Towards Relationship-Based Navigation

Subodh Bapat

BlacTie Systems Consulting

16441 Blatt Blvd., Suite 203, Ft. Lauderdale, Florida 33326, USA

Phone: +1 305 389 8347

Email: bapat@gate.net

Abstract

This paper builds upon the OSI General Relationship Model and presents mechanisms to perform relationship-based navigation among the managed object classes of the OSI Management Information Model. Examples demonstrate how such relationship-based navigation through the semantic network can permit extended reasoning and inferencing during network management.

1 INTRODUCTION

In the OSI General Relationship Model (GRM) [X.725], a relationship among two or more managed object classes is specified using a *managed relationship class*. A managed relationship class describes the characteristics of the managed relationship independent of the actual classes that may participate in that relationship. Such characteristics include the roles which its participant managed objects play, the cardinalities with which they participate in the relationship, the behavior of the relationship, and any additional constraints and dependencies that may govern the participation of managed objects in that relationship.

The participation of a specific set of managed object classes in a managed relationship is described in a *role binding*. A role binding asserts that a particular managed relationship holds between particular managed object classes, and also indicates the roles played by each participant managed object class in the relationship. A role binding also specifies additional behavior, constraints on roles, and the conditions under which participant managed objects can enter and exit the managed relationship.

The same relationship class may be used in different role bindings to bind different groups of managed object classes in relationships. For example, the roles of a backup relationship — such as *backs-up* and *is-backed-up-by* — may be defined in the *backup* managed relationship class, independent of the managed object classes that participate in such relationships. Once this relationship class is established, one role binding may bind the *dialUpCircuit* managed object class in the *backs-up* role with the *dedicatedCircuit* managed object class in the *is-backed-up-by* role (indicating that a dial-up circuit may back up a dedicated circuit, as is typical in private data networks), while another role binding may bind the *serviceControlPoint* managed object class in the *backs-up* role with the *adjunctProcessor* managed object class in the *is-backed-up-by* role (indicating that an SCP may back up an Adjunct Processor, as is typical in Intelligent Network architectures).

Although relationships are formally specified using the template notation of the General Relationship Model, it is often helpful for comprehension to depict them graphically as well. We

depict them using *extended Kilov diagrams*. In a Kilov diagram [Kilo94a, Kilo94b], the relationship construct is indicated in a rectangle, as are the participating object classes; the triangle construct “Rel” between them indicates that the classes participate in the indicated relationship. In this paper, we represent each role binding with a Kilov diagram, extending it to also depict the roles which the participant classes play with each other.

Figure 1. Extended Kilov Diagrams for Role Bindings of the Backup Relationship Class.

The formal specification of managed relationship classes and role bindings is performed using templates defined for that purpose. For example, the backup managed relationship class (simplified for the purposes of this paper) may be defined as

```
backup RELATIONSHIP CLASS
  BEHAVIOUR backupBehaviour BEHAVIOUR DEFINED AS
    "Backup object assumes failover operation for backed-up object"
  ROLE backs-up REGISTERED AS {...}
  ROLE is-backed-up-by REGISTERED AS {...}
  REGISTERED AS {...};
```

Different role bindings may now be established for this relationship class. As one example, assuming that dialUpCircuit and dedicatedCircuit are both managed object classes defined and registered elsewhere in their own MANAGED OBJECT CLASS templates, the following role binding establishes the required backup relationship:

```
dialUpCkt-backsUp-dedicatedCkt ROLE BINDING
RELATIONSHIP CLASS backup
  BEHAVIOUR circuitBackupBehaviour BEHAVIOUR DEFINED AS
    "dialUpCircuit assumes failover operation for dedicatedCircuit"
  ROLE backs-up RELATED CLASSES dialUpCircuit AND SUBCLASSES
  ROLE is-backed-up-by RELATED CLASSES dedicatedCircuit AND SUBCLASSES
  REGISTERED AS {...};
```

The same relationship may be established between other managed object classes using other role bindings. The templates above have been intentionally simplified to keep the focus on roles played participant object classes; role cardinality constraints, for example, have not been specified. The complete RELATIONSHIP CLASS template and ROLE BINDING template define many additional characteristics of a relationship. The RELATIONSHIP CLASS template specifies the *constraints* which must be satisfied by managed object instances in order to be participants in the relationship. It also specifies various *dependencies* which describe how the participation of a managed object instance in the relationship is influenced by its participation in other relationships. It specifies *relationship management operations* that may be performed, e.g. relationship establishment, binding, querying, notification, unbinding, and termination. The managed relationship class template also specifies the conditions governing the *dynamic entry* and *dynamic departure* of a participant managed instance in an established relationship.

Aside from binding managed managed object classes is a relationship, the ROLE BINDING template also specifies various ways of representing the relationship. For example, a relationship

may be represented by a separate *relationship object* (an instance of the relationship class) whose attributes indicate the names of the participating managed objects. Such an implementation is typical for relationships having a many-to-many role cardinality. A relationship may alternatively be represented by “pointer attributes” within each participant managed instance, whose value indicates the other managed object instance(s) to which that object is currently bound. An implementation using such conjugate pointers is typical for many relationships having a one-to-one role cardinality. A ROLE BINDING template also specifies an *operations mapping*, which indicates how relationship management operations map to ordinary systems management operations on managed object classes. For example, in a conjugate pointer implementation, the relationship management operation to unbind the relationship may simply map to the systems management operation of setting the values of the conjugate pointer attributes in the participant managed objects to null.

Although all these aspects are important for the complete specification of a relationship, we will not concentrate on them in this paper, as our focus is on the semantics of *relationship-based navigation*. For simplicity, our examples will omit those clauses of the RELATIONSHIP CLASS and ROLE BINDING templates which are not relevant to semantic extensions we propose; it ought to be borne in mind, however, that a complete (compilable) specification of a relationship should include all the required template clauses.

In this paper, we introduce new concepts in the modeling of relationships by exploiting special properties of the *roles* of a relationship class. By defining operations on roles, we can enhance the GRM to include several semantically useful concepts. These concepts allow us to express extended relationships within our model precisely and succinctly.

2 CONCEPTUAL BACKGROUND

A *virtual relationship* is a relationship whose existence can be inferred from other relationships [Bapa94]. A virtual relationship is not created by relationship establishment; it is dynamically computed and resolved within the management information repository from existing established relationships. The supporting relationships which give rise to a virtual relationship are termed *base relationships*. A virtual relationship implicitly arises when the roles of its base relationships have certain special *properties*.

We define an *actual relationship* as a relationship which cannot be inferred from the properties of roles of other relationships, and therefore must be explicitly created by the architect using a ROLE BINDING template. The base relationships which give rise to a virtual relationship may be actual relationships, or may themselves be virtual [Bapa93b].

A *virtual relationship instance* is formed by the set of object instances which participate in the virtual relationship. A virtual relationship does not make existing objects participants in a new relationship. Rather, objects which are already participants in *actual relationship instances* become automatic participants in virtual relationship instances, because of the special properties of the roles they play in their actual relationships. Thus, although a virtual relationship may have instances, it can never be *imperatively established*; only actual relationships can.

Therefore, operations such as BIND, UNBIND, ESTABLISH and TERMINATE are illegal on a virtual relationship. A virtual relationship is *automatically* established and terminated as and when its supporting base relationships are established and terminated. Objects are *automatically* bound and unbound in a virtual relationship as and when they are bound and unbound in its supporting base relationships. Any change made to its supporting actual base relationships will be automatically reflected in the virtual relationship, since the virtual relationship instances are, in

effect, dynamically resolved from actual base relationship instances every time they are queried. As far as the user is concerned, the QUERY operation works exactly the same way on a virtual relationship as it does on an actual relationship.

3 PROPERTIES OF ROLES

A virtual relationship arises as a consequence of special *properties* possessed by the roles of its supporting base relationships. A property of a role is a shorthand mechanism for specification reuse, which allows us to define many extended relationships from a single construct. By indicating the properties a role possesses, we create a mechanism which captures within a single relationship class *more* semantics than just the usual association between participant object classes, their cardinalities, participation constraints, and roles. By specifying our knowledge of the special properties of roles in extension clauses of the RELATIONSHIP CLASS template, we can compile into our management information repository the ability to perform extended navigation through relationship semantics.

There are five important properties which the roles of a relationship class may possess:

- The *Commutativity* property;
- The *Transitivity* property;
- The *Distribution* property;
- The *Convolution* property; and
- The *Implication* property.

It is important to emphasize that these properties belong to the roles of relationship classes, and not to role bindings. Thus, if the roles of a relationship class possess these properties, they will be operative in all role bindings in which that relationship class is used.

4 COMMUTATIVE VIRTUAL RELATIONSHIPS

A *commutative virtual relationship* is a relationship which arises from the *commutativity* property of the roles in its base relationship class. We define commutative roles as follows:

*A pair of roles $\{r_1, r_2\}$ of a relationship class is said to be **commutative** if, given two managed object classes A and B, and a role binding in which A plays role r_1 with B and B plays role r_2 with A, then it can be inferred that B plays role r_1 with A and A plays role r_2 with B.*

This states that any role binding of a base relationship class with commutative roles automatically implies the existence of another role binding in which the roles are “flipped around”.

Many examples of commutative virtual relationships exist in network modeling. The most common examples of these are *connectivity* and *interconnectivity* relationships between instances of network devices. These relationships can be traversed from either direction in the semantic network: if an object a_0 is interconnected (perhaps via many intermediate nodes) to object b_0 , then the repository can infer (e.g. in a topology display application) that object b_0 is interconnected to object a_0 .

Another common example of commutative relationships are *mutual backup relationships*. It is important to understand that *all* backup relationships are not commutative. For example, the backup relationship class we defined earlier with the roles *backs-up* and *is-backed-up-by* did not have the commutativity property. Being non-commutative, this relationship is not mutual; it is only a one-way relationship. When we use this relationship to inform the repository

that managed object class A plays the role `backs-up` with managed object class B (or that B is-backed-up-by A), the repository *cannot* infer that B backs-up A (or its reciprocal, A is-backed-up-by B).

We might not wish to invest the roles `backs-up` and `is-backed-up-by` with the commutativity property, because in some contexts (as in the `dialupCkt-backsUp-dedicatedCkt` ROLE BINDING above) it may be used as a non-commutative (one-way) backup relationship. We specify a new relationship class — say, the `mutualBackup` relationship class. (Since relationship classes may derive from each other due to inheritance, it is possible for the `mutualBackup` relationship class to derive from the `backup` relationship class. This is omitted here for simplicity.) Assume that the `mutualBackup` relationship class has the roles `mbacks-up` and `is-mbacked-up-by`, standing for “mutually backs up” and “is mutually backed up by”. We invest *these* roles with the commutativity property. Thus, this relationship class carries more semantics than the one-way backup relationship class. (In a later section, we will see how the semantics of a single `mutualBackup` relationship can be made to imply the semantics of the one-way backup relationship in both directions.)

```
mutualBackup RELATIONSHIP CLASS
  ROLE mbacks-up COMMUTATIVE REGISTERED AS {...}
  ROLE is-mbacked-up-by COMMUTATIVE REGISTERED AS {...}
REGISTERED AS {...};
```

Consider an example of a `signalTransferPoint` object class, instances of which are generally deployed in “mated pairs” with each other in an Intelligent Network. By cross-linking pairs of `signalTransferPoint` objects, we provide redundancy in the signalling network. This may be specified as a relationship with `mutualBackup` roles. If so, it implies that for each relationship instance between pairs of `signalTransferPoint` objects, the repository can also infer a commutatively derived virtual relationship instance.

```
stp-mbacksUp-stp ROLE BINDING
RELATIONSHIP CLASS backup
  ROLE mbacks-up RELATED CLASSES signalTransferPoint AND SUBCLASSES
  ROLE is-mbacked-up-by
    RELATED CLASSES signalTransferPoint AND SUBCLASSES
REGISTERED AS {...};
```

5 TRANSITIVE VIRTUAL RELATIONSHIPS

A *transitive virtual relationship* is a relationship which arises from the *transitivity* property of the roles in its base relationship class. We define transitive roles as follows:

*A pair of roles $\{r1, r2\}$ of a relationship class is said to be **transitive** if, given the managed object classes A, B, and C, a first role binding in which A plays role $r1$ with B and B plays role $r2$ with A, and a second role binding in which B plays role $r1$ with C and C plays role $r2$ with B, then it can be inferred that A plays role $r1$ with C and C plays role $r2$ with A.*

This definition implies that relationship roles are transitive if, given a common “linking” participant, they can be “chained together”. Well-known examples of transitive relationships are interconnectivity relationships which are important for fault diagnostics and topology display purposes. For example, if we model the interconnectivity relationship class with the roles `interconnects-to` and `is-interconnected-to`, then we know that if A `interconnects-to` B and B `interconnects-to` C, it follows that A `interconnects-to` C.

There are other examples of transitive relationships in network modeling. Consider a network of electronic mail application processes, which exchange electronic mail messages among themselves over local and wide-area computer networks. These application processes may all have different implementations and protocols: some could be Message Transfer Agents, some could be mail handling demon processes, and so on. Although these mail handlers may use different standards, through the use of programs like `sendmail` or other electronic mail gateways with address translation mechanisms, they may all have the ability to forward mail to each other.

The `mailForwarding` relationship class may be modeled with the roles `{forwards-mail-to, receives-mail-from}`. Clearly, this relationship is commutative. We could further provide additional information about this relationship to the repository by specifying this relationship as being **transitive**. This implies that once we create any two different `mailForwarding` role bindings between any three different managed object classes (such as `x400mta`, `smtpDemon` and `uucpDemon`), we also automatically create a transitively derived virtual relationship instance between the third pair.

```
mailForwarding RELATIONSHIP CLASS
  ROLE forwards-mail-to COMMUTATIVE TRANSITIVE REGISTERED AS {...}
  ROLE receives-mail-from COMMUTATIVE TRANSITIVE REGISTERED AS {...}
REGISTERED AS {...};

x400-forwardsto-smtp ROLE BINDING
  RELATIONSHIP CLASS mailForwarding
  ROLE forwards-mail-to RELATED CLASSES x400mta AND SUBCLASSES
  ROLE receives-mail-from RELATED CLASSES smtpDemon AND SUBCLASSES
REGISTERED AS {...};

smtp-forwardsto-uucp ROLE BINDING
  RELATIONSHIP CLASS mailForwarding
  ROLE forwards-mail-to RELATED CLASSES smtpDemon AND SUBCLASSES
  ROLE receives-mail-from RELATED CLASSES uucpDemon AND SUBCLASSES
REGISTERED AS {...};
```

Because the `mailForwarding` relationship class has transitive roles, given the role bindings above the repository can automatically infer a role binding for the `mailForwarding` relationship between `x400mta` and `uucpDemon`, even though such a role binding has not been explicitly specified in the information model.

Figure 3. A Transitive Virtual Relationship.

6 DISTRIBUTIVE VIRTUAL RELATIONSHIPS

A *distributive virtual relationship* is a relationship which arises from the *distribution* property of the roles in its base relationship class. We define distributive roles as follows:

*A pair of roles {r1, r2} of a relationship class is said to **distribute over** another pair of roles {r3, r4} of another relationship class if, given the managed object classes A, B, and C, a first role binding in which A plays role r1 with B and B plays role r2 with A, and a second role binding in which B plays role r3 with C and C plays role r4 with B, then it can be inferred that A plays role r1 with C and C plays role r2 with A. The role r1 is said to **distribute ahead** of role r3, and the role r2 is said to **distribute behind** role r4.*

This definition states that, given a common “linking” participant object class, the *distributing* roles {r1, r2} *distribute over* the *distributand* roles {r3, r4}. That is, the roles {r1, r2} can be virtually established between A and C.

As an example, suppose we wish to model geographic information about equipment objects. We model this as a housing relationship between the location object class and the equipment object class, in which location plays the role houses with equipment, which reciprocates by playing is-housed-at with location. (The class location is modeled as a separate class because it could have its own attributes, such as *streetAddress*, *telephoneNumber*, and so on.)

There may also be a termination relationship between the equipment object class and the circuit object class, in which equipment plays the role terminates with circuit, which plays the role is-terminated-at with equipment. Occasionally, rather than knowing which equipment terminates a given circuit, it may be useful for certain outside-plant engineers to know the physical address or location where the circuit is-terminated-at. Ordinarily, we would have to perform two queries for this information: one to determine which instance of equipment the circuit is-terminated-at, and another to determine which instance of location that equipment is-housed-at.

To make this more concise, we may simply say that the roles of the termination relationship class *distribute over* the roles of the housing relationship class. More specifically, this means that the is-terminated-at role distributes *ahead* of the is-housed-at role. This creates a *distributive virtual relationship* between circuit and location: thus, if instance c0 of circuit is-terminated-at instance d0 of equipment, and instance d0 of equipment is-housed-at instance l0 of location, we may infer (e.g., for topology display purposes) that circuit c0 is-terminated-at location l0.

The same semantics could be equivalently stated in terms of reciprocal roles. We could say that the *terminates* role *distributes behind* the *houses* role. This means that if some equipment terminates a circuit, and some location houses that equipment, then that location also terminates the circuit.

```
housing RELATIONSHIP CLASS
  ROLE is-housed-at REGISTERED AS {...}
  ROLE houses REGISTERED AS {...}
REGISTERED AS {...};

termination RELATIONSHIP CLASS
  ROLE is-terminated-at DISTRIBUTES AHEAD OF is-housed-at
  REGISTERED AS {...}
  ROLE terminates DISTRIBUTES BEHIND houses REGISTERED AS {...}
REGISTERED AS {...};

equipment-terminates-circuit ROLE BINDING
  RELATIONSHIP CLASS termination
  ROLE is-terminated-at RELATED CLASSES circuit AND SUBCLASSES
  ROLE terminates RELATED CLASSES equipment AND SUBCLASSES
REGISTERED AS {...};

location-houses-equipment ROLE BINDING
  RELATIONSHIP CLASS housing
  ROLE is-housed-at RELATED CLASSES equipment AND SUBCLASSES
  ROLE houses RELATED CLASSES location AND SUBCLASSES
REGISTERED AS {...};
```

Given the definitions above, the repository can automatically infer the existence of a role binding of the *termination* relationship class between *circuit* and *location*, even though such a role binding is not explicitly created by the architect. In general, relationships may distribute over base relationships regardless of whether the base relationships are actual or virtual.

Since a virtual relationship instance may be queried exactly like an actual relationship instance, this implies that if we queried an instance of a *circuit* for the *location* where it terminated (that is, we tracked the *location* object to which it is bound via its *is-terminated-at* role) we would directly get the correct instance of *location*, without having to compose any relational joins in our query to include the intermediate *equipment* object class. Under conventional modeling, some form of a join condition between entities would be required in the query in order to elicit the desired response — even if the implementation platform for the management information repository is not relational.

A little reflection indicates that a transitive virtual relationship is a special case of a distributive virtual relationship in which both the distributing and distributand roles are the same.

7 CONVOLUTE VIRTUAL RELATIONSHIPS

A *convolute virtual relationship* is a relationship which arises from the *convolution* property of the roles in its base relationship class. We define convolute roles as follows:

*A pair of roles $\{r_1, r_2\}$ is said to **convolute from** the roles $\{r_3, r_4\}$ and $\{r_5, r_6\}$ if, given three participant managed object classes A, B, and C, a first role binding in which A plays role r_3 with B and B plays role r_4 with A, and a second role binding in which B plays role r_5 with C and C plays role r_6 with B, then it can be inferred that A plays role r_1 with C and C plays role r_2 with A. The role r_1 is said to **convolute above** the roles r_3 and r_5 , and the role r_2 is said to **convolute below** the roles r_4 and r_6 .*

This definition states that, under certain circumstances, two role bindings of separate relationship classes may give rise to a virtual relationship with an entirely different pair of roles.

Consider an example in which an information service provider operates a data network which supplies multiple information services, such as stock and bond price quotations, to multiple clients. Clients such as brokerage firms subscribe to these information services. The terms of a typical subscription require the information service to automatically provide the information of interest to the client. The quotation service downloads this information to the trading computers of all employees of the corporate client. Because the information of interest is dynamic and useful only within a small life span, each trading computer downloads this information directly from the service provider, rather than from any internal central redistribution database owned and operated by the client company itself. If the client adds more trading computers, they too will receive information directly from the service provider.

We model the subscription relationship class with the roles *subscribes-to* and *is-subscribed-to-by*. Thus, we might model a role binding in which a *quotationService* object class *is-subscribed-to-by* a *brokerageFirm* object class. Further, we might also model a *tradingComputer* object class which participates in an ownership relationship with the *brokerageFirm* class. The *brokerageFirm* class plays the role *owns* with the *tradingComputer* class, which reciprocates with the role *is-owned-by*. We also know that the *quotationService* class must play a *downloads-to* role with respect to the *tradingComputer* class. This would be a download relationship, in which the *tradingComputer* reciprocates with the role *downloads-from*.

Figure 5. A Conventional Relationship Model among Three Participant Classes.

In this situation, each time the *brokerageFirm* adds a new *tradingComputer*, we must create a new instance of the ownership relationship between these two classes. Ordinarily, we must also establish a separate instance of the download relationship between the *quotationService* object class and the same instance of *tradingComputer*. We will always have to ensure that all such parallel relationships are consistently maintained. Every time the *brokerageFirm* decommissions or scraps a *tradingComputer*, we must delete the own-

ership relationship instance with that tradingComputer. We must then ensure that we also delete the corresponding instance of the download relationship between the quotationService object and the same tradingComputer. We would have to maintain this consistency using some mechanism external to the relationships, since we have no mechanism *within* the relationships to *automatically shadow* the changes of one set of relationship instances in another.

We can eliminate this problem entirely by defining the download relationship to be a *convolute virtual relationship* which *convolutes from* the subscription and ownership relationships. The role downloads-to *convolutes above* the first role is-subscribed-to-by played by quotationService with brokerageFirm and the second role owns played by brokerageFirm with tradingComputer. The reciprocal role downloads-from *convolutes below* the two roles subscribes-to and is-owned-by. With this specification, a virtual relationship instance of the download relationship is automatically created or destroyed every time an actual relationship instance of the ownership relationship is created or destroyed. (Or, several instances of the download relationship are automatically created or destroyed each time a subscription relationship is created or destroyed.)

```
subscription RELATIONSHIP CLASS
  ROLE subscribes-to REGISTERED AS {...}
  ROLE is-subscribed-to-by REGISTERED AS {...}
REGISTERED AS {...};

ownership RELATIONSHIP CLASS
  ROLE owns REGISTERED AS {...}
  ROLE is-owned-by REGISTERED AS {...}
REGISTERED AS {...};

download RELATIONSHIP CLASS
  ROLE downloads-to CONVOLUTES ABOVE is-subscribed-to-by AND owns
    REGISTERED AS {...}
  ROLE downloads-from CONVOLUTES BELOW subscribes-to AND is-owned-by
    REGISTERED AS {...}
REGISTERED AS {...};

firm-subscribes-to-infoService ROLE BINDING
  RELATIONSHIP CLASS subscription
  ROLE is-subscribed-to-by
    RELATED CLASSES quotationService AND SUBCLASSES
  ROLE subscribes-to RELATED CLASSES brokerageFirm AND SUBCLASSES
REGISTERED AS {...};

firm-owns-computer ROLE BINDING
  RELATIONSHIP CLASS ownership
  ROLE owns RELATED CLASSES brokerageFirm AND SUBCLASSES
  ROLE is-owned-by RELATED CLASSES tradingComputer AND SUBCLASSES
REGISTERED AS {...};
```

Given the definitions above, the repository can automatically infer the existence of a role binding of the download relationship class between quotationService and tradingComputer, even though such a role binding is not explicitly created by the architect.

A little reflection indicates that a distributive virtual relationship is a special case of a convolute virtual relationship in which the convolute virtual roles are the same as the base distributing roles.

8 IMPLICATE VIRTUAL RELATIONSHIPS

An *implicate virtual relationship* is a relationship which arises from the *implication* property of the roles in its base relationship. We define implicate roles as follows:

*A pair of roles $\{r1, r2\}$ of a relationship class is said to **implicate** another pair of roles $\{r3, r4\}$ if, given two participant managed object classes A and B, and a role binding in which A plays role $r1$ with B and B plays role $r2$ with A, it can be inferred that A plays $r3$ with B and B plays role $r4$ with A.*

This definition states that a role binding with the base roles automatically *implicates* (implies) the existence of a virtual relationship between the same participant managed objects with different roles.

Implication is actually one of the most general forms of virtual relationships. It is important to note the details of its definition: the definition does not require that the implicate roles have the same *properties* as the base roles. It is possible that the base roles have properties such as commutativity, transitivity, and so on, which do not necessarily carry over to the implicate roles. It is also possible that the implicate roles possess properties which the base roles do not. Generally, the implicate virtual relationship conveys different semantics than its base relationship. A few examples will clarify this point.

An implicate virtual relationship arises when the base relationship acts as a “stronger” expression or shorthand for a collection of several “looser” relationships. In general, implicate virtual relationships arise because they indicate consequential roles which exist because the base roles exist. As a general guideline, if consequential roles arise from a single base relationship, they are specified as an implicate virtual relationship. If consequential roles arise from two base relationships with a common “linking” participant, they are specified as a transitive, distributive, or convolute virtual relationship.

Consider the backup relationship class whose roles are backs-up and is-backed-up-by. These roles are not *transitive*: if in one role binding object class A backs-up object class B and in another role binding object class B backs-up object class C, it is not true that object class A backs-up object class C. This is because if an instance of C fails, it is not true that an instance of A will take over.

However, these roles give rise to the implicate roles of the pointOfFailure relationship class, i.e. {is-a-point-of-failure-for, has-point-of-failure}. Thus if A backs-up B and B backs-up C, then A is-a-point-of-failure-for B and B is-a-point-of-failure-for C. These roles convey different semantics than the roles of the backup relationship class, and also have different properties: they *are* transitive. While it is not the case that A backs-up C, due to the transitivity of A is-a-point-of-failure-for B and B is-a-point-of-failure-for C, it is true that A is-a-point-of-failure-for C. Further, the implicate roles also have a *distribution* property which its base roles do not have: the role has-point-of-failure distributes ahead of the contains role which

component objects play with composite objects. If C has-point-of-failure A and A contains D, then it is true that C has-point-of-failure D. Thus, it is possible to have implicate virtual relationships with completely different properties than their supporting base relationships.

```

pointOfFailure RELATIONSHIP CLASS
  ROLE has-point-of-failure TRANSITIVE
    DISTRIBUTES AHEAD OF contains REGISTERED AS {...}
  ROLE is-a-point-of-failure-for TRANSITIVE
    DISTRIBUTES BEHIND is-contained-in REGISTERED AS {...}
REGISTERED AS {...};

backup RELATIONSHIP CLASS
  ROLE backs-up IMPLICATES is-a-point-of-failure-for REGISTERED AS {...}
  ROLE is-backed-up-by IMPLICATES has-point-of-failure REGISTERED AS {...}
REGISTERED AS {...};

```

We established earlier a role binding for the backup relationship class between the managed object classes dialUpCircuit and dedicatedCircuit. The repository can now automatically infer the existence of an implicate pointOfFailure role binding between the same two classes.

Implicate virtual relationships are a powerful mechanism to capture extended semantics in a concise manner. In the example above, we create only the three role bindings for the actual relationships A backs-up B, A contains D, and B backs-up C. Because the backup relationship gives rise to implicate virtual relationships which have a distribution property, the three actual relationships generate four virtual relationships, as shown below.

Figure 7. Some Implicate Virtual Relationships.

If we query an instance of C for all its points of failure (that is, all its related objects via the has-point-of-failure role) the response will include the instances of B, A, and D. In fact, due to the transitivity of containment, the transitivity of pointOfFailure, and the distribution of pointOfFailure over containment, the response will include *all* component objects of A, *all* component objects of B, the transitive closure of A's has-point-of-failure role (that is, all objects which may back-up A, their back-ups, and so on) and all *their* components as well. By simply specifying the correct properties for relationship roles, we can equip network management applications with the power to navigate through an extensive semantic network in our management information repository.

Implicate virtual relationships are sometimes used to “break down” commutative relationships into two one-way relationships where necessary. For example, the roles of the `mutualBackup` relationship class can be broken down into the roles of two one-way backup role bindings. This can be accomplished by specifying that the roles {`mbacks-up`, `is-mbacked-up-by`} implicate *both* the one-way role pairs {`backs-up`, `is-backed-up-by`} and {`is-backed-up-by`, `backs-up`}:

```
mutualBackup RELATIONSHIP CLASS
  ROLE mbacks-up COMMUTATIVE
    IMPLICATES backs-up, is-backed-up-by REGISTERED AS {...}
  ROLE is-mbacked-up-by COMMUTATIVE
    IMPLICATES is-backed-up-by, backs-up REGISTERED AS {...}
REGISTERED AS {...};
```

9 CONCLUSION

Virtual relationships provide an effective mechanism for extending relationship semantics. Because of their ability to automatically shadow the changes of one set of relationship instances in another, they reduce the potential for inconsistency. If an object is virtually bound to another via a chain of supporting actual relationships, we can query the object for its virtually bound object exactly as we query it for an actually bound object. The run-time environment in the repository internally and transparently resolves the virtual relationship in terms of its chain of supporting actual relationships. This eliminates the need for us to compose any relational joins in our query, which otherwise can be quite complex. Consequently, virtual relationships considerably enhance the semantic richness of our model [Bapa93a].

It is important to remember that virtual relationships arise as properties of the roles of a relationship class, and not in role bindings. All the properties of the roles of a relationship class continue to hold in *every* role binding of that relationship class. A role binding cannot choose to “drop” certain properties of roles of its relationship class, nor can it invest those roles with new properties which hold only in that particular role binding.

We present below a concise summary of the types of virtual relationship we have defined using informal logical expressions. In these expressions, *A*, *B*, and *C* are managed object classes, and *r*, *s*, and *t* are roles of relationship classes. The construct *r* (*A*, *B*) is interpreted as a role binding, and is read as “*r* is the role played by *A* with *B*”. If “*∧*” is read as “*and*” and “*→*” is read as “*gives rise to*”, then

- *Commutative Virtual Relationship*: $r(A, B) \rightarrow r(B, A)$
- *Transitive Virtual Relationship*: $r(A, B) \wedge r(B, C) \rightarrow r(A, C)$
- *Distributive Virtual Relationship*: $r(A, B) \wedge s(B, C) \rightarrow r(A, C)$
- *Convolute Virtual Relationship*: $r(A, B) \wedge s(B, C) \rightarrow t(A, C)$
- *Implicate Virtual Relationship*: $r(A, B) \rightarrow s(A, B)$

Virtual relationships provide us with a robust mechanism to enforce consistency between a chain of links in a semantic network of objects. The presence of virtual relationships enables us to drop certain constraints which would otherwise be imposed across the semantic network.

For example, a requirement which traverses many links in the semantic network, such as: “*The operator responsible for addressing an alarm generated by a network device must be an employee of the outsourcing vendor who administers the location which houses that network device*” is normally specified in most systems of knowledge as a sequence of multiple *constraints*.

These constraints are explicit: the user must specify how to enforce them by equating values of identifier attributes of *pairs of objects* across binary links in the semantic network.

A role binding asserts a relationship between instances of object classes as a statement of a fact, just as an attribute value assertion is a statement of a fact. By a logical conjunction of such facts, we can *infer* the existence of other facts across *multiple* links in the semantic network. By specifying virtual relationships between objects such as `operator`, `alarm`, `equipment`, `outsourcingVendor`, and `location` extending over actual roles such as `is-responsible-for`, `is-generated-by`, `is-employed-by`, `administers`, and `houses`, the semantic constraint above falls out automatically and does not have to be explicitly specified. Because virtual relationships automatically reflect the changes of one relationship in another, they provide us with the ability to extend the “reach” of nodes in the semantic network to nodes other than their immediate neighbors. As such, they are powerful mechanism to facilitate extended navigation, reasoning, and inferencing within the management information repository.

10 REFERENCES

- [Bapa93a] Bapat, Subodh, “Richer Modeling Semantics for Management Information”, *Integrated Network Management III: Proceedings of the 1993 IFIP International Symposium on Integrated Network Management*, pp. 15-28.
- [Bapa93b] Bapat, Subodh, “Towards Richer Relationship Modeling Semantics”, *IEEE Journal on Selected Areas in Communications*, 11(9), Dec. 1993, pp. 1373 - 1384.
- [Bapa94] Bapat, Subodh, *Object-Oriented Networks: Models for Architecture, Operations, and Management*, Prentice-Hall, 1994.
- [Kilo94a] Kilov, Haim, and James Ross, *Information Modeling: An Object-Oriented Approach*, Prentice-Hall, 1994.
- [Kilo94b] Kilov, Haim, “Generic Concepts for Modeling Relationships”, *Proceedings of the IEEE Network Operations and Management Symposium (NOMS) 1994*.
- [X.725] “Information Technology - Open Systems Interconnection - Structure of Management Information - Part 7: General Relationship Model”, ITU-T Rec. X.725, 1994.

11 BIOGRAPHY

Subodh Bapat is Principal of BlacTie Systems Consulting, and has worked with several network equipment vendors and telecommunications carriers in the areas of applying object-oriented modeling techniques to network architecture, and to the development of network management software. As a lead architect and implementer of standards-based network management systems, he made leading contributions in the area of applying object-oriented techniques to the architecture of networking equipment and to information modeling for databases used in network management and operations support. His involvement extended over the complete product life-cycle, including the architecture, design, development, testing, and maintenance phases. Subodh is the author of “*Object-Oriented Networks: Models for Network Architecture, Operations and Management*,” (Prentice Hall, 1994, 757 pp.), a state-of-the-art book which demonstrates how the application of second-generation object-oriented modeling techniques can lead to sophisticated, intelligent, and highly automated network systems. He has published several articles in leading technical journals and has presented papers at major industry conferences. He has been awarded a number of patents in the area of implementing network management software.