


Donald G. Firesmith B. Henderson-Sellers

Upgrading OML to Version 1.1: Referential Relationships

In our last column, we investigated some new ideas for modeling specialized forms of associations (specifically aggregations, membership and containment). In this column (and the next) we expand on how these and other improvements have been incorporated into one specific OMG-compliant OO modeling language: the OPEN Modeling Language (OML). Wherever possible, OML attempts to adopt the OMG's constructs. However, OML favors the concepts underpinning a responsibility-driven approach rather than the more data-centric and use case-driven focus of UML. In addition, OML aims at having an intuitive notation that does not require modelers to memorize symbols by rote.

OML's version 1.0 meta-model clarifies a number of issues not previously understood and some that have been obfuscated.

The *OPEN Modeling Language (OML) Reference Manual*[1] documents OML version 1.0, which was summarized in previous *ROAD* columns.[2,3] In this column, we summarize the changes to version 1.0 that bring the language up to date and make it more OMG-compliant. At the same time, OML version 1.1 extends the ideas endorsed by the OMG, filling in gaps and clarifying semantics when appropriate (last month's column was one example of this). We also provide a formal addendum to the *OPEN Modeling Language (OML) Reference Manual* documenting the new version's view of referential relationships.

OML consists of a metamodel and a notation (called COMN), as does any modeling language. Some of the updates to OML amend the metamodel, some the notation. We will discuss here the two together, first informally and then formally. The main changes include a greater focus on responsibilities in the metamodel—responsibilities are a primary focus of the associated OPEN methodology, and are fully supported in the notation.[4,5] However, their presence in the metamodel was inadequate in OML version 1.0. Second, the tablet icon used in the first version for classes (but with the rectangle for metaclasses) needs to be updated. Now that the OMG has endorsed the use of rectangles for class icons, there is less reason to retain the tablet icon in OML for classes, and in OML 1.1 we replace this with the now standard rectangle. OML version 1.1 is then able to reuse the tablet icon to represent a generalized class/instance/role/type (CIRT) when modelers cannot be sure which of the four metatypes they eventually intend. Using CIRTs in this manner is most valuable during early analysis/requirements engineering.

At a higher level of abstraction, the growth of the Java community and the emergence of the OMG standard leads us to replace the version 1.0 term "cluster" with the term "package." However, this is merely a name change, because version 1.1 retains the semantics of the OML package. Therefore, package is a higher level grouping mechanism that is encapsulated and can have information hidden inside it (visually inside the package icon) or visible (visually straddling the package boundary). In addition, the new version now permits two stereotypes of package: {logical} and {physical}.

OML also clarifies the various scenario types, as well as clarifying and extending the association metamodel. While the initial version provided inadequate support for concurrency and distribution, new and improved notation for these important application areas is available in OML version 1.1. Finally, it provides some enhancements to the OML sequence diagrams. These major improvements together with some additional minor changes are discussed in this column (associations) and in our next column (responsibilities, roles, concurrency, etc.).

Donald G. Firesmith has recently joined StorageTek as a Senior Advisory Software Engineer. He may be contacted at firesdg@sweng.stortek.com.

Brian Henderson–Sellers is Professor of Computer Science (Object Technology), School of Information Technology, Swinburne University of Technology, Hawthorn, Victoria, Australia. He may be contacted at brian@csse.swin.edu.au.

DEFINITION OF REFERENTIAL RELATIONSHIPS

In OML version 1.1, a *referential relationship* is the kind of binary, unidirectional dependency relationship (see Fig. 1) in which one modeling element (the client node) refers to another (the server node).[6] A referential relationship provides the client with visibility and access to the server. In this manner, referential relationships are primarily used to allow certain model elements (e.g., objects, classes, types, and packages) to collaborate and delegate. Modelers can view referential relationships as the roads between nodes in a model, and messages and exceptions as the traffic that travels along these roads. OO programming languages typically implement referential relationships using properties (e.g., fields in Java).

Modelers document referential relationships in various kinds of diagrams including semantic nets, interaction diagrams, and scenario class diagrams. As the most common kind of relationship, OML's Common Object Modeling Notation (COMN) uses a simple single arc to represent a referential relationship between two nodes on a diagram.

PARTITIONING REFERENTIAL RELATIONSHIPS

A stereotype is a specialized sub-metatype in the OML metamodel. For example, an abstract class is a kind of class and a referential relationship is a kind of dependency relationship. Referential relationship is an abstract metatype in the OML metamodel. OML version 1.1 categorizes referential relationships into numerous orthogonal partitions of stereotypes, which are based on the modeler's purpose and how the developer chooses to implement the relationship. Because they are logically orthogonal (if often correlated), the resultant cross-product of stereotypes of referential relationships is very large, providing modelers with a wide choice for modeling, but CASE tool vendors with a challenge when developing database schemas.

MODELING-LEVEL PARTITIONS

The following partitions of referential relationships involve modeling-level decisions.

Level

As illustrated in Figure 2, version 1.1 partitions referential relationships into sub-metatypes depending on the level of the client's and server's metatypes.* This partition is part of OML Light because it is fundamental and universally useful. No special COMN notation is required because the metatypes of nodes connected determine the stereotype. This partition contains the following three stereotypes of referential relationships:

- *Linkage*—A linkage (a.k.a., link) referential relationship exists at the instance-level. Either the client or server must be an object (Client L1), role (Server L2), or package instance.
- *Association*—An association referential relationship exists at the definitional-level. Both the client and server must be classes (Client L2), types (Server L2), or packages. An association is a class of linkages. Optional integer ranges on the client and server ends of the arcs signify the multiplicity of the connected nodes.
- *Level-TBD*—In a level-TBD referential relationship, the level of the client and the server is either unknown or unimportant (e.g., a CIRT such as Client L3 and Server L3).

Direction

As illustrated in Figure 3, the new OML partitions referential relationships into sub-metatypes depending on the direction of the relationship. This partition is part of OML Light because it is fundamental and universally useful. This partition contains the following four stereotypes of referential relationships:

- *Unidirectional*—In a unidirectional referential relationship a client refers to a server. Because this is the most commonly used sub-metatype in this partition, COMN signifies a unidirectional relationship by using a single arrowhead on the server end of the arc. A label (typically a verb phrase) naming the relationship is optionally placed in the middle of the arc.
- *Bidirectional*—A bidirectional referential relationship is really a shorthand form representing two unidirectional relationships, each of which is a semi-strong inverse of the other. The two nodes connected are really peers of each other. COMN signifies a bidirectional relationship by using an arrowhead on each end of the arc. Optionally, a forward label is placed next to a small forward-pointing arrow on the port (left) of the arc, and a reverse label is placed next to a reverse-pointing arrow on the starboard (right) of the arc.
- *Observer*—An observer referential relationship is a specialized kind of bidirectional relationship that implements some variation of the observer pattern (e.g., self-addressed stamped envelope). Because the client registers itself with the server that notifies the

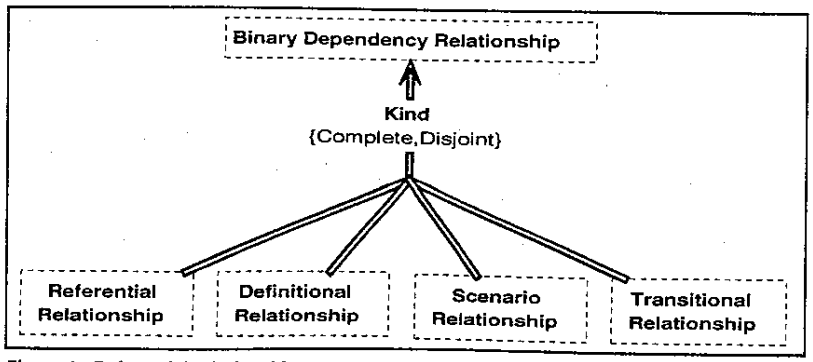


Figure 1. Referential relationships.

* However, in OML version 1.0 this information appeared in the metamodel on two separate diagrams (Figures 3.55 and 3.56 of the *OML Reference Manual*, which are in all other respects duplicates). To tidy up the metamodel, association and linkage become a new partition of stereotypes (see Fig. 2). Note that this partition is independent of and orthogonal to other partitions. Therefore, the idea of "class level" versus "instance level" (as depicted in Fig. 3) is relevant to all other partitions so that we can discuss aggregation, membership and containment associations and linkages.

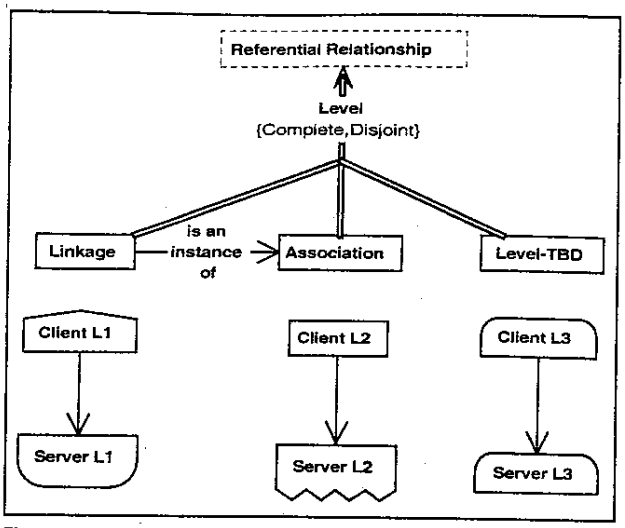


Figure 2. Level Partition Metamodel and notation.

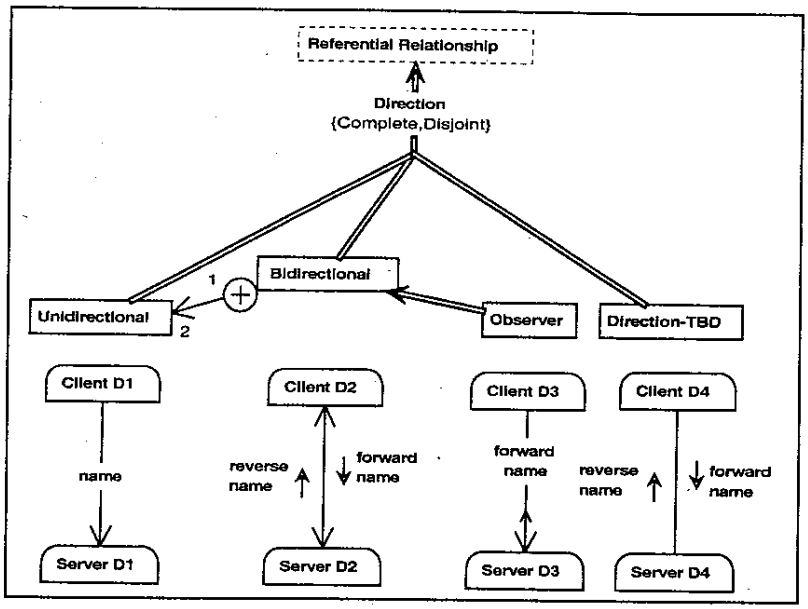


Figure 3. Direction Partition Metamodel and notation.

client when it changes, COMN signifies an observer relationship by using a paired arrowhead on the server end of the arc. A label is optionally placed in the middle of the arc.

- *Direction-TBD*—In a direction-TBD referential relationship, direction has yet to be decided. Usually appropriate only during initial modeling, this stereotype is typically prohibited in deliverable documentation. COMN signifies this sub-metatype by not using arrowheads on either end of the arc. The label of the arc is usually omitted.

Meaning

As illustrated in Figure 4, OML now partitions referential relationships into sub-metatypes depending on the meaning of the relationship.[7] This partition is part of OML Light because it is fundamental and universally useful. It contains the following four concrete stereotypes of referential relationships:

- *Attribution*—An attribution referential relationship connects the client to a descriptive attribute. Because attributes are internal rather than architectural, COMN indicates attribution either by using a drop-down box to list the attribute(s) or by embedding the attribute within the icon of the client (e.g., on internal collaboration diagrams).
- *External*—An external referential relationship connects one node to another external node. External referential relationships are used to link clients with shared, externally visible objects. Because this is the most commonly used stereotype in this partition, COMN provides no special notation or annotation.
- *Whole/Part*—A whole/part (a.k.a., meronymic) referential relationship is one that connects a whole to one of its parts. The whole cannot exist independently of its parts and in fact exists *because of* its parts: Without the parts, the whole has no meaning. The whole provides at least one emergent capability that is not found in its parts,[8] and is therefore more than the sum of its parts. The whole usually sends nontrivial messages to its parts. This sub-metatype can be further partitioned into two stereotypes, as shown in Table 1.
- *Containment*†—A containment relationship links a container to one of its entries. The contents often have an outside existence independent of their location within a container. Furthermore, the container is often highly reusable and can be empty (i.e., it has meaning even if it has no contents). Thus, while the container object will hold references to its contents, a container typically does not use these "roads" of potential message passing.‡ Other than the most general object-level messages, a container object rarely sends a direct message to one of its entries because it does *not* know its contents. Examples of containers include Java's `Vector` and `Hashtable` and Smalltalk's `OrderedCollection`.

COMN signifies containment by placing a small circle containing a union symbol (∪) on the client (container) end of the arc. A fork structure is used in the rare case where the container is

† Note that the metamodel in Figure 3.54 of the *OML Reference Manual* incorrectly shows the containment metaclass to be a subclass of the whole-part relationship. Instead, it should have been a subclass of a referential relationship as shown in Figure 2. On the other hand, the new membership relationship *is* a subclass of the whole-part relationship that is added to the metamodel. OML version 1.1 modifies the nomenclature somewhat and adopts a metamodel in which containment, whole-part, and normal are all subclasses of the (abstract) referential relationship.

‡ See usage in Table 1.

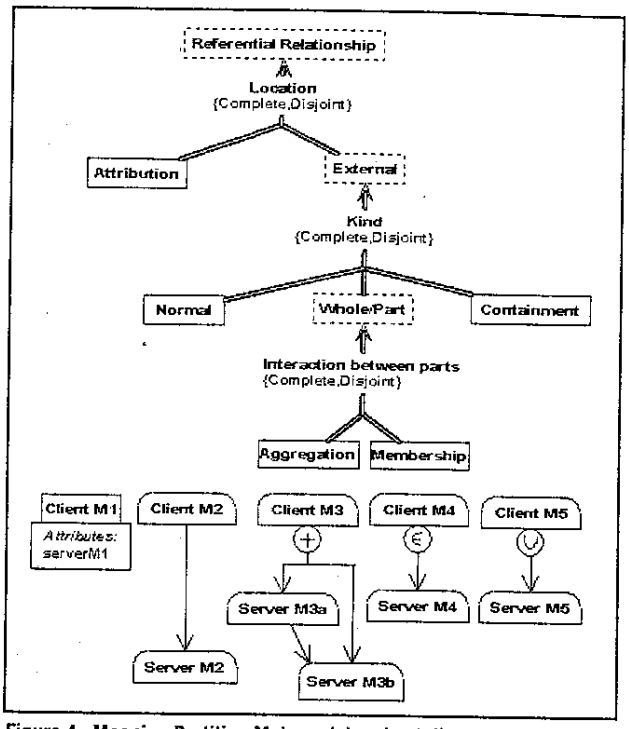


Figure 4. Meaning Partition Metamodel and notation.

heterogeneous and more than one content element is to be diagrammed.

Table 2 summarizes some of the differences among the above stereotypes of referential relationships.

Derivation

As illustrated in Figure 5, a referential relationship may be partitioned into sub-metatypes depending on whether the relationship from the client to the server is direct or derived. This partition contains the following two stereotypes of referential relationships:

- *Direct*—A direct referential relationship is one that directly connects the client to the server. Because this is the most commonly used sub-metatype in this partition, COMN requires no special notation to signify a direct referential relationship.
- *Derived*—A derived referential relationship only logically exists as a concatenation of direct referential relationships. Thus, a derived relationship can be simulated by the navigation along the chain of intermediate direct references. COMN signifies derived relationships by annotating the relationship arc with a short diagonal line segment just prior to any other annotations on the server end of the arc.

Life span dependency

As seen in Figure 6, OML version 1.1 adds a new partition to referential, based on whether the server must be automatically destroyed when the client is destroyed. This partition contains two stereotypes of referential relationships, dependent and independent.

- *Dependent*—In a dependent referential relationship, the life span of the server is limited by the life span of the client (i.e., the server is destroyed when the client that "owns" it is destroyed).

  Destruction of the server occurs regardless of whether the server was created before or after the client was created. In addition, the server may be destroyed before the client is destroyed. This constraint holds only as long as the relationship exists: If a client transfers the relationship to another client, the resulting relationship can be either dependent or independent.

  Note that the server is typically not shared among clients via referential relationships. If multiple clients reference a server that is destroyed when one of the clients is destroyed, the other clients will be left with dangling references unless adequate steps are taken as part of the server's destructor.

  COMN signifies dependent relationships by annotating the relationship arc with a small black tombstone on the right side of the server end of the arc. The tombstone

Table 1. Two stereotypes of the whole/part referential relationship.

Aggregation—An aggregation relationship is a configurational relationship from an aggregate (a.k.a., structure) to one of its component parts. Because at least some of the components must be interrelated in a specific way for an aggregate to function properly, it must contain at least two components. A car engine is a good example of an aggregate; the parts of a functioning car engine must be linked together for the engine to work.

COMN signifies aggregation by placing a small circle containing a plus sign on the client (aggregate) end of the arc. One memory aid is to think of the whole as the sum (plus sign) of its parts. Another is that the circle with a plus sign looks like the head of a Philips screw used to connect the parts of an aggregate. A fork structure can be used if more than one part is to be diagrammed.

Membership—A membership relationship is a nonconfiguration relationship from a group to one of its members. Although the members of a group may be related by references, such relationships need not exist and are not intrinsic to the group. A group must contain at least one member. Example membership relationships are the relationships from a club to its members or a forest of trees.

COMN signifies membership by placing a small circle containing an ε (the greek letter "e," signifying membership in basic set theory) on the client (group) end of the arc. A fork structure can be used if more than one member is to be diagrammed.

§ Note that sharing is not a partition of referential relationships because sharing requires the existence of at least two referential relationships to the same server.

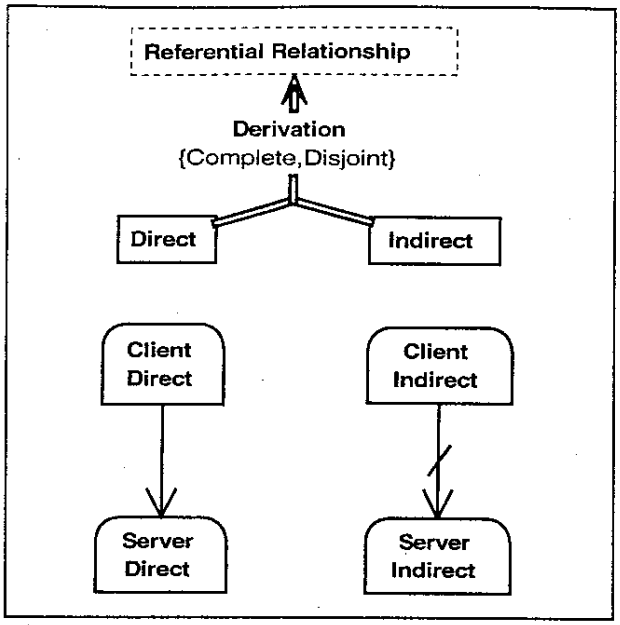


Figure 5. Derivation partition metamodel and notation.

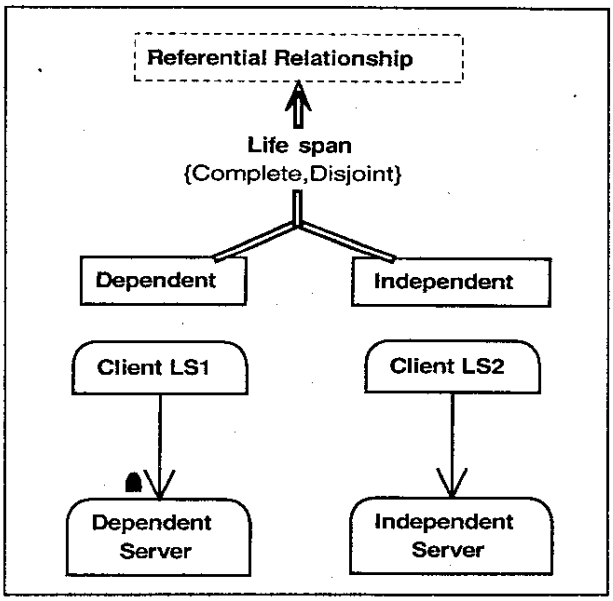


Figure 6. Life span metamodel and notation.

symbolizes death and is thus intuitively understandable, as well as consistent with role and CIRT symbols used as nodes in semantic nets and interaction diagrams.

- *Independent*—In an independent referential relationship the life span of the server is independent of the life span of the client. This often (but not always) results from multiple clients "sharing" the same server via referential relationships.[5] Because this is the most commonly used sub-metatype in this partition, COMN requires no special notation to signify an independent life span relationship; the absence of a tombstone indicates no such lifetime dependency.

Variability

As illustrated in Figure 7, a referential relationship may be partitioned into sub-metatypes depending on whether the relationship can be modified to refer to a new server. This partition contains the following three stereotypes of referential relationships:

- *Variable*—A variable referential relationship can be modified to refer to a new server, and is implemented by supplying a setter operation for the corresponding property. COMN signifies a variable relationship by annotating the relationship arc with the {Variable} stereotype.
- *Constant*—A constant referential relationship cannot be modified to refer to a new server, and is implemented by not supplying a setter operation for the corresponding property. Some languages also provide a const or constant reserved word to guarantee immutability. COMN signifies a constant relationship by annotating the relationship arc with the {Constant} stereotype.
- *Variability-TBD*—A variability-TBD referential relationship is one in which the server is destroyed whenever the client is destroyed. Because this is the most commonly used sub-metatype in this partition, COMN requires no special annotation to signify this stereotype.

Optionality

As illustrated in Figure 8, a referential relationship may be partitioned into sub-metatypes depending on whether it is mandatory or optional. This partition contains the following three stereotypes of referential relationships:

- *Mandatory*—A mandatory referential relationship must always refer to a non-null server between the execution of visible operations. COMN signifies a mandatory relationship by supplying a server multiplicity that is greater than zero.

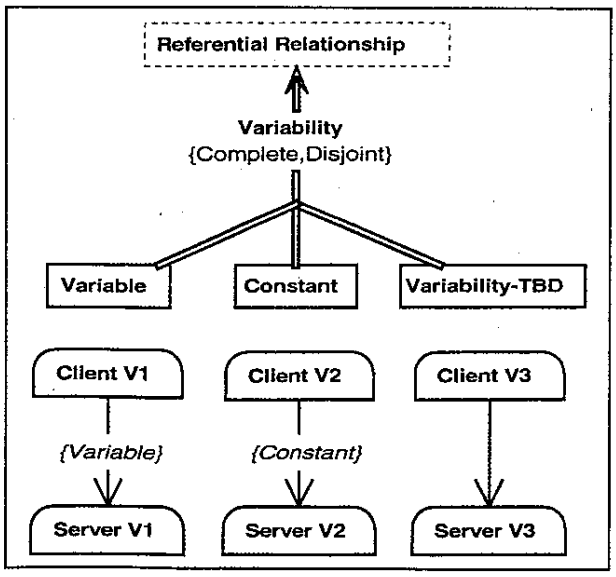


Figure 7. Variability partition metamodel and notation.

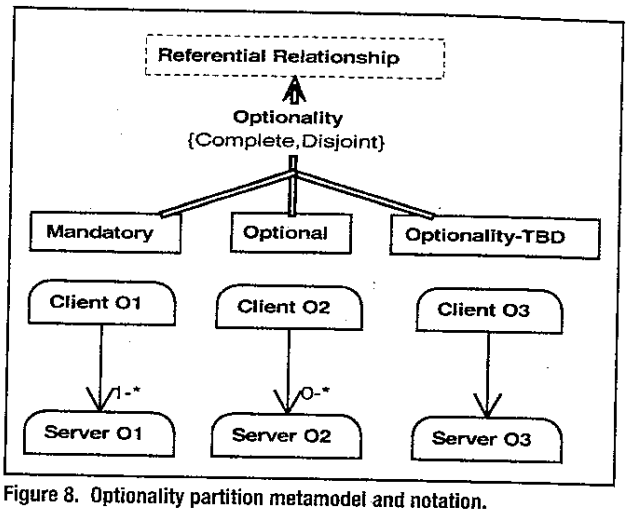


Figure 8. Optionality partition metamodel and notation.

- *Optional*—An optional referential relationship on the other hand can refer to a non-null server between the execution of visible operations. COMN signifies an optional relationship by using a server multiplicity that allows zero.
- *Optionality-TBD*—In an optionality-TBD referential relationship optionality is either unknown or unimportant. COMN signifies an optionality-TBD relationship by not including a server multiplicity.

Usage

As illustrated in Figure 9, a referential relationship may be partitioned into sub-metatypes depending on whether the relationship will be used (traversed) by any messages or exceptions. This partition contains the following three stereotypes of referential relationships:

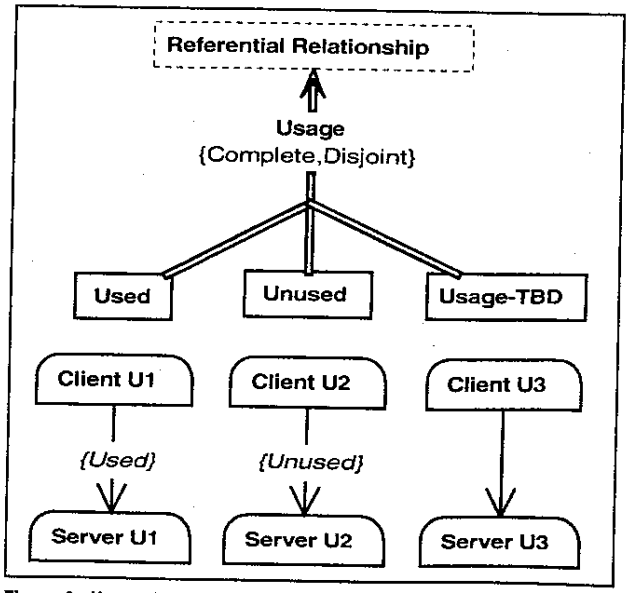


Figure 9. Usage Partition Metamodel and notation.

- *Used*—A used referential relationship will be traversed by at least one message or exception. Most referential relationships exist to send messages and receive exceptions. COMN signifies this sub-metatype by annotating the relationship arc with the {Used} stereotype.
- *Unused*—An unused referential relationship will not be traversed by any messages or exceptions. For example, objects sometimes obtain references to other objects (e.g., via message parameters) for the sole purpose of passing on these references to yet other objects. COMN signifies this sub-metatype by annotating the relationship arc with the {Unused} stereotype.
- *Usage-TBD*—In a usage-TBD referential relationship whether the relationship will be traversed is either unknown or unimportant. Because this is the most commonly used sub-metatype in this partition, no special annotation is required.

IMPLEMENTATION-LEVEL PARTITIONS

The following partitions of referential relationships involve implementation-level design decisions. They are typically used to provide developers with implementation-level constraints or hints. UpperCASE tools may also automatically determine them during reverse engineering. They are optional, only occasionally useful, and therefore not a part of OML Light.

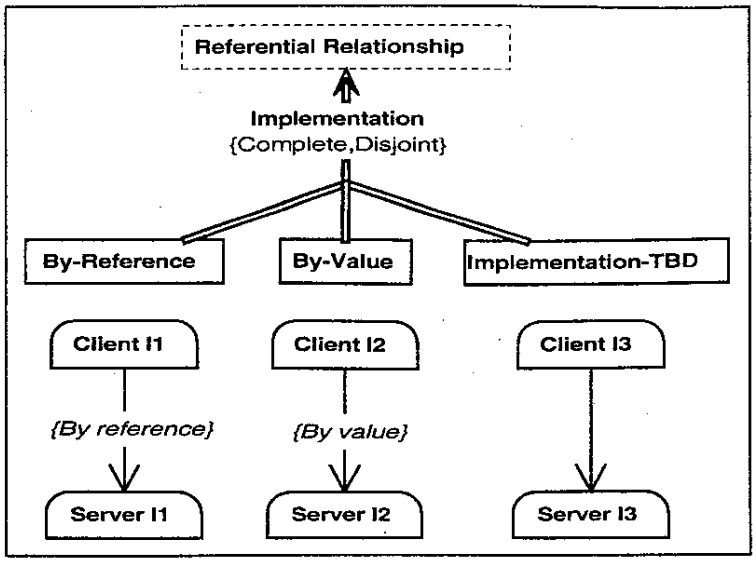


Figure 10. Implementation partition metamodel and notation.

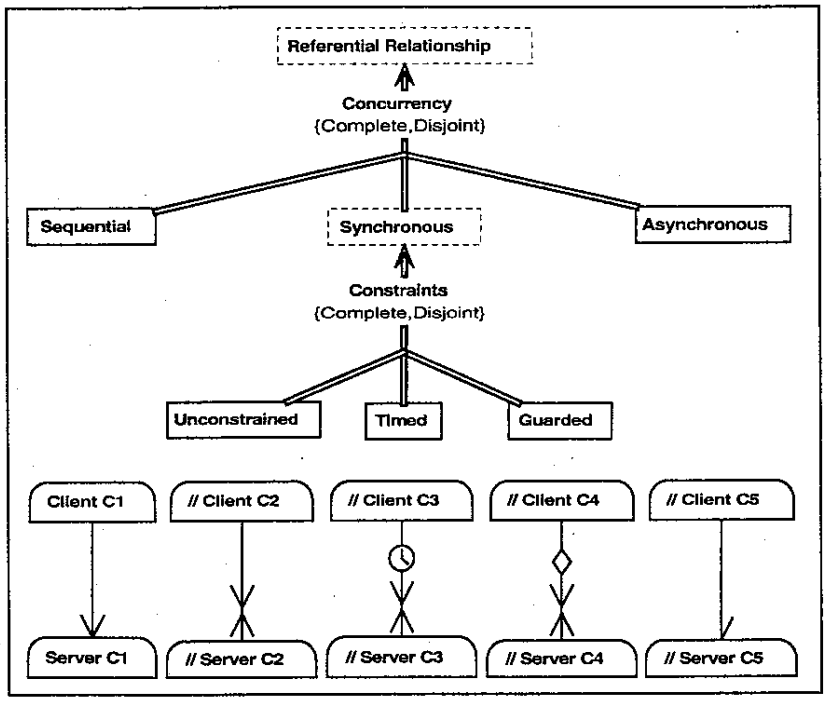


Figure 11. Concurrency partition metamodel and notation.

Implementation

As illustrated in Figure 10, a referential relationship may be partitioned into sub-metatypes depending on how the modeler intends to implement it and whether the language supports the intended implementation.# This partition contains the following three stereotypes of referential relationships:

- *By-reference*—A by-reference relationship is implemented by giving the client a reference to the server, and supports by languages like Java and Smalltalk. COMN signifies by-reference by annotating the relationship arc with the {By-Reference} stereotype.
- *By-value*—A by-value referential relationship is implemented by giving the client a pointer to the server. A by-value relationship is typically used when only the client has visibility and access to the server. COMN signifies by-value by annotating the relationship arc with the {By-Value} stereotype.
- *Implementation-TBD*—In an implementation-TBD referential relationship it is unknown or unimportant how the relationship is to be implemented. Because this is the most commonly used sub-metatype in this partition, no special annotation is required.

Concurrency

As illustrated in Figure 11, a referential relationship may be partitioned into sub-metatypes depending on the concurrency of the messages that travel along it. The use of this partition is practical if only a single kind of message is sent along the referential relationship (the usual case, and the only case for sequence diagrams, which capture only a single message at a time). This partition contains the following three stereotypes of referential relationships:

- *Sequential*—A sequential referential relationship is traversed by sequential messages, which involve only a single thread of control. Because this is the most commonly used sub-metatype in this partition, COMN signifies sequential relationships by using a standard arrowhead on the server end of the arc.
- *Synchronous*—A synchronous relationship is traversed by synchronous messages, which involve two threads of control that must synchronize during the interaction. In order to imply the synchronization of the two threads, COMN signifies synchronous relationships by using two arrowheads aimed at each other at the server end of the arc. In version 1.1 of OML, synchronous relationships can be partitioned into the three stereotypes seen in Table 3.
- *Asynchronous*—An asynchronous relationship is traversed by asynchronous messages, which involve two threads of control that do not synchronize during message passing. Because asynchronous messages are "fire and forget" and therefore do not

For example, C++ supports both by-value and by-reference properties, whereas Java and Smalltalk support only by-reference.

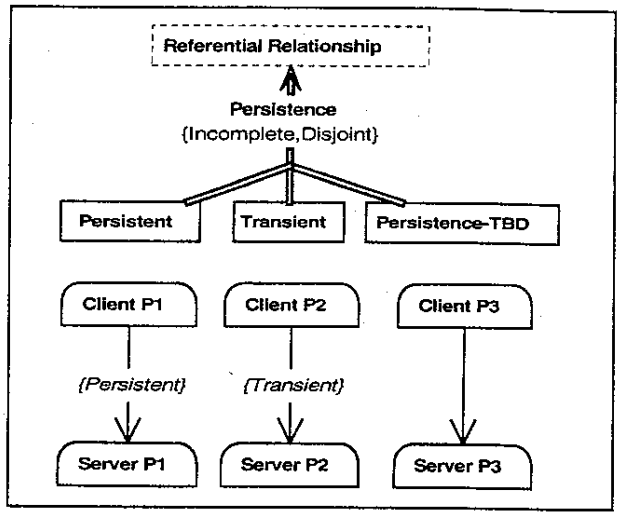


Figure 12. Persistence partition metamodel and notation.

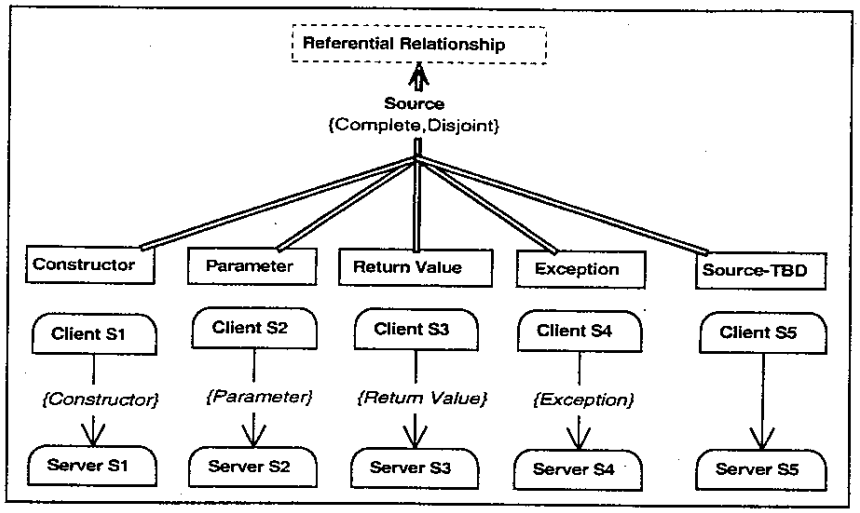


Figure 13. Source partition metamodel and notation.

have return values, COMN signifies asynchronous relationships by using a partial arrowhead.

- *Concurrency-TBD*—In a concurrency-TBD referential relationship the concurrency of the messages that traverse the relationship is unknown or unimportant, and no special annotation is required.

Persistence

As Figure 12 shows, a referential relationship may be partitioned into sub-metatypes depending on its persistence. This partition contains the following three stereotypes of referential relationships:

- *Persistent*—A persistent relationship is structural, persisting as long as the associated object(s) exist. It is usually established during construction or initialization and is implemented via a field. COMN signifies this sub-metatype by annotating the relationship arc with the {Persistent} stereotype.
- *Transient*—A transient referential relationship does not persist past the execution of a single operation. It is usually established after initialization and implemented via a temporary variable that is set to the value of a message parameter, a message return value, or an exception. COMN signifies this sub-metatype by annotating the relationship arc with the {Transient} stereotype.
- *Persistence-TBD*—In a persistence-TBD referential relationship the persistence of the relationship is unknown or unimportant—no special annotation is required.

Source

As illustrated in Figure 13, a referential relationship may be partitioned into sub-metatypes depending on its source. This partition contains the following five stereotypes of referential relationships:

- *Constructor*—The value of a constructor referential relationship comes in the form of a constructor parameter or temporary variable. COMN signifies this sub-metatype by annotating the relationship arc with the {Constructor} stereotype.
- *Parameter*—The value of a parameter relationship comes in the form of a message parameter. COMN signifies this sub-metatype by annotating the relationship arc with the {Parameter} stereotype.
- *Return Value*—The value of a return value referential relationship comes in the form of a message return value. COMN signifies this sub-metatype by annotating the relationship arc with the {Return Value} stereotype.
- *Exception*—An exceptional relationship's value comes in the form of an exception, and COMN signifies it by annotating the relationship arc with the {Exception} stereotype.

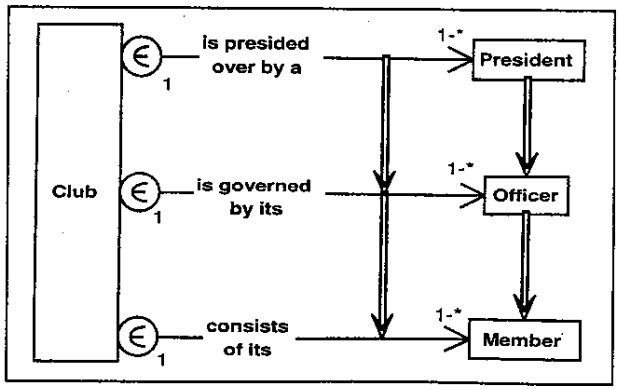


Figure 14. Inheritance between associations.

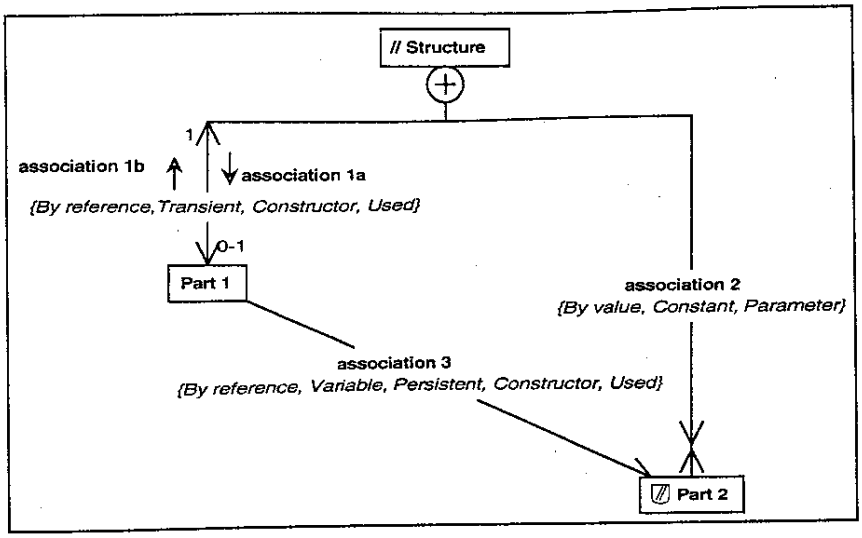


Figure 15. Example use of stereotypes.

- *Source-TBD*—A source-TBD referential relationship is one in which the source of the relationship is unknown or unimportant. Because this is the most commonly used sub-metatype in this partition, no special annotation is required.

INHERITANCE BETWEEN ASSOCIATIONS

Because an association is a class of links, it is reasonable to support the inheritance between associations. As illustrated in Figure 14, the president of a club is a kind of officer which in turn is a kind of member. Therefore, the "is presided over by a" association is a subclass of the "is governed by its" association that in turn is a subclass of the "consists of its" association. OML version 1.1 indicates inheritance between associations in the same way as any other definitional relationship (i.e., with a double arrow).

EXAMPLE USE OF REFERENTIAL RELATIONSHIP STEREOTYPES

Figure 15 illustrates the use of referential relationship stereotypes. A concurrent class (Structure) is connected by aggregation associations to a sequential part (Part 1) and a threadsafe concurrent part (Part 2). The aggregation relationship between the structure and Part 1 is bidirectional, optional (because of the server's multiplicity), implemented by reference, transient, used to send messages, and traversed by sequential messages. It was formed during the execution of the constructor of the structure. The aggregation relationship from the structure to Part 2 is implemented by value, constant, and is traversed by synchronous messages. It was formed using a parameter of a message sent to the structure. Association 3 is implemented by reference, variable, persistent, formed during the execution of the constructor of Part 1, and used for the traversal of asynchronous messages. Although diagrams can become cluttered if too many stereotypes are used, it is nice to know that a wide range of optional stereotypes are available if needed.

Table 2. Differences among stereotypes of referential relationships.

	Normal	Aggregation	Membership	Containment
Symbol	None	+ in circle	∈ in circle	∪ in circle
Client Name	Client	Aggregate, Structure	Group	Container
Server Name	Server	Part	Member	Entry
Min. Client Size	N/A	2	1	0
Emergent Properties	No	Yes	Yes	No
Messages Sent	Typically	Typically	Typically	Rarely
Example		Car Engine/Piston	Forest/Trees	Vector

Table 3. The three types of sychronous relationships.

Unconstrained—An unconstrained synchronous relationship does not place any constraints on the synchronous messages that traverse it. Because this is the most commonly used stereotype in the partition, COMN requires no special annotation.

Timed—In a timed synchronous relationship the client will abandon the associated synchronous message if the server is not willing to accept the message within a time limit specified by the client. COMN signifies a timed synchronous relationship by placing a small clock icon just prior to the synchronous arrowheads on the server end of the arc.

Guarded—In a guarded synchronous relationship the server will reject the associated synchronous message if the server's guard condition is violated. COMN signifies a guarded synchronous relationship by placing a diamond (guard icon) just prior to the synchronous arrowheads on the server end of the arc.

THINGS NOT EXPLICITLY SUPPORTED

N-ary relationships

OML does not support the UML concept of "n-ary relationships" because such relationships are best modeled as a normal class having *n* associations to other classes.

Associative classes

OML does not support the UML concept of "associative class," that is, an association with behavior and properties. This is because such an association is best modeled as a nor-

mal class, having behavior and properties and also associations to other classes. In other words, the reification of an association changes its metatype from Association to Class; there is therefore no need to use multiple inheritance from association and class in this case.

REFERENTIAL RELATIONSHIPS WITH ATTRIBUTES

OML does not support the UML concept of "an association with attributes." This is because such a relationship is best modeled as a normal class with attributes and also associations to other classes.

Qualified associations

OML does not support the UML concept of a "qualified association" because its use is generally indicative of missing classes.

Shared associations

The UML concept of a "shared association" is not supported because multiple referential relationships to the same server are required to achieve the sharing of the server, i.e., sharing cannot refer to a single relationship but is a characteristic of specific servers (classes).

CONCLUSION

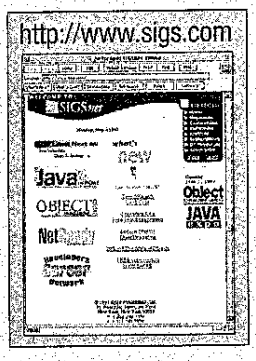
In extending OML's version 1.0 metamodel,[1] we have clarified a number of issues not previously understood and some that have been obfuscated in the OMG's UML metamodel and notation. These refinements improve the OML metamodel and, in so doing, create the first part of the version 1.1 definition of OML as well as providing possible extensions for the next version of UML. ■

Acknowledgment

This is contribution number 98/2 of the Centre for Object Technology Applications and Research.

References

1. Firesmith, D. G., B. Henderson-Sellers, and I. Graham. *OPEN Modeling Language (OML) Reference Manual*, SIGS Books, New York, 1997.
2. Henderson-Sellers, B. and D. G. Firesmith. "COMMA: Proposed Core Model," *JOOP*, 9(8):48–53, Jan. 1997.
3. Henderson-Sellers, B., D. Firesmith, and I. M. Graham. "The Benefits of Common Object Modeling Notation," *JOOP*, 10(5):28–34, Sept. 1997.
4. Graham, I., B. Henderson-Sellers, and H. Younessi. *The OPEN Process Specification*, Addison-Wesley Longman, London, UK, 1997.
5. Henderson-Sellers, B., A. J. H. Simons, and H. Younessi. *OPEN's Toolbox of Techniques*, Addison-Wesley Longman, London, UK, 1998.
6. Firesmith, D. G. and B. Henderson-Sellers. "Clarifying Specialized Forms of Association in UML and OML," *JOOP*, 11(2):47–50, May 1998.
7. Henderson-Sellers, B. "OPEN Relationships—Compositions and Containment," *JOOP*, 10(7):51–55, Nov./Dec. 1997.
8. Kilov, H. and J. Ross. *Information Modeling: An Object-Oriented Approach*, Prentice Hall, Englewood Cliffs, NJ, 1994.



JOOP
The Journal of Object-Oriented Programming
The Global Authority on Object Development
June 1998 Vol. 11, No. 3

Cover: Dave Stewart/FPG