

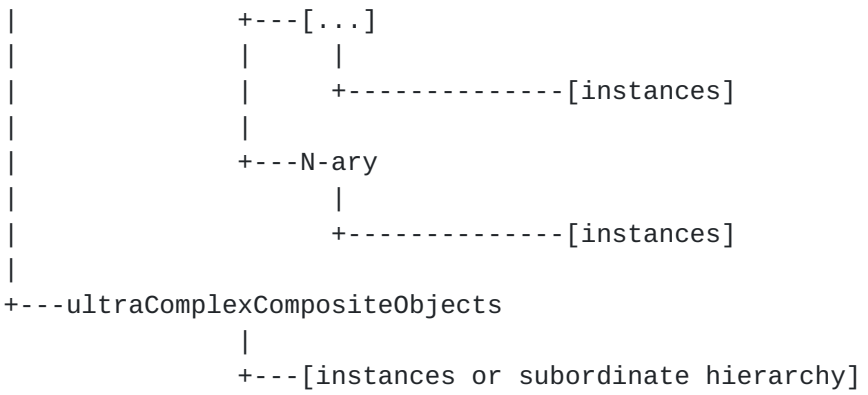
Although not strictly required by COD, the designer of a composite object can assign a numeric OID to the prototype. The OID unequivocably signifies the uniqueness of the prototype, thereby enhancing its potential for reusability. Composite object designers who extend an existing design can also signify the base type, the composite object prototype which their design extends.

COD strongly recommends that information component designers use only numeric-form OIDs to designate unique composite object types. Avoid the inevitable collisions in the uncontrolled string namespace by using only the numeric form of OID.

Since the repository and the prototypes are (merely) Directory information, they can be shared among Directory instances by existing and future mechanisms which are designed for the tasks of replicating and exporting Directory information.

As for implementation of the repository itself, a compositeElement/compositeObject construct is recommended. See Figure 3 for an illustration of one of many possible implementations.

3.1. Figure 3
Information Component Prototype Repository as a Composite Object


+---[...]
+---[instances]
+---N-ary
+---[instances]
+---ultraComplexCompositeObjects
+---[instances or subordinate hierarchy]

4. Security Considerations

Appendix F describes a Directory infrastructure which may be used to implement and manage a security policy for objects which exist outside the Directory. The data described by this model should be protected from casual observance (i.e. "browsing") and must be protected from anonymous or unauthorized manipulation. Implementors must exercise due diligence in assuring the authenticated identity of any entities which are allowed to access and manipulate the data described by this schema. The degree of rigor applied to the authentication process must be commensurate with the sensitivity of the data or processes which are represented by the schema's objects.

To this end, the authorization parameters of the Directory implementation underlying the LDAP interface, as well as the authorization policies of the LDAP interface itself, should be set to the maximum level of restriction which allows the intended functionality.

Failure to apply this strategy with due diligence may result in exposure of the assets the strategy is intended to shield.

5. Influences

In addition to the fact that there is nothing new under the sun...

The history of science and technology is full of incidents in which ideas, methods, and strategies from seemingly disparate disciplines combine with or influence one another, resulting in advances which might not have otherwise been achieved. Even more basic, but no less profound, the disciplines of horticulture and agriculture actively employ selective cross-pollination to achieve stronger, more productive hybrid varieties.

The practices of object-oriented analysis, design, and programming provide this draft's emphasis on design resue, higher-order expression through composition, and encapsulation of data, semantics, and behavior.

The influence of the design patterns movement has (I hope) kept this work on the path of reusing well-proven, well-documented designs, such as object composition, whole-part relationships, UML object associations, and the facade pattern for interfaces.

The work [16] on the science of complexity is cited for its inspiration to create a Directory information model which facilitates emergence of complex capabilities through composition and connectionism. See particularly Chapter 8, "Waiting for Carnot".

The discipline of dimensional database design [17] reinforces the appropriateness of dimensional modeling for the Directory. The lingo and jargon are SQL-oriented. But as you read Kimball's article, look for the parallelisms to the state of the traditional Directory information model. His arguments against entity-relationship (ER) data modeling apply very well to arguments against stiff and strict Directory object containerization. Just as many database traditionalists are "stuck" in the 1980's ER paradigm, the traditional Directory information model is also "stuck". And why not? It's a child of the 80's, too. In a simpler time, when the Directory was designed to serve simpler needs (and more stable organizations!), that simple model was entirely adequate. But that model has been doomed for a long time. It can't cope with increasing complexity and accelerating change. Some directory products tout their flexibility, bragging they allow implementations to choose organization-based OR location-based hierarchy for the construction of the DIT. Kimball (and COD) show us that this choice is no choice. We're much better off choosing both and more, with a Directory model which is not constricted by containerism. We're aiming to model the staggering complexity of our networked world, in a scope which wasn't imagined ten years ago. Act accordingly.

The discipline of fuzzy logic contributes greater precision through fuzz. You'll just have to read the books or follow some of the cited URLs.

A layman's exposure to hyperdimensional physics reveals the benefit of thinking very far outside the box. We can leverage the symmetries which are possible only in higher dimensions to integrate objects which appear to be completely separate and unrelated in lower dimensions.

6. References

NOTE: URL citations are current as of the date of this publication. Their content is the property of their respective authors and publishers, unless otherwise noted at the hosting site.

[1] The Directory --- overview of concepts, models and services, 1993. CCITT X.500 Series Recommendations.

[2] Wahl, M., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.
URL:http://www.ietf.org/rfc/rfc2251.txt

[3] Wahl, M., Coulbeck, A., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.
URL:http://www.ietf.org/rfc/rfc2252.txt

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
URL:http://www.ietf.org/rfc/rfc2119.txt

[5] [UML] Rational Software Corporation, "Unified Modeling Language, Version 1.1", September 1, 1997.
URL:http://www.rational.com/uml/

[6] [Java] Ken Arnold and James Gosling, "The Java(tm) Programming Language," Second Edition, ISBN 0-201-31006-6.

[7] [CORBA] The Object Management Group, "Common Object Request Broker Architecture Specification 2.0,"
URL:http://www.omg.org

[8] [XML] World Wide Web Consortium, "Extensible Markup Language",
URL:http://www.w3.org/XML/

[9] [UML Semantics] Rational Software Corporation, "UML Semantics, Version 1.1", September 1, 1997.
URL:http://www.rational.com/uml/

[10] S. Kille, "Representing Tables and Subtrees in the X.500 Directory. RFC 2293",March, 1998.
URL:http://www.ietf.org/rfc/rfc2293.txt

[11] [GHJV95] Erich Gamma, et al, "Design Patterns, Elements of Reusable Object-Oriented Software", Addison Wesley Longman, Inc., Reading, Massachusetts, 1995.

[11.b] [composite] A description of the Composite pattern is available at
URL:http://www.cpsc.ucalgary.ca/~jonesb/seng/609.04/composite.html
[11.c] [facade] A description of the Facade pattern is available at
URL:http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/patterns/tutorial.html

[12] [POSA] Frank Buschmann, et al, "Pattern-Oriented Software Architecture, A System of Patterns", John Wiley & Sons, Baffins Lane, Chichister, West Sussex, England, 1996.
[12.b] [whole-part] A description of the Whole-part pattern is available at
URL:http://www.openloop.com/softwareEngineering/patterns/designPattern/dPattern_wholePart.htm

[13] [CORBA Relationships] Object Management Group, "CORBAservices: Common Object Services Specification", Framingham, Massachusetts, December 9, 1998. Chapter 9, "Relationship Service Specification"
URL:http://www.omg.org/

[14] [Shlaer-Mellor] Sally Shlaer and Stephen J. Mellor, "Object-oriented Systems Analysis: Modeling the World in Data", Yourdon Press Computing Series, Prentiss Hall, Englewood Cliffs, NJ, 1988.

[15] [UML Notation] Rational Software Corporation, "UML Notation Guide, Version 1.1", September 1, 1997.
URL:http://www.rational.com/uml/

[16] M. Mitchell Waldrop, "Complexity, the Emerging Science at the Edge of Order and Chaos", Touchstone, Simon & Schuster, New York, 1992.

[17] Ralph Kimball, "A Dimensional Modeling Manifesto", DBMS Magazine, August 1997
URL:http://www.dbmsmag.com/9708d15.html

[18] Bart Kosko, "Fuzzy Thinking: the New Science of Fuzzy Logic", Hyperion, 114 Fifth Avenue, New York, NY 10011, 1993.
[18.b] Bart Kosko's Page
URL:http://sipi.usc.edu/~kosko/
[18.c] Berkeley Initiative in Soft Computing, Dr. Lotfi Zadeh
URL:http://http.cs.berkeley.edu/projects/Bisc/bisc.welcome.html
[18.d] Ortech Engineering's Fuzzy Logic Resevoir
URL:http://www.ortech-engr.com/fuzzy/reservoir.html

[19] Daniel McNeill and Paul Frieberger, "Fuzzy Logic: the Revolutionary Computer Technology that is Changing Our World", Touchstone, Simon & Schuster, New York, 1993.

[20] Michio Kaku, "Hyperspace: A Scientific Odyssey Through Parallel Universes,Time Warps, and the 10th Dimension", Oxford Univ. Press, 1994.
the following URL cite contains a pertinent extract from the book
URL:http://www.dorsai.org/~mkaku/mk-artcl.html#hyperspace

[21] [RMI] Java Software, Sun Microsystems, Inc., "Remote Method Invocation," November 1998.
URL:http://java.sun.com/products/jdk/1.2/docs/guide/rmi

[22] [Voyager] Voyager, Objectspace, Inc., "Voyager ORB", 1995-1999.
URL:http://www.objectspace.com/voyager/

[23] [HORB] Satoshi Hirano (Hirano-SAN!), "HORB: Distributed Execution of Java Programs", Electrotechnical Laboratory and RingServer Project, 1-1-4 Umezono Tsukuba, 305 Japan, 1997-1999.
URL:http://ring.etl.go.jp/openlab/horb/

[24] David F. Ferraiolo and Richard Kuhn, "Role Based Access Control", Proceedings of the 15th NIST-NSA National Computer Security Conference, Baltimore, MD, 13-16 October 1992.
URL:http://hissa.ncsl.nist.gov/rbac/paper/rbac1.html

[25] David F. Ferraiolo, Janet A. Cugini, D. Richard Kuhn, "Role-Based Access Control (RBAC):Features and Motivations", National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD 20899, 11th Annual Computer Security Applications Proceedings 1995.
URL:http://hissa.ncsl.nist.gov/rbac/newpaper/rbac.html

[26] L. Bartz, "LDAP Schema for Role Based Access Control", INTERNET-DRAFT <draft-bartz-hyperdrive-ldap-rbac-schema-00.txt>, October, 1997
URL:http://members.aol.com/sowsearsdg/hyperDRIVE/
draft-bartz-hyperdrive-ldap-rbac-schema-00.txt

[27] L. Bartz, "hyperDrive: Leveraging LDAP to Implement RBAC on the Web", in Proceedings of the Second ACM RBAC Workshop, pgs. 69-74, November, 1997
URL:http://members.aol.com/sowsearsdg/hyperDRIVE/ACMpaper/

[28] Anant Agarwal, "Raw Computation", Scientific American, August 1999.
URL:http://www.sciam.com/1999/0899issue/0899agarwal.html

[29] M. Smith, "Definition of an X.500 Attribute Type and an Object Class to Hold Uniform Resource Identifiers (URIs)", RFC 2079, January 1997.
URL:http://www.ietf.org/rfc/rfc2079.txt

[30] V. Ryan, R. Lee, S. Seligman, "Schema for Representing Java(tm) Objects in an LDAP Directory", <draft-ryan-java-schema-02.txt>, April 1999.
URL:ftp://ftp.ietf.org/internet-drafts/draft-ryan-java-schema-02.txt

[31] V. Ryan, R. Lee, S. Seligman, "Schema for Representing CORBA Objects in an LDAP Directory", <draft-ryan-corba-schema-02.txt>, August 1999.
URL:ftp://ftp.ietf.org/internet-drafts/draft-ryan-corba-schema-02.txt

[32] Gordon Good, "The LDAP Data Interchange Format (LDIF) - Technical Specification", <draft-good-ldap-ldif-04.txt>, June 1999.
URL:ftp://ftp.ietf.org/internet-drafts/draft-good-ldap-ldif-04.txt

7. Author's Address

Larry Bartz
Internal Revenue Service
575 N. Pennsylvania Street
Attn: Stop 15
Indianapolis, IN 46204
USA

Phone: +1 317 226-7060
Email: lbartz@parnelli.indy.cr.irs.gov

A. Appendix A - Objectclasses and Attributes

The attribute type and object class definitions are written using the BNF form of AttributeTypeDescription and ObjectClassDescription given in [3]. Lines have been folded for readability.

A.1. Numeric OID production

2 - ISO/ITU-T jointly assigned OIDs

2.16 - Joint assignments by country

2.16.840 - USA

2.16.840.1 - USA Company

2.16.840.1.101 - U.S. Government

2.16.840.1.101.10 - General Services Administration Root

2.16.840.1.101.10.2 - Department of the Treasury

2.16.840.1.101.10.2.30 - Internal Revenue Service

2.16.840.1.101.10.2.30.1 - Directory Attributes

2.16.840.1.101.10.2.30.1.2 - COD project attributes

2.16.840.1.101.10.2.30.2 - Directory Objectclasses

2.16.840.1.101.10.2.30.2.2 - COD project objectclasses

A.2. Composite Object base objectclass specifications

A.2.1. compositeElement

The elemental building block of information composition in COD. Many subtypes are possible. All designs must explicitly subtype for precise, unambiguous, and typesafe usage. This objectclass does not contain an attribute for the "value" component of the "name=value" pair.

```
( 2.16.840.1.101.10.2.30.2.2.1.1
    NAME 'compositeElement'
    SUP top
    STRUCTURAL
    MUST compositeElementName
    MAY ( description $ infoType $
    minInstanceMultiplicity $
    maxInstanceMultiplicity $
    order $ compositeOID $ extendsCompositeOID $
    isCompositeTemplate ) )
```

A.2.2. compositeObject

The "head" or "root" of a composite object. A compositeObject contains compositeElements and/or compositeObjects. It is a subtype of compositeElement so that components which possess internal hierarchy can be assembled. See "Composite" of [GHJV95].

```
( 2.16.840.1.101.10.2.30.2.2.1.2
    NAME 'compositeObject'
    SUP compositeElement
    STRUCTURAL )
```

A.2.3. typedObjectContainerCompositeObject

A compositeObject container in which the types of the subordinates can be specified. Assurance of these types requires compliance with this object model. The Directory protocol itself will not assure typesaftey.

```
( 2.16.840.1.101.10.2.30.2.2.1.15
    NAME 'typedObjectContainerCompositeObject'
    SUP compositeObject
    STRUCTURAL
    MUST targetType
```

```
                    MAY targetInfoType  )
```

A.2.4. boxOfCompositrons

A typedObjectContainerCompositeObject explicitly for holding objects, such as described in [29], [30], and [31], which are references to compositrons.

```
                ( 2.16.840.1.101.10.2.30.2.2.1.16
                    NAME 'boxOfCompositrons'
                    SUP typedObjectContainerCompositeObject
                    STRUCTURAL
                    )
```

A.2.5. compositeObjectAuxClass

By applying this to an entry (entry becomes a member of this object-class), the entry becomes the "head" or "root" of a composite object.

```
                ( 2.16.840.1.101.10.2.30.2.2.1.17
                    NAME 'compositeObjectAuxClass'
                    SUP top
                    AUXILIARY
                    MAY ( compositeElementName $ description
                    $ infoType $ superstructureObjectClass  )  )
```

A.3. attribute specifications for Composite Object base objectclasses

A.3.1. compositeElementName

```
                ( 2.16.840.1.101.10.2.30.1.2.1.1
                    NAME 'compositeElementName'
                    DESC 'naming attribute for a compositeElement
                    object'
                    EQUALITY caseIgnoreMatch
                    SUBSTR caseIgnoreSubstringsMatch
                    ORDERING caseIgnoreOrderingMatch
                    SYNTAX '1.3.6.1.4.1.1466.115.121.1.15' )
```

A.3.2. infoType

Potentially multi-valued string which signifies nuances of the object instance. This attribute can be used to allow information to describe

itself, or for information models to describe their components within the context of the composite object.

```
( 2.16.840.1.101.10.2.30.1.2.1.2
    NAME 'infoType'
    DESC 'name by which implementations can
    form informal classification systems for
    subtypes of composite objects'
    EQUALITY caseIgnoreMatch
    SUBSTR caseIgnoreSubstringsMatch
    ORDERING caseIgnoreOrderingMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.15' )
```

A.3.3. minInstanceMultiplicity

For compositeObject, the minimum number of subordinate objects.
For attribute-bearing compositeElements, the minimum number of values in the *Value attribute.
For reference-bearing compositeElements, the minimum number of objects referenced.

```
( 2.16.840.1.101.10.2.30.1.2.1.5
    NAME 'minInstanceMultiplicity'
    DESC 'minimum number of instances of
    the primitive type which are listed held
    in or referenced by this composite element'
    EQUALITY integerMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.27'
    SINGLE-VALUE )
```

A.3.4. maxInstanceMultiplicity

For compositeObject, the maximum number of subordinate objects.
For attribute-bearing compositeElements, the maximum number of values in the *Value attribute.
For reference-bearing compositeElements, the maximum number of objects referenced.

```
( 2.16.840.1.101.10.2.30.1.2.1.6
    NAME 'maxInstanceMultiplicity'
    DESC 'maximum number of instances of the
    primitive type which are listed held in
    or referenced by this composite element'
    EQUALITY integerMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.27'
    SINGLE-VALUE )
```

A.3.5. order

```
( 2.16.840.1.101.10.2.30.1.2.1.7
    NAME 'order'
    DESC 'integer indication of sequence or
    order in which peer compositeElements are
    to be traversed or evaluated'
    EQUALITY integerMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.27'
    SINGLE-VALUE )
```

A.3.6. targetType

```
( 2.16.840.1.101.10.2.30.1.2.2.2
    NAME 'targetType'
    DESC 'an OID which unambiguously designates
    the type of the targeted object'
    EQUALITY objectIdentifierMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.38'
    )
```

A.3.7. targetInfoType

```
( 2.16.840.1.101.10.2.30.1.2.2.3
    NAME 'targetInfoType'
    DESC 'matches the infoType value of the
    targeted objects'
    EQUALITY caseIgnoreMatch
    SUBSTR caseIgnoreSubstringsMatch
    ORDERING caseIgnoreOrderingMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.15'
    )
```

A.4. Object Association base objectclass specifications

A.4.1. objectAssociation

This objectclass represents the relationship itself as an object. Just as a relationship is composed of the "ends" [UML Semantics] or "roles" [CORBA Relationships], an instance of objectAssociation contains objectAssociationEnd objects.

```
( 2.16.840.1.101.10.2.30.2.2.3.1
```

```
NAME 'objectAssociation'
SUP typedObjectContainerCompositeObject
STRUCTURAL
MAY degree  )
```

A.4.2. objectAssociationEnd

A hierarchical subordinate of an objectAssociation, this composite object represents a role in the relationship. It is a container of references to nodes (the actors) which play this role in the relationship.

```
( 2.16.840.1.101.10.2.30.2.2.3.5
    NAME 'objectAssociationEnd'
    SUP typedObjectContainerCompositeObject
    STRUCTURAL
    )
```

A.4.3. objectAssociationRole

An actor in a role contains or refers to this composite object, which contains references to the roles the actor plays.

```
( 2.16.840.1.101.10.2.30.2.2.3.3
    NAME 'objectAssociationRole'
    SUP typedObjectContainerCompositeObject
    STRUCTURAL )
```

A.4.4. typedObjRefCompositeElement

Provides NOT for typesafety, but for type-signing. The container is signed with the type of its intended target. The targetType attribute signifies the intended type of the object referenced by the distinguishedName attribute "target".

Directory clients which adhere to this information model MUST use this information to assure typesafety.

```
( 2.16.840.1.101.10.2.30.2.2.1.14
    NAME 'typedObjRefCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MUST ( target $ targetType )
    MAY ( targetInfoType $ fuzzFactor ) )
```

A.4.5. associationRoleRef

This composite element is contained within an objectAssociationRole composite object. It refers to a role.

```
( 2.16.840.1.101.10.2.30.2.2.3.6
    NAME 'associationRoleRef'
    SUP typedObjRefCompositeElement
    STRUCTURAL
    )
```

A.4.6. associationEndNodeRef

This composite element is contained within an objectAssociationEnd composite object. It refers to an actor of the role.

```
( 2.16.840.1.101.10.2.30.2.2.3.7
    NAME 'associationEndNodeRef'
    SUP typedObjRefCompositeElement
    STRUCTURAL
    )
```

A.4.7. objectAssociationAuxClass

By applying this to an entry (entry becomes a member of this object-class), the entry becomes the "head" or "root" of an object association.

```
( 2.16.840.1.101.10.2.30.2.2.3.8
    NAME 'objectAssociationAuxClass'
    SUP compositeObjectAuxClass
    AUXILIARY
    MAY ( degree $
    superstructureObjectClass ) )
```

A.5. attribute specifications for Object Association base objectclasses

A.5.1. degree

The degree of an object association is the number of roles which participate in the relationship.

```
( 2.16.840.1.101.10.2.30.1.2.3.1
    NAME 'degree'
```

```
        DESC 'integer indication of the number of
        objectAssociationEnd or relationship roles
        which compose the relationship or association'
        EQUALITY integerMatch
        SYNTAX '1.3.6.1.4.1.1466.115.121.1.27'
        SINGLE-VALUE )
```

A.5.2. target

```
    ( 2.16.840.1.101.10.2.30.1.2.1.18
        NAME 'target'
        DESC 'DN of the targeted object'
        EQUALITY distinguishedNameMatch
        SYNTAX '1.3.6.1.4.1.1466.115.121.1.12' )
```

A.5.3. fuzzFactor

```
    ( 2.16.840.1.101.10.2.30.1.2.1.25
        NAME 'fuzzFactor'
        DESC 'indication of the percentage of a
        whole which is contained in this part.
        Represents a value between 0 and 1 with an
        implicit leading decimal point. LEADING
        ZEROES ARE SIGNIFICANT!'
        EQUALITY numericStringMatch
        SUBSTR numericStringSubstringsMatch
        SYNTAX '1.3.6.1.4.1.1466.115.121.1.36' )
```

A.6. attribute-bearing "Raw" composite element objectclass specifications

See 2.1.2.2. for discussion of raw types.

These compositeElement subtypes support many of the attribute syntaxes described in 4.3.2 of RFC 2252 [3]; at least those syntaxes which appeared interesting for creating composite objects. If I've omitted any which should be included, please let me know.

These objectclasses represent attributes and lists of attributes in the composite object framework. Use of these objectclasses to represent attributes allows the implementor to impart qualities to the attributes; qualities which which cannot be expressed in the Directory schema.

These objectclasses also impart the qualities of addressability and

sharability to the values they contain. Thus, attributes attain the status of objects for many purposes.

A.6.1. stringCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.3
    NAME 'stringCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY stringCompositeElementValue  )
```

A.6.2. integerCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.4
    NAME 'integerCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY integerCompositeElementValue  )
```

A.6.3. numericStringCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.5
    NAME 'numericStringCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY numericStringCompositeElementValue  )
```

A.6.4. generalizedTimeCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.6
    NAME 'generalizedTimeCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY generalizedTimeCompositeElementValue  )
```

A.6.5. labeledURIcompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.7
    NAME 'labeledURIcompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY ( labeledURIcompositeElementValue $
    targetType $ targetInfoType  )  )
```

A.6.6. jpegCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.8
    NAME 'jpegCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY jpegCompositeElementValue  )
```

A.6.7. audioCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.9
    NAME 'audioCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY audioCompositeElementValue  )
```

A.6.8. telephoneNumberCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.10
    NAME 'telephoneNumberCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY telephoneNumberCompositeElementValue  )
```

A.6.9. octetStringCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.11
    NAME 'octetStringCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY octetStringCompositeElementValue  )
```

A.6.10. booleanCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.12
    NAME 'booleanCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY booleanCompositeElementValue  )
```

A.6.11. binaryCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.20
    NAME 'binaryCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY binaryCompositeElementValue  )
```

A.6.12. oidCompositeElement

```
( 2.16.840.1.101.10.2.30.2.2.1.13
    NAME 'oidCompositeElement'
    SUP compositeElement
    STRUCTURAL
    MAY oidCompositeElementValue  )
```

A.6.13. typedAlias

A special building block of composite information. Must be outside of the compositeElement inheritance hierarchy in order to take advantage of the Directory's special treatment of the alias base type.

Provides NOT for typesafety, but for type-signing. The alias is signed with the type of its intended target. The targetType attribute signifies the intended type of the object referenced by the alias.

Directory clients which adhere to this information model MUST use this information to assure typesafety.

```
( 2.16.840.1.101.10.2.30.2.2.2.1
    NAME 'typedAlias'
    SUP alias
    STRUCTURAL
    MUST ( typedAliasName $ targetType  )
    MAY ( targetInfoType $ order $ fuzzFactor )  )
```

A.7. attribute specifications for attribute-bearing "Raw" composite element objectclasses

A.7.1. booleanCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.19
    NAME 'booleanCompositeElementValue'
    DESC 'boolean value '
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.7'
    SINGLE-VALUE )
```

A.7.2. binaryCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.20
    NAME 'binaryCompositeElementValue'
    DESC 'binary value '
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.5'
    SINGLE-VALUE )
```

A.7.3. stringCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.8
    NAME 'stringCompositeElementValue'
    DESC 'text or a string '
    EQUALITY caseIgnoreMatch
    SUBSTR caseIgnoreSubstringsMatch
    ORDERING caseIgnoreOrderingMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.15' )
```

A.7.4. oidCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.9
    NAME 'oidCompositeElementValue'
    DESC 'an OID '
    EQUALITY objectIdentifierMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.38' )
```

A.7.5. integerCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.10
    NAME 'integerCompositeElementValue'
    DESC 'an integer value '
    EQUALITY integerMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.27' )
```

A.7.6. numericStringCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.11
    NAME 'numericStringCompositeElementValue'
    DESC 'a numeric string '
    EQUALITY numericStringMatch
    SUBSTR numericStringSubstringsMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.36' )
```

A.7.7. generalizedTimeCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.12
    NAME 'generalizedTimeCompositeElementValue'
    DESC 'a generalizedTime parameter'
    EQUALITY generalizedTimeMatch
    ORDERING generalizedTimeOrderingMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.24' )
```

A.7.8. labeledURIcompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.13
    NAME 'labeledURIcompositeElementValue'
    DESC 'a labeledURI as per rfc2079 '
    EQUALITY caseExactIA5Match
    SUBSTR caseIgnoreSubstringsMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.26' )
```

A.7.9. jpegCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.14
    NAME 'jpegCompositeElementValue'
    DESC 'jpeg value '
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.28' )
```

A.7.10. audioCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.15
    NAME 'audioCompositeElementValue'
    DESC 'audio value '
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.4' )
```

A.7.11. telephoneNumberCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.16
    NAME 'telephoneNumberCompositeElementValue'
    DESC 'telephoneNumber value '
    EQUALITY telephoneNumberMatch
    SUBSTR telephoneNumberSubstringsMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.50' )
```

A.7.12. octetStringCompositeElementValue

```
( 2.16.840.1.101.10.2.30.1.2.1.17
    NAME 'octetStringCompositeElementValue'
    DESC 'octetString value '
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.40' )
```

A.8. design-supporting attributes

These attributes are used primarily as design specifications in the Information Component Prototype Repository.

A.8.13. compositeOID

This attribute looks like a numeric OID, but is a directoryString. The canonical OID syntax and related matching rules are not used here. Since compositeOID is not part of the Directory's schema, it cannot use the OID syntax and matching rules.

```
( 2.16.840.1.101.10.2.30.1.2.1.21
    NAME 'compositeOID'
    DESC 'directoryString OID-semantic which
    uniquely identifies a composite prototype'
    EQUALITY caseIgnoreMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.15'
    SINGLE-VALUE )
```

A.8.2. extendsCompositeOID

```
( 2.16.840.1.101.10.2.30.1.2.1.22
    NAME 'extendsCompositeOID'
    DESC 'compositeOID of composite prototype which
    this extends'
    EQUALITY caseIgnoreMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.15'
    SINGLE-VALUE )
```

A.8.3. isCompositeTemplate

```
( 2.16.840.1.101.10.2.30.1.2.1.24
    NAME 'isCompositeTemplate'
    DESC 'TRUE indicates this information
    is a template or prototype'
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.7'
    SINGLE-VALUE )
```

A.8.4. superstructureObjectClass

Exists as a design specification attribute for compositeObjectAux-Class and objectAssociationAuxClass. Necessary for use in LDAP because (as asserted in RFC 2251) many LDAP servers do not use dit-ContentRule.

```
( 2.16.840.1.101.10.2.30.1.2.1.23
    NAME 'superstructureObjectClass'
    DESC 'OID of structural and auxiliary
    classes to which this entry must belong '
    EQUALITY objectIdentifierMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.38' )
```

A.8.5. typedAliasName

```
( 2.16.840.1.101.10.2.30.1.2.2.1
    NAME 'typedAliasName'
    DESC 'naming attribute for a typedAlias
    object'
    EQUALITY caseIgnoreMatch
    SUBSTR caseIgnoreSubstringsMatch
    ORDERING caseIgnoreOrderingMatch
    SYNTAX '1.3.6.1.4.1.1466.115.121.1.15' )
```

B. Appendix B - Context and Motivation: directory object relationship practices

We describe current practice, limitations, and deficiencies as a context which motivates this model for Directory object relationships.

In current practice, Directory object relationships are created and described by structural containment, by object references, and by various implementation-specific constructs which are combinations of these.

B.1 Structural Containment

Containment hierarchies within the Directory information tree (DIT) are often employed to impart relational structure to collections of Directory objects. This is a natural consequence of one of the Directory's central design themes, its built-in hierarchical order. Structural containment naturally suits the requirements of object relationships which are hierarchical and relatively stable.

Object relationships constructed from containment are difficult to reuse. As any given Directory object can have only one immediate structural superior, structural containment inhibits the reusability of the information held by Directory objects. The relational context of superior/subordinate can be described only once within a given DIT with respect to a particular object.

Object relationships constructed by containment are brittle relationships. The lifecycle of a subordinate object is limited to the lifecycle of its immediate DIT superior.

Many potential object relationships are not hierarchical and stable. The superior/subordinate relationship represents only one of many potential kinds of Directory object relationships. Structural containment alone cannot describe ternary and N-ary relationships among objects.

B.2 Object References

The Directory standards provide the alias objectclass and distinguishedName attribute for establishing relationships between Directory objects which are neither based upon, nor constrained by, structural containment. These object references help Directory-based information models and Directory implementations overcome or avoid some of the restrictions of structural containment, but not all.

These simple reference mechanisms are suitable for describing

unidirectional binary relationships; one object refers to another. A single object may refer to many, by virtue of a list of distinguishedName attributes, or by containment of more than one alias object. There is no standard construct for describing ternary or N-ary Directory object relationships.

The canonical distinguishedName and alias types possess no information regarding the types of the objects to which they refer.

Standard subclasses of these simple reference types have been promulgated, in efforts to impart a sense of the intended type of the target of the reference. These subclasses rely simply upon their own names and descriptions as hints to Directory administrators, programmers, and clients. These Directory users must, in turn, be aware of this purely semantic information and act accordingly. There is no way to computationally discover or assure the type of the target object from the name and description of the subclassed alias objectclass or subtyped distinguishedName attribute.

B.3 implementation-specific Constructs

Custom-made, single- or limited-purpose constructs can represent complex object relationships in the Directory. The problem with such constructs is that they are not generally reusable because they are not designed for reuse. Instead, each is designed to solve a specific and limited problem.

Moreover, more sophisticated information models are denied their full realization in the Directory because there is no relationship model which is expressive enough to describe them.

Worse, the client software (and wetware!) necessary to cope with a myriad of implementation-specific object relationship constructs cannot be reused.

C. Appendix C - Composite Semantics and Behavior

C.1. Composite Object and Object Relationship Semantics

We've talked about learning and knowledge in the Directory, capabilities which are facilitated by COD. But where is the intelligence? How can the Directory harbor the semantics and behavior which are intended by the design of composite objects and object associations? Following the analogy of the play, we have the roles and actors, now where's the script?

C.1.1. Wetware

Since composite objects are composed of "traditional" Directory objects and relationships, they can be created (in their elemental parts) and navigated by traditional Directory client software. This places the entire responsibility for interpretation, for composite intelligence, in wetware. Since the configuration and performance of wetware varies widely, this approach is unlikely to provide consistently satisfying results.

C.1.2. Custom client software

Designers and implementors of COD-enabled systems will write custom software, leveraging the canonical Directory APIs and widely available toolkits. Such software will create and interpret composite objects according to their design parameters.

C.1.3. Compositrons - The hyper-Dimensional Directory

Sometimes we have to do more than think outside the box. We have to think outside the universe. Theoretical physicists have determined that it is very likely that our universe is composed of ten or twenty-six dimensions, rather than the obvious three or four which we ordinarily perceive. Working in these higher dimensions, physicists find large-scale symmetries, integrations of rules and behaviors which, in our four-dimensional universe, appear to be disjoint and unrelated.

C.1.3.1. Life in Flatland

In "Parable of the Gemstone" in [20], Dr. Michio Kaku succinctly illustrates the problems human beings face when they attempt to perceive objects and behaviors which exist in universes of more than three spatial dimensions.

In Kaku's parable, beings called Flatlanders live in a two-dimensional "flat" universe. An object from another dimension, a three-dimensional gemstone, falls into Flatland and shatters. Although the Flatlanders can identify the pieces of the gemstone, their limited (two-dimensional) perception of the universe prevents them from beholding the totality and true beauty of the gemstone. They can't reassemble the gemstone and see it for what it is because they cannot perceive "up".

Our composite objects, insofar as they exist in the Directory, are comparable to the Flatlanders' gemstone. We can see and touch pieces (attributes and objectclasses) of the gemstone in our flat Directory universe. But without the additional dimensions of composite semantics and composite behavior, we can never behold the true beauty and meaning of the gemstone (composite object) in its totality.

Flatland has no "up" dimension. The Directory has no dimensions which can describe the integrated semantics and behavior of composite objects.

Just as the two-dimensional Flatlanders had to intellectualize an unknown, unseen third dimension to explain new phenomena, we'll contruct a linkage to another dimension in order to effect higher order capabilities for COD.

For COD, the higher dimension is distributed object computing; technologies such CORBA, Java RMI [21] [RMI], Voyager [22] [Voyager], HORB [23] [HORB], and the like.

C.1.3.2. Benny and the Compositrons: Episode I, Escape from Flatland

In which Benny, a Multi-Dimensional marooned in Flatland, meets the Compositrons.

[cue theme music, to the tune of "Benny and the Jets", Elton John, Bernie Taupin, 1973]

Hey, kids, plug into the faithful!
They're distributed objects,
The Direct'ry makes 'em stateful.
We'll tweak the paradigm tonight,
So stick around.
You're gonna hear a lot o' tech-speak,
Such a wonderful sound!

Oh, but they're weird and wonderful,
Compositrons are really keen!

They've got intelligence, make perfect sense,
I read about 'em in a Web e-zine!
Oh, oh, B-b-b-Benny and the Compositrons!

EDIT !!! tell Benny's story

C.1.3.3. It's a Facade

Compositrons are little software machines, distributed software nanobots, which are referenced from the "head" or "root" of a composite object. They are designed as per the "facade" [11.c] [facade] pattern of [GHJV95]. Compositrons provide a facade, an alternate interface, to composite objects.

The boxOfCompositrons objectclass is a typedObjectContainerCompositeObject explicitly for holding objects, such as described in [29], [30], and [31], which are references to compositrons.

Each Compositron is designed to support a particular kind of composite object. It provides a Compositronic Service for its own kind of composite object. A compositron is called upon to support an instance of its composite object at runtime, with the distinguishedName of the composite object instance as one of the arguments of the call.

Compositrons:

- are NOT triggers
- are distributed objects
- can be implemented in one or several distributed object flavors
- provide an alternate interface to Directory objects, a facade which integrates the composite
- encapsulate the information, semantics, and behavior of the composite object
- provide a distributed object interface for the CRUD (create, read, update, delete) operations of the entire composite object, provide composed integration
- provide the intelligent behavior intended by the composite object's designer
- provide a service, which is a distributed object interface to the composite object, its integrated semantics and behaviors

- provide higher-order dimensions, for expressing integrated semantics and behaviors of composite objects

- take their base specification from the template of the composite object

- are elaborations of the base specification

- are initialized (runtime instances) with the distinguishedName of the "head" or "root" of the composite they represent, to attain statefulness

- enforce well-formedness rules for the composite object, rules which are internalized from the composite template specification

- can interact with other Compositrons, achieving composite behavior, synthesis, and integration in the Directory

- can interact with distributed objects and other networked entities which are completely outside the Directory, to achieve synthesis and integration beyond the Directory

- provide whatever extra-Directory behavior is intended by the composite object's designer

Compositronic services allow distributed object clients to realize the potential of composite objects. Clients which are computationally unable to take advantage of Compositronic intelligence will simply use and follow the Directory data in "traditional" ways, supplying their own intelligence. Pity the Flatlanders.

D. Appendix D - the Fuzzy Directory

Fuzzy logic [18], [19] presents an alternative to traditional notions of set membership. Opposed to traditional black/white bivalence, fuzzy logic provides for infinite greyscale in expressing subsethood. Fuzzy logic admits possibility, and includes probability as a special case.

In COD, we introduce an attribute called fuzzFactor, which can describe the degree of subsethood and the strength of a relationship.

fuzzFactor is not a precise measurement, per se. It is not inches, millimeters, pounds, kilograms, hours, or electronVolts. fuzzFactor is a relational measure, which takes its meaning from its context. The context is a set, and fuzzFactor is:

- extent to which the referent is a member of this set
- referent's degree of membership in this set
- strength of the relatedness of the referent to this set
- part (percentage) of the whole to which this refers is in this set
- initialized and tuned by experience

We represent fuzzFactor in COD as an attribute of the typed relationship objectclasses. The fuzzFactor attribute is a numeric string with an implied leading decimal point. Its value is interpreted as always being between zero and one. Leading zeroes are significant.

We illustrate with two simplistic examples:

Example 1, fuzzy location: An employee works at two locations, 75% in Indianapolis, 25% in Washington, DC. How does the Directory represent his location? In a black/white, either/or model, we typically quantize such information, choosing only one location; the one with the highest percentage. This is a very lossy quantization. We lose important information. The fact that the employee spends 25% of his worklife in Washington is lost. We can use fuzzy set membership to more accurately represent the truth. Make the employee a member of the set of employees located at Indianapolis with a fuzzFactor of 75, and a member of the set of employees located at Washington with a fuzzFactor of 25.

Example 2, fuzzy organizational unit: An employee is the subject of an organizational management strategy known as matrixed management. The "official" formal organization chart (and traditional Directory DIT) shows the employee as a 100% member of one specific organizational unit (OU-a). But in truth, the employee is assigned duties and responsibilities in three organizational units. By formal agreement of the three managers of the three organizational units, the employee's work efforts are to be divided among the units as OU-a=51%, OU-b=25%, OU-c=24%. Use the percentages as fuzzFactors, to accurately represent the employee's degrees of membership in each of the organizational units. The same strategy can be applied to project teams, task forces, and the like.

This extension can facilitate Hebbian learning, fuzzy neural net, and fuzzy cognitive map behaviors in the Directory.

In Hebbian (after neuroscientist Donald Hebb) learning, knowledge is classified and used based on the strengths of the relationships among information nodes, the strengths of the synapses which connect the neurons. In COD, the Compositrons are active representatives of the composite objects. Compositrons are neurons. Fuzzy relationships (among composite objects and their Compositrons) are the synapses.

As for fuzzy neural net and fuzzy cognitive map behaviors, read the books and follow the links given in this paper's References section. Think of Compositrons as neurons (active nodes of knowledge or facts), fuzzy relationships as synapses. Imagine the behaviors of Compositrons intertwining and interlocking, swirling and cascading, all in response to perhaps a single external event.

E. Appendix E - example usage - adaptive XML repository

What is an XML document? It's just a composite object, isn't it? And shouldn't XML be stored in an object-oriented data repository? And what is the most widely deployed object-oriented data repository around? Maybe it's not the Directory, but why shouldn't it be? Are you with me here?

XML could benefit by using the Directory as a datastore. The Directory is hierarchical AND object-oriented, thorough indexing is built in, and it supports high performance search and retrieval.

What about XML standards? Are they stable? Will they ever stop creating new types? Do you have enough time to create traditional Directory attributes and object classes to keep up? How can you, when one of XML's principal paradigms is "roll your own"?

Here's where we capitalize upon the capability of composition to enable emergence. Dynamic adaptabilty.

XML describes itself. Isn't that the mantra? We can use COD to listen to that description and behave adaptively.

Here's a possible scenario. You develop an application which accepts XML documents as input. As one comes in, parse it (or read its DTD) to get its structure (XML elements). From this information, create a hierarchical composite object skeleton which matches the structure of the XML document. From the document's (or DTD's) description of the attributes, use COD's "raw" stringCompositeElement to hold the values of the attributes. Use the "order" attribute to preserve the XML document's sequencing of elements and attributes.

Use the composite types, in conjunction with arbitrarily complex hierarchy enabled by COD, and the multi valued infoType attribute to create types of XML elements and attributes on the fly.

When you're done, we'll have an object-oriented repository which can store and return any XML document.

Some may encounter a limitation in storing XML documents in the Directory. XML requires support for the international UTF-8 AND UTF-16 character representations, but LDAP only supports UTF-8. So COD's stringCompositeElement objectclass is not adequate for UTF-16 data. LDAP does provide the octetString syntax, which should be suitable for storing UTF-16 data. COD provides the octetStringCompositeElement objectclass which supports it. Your XML-parser-directory-stuffer-retriever application could use COD's octetStringCompositeElement to store UTF-16 data. The catch is that your Directory

implementation may not be capable of indexing or searching octet-String. The matching syntax family for octetString (octetStringMatch, octetStringOrderingMatch, octetStringSubstrinsMatch) are not included among the identified "SHOULD" matching syntaxes of RFC 2252. Your LDAP may support the octetString matching, or not.

I don't think this COD usage conflicts with the DSML (http://www.dsml.org), at least not from the reading I've done. DSML will provide an XML interface to Directory objects, a mechanism to export Directory objects to the XML universe. What I'm talking about is using the Directory as a repository for XML documents. It's another side of the coin.

E.1. XML example with LDIF representation

E.1.1. a simple XML document

A simple XML example, borrowed from Ron Bourret's "Declaring Elements and Attributes in an XML DTD", http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/xmldtd.html

```
<?xml version="1.0" ?>
<Book Author="Anonymous">
   <Title>Sample Book</Title>
   <Chapter id="1">
       This is chapter 1.  It is not very long or interesting.
   </Chapter>
   <Chapter id="2">
       This is chapter 2.  Although it is longer than chapter 1,
       it is not any more interesting.
   </Chapter>
</Book>
```

and its DTD

```
<!DOCTYPE Book [
   <!ELEMENT Book (Title, Chapter+)>
   <!ATTLIST Book Author CDATA #REQUIRED>
   <!ELEMENT Title (#PCDATA)>
   <!ELEMENT Chapter (#PCDATA)>
   <!ATTLIST Chapter id ID #REQUIRED>
]>
```

E.1.2. LDIF of the XML document, as a COD composite object

The example is completely described and implemented in terms of the objectclasses and attributes contained in COD. There is no requirement to extend COD for objectclasses such as "XML document", "Book", or "Chapter".

```
version: 1
dn: compositeElementName=xml199910081052-1, ou=XML Documents,
 dc=xmlMightBeThisEasy, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
compositeElementName: xml199910081052-1
infoType: xmlDocument

dn: compositeElementName=xml version,
 compositeElementName=xml199910081052-1, ou=XML Documents,
 dc=xmlMightBeThisEasy, dc=com
objectclass: top
objectclass: compositeElement
objectclass: stringCompositeElement
compositeElementName: xml version
stringCompositeElementValue: 1.0
infoType: xml version
order: 1

dn: compositeElementName=Book,
 compositeElementName=xml199910081052-1, ou=XML Documents,
 dc=xmlMightBeThisEasy, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
compositeElementName: Book
infoType: Book
order: 2

dn: compositeElementName=Book Author,
 compositeElementName=Book,
 compositeElementName=xml199910081052-1, ou=XML Documents,
 dc=xmlMightBeThisEasy, dc=com
objectclass: top
objectclass: compositeElement
objectclass: stringCompositeElement
compositeElementName: Book Author
stringCompositeElementValue: Anonymous
infoType: Book Author
order: 1
```

```
dn: compositeElementName=Title,
 compositeElementName=Book,
 compositeElementName=xml199910081052-1, ou=XML Documents,
 dc=xmlMightBeThisEasy, dc=com
objectclass: top
objectclass: compositeElement
objectclass: stringCompositeElement
compositeElementName: Title
stringCompositeElementValue: Sample Book
infoType: Title
order: 2

dn: compositeElementName=Chapter 1,
 compositeElementName=Book,
 compositeElementName=xml199910081052-1, ou=XML Documents,
 dc=xmlMightBeThisEasy, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
compositeElementName: Chapter 1
infoType: Chapter
order: 3

dn: compositeElementName=id,
 compositeElementName=Chapter 1,
 compositeElementName=Book,
 compositeElementName=xml199910081052-1, ou=XML Documents,
 dc=xmlMightBeThisEasy, dc=com
objectclass: top
objectclass: compositeElement
objectclass: stringCompositeElement
compositeElementName: id
stringCompositeElementValue: 1
infoType: Chapter ID
order: 1

dn: compositeElementName=PCDATA,
 compositeElementName=Chapter 1,
 compositeElementName=Book,
 compositeElementName=xml199910081052-1, ou=XML Documents,
 dc=xmlMightBeThisEasy, dc=com
objectclass: top
objectclass: compositeElement
objectclass: stringCompositeElement
compositeElementName: PCDATA
stringCompositeElementValue: This is chapter 1.  It is
 not very long or interesting.
infoType: PCDATA
```

```
   order: 2

   dn: compositeElementName=Chapter 2,
    compositeElementName=Book,
    compositeElementName=xml199910081052-1, ou=XML Documents,
    dc=xmlMightBeThisEasy, dc=com
   objectclass: top
   objectclass: compositeElement
   objectclass: compositeObject
   compositeElementName: Chapter 2
   infoType: Chapter
   order: 4

   dn: compositeElementName=id,
    compositeElementName=Chapter 2,
    compositeElementName=Book,
    compositeElementName=xml199910081052-1, ou=XML Documents,
    dc=xmlMightBeThisEasy, dc=com
   objectclass: top
   objectclass: compositeElement
   objectclass: stringCompositeElement
   compositeElementName: id
   stringCompositeElementValue: 2
   infoType: Chapter ID
   order: 1

   dn: compositeElementName=PCDATA,
    compositeElementName=Chapter 2,
    compositeElementName=Book,
    compositeElementName=xml199910081052-1, ou=XML Documents,
    dc=xmlMightBeThisEasy, dc=com
   objectclass: top
   objectclass: compositeElement
   objectclass: stringCompositeElement
   compositeElementName: PCDATA
   stringCompositeElementValue: This is chapter 2.  Although it
    is longer than chapter 1, it is not any more interesting.
   infoType: PCDATA
   order: 2
```

F. Appendix F - example usage - hyperDRIVE, RBAC for network applications

Think about how enterprises are continually reinventing themselves. How stable has your organizational hierarchy been over the last three years? Will it be any more stable over the next three? And beyond?

Business process and organizational infrastructure specialists tell us that the future holds little in store for stable organizational structures. Employees, consultants, contractors, project teams, and work groups will shift, change, and blur in a continuous dance of adaptation.

This future requires an information model which adapts nimbly, quickly, and capably. Mapping persons to their e-work and work groups will not be as simple as plucking them from one organizational unit and plopping them into another. The DIT ain't gonna cut it. The unfortunate pun fits exactly; it's too wooden, stiff and brittle.

More likely, persons will map into more than one "organization", which we might as well start calling by a different name. "Organization" implies stability and hierarchy. Workers in the now and future enterprise are engaged in an evolving variety of relationships in which they are are connected to projects, tasks, business goals. These relationships (now call them roles) are a much closer match to business reality than the traditional wooden DIT.

Role Based Access Control (RBAC, [24], [25]) is an authorization strategy in which an entity's permission to access and manipulate targeted resources is determined by the entity's role or function within a certain organizational context. RBAC's principal motivation is to streamline security policy administration. Many discrete authorizations can be aggregated within a defined role. One or many roles may be assigned or attributed to individuals.

hyperDRIVE [26], [27] is the working name of a strategy for implementing authentication, authorization, and access services in an n-tiered internet computing environment. hyperDRIVE employs LDAP to store and access hyperDRIVE-defined data objects which are evaluated and manipulated to implement Role Based Access Control.

The widespread transition to Web-based and associated internet technology computing platforms has provided fertile ground for the germination and cultivation of authorization strategies. The degrees of sophistication and effectiveness of the many currently available approaches vary widely. None has yet proven itself clearly superior. None yet provides for the economical scalability necessary to support integrated internet or intranet computing environments which are

composed of many applications, hosted by many servers.

While RBAC is well recognized as a strategy which reduces the cost and complexity of security administration, RBAC which is implemented on a host-by-host basis is still potentially inadequate in a multiple host environment. The core feature of internet computing, its facile interconnectedness, reveals the limited effectiveness of authorization strategies which are implemented on a per-host basis. When the customers of network computing resources shift their computing focus among many multiply-tiered applications (which are hosted by many separate servers) via mere mouse clicks, the potential for incongruity and inconsistency among the many host-based authorization implementations becomes obvious. Who (or what mechanism) can assure that an individual's authorizations don't conflict when the authorization controls are host-based?

Host-based authorization strategies in a network computing environment are inherently difficult to audit and manage: How many applications and systems is a person authorized to access? How big is the company? How many systems does it possess? If the authorization scheme is host-based, can one ever be sure of the complete scope of an individual's authorizations? And in this fragmented host-based authentication and authorization environment, how many lognames and passwords must an individual remember? Single sign-on is an attractive target functionality.

In the context of a large, multiple application, multiple host network computing environment, an integrated, centralized, auditable, manageable authorization database is clearly preferrable to the fragmented, disintegrated host-based alternative.

These considerations provide the motivation for the design of hyperDRIVE. The targeted functionality is a consistent, integrated, auditable, manageable RBAC implementation which is employed by many servers, clients, and applications, scalable to support thousands of clients and hundreds of servers and applications .

A Directory-enabled "late" or "run-time" binding of clients to services dramatically improves the flexibility of the distributed object computing environment. Service providers can offer, withdraw, change location, and change the operational characteristics of service implementations as necessary, confident that their clients will be able to dynamically react to the change, via Directory information. Client implementors are likewise freed of previous requirements to know in advance and for all time the exact location and operational parameters of the services they require.

A least-privilege navigation guide service emerges conveniently from

the infrastructure described here. The guide can be a navigational tool for people. The guide appears to the user as a menu. The contents and targets of the menu are services for which the user is authorized via hyperDRIVE's RBAC Role mappings to services.

Note that the guide itself is not an authorization mechanism.

At run-time, a service which wishes to protect itself from unauthorized access will require authentication of the client principal's identity. This authenticated identity binds the client principal to an object in the Directory. The service is also capable of asserting and authenticating its own identity, which is also bound to a Directory object.

A third entity, a policy decision unit (PDU), is familiar with the hyperDRIVE RBAC Role composite object. The PDU is capable of navigating the composite graph from edge to edge. The PDU may even internalize and cache the hyperDRIVE RBAC Role composite object on some regular cycle as a performance enhancement.

To determine whether a candidate client is authorized to access the service, the service sends a message to the PDU. The message contains the Directory distinguishedName of the client and the Directory distinguishedName of the service. The PDU uses these two data, as compared to the Role object, to determine whether the client is mapped to the service. The PDU returns a simple "yes" or "no" answer to the service. The service grants or denies access, based upon the authoritative answer returned by the PDU.

Alternatively, the service could serve as its own PDU, navigating the Role itself.

Both the Navigation Guide and the PDU services could be implemented as methods or services of a Compositron, or a dynamically linked set of Compositrons, which is/are are referenced from the "head" or "root" of the hyperDRIVE RBAC Role composite object.

Figure 4, following, illustrates an object association which relates clients to the services they are authorized (by an RBAC policy) to access.

Compare and contrast this composite object construct with the objectclasses and attributes which were defined in [26], [27].

The objectclasses and attributes of [26], [27] were specifically created to serve a specific purpose. They fulfilled their responsibilities in that limited information domain. It worked.

But those objectclasses could not be repurposed, could not be reused in another logical realm. Even worse, the client code which was written specifically to interpret those objects, interpret their design-implied relationships, and behave accordingly, could not be reused or repurposed.

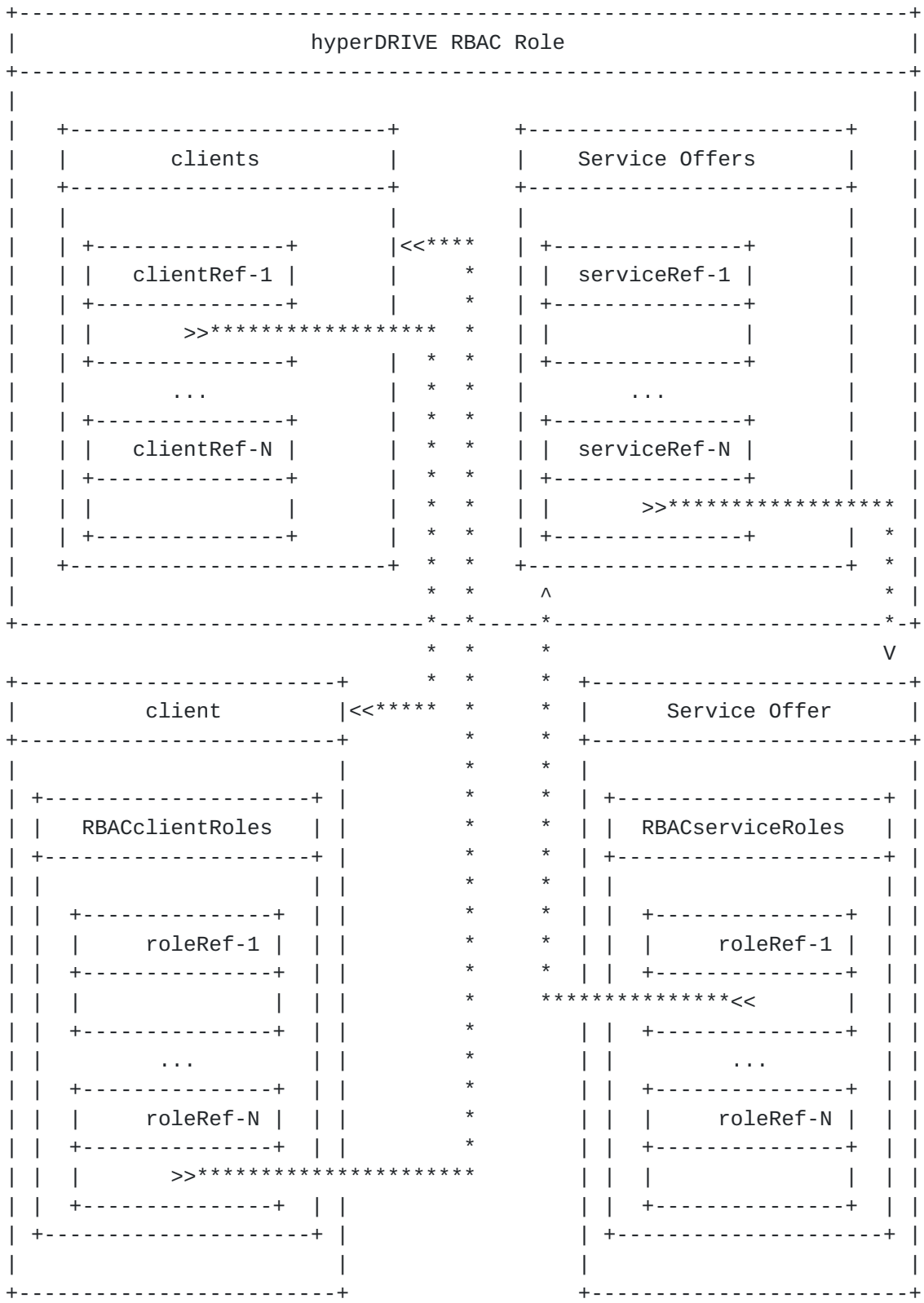
The original hyperDRIVE was not designed for reuse. Bad.

The hyperDRIVE RBAC Role composite object of Figure 4 is the same basic construct as described elsewhere in this document. See Figure 2. This is a reuse, a specific use, of a design which is intended for reuse.

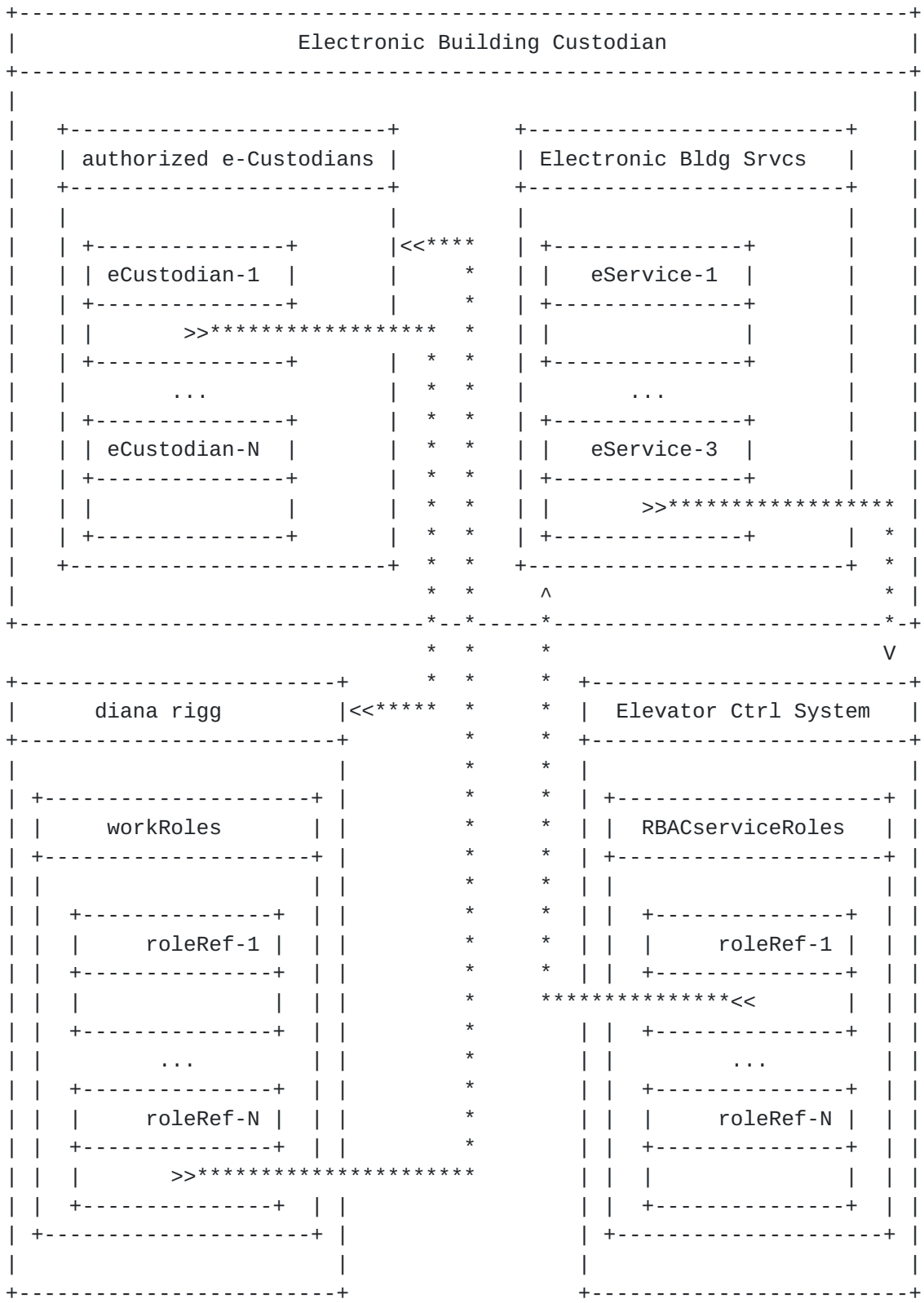
Client and server code which is written to use COD's composite objects and object associations can simply be tweaked, extended slightly, to be reused in hyperDRIVE's Compositronic Navigation Guide and Compositronic PDU.

Figure 5 illustrates an instance of hyperDRIVE RBAC. An LDIF representation of Figure 5 follows the figure.

F.1. Figure 4
hyperDRIVE RBAC



F.2. Figure 5
hyperDRIVE RBAC Example



F.3. LDIF representation of Figure 5

In order to provide a concrete example, some names of the figure's objects are given specific instance names:

- "Electronic Building Custodian" is an implementation instance of "hyperDRIVE RBAC Role"
- "authorized e-Custodians" is an implementation instance of "clients"
- "Electronic Building Services" is an implementation instance of "service Offers"

People assigned the "Electronic Building Custodian" role are authorized to use services such as "Heating-Ventilation-Air-Conditioning System", "Elevator Control System", and "Smoke-Fire-Intrusion Alarm System".

The example is completely described and implemented in terms of the objectclasses and attributes contained in COD. There is no requirement to extend COD for objectclasses such as "hyperDRIVE RBAC Role", "clients", or "Service Offers".

F.3.1. the "Electronic Building Custodian" "hyperDRIVE RBAC Role" relationship

The "Electronic Building Custodian" object is an instance of objectAssociation.

```
version: 1
dn: compositeElementName=Electronic Building Custodian, ou=RBAC Roles,
 dc=electronicRealtyManagement, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
objectclass: typedObjectContainerCompositeObject
objectclass: objectAssociation
compositeElementName: Electronic Building Custodian
description: in your spare time, from the comfort of your own home!
infoType: hyperDRIVE
infoType: hyperDRIVE RBAC Role
targetType: objectAssociationEnd
targetInfoType: hyperDRIVE
targetInfoType: hyperDRIVE RBAC clients
targetInfoType: hyperDRIVE RBAC service offers
degree: 2
```

F.3.2. relationship roles

In the "authorized e-Custodians" (clients) and "Electronic Building Services" (service offers) roles of the "Electronic Building Custodian" "hyperDRIVE RBAC Role" relationship, the "clients" and "service offers" objects are instances of objectAssociationEnd.

F.3.2.1. container of references to clients

```
dn: compositeElementName=authorized e-Custodians,
 compositeElementName=Electronic Building Custodian, ou=RBAC Roles,
 dc=electronicRealtyManagement, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
objectclass: typedObjectContainerCompositeObject
objectclass: objectAssociationEnd
compositeElementName: authorized e-Custodians
description: references to people who are authorized e-Custodians
infoType: hyperDRIVE
infoType: hyperDRIVE RBAC clients
infoType: workRole
targetType: associationEndNodeRef
targetInfoType: e-CustodianReference
minInstanceMultiplicity: 1
maxInstanceMultiplicity: 6000
```

F.3.2.2. container of references to services

```
dn: compositeElementName=Electronic Building Services,
 compositeElementName=Electronic Building Custodian, ou=RBAC Roles,
 dc=electronicRealtyManagement, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
objectclass: typedObjectContainerCompositeObject
objectclass: objectAssociationEnd
compositeElementName: Electronic Building Services
description: references to service offers
infoType: hyperDRIVE
infoType: hyperDRIVE RBAC service offers
targetType: associationEndNodeRef
targetInfoType: e-BuildingServiceReference
minInstanceMultiplicity: 1
```

F.3.3. references to actors of the relationship roles

The references to the actors are instances of associationEndNodeRef.

F.3.3.1. client target references

```
# one of many subordinates of "authorized e-Custodians"
dn: compositeElementName=eCustodian-1,
 compositeElementName=authorized e-Custodians,
 compositeElementName=Electronic Building Custodian, ou=RBAC Roles,
 dc=electronicRealtyManagement, dc=com
objectclass: top
objectclass: compositeElement
objectclass: typedObjRefCompositeElement
objectclass: associationEndNodeRef
compositeElementName: eCustodian-1
description: reference to a person who can be an e-Custodian
infoType: hyperDRIVE
infoType: hyperDRIVE RBAC client reference
infoType: e-CustodianReference
targetType: person
targetInfoType: e-Custodian
target: cn=diana rigg, ou=people, dc=someOtherISP, dc=com
order: 1
```

F.3.3.2. service target references

```
# one of several subordinates of "Electronic Building Services"
dn: compositeElementName=eService-3,
 compositeElementName=Electronic Building Services,
 compositeElementName=Electronic Building Custodian, ou=RBAC Roles,
 dc=electronicRealtyManagement, dc=com
objectclass: top
objectclass: compositeElement
objectclass: typedObjRefCompositeElement
objectclass: associationEndNodeRef
compositeElementName: eService-3
description: reference to an e-BuildingService offer
infoType: hyperDRIVE
infoType: hyperDRIVE RBAC serviceOffer reference
infoType: e-BuildingServiceReference
targetType: labeledURIobject
targetInfoType: hyperDRIVEserviceOffer
infoType: e-BuildingService
target: compositeElementName=Elevator Control System, ou=Service Offers,
 dc=electronicRealtyManagement, dc=com
order: 3
```

F.3.4. targets - actors in the relationship

These are the Directory entries for the clients and services which participate in the hyperDRIVE RBAC relationship.

F.3.4.1. clients

See 2.2.6.2.4.1. for the "diana rigg" object and subordinates.

F.3.4.2. services

The extra-Directory service is represented here in the Directory. The structure of the service's relationship to the RBAC association is similar to the structure of a client's relationship to the RBAC association.

F.3.4.2.1. serviceOffer

```
# one of several subordinates of "Service Offers"
dn: compositeElementName=Elevator Control System, ou=Service Offers,
 dc=electronicRealtyManagement, dc=com
objectclass: top
objectclass: compositeObject
objectclass: labeledURIobject
compositeElementName: Elevator Control System
description: Elevator Control System
infoType: hyperDRIVEserviceOffer
infoType: e-BuildingService
# SSL-enabled service will challenge user for X.509 certificate
labeledURI:https://hyperDRIVEnSystems.electronicRealtyManagement.com/
 serviceOffers/elevatorControlSystem.html

# container for references to RBAC service roles
dn: compositeElementName=RBACserviceRoles,
 compositeElementName=Elevator Control System, ou=Service Offers,
 dc=electronicRealtyManagement, dc=com
objectclass: top
objectclass: compositeElement
objectclass: compositeObject
objectclass: typedObjectContainerCompositeObject
objectclass: objectAssociationRole
compositeElementName: RBAC service roles
description: this is my work
infoType: workRoles
targetType: associationRoleRef
targetInfoType: workRoleRef

# references to RBAC service roles

# reference to service role in Electronic Building Services
```

```
dn: compositeElementName=eCustodianAccess,
 compositeElementName=RBACserviceRoles,
 compositeElementName=Elevator Control System, ou=Service Offers,
 dc=electronicRealtyManagement, dc=com
objectclass: top
objectclass: compositeElement
objectclass: typedObjRefCompositeElement
objectclass: associationRoleRef
compositeElementName: eCustodianAccess
description: this is how we are related to eCustodians
infoType: workRoleRef
targetType: objectAssociationEnd
targetInfoType: hyperDRIVE RBAC service offers
target: compositeElementName=Electronic Building Services,
 compositeElementName=Electronic Building Custodian, ou=RBAC Roles,
 dc=electronicRealtyManagement, dc=com
```

8. Full Copyright Statement

this Internet Draft Expires April, 2000