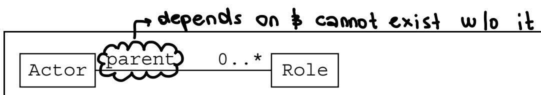
Streamlined Object Modeling Summary

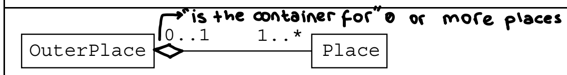
B

12 Collaboration Patterns



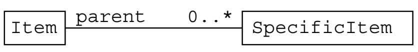
Use to model the participation of a person, organization, place, or thing in a context.

- An actor knows about zero to many roles, but typically takes on only one of each kind.
- A role represents a unique view of its actor within a context. The role depends on its actor and cannot exist without it.



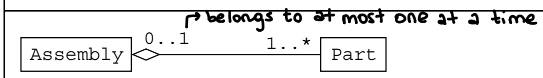
Use to model a hierarchy of locations where events happen.

- An outer place is the container for zero or more places.
- A place knows at most one outer place. The place's location depends on the location of its outer place.



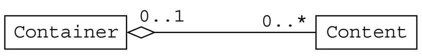
Use to model a thing that exists in several distinct variations.

- An item is the common description for zero to many specific items.
- A specific item knows and depends on one item. The specific item's property values distinguish it from other specific items described by the same item.



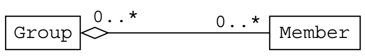
Use to model an ensemble of things.

- An assembly has one or more parts. Its parts determine its properties, and the assembly cannot exist without them.
- A part belongs to at most one assembly at a time. The part can exist on its own.



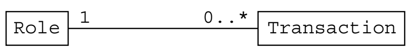
Use to model a receptacle for things.

- A container holds zero or more content objects. Unlike an assembly, it can be empty.
- A content object can be in at most one container at a time. The content object can exist on its own.



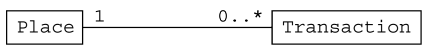
Use to model a classification of things.

- A group contains zero or more members. Groups are used to classify objects.
- A member, unlike a part or content objects, can belong to more than one group.



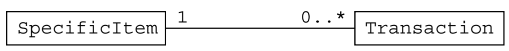
Use to record participants in events.

- A transaction knows one role, the doer of its interaction.
- A role knows about zero or more transactions. The role provides a contextual description of the person, organization, thing, or place involved in the transaction.



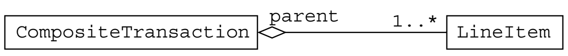
Use to record where an event happens.

- A transaction occurs at one place.
- A place knows about zero to many transactions. The transactions record the history of interactions at the place.



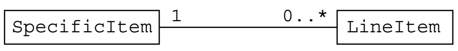
Use to record an event involving a single thing.

- A transaction knows about one specific item.
- A specific item can be involved in zero to many transactions. The transactions record the specific item's history of interactions.



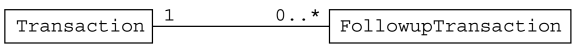
Use to record an event involving more than one thing.

- A composite transaction must contain at least one line item.
- A line item knows only one composite transaction. The line item depends on the composite transaction and cannot exist without it.



Use to record the particular involvement of a thing in an event involving multiple things.

- A specific item can be involved in zero to many line items.
- A line item knows exactly one specific item. The line item captures details about the specific item's interaction with a composite transaction.



Use to record an event that occurs only after a previous event.

- A transaction knows about some number of follow-up transactions.
- A follow-up transaction follows and depends on exactly one previous transaction.

Three Fundamental Patterns

Generic – Specific	• actor - role • item - specific item • composite transaction - line item
Whole – Part	• outer place - place • assembly - part • container - content • group - member
Specific – Transaction	• role - transaction • place - transaction • specific item - transaction • transaction - follow-up transaction • specific item - line item

Five Kinds of Collaboration Rules

Type	• Is the potential collaborator the right type for me? • In entity collaborations, the most specific collaborator owns the rule. • In event collaborations, the interacting entity owns the rule.
Multiplicity	• Do I have too many collaborations to establish another? • Will I have too few collaborations if I remove one? • Each collaborator checks its own multiplicity rules.
Property	• Verify my property values or the potential collaborator's property values against a constant standard. • The collaborator who knows the standard owns the rule. • Compare my property values with a potential collaborator's property values. • The collaborator who knows the acceptable range of values owns the rule
State	• Am I in the proper state for establishing or dissolving a collaboration? • Each collaborator checks its own state rules.
Conflict	• Do any of my collaborators conflict with the potential collaborator? • Since conflict rules are just collaboration rules between indirect collaborators, the same principles for deciding who owns the collaboration rule apply.

Pattern Player Collaboration Rules

	Type	Multiplicity	Property	State	Conflict
Actor					
Role	✓	✓	✓	✓	✓
Outer Place		✓	✓	✓	
Place	✓	✓	✓	✓	✓
Item		✓		✓	
Specific Item	✓	✓	✓	✓	✓
Assembly		✓	✓	✓	
Part	✓	✓	✓	✓	✓
Container		✓	✓	✓	
Content	✓	✓	✓	✓	✓
Group		✓	✓	✓	✓
Member	✓	✓	✓	✓	✓
Role	✓	✓	✓	✓	✓
Transaction		✓			
Place	✓	✓	✓	✓	✓
Transaction		✓			
Specific Item	✓	✓	✓	✓	✓
Transaction		✓			
Composite Transaction		✓			
Line Item	✓	✓			
Specific Item		✓			
Line Item		✓			
Transaction	✓	✓	✓	✓	✓
Follow-up Transaction		✓			

Three Kinds of Services

Conduct Business	• A service that kick offs processes and accomplishes an action rather than answers a question. • A service that creates new objects or changes objects' states. • Typical conduct business services include creating new objects, establishing collaborations, and setting property values. • In entity collaborations, the most specific collaborator directs the process. • In event collaborations, the transaction directs the process.
Determine Mine	• A service that satisfies requests for current information about the object's properties, state, and collaborations. • A service that should never alter the states of any objects. • Typical determine mine services include returning property values and collaborators, working with collaborators to determine an aggregate value, and performing a search.
Analyze Transactions	• A service that assesses historical or future information captured in associated events. • A service that should never alter the states of any objects. • Typical analyze transactions services compute summary results from past transactions, compute summary results from collaborators of past transactions, and locate future scheduling conflicts.

Six Kinds of Properties

Descriptive	• Domain-specific and tracking properties
Time	• Date or time properties
Lifecycle State	• Status of one-way state transitions (e.g., nomination status: pending, in review, approved, rejected)
Operating State	• Status of two-way state transitions (e.g., sensor state: off, on)
Role	• Classification of people (e.g., team member role: chair, admin, member)
Type	• Classification of places, things, and events (e.g., store type: physical, online, phone)

Methods for Enforcing Collaboration Rules

add (aCollaborator) **remove** (aCollaborator)	• Adds or removes the collaborator if the collaboration rules allow. • Calls the corresponding "test" and "do" methods. • *Example*: `addNomination`
testAdd (aCollaborator) **testRemove** (aCollaborator)	• Tests the collaborator against the receiver object's collaboration rules and raises an exception if any rules fail. • *Example*: `testAddNomination`
testAddConflict (directCollaborator, indirectCollaborator, ….) **testRemoveConflict** (directCollaborator, indirectCollaborator, ….)	• Checks for conflicts with the direct collaborator and one or more indirect collaborators and raises an exception if any rules fail. • *Example*: `testAddNominationConflict`
doAdd (aCollaborator) **doRemove** (aCollaborator)	• Adds or removes the collaborator into or out of the receiver object's collaboration variable or collection without rule checking. • *Example*: `doAddNomination`

AppB.fm Page 340 Friday, August 24, 2001 11:07 AM

Methods for Enforcing Property Rules

set (aValue) **setValue**( )	• Sets property to a given or enumerated value if property rules allow. • Calls the corresponding "test" and "do" methods. • *Example*: `setName` • *Example*: `setStatusAccepted`
testSet (aValue) **testSetValue**( )	• Tests the value against the receiver object's property rules and raises an exception if any rules fail. • *Example*: `testSetName` • *Example*: `testSetStatusAccepted`
doSet (aValue) **doSetValue**( )	• Assigns a value into the object's property variable without rule checking. • *Example*: `doSetName` • *Example*: `doSetStatusAccepted`

Collaboration Rule Directors

Generic – Specific	• `specific`
Whole – Part	• `part`
Specific – Transaction	• `transaction`

AppB.fm Page 341 Friday, August 24, 2001 11:07 AM

Object Definition DIAPER

Define	• Name the class, indicate its superclass, and specify any interfaces exhibited. • Define variables for properties. • Define variables for collaborations.
Initialize	• Create construction method that has parameters for property values and collaborations necessary for the object to exist. • Construction method sets remaining properties to default or initial values. • Construction method creates collections for collective collaborations.
Access	• Write property accessors and collaboration accessors. • Write "test" methods for checking property and collaboration rules. • Write "do" methods for assigning and removing property values and collaborators.
Print	• Describe values of select properties and ask select collaborators to describe themselves. • The most specific collaborator asks generic collaborators to describe themselves. • An event asks interacting entity collaborators to describe themselves.
Equals	• Check if the receiving object is equal to another by comparing property values and select collaborators. • The most specific collaborator asks generic collaborators to compare themselves. • An event asks interacting entity collaborators to compare themselves.
Run	• Create sample objects with typical property values and sample objects for select collaborators. • The class of the most specific collaborator creates its sample objects by using sample objects from the classes of the generic collaborators. • The class of an event creates its sample objects by using sample objects from the classes of the event collaborators.

AppB.fm Page 342 Friday, August 24, 2001 11:07 AM

Object Inheritance Interfaces

Profile	• Specifies parent services that are object inherited by its child objects. • Includes most determine mine services, except when they summarize information about other child objects. • Includes analyze transactions services if the child object inherits the transaction collaborators analyzed. • Includes no conduct business services.
Conduct Business	• Specifies parent services that are not object inherited by its child objects. • Includes all public conduct business services, plus determine mine and analyze transactions services not in the profile interface. • Extends the profile interface. • When used to specify services of objects not involved in object inheritance, includes all public conduct business, determine mine, and analyze transactions services.