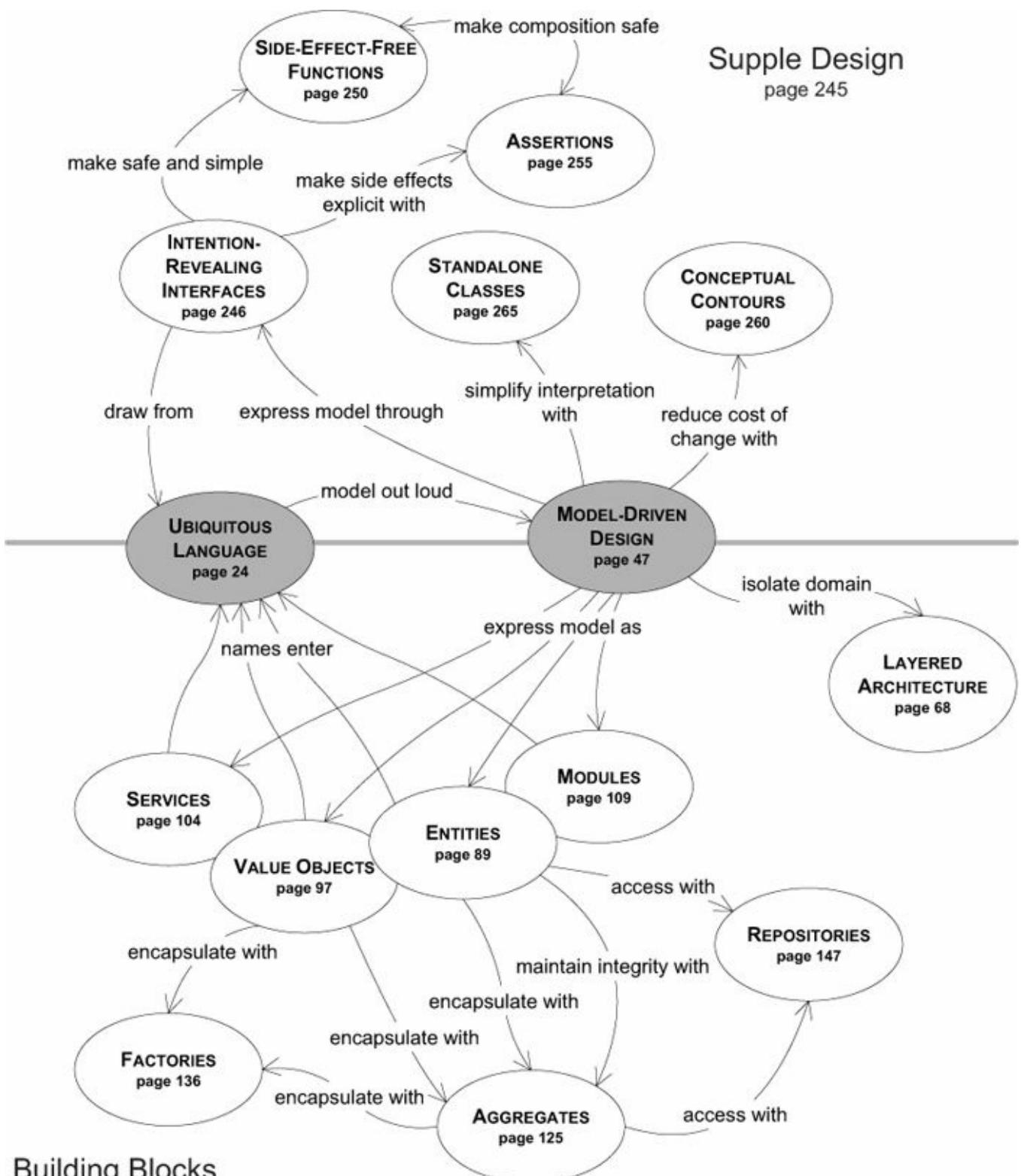


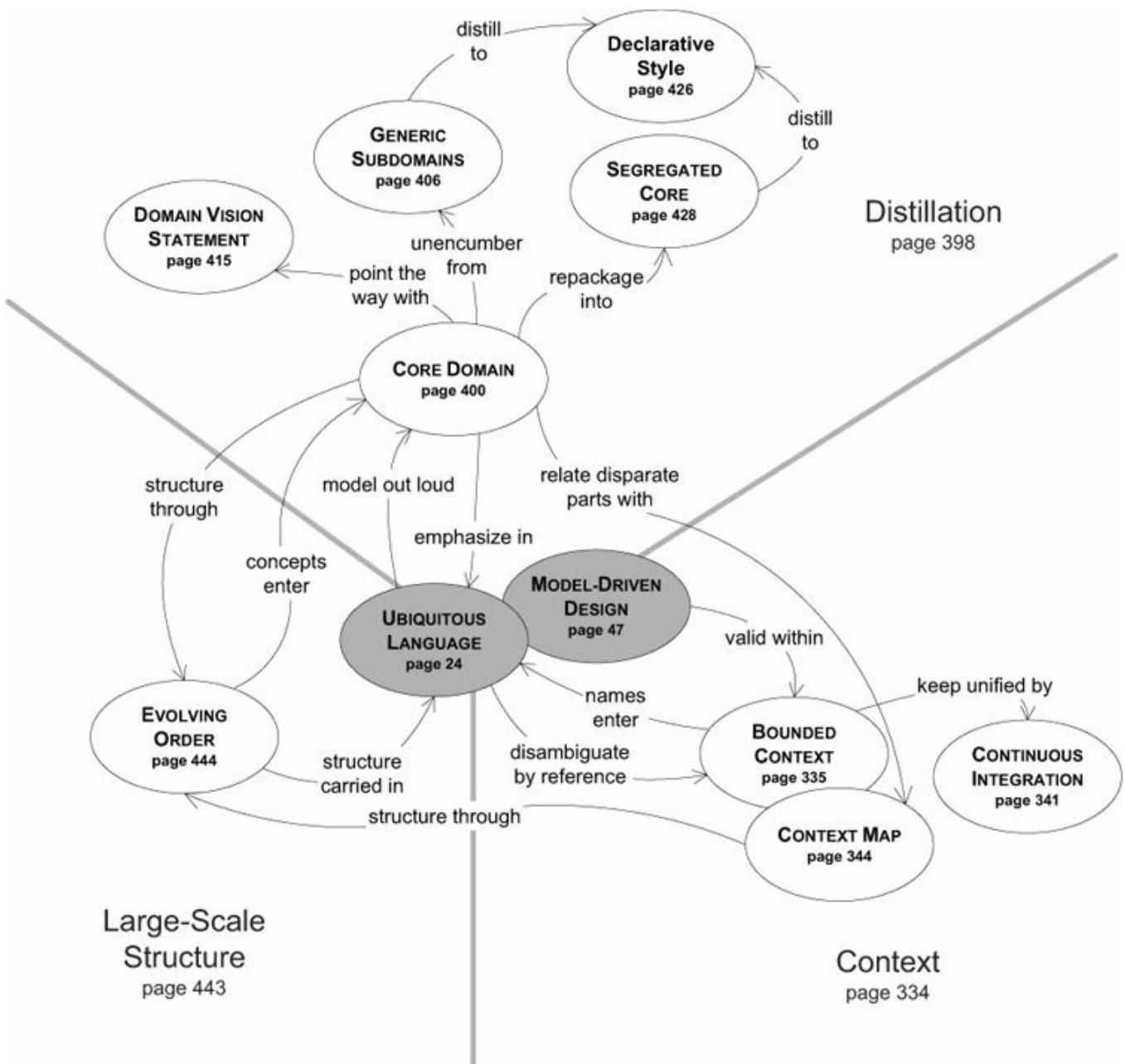
Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

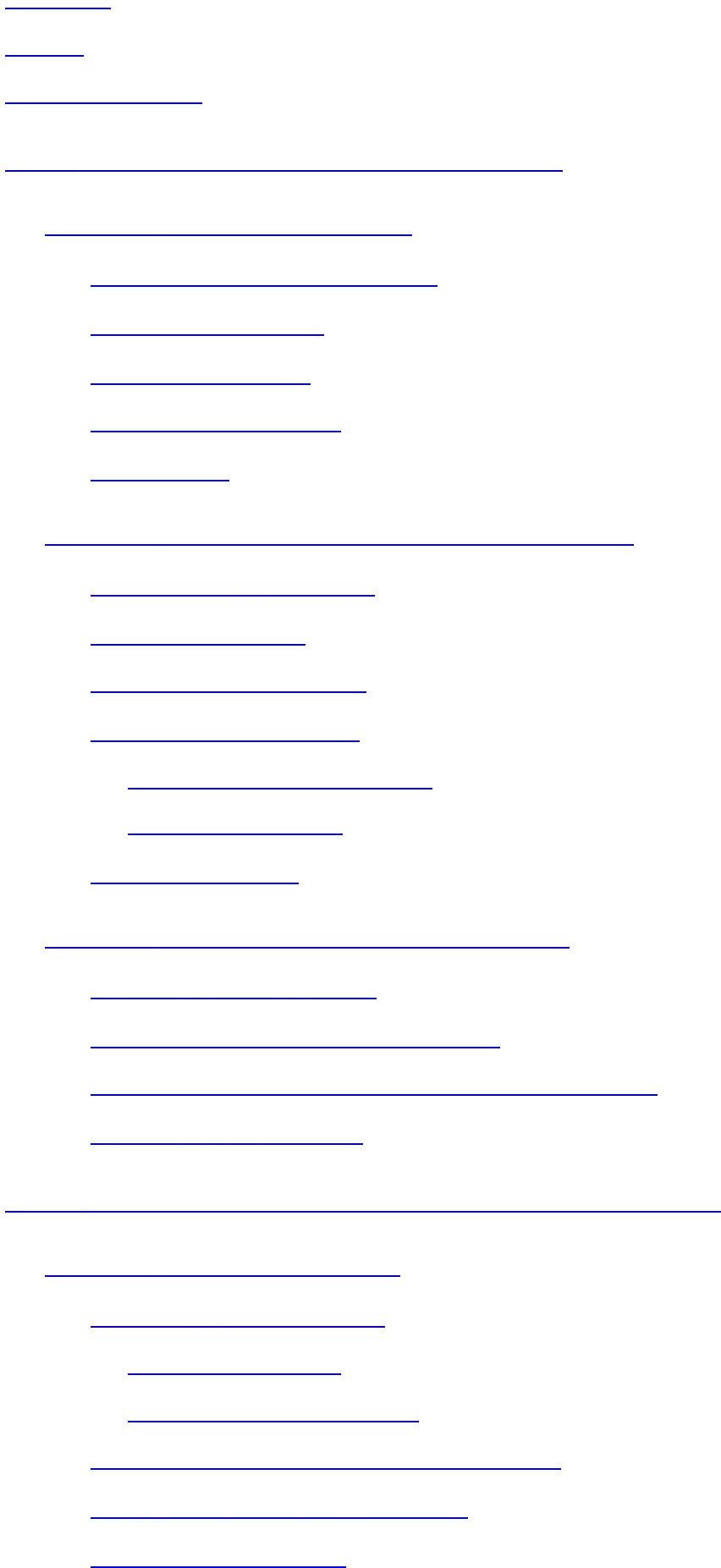


Eric Evans
Foreword by Martin Fowler

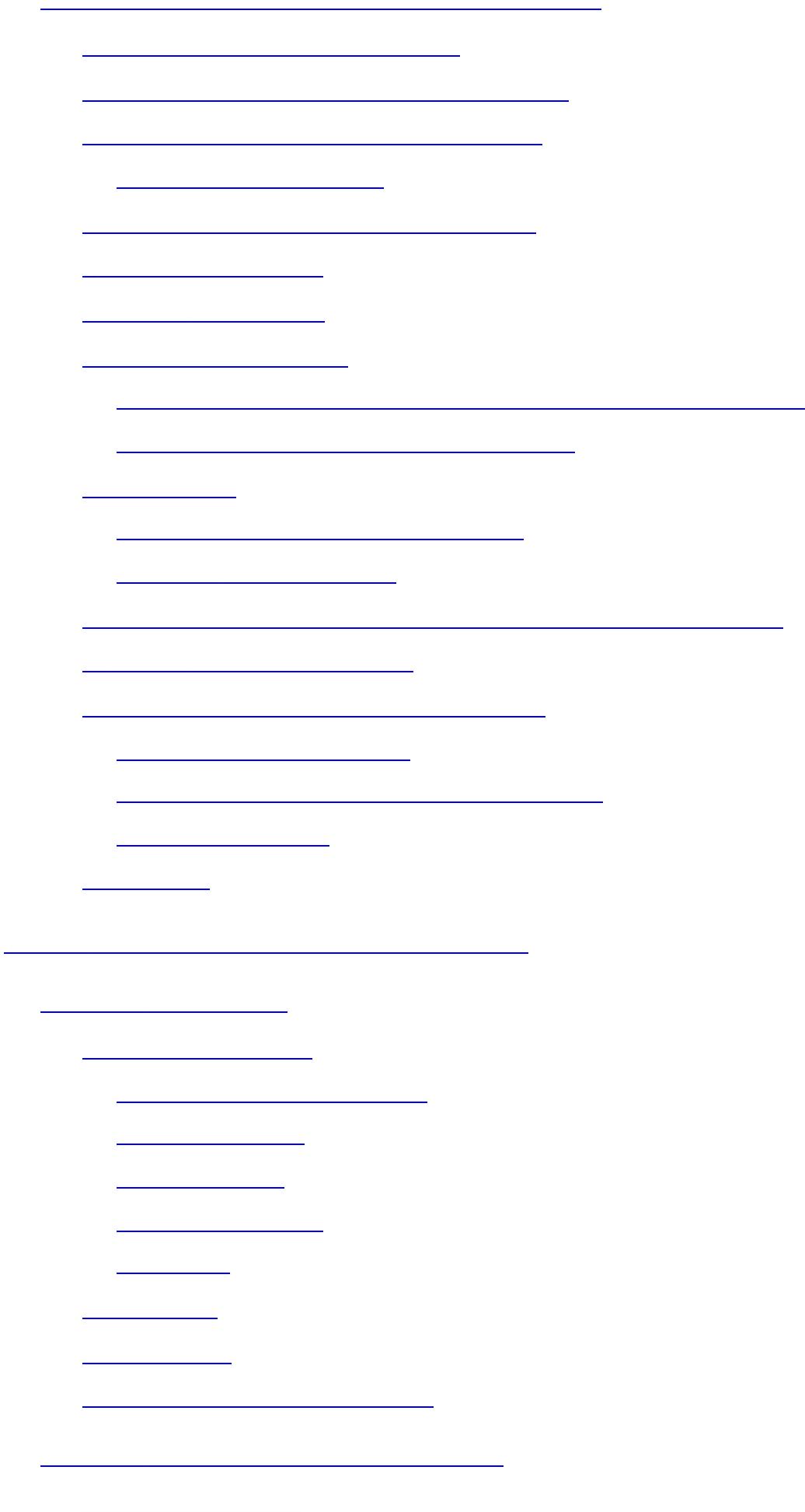


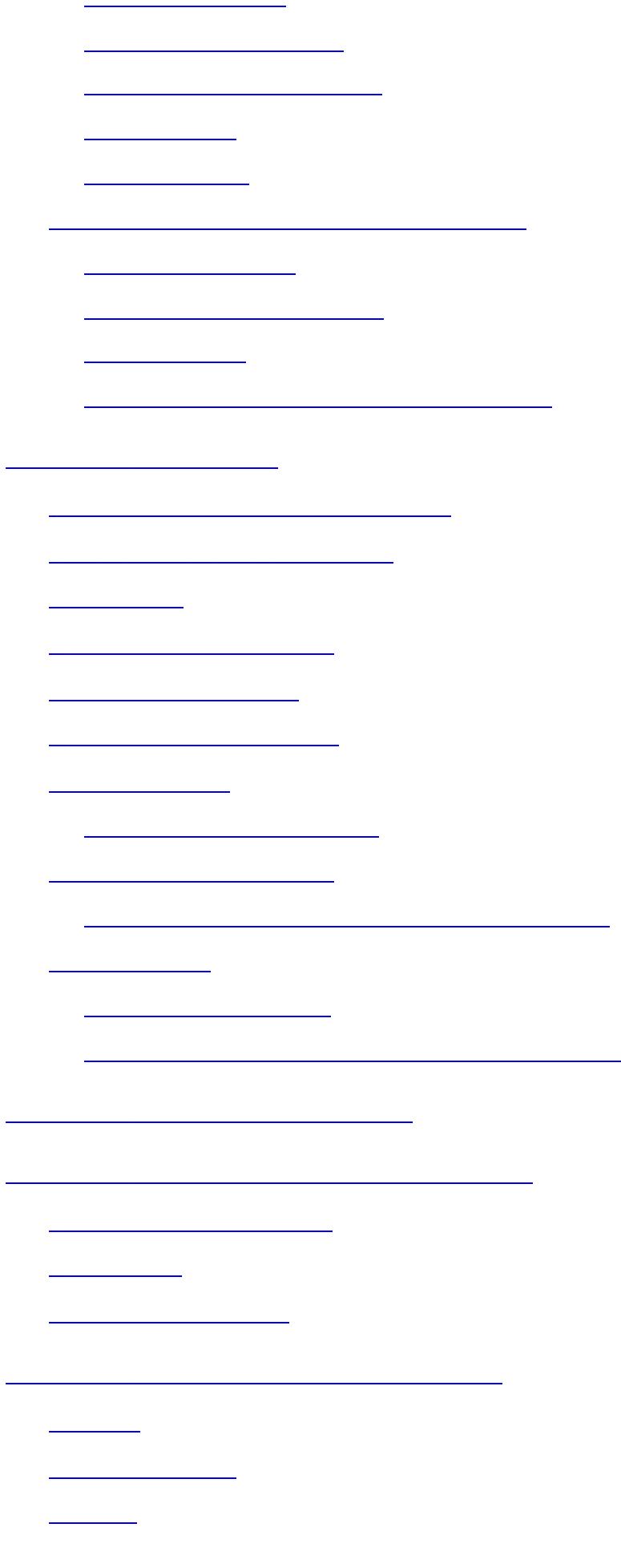


◆ Addison-Wesley

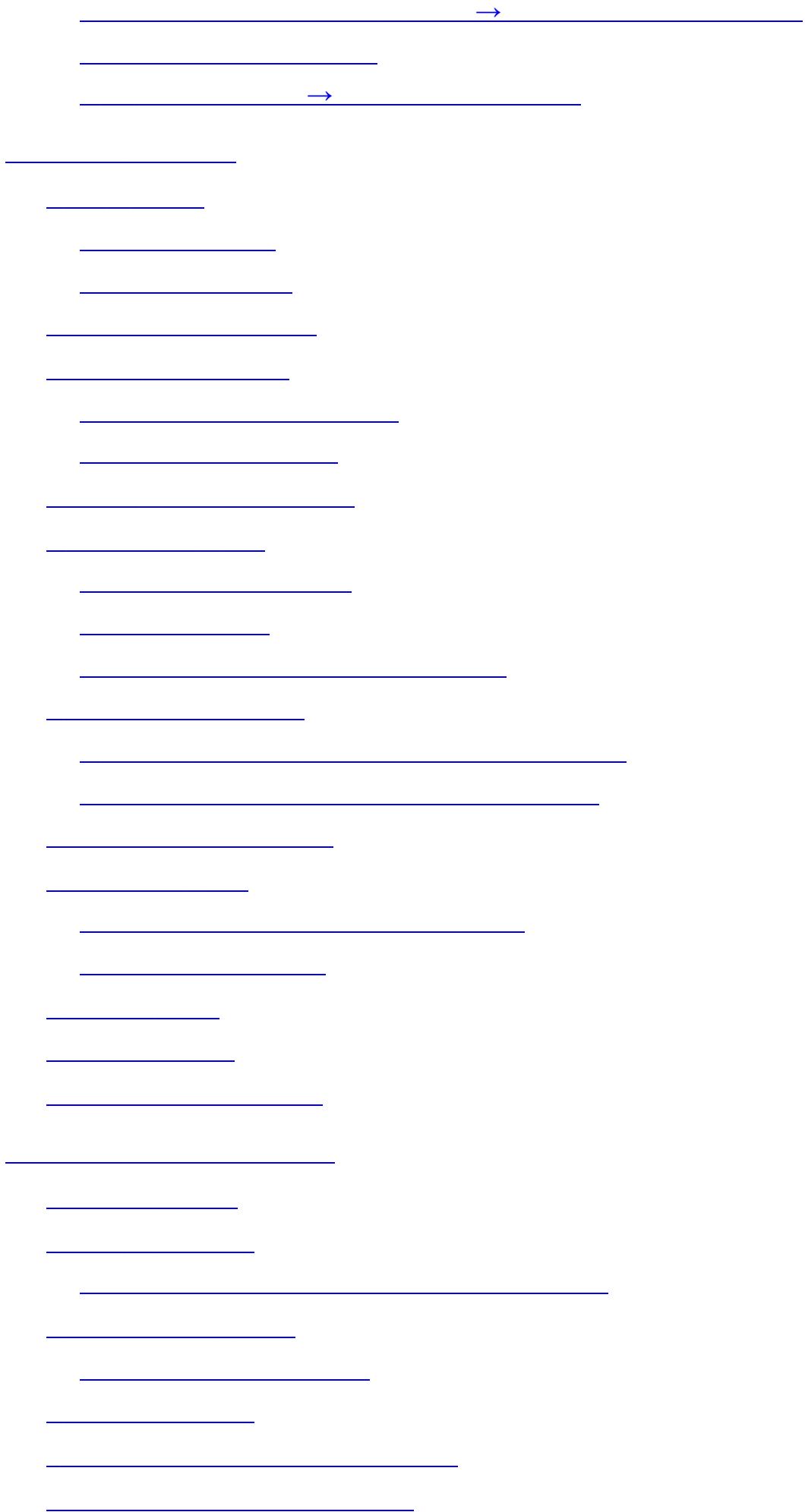


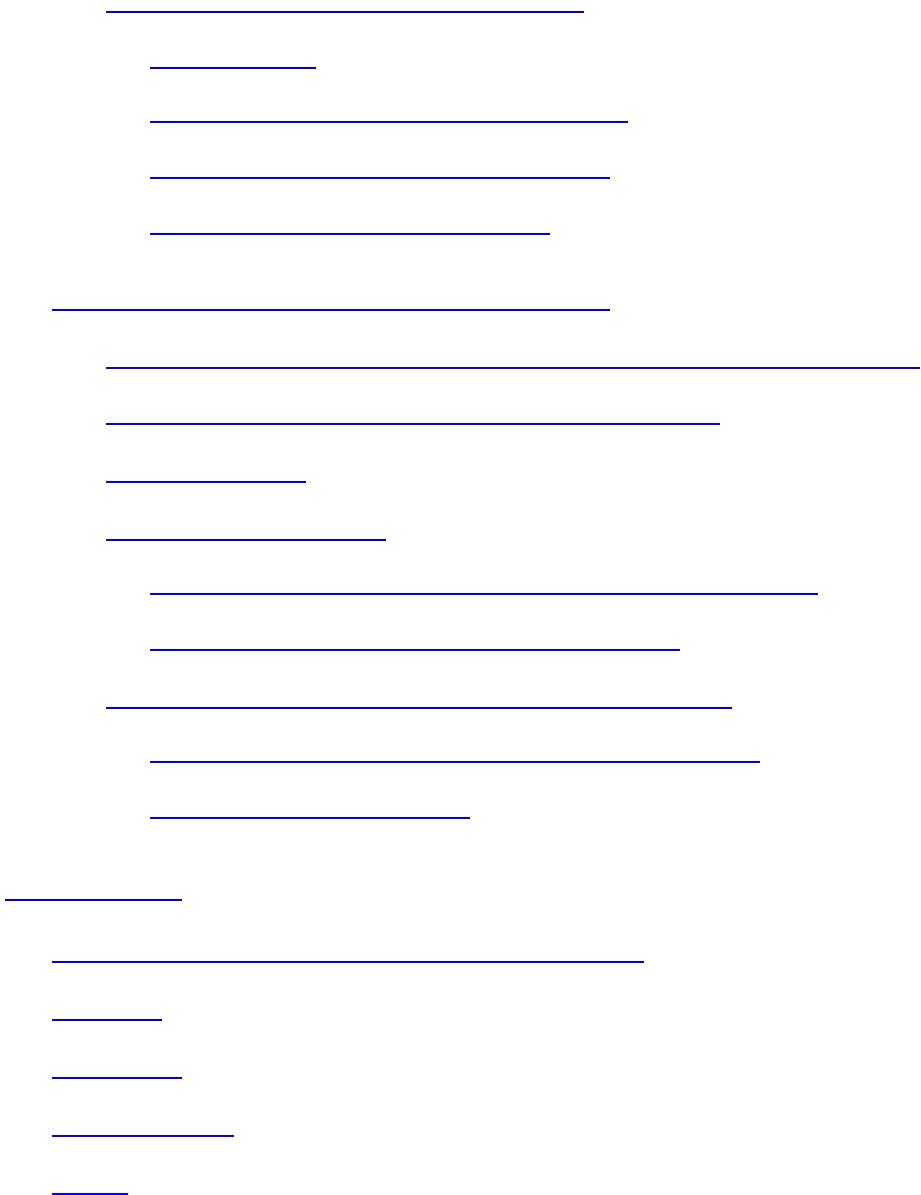






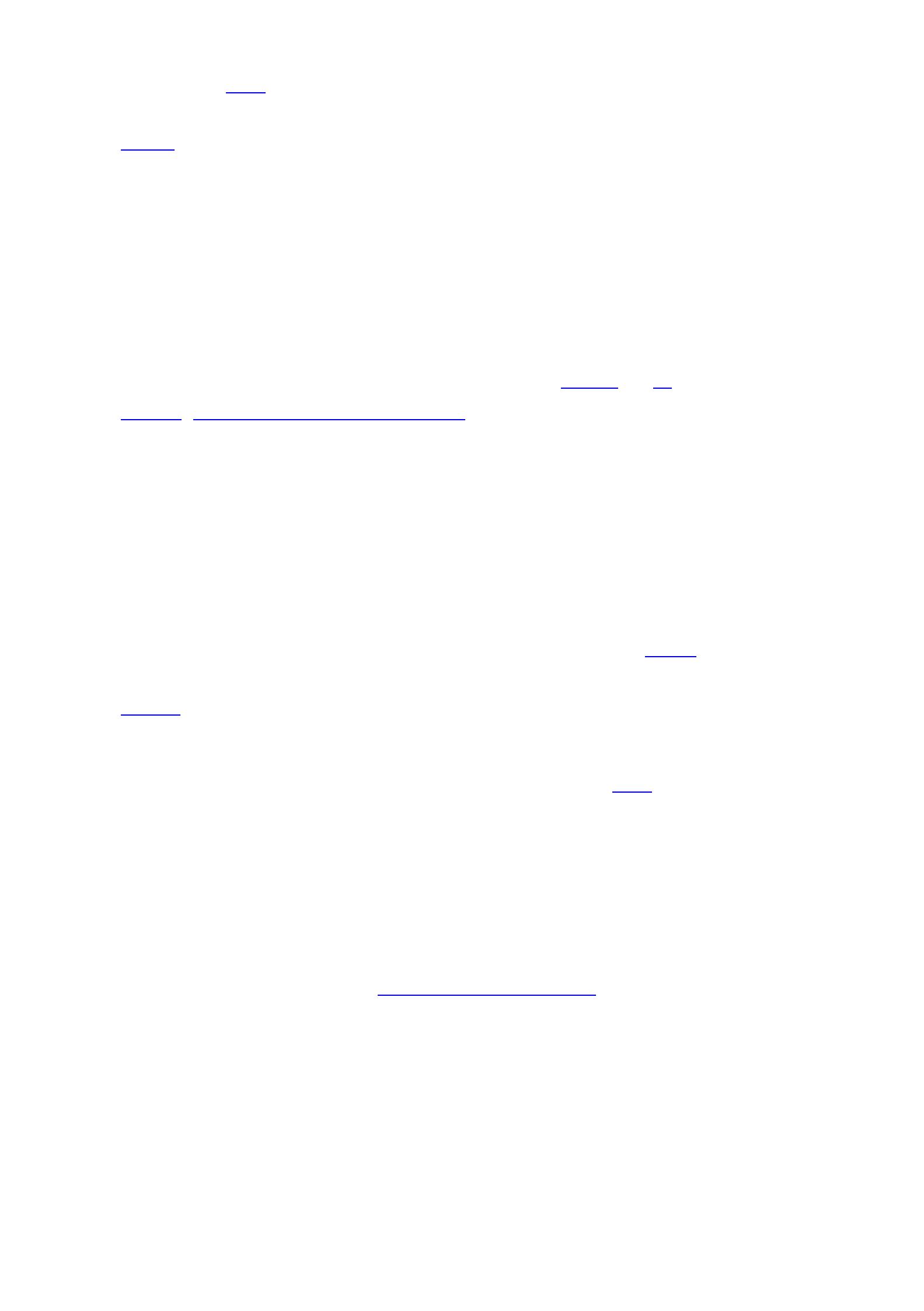








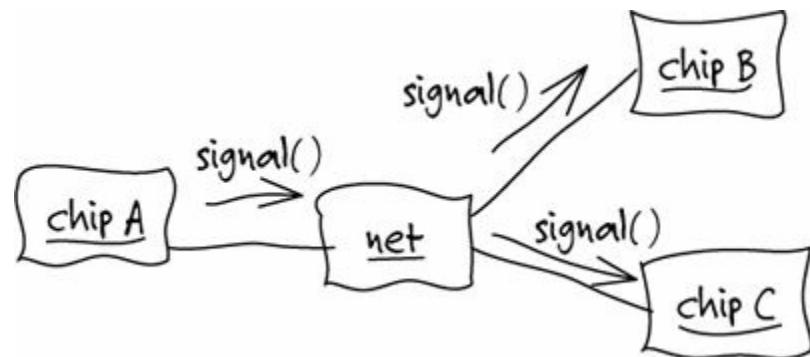


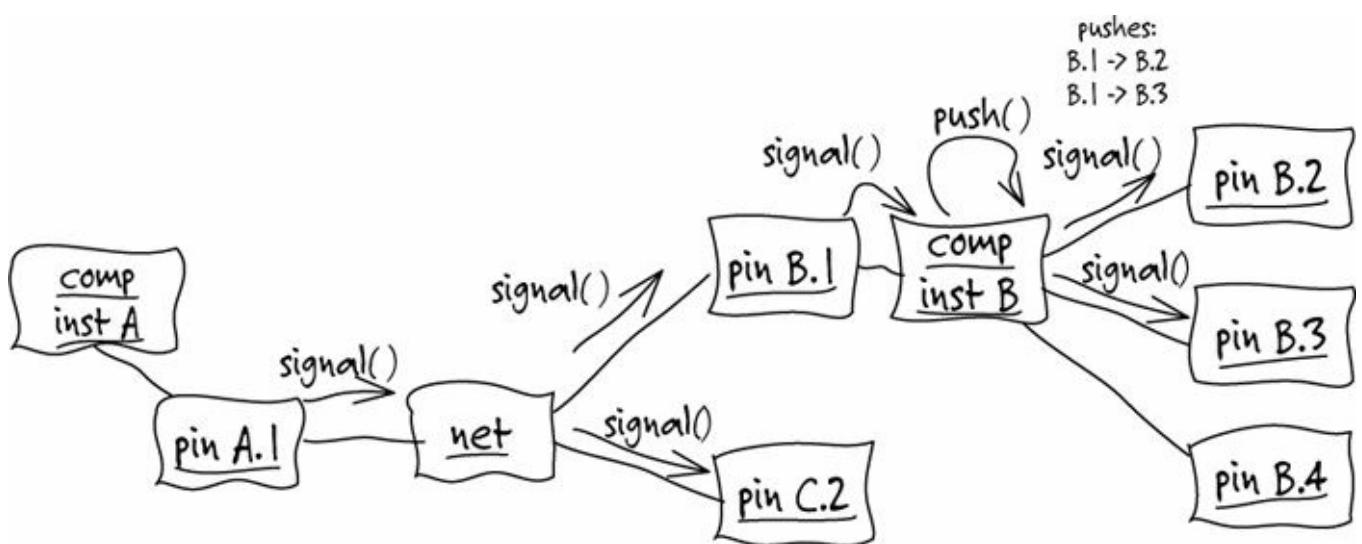
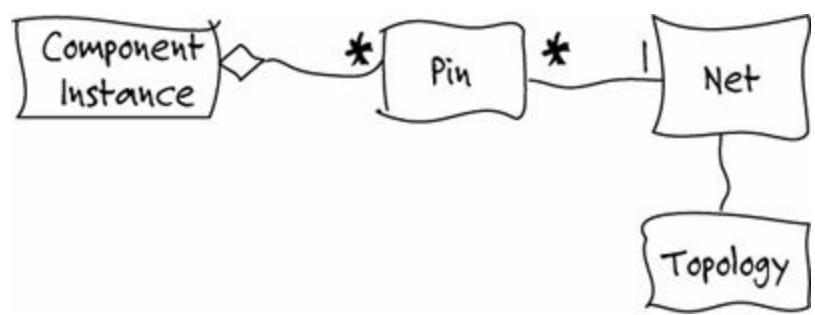


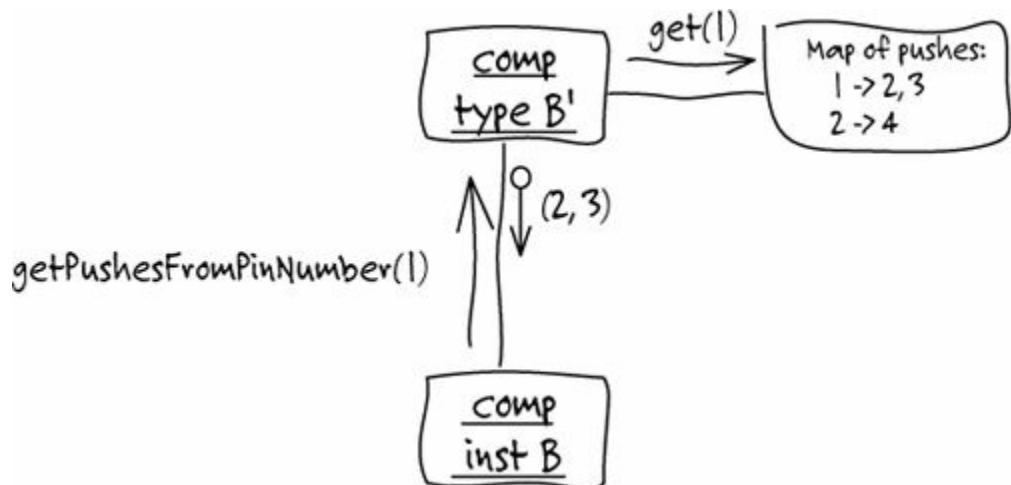
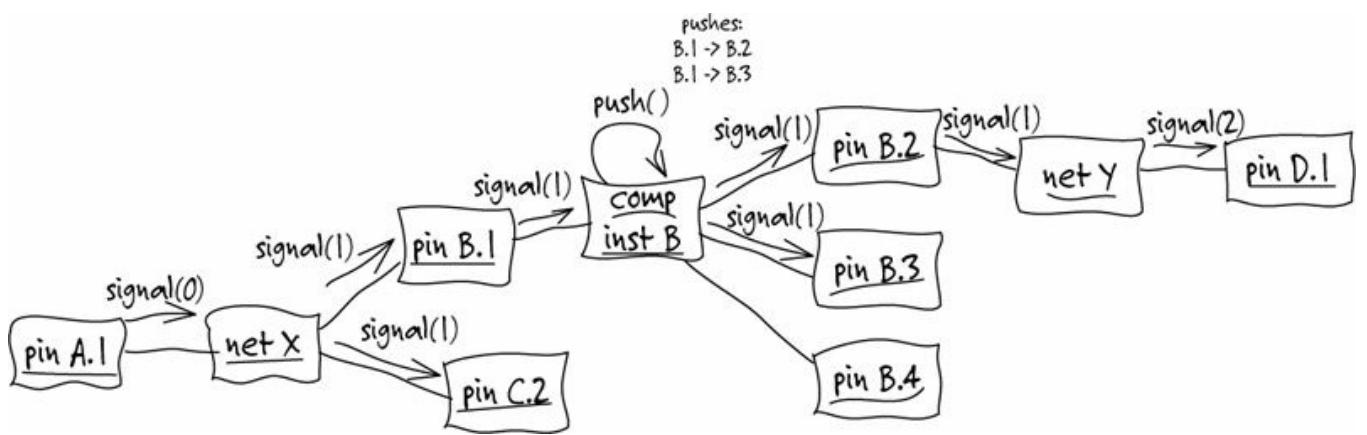
—

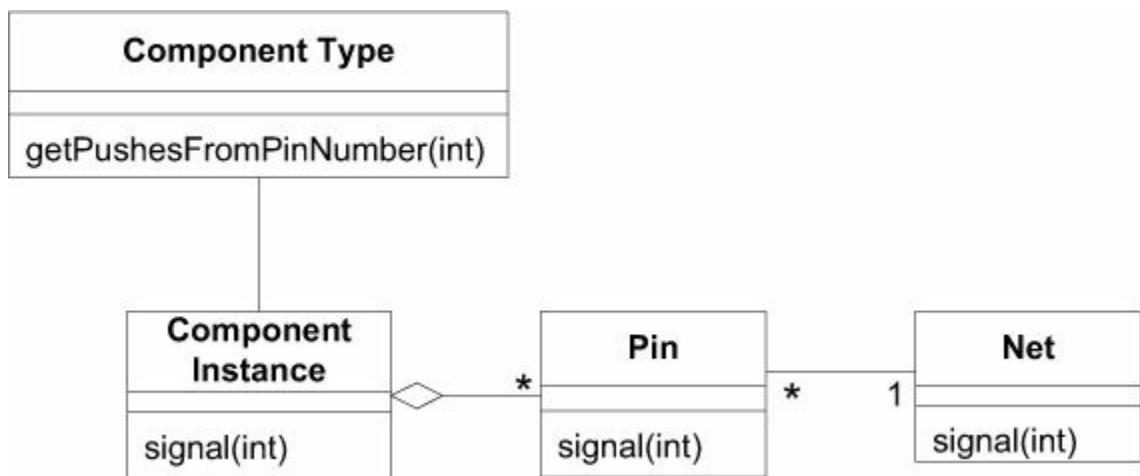
— —

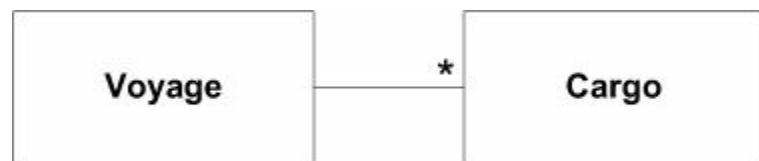










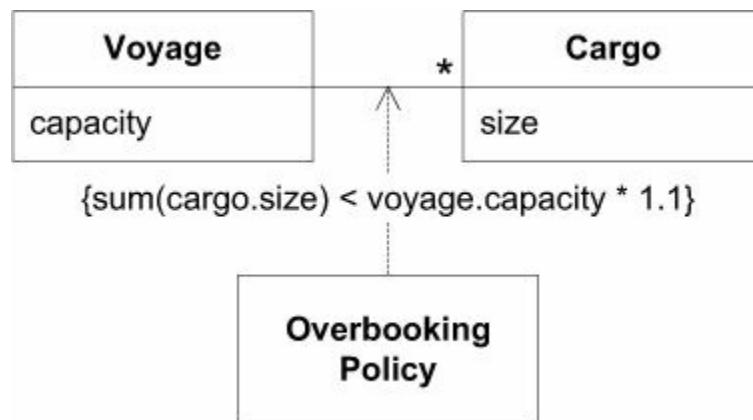


```
public int makeBooking(Cargo cargo, Voyage voyage) {  
    int confirmation = orderConfirmationSequence.next();  
    voyage.addCargo(cargo, confirmation);  
    return confirmation;  
}
```



```

public int makeBooking( Cargo cargo, Voyage voyage) {
    double maxBooking = voyage.capacity() * 1.1;
    if ((voyage.bookedCargoSize() + cargo.size()) > maxBooking)
        return -1;
    int confirmation = orderConfirmationSequence.next();
    voyage.addCargo(cargo, confirmation);
    return confirmation;
}
  
```



```
public int makeBooking(Cargo cargo, Voyage voyage) {  
    if (!overbookingPolicy.isAllowed(cargo, voyage)) return -1;  
    int confirmation = orderConfirmationSequence.next();  
    voyage.addCargo(cargo, confirmation);  
    return confirmation;  
}
```

```
public boolean isAllowed(Cargo cargo, Voyage voyage) {  
    return (cargo.size() + voyage.bookedCargoSize()) <=  
        (voyage.capacity() * 1.1);  
}
```

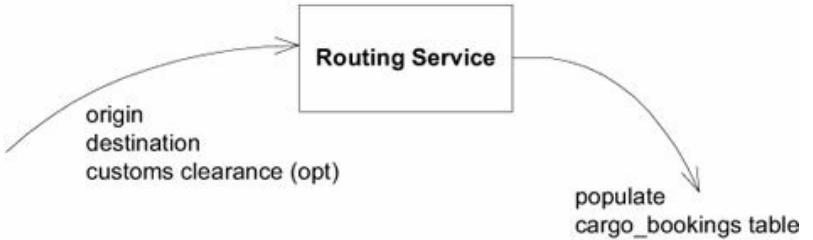
* * *



* * *

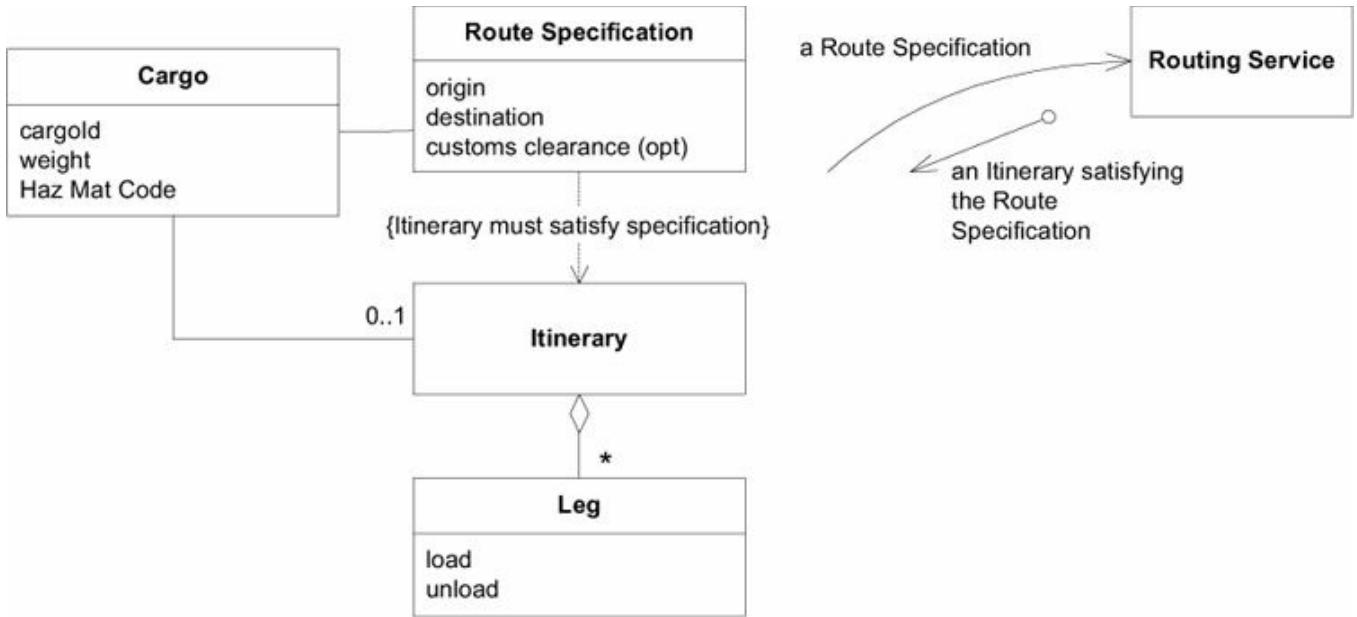


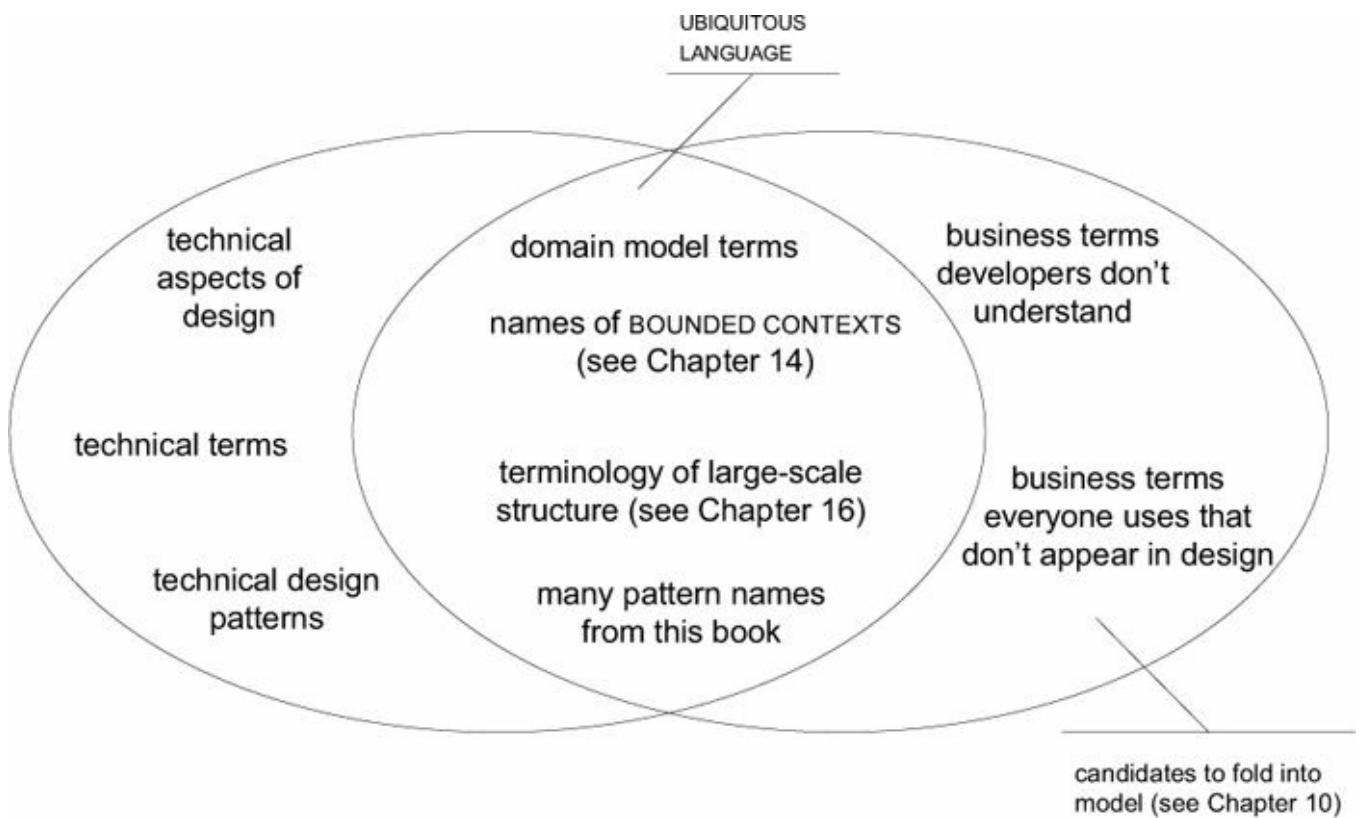
Cargo
cargold
origin
destination
customs clearance (opt)
weight
Haz Mat Code

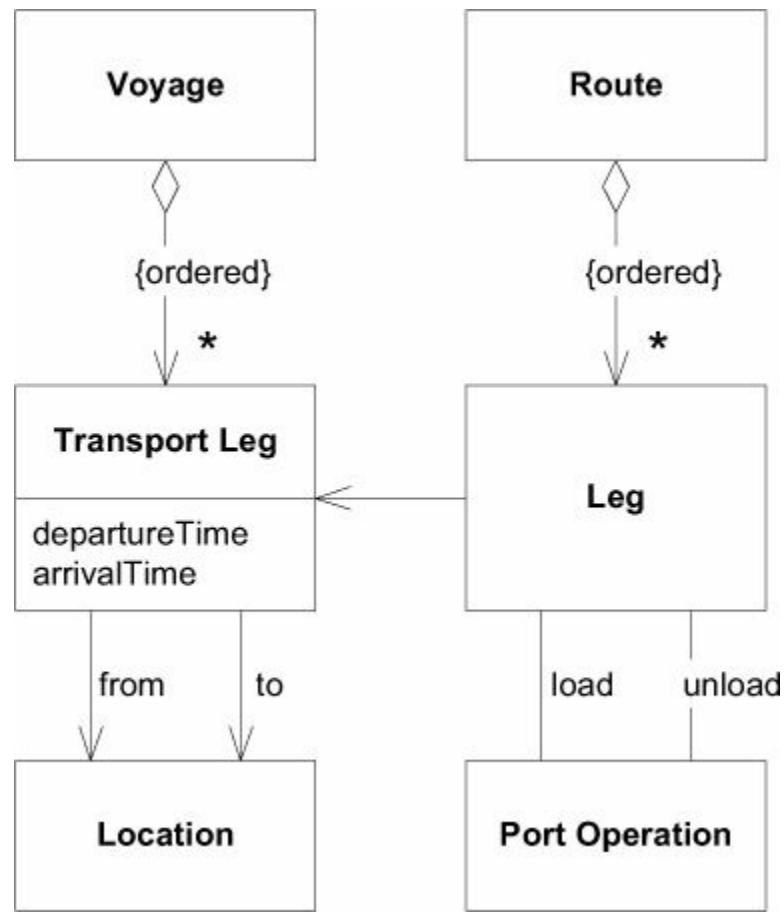


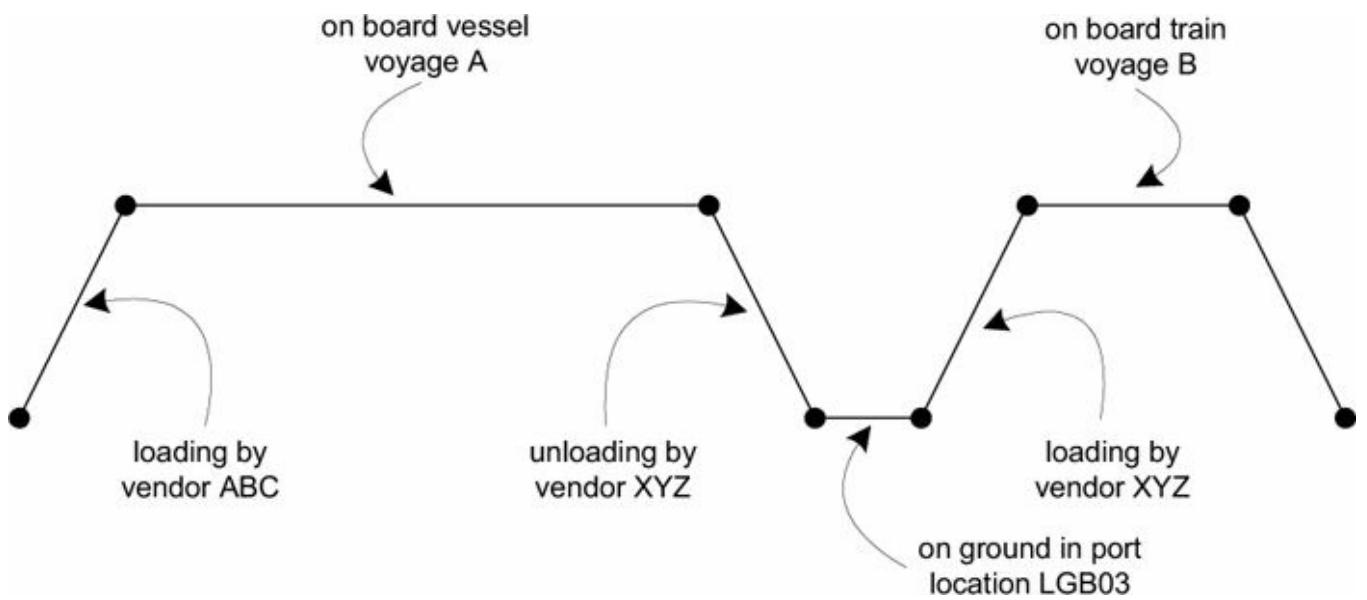
Database table: cargo_bookings

Cargo_ID	Transport	Load	Unload





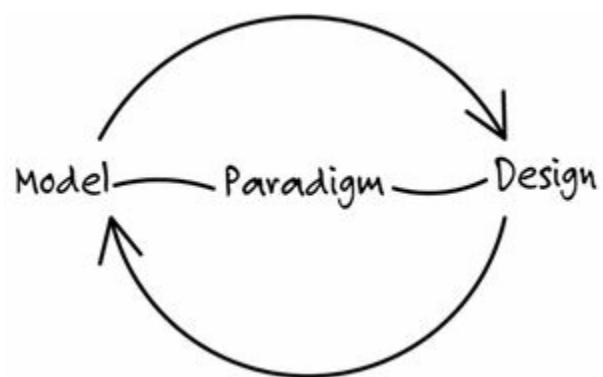


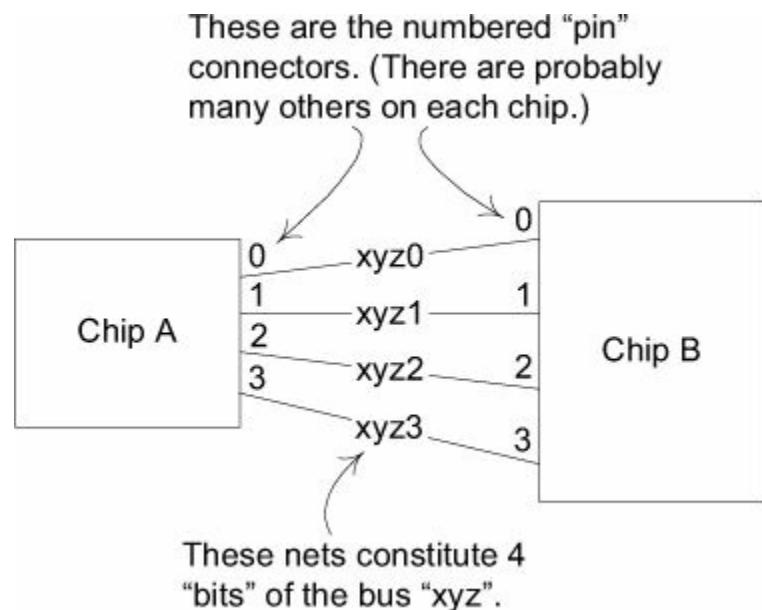




* * *

* * *





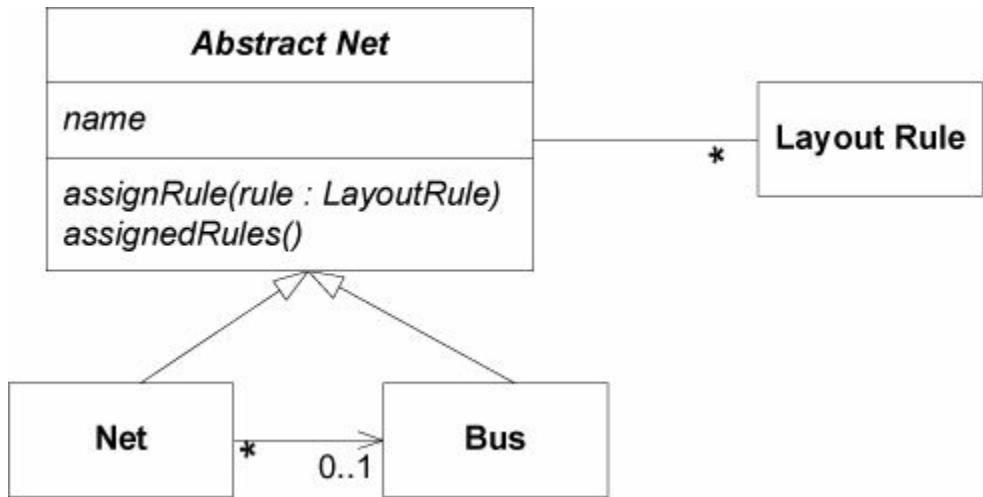
Net Name	Component.Pin
Xyz0	A. 0, B. 0
Xyz1	A. 1, B. 1
Xyz2	A. 2, B. 2
...	

Net Name	Rule Type	Parameters
Xyz1	min_linewidth	5
Xyz1	max_delay	15
Xyz2	min_linewidth	5
Xyz2	max_delay	15
...		

1. Sort net list file by net name.
2. Read each line in file, seeking first one that starts with bus name pattern.
3. For each line with matching name, parse line to get net name.
4. Append net name with rule text to rules file.
5. Repeat from 3 until left of line no longer matches bus name.

Bus Name	Rule Type	Parameters
Xyz	max_vias	3

Net Name	Rule Type	Parameters
Xyz0	max_vias	3
Xyz1	max_vias	3
Xyz2	max_vias	3
...		



assignRule()

assignedRules()

```

abstract class AbstractNet {
    private Set rules;

    void assignRule( LayoutRule rule) {
        rules.add( rule);
    }

    Set assignedRules() {
        return rules;
    }
}

class Net extends AbstractNet {
    private Bus bus;

    Set assignedRules() {
        Set result = new HashSet();
        result.addAll( super.assignedRules());
        result.addAll( bus.assignedRules());
        return result;
    }
}

```

Service	Responsibility
Net List import	Reads Net List file, creates instance of Net for each entry
Net Rule export	Given a collection of Nets, writes all attached rules into the Rules File

Class	Responsibility
Net Repository	Provides access to Nets by name
Inferred Bus Factory	Given a collection of Nets, uses naming conventions to infer Buses, creates instances
Bus Repository	Provides access to Buses by name

```

Collection nets = NetListImportService.read( aFile );
NetRepository.addAll( nets );
Collection buses = InferredBusFactory.groupIntoBuses( nets );
BusRepository.addAll( buses );

public void testBusRuleAssignment() {
    Net a0 = new Net("a0");
    Net a1 = new Net("a1");
    Bus a = new Bus("a"); //Bus is not conceptually dependent
    a.addNet(a0);           //on name-based recognition, and so
    a.addNet(a1);           //its tests should not be either.

    NetRule minWidth4 = NetRule.create( MIN_WIDTH, 4 );
    a.assignRule( minWidth4 );

    assertTrue( a0.assignedRules().contains( minWidth4 ) );
    assertEquals( minWidth4, a0.getRule( MIN_WIDTH ) );
    assertEquals( minWidth4, a1.getRule( MIN_WIDTH ) );
}

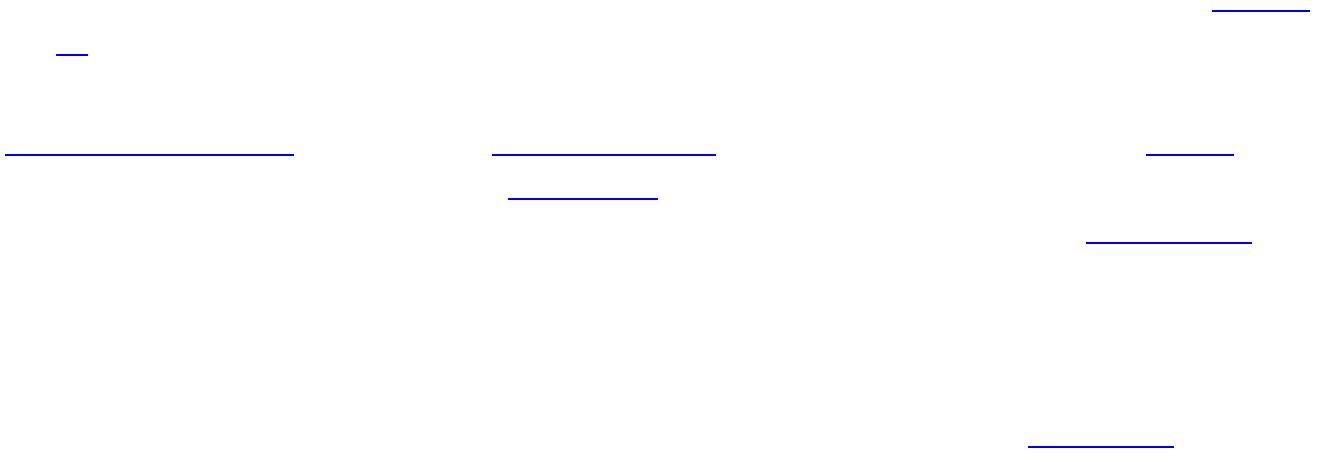
public void assignBusRule( String busName, String ruleType,
    double parameter ){
    Bus bus = BusRepository.getByName( busName );
    bus.assignRule( NetRule.create( ruleType, parameter ) );
}

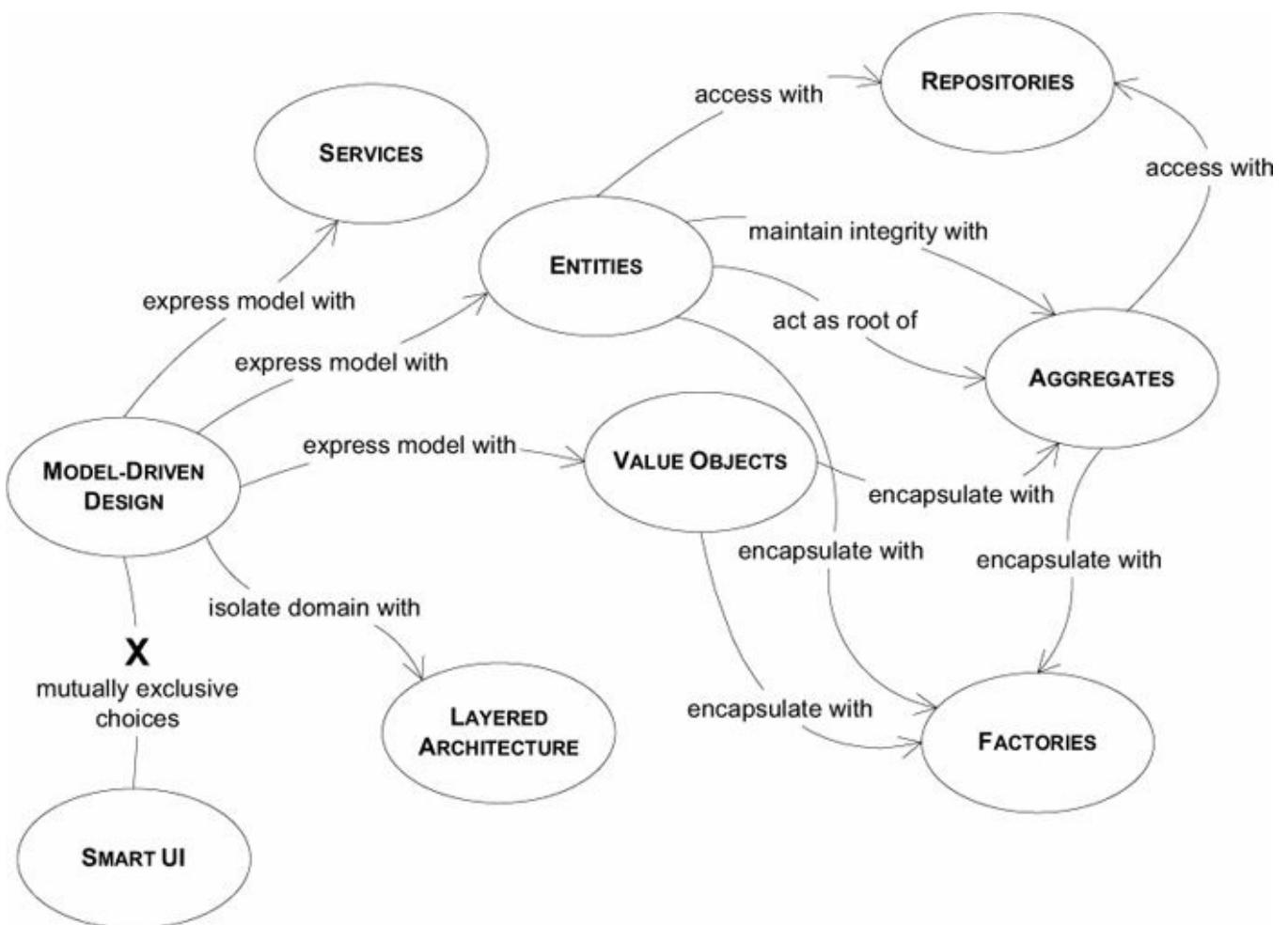
NetRuleExport.write( aFileName, NetRepository.allNets() );
    assignedRules()

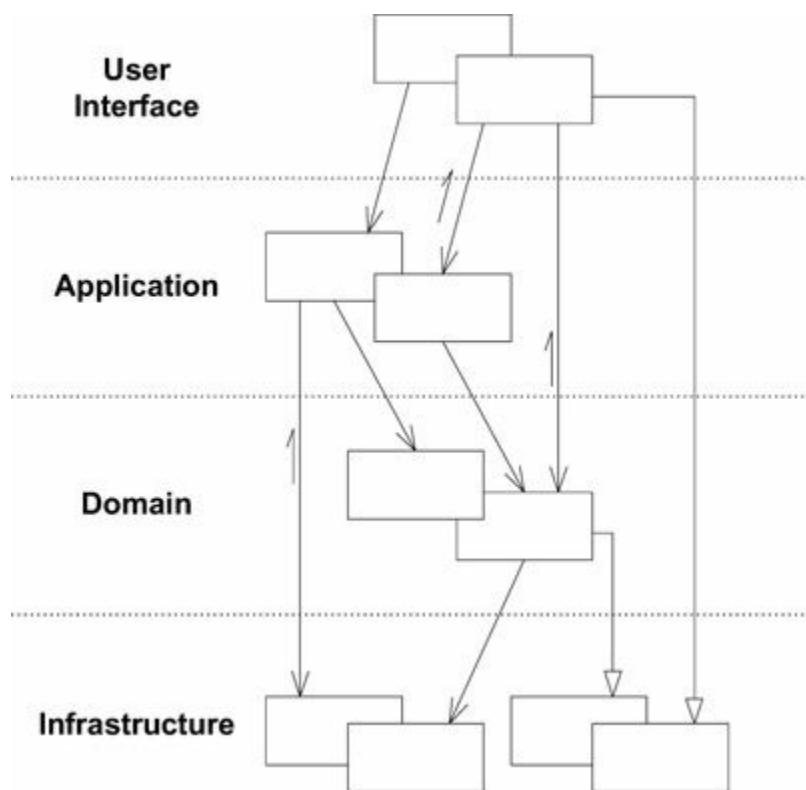
```

-

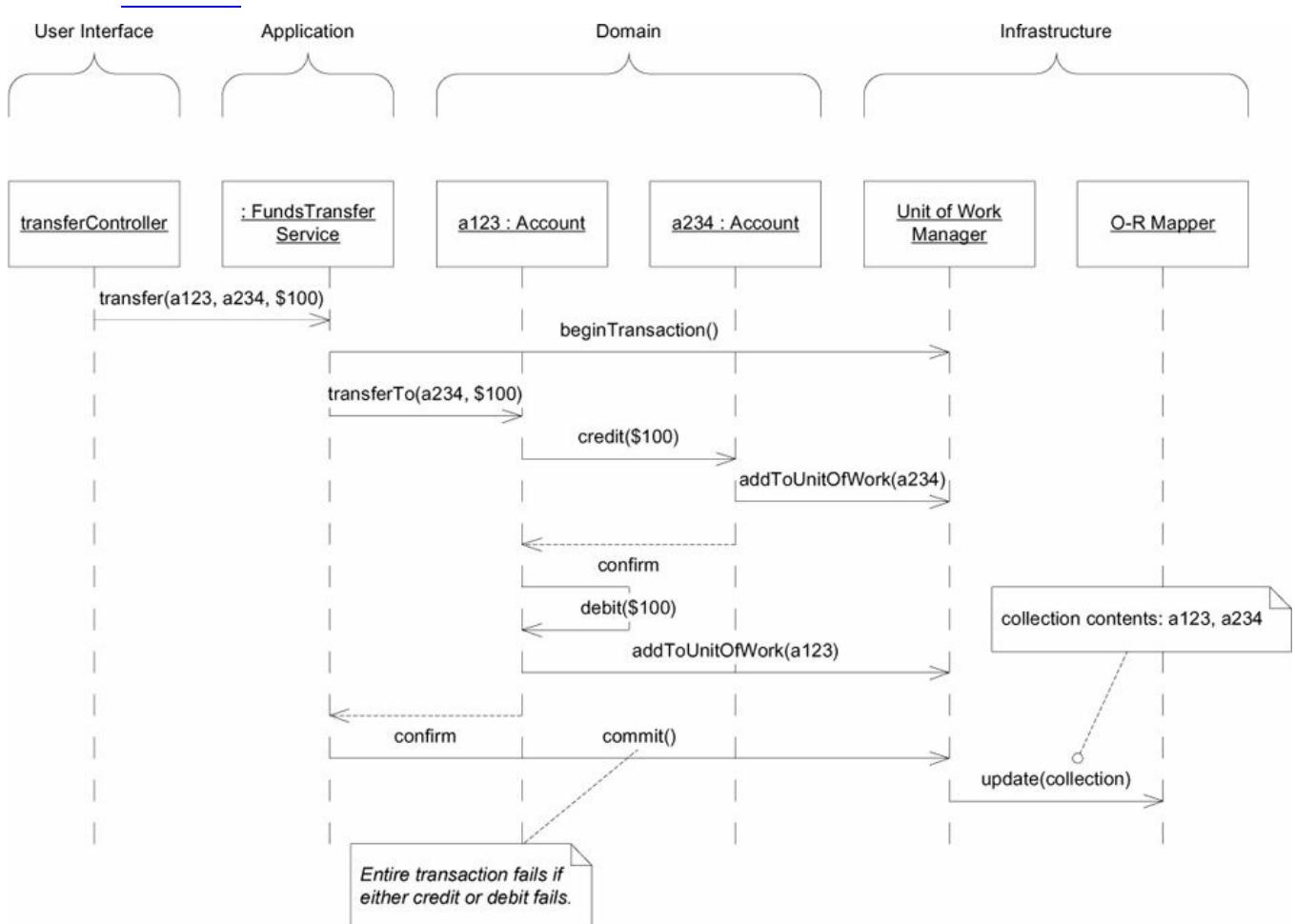
* * *





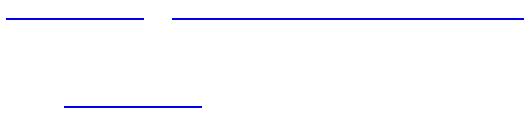


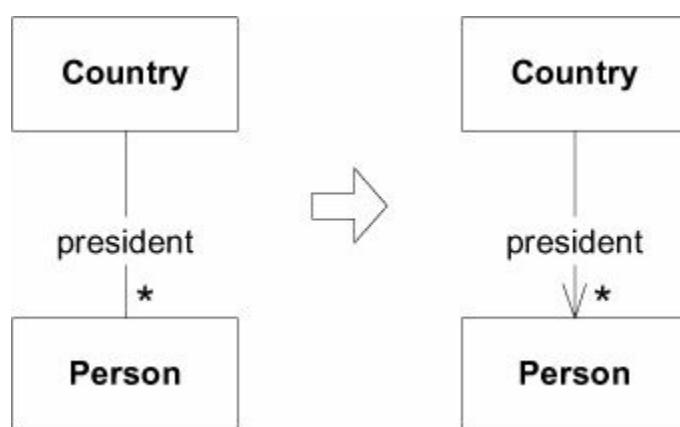
User Interface (or Presentation Layer)	Responsible for showing information to the user and interpreting the user's commands. The external actor might sometimes be another computer system rather than a human user.
Application Layer	<p>Defines the jobs the software is supposed to do and directs the expressive domain objects to work out problems. The tasks this layer is responsible for are meaningful to the business or necessary for interaction with the application layers of other systems.</p> <p>This layer is kept thin. It does not contain business rules or knowledge, but only coordinates tasks and delegates work to collaborations of domain objects in the next layer down. It does not have state reflecting the business situation, but it can have state that reflects the progress of a task for the user or the program.</p>
Domain Layer (or Model Layer)	<p>Responsible for representing concepts of the business, information about the business situation, and business rules. State that reflects the business situation is controlled and used here, even though the technical details of storing it are delegated to the infrastructure.</p> <p><i>This layer is the heart of business software.</i></p>
Infrastructure Layer	Provides generic technical capabilities that support the higher layers: message sending for the application, persistence for the domain, drawing widgets for the UI, and so on. The infrastructure layer may also support the pattern of interactions between the four layers through an architectural framework.

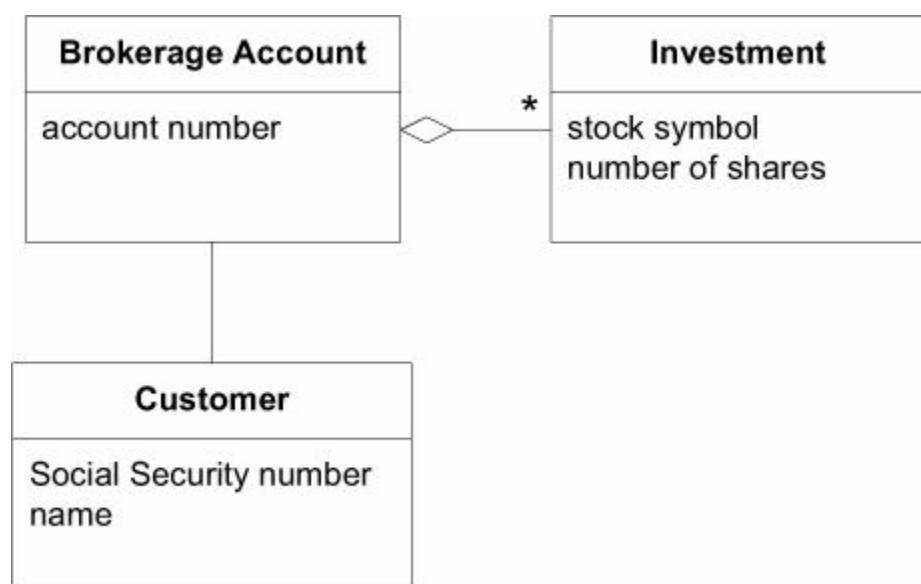
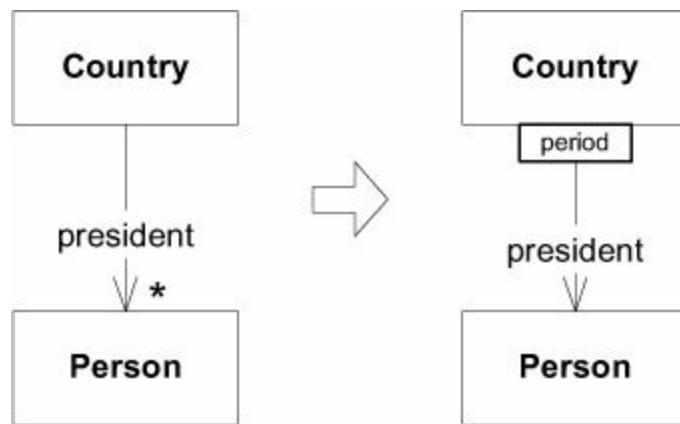


* * *

* * *







```

public class BrokerageAccount {
    String accountNumber;
    Customer customer;
    Set investments;
  
```

```

// Constructors, etc. omitted

public Customer getCustomer() {
    return customer;
}
public Set getInvestments() {
    return investments;
}
}

```

ACCOUNT_NUMBER	CUSTOMER_SS_NUMBER

SS_NUMBER	NAME

ACCOUNT_NUMBER	STOCK_SYMBOL	AMOUNT

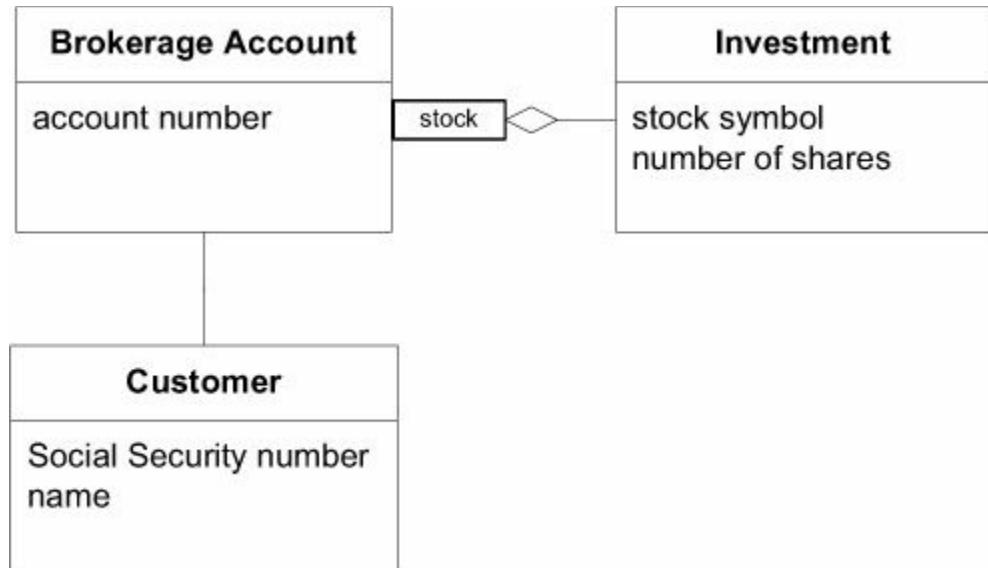
```

public class BrokerageAccount {
    String accountNumber;
    String customerSocialSecurityNumber;

    // Omit constructors, etc.

    public Customer getCustomer() {
        String sqlQuery =
            "SELECT * FROM CUSTOMER WHERE" +
            "SS_NUMBER=' "+customerSocialSecurityNumber+"'";
        return QueryService.findSingleCustomerFor(sqlQuery);
    }
    public Set getInvestments() {
        String sqlQuery =
            "SELECT * FROM INVESTMENT WHERE" +
            "BROKERAGE_ACCOUNT=' "+accountNumber+"'";
        return QueryService.findInvestmentsFor(sqlQuery);
    }
}

```



```

public class BrokerageAccount {
    String accountNumber;
    Customer customer;
    Map investments;

    // Omitting constructors, etc.

    public Customer getCustomer() {
        return customer;
    }
    public Investment getInvestment( String stockSymbol) {
        return (Investment)investments.get(stockSymbol);
    }
}

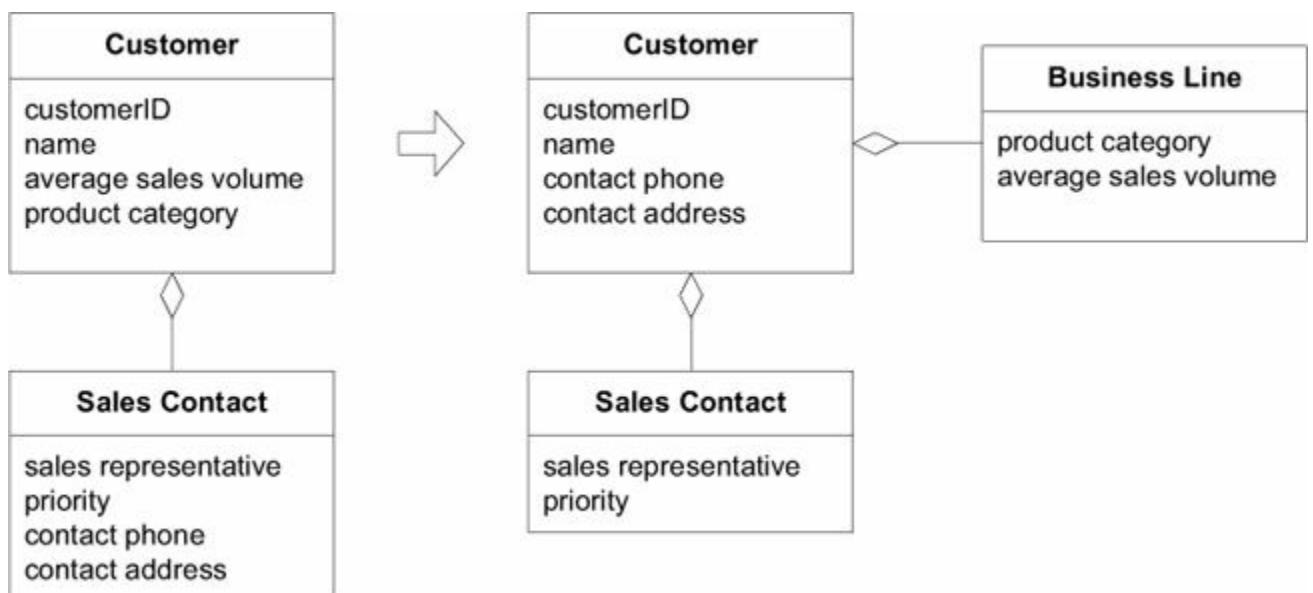
public class BrokerageAccount {
    String accountNumber;
    String customerSocialSecurityNumber;

    //Omitting constructors, etc.
    public Customer getCustomer() {
        String sqlQuery = "SELECT * FROM CUSTOMER WHERE SS_NUMBER='"
            + customerSocialSecurityNumber + "'";
        return QueryService.findSingleCustomerFor(sqlQuery);
    }
    public Investment getInvestment( String stockSymbol) {
        String sqlQuery = "SELECT * FROM INVESTMENT "
            + "WHERE BROKERAGE_ACCOUNT=' " + accountNumber + "'"
            + "AND STOCK_SYMBOL=' " + stockSymbol + "'";
        return QueryService.findInvestmentFor(sqlQuery);
    }
}
  
```

}{



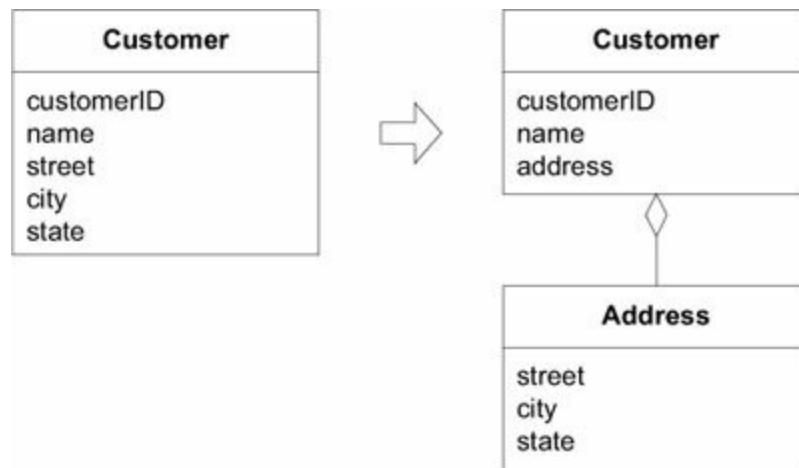
* * *





* * *

-



* * *



* * *

* * *

Application	<i>Funds Transfer App Service</i> <ul style="list-style-type: none">• Digests input (such as an XML request).• Sends message to domain service for fulfillment.• Listens for confirmation.• Decides to send notification using infrastructure service.
Domain	<i>Funds Transfer Domain Service</i> <ul style="list-style-type: none">• Interacts with necessary Account and Ledger objects, making appropriate debits and credits.• Supplies confirmation of result (transfer allowed or not, and so on).
Infrastructure	<i>Send Notification Service</i> <ul style="list-style-type: none">• Sends e-mails, letters, and other communications as directed by the application.

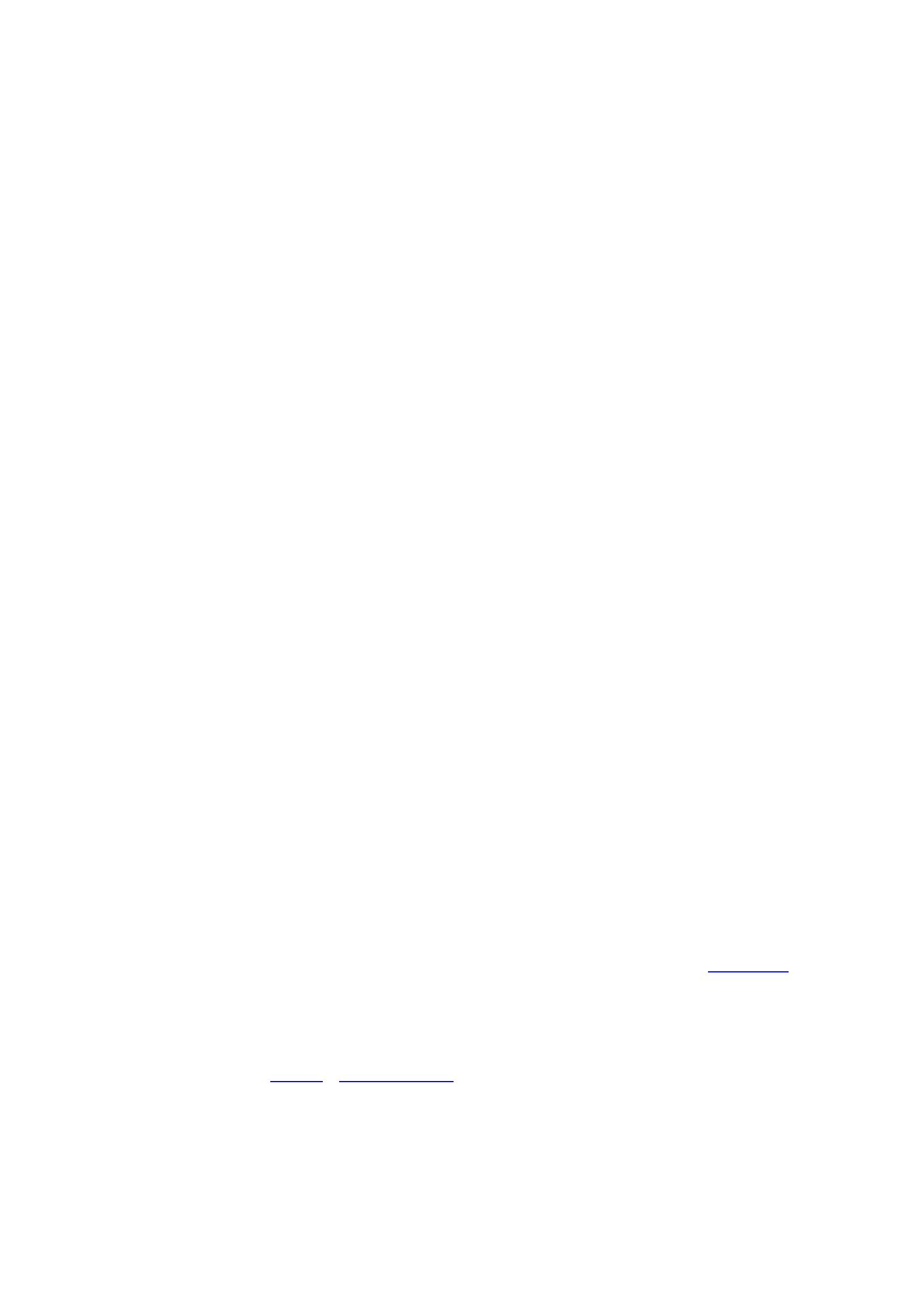
* * *

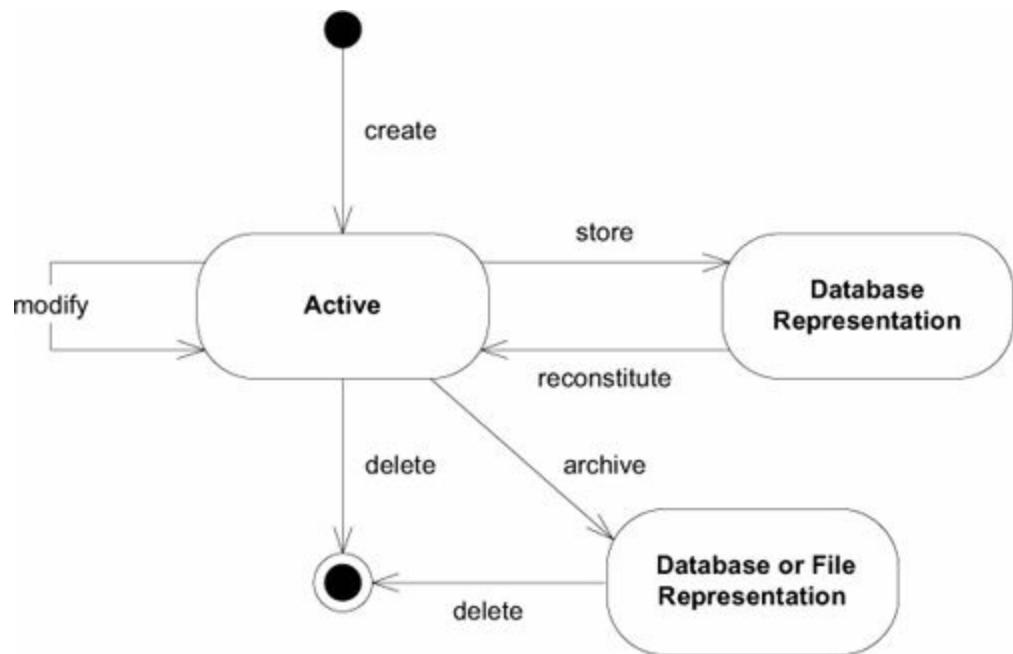


* * *

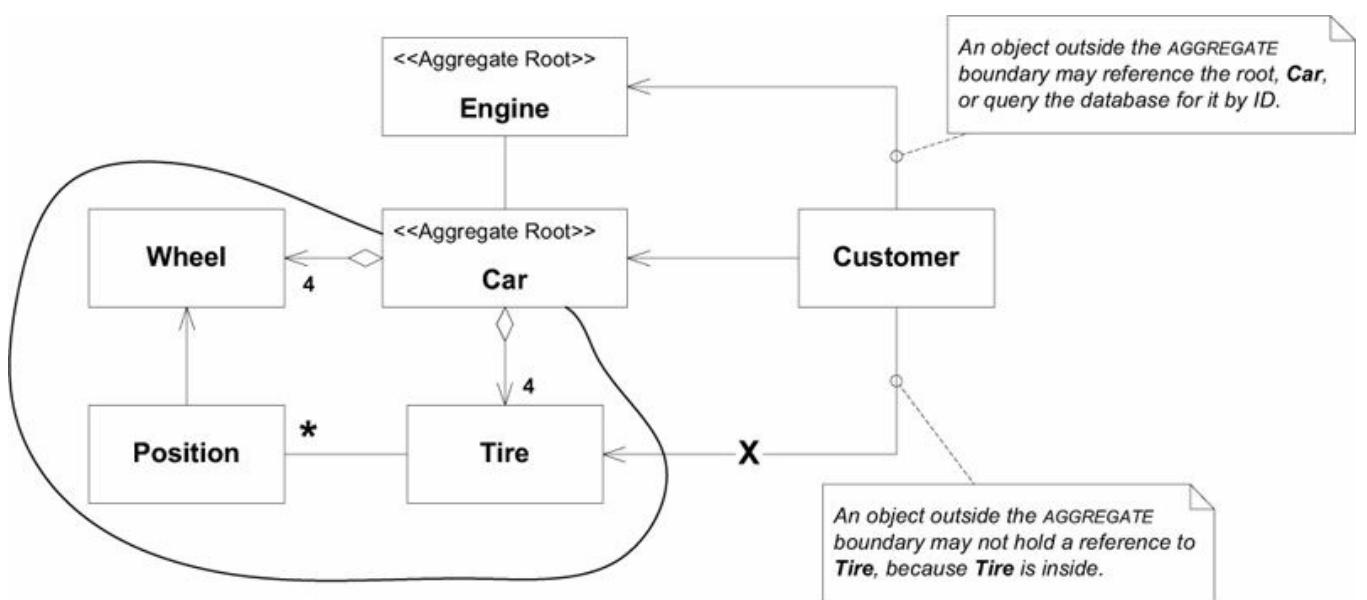
```
ClassA1
import packageB.ClassB1;
import packageB.ClassB2;
import packageB.ClassB3;
import packageC.ClassC1;
import packageC.ClassC2;
import packageC.ClassC3;
. . .
```

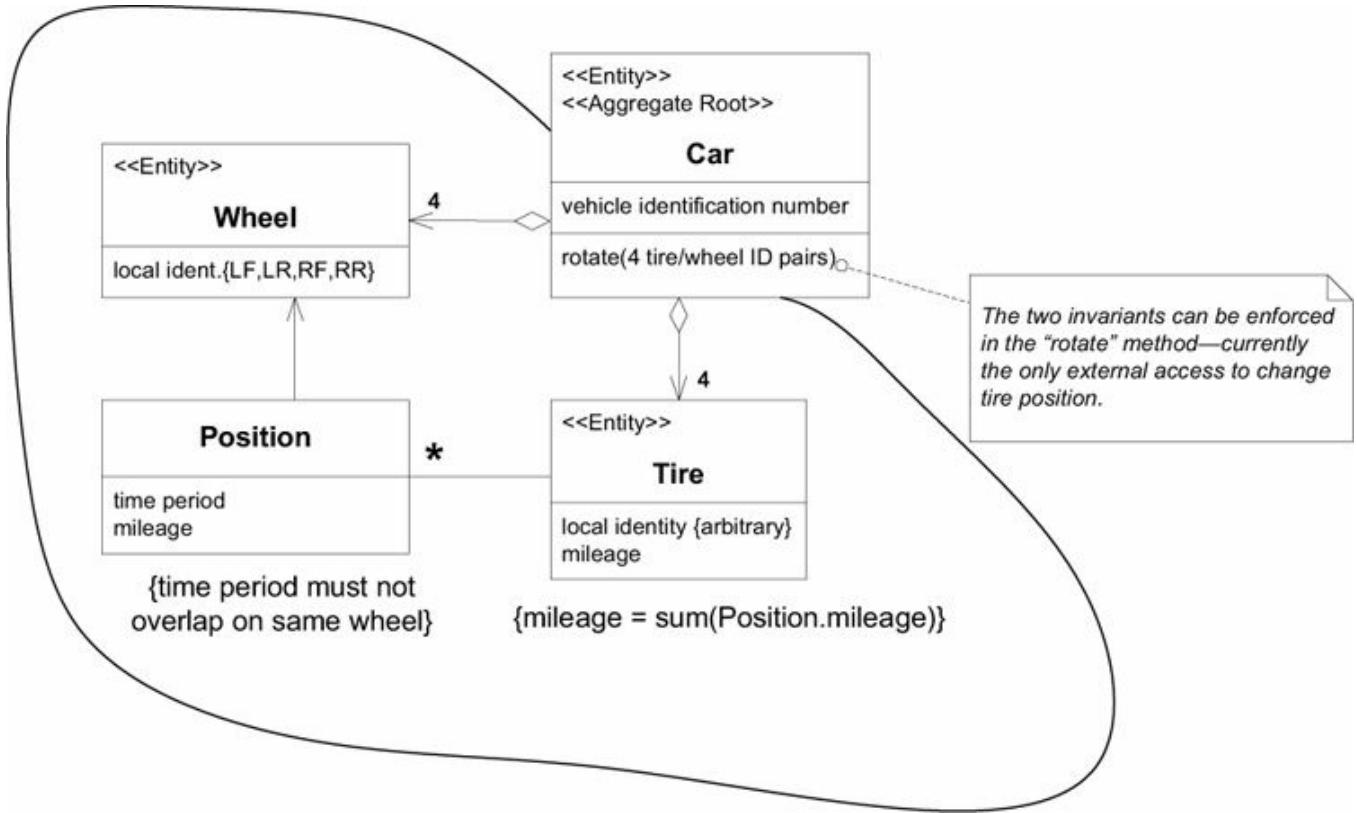
```
ClassA1  
import packageB.*;  
import packageC.*;  
. . .
```

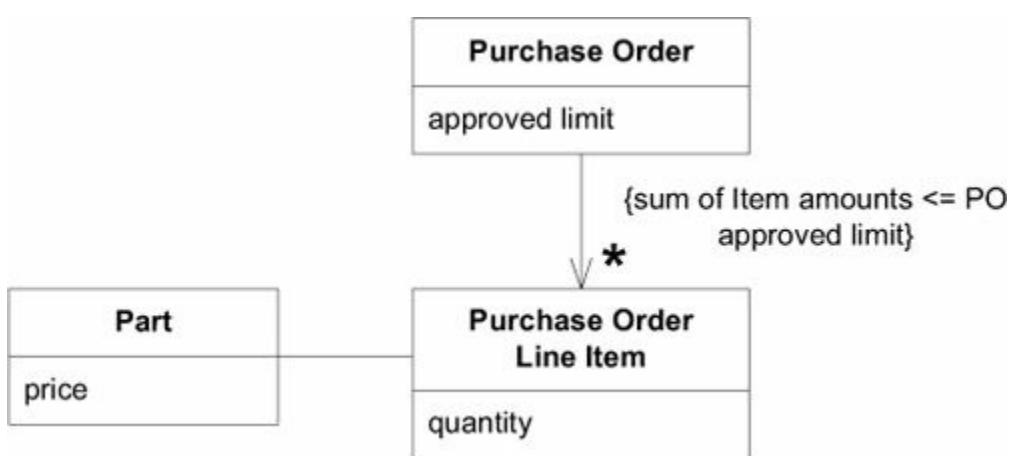












PO #0012946 Approved Limit: \$1,000.00				
Item #	Quantity	Part	Price	Amount
001	3	Guitars	@ 100.00	300.00
002	2	Trombones	@ 200.00	400.00
			Total:	700.00

001	3	Guitars	@ 100.00	300.00
002	2	Trombones	@ 200.00	400.00
			Total:	700.00

George adds guitars in his view

PO #0012946 Approved Limit: \$1000.00				
Item #	Quantity	Part	Price	Amount
001	5	Guitars	@ 100.00	500.00
002	2	Trombones	@ 200.00	400.00
			Total:	900.00

001	5	Guitars	@ 100.00	500.00
002	2	Trombones	@ 200.00	400.00
			Total:	900.00

Amanda adds a trombone in her view

PO #0012946 Approved Limit: \$1,000.00				
Item #	Quantity	Part	Price	Amount
001	3	Guitars	@ 100.00	300.00
002	3	Trombones	@ 200.00	600.00
			Total:	900.00

001	3	Guitars	@ 100.00	300.00
002	3	Trombones	@ 200.00	600.00
			Total:	900.00

PO #0012946 Approved Limit: \$1,000.00				
Item #	Quantity	Part	Price	Amount
001	5	Guitars	@ 100.00	500.00
002	3	Trombones	@ 200.00	600.00
			Total:	1,100.00

001	5	Guitars	@ 100.00	500.00
002	3	Trombones	@ 200.00	600.00
			Total:	1,100.00

George edits his view**Amanda is locked out of PO #0012946**

PO #0012946 Approved Limit: \$1000.00				
Item #	Quantity	Part	Price	Amount
001	5	Guitars	@ 100.00	500.00
002	2	Trombones	@ 200.00	400.00
Total:				900.00

George's changes have been committed**Amanda gets access; George's change shows**

PO #0012946 Approved Limit: \$1000.00				
Item #	Quantity	Part	Price	Amount
001	5	Guitars	@ 100.00	500.00
002	3	Trombones	@ 200.00	600.00
Limit exceeded →				Total: 1,100.00

George editing PO**Guitars and trombones locked****PO #0012946 Approved Limit: \$1,000.00**

Item #	Quantity	Part	Price	Amount
001	2	Guitars	@ 100.00	200.00
002	2	Trombones	@ 200.00	400.00
Total:				600.00

Amanda adds trombones; must wait on George**Violins locked****PO #0012932 Approved Limit: \$1,850.00**

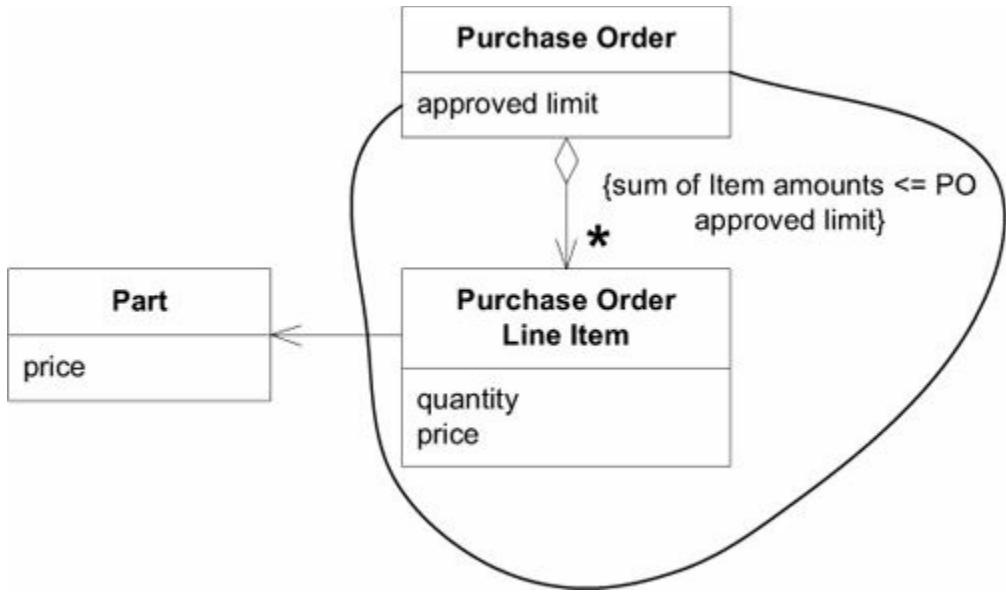
Item #	Quantity	Part	Price	Amount
001	3	Violins	@ 400.00	1,200.00
002	2	Trombones	@ 200.00	400.00
Total:				1,600.00

Sam adds trombones; must wait on George**PO #0013003 Approved Limit: \$15,000.00**

Item #	Quantity	Part	Price	Amount
001	1	Piano	@ 1,000.00	1,000.00
002	2	Trombones	@ 200.00	400.00
Total:				1,400.00

George adds violins; must wait on Amanda (!)**PO #0012946 Approved Limit: \$1,000.00**

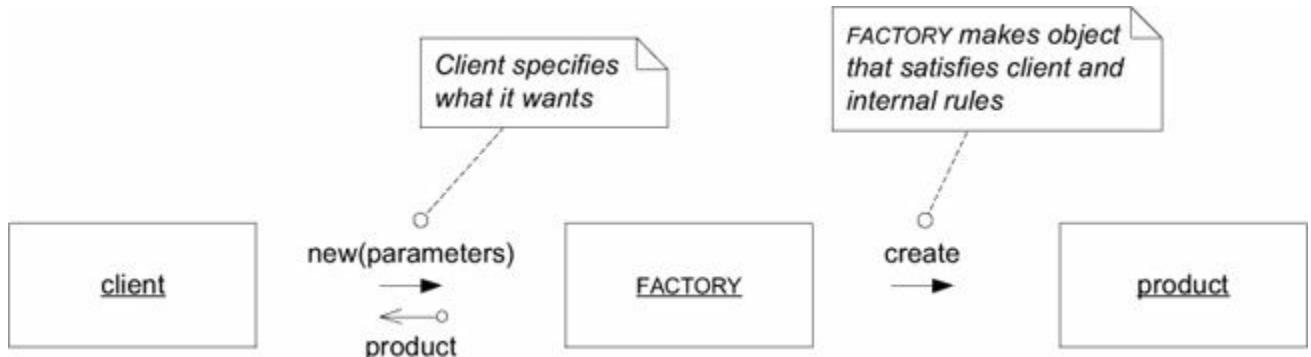
Item #	Quantity	Part	Price	Amount
001	2	Guitars	@ 100.00	200.00
002	2	Trombones	@ 200.00	400.00
003	1	Violins	@ 400.00	400.00
Total:				1,000.00

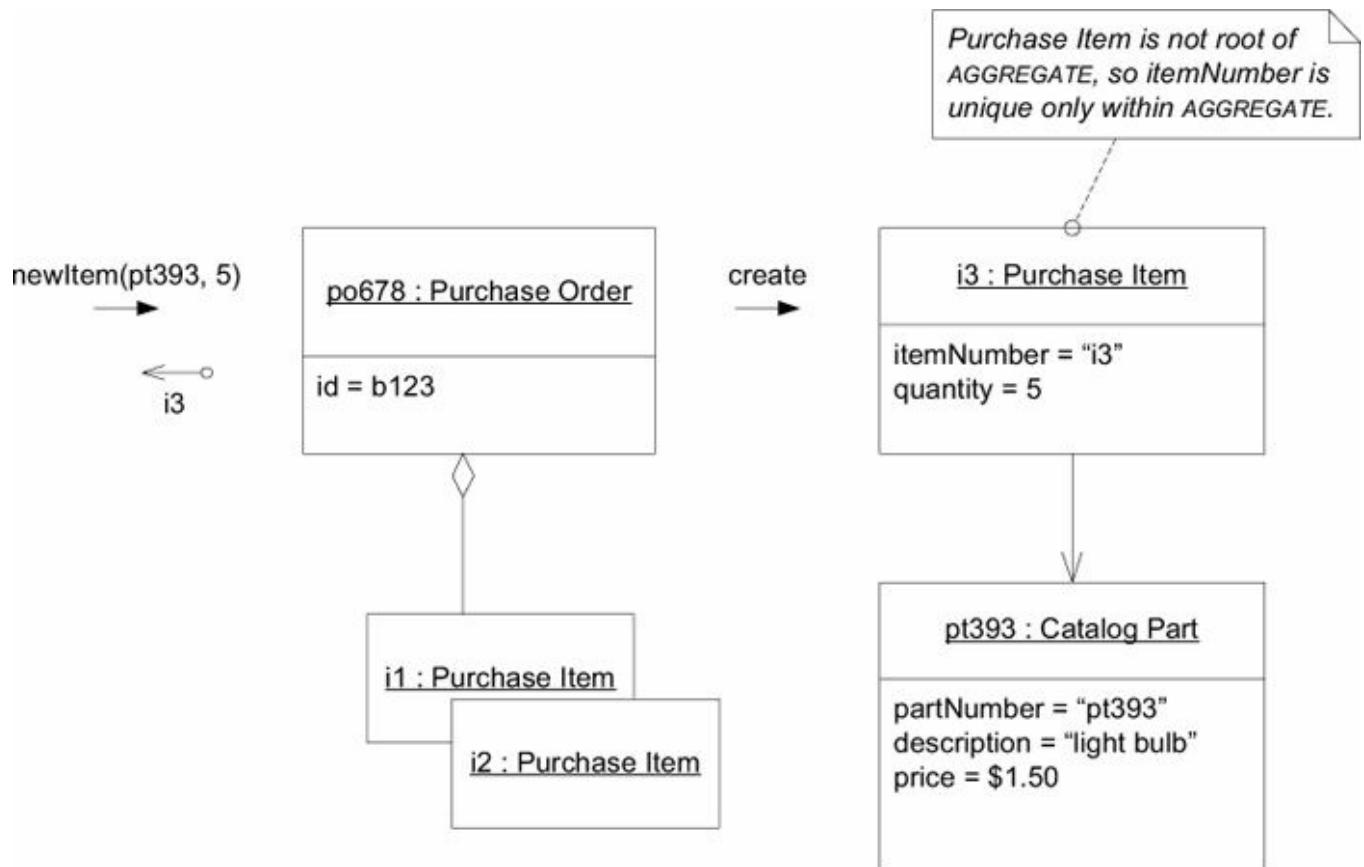


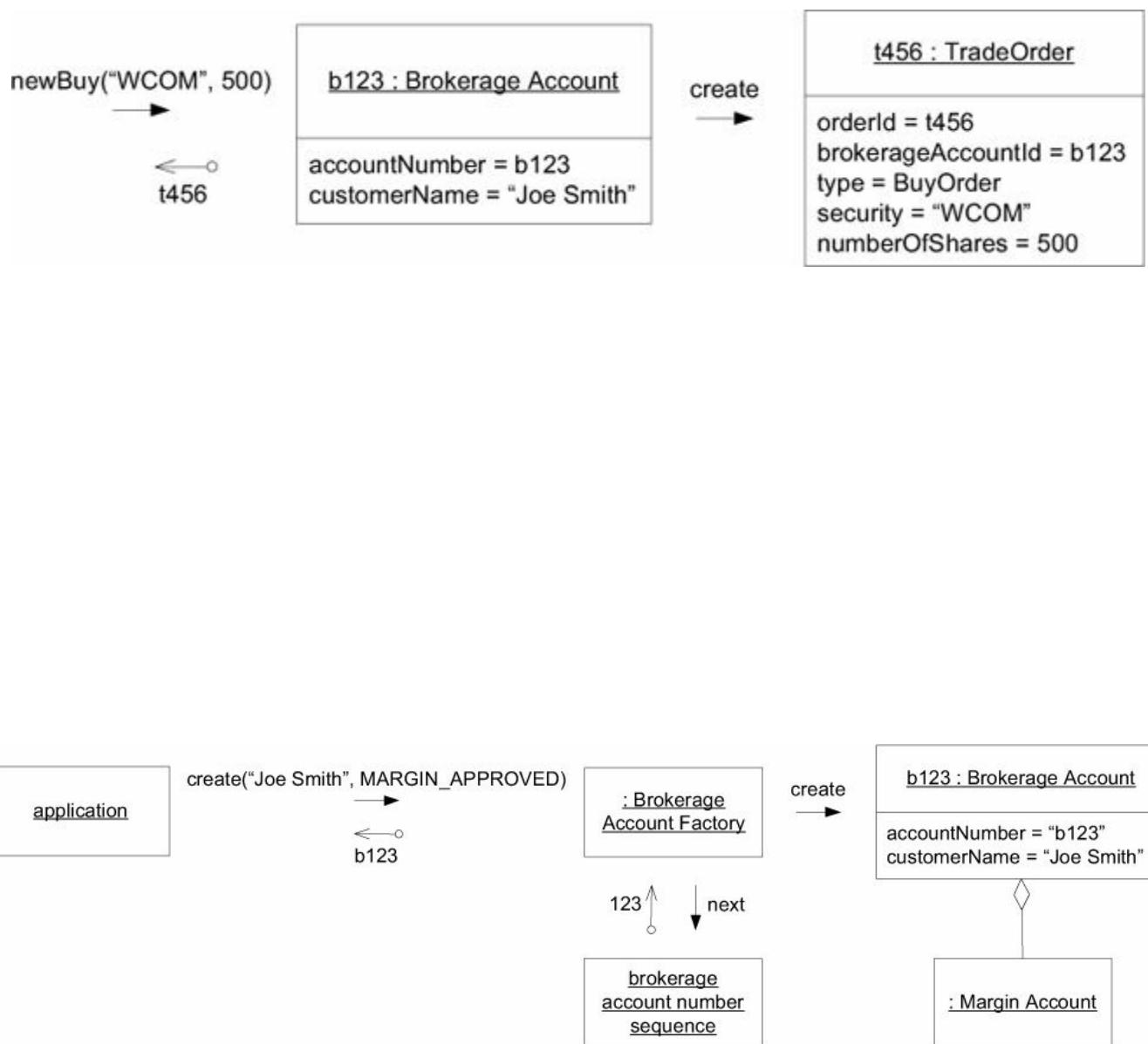
* * *

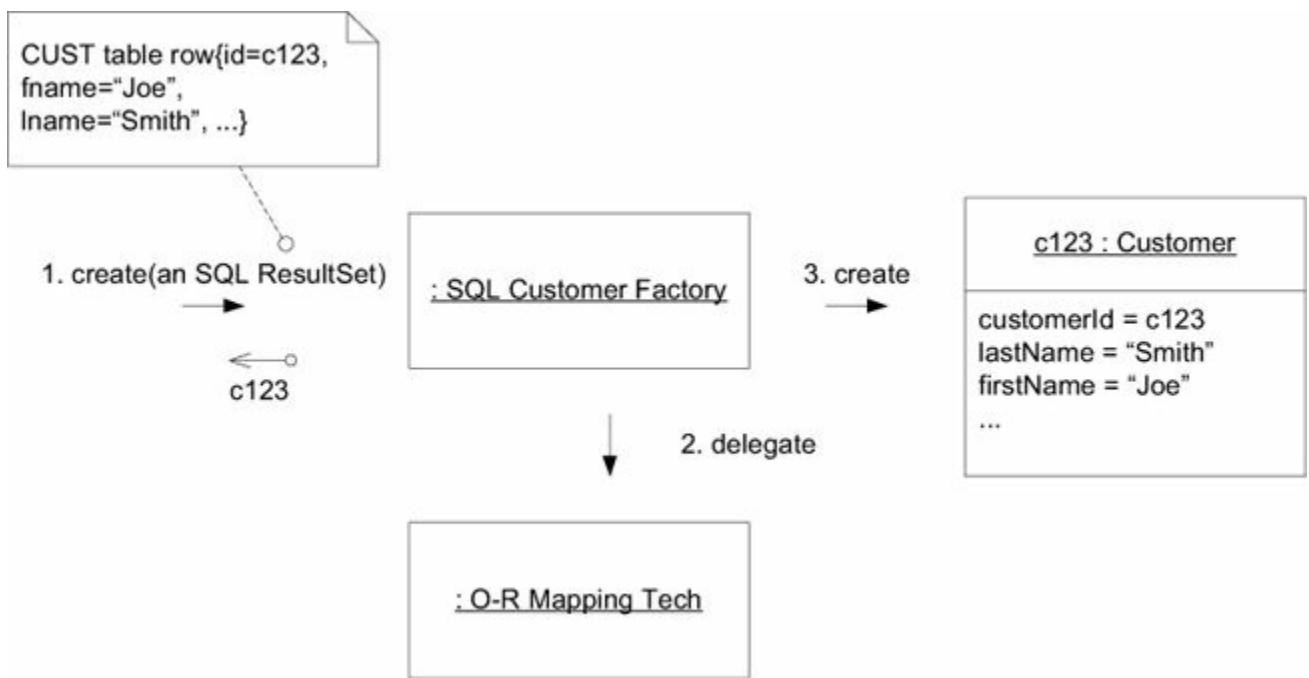


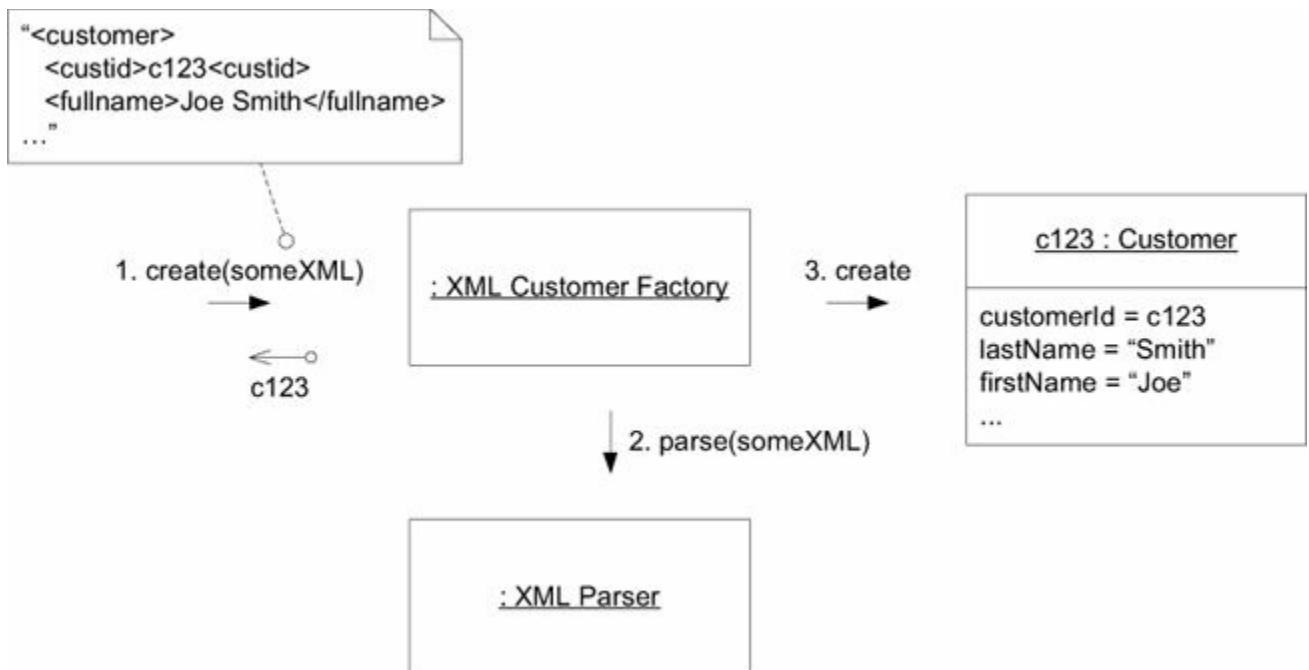
* * *



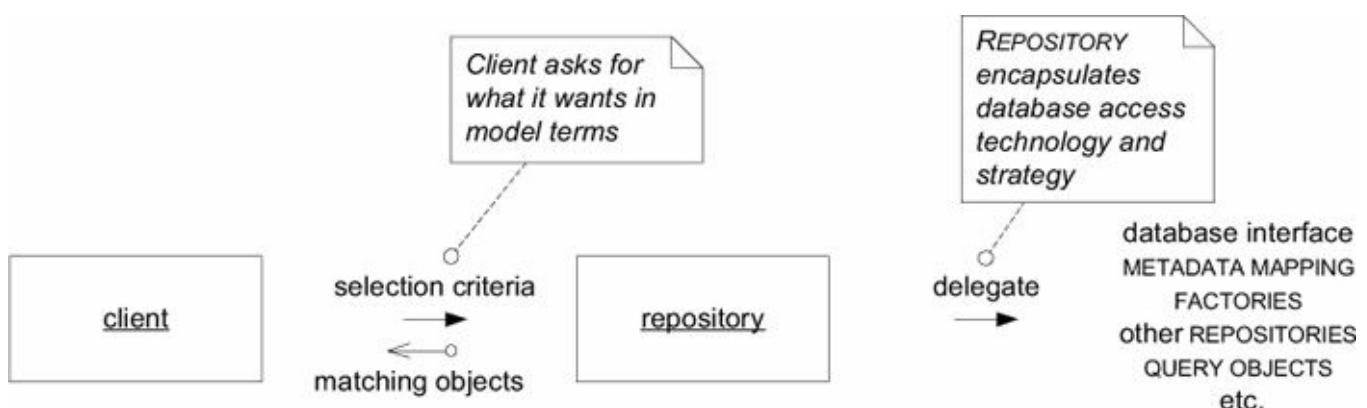


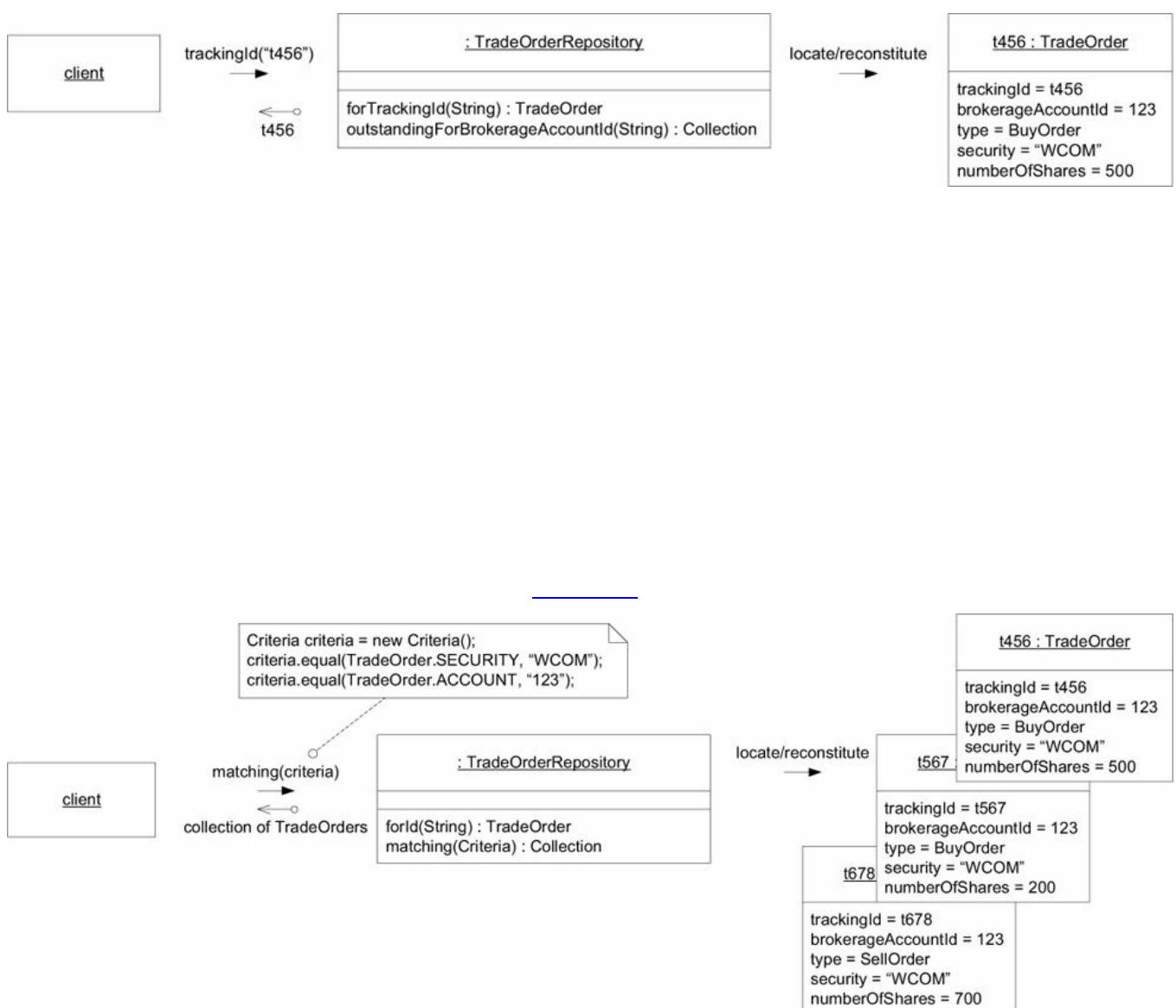




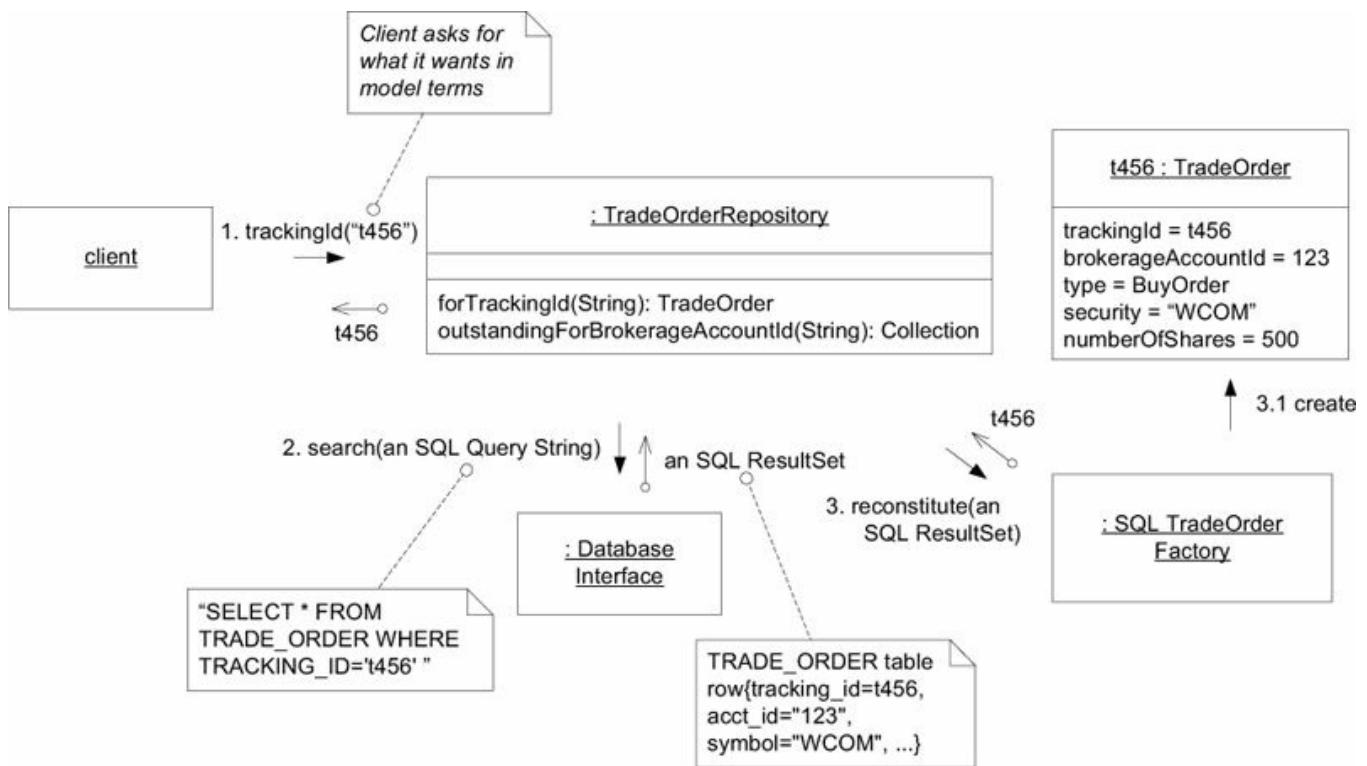


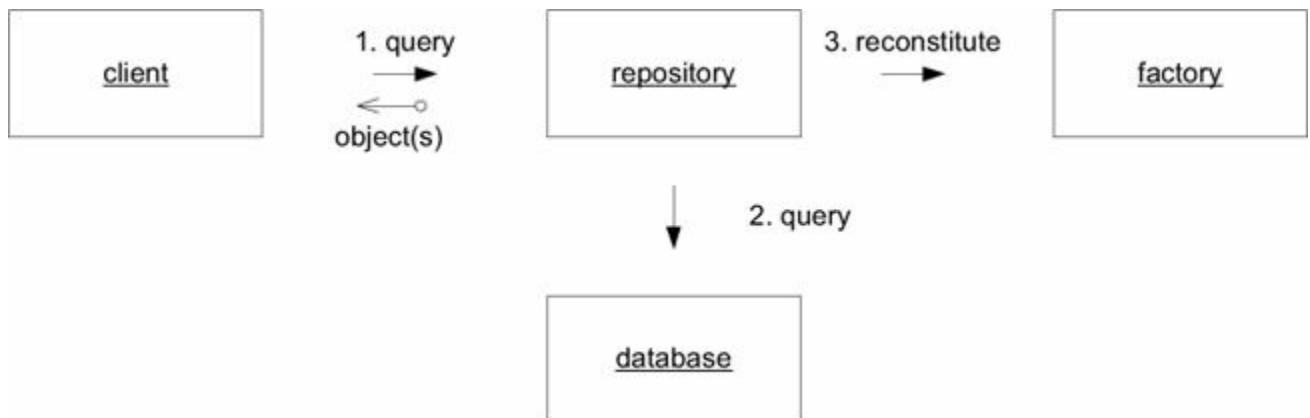
* * *

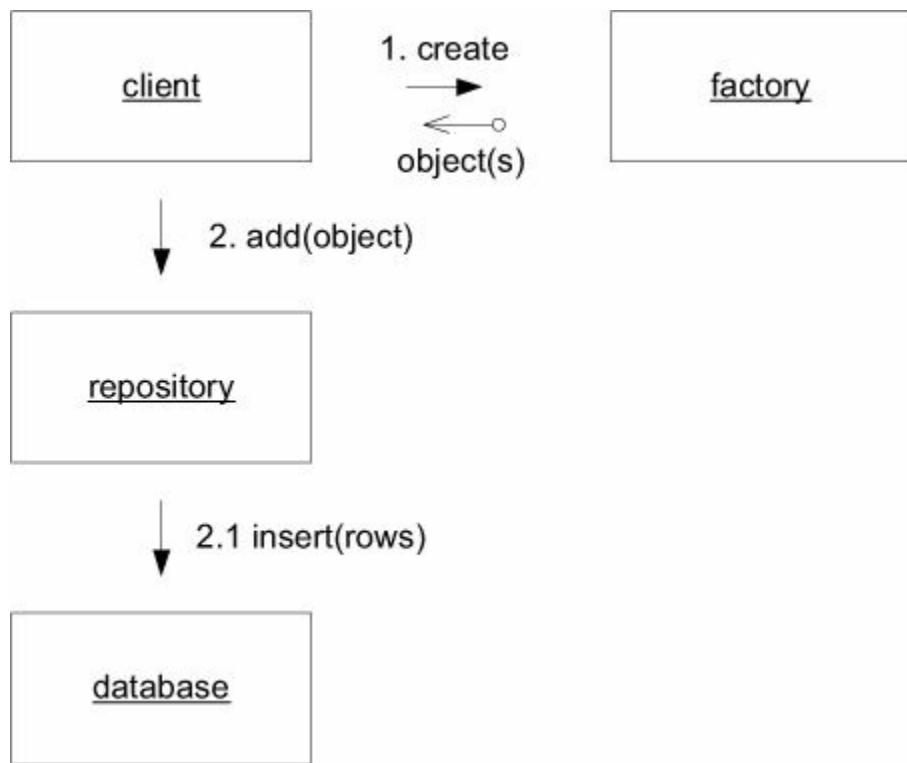


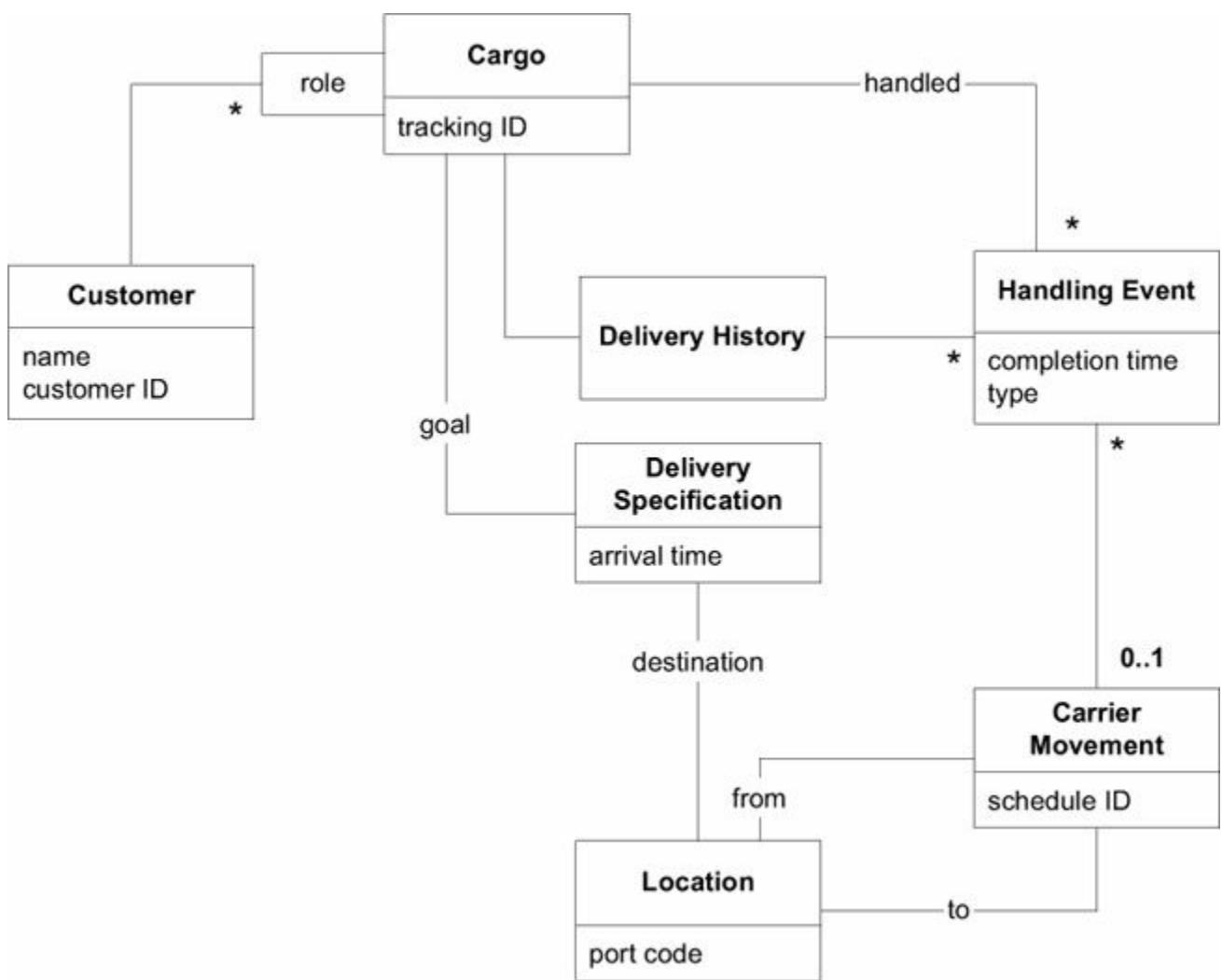


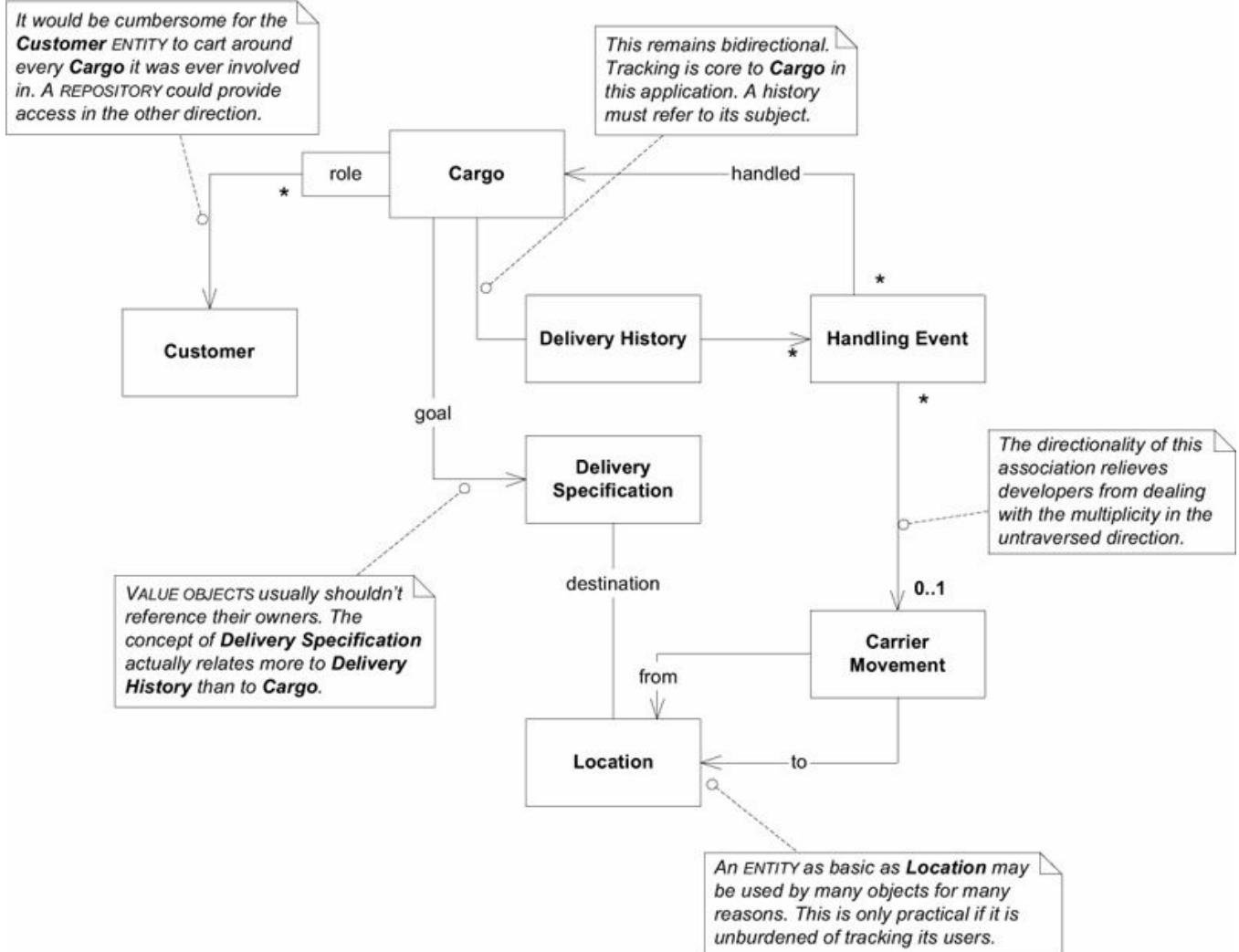


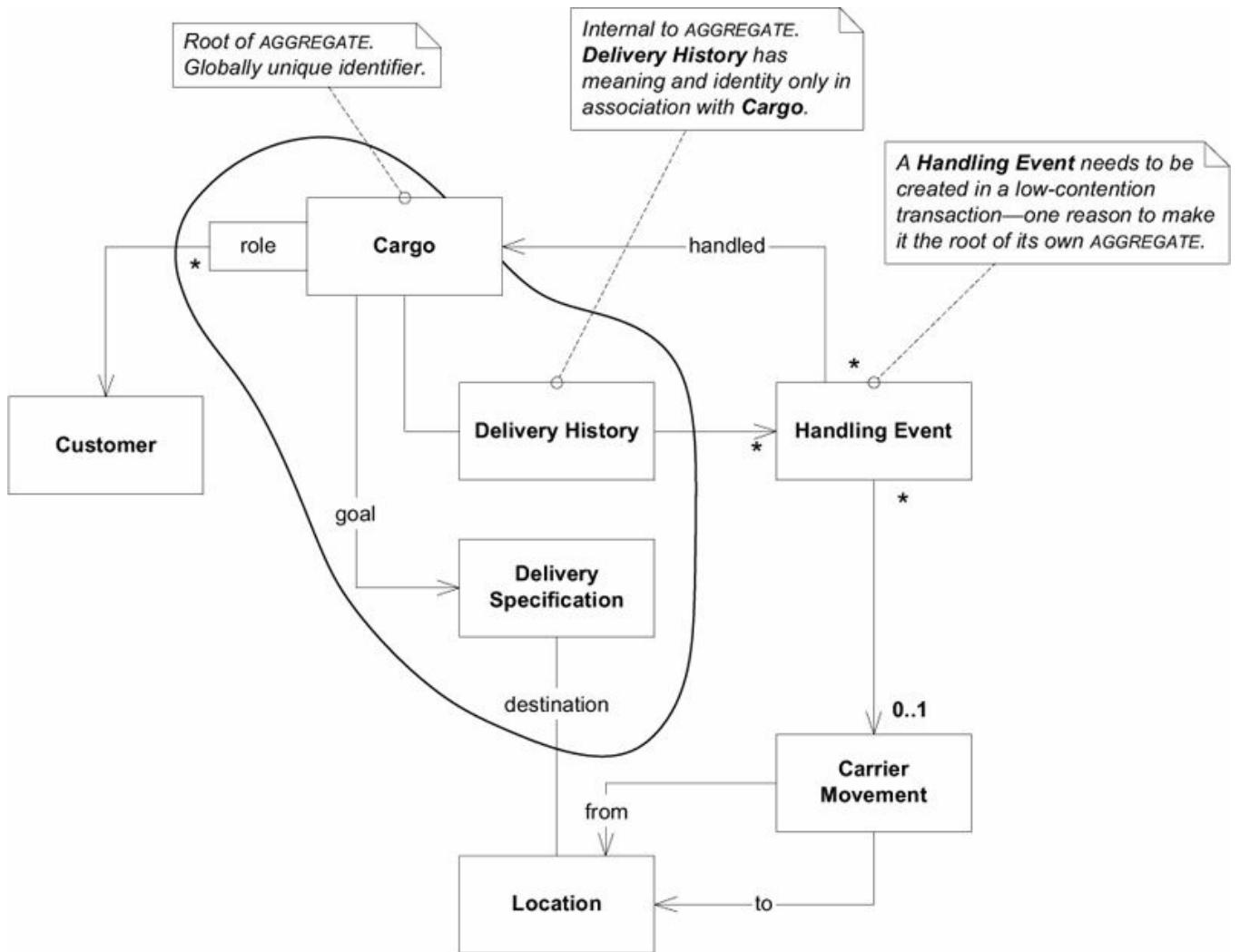


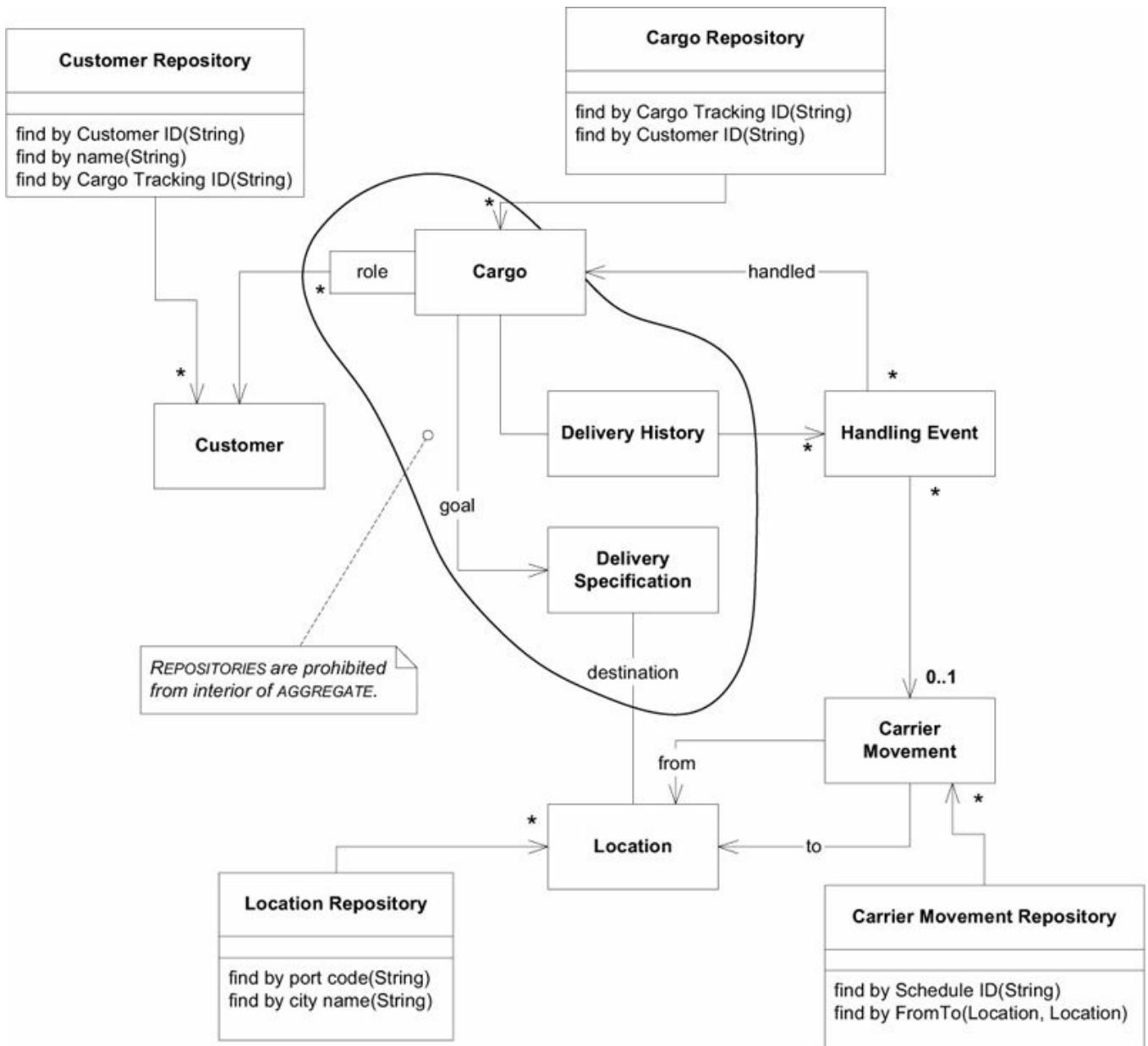














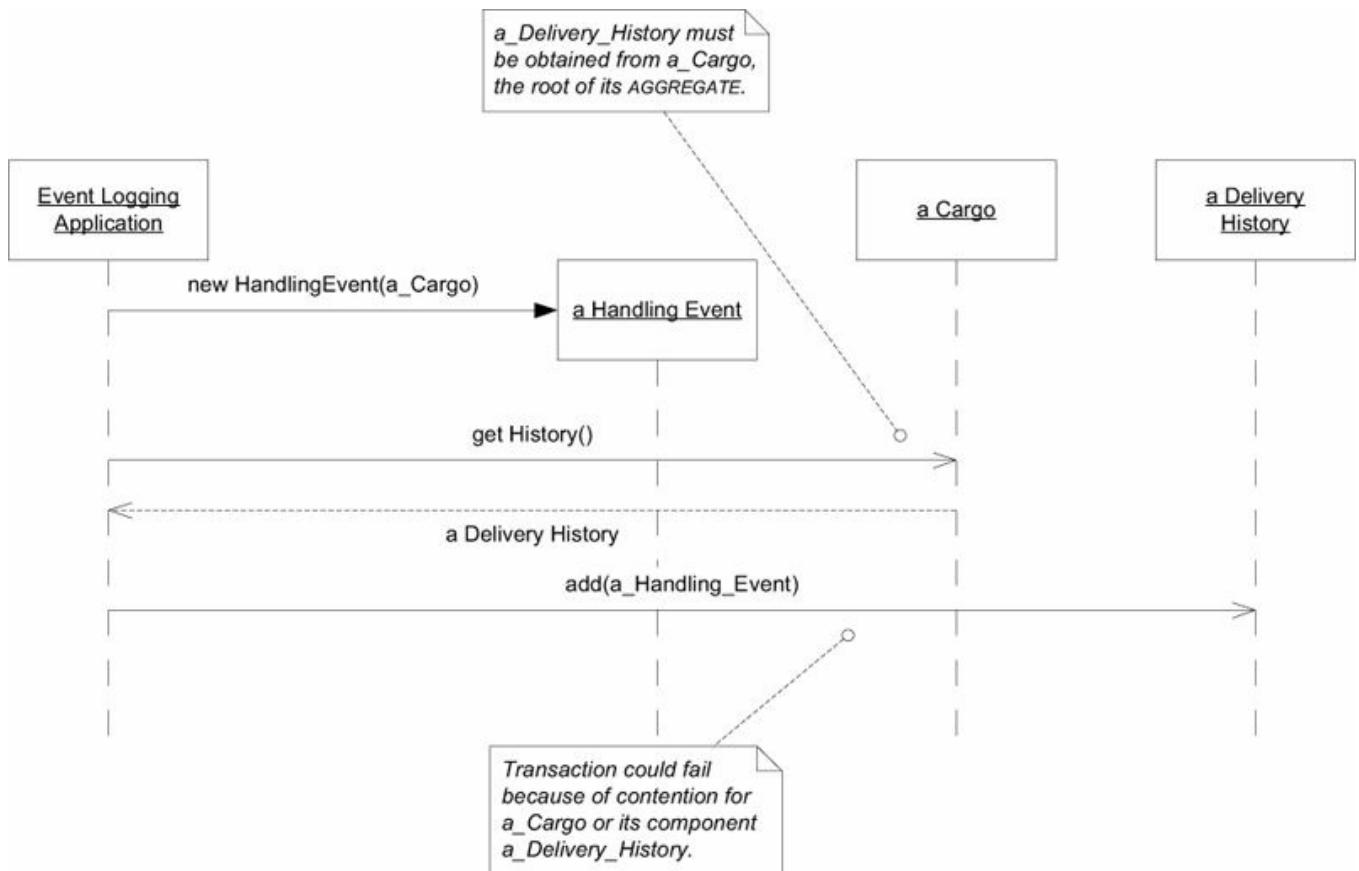
```
public Cargo copyPrototype( String newTrackingID)
```

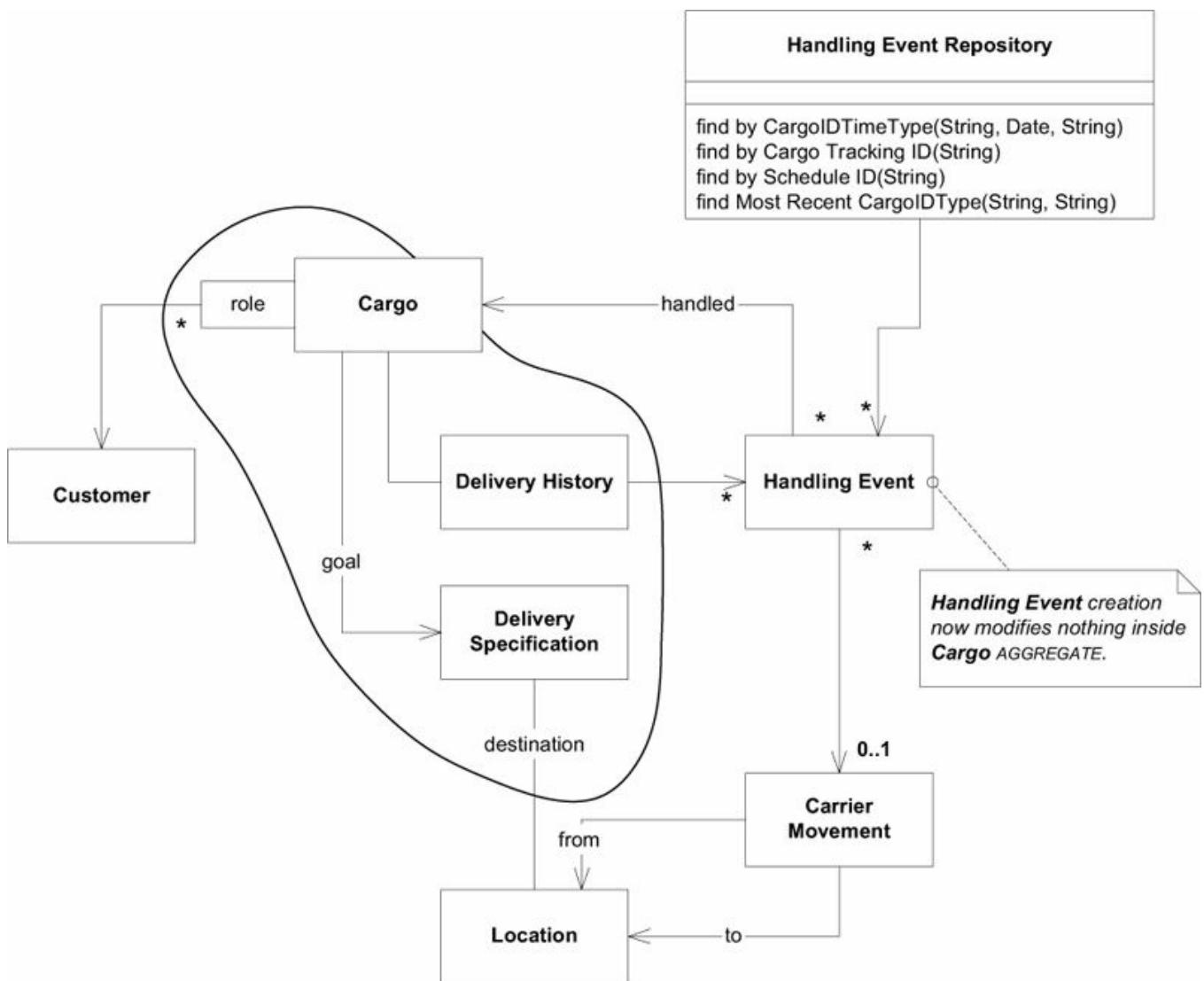
```
public Cargo newCargo( Cargo prototype, String newTrackingID)
```

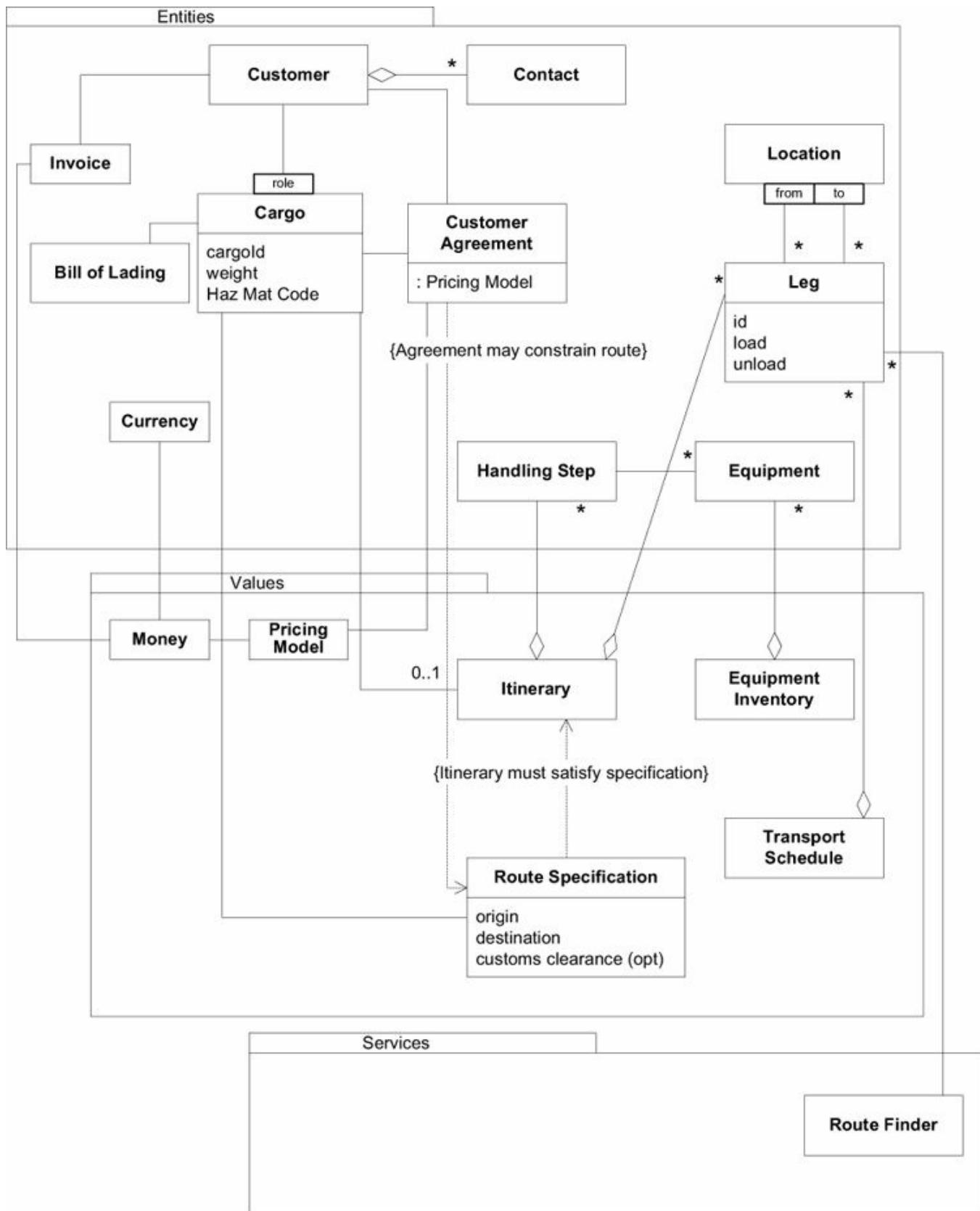
```
public Cargo newCargo( Cargo prototype)
```

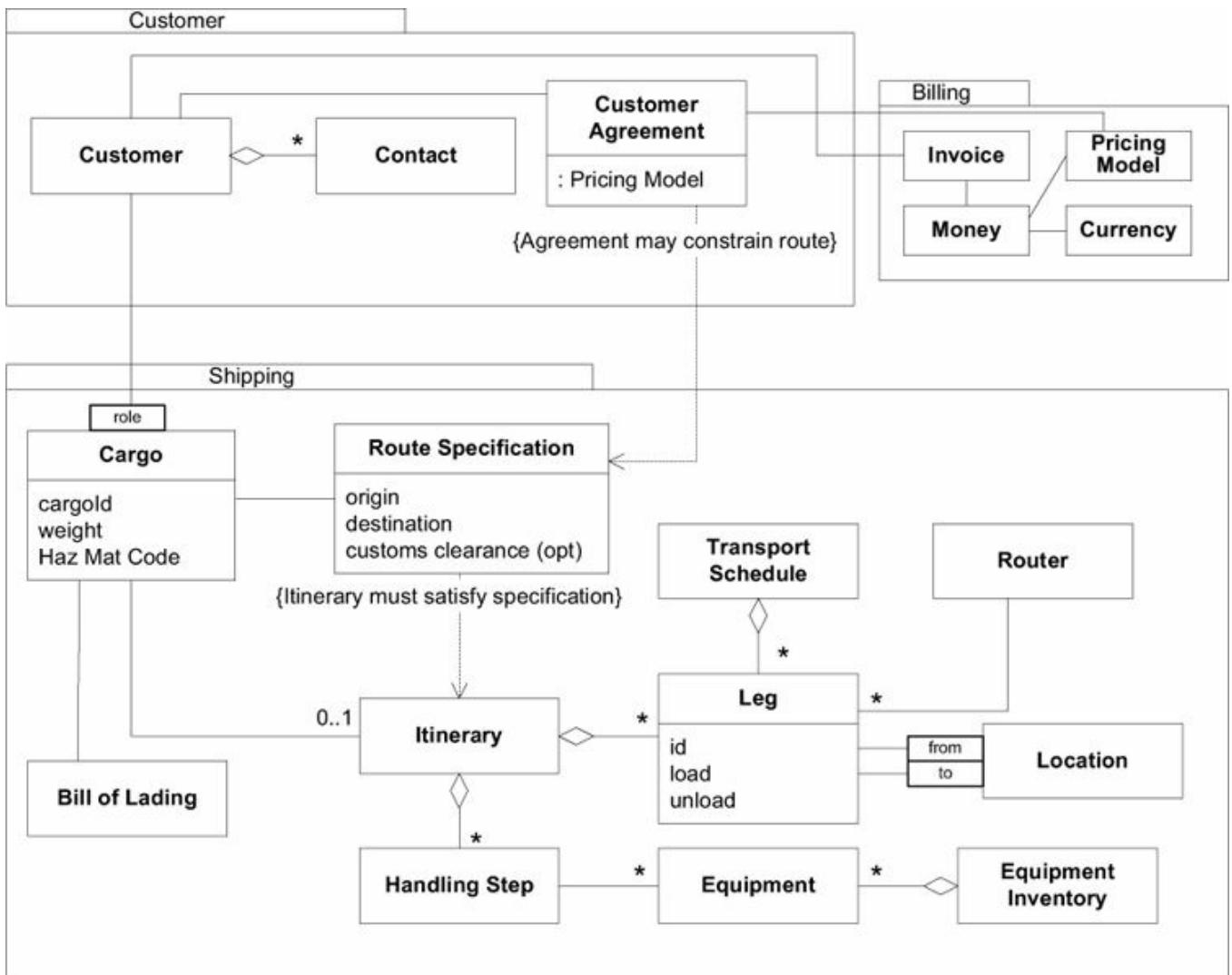
```
public Cargo( String id) {  
    trackingID = id;  
    deliveryHistory = new DeliveryHistory( this);  
    customerRoles = new HashMap( );  
}
```

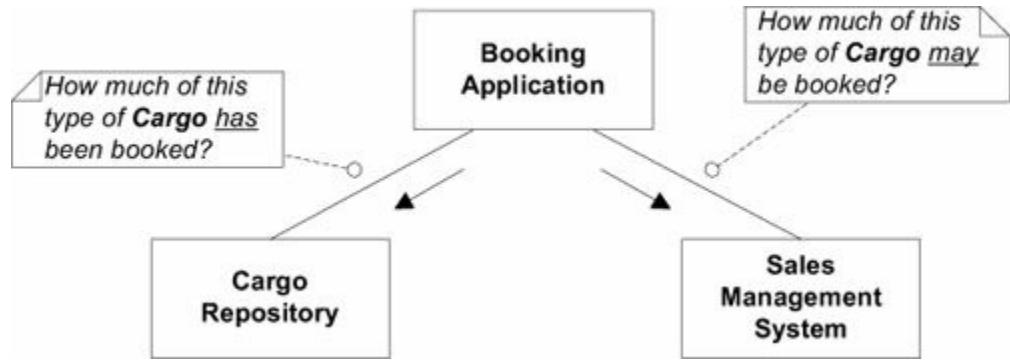
```
public HandlingEvent( Cargo c, String eventType, Date timeStamp) {  
    handled = c;  
    type = eventType;  
    completionTime = timeStamp;  
}  
  
public static HandlingEvent newLoading(  
    Cargo c, CarrierMovement loadedOnto, Date timeStamp) {  
    HandlingEvent result =  
        new HandlingEvent( c, LOADING_EVENT, timeStamp);  
    result.setCarrierMovement( loadedOnto);  
    return result;  
}
```

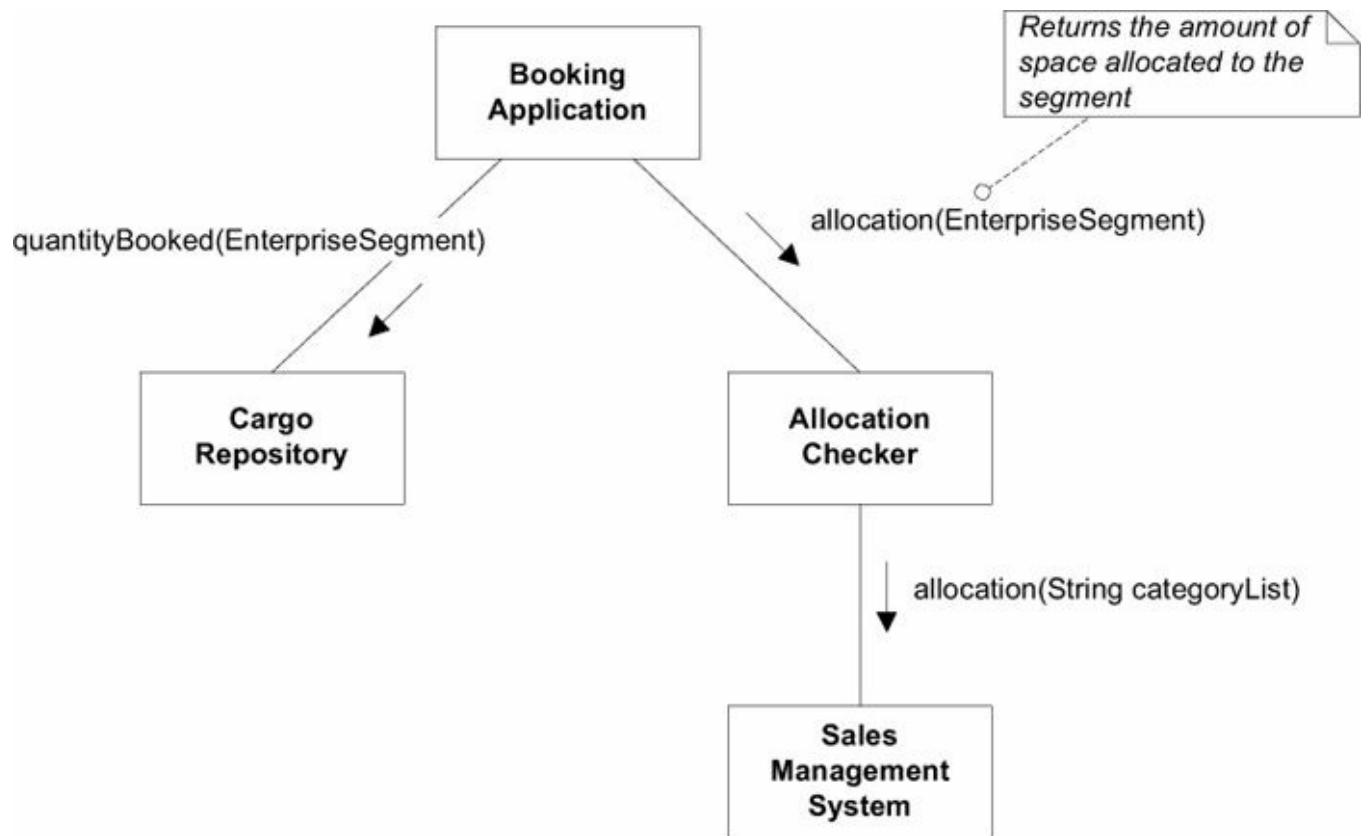


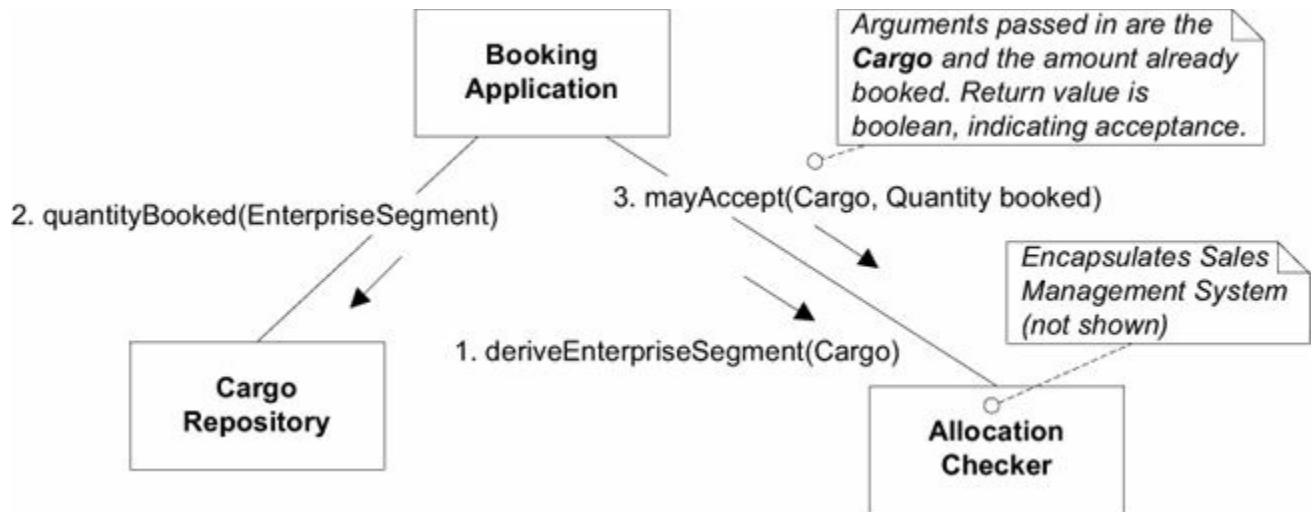




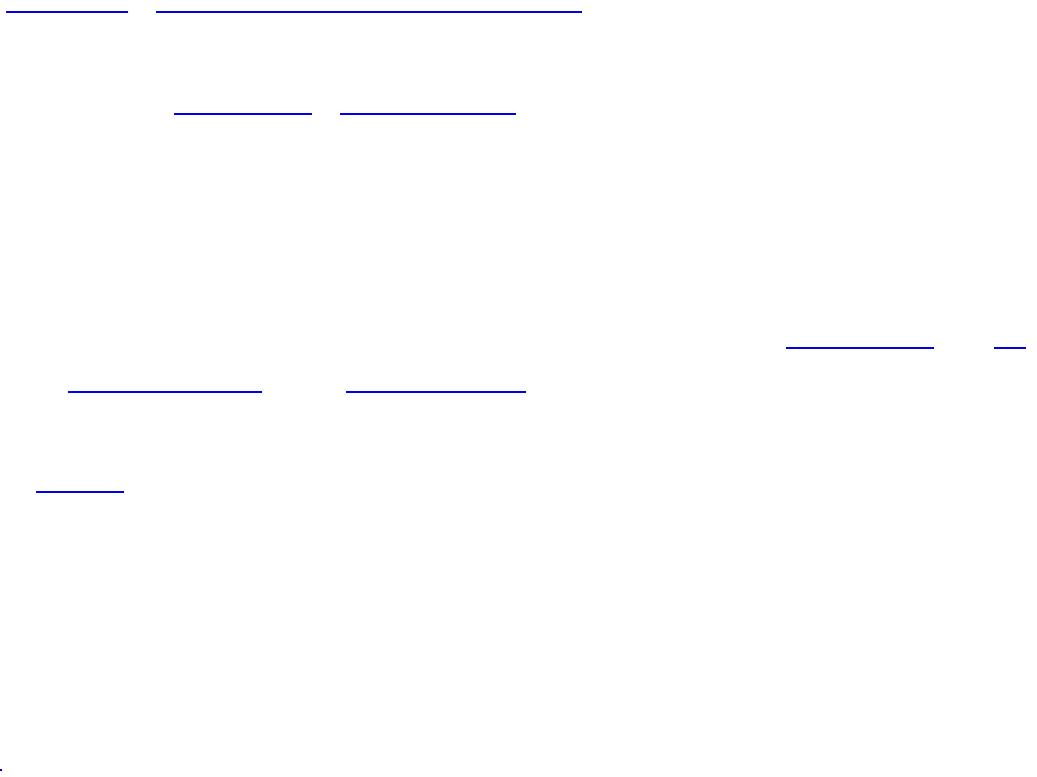


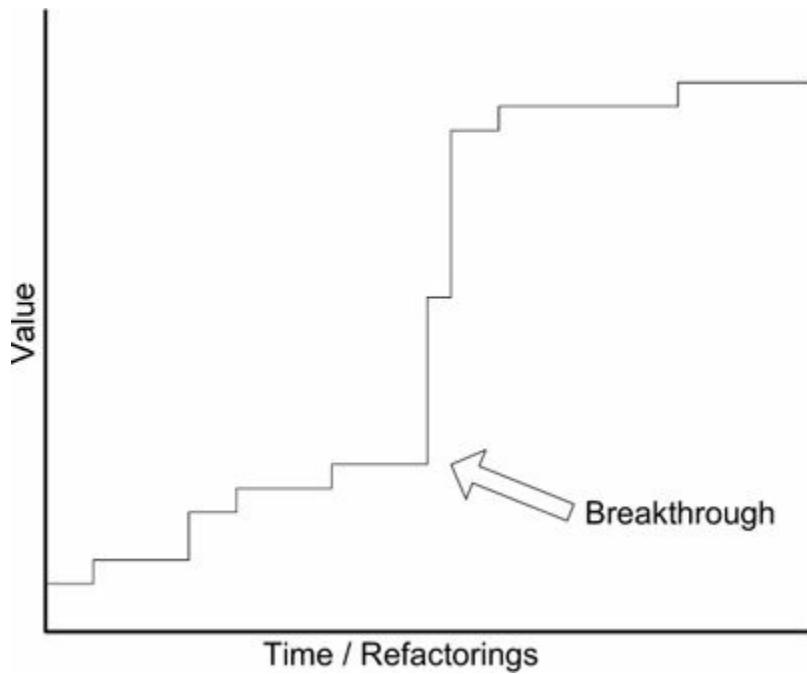


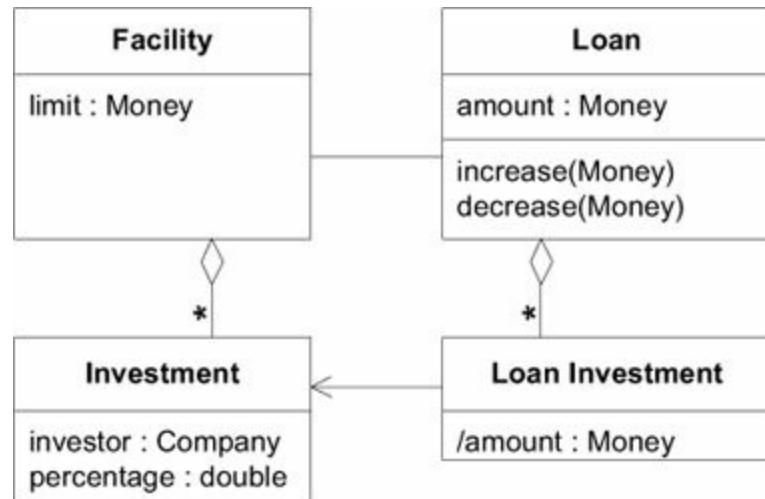


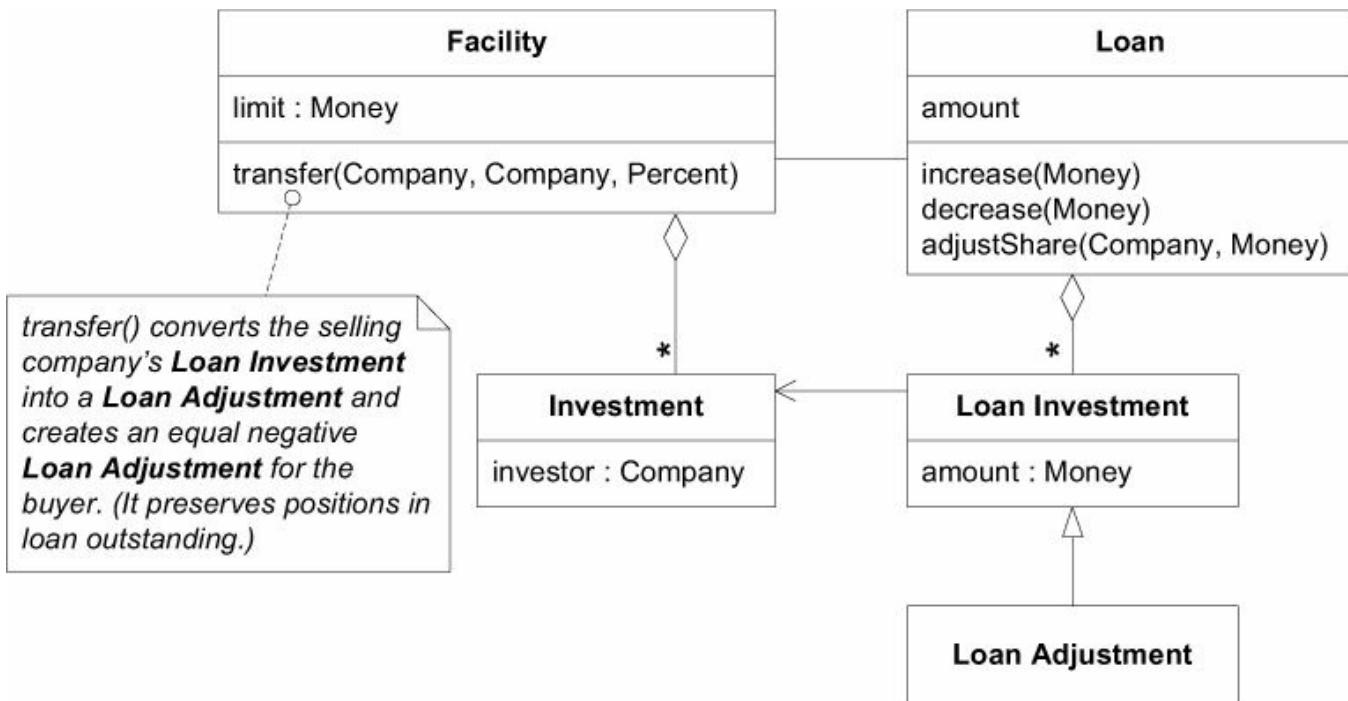


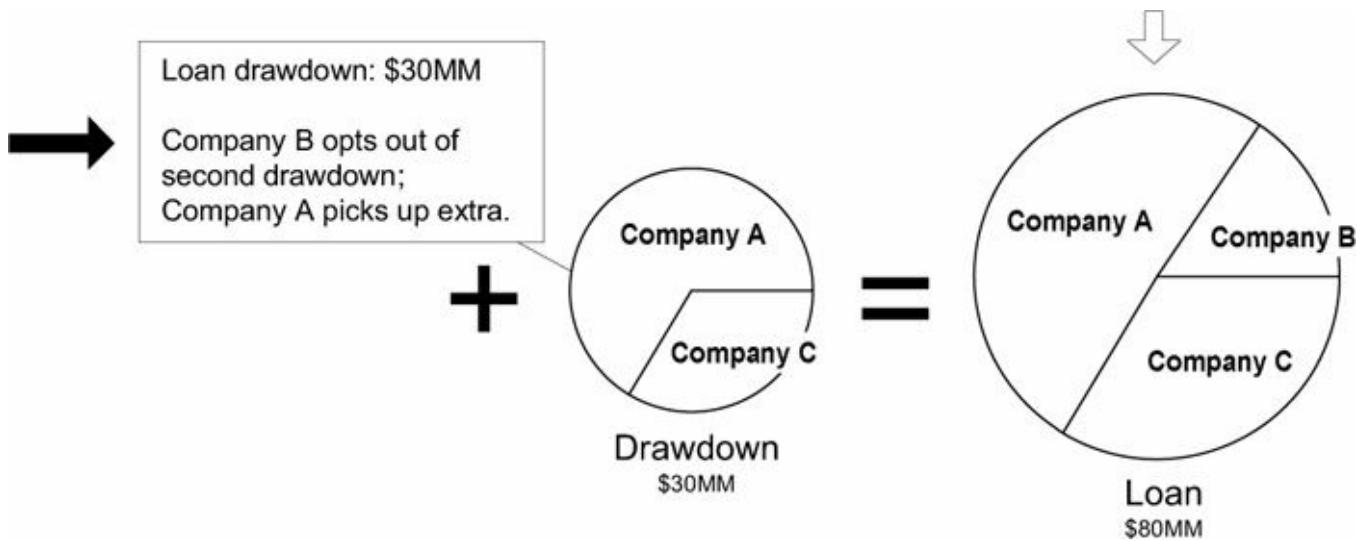
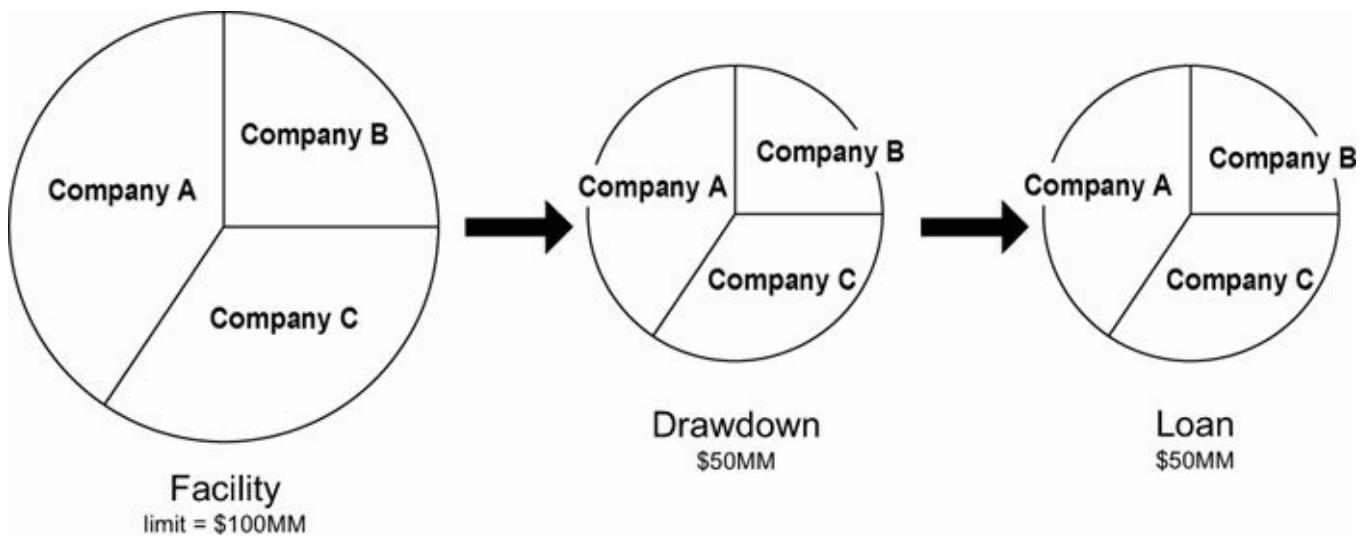








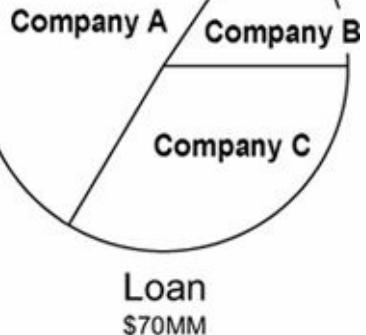
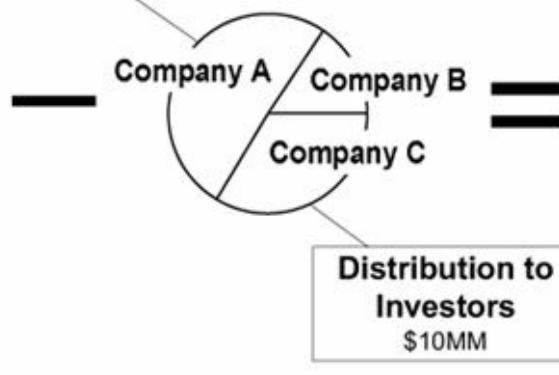






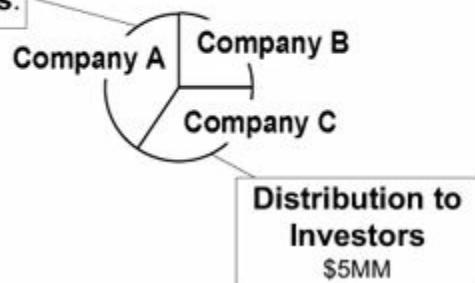
Principal payment: \$10MM

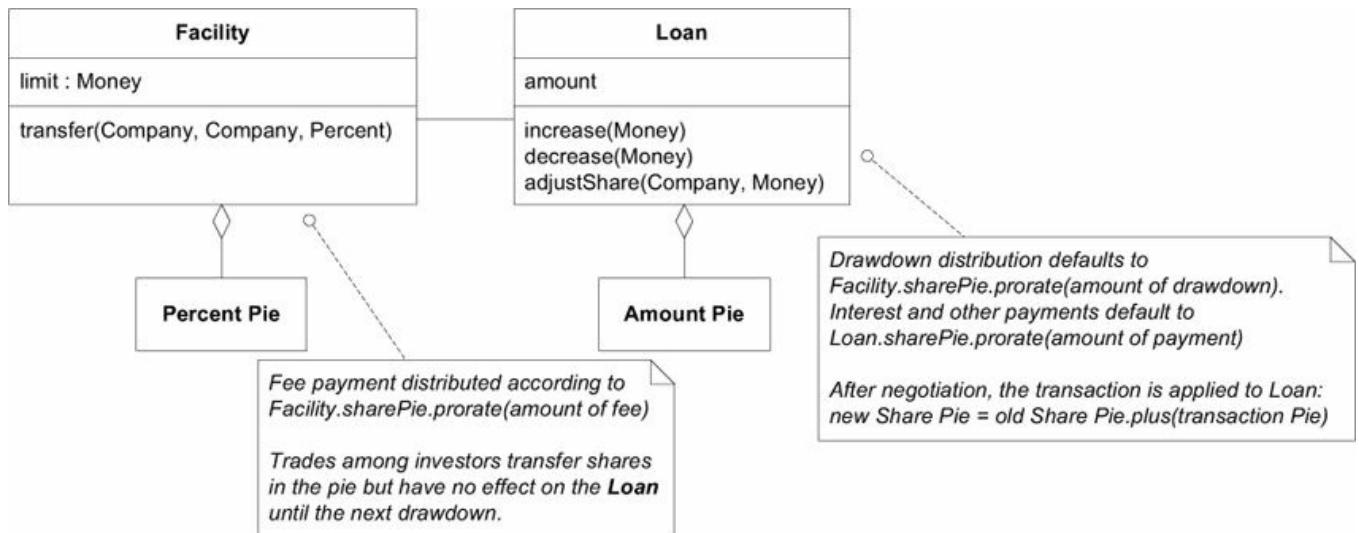
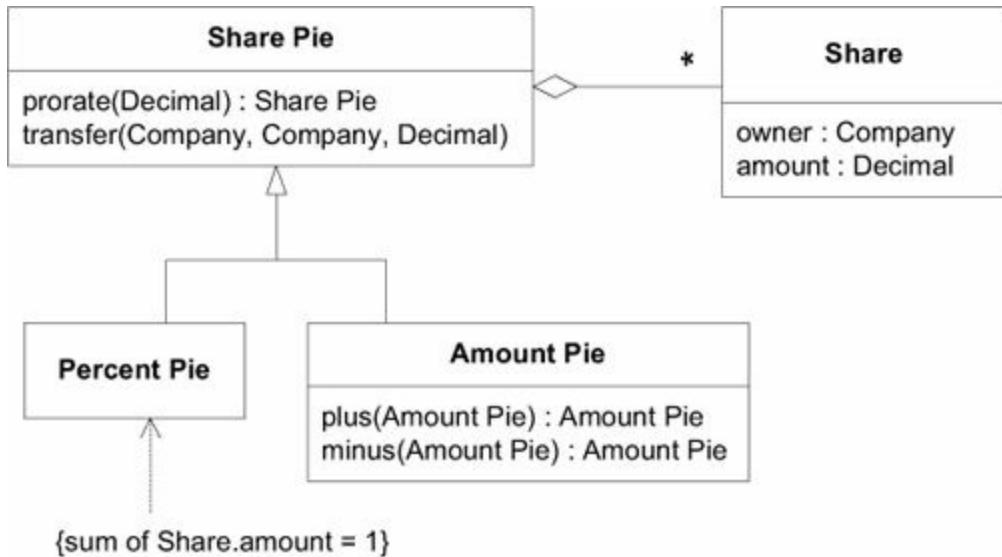
Distribution to investors is prorated by **Loan Shares**.

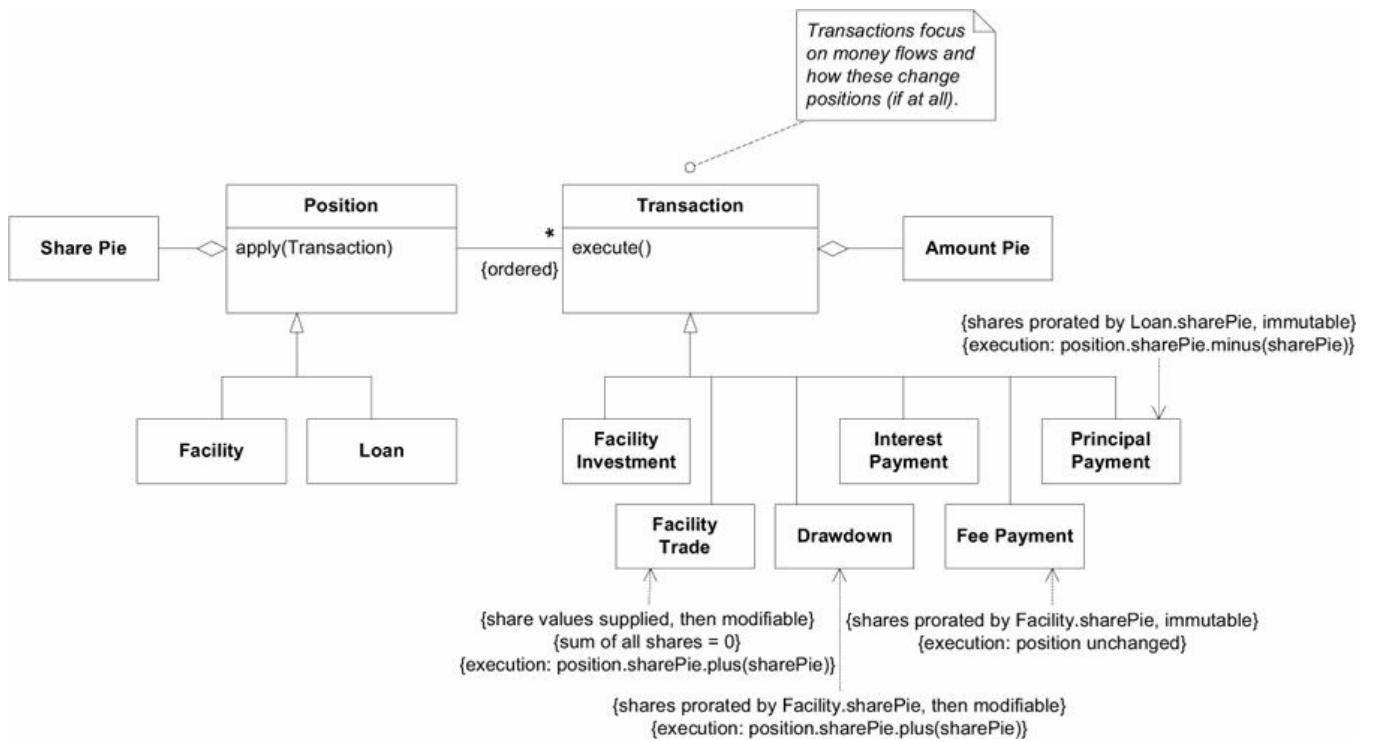


Fee payment: \$5MM

Distribution to investors is prorated by **Facility Shares**.







Cargo
cargoid
origin
destination
weight

origin
destination

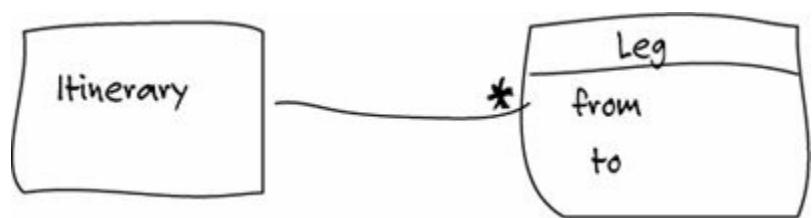


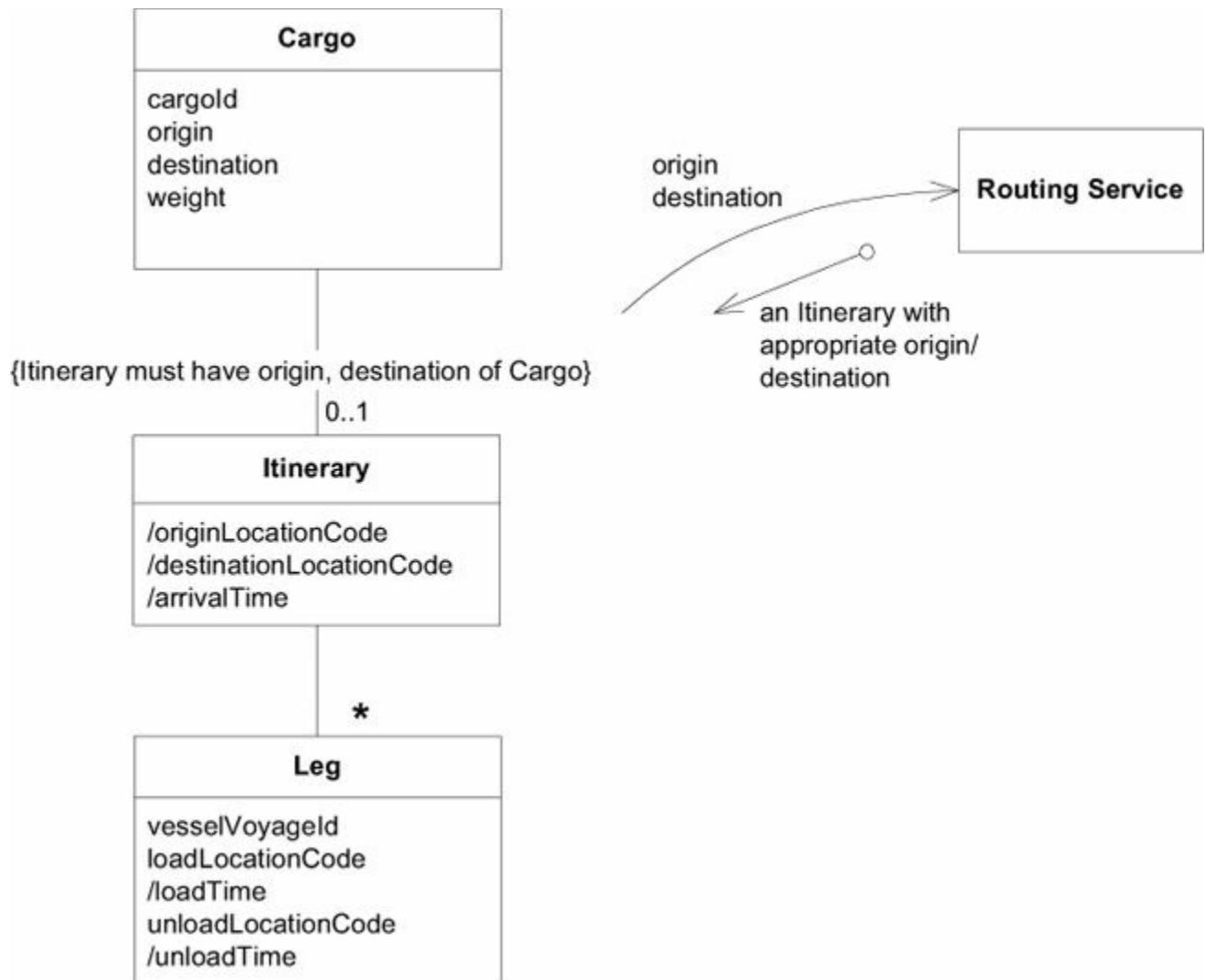
populate
cargo_bookings table

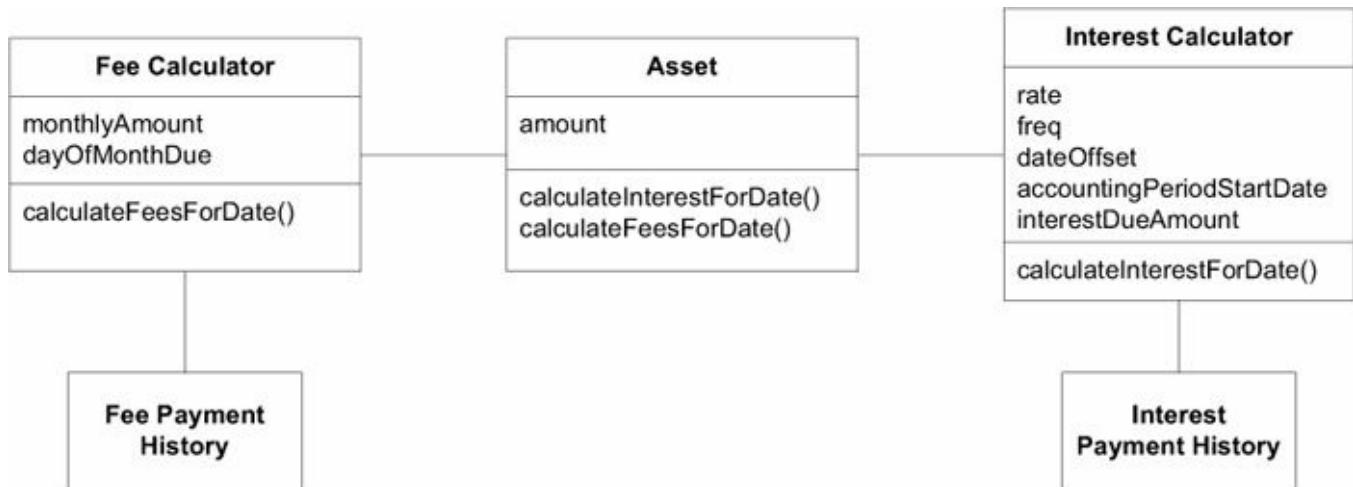
Database table: cargo_bookings

Cargo_ID Voyage_ID Load_Loc Unload_Loc

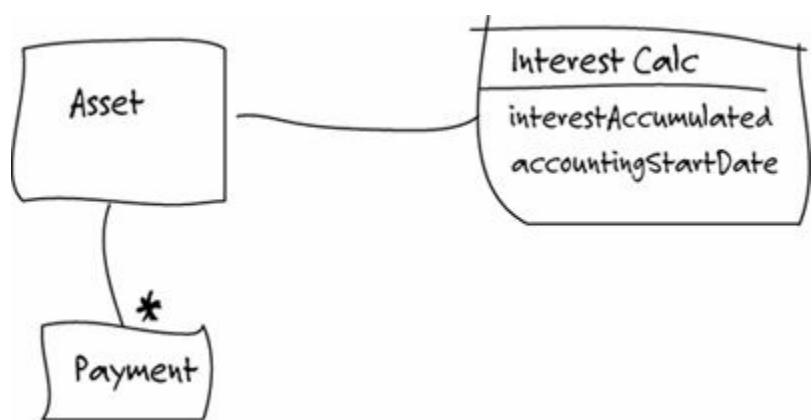
--	--	--	--

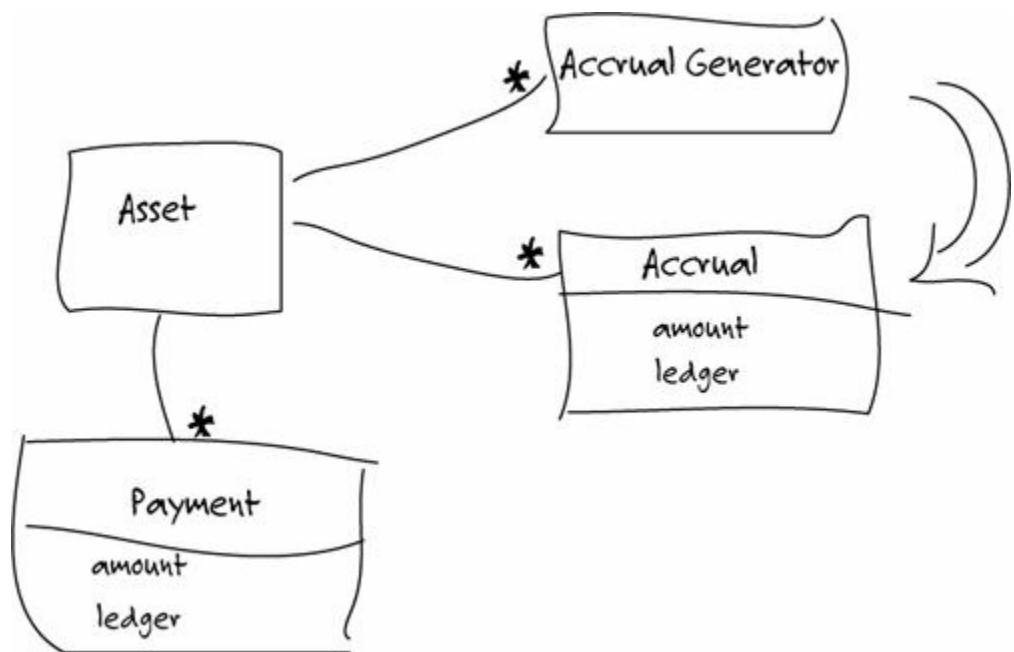
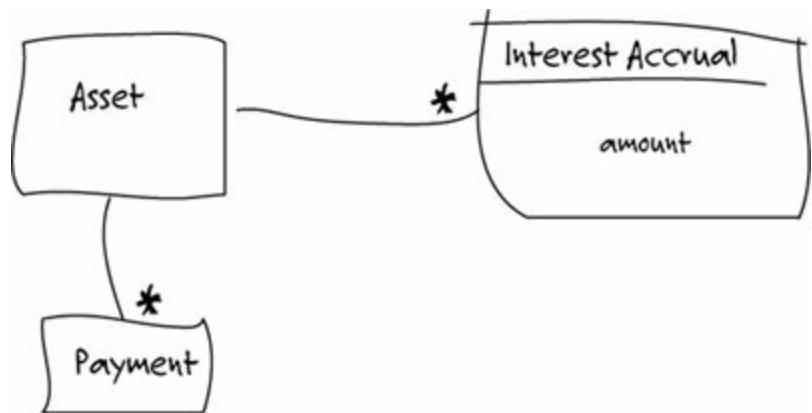


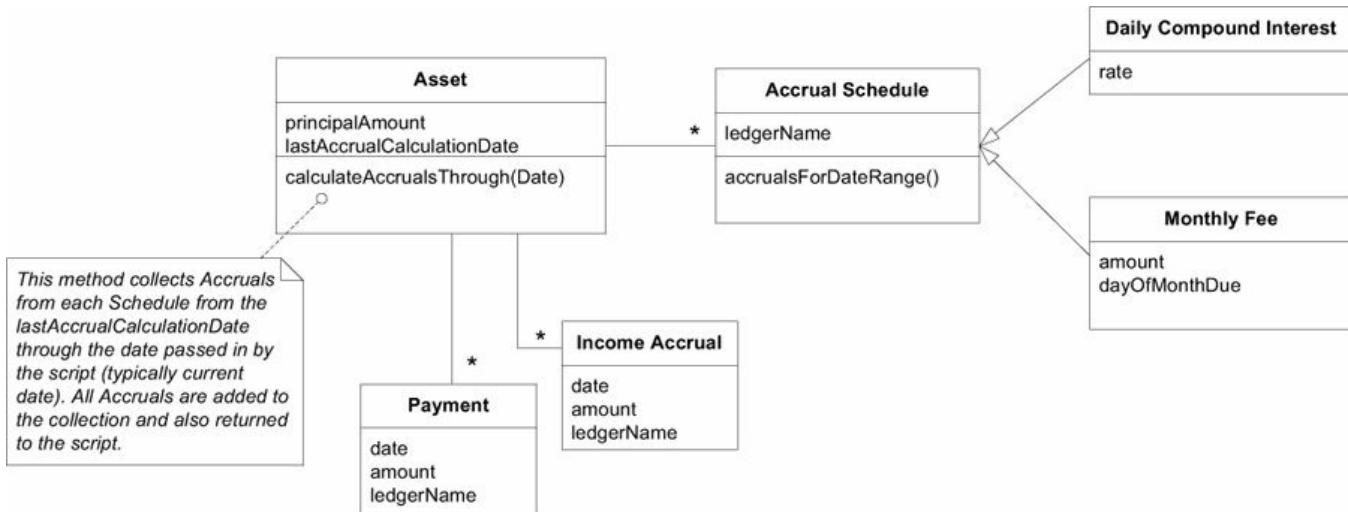




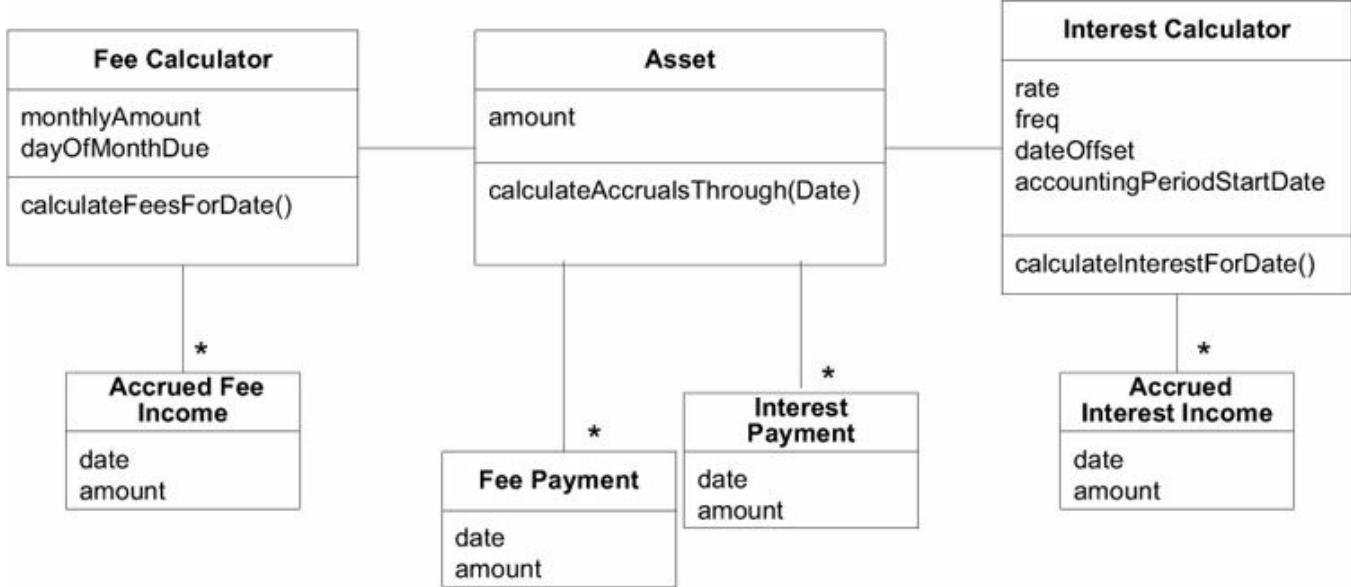
`calculateInterestForDate()`

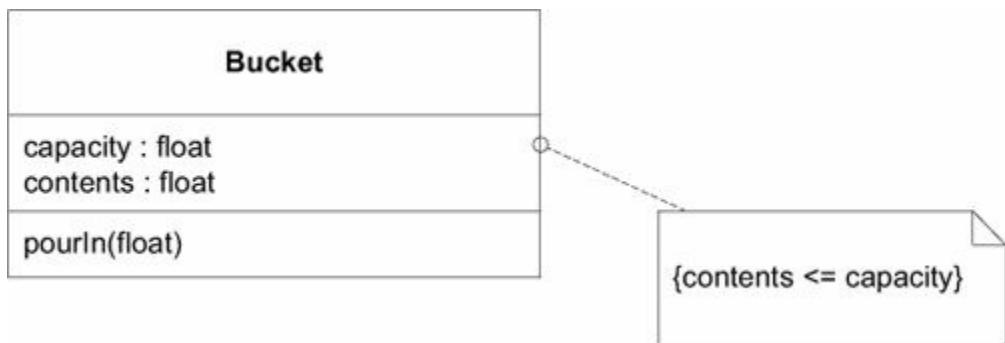






calculateAccrualsThrough()





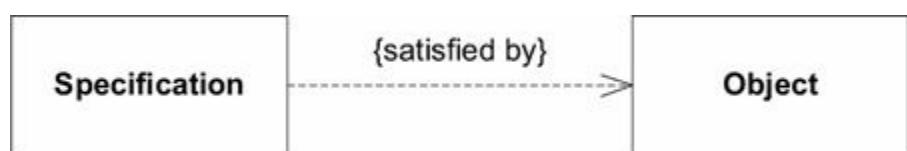
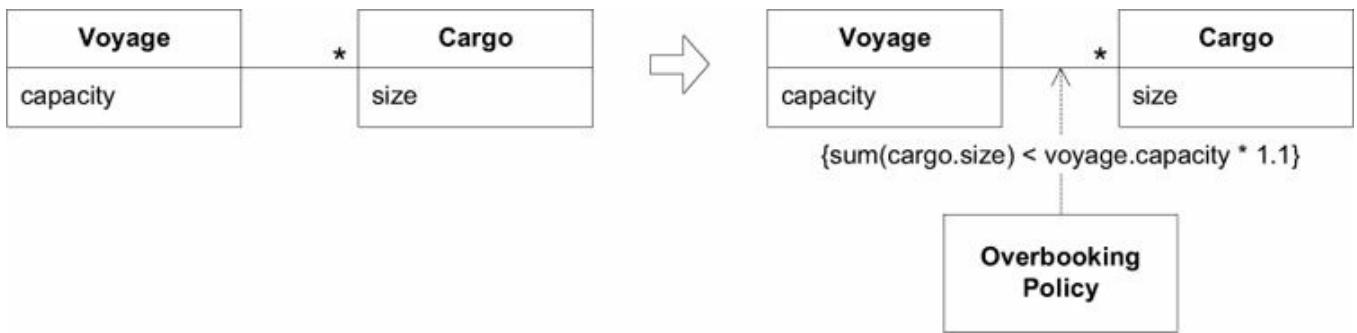
```
class Bucket {  
    private float capacity;  
    private float contents;  
  
    public void pourIn( float addedVolume) {  
        if (contents + addedVolume > capacity) {  
            contents = capacity;  
        } else {  
            contents = contents + addedVolume;  
        }  
    }  
}
```

```
class Bucket {  
    private float capacity;
```

```
private float contents;
public void pourIn( float addedVolume) {
    float volumePresent = contents + addedVolume;
    contents = constrainedToCapacity( volumePresent);
}

private float constrainedToCapacity( float volumePlacedIn) {
    if (volumePlacedIn > capacity) return capacity;
    return volumePlacedIn;
}
}
```

pourIn()



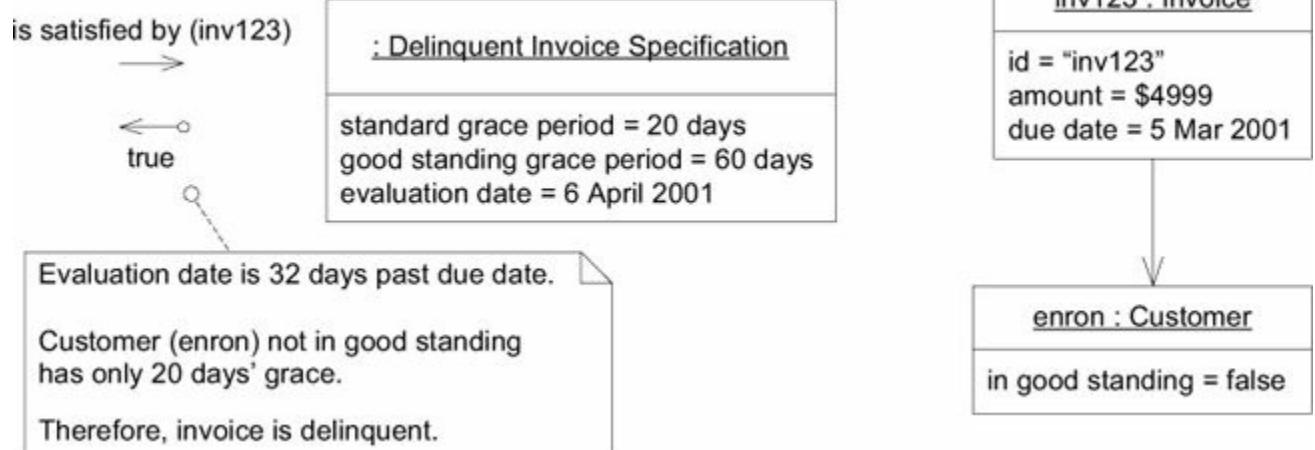
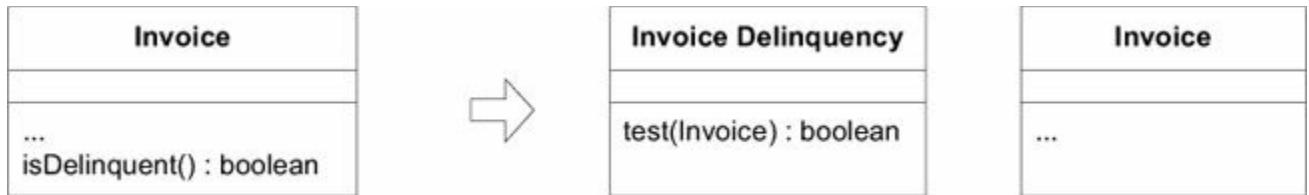
`anIterator.hasNext()`

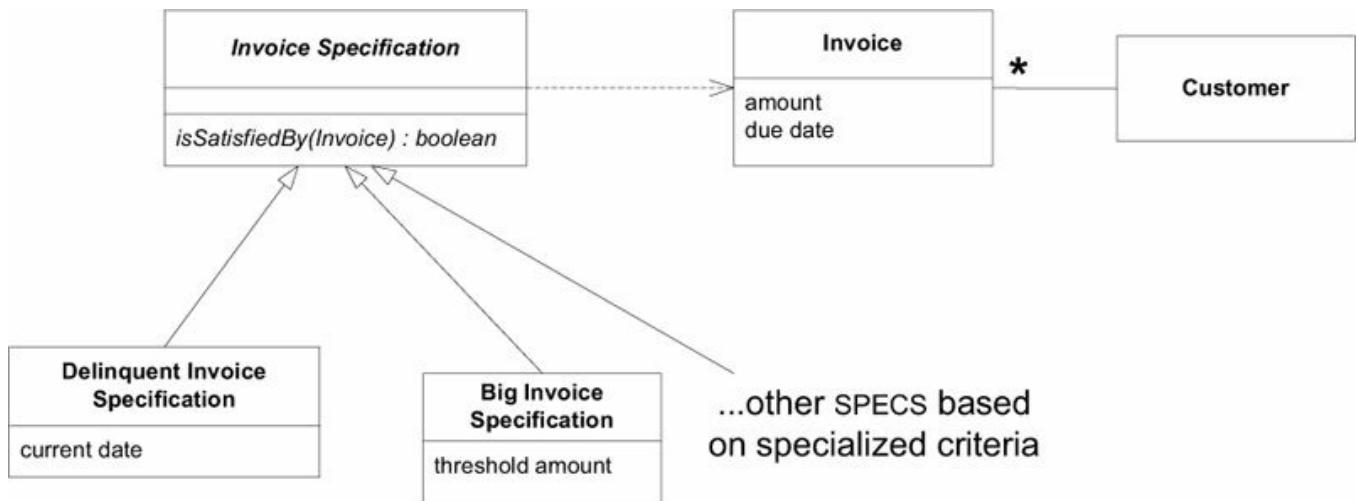
```
anInvoice.isOverdue()
```

```
isOverdue()
```

```
public boolean isOverdue() {  
    Date currentDate = new Date();  
    return currentDate.after(dueDate);  
}
```

```
anInvoice.isDelinquent()
```





```

class DelinquentInvoiceSpecification extends
    InvoiceSpecification {
    private Date currentDate;
    // An instance is used and discarded on a single date

    public DelinquentInvoiceSpecification(Date currentDate) {
        this.currentDate = currentDate;
    }
}
  
```

```

public boolean isSatisfiedBy(Invoice candidate) {
    int gracePeriod =
        candidate.customer().getPaymentGracePeriod();
    Date firmDeadline =
        DateUtility.addDaysToDate(candidate.dueDate(),
            gracePeriod);
    return currentDate.after(firmDeadline);
}

}

public boolean accountIsDelinquent(Customer customer) {
    Date today = new Date();
    Specification delinquentSpec =
        new DelinquentInvoiceSpecification(today);
    Iterator it = customer.getInvoices().iterator();
    while (it.hasNext()) {
        Invoice candidate = (Invoice) it.next();
        if (delinquentSpec.isSatisfiedBy(candidate)) return true;
    }
    return false;
}

public Set selectSatisfying(InvoiceSpecification spec) {
    Set results = new HashSet();
    Iterator it = invoices.iterator();
    while (it.hasNext()) {
        Invoice candidate = (Invoice) it.next();
        if (spec.isSatisfiedBy(candidate)) results.add(candidate);
    }
    return results;
}

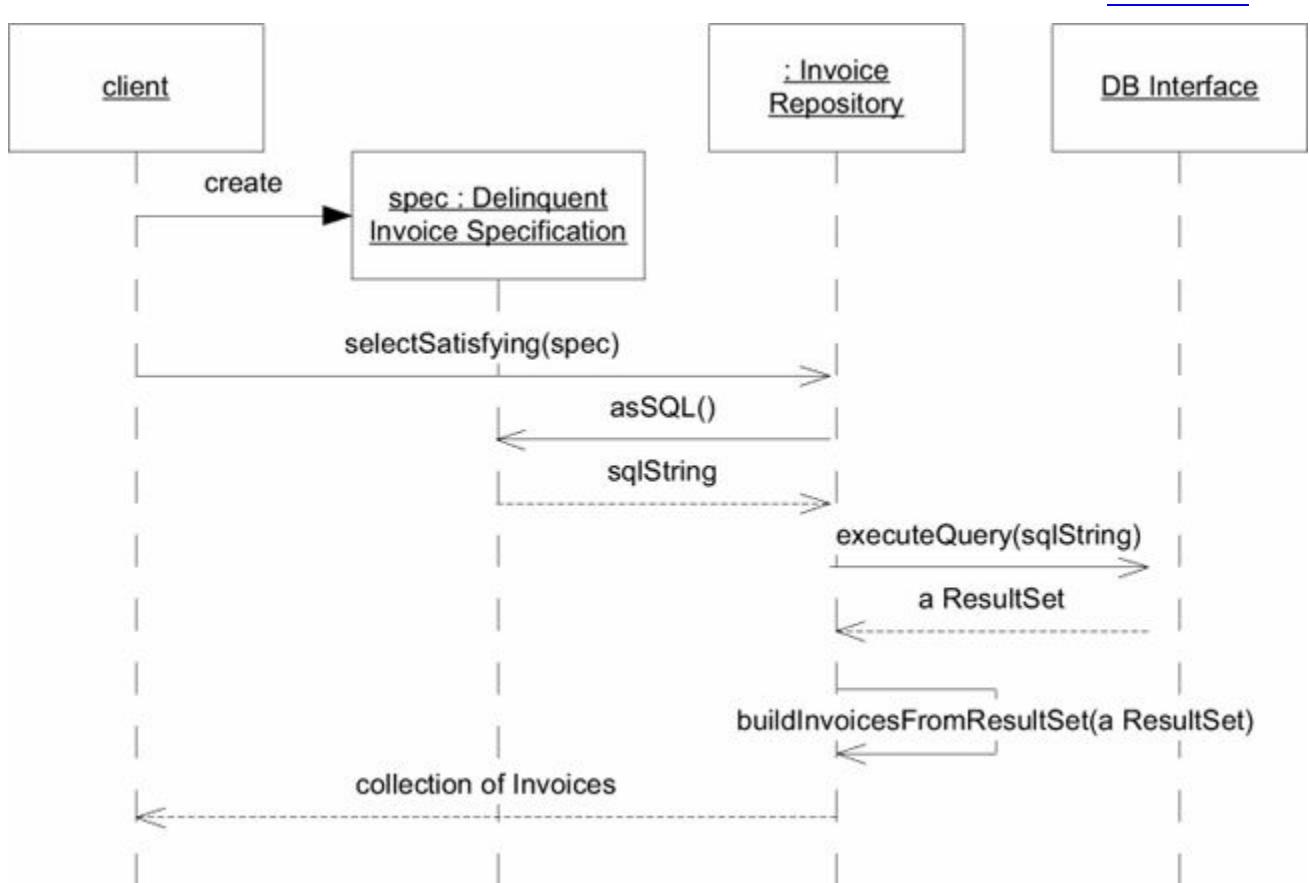
Set delinquentInvoices = invoiceRepository.selectSatisfying(
    new DelinquentInvoiceSpecification(currentDate));

```

```

public String asSQL( ) {
    return
        "SELECT * FROM INVOICE, CUSTOMER" +
        " WHERE INVOICE.CUST_ID = CUSTOMER.ID" +
        " AND INVOICE.DUE_DATE + CUSTOMER.GRACE_PERIOD" +
        " < " + SQLUtility.dateAsSQL( currentDate );
}

```



```

public class InvoiceRepository {

    public Set selectWhereGracePeriodPast( Date aDate) {
        //This is not a rule, just a specialized query
        String sql = whereGracePeriodPast_SQL( aDate);
        ResultSet queryResultSet =
            SQLDatabaseInterface.instance().executeQuery( sql);
        return buildInvoicesFromResultSet( queryResultSet);
    }

    public String whereGracePeriodPast_SQL( Date aDate) {
        return
            "SELECT * FROM INVOICE, CUSTOMER" +
            " WHERE INVOICE.CUST_ID = CUSTOMER.ID" +
            " AND INVOICE.DUE_DATE + CUSTOMER.GRACE_PERIOD" +
            " < " + SQLUtility.dateAsSQL( aDate);
    }

    public Set selectSatisfying( InvoiceSpecification spec) {
        return spec.satisfyingElementsFrom( this);
    }
}

assSql()
satisfyingElementsFrom(InvoiceRepository)

public class DelinquentInvoiceSpecification {
    // Basic DelinquentInvoiceSpecification code here

    public Set satisfyingElementsFrom(
                    InvoiceRepository repository) {
        //Delinquency rule is defined as:
        //    "grace period past as of current date"
        return repository.selectWhereGracePeriodPast( currentDate);
    }
}

```

```

public class InvoiceRepository {

    public Set selectWhereDueDateIsBefore( Date aDate) {
        String sql = whereDueDateIsBefore_SQL( aDate);
        ResultSet queryResultSet =
            SQLDatabaseInterface.instance().executeQuery( sql);
        return buildInvoicesFromResultSet( queryResultSet);
    }

    public String whereDueDateIsBefore_SQL( Date aDate) {
        return
            "SELECT * FROM INVOICE" +
            " WHERE INVOICE.DUE_DATE" +
            "      < " + SQLUtility.dateAsSQL( aDate);
    }

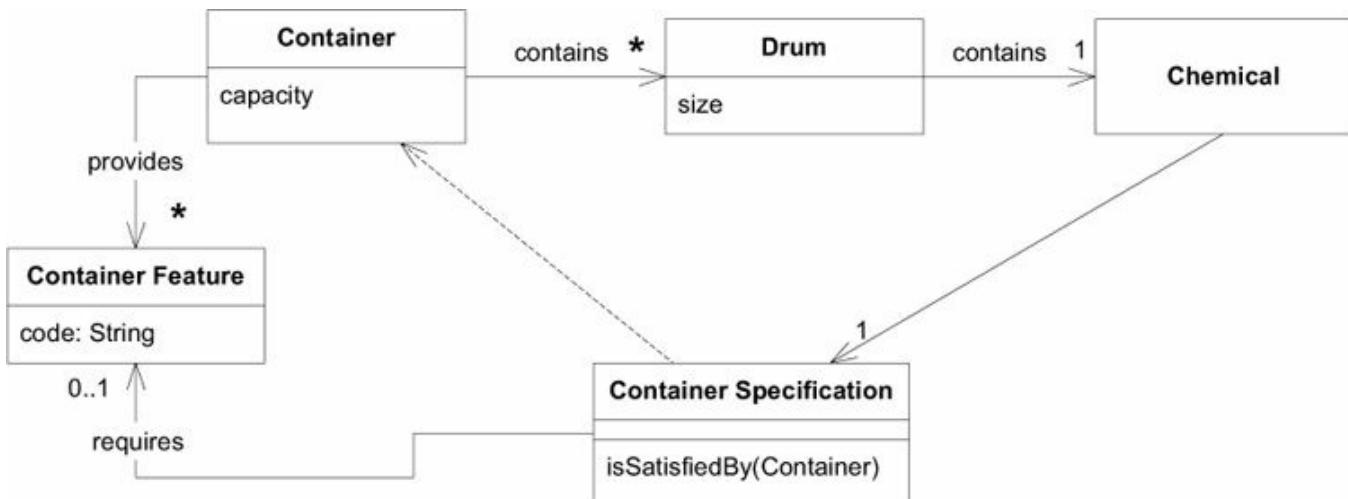
    public Set selectSatisfying( InvoiceSpecification spec) {
        return spec.satisfyingElementsFrom( this);
    }
}

public class DelinquentInvoiceSpecification {
    //Basic DelinquentInvoiceSpecification code here

    public Set satisfyingElementsFrom(
                    InvoiceRepository repository) {
        Collection pastDueInvoices =
            repository.selectWhereDueDateIsBefore( currentDate);

        Set delinquentInvoices = new HashSet();
        Iterator it = pastDueInvoices.iterator();
        while (it.hasNext()) {
            Invoice anInvoice = (Invoice) it.next();
            if (this.isSatisfiedBy( anInvoice))
                delinquentInvoices.add( anInvoice);
        }
        return delinquentInvoices;
    }
}

```

Chemical	Container Specification
TNT	Armored container
Sand	
Biological Samples	Must not share container with explosives
Ammonia	Ventilated container

Container Features	Contents	Specification Satisfied?
Armored	20 lbs. TNT 500 lbs. sand	✓
	50 lbs. biological samples	✓
	Ammonia	✗

`isSatisfied()`

```

public class ContainerSpecification {
    private ContainerFeature requiredFeature;
    public ContainerSpecification( ContainerFeature required) {
        requiredFeature = required;
    }

    boolean isSatisfiedBy( Container aContainer){
        return aContainer.getFeatures().contains( requiredFeature);
    }
}
  
```

```

tnt.setContainerSpecification(
    new ContainerSpecification( ARMORED ) ;

        isSafelyPacked() ,


boolean isSafelyPacked( ){
    Iterator it = contents.iterator();
    while (it.hasNext()) {
        Drum drum = (Drum) it.next();
        if (!drum.containerSpecification().isSatisfiedBy( this ))
            return false;
    }
    return true;
}

Iterator it = containers.iterator();
while (it.hasNext()) {
    Container container = (Container) it.next();
    if (!container.isSafelyPacked())
        unsafeContainers.add( container );
}

public interface WarehousePacker {
    public void pack( Collection containersToFill,
        Collection drumsToPack) throws NoAnswerFoundException;

    /* ASSERTION: At end of pack(), the ContainerSpecification
       of each Drum shall be satisfied by its Container.
       If no complete solution can be found, an exception shall
       be thrown. */
}

```

```

public class Container {
    private double capacity;
    private Set contents; //Drums

    public boolean hasSpaceFor( Drum aDrum) {
        return remainingSpace() >= aDrum.getSize();
    }

    public double remainingSpace() {
        double totalContentSize = 0.0;
        Iterator it = contents.iterator();
        while (it.hasNext()) {
            Drum aDrum = (Drum) it.next();
            totalContentSize = totalContentSize + aDrum.getSize();
        }
        return capacity - totalContentSize;
    }

    public boolean canAccommodate( Drum aDrum) {
        return hasSpaceFor(aDrum) &&
            aDrum.getContainerSpecification().isSatisfiedBy( this);
    }
}

public class PrototypePacker implements WarehousePacker {

    public void pack( Collection containers, Collection drums)
        throws NoAnswerFoundException {

        /* This method fulfills the ASSERTION as written. However,
           when an exception is thrown, Containers' contents may
           have changed. Rollback must be handled at a higher
           level. */

        Iterator it = drums.iterator();
        while (it.hasNext()) {
            Drum drum = (Drum) it.next();
            Container container =
                findContainerFor( containers, drum);
            container.add( drum);
        }
    }

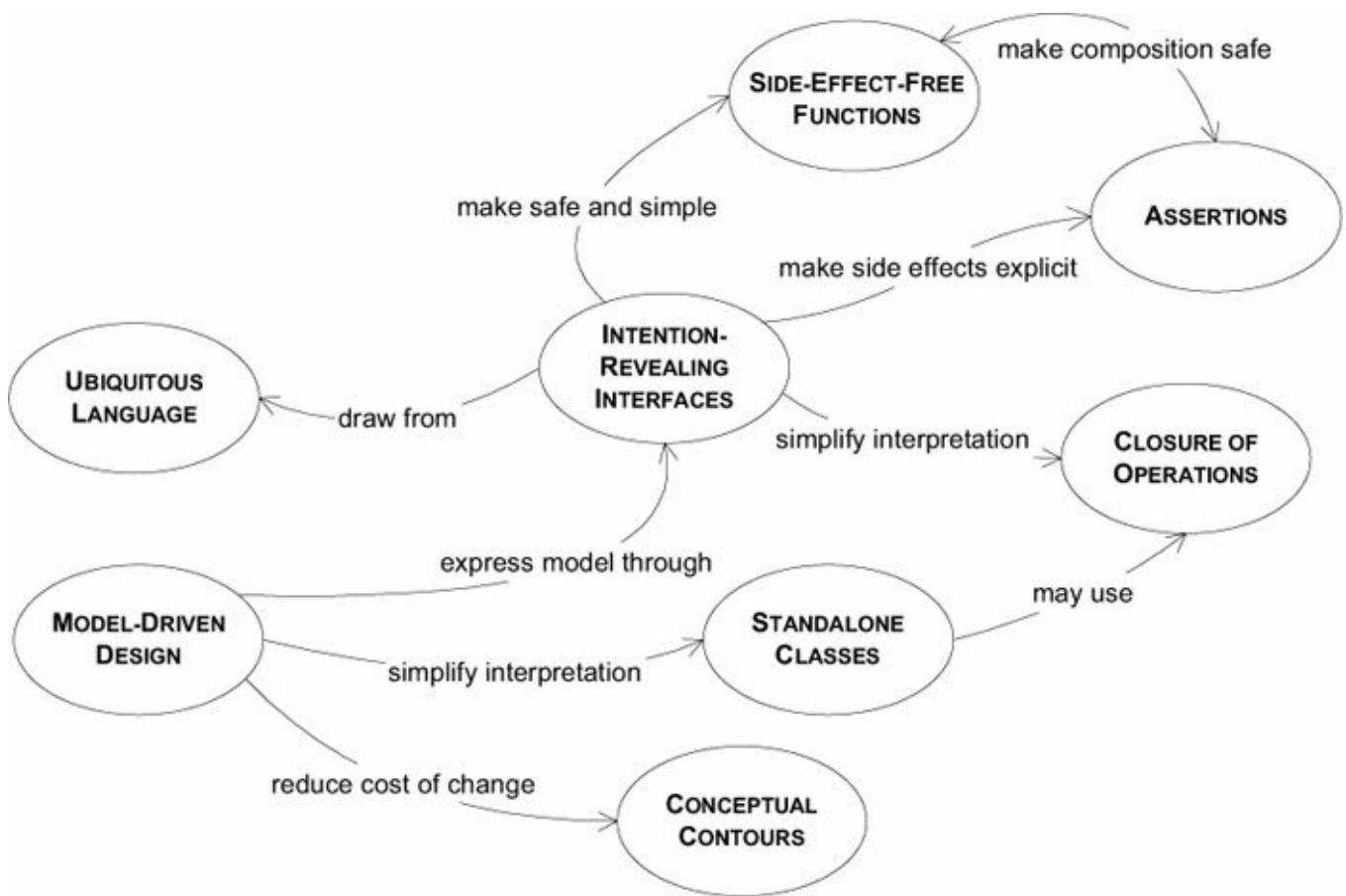
    public Container findContainerFor(
        Collection containers, Drum drum)
        throws NoAnswerFoundException {
        Iterator it = containers.iterator();
        while (it.hasNext()) {

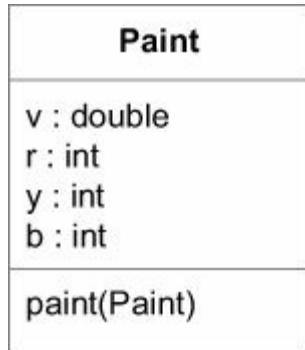
```

```
        Container container = (Container) it.next();
        if (container.canAccommodate( drum ))
            return container;
    }
    throw new NoAnswerFoundException();
}


```







```

    paint(Paint)

public void paint(Paint paint) {
    v = v + paint.getV(); //After mixing, volume is summed
    // Omitted many lines of complicated color mixing logic
    // ending with the assignment of new r, b, and y values.
}

```

```

public void testPaint() {
    // Create a pure yellow paint with volume=100
    Paint yellow = new Paint(100.0, 0, 50, 0);
    // Create a pure blue paint with volume=100
    Paint blue = new Paint(100.0, 0, 0, 50);

    // Mix the blue into the yellow
    yellow.paint(blue);

    // Result should be volume of 200.0 of green paint
    assertEquals(200.0, yellow.getV(), 0.01);
    assertEquals(25, yellow.getB());
    assertEquals(25, yellow.getY());
    assertEquals(0, yellow.getR());
}

```

```

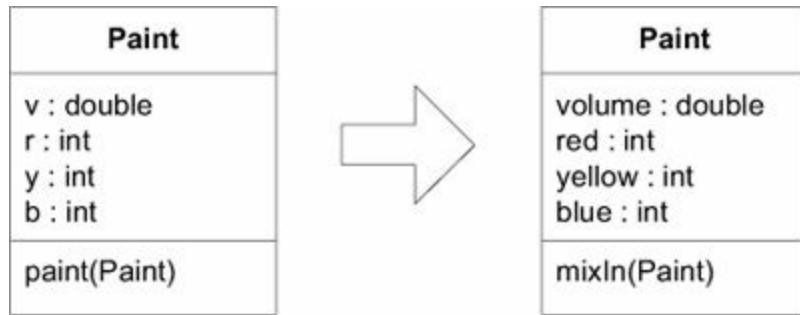
public void testPaint() {
    // Start with a pure yellow paint with volume=100
    Paint ourPaint = new Paint(100.0, 0, 50, 0);
    // Take a pure blue paint with volume=100
    Paint blue = new Paint(100.0, 0, 0, 50);

    // Mix the blue into the yellow
    ourPaint.mixIn(blue);

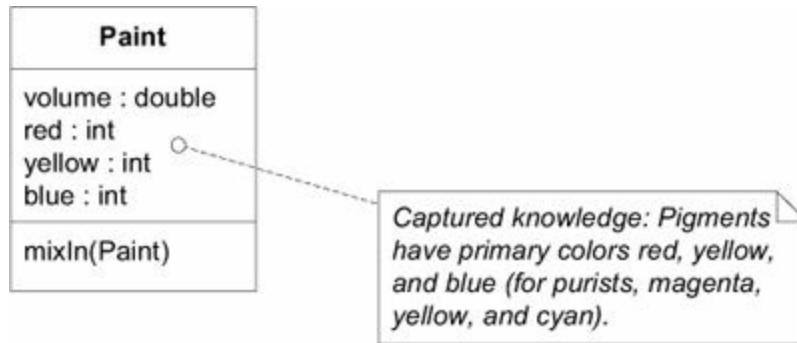
    // Result should be volume of 200.0 of green paint
    assertEquals(200.0, ourPaint.getVolume(), 0.01);
}

```

```
assertEquals( 25, ourPaint.getBlue() );
assertEquals( 25, ourPaint.getYellow() );
assertEquals( 0, ourPaint.getRed() );
}
```

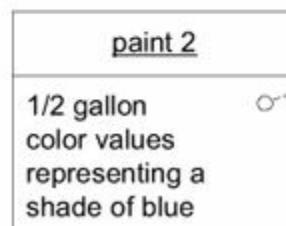
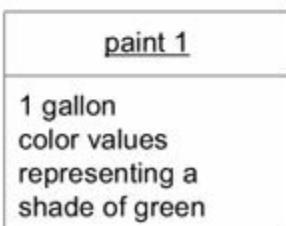
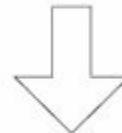
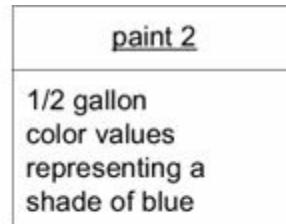
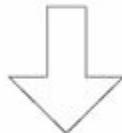
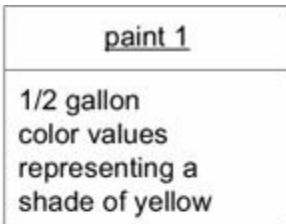


* * *



```
public void mixIn(Paint other) {
    volume = volume.plus(other.getVolume());
    // Many lines of complicated color-mixing logic
    // ending with the assignment of new red, blue,
    // and yellow values.
}
```

mixIn(paint2)

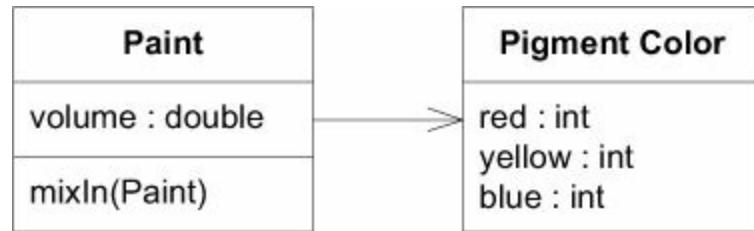


*What should happen here?
The original developers seem to have been uninterested and never specified.*

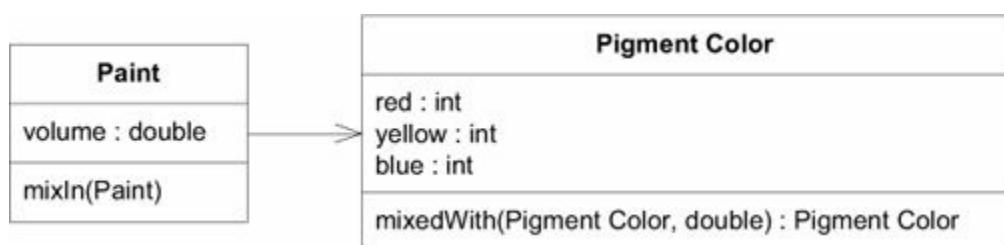
mixIn()

mixIn()

mixIn()



`mixIn()`



```

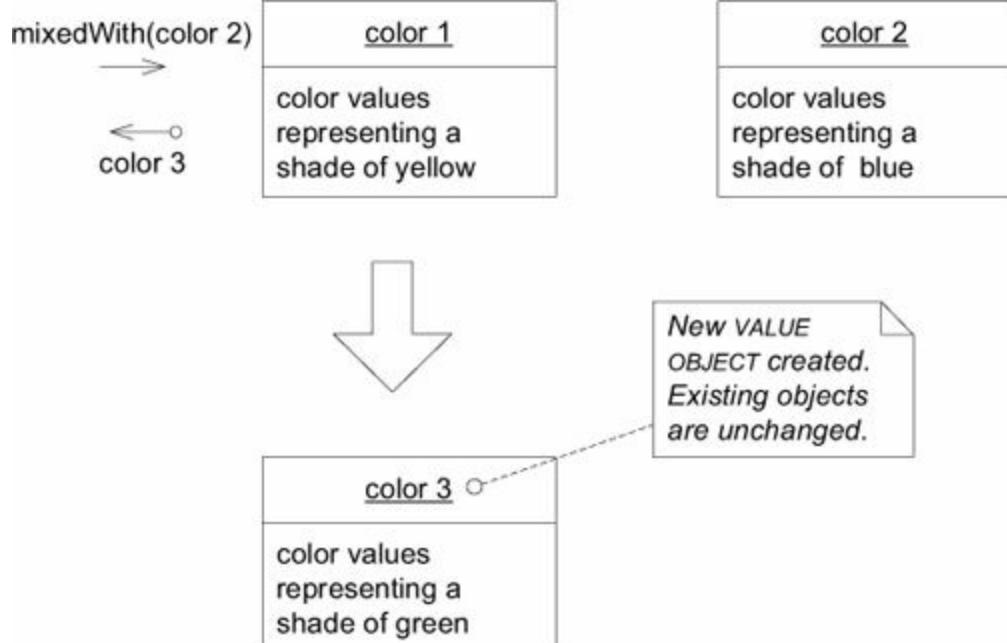
public class PigmentColor {

    public PigmentColor mixedWith( PigmentColor other,
                                    double ratio) {
        // Many lines of complicated color-mixing logic
        // ending with the creation of a new PigmentColor object
        // with appropriate new red, blue, and yellow values.
    }
}

public class Paint {

    public void mixIn( Paint other) {
        volume = volume + other.getVolume();
        double ratio = other.getVolume() / volume;
        pigmentColor =
            pigmentColor.mixedWith( other.pigmentColor(), ratio);
    }
}

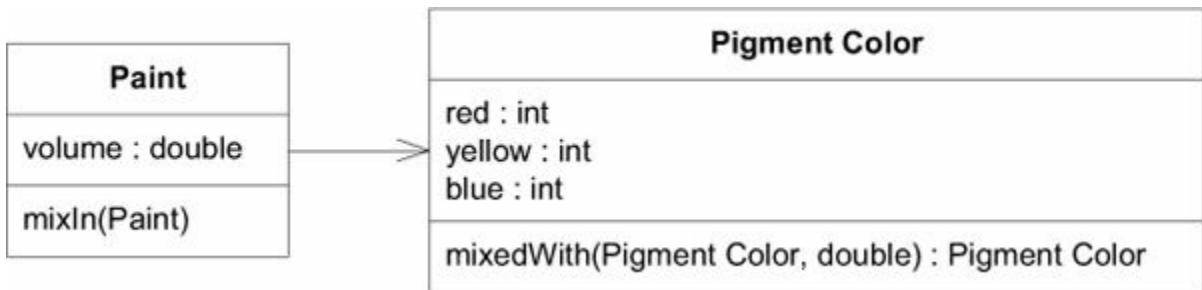
```



* * *

* * *

`mixIn(Paint)`



```
mixIn()

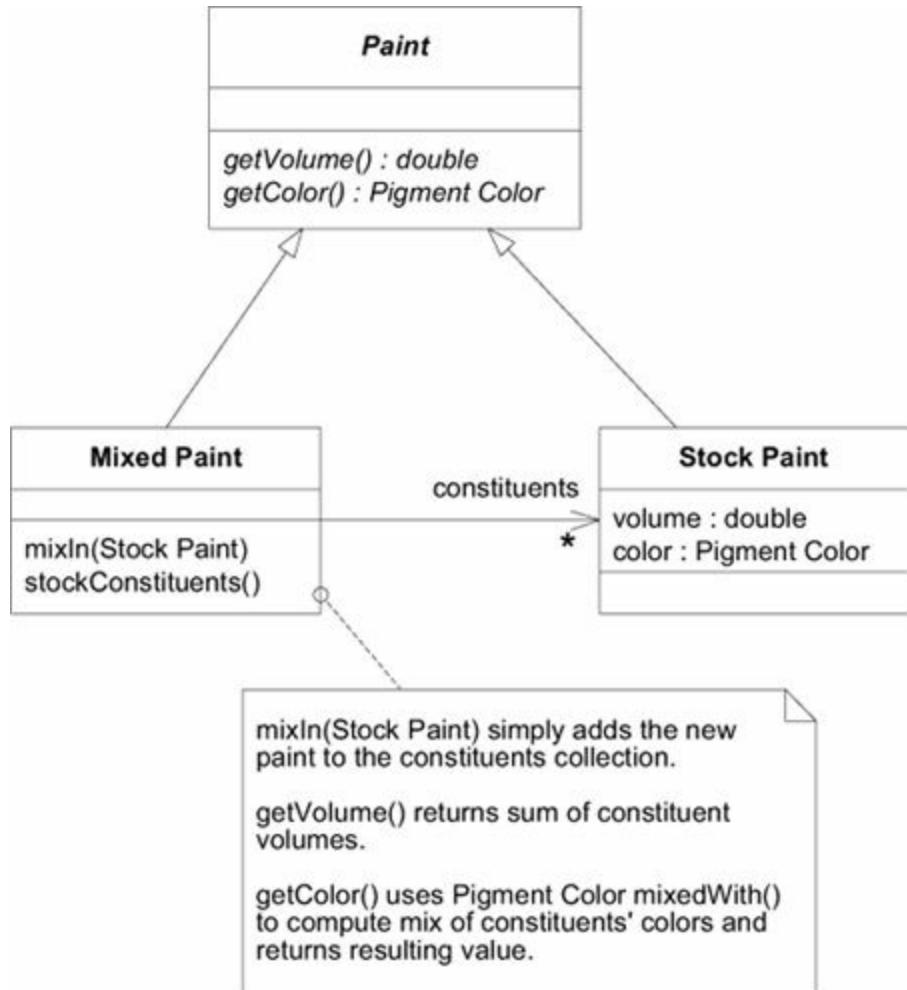
p1.mixIn(p2):
    p1.volume          p2.volume
    p2.volume
```

mixIn()

```
public void testMixingVolume {
    PigmentColor yellow = new PigmentColor(0, 50, 0);
    PigmentColor blue = new PigmentColor(0, 0, 50);

    StockPaint paint1 = new StockPaint(1.0, yellow);
    StockPaint paint2 = new StockPaint(1.5, blue);
    MixedPaint mix = new MixedPaint();

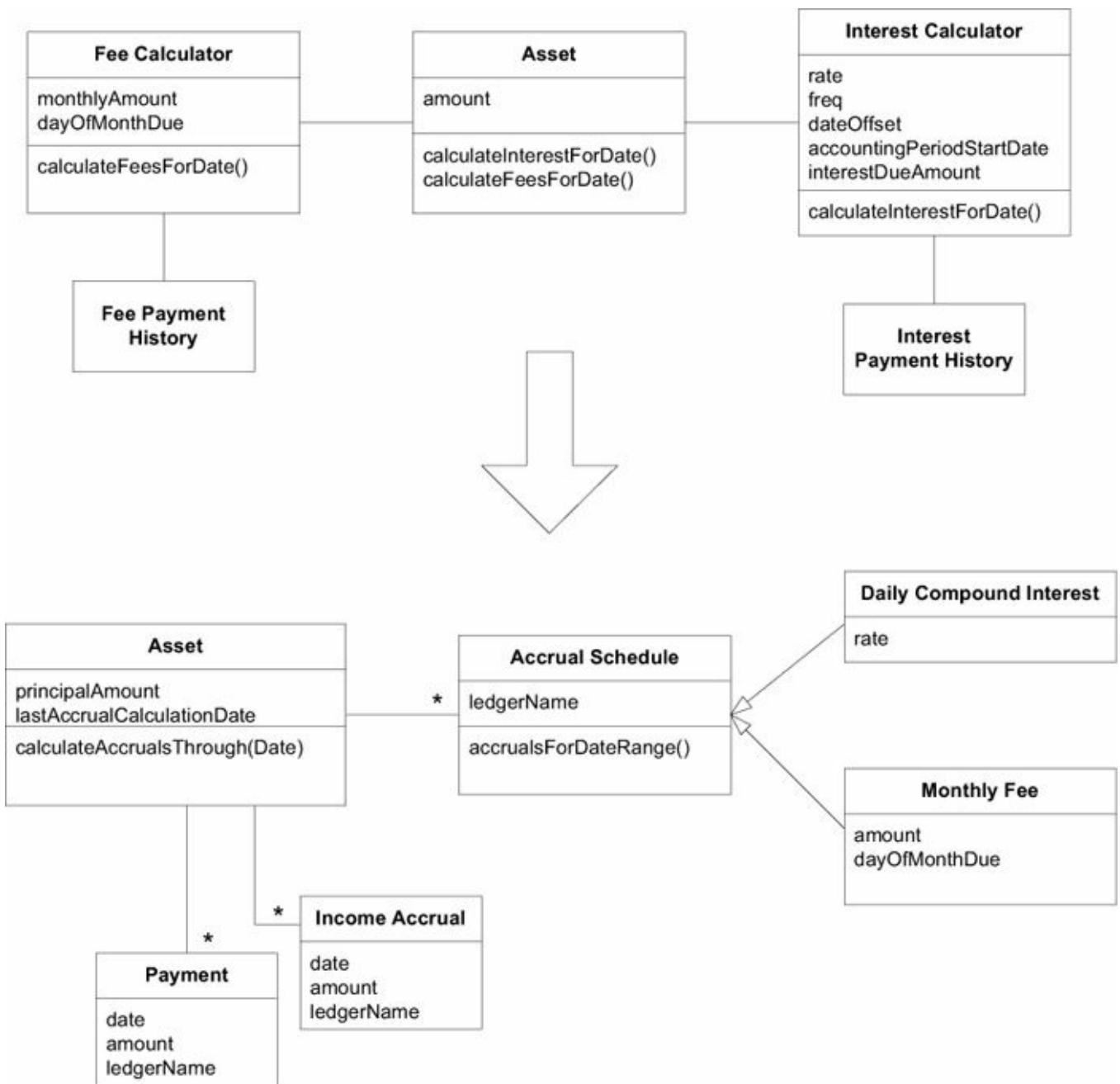
    mix.mixIn(paint1);
    mix.mixIn(paint2);
    assertEquals(2.5, mix.getVolume(), 0.01);
}
```

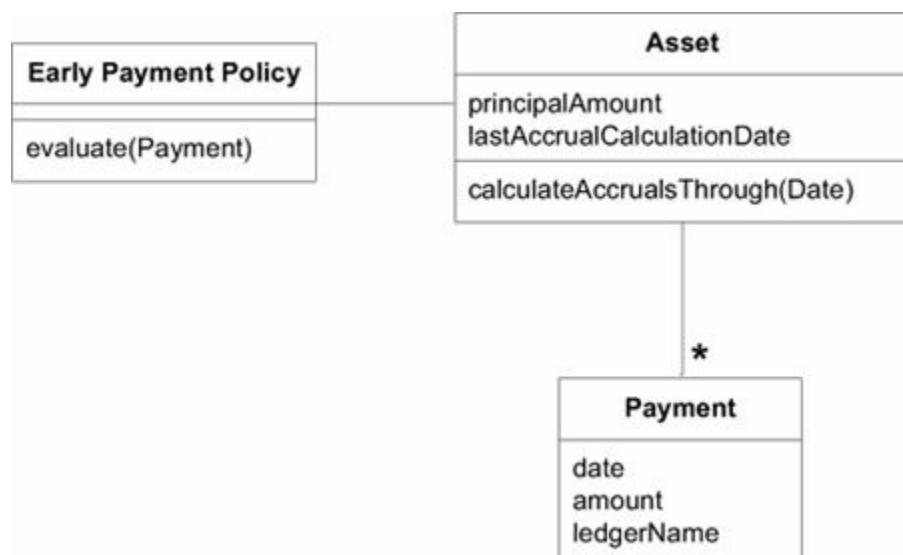
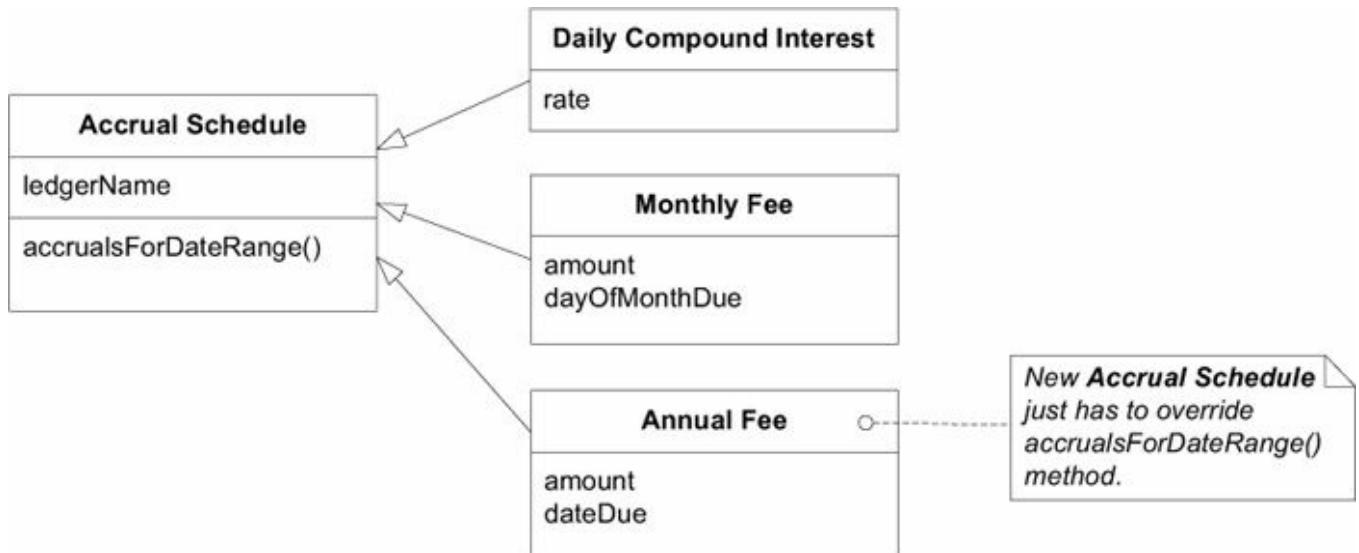


* * *

add()

-





size() int

＊＊＊

```
mixedWith()
```

```
Set employees = (some Set of Employee objects);
Set lowPaidEmployees = new HashSet();
Iterator it = employees.iterator();
while (it.hasNext()) {
    Employee anEmployee = it.next();
    if (anEmployee.salary() < 40000)
        lowPaidEmployees.add(anEmployee);
}

employees := (some Set of Employee objects).
lowPaidEmployees := employees select:
    [:anEmployee | anEmployee salary < 40000].
```

* * *



```
public interface Specification {  
    boolean isSatisfiedBy( Object candidate);  
}
```

```
public class ContainerSpecification implements Specification {  
    private ContainerFeature requiredFeature;  
  
    public ContainerSpecification( ContainerFeature required) {  
        requiredFeature = required;  
    }  
  
    boolean isSatisfiedBy( Object candidate){  
        if (!candidate instanceof Container) return false;  
  
        return  
    (Container) candidate.getFeatures().contains( requiredFeature);  
    }  
}  
  
  
public interface Specification {  
    boolean isSatisfiedBy( Object candidate);  
  
    Specification and( Specification other);
```

```
Specification or( Specification other);
Specification not();
}

Specification ventilated = new ContainerSpecification( VENTILATED );
Specification armored = new ContainerSpecification( ARMORED );
Specification both = ventilated.and( armored );

Specification ventilatedType1 =
    new ContainerSpecification( VENTILATED_TYPE_1 );
Specification ventilatedType2 =
    new ContainerSpecification( VENTILATED_TYPE_2 );

Specification either = ventilatedType1.or( ventilatedType2 );
```

```
Specification cheap = ( ventilated.not()).and( armored.not());
```

```
public abstract class AbstractSpecification implements
    Specification {
    public Specification and( Specification other) {
        return new AndSpecification(this, other);
    }
    public Specification or( Specification other) {
        return new OrSpecification(this, other);
    }
    public Specification not() {
        return new NotSpecification(this);
    }
}

public class AndSpecification extends AbstractSpecification {
    Specification one;
    Specification other;
```

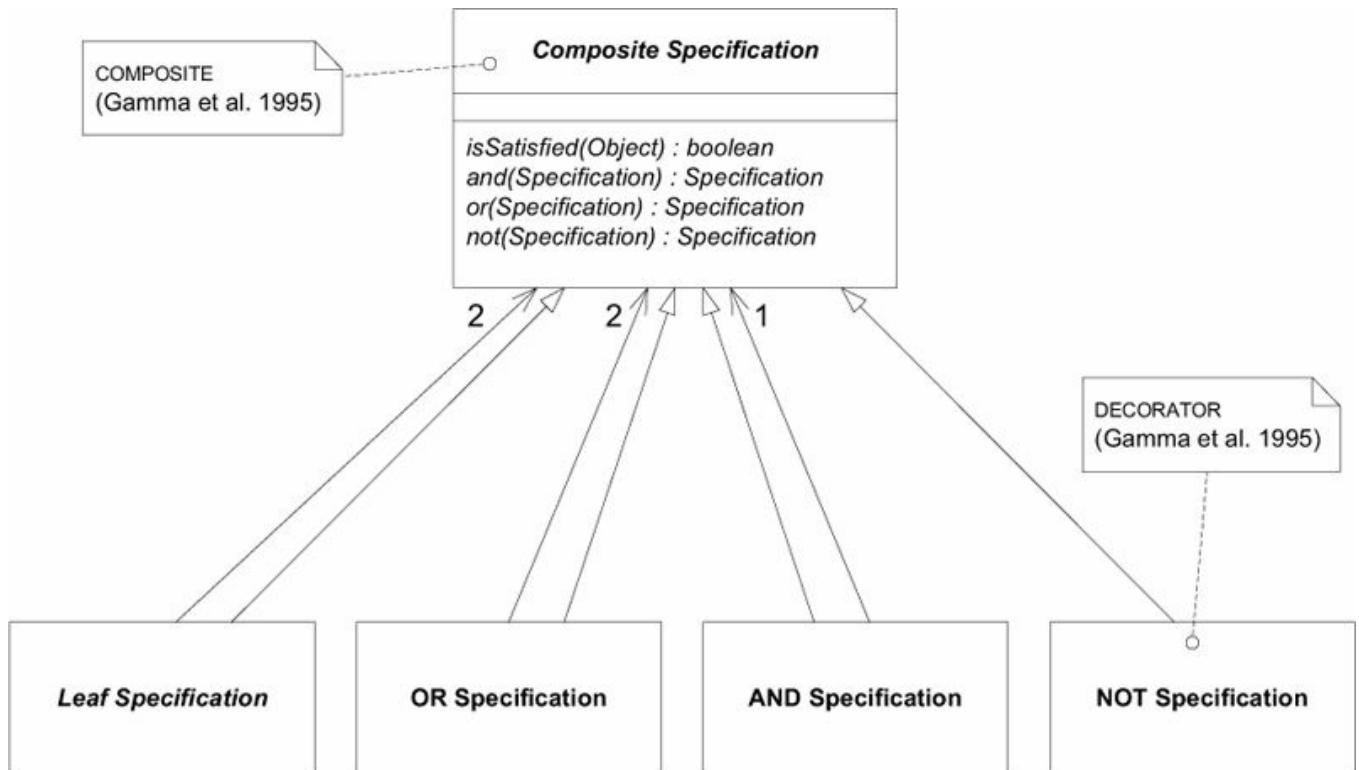
```

public AndSpecification( Specification x, Specification y) {
    one = x;
    other = y;
}
public boolean isSatisfiedBy( Object candidate) {
    return one.isSatisfiedBy( candidate) &&
        other.isSatisfiedBy( candidate);
}
}

public class OrSpecification extends AbstractSpecification {
    Specification one;
    Specification other;
    public OrSpecification( Specification x, Specification y) {
        one = x;
        other = y;
    }
    public boolean isSatisfiedBy( Object candidate) {
        return one.isSatisfiedBy( candidate) ||
            other.isSatisfiedBy( candidate);
    }
}

public class NotSpecification extends AbstractSpecification {
    Specification wrapped;
    public NotSpecification( Specification x) {
        wrapped = x;
    }
    public boolean isSatisfiedBy( Object candidate) {
        return !wrapped.isSatisfiedBy( candidate);
    }
}

```



SPECIFICATION Stack Content for “Cheap Container”

Top	AndSpecificationOperator (FLY WEIGHT)
	NotSpecificationOperator (FLY WEIGHT)
	Armored
	NotSpecificationOperator
	Ventilated

and(not(armored) , not(ventilated))

boolean subsumes(Specification other);

subsumes(old)



true

new : Container Specification

ventilated AND armored

subsumes(new)



false

old : Container Specification

ventilated



→

```
public class MinimumAgeSpecification {  
    int threshold;  
  
    public boolean isSatisfiedBy( Person candidate) {  
        return candidate.getAge( ) >= threshold;  
    }  
  
    public boolean subsumes( MinimumAgeSpecification other) {  
        return threshold >= other.getThreshold();  
    }  
}
```

```
drivingAge = new MinimumAgeSpecification( 16);  
votingAge = new MinimumAgeSpecification( 18);  
assertTrue( votingAge. subsumes( drivingAge));
```

```
public interface Specification {  
    boolean isSatisfiedBy( Object candidate);  
    Specification and( Specification other);  
    boolean subsumes( Specification other);  
}
```

→

→

```
public boolean subsumes( Specification other) {  
    if (other instanceof CompositeSpecification) {  
        Collection otherLeaves =  
            ( CompositeSpecification) other. leafSpecifications();  
        Iterator it = otherLeaves.iterator();  
        while (it.hasNext()) {  
            if (!leafSpecifications(). contains( it. next()))  
                return false;  
        }  
    } else {  
        if (!leafSpecifications(). contains( other))  
            return false;
```

```
    }  
    return true;  
}
```

Aristotle on SPECIFICATIONS

All men are mortal.

```
Specification manSpec = new ManSpecification();  
Specification mortalSpec = new MortalSpecification();  
assert manSpec.subsumes(mortalSpec);
```

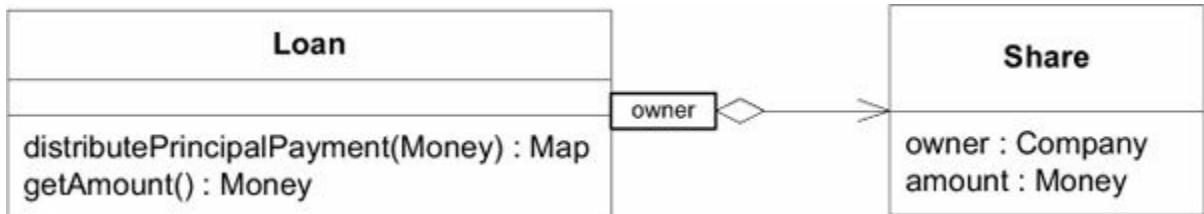
Aristotle is a man.

```
Man aristotle = new Man();  
assert manSpec.isSatisfiedBy(aristotle);
```

Therefore, Aristotle is mortal.

```
assert mortalSpec.isSatisfiedBy(aristotle);
```





```

public class Loan {
    private Map shares;

    //Accessors, constructors, and very simple methods are excluded

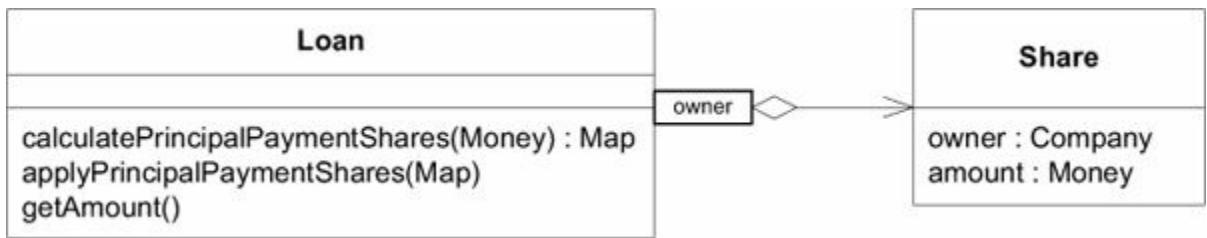
    public Map distributePrincipalPayment(double paymentAmount) {
        Map paymentShares = new HashMap();
        Map loanShares = getShares();
        double total = getAmount();
        Iterator it = loanShares.keySet().iterator();
        while( it.hasNext() ) {
            Object owner = it.next();
            double initialLoanShareAmount = getShareAmount( owner );
            double paymentShareAmount =
                initialLoanShareAmount / total * paymentAmount;
            Share paymentShare =
                new Share( owner, paymentShareAmount );
            paymentShares.put( owner, paymentShare );

            double newLoanShareAmount =
                initialLoanShareAmount - paymentShareAmount;
            Share newLoanShare =
                new Share( owner, newLoanShareAmount );
            loanShares.put( owner, newLoanShare );
        }
        return paymentShares;
    }

    public double getAmount() {
        Map loanShares = getShares();
        double total = 0.0;
        Iterator it = loanShares.keySet().iterator();
        while( it.hasNext() ) {
            Share loanShare = ( Share ) loanShares.get( it.next() );
            total = total + loanShare.getAmount();
        }
        return total;
    }
}

```

```
distributePaymentPrincipal()
```



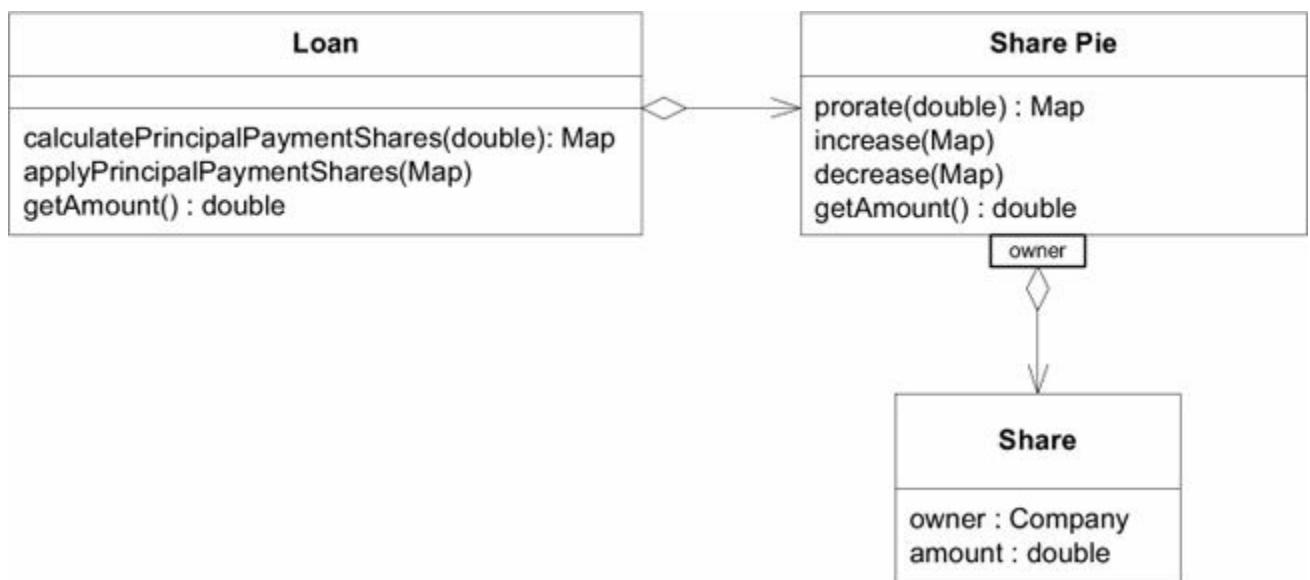
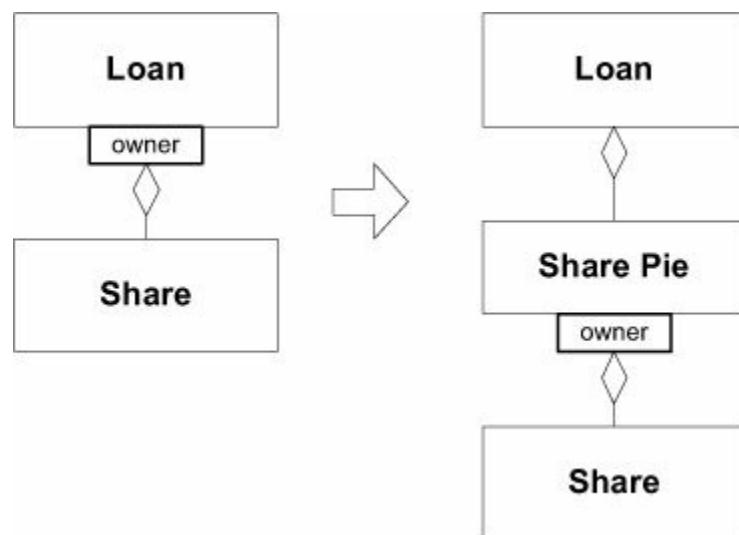
```
public void applyPrincipalPaymentShares( Map paymentShares) {
    Map loanShares = getShares();
    Iterator it = paymentShares.keySet().iterator();
    while( it.hasNext() ) {
        Object lender = it.next();
        Share paymentShare = ( Share) paymentShares.get( lender);
        Share loanShare = ( Share) loanShares.get( lender);
        double newLoanShareAmount = loanShare.getAmount() -
            paymentShare.getAmount();
        Share newLoanShare = new Share( lender, newLoanShareAmount);
        loanShares.put( lender, newLoanShare);
    }
}

public Map calculatePrincipalPaymentShares( double paymentAmount) {
    Map paymentShares = new HashMap();
    Map loanShares = getShares();
    double total = getAmount();
    Iterator it = loanShares.keySet().iterator();
    while( it.hasNext() ) {
        Object lender = it.next();
        Share loanShare = ( Share) loanShares.get( lender);
        double paymentShareAmount =
            loanShare.getAmount() / total * paymentAmount;
        Share paymentShare = new Share( lender, paymentShareAmount);
        paymentShares.put( lender, paymentShare);
    }
    return paymentShares;
}

Map distribution =
    aLoan.calculatePrincipalPaymentShares( paymentAmount);
aLoan.applyPrincipalPaymentShares( distribution);

applyDrawdown()

calculateFeePaymentShares()
```



```

public class Loan {
    private SharePie shares;

    //Accessors, constructors, and straightforward methods
    //are omitted
  
```

```

public Map calculatePrincipalPaymentDistribution(
    double paymentAmount) {
    return getShares().prorated(paymentAmount);
}
public void applyPrincipalPayment( Map paymentShares) {
    shares.decrease(paymentShares);
}
}

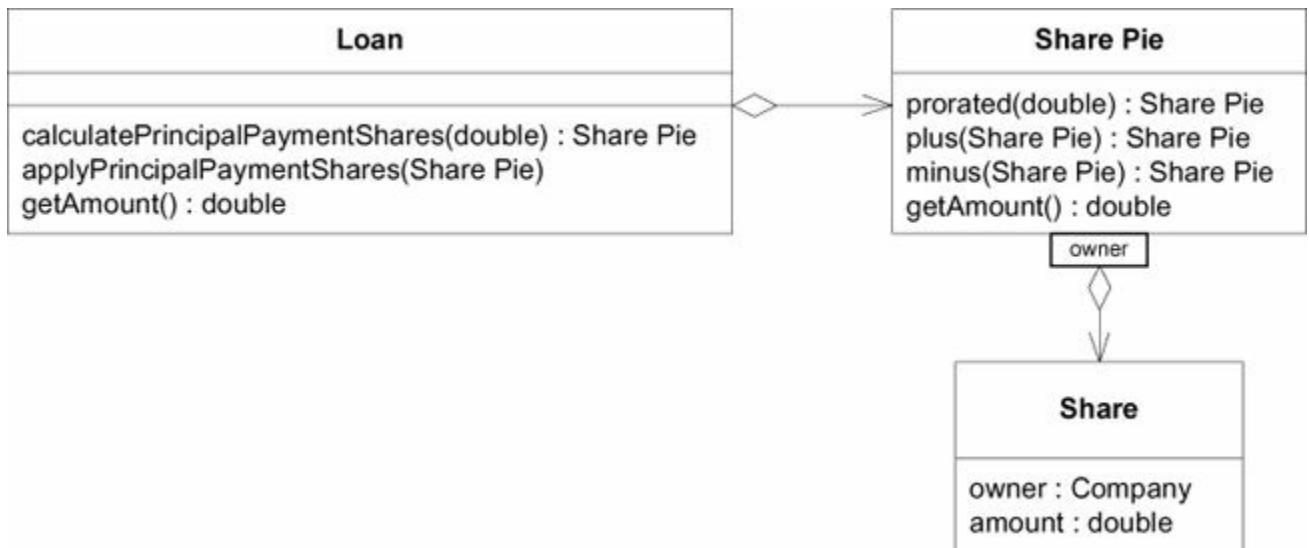
```

increase (Map) decrease (Map)

addShares (Map)

prorate()

prorated()



```

public class SharePie {
    private Map shares = new HashMap();
    //Accessors and other straightforward methods are omitted
    public double getAmount() {

```

```

        double total = 0.0;
        Iterator it = shares.keySet().iterator();
        while( it.hasNext() ) {
            Share loanShare = getShare( it.next() );
            total = total + loanShare.getAmount();
        }
        return total;
    }

```

The whole is equal to the sum of its parts.

```

public SharePie minus( SharePie otherShares) {
    SharePie result = new SharePie();
    Set owners = new HashSet();
    owners.addAll( getOwners() );
    owners.addAll( otherShares.getOwners() );
    Iterator it = owners.iterator();
difference
    while( it.hasNext() ) {
        Object owner = it.next();
        double resultShareAmount = getShareAmount( owner ) -
            otherShares.getShareAmount( owner );
        result.add( owner, resultShareAmount );
    }
    return result;
}

```

The difference between two Pies is the

between each owner's share.

```

public SharePie plus( SharePie otherShares) {
    //Similar to implementation of minus()
of
}

```

The combination of two Pies is the combination of each owner's share.

```

public SharePie prorated( double amountToProrate) {
    SharePie proration = new SharePie();
    double basis = getAmount();
divided
    Iterator it = shares.keySet().iterator();
    while( it.hasNext() ) {
        Object owner = it.next();
        Share share = getShare( owner );
        double proratedShareAmount =
            share.getAmount() / basis * amountToProrate;
        proration.add( owner, proratedShareAmount );
    }
    return proration;
}

```

An amount can be proportionately among all shareholders.

```

public class Loan {
    private SharePie shares;

    //Accessors, constructors, and straightforward methods
    //are omitted

    public SharePie calculatePrincipalPaymentDistribution(
        double paymentAmount) {

```

```
        return shares.prorated( paymentAmount) ;
    }

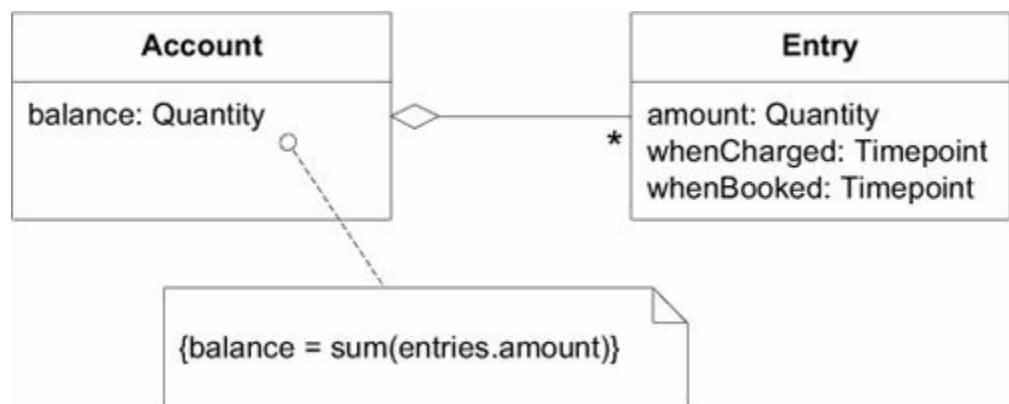
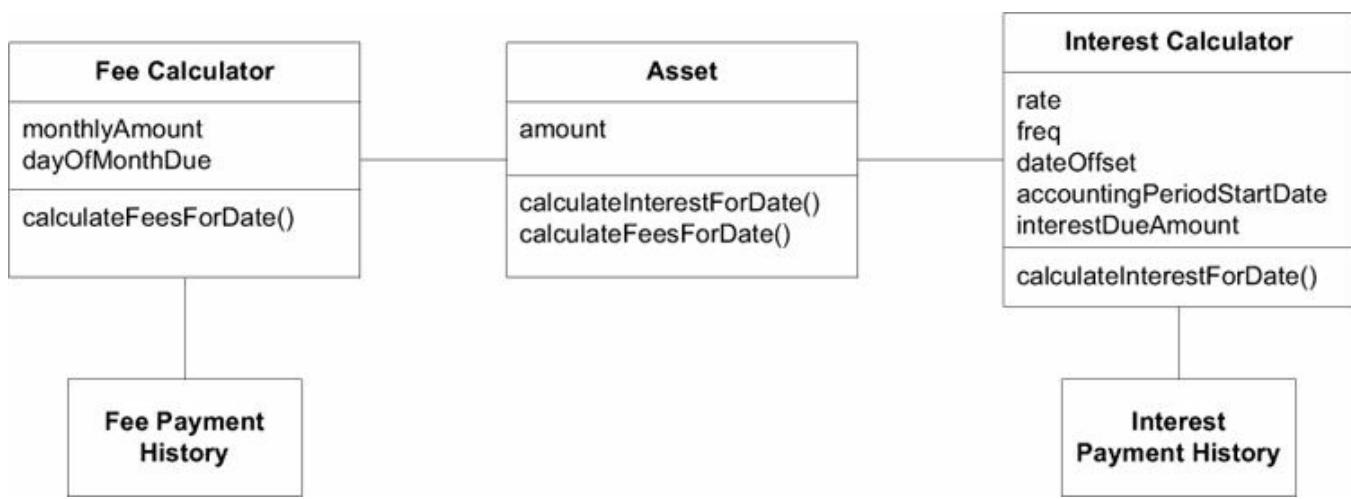
    public void applyPrincipalPayment( SharePie paymentShares) {
        setShares( shares.minus( paymentShares)) ;
    }

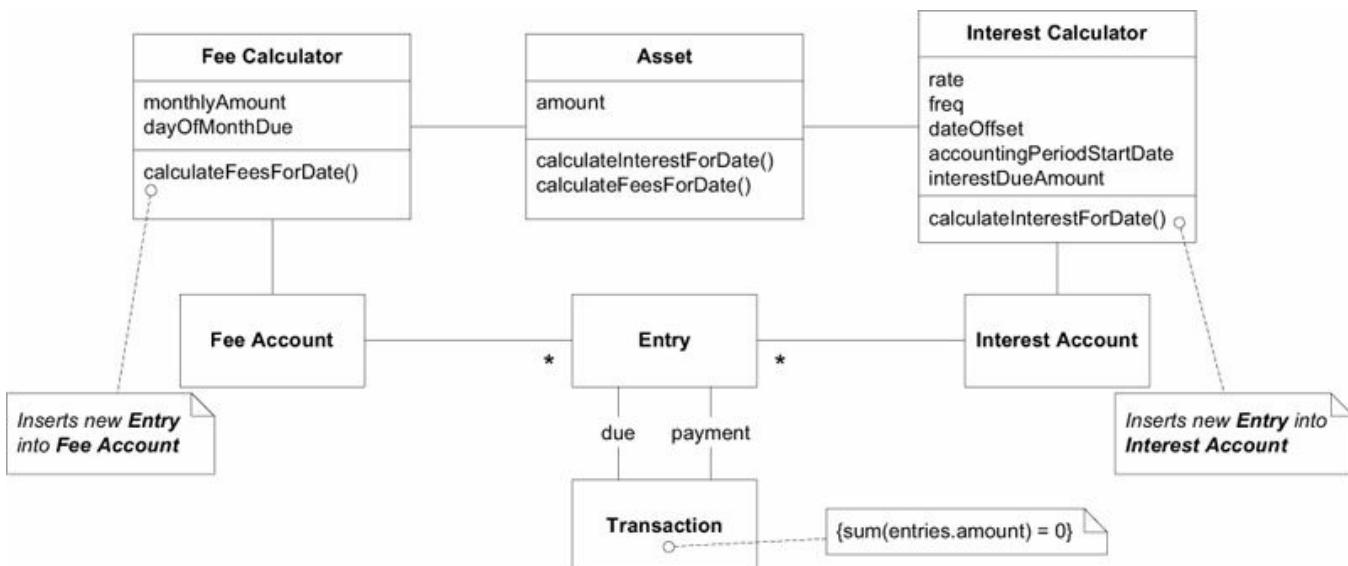
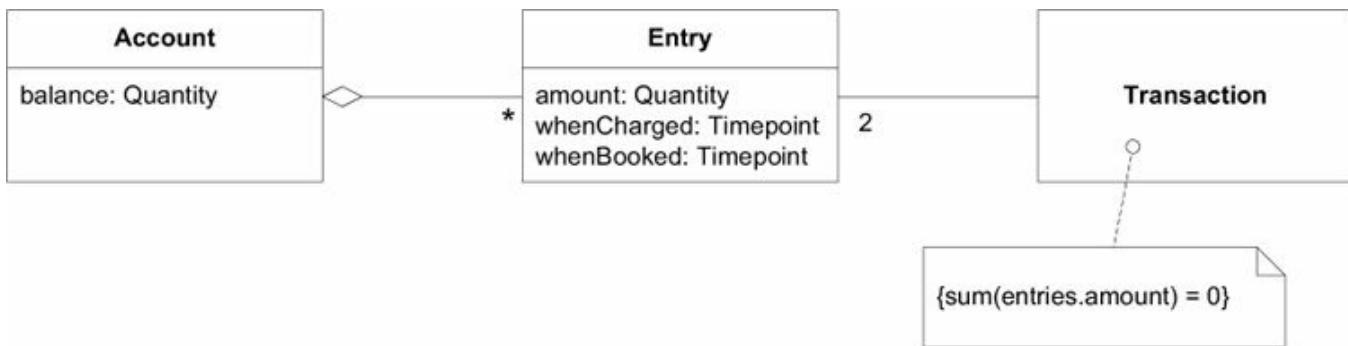
public class Facility {
    private SharePie shares;
    . .
    public SharePie calculateDrawdownDefaultDistribution(
                    double drawdownAmount) {
        return shares.prorated( drawdownAmount);
    }
}

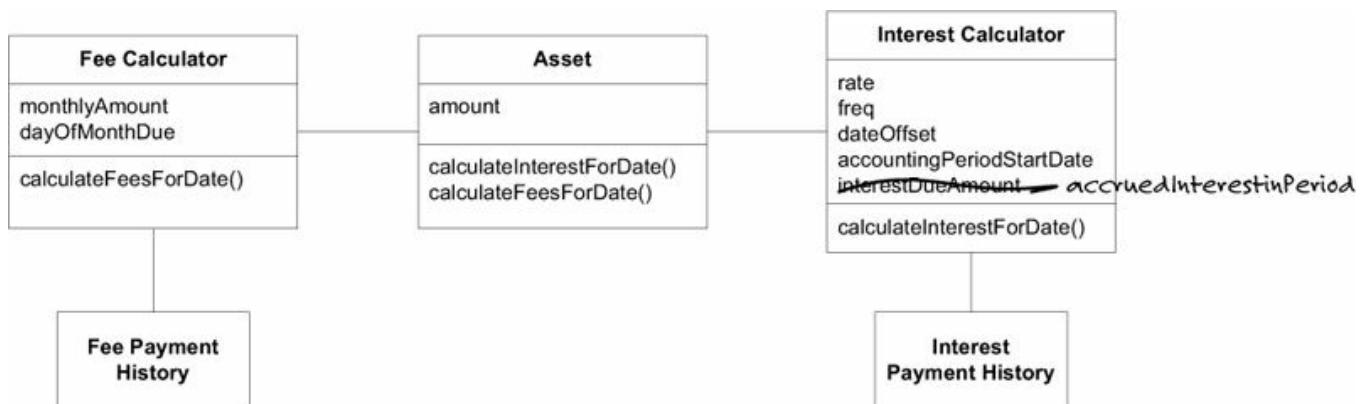
public class Loan {
    . .
    public void applyDrawdown( SharePie drawdownShares) {
        setShares( shares.plus( drawdownShares));
    }
}

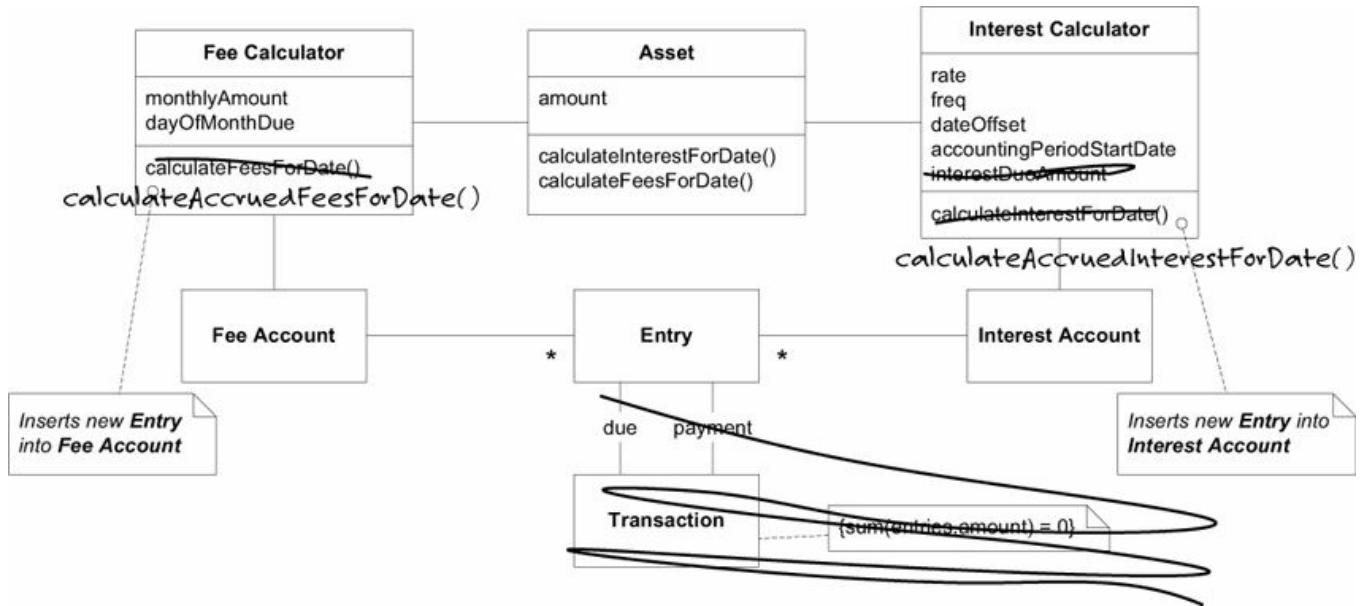
SharePie originalAgreement =
    aFacility.getShares().prorated( aLoan.getAmount());
SharePie actual = aLoan.getShares();
SharePie deviation = actual.minus( originalAgreement);

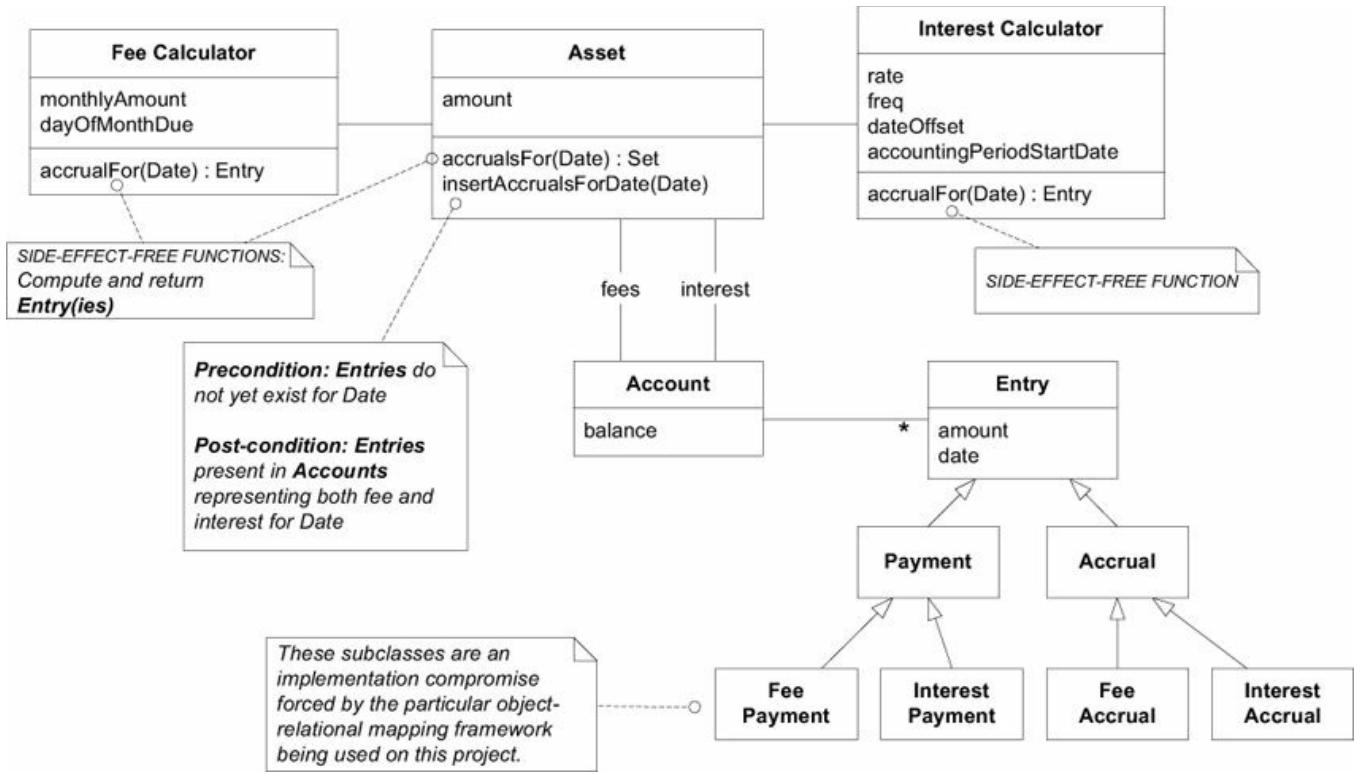
plus() minus()      prorated()
```

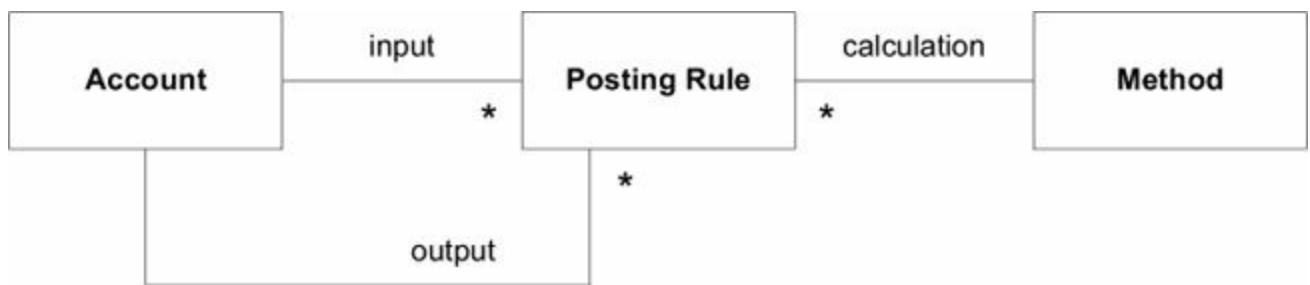


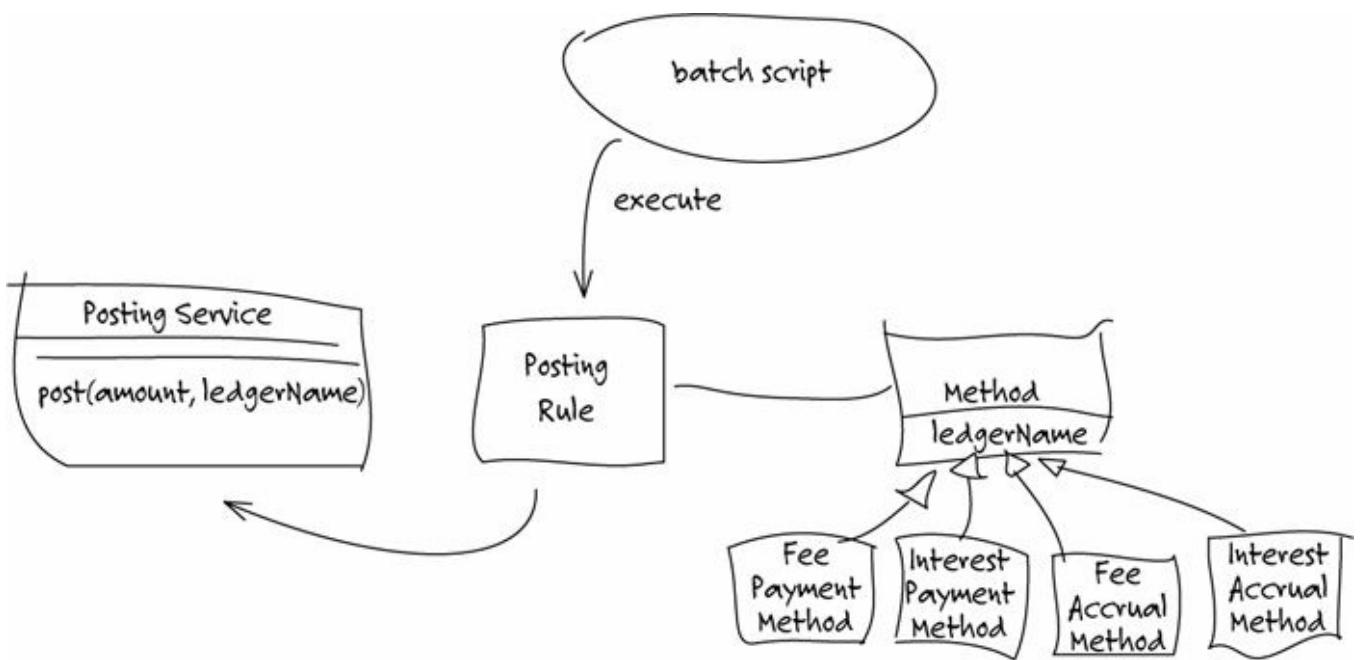


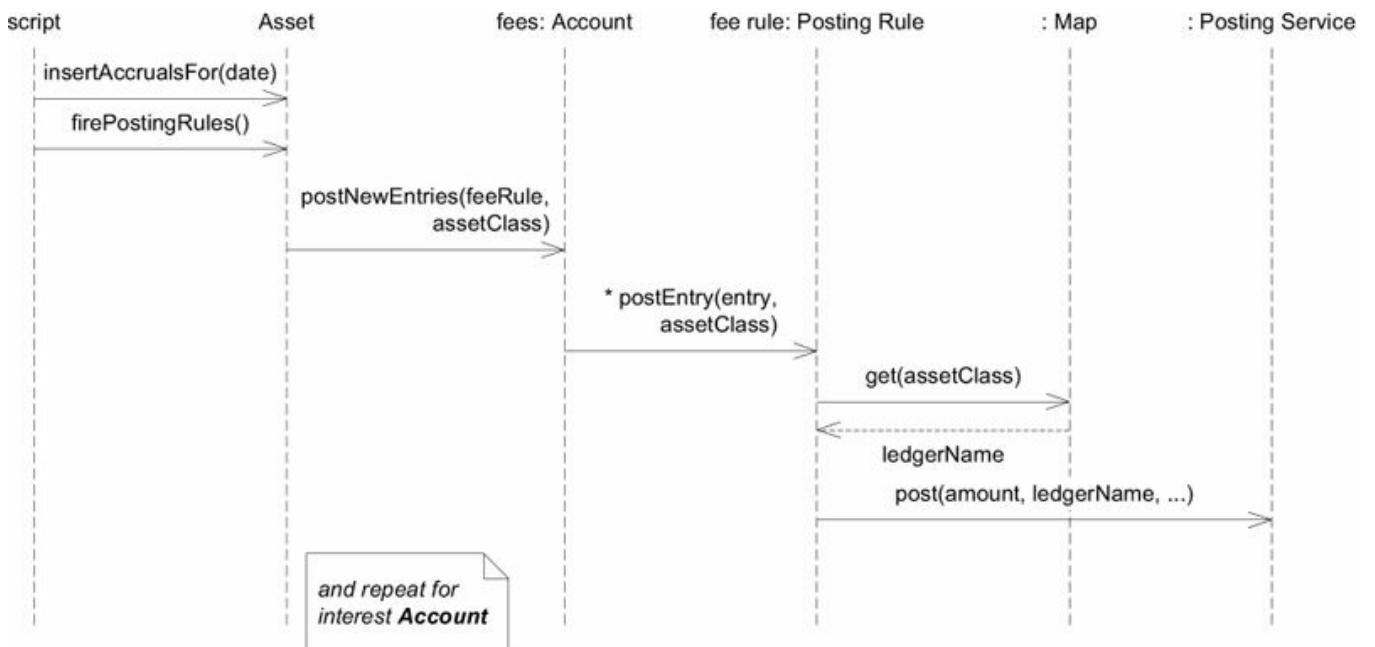
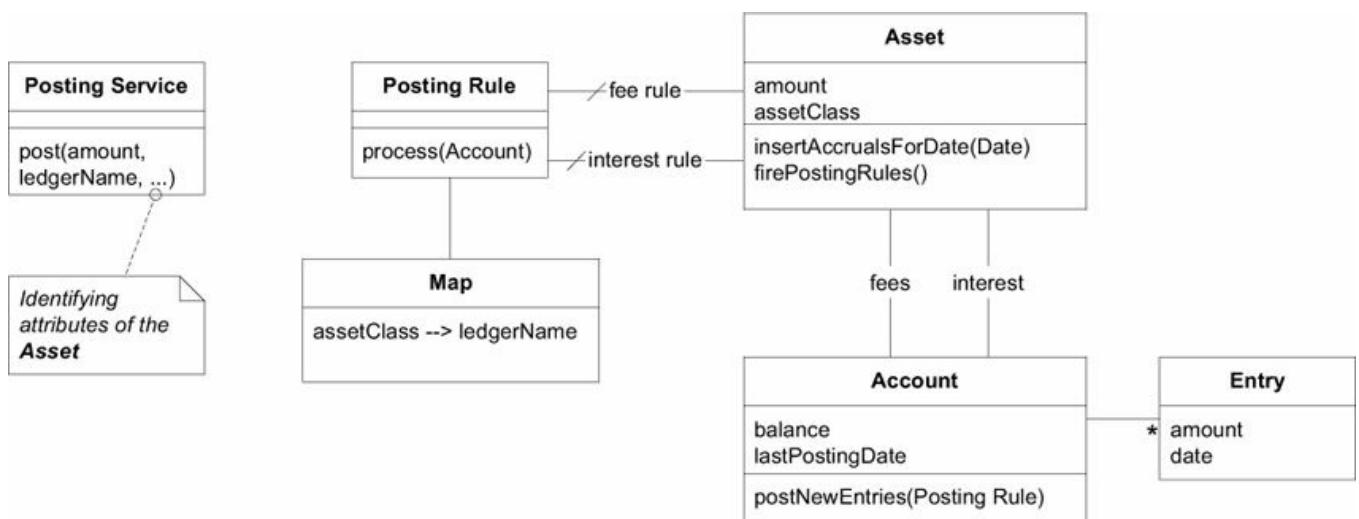


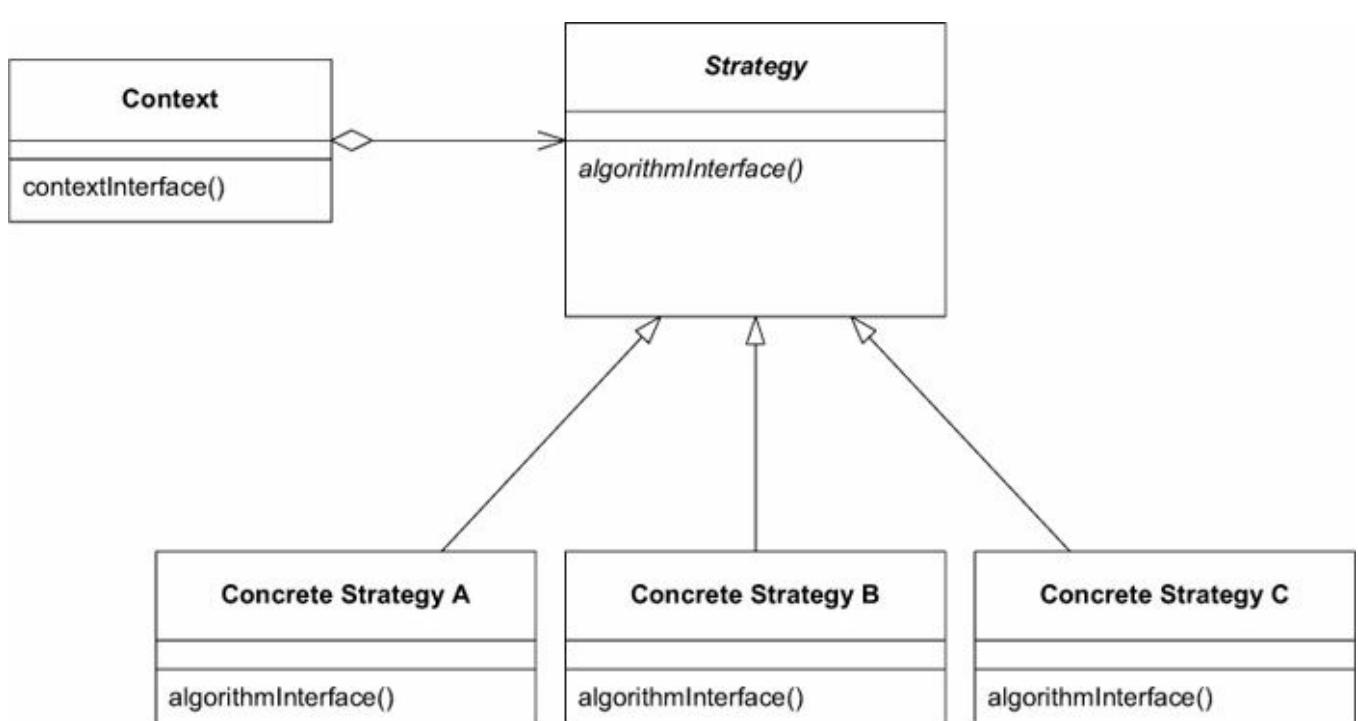


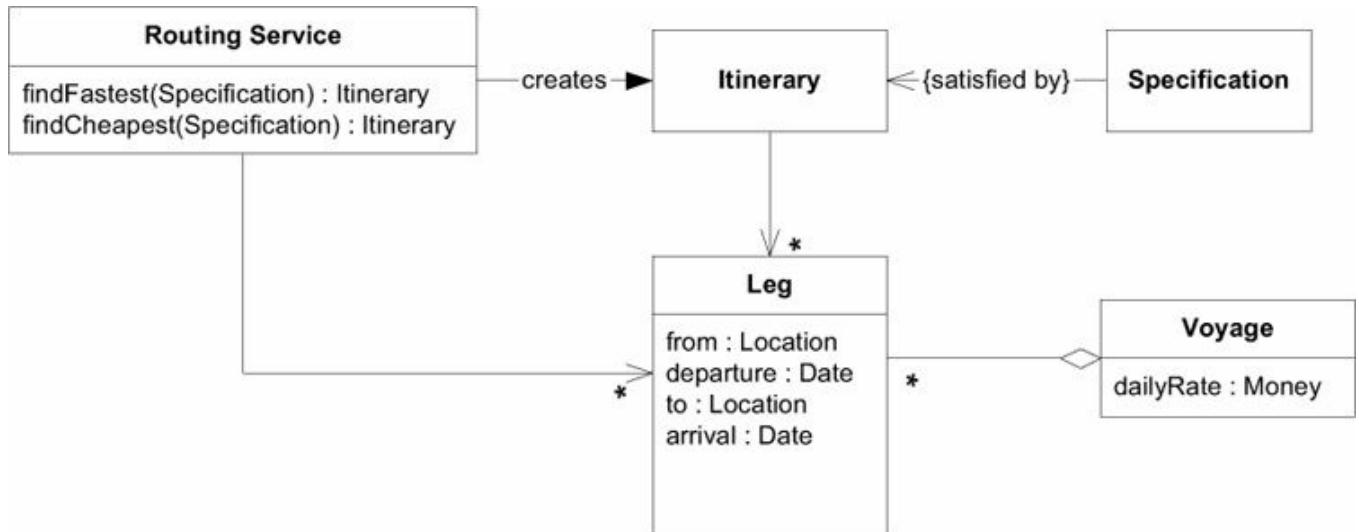


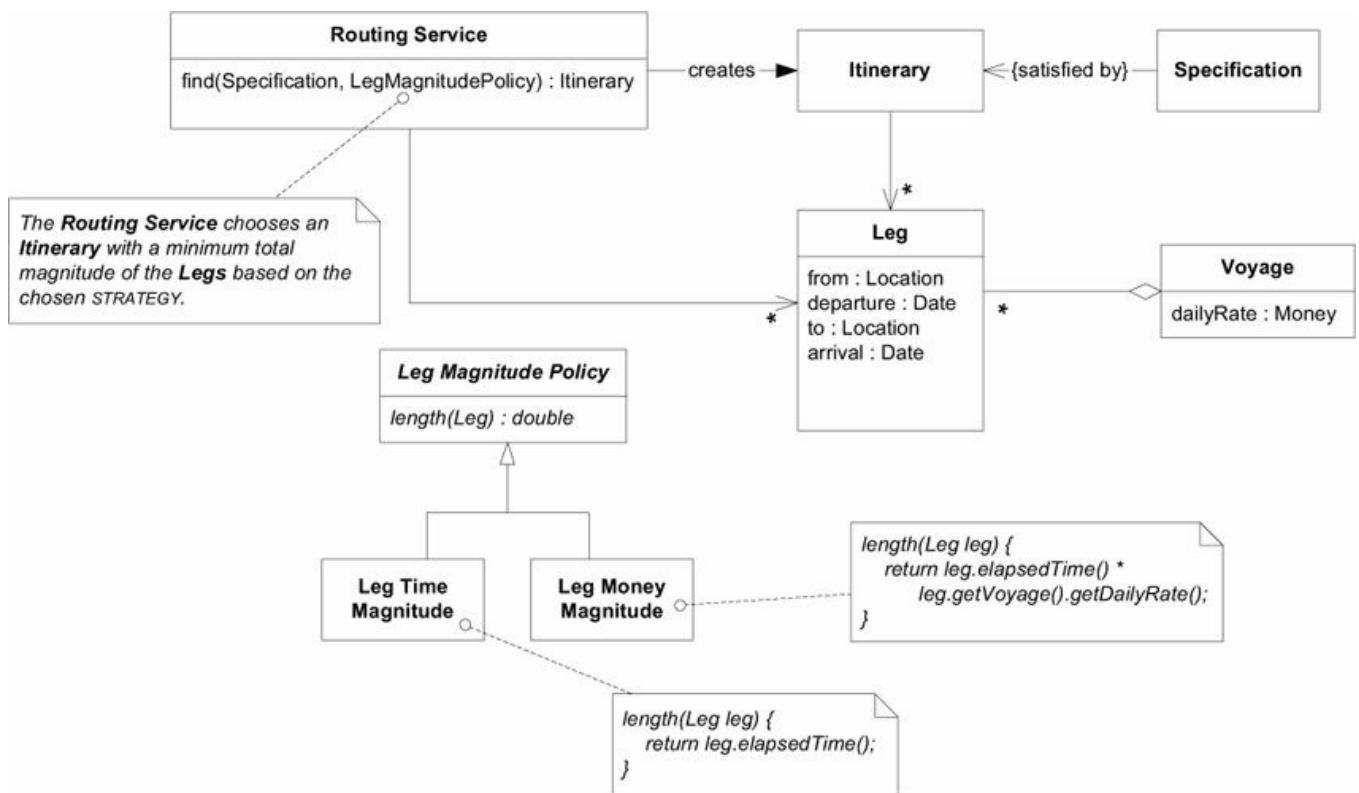




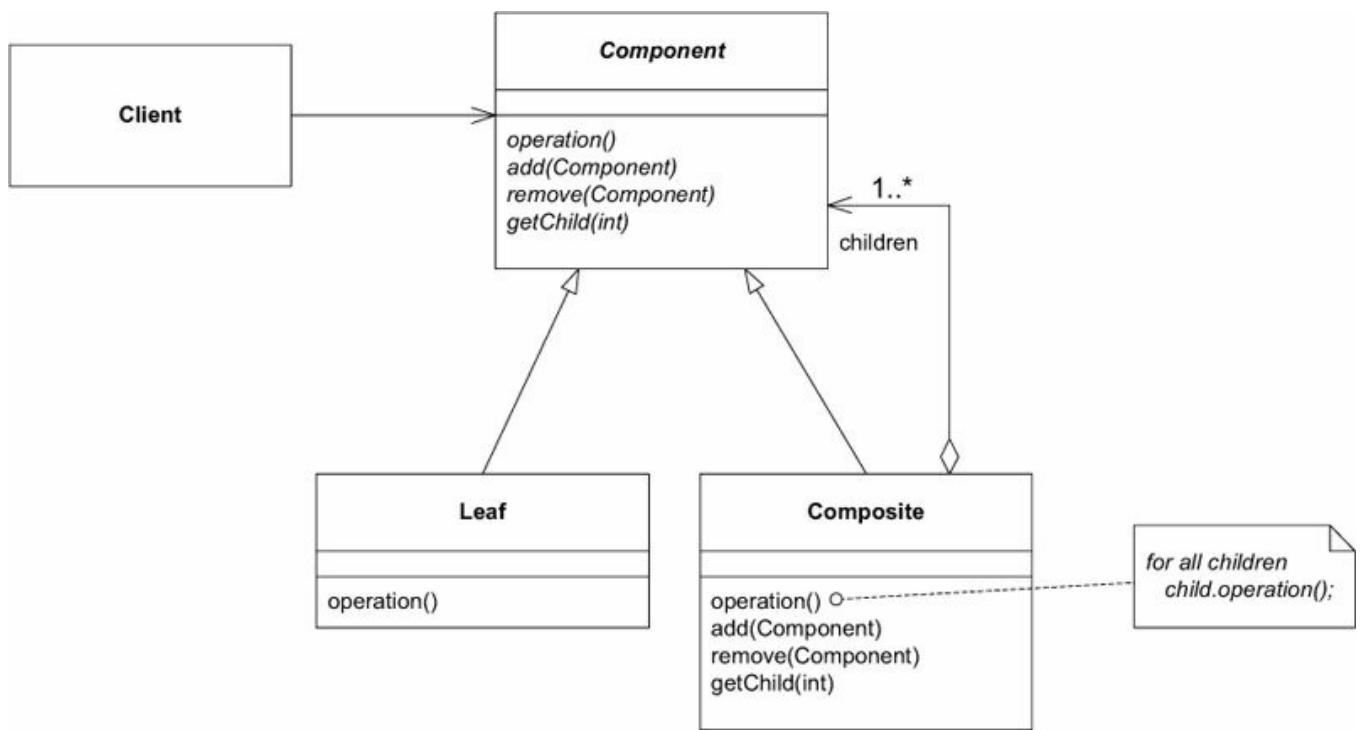


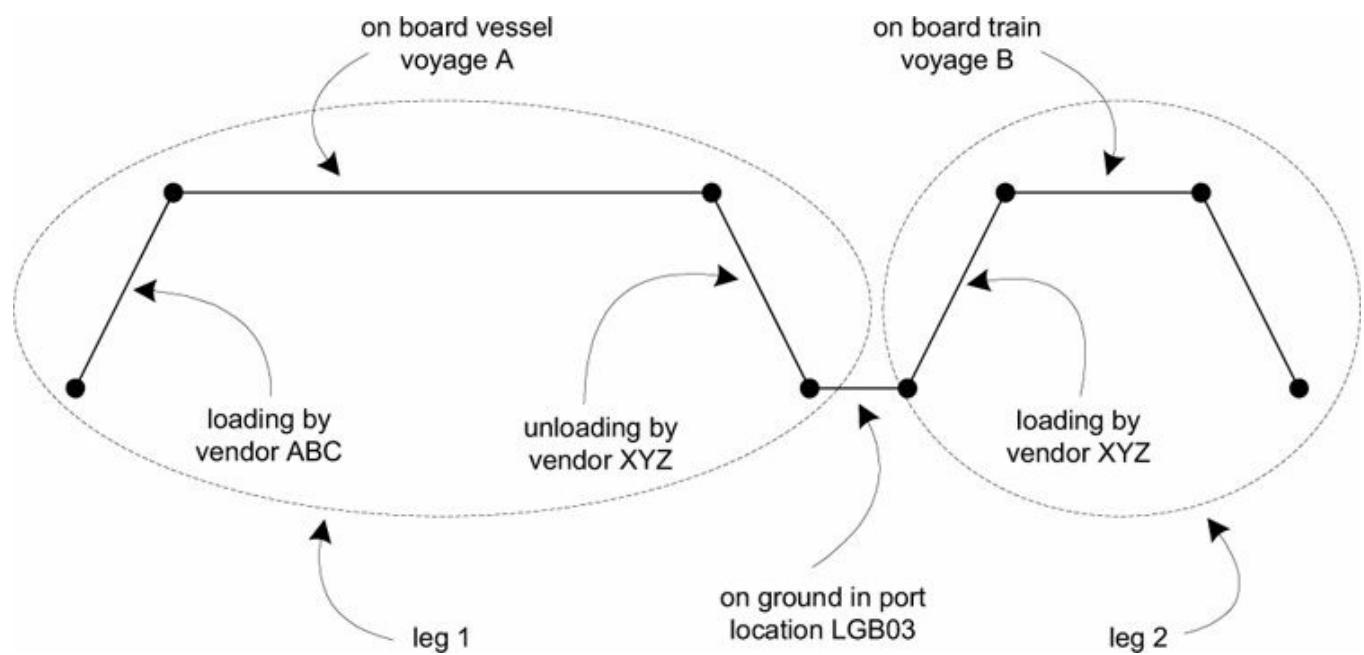


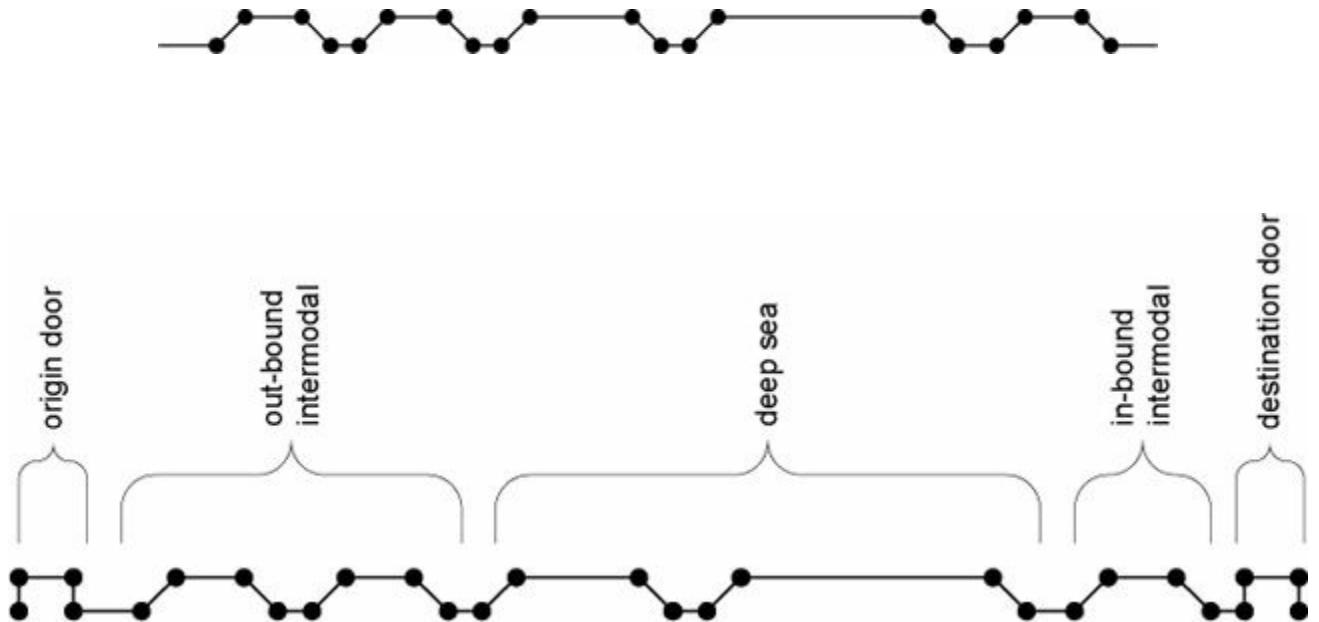
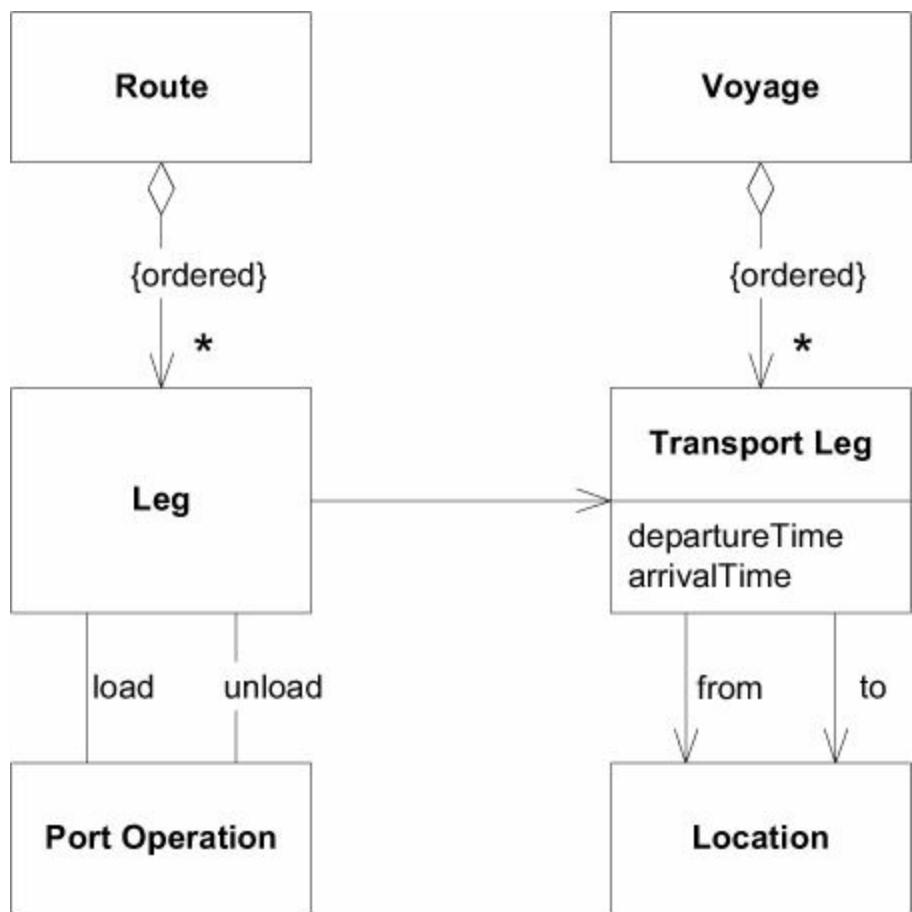


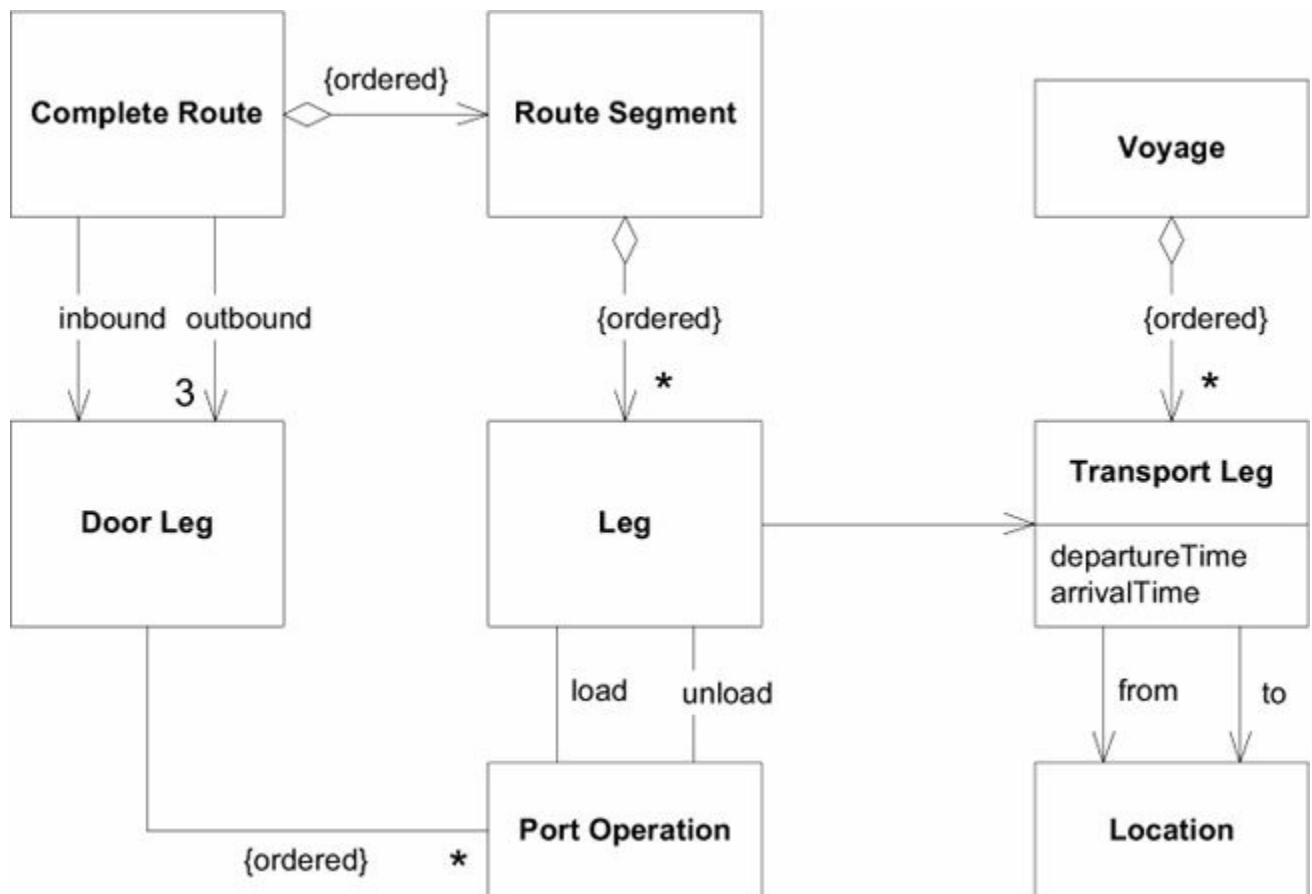


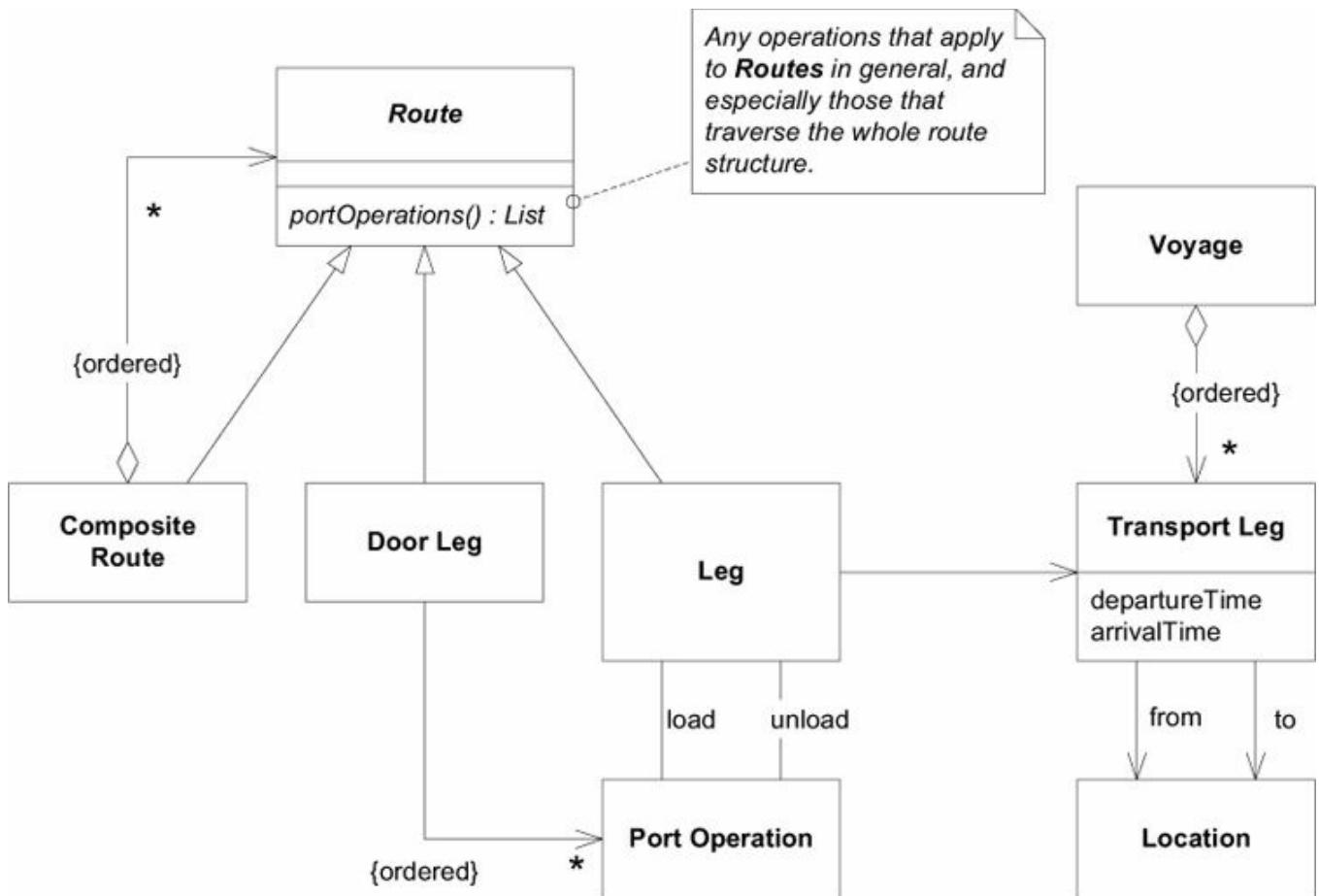
* * *

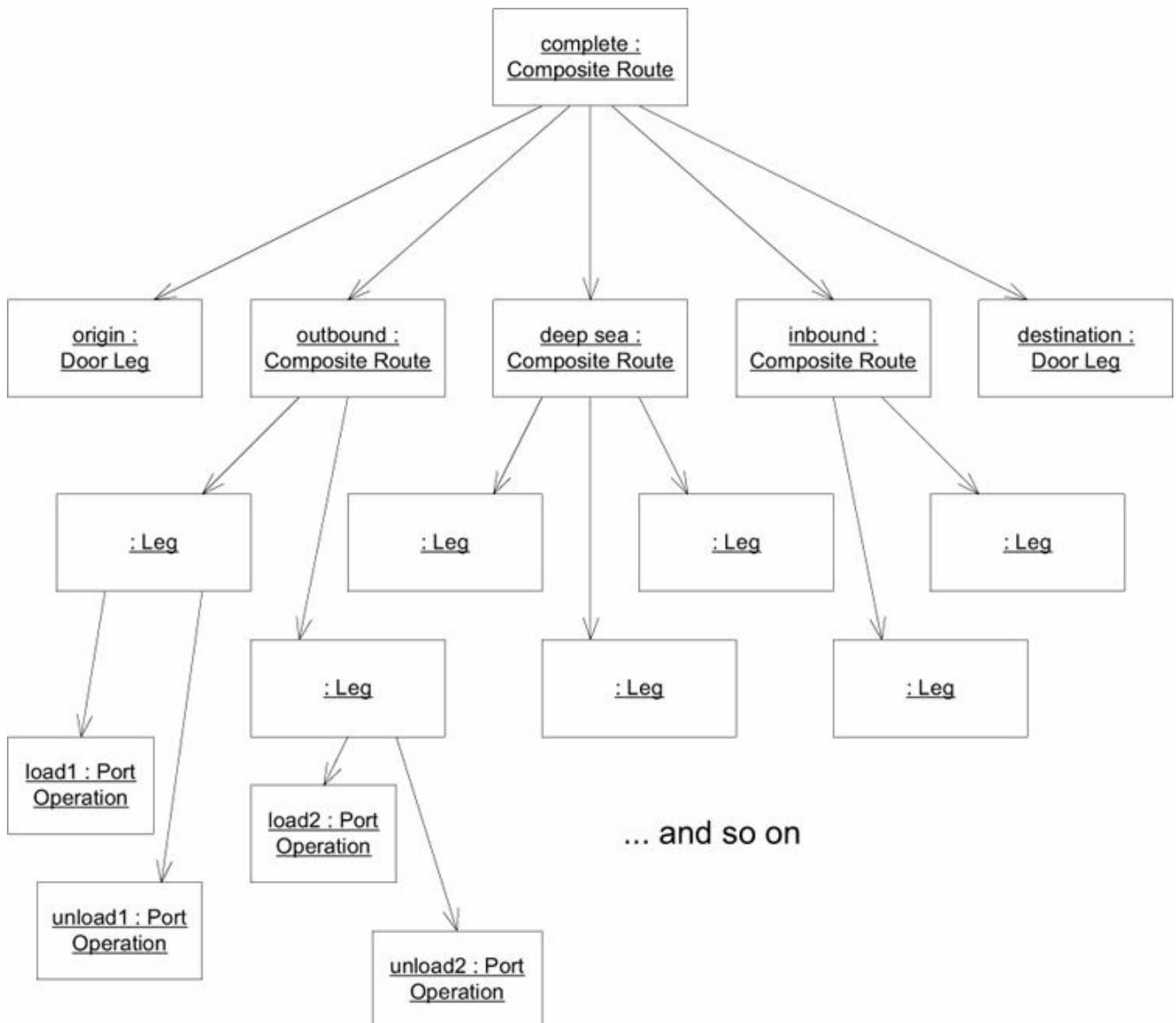


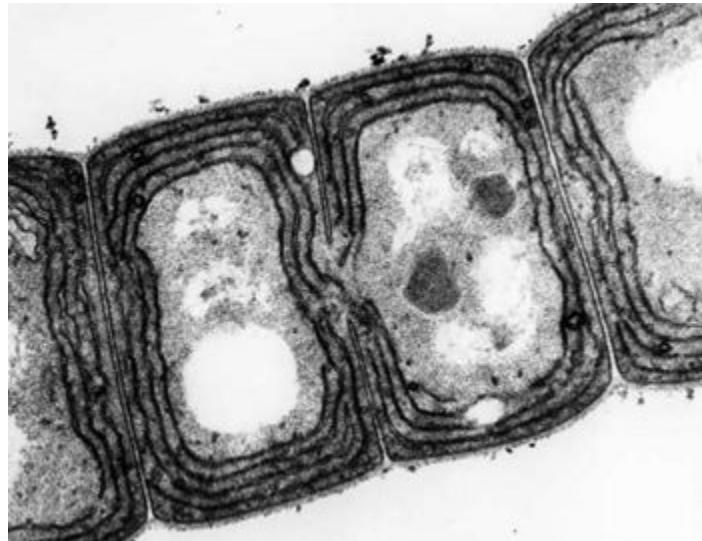
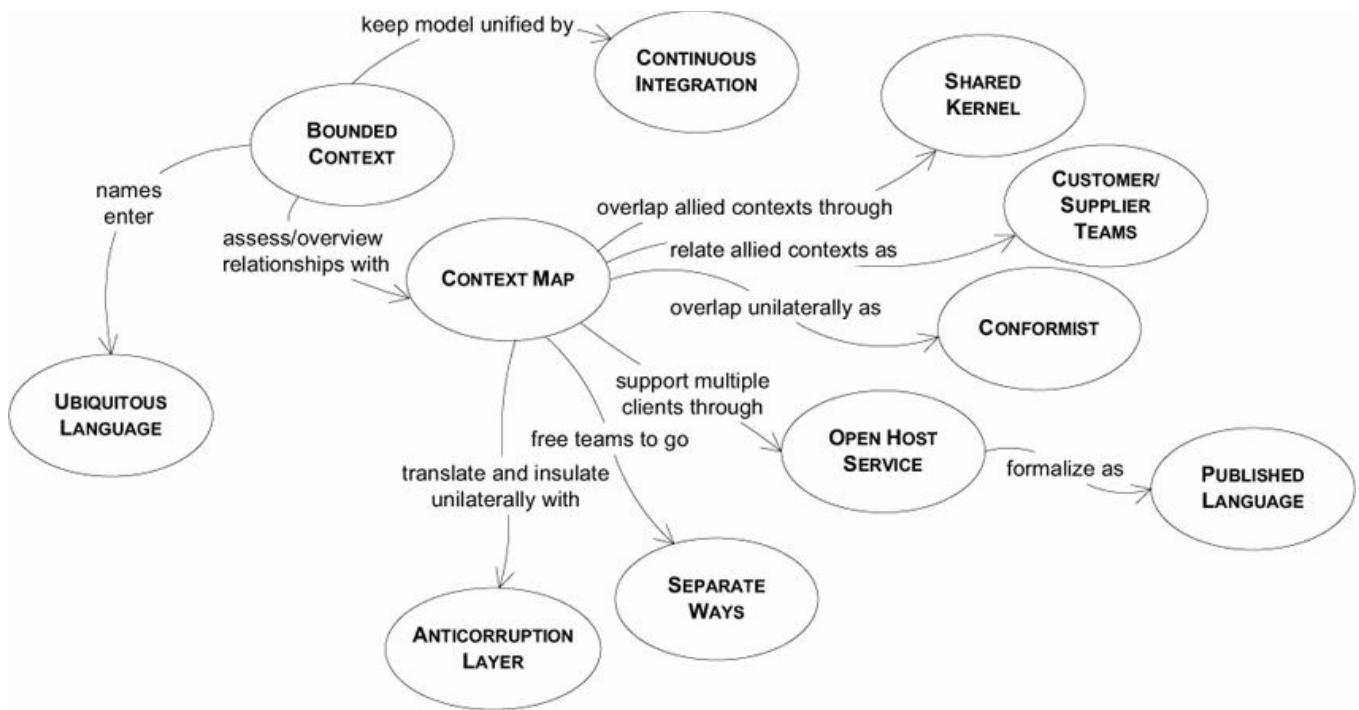










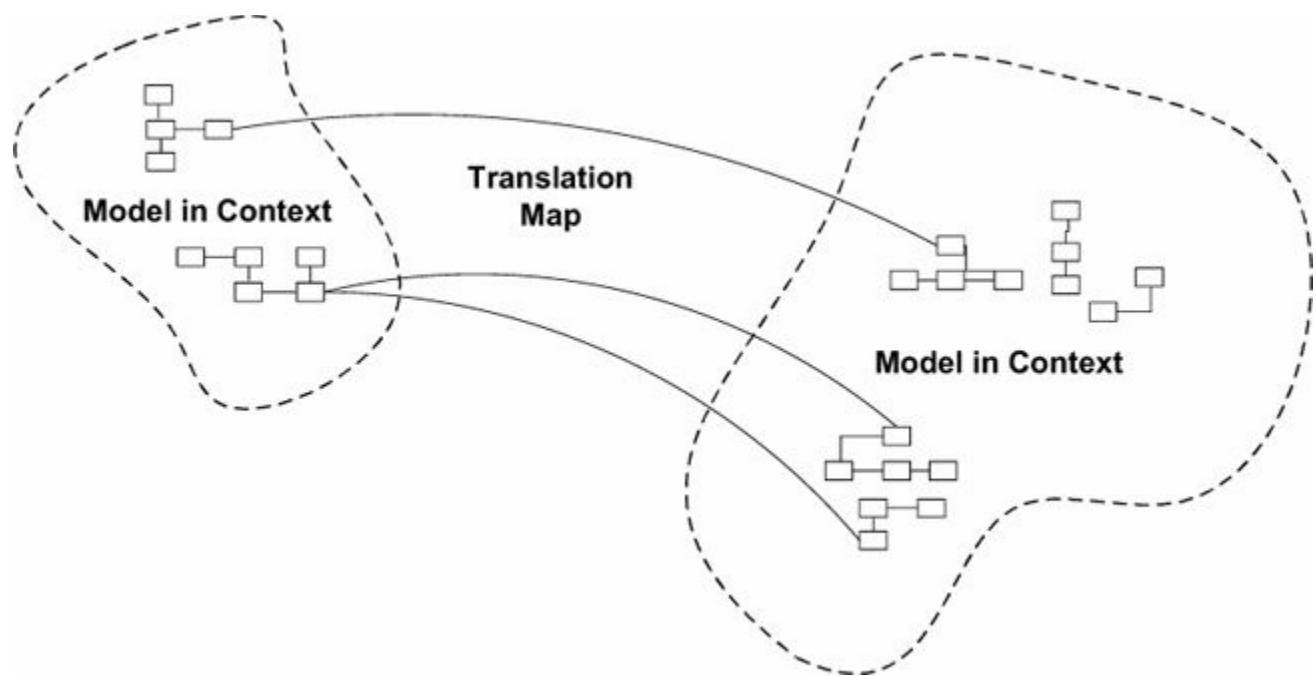


* * *



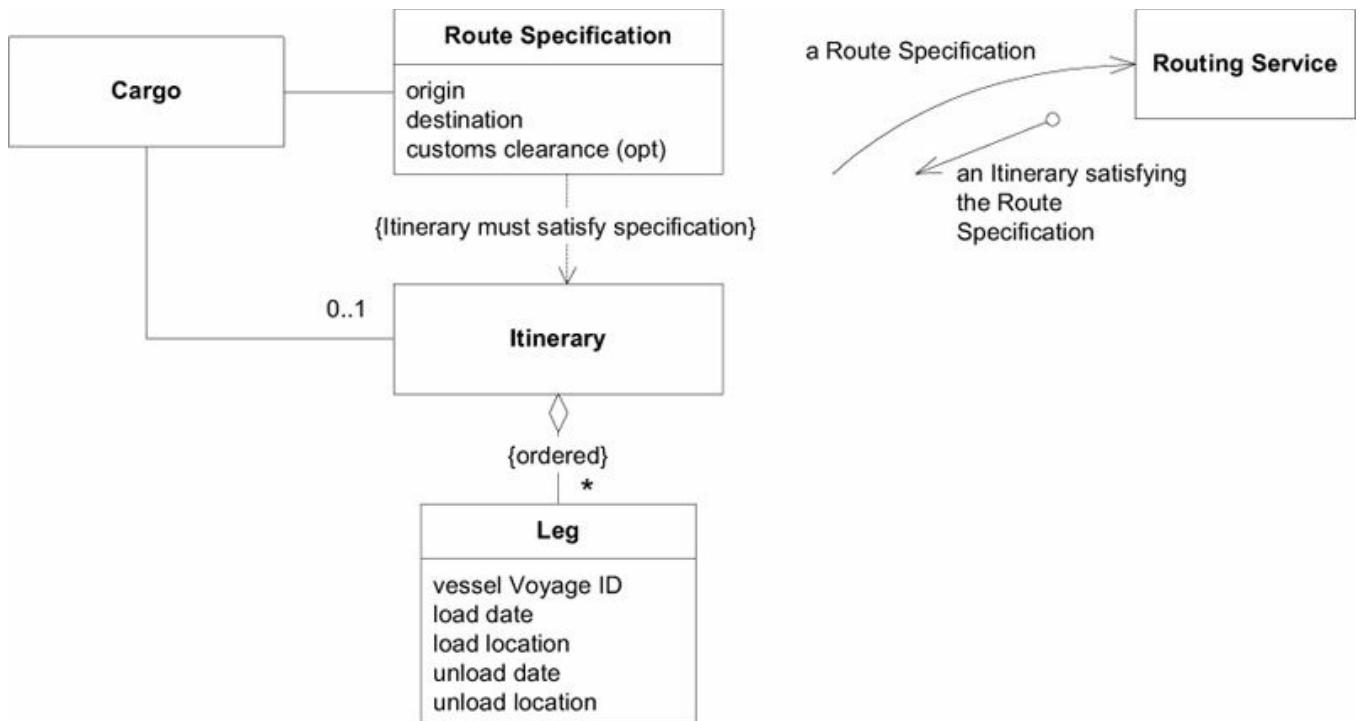
* * *

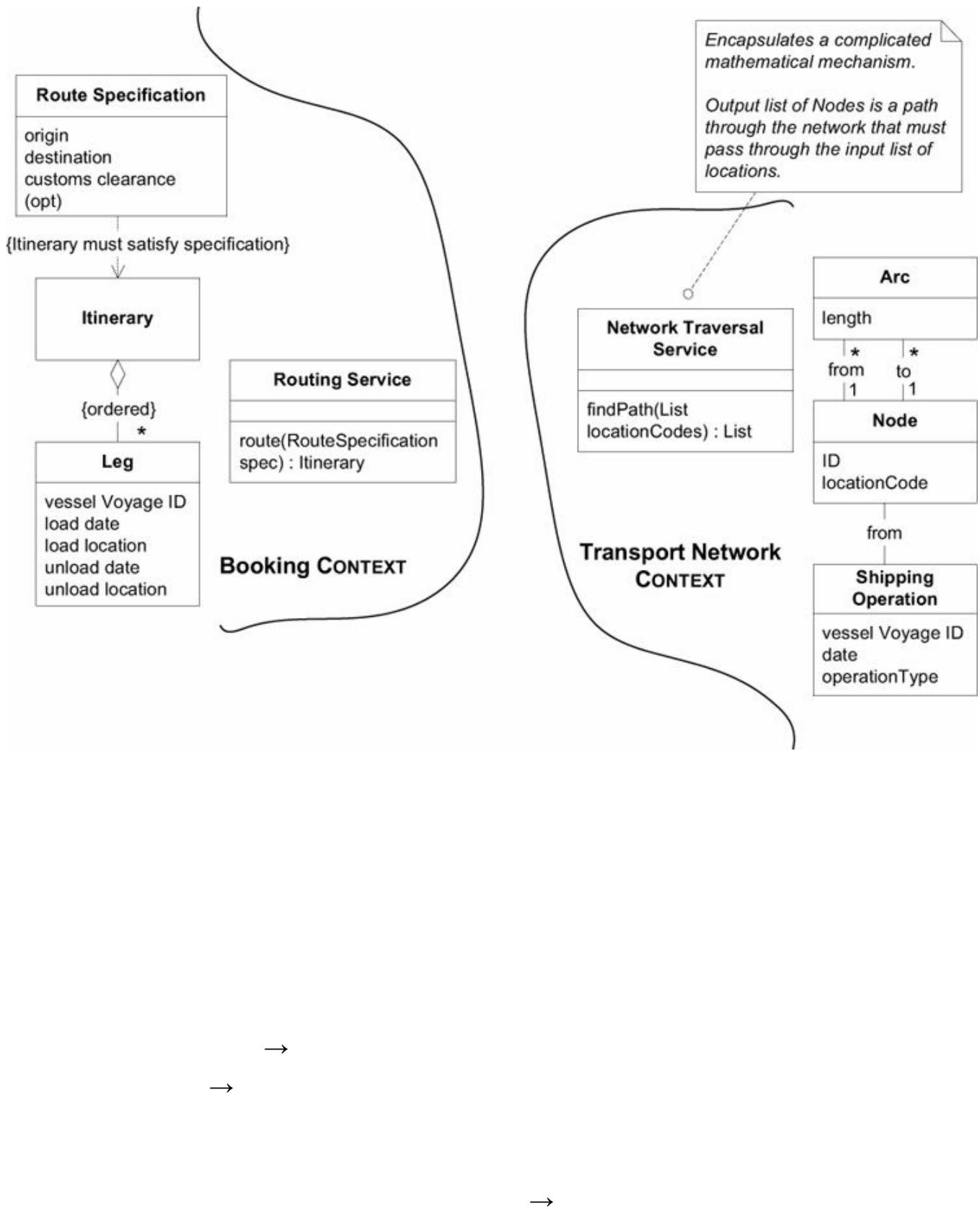
* * *

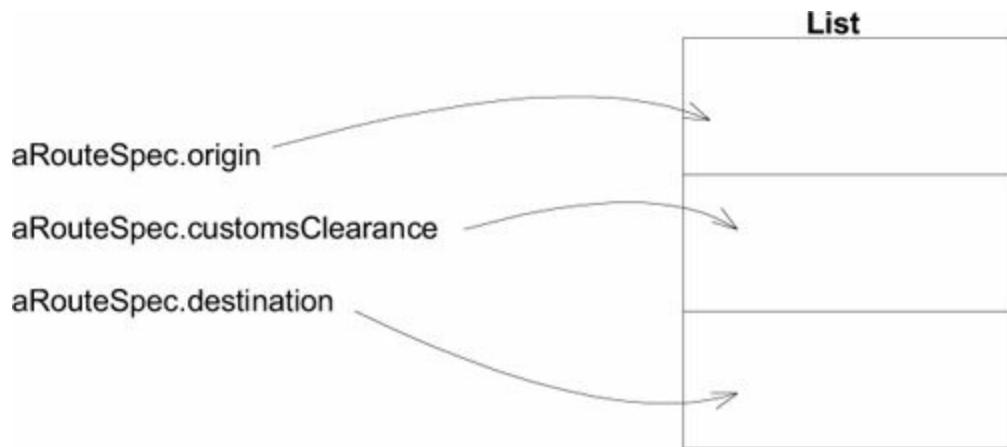


* * *

* * *

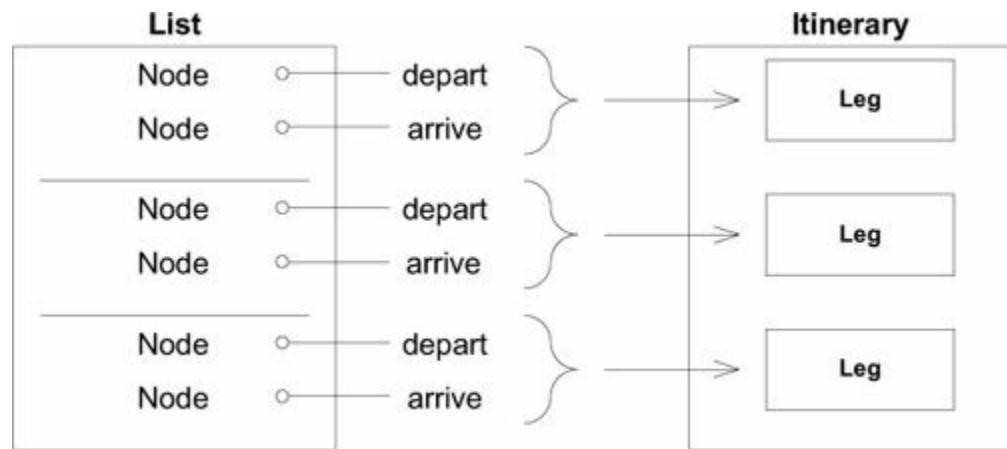






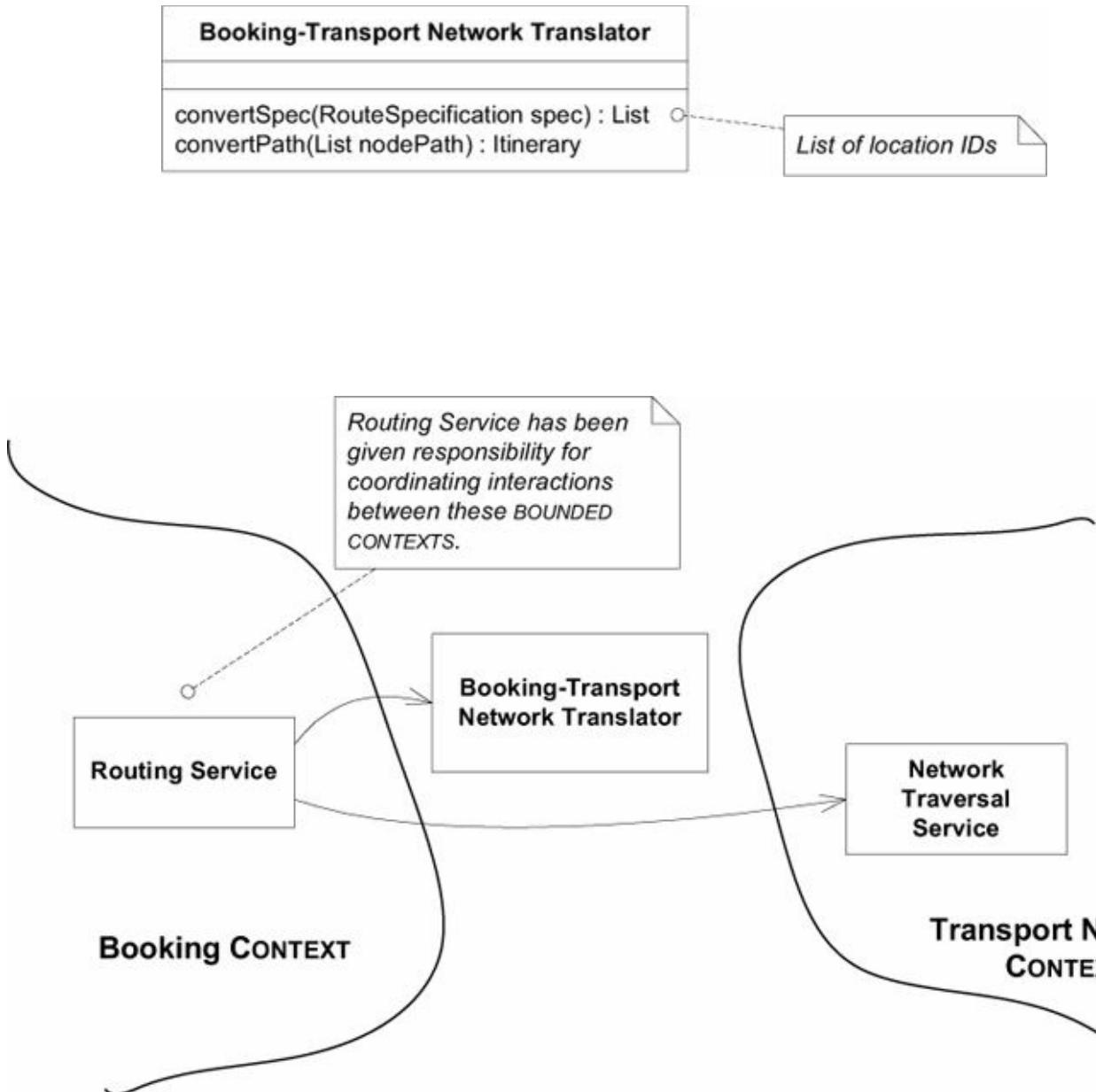
→

`operationType-Code`



```

departureNode.shippingOperation.vesselVoyageId → leg.vesselVoyageId
departureNode.shippingOperation.date → leg.loadDate
departureNode.locationCode → leg.loadLocationCode
arrivalNode.shippingOperation.date → leg.unloadDate
arrivalNode.locationCode → leg.unloadLocationCode
  
```



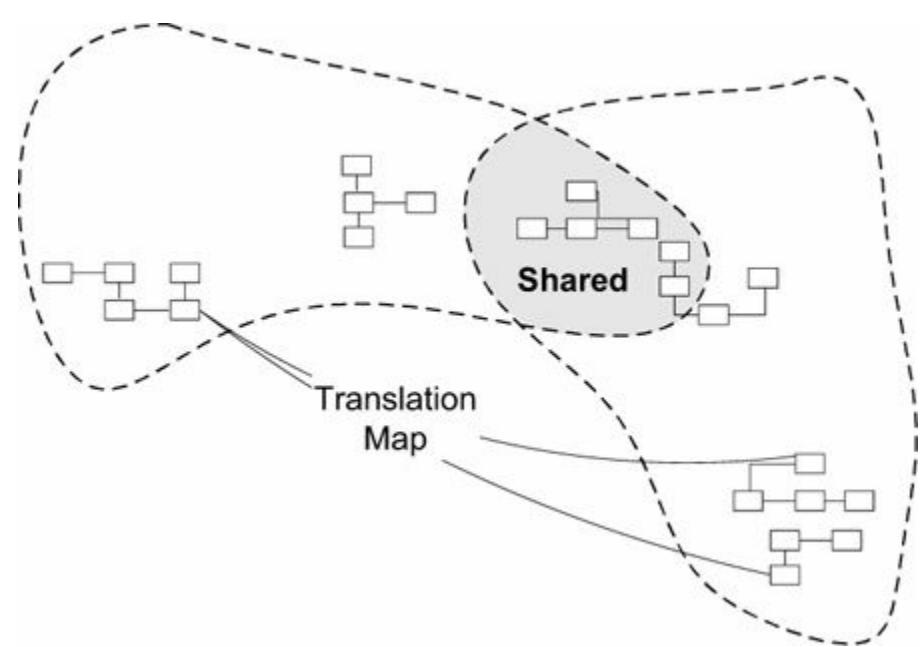
```

public Itinerary route( RouteSpecification spec ) {
    Booking_TransportNetwork_Translator translator =
        new Booking_TransportNetwork_Translator();

    List constraintLocations =
        translator.convertConstraints( spec );

    // Get access to the NetworkTraversalService
    List pathNodes =
        traversalService.findPath( constraintLocations );

    Itinerary result = translator.convert( pathNodes );
    return result;
}
  
```

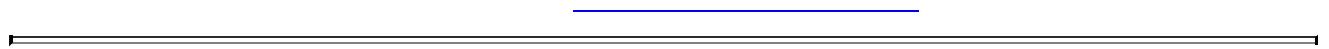
* * *



* * *

* * *



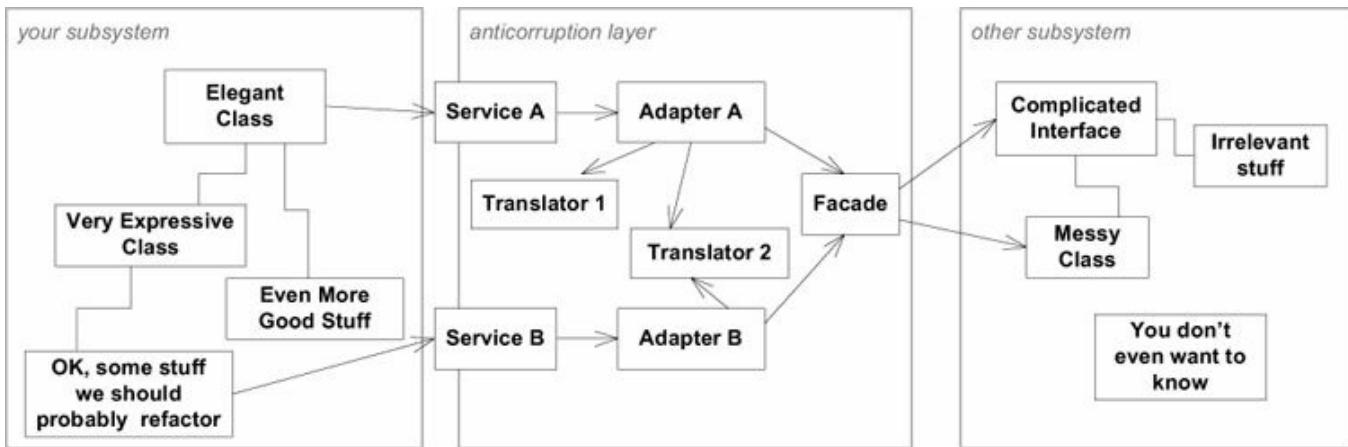


* * *



* * *

* * *





* * *

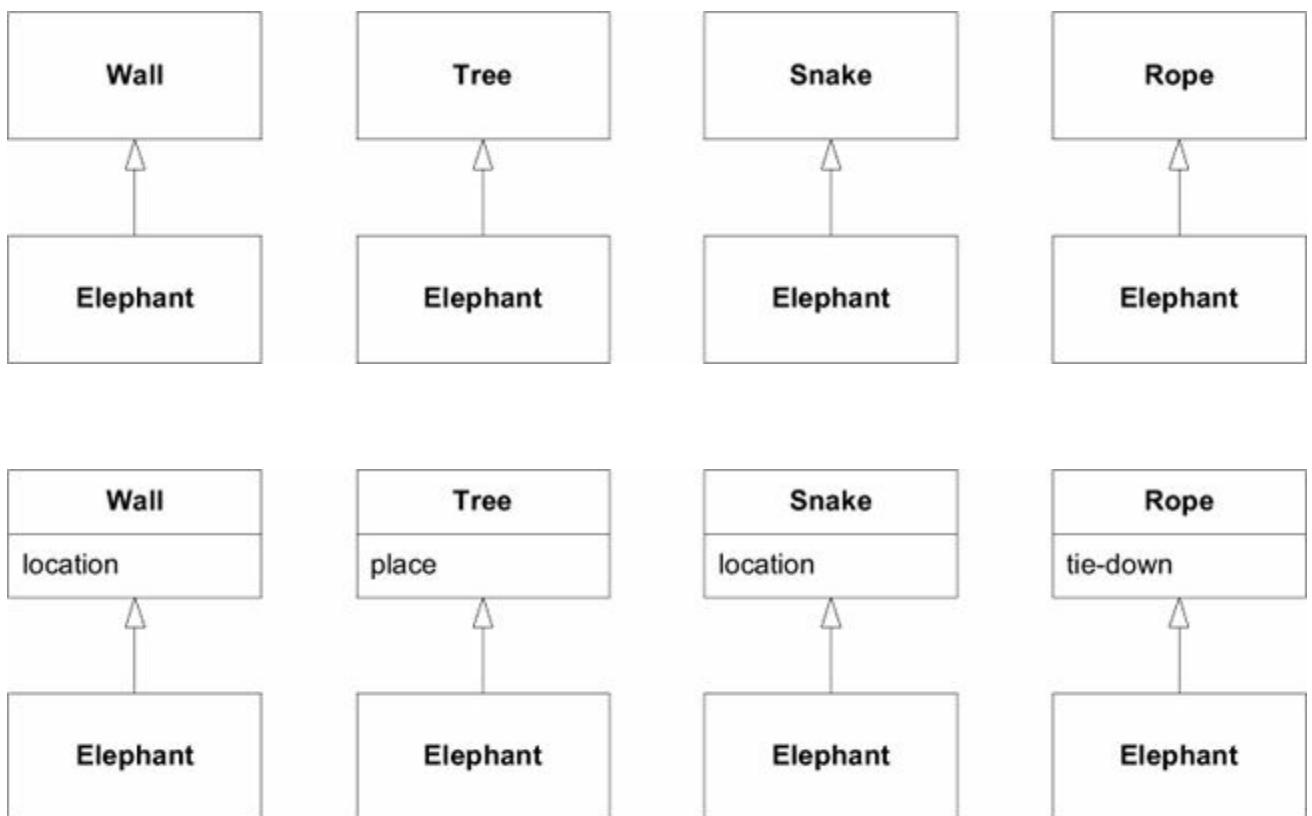
* * *

* * *

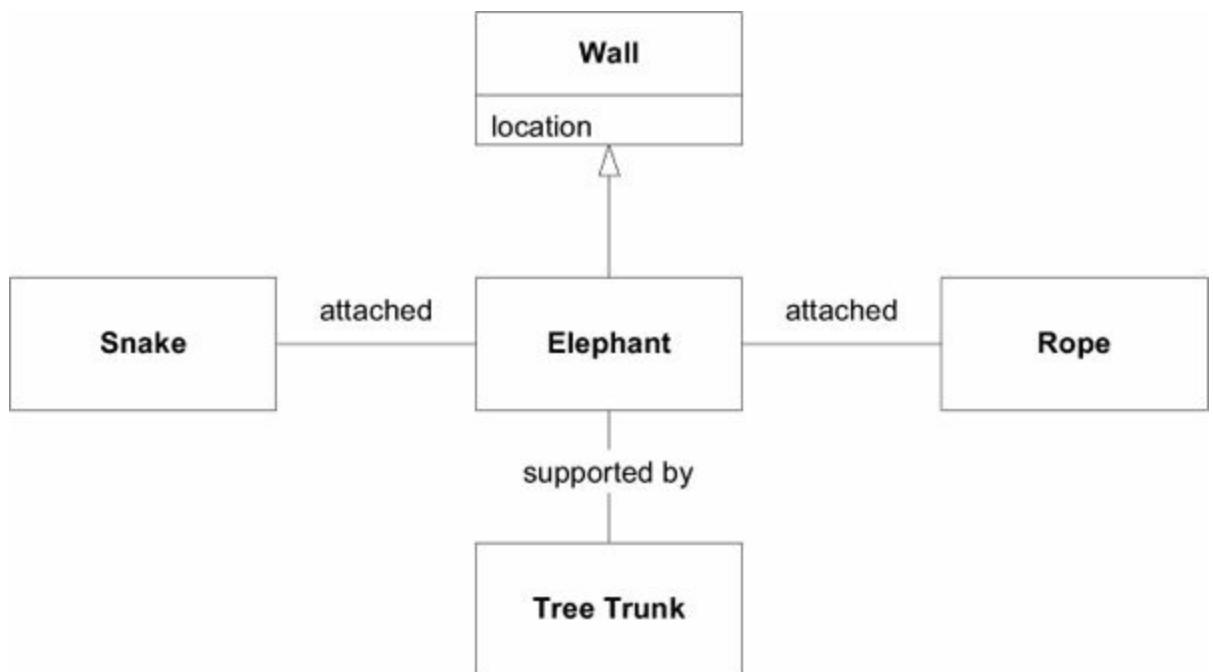
* * *

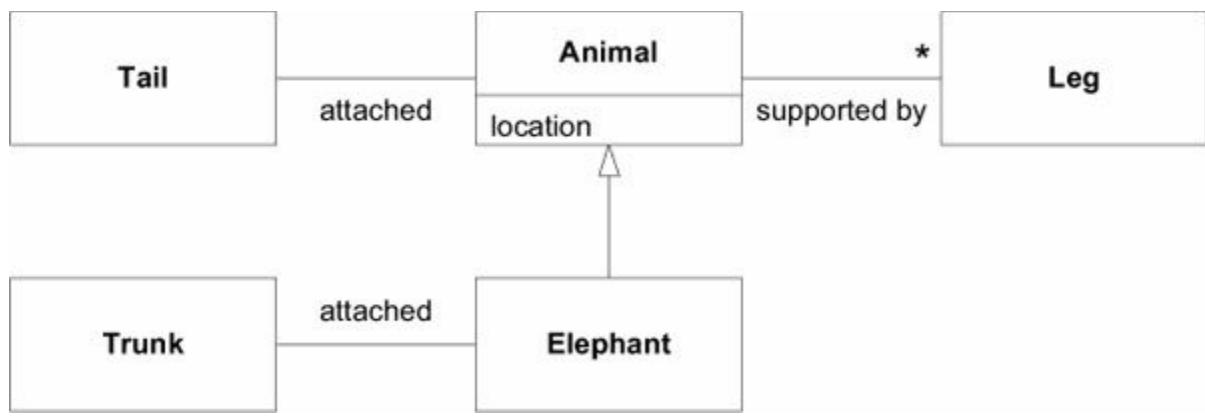
* * *

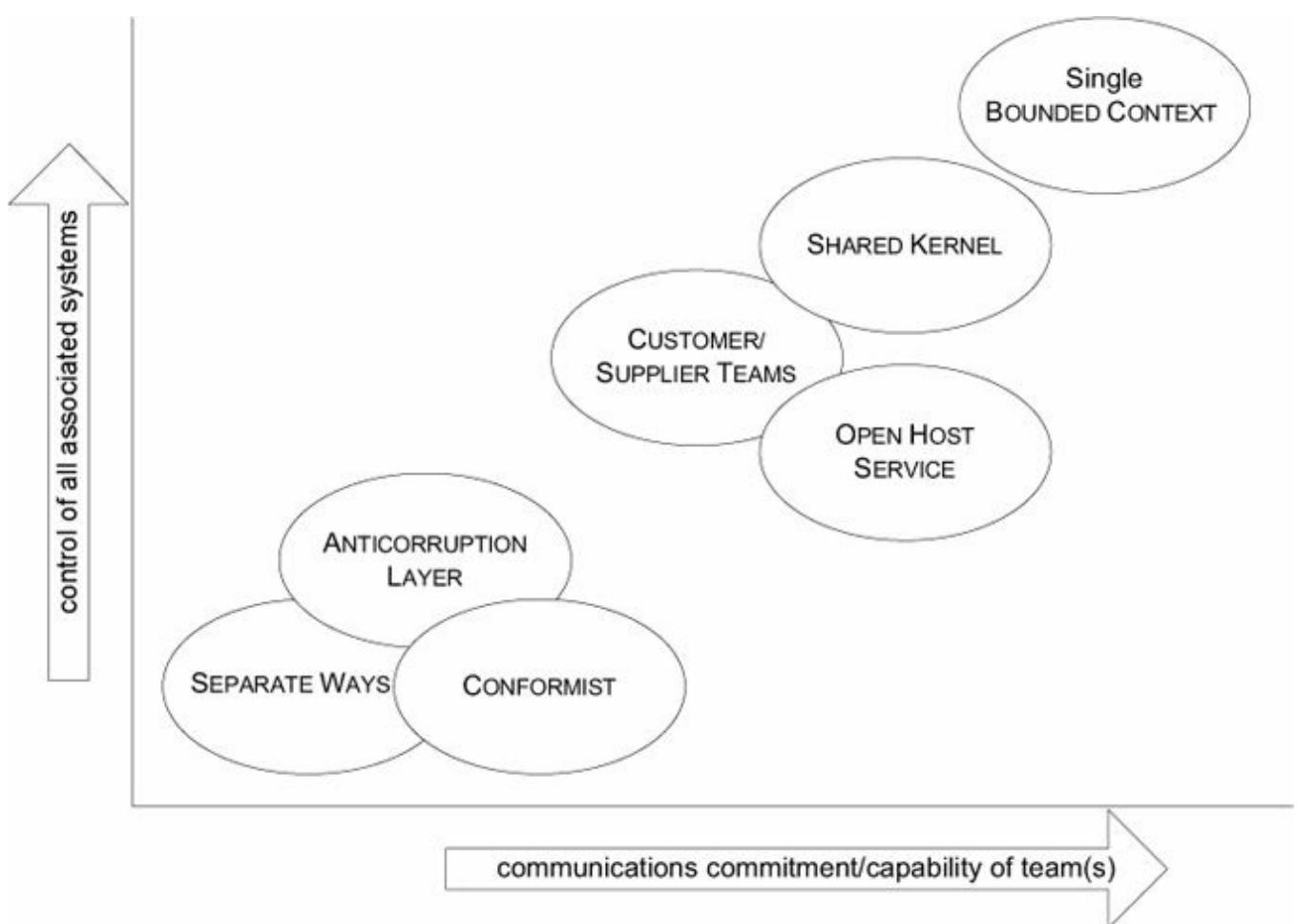
```
<CML.ARR ID="array3" EL.TYP=FLOAT NAME="ATOMIC ORBITAL ELECTRON  
POPULATIONS" SIZE=30 GLO.ENT=CML.THE.AOEPOPS>  
  1.17947  0.95091  0.97175  1.00000  1.17947  0.95090  0.97174  1.  
  1.17946  0.98215  0.94049  1.00000  1.17946  0.95091  0.97174  1.  
  1.17946  0.95091  0.97174  1.00000  1.17946  0.98215  0.94049  1.  
  0.89789  0.89790  0.89789  0.89789  0.89790  0.89788  
</CML.ARR>
```



Translations: {Wall.location \leftrightarrow Tree.place \leftrightarrow Snake.location \leftrightarrow Rope.tie-down}







→

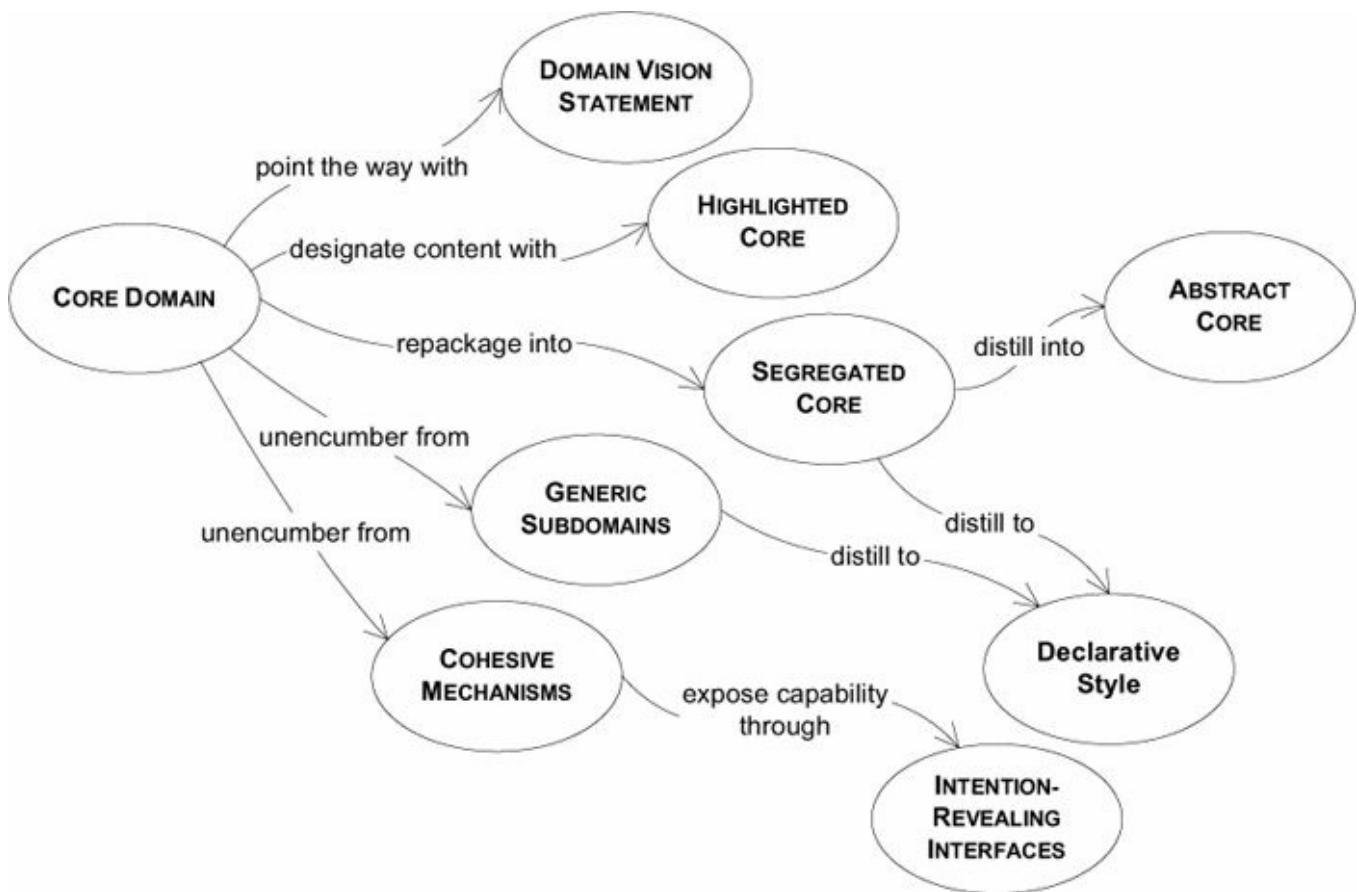
→

$$\nabla \bullet \mathbf{D} = \rho$$

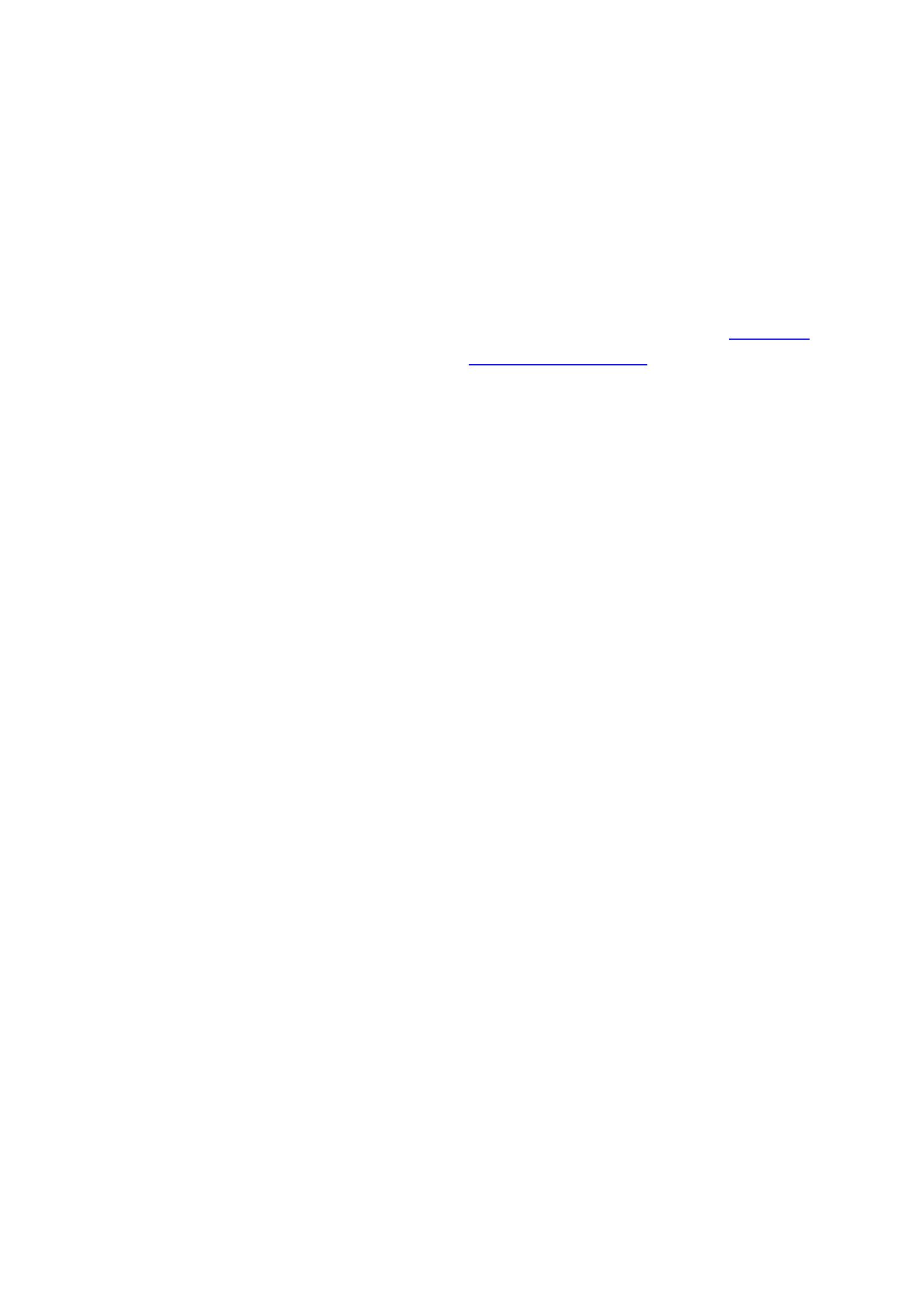
$$\nabla \bullet \mathbf{B} = 0$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t}$$



* * *



* * *

Shipping Project's Strategy

Advantages

- GENERIC model decoupled from CORE.
- CORE model mature, so resources could be diverted without stunting it.
- Knew exactly what they needed.
- Critical support functionality for international scheduling.
- Programmer on short-term contract used for GENERIC task.

Disadvantage

- Diverted top programmer from core.

Insurance Project's Strategy

Advantage

- GENERIC model decoupled from CORE.

Disadvantages

- CORE model undeveloped, so attention to other issues continued this neglect.
- Unknown requirements led to attempt at full generality, where simpler North America-specific conversion might have sufficed.
- Long-term programmers were assigned who could have been repositories of domain knowledge.

**This is part of a DOMAIN
VISION STATEMENT*****Airline Booking System***

The model can represent passenger priorities and airline booking strategies and balance these based on flexible policies. The model of a passenger should reflect the “relationship” the airline is striving to develop with repeat customers. Therefore, it should represent the history of the passenger in useful condensed form, participation in special programs, affiliation with strategic corporate clients, and so on.

Different roles of different users (such as passenger, agent, manager) are represented to enrich the model of relationships and to feed necessary information to the security framework.

Model should support efficient route/seat search and integration with other established flight booking systems.

**This is part of a DOMAIN
VISION STATEMENT*****Semiconductor Factory Automation***

The domain model will represent the status of materials and equipment within a wafer fab in such a way that necessary audit trails can be provided and automated product routing can be supported.

The model will not include the human resources required in the process, but must allow selective process automation through recipe download.

The representation of the state of the factory should be comprehensible to human managers, to give them deeper insight and support better decision making.

**This, though important, is *not* part of a
DOMAIN VISION STATEMENT*****Airline Booking System***

The UI should be streamlined for expert users but accessible to first-time users.

Access will be offered over the Web, by data transfer to other systems, and maybe through other UIs, so interface will be designed around XML with transformation layers to serve Web pages or translate to other systems.

A colorful animated version of the logo needs to be cached on the client machine so that it can come up quickly on future visits.

When customer submits a reservation, make visual confirmation within 5 seconds.

A security framework will authenticate a user's identity and then limit access to specific features based on privileges assigned to defined user roles.

**This, though important, is *not* part of a
DOMAIN VISION STATEMENT*****Semiconductor Factory Automation***

The software should be Web enabled through a servlet, but structured to allow alternative interfaces.

Industry-standard technologies should be used whenever possible to avoid in-house development and maintenance costs and to maximize access to outside expertise. Open source solutions are preferred (such as Apache Web server).

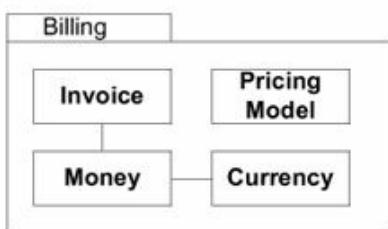
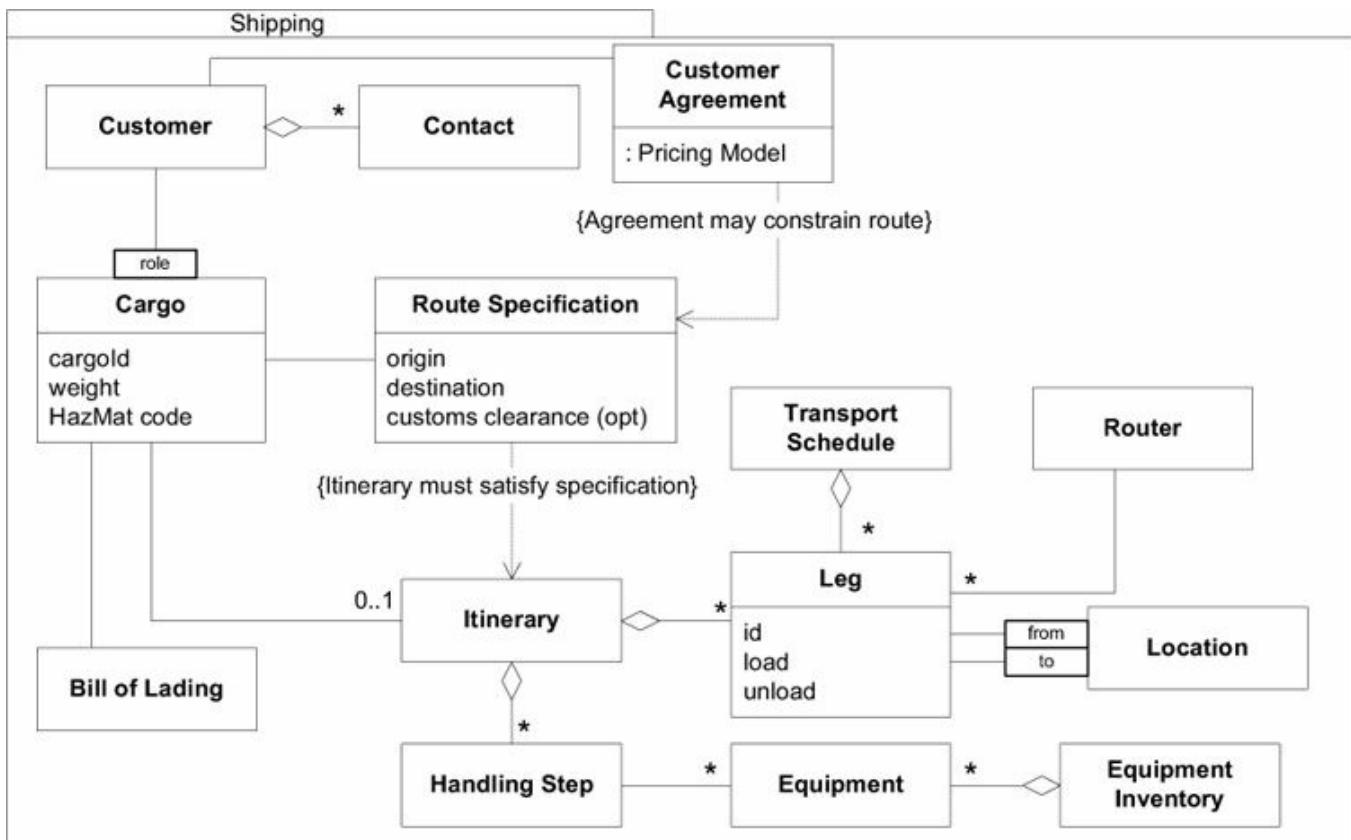
The Web server will run on a dedicated server. The application will run on a single dedicated server.

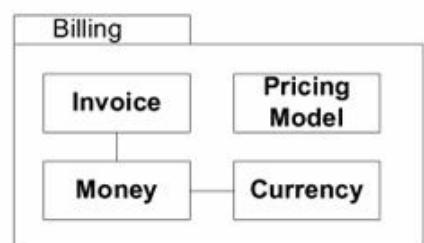
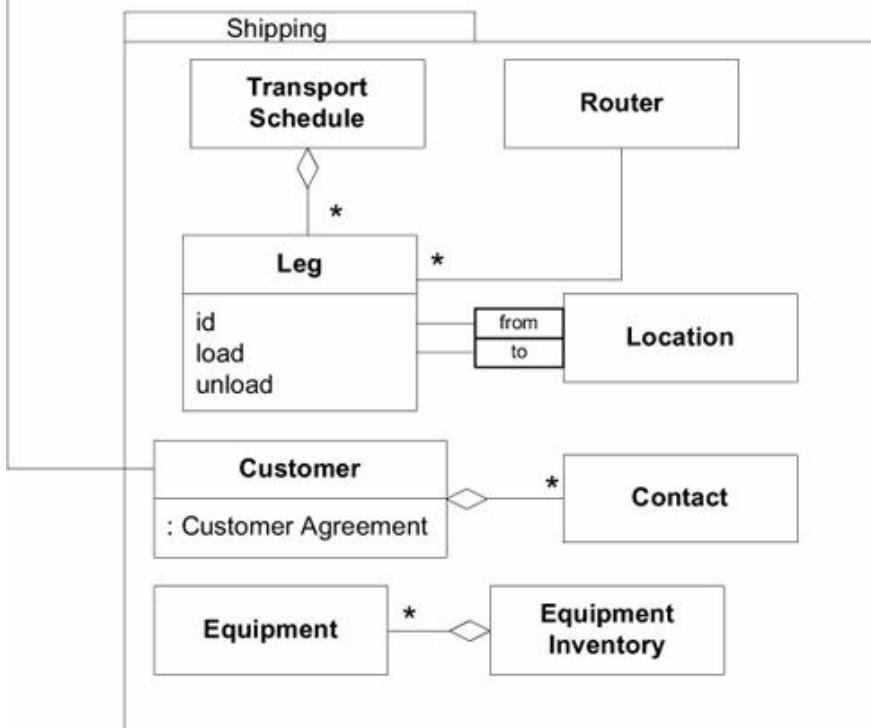
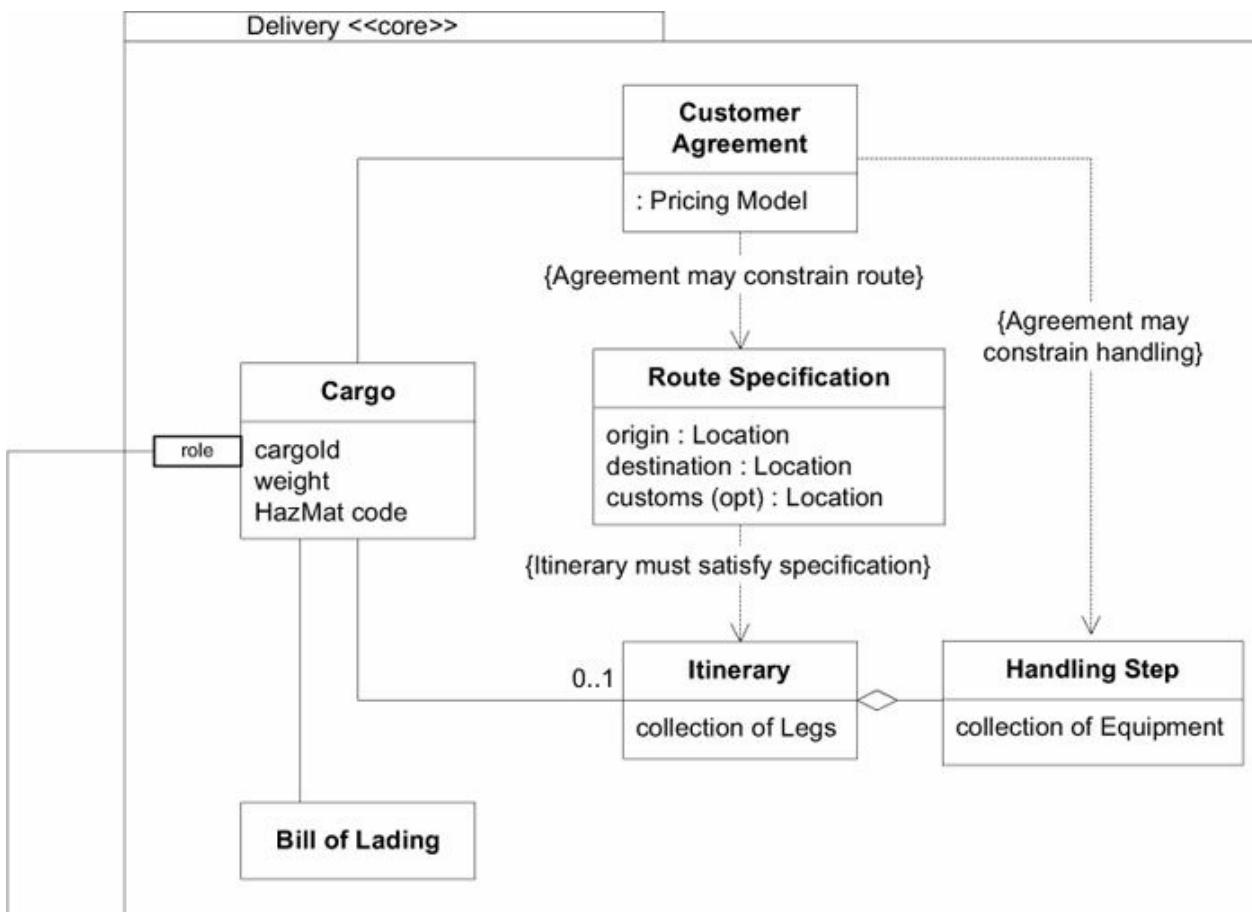
* * *

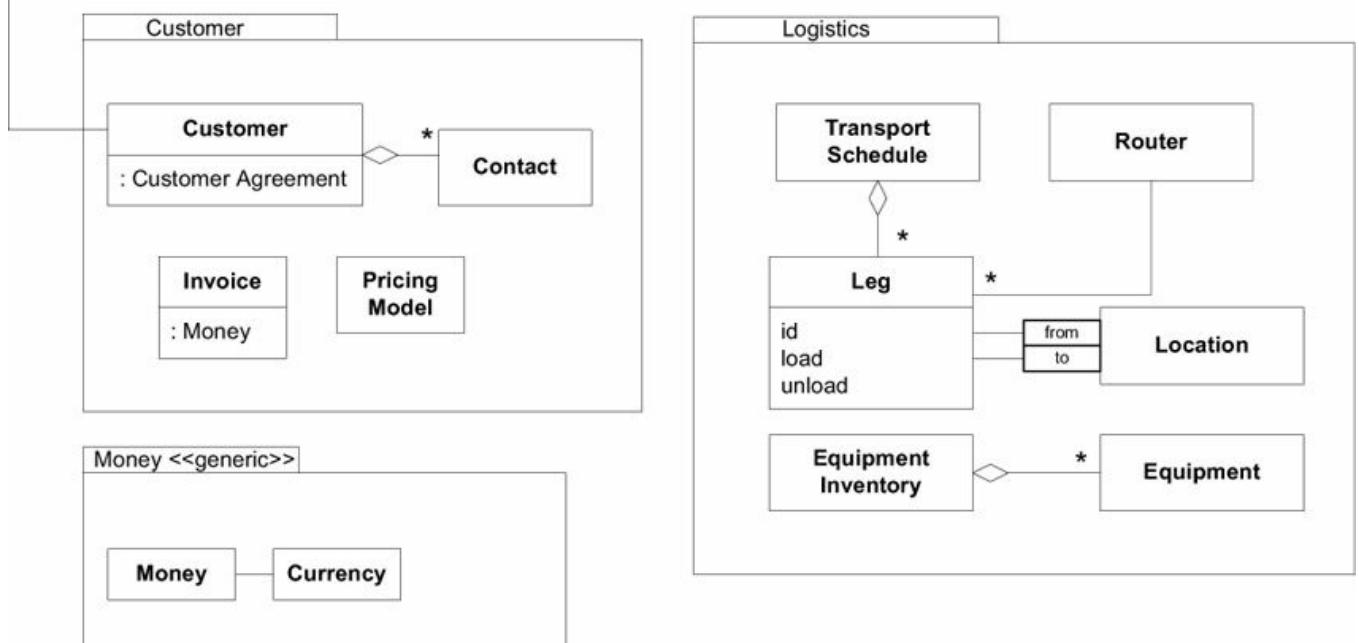
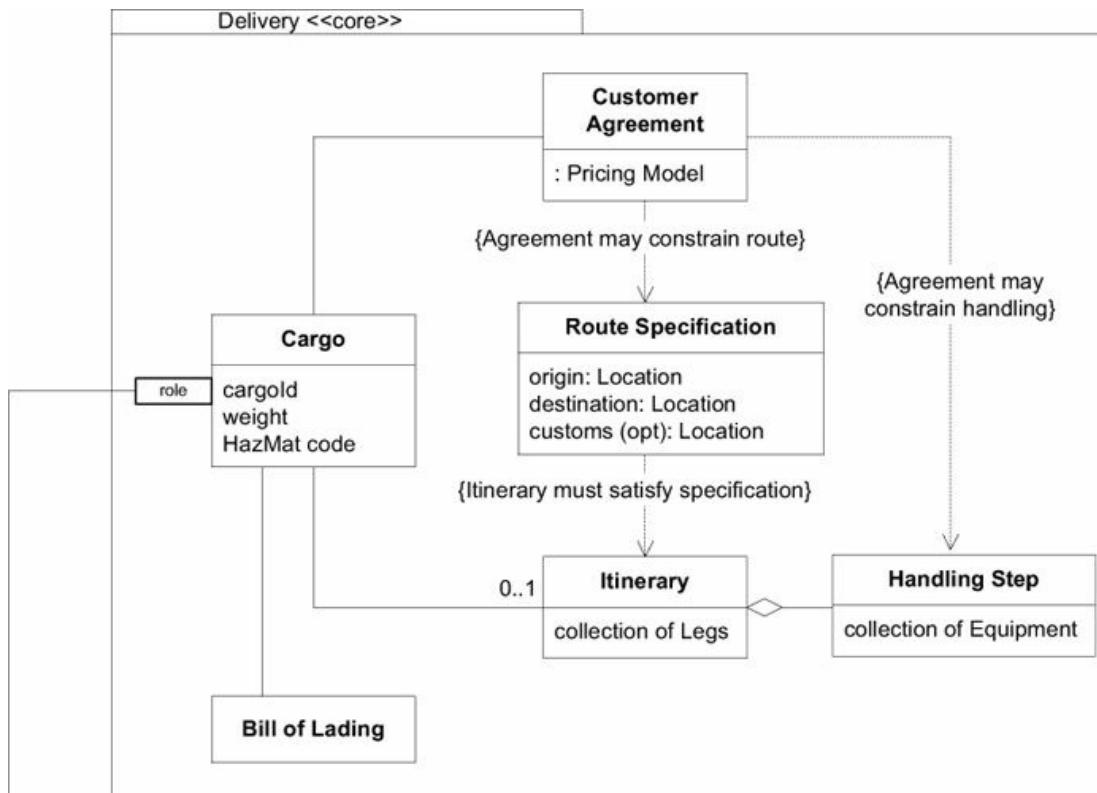
* * *

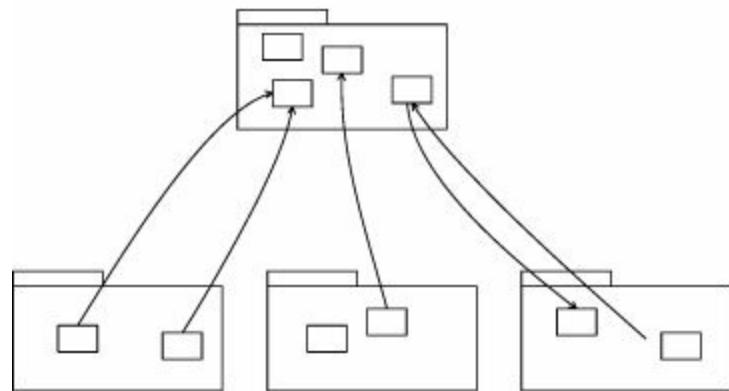
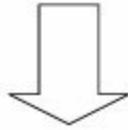
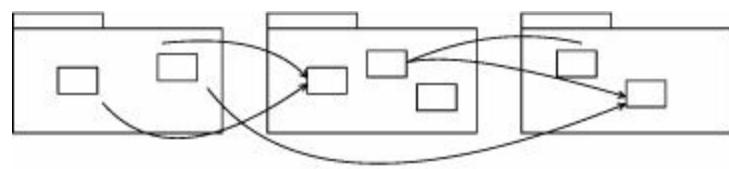


* * *





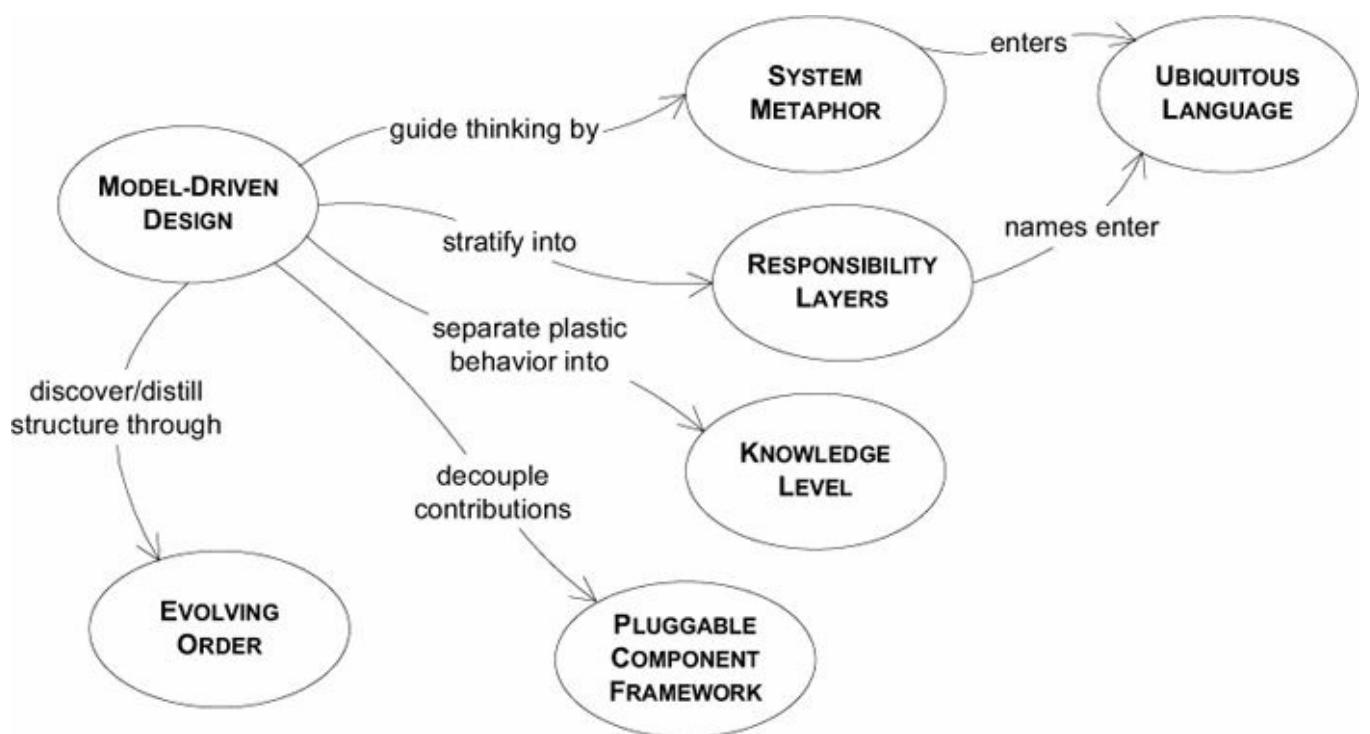




* * *

* * *





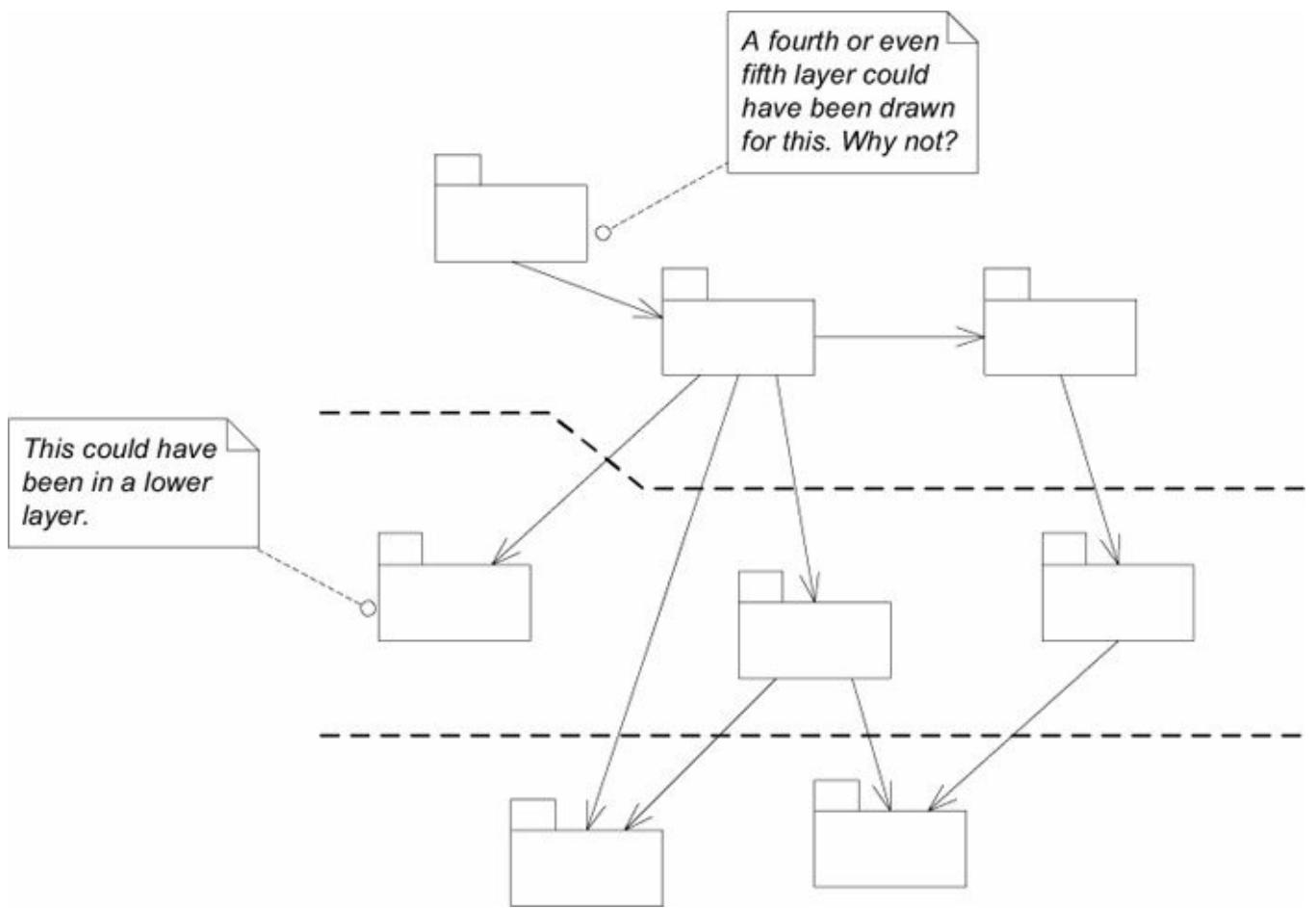
* * *

* * *

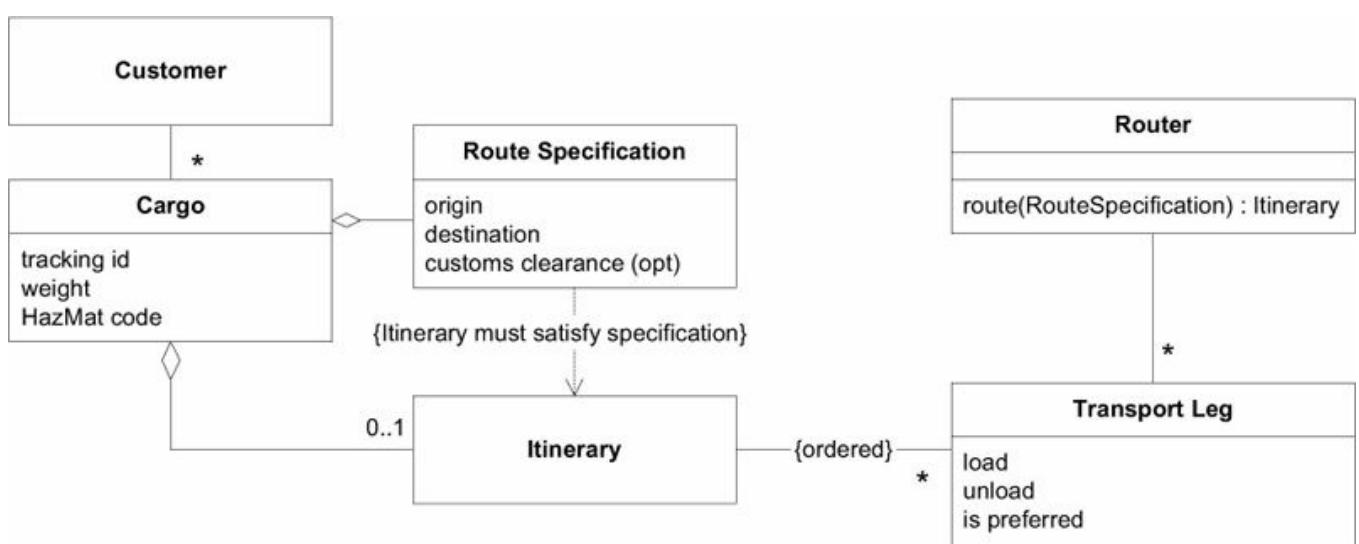
-

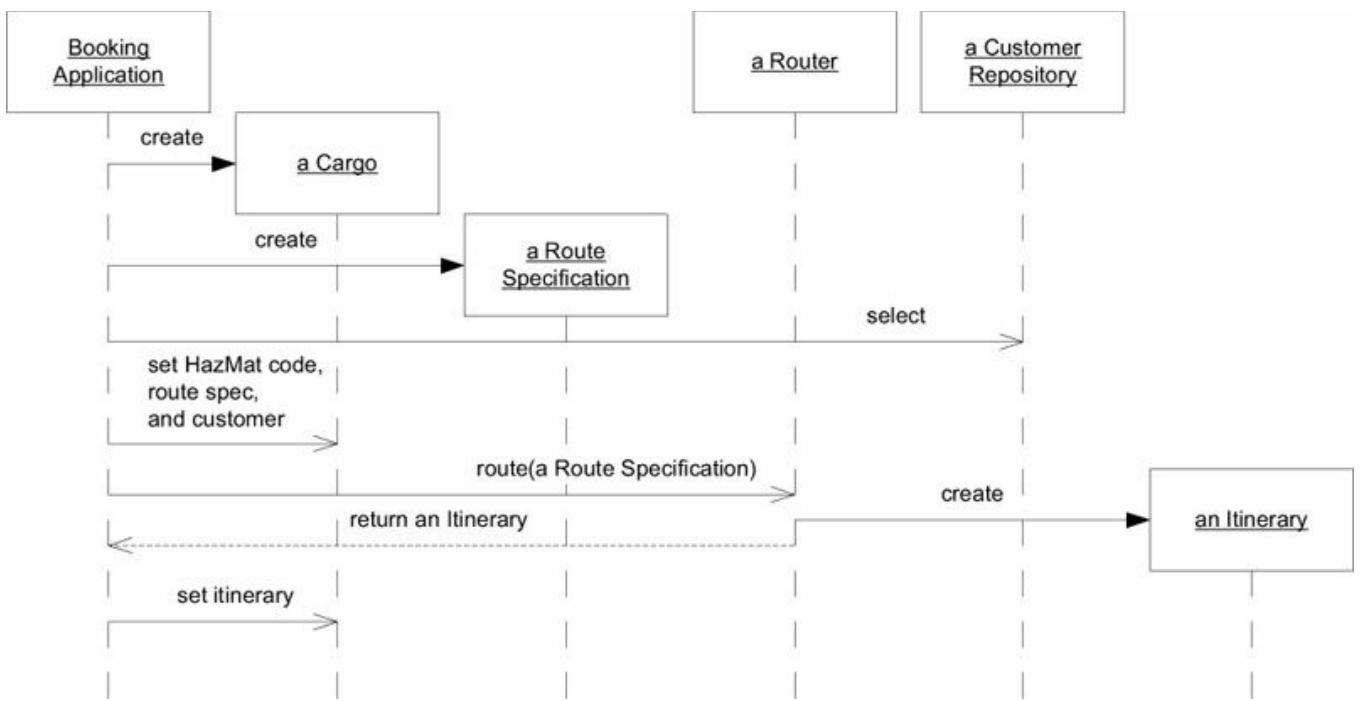
* * *

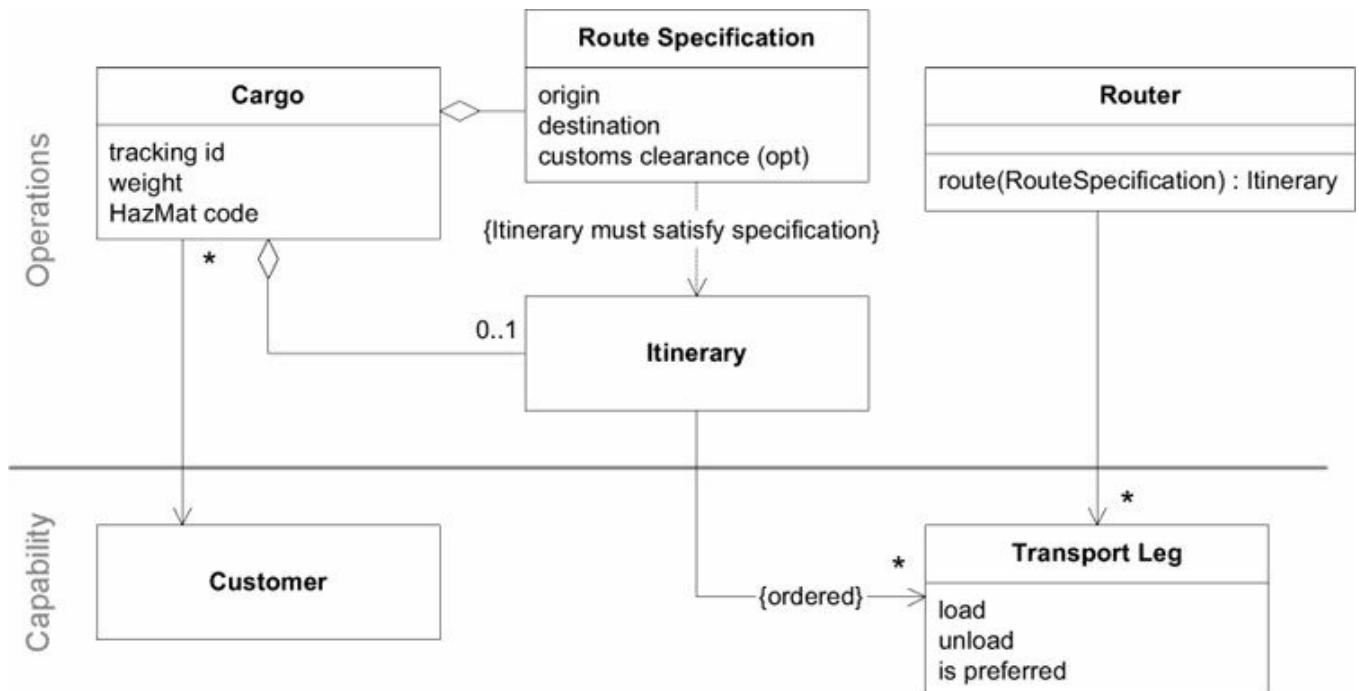
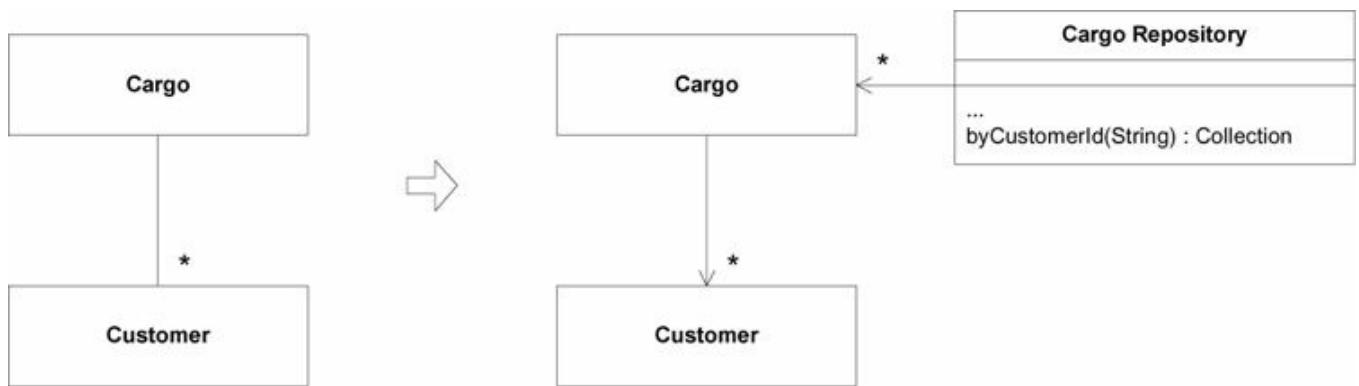
* * *

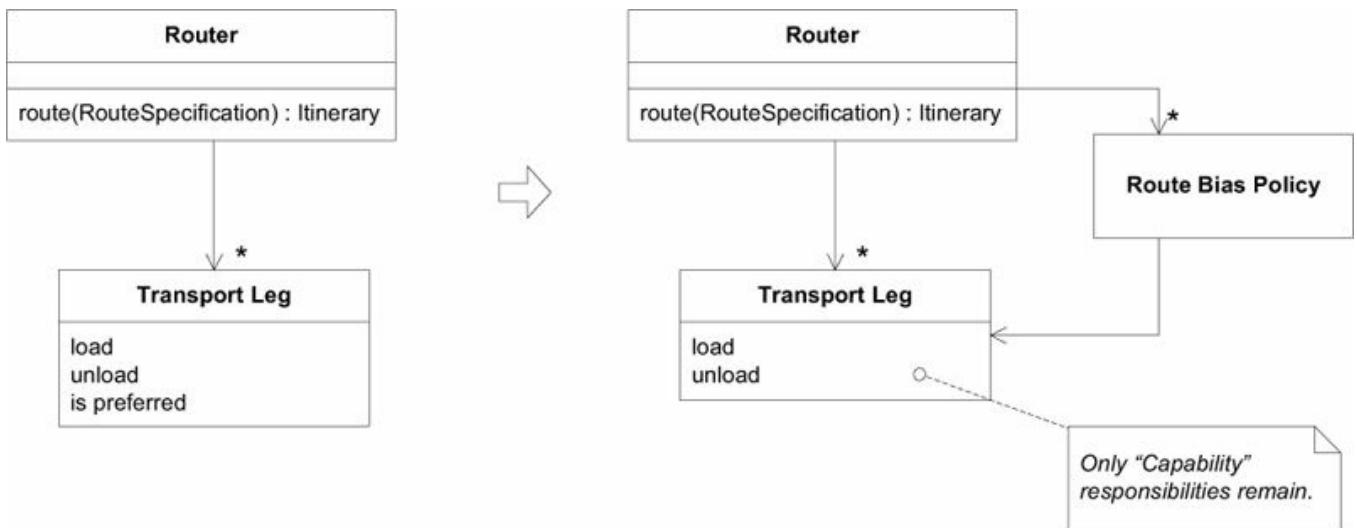


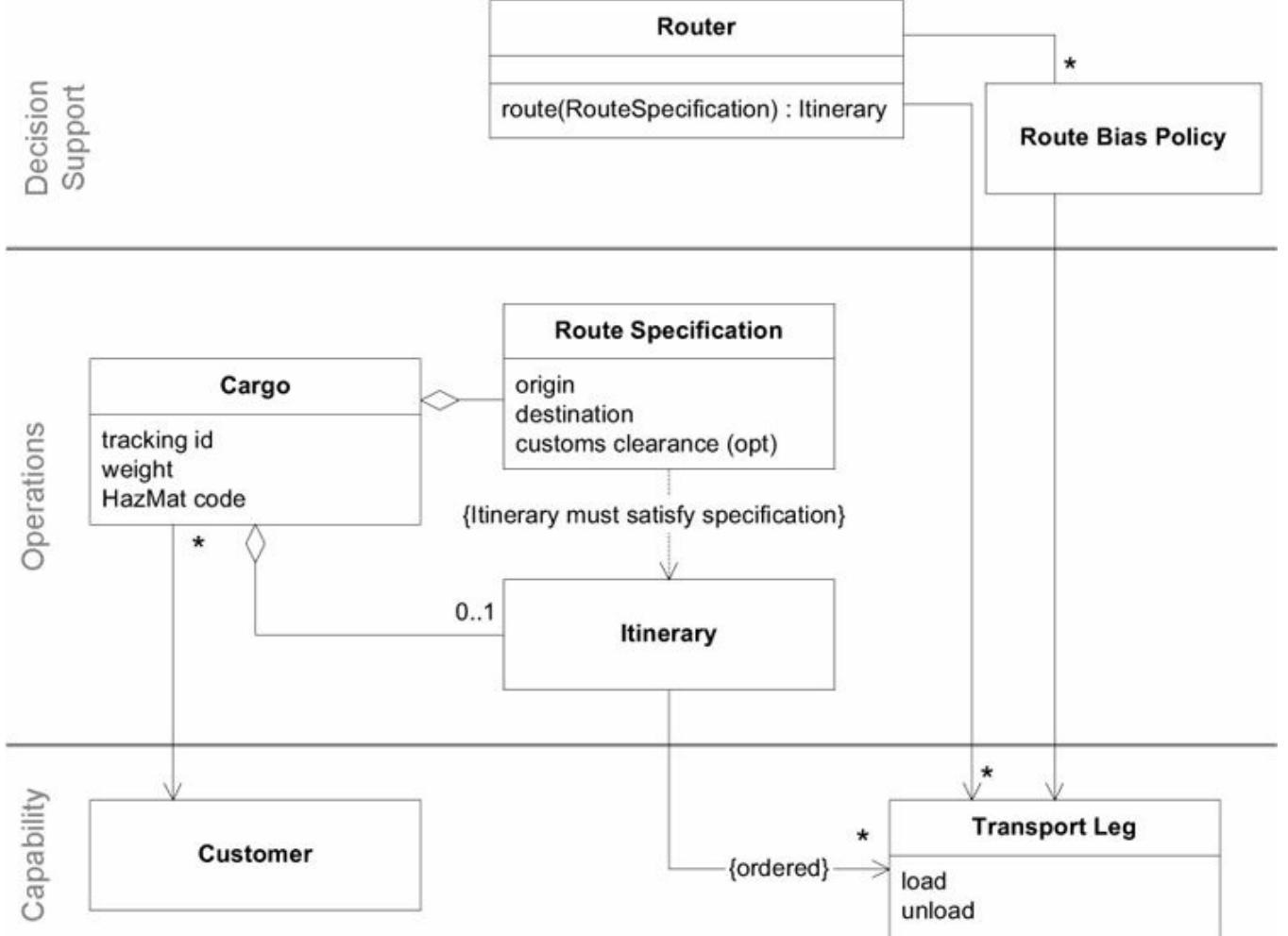
* * *

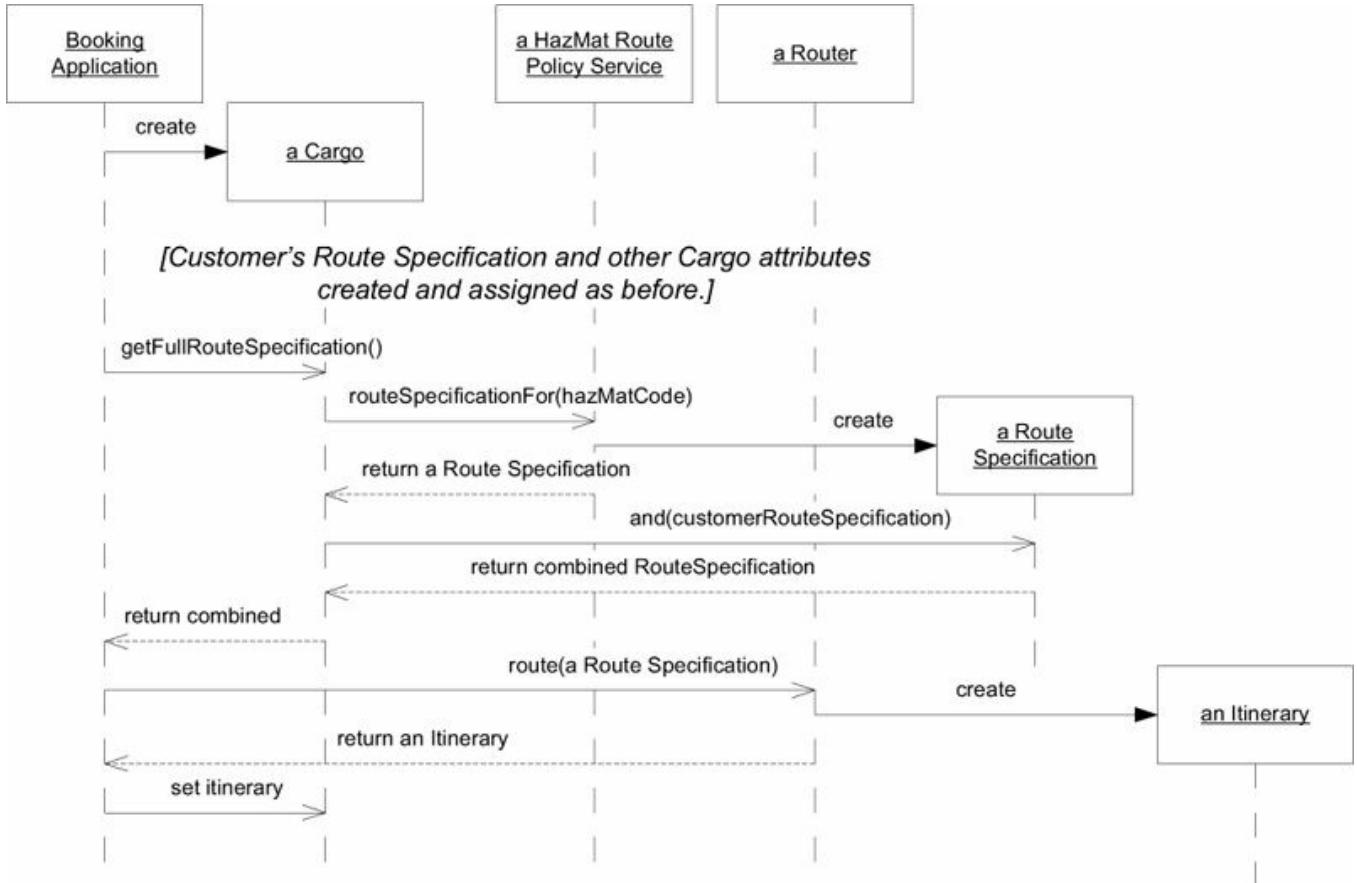
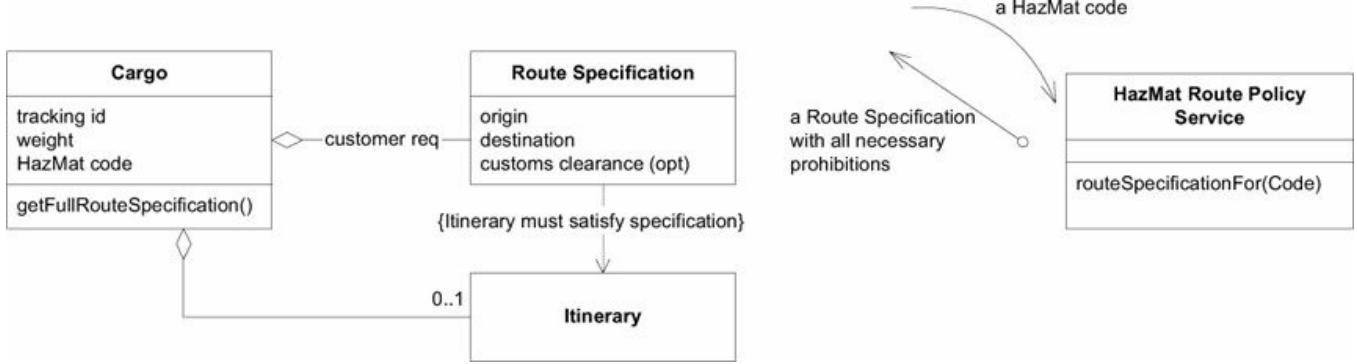


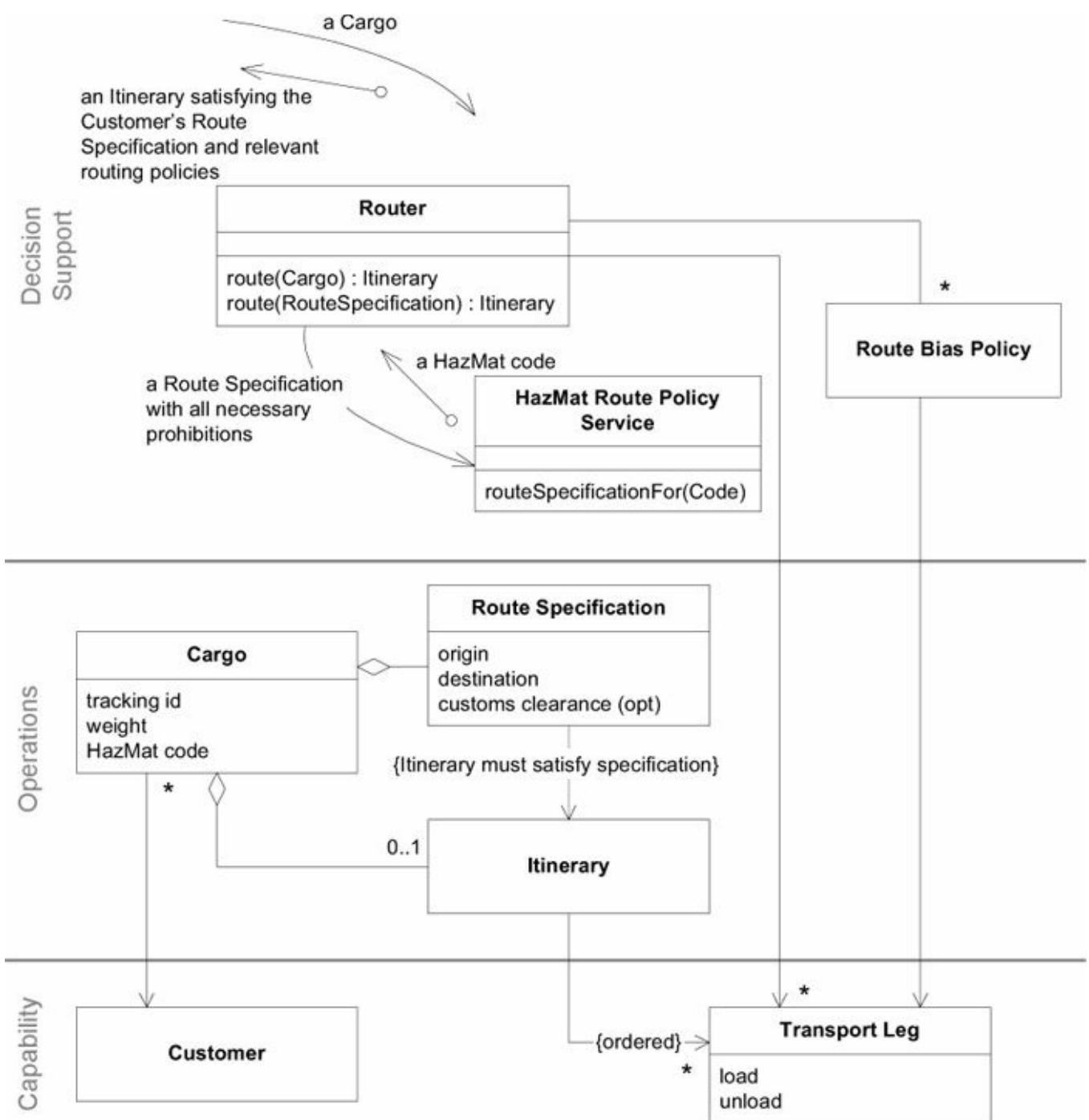


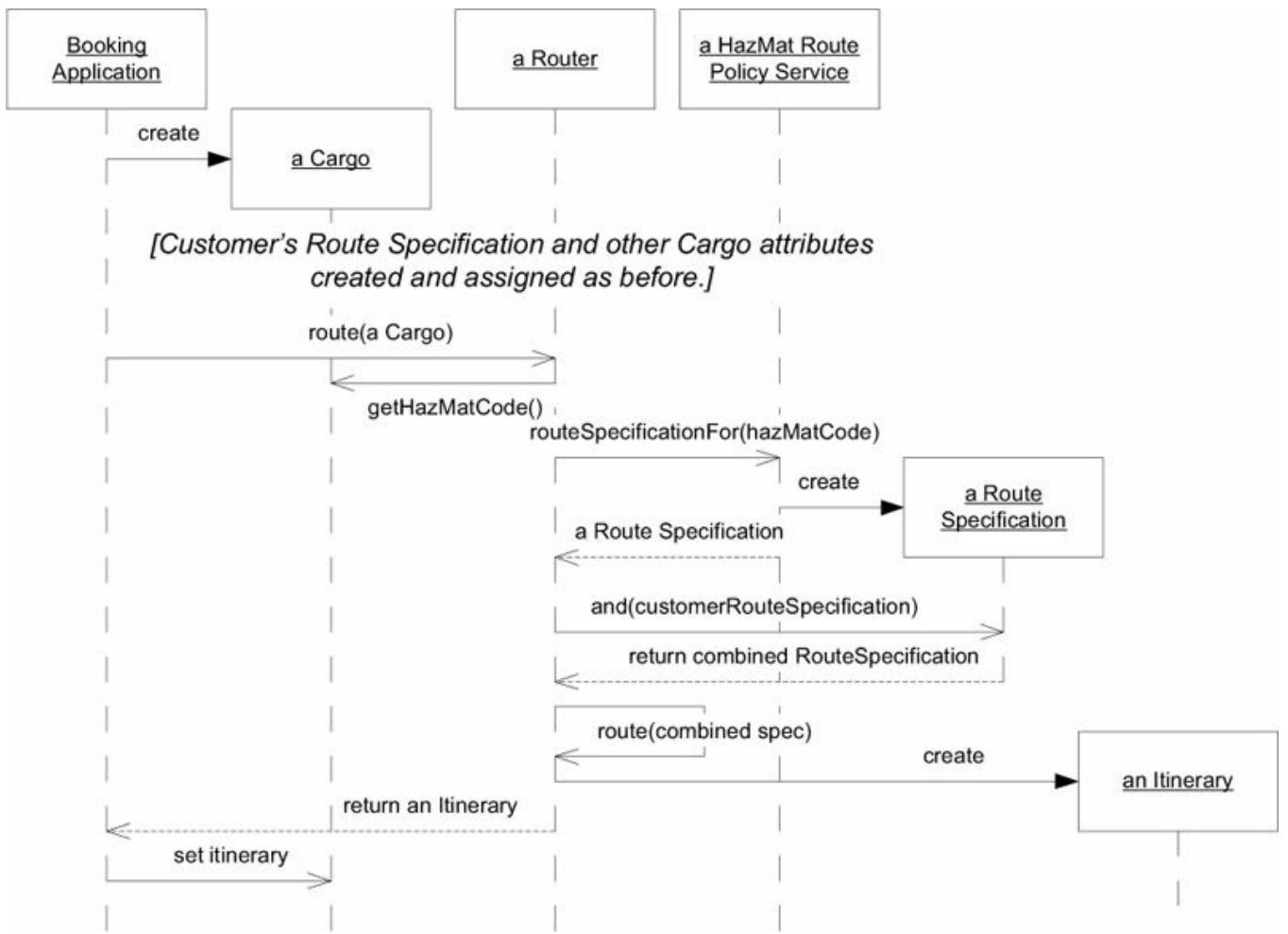












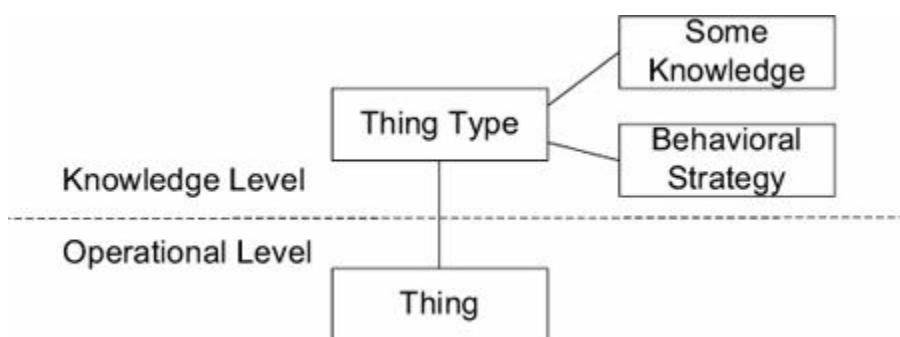
Decision	Analytical mechanisms	Very little state, so little change	Management analysis Optimize utilization Reduce cycle time ...
Policy	Strategies Constraints (based on business goals or laws)	Slow state change	Priority of products Recipes for parts ...
Operation	State reflecting business reality (of activities and plans)	Rapid state change	Inventory Status of unfinished parts ...
Potential	State reflecting business reality (of resources)	Moderate rate of state change	Process capability of equipment Equipment availability Transport through factory ...

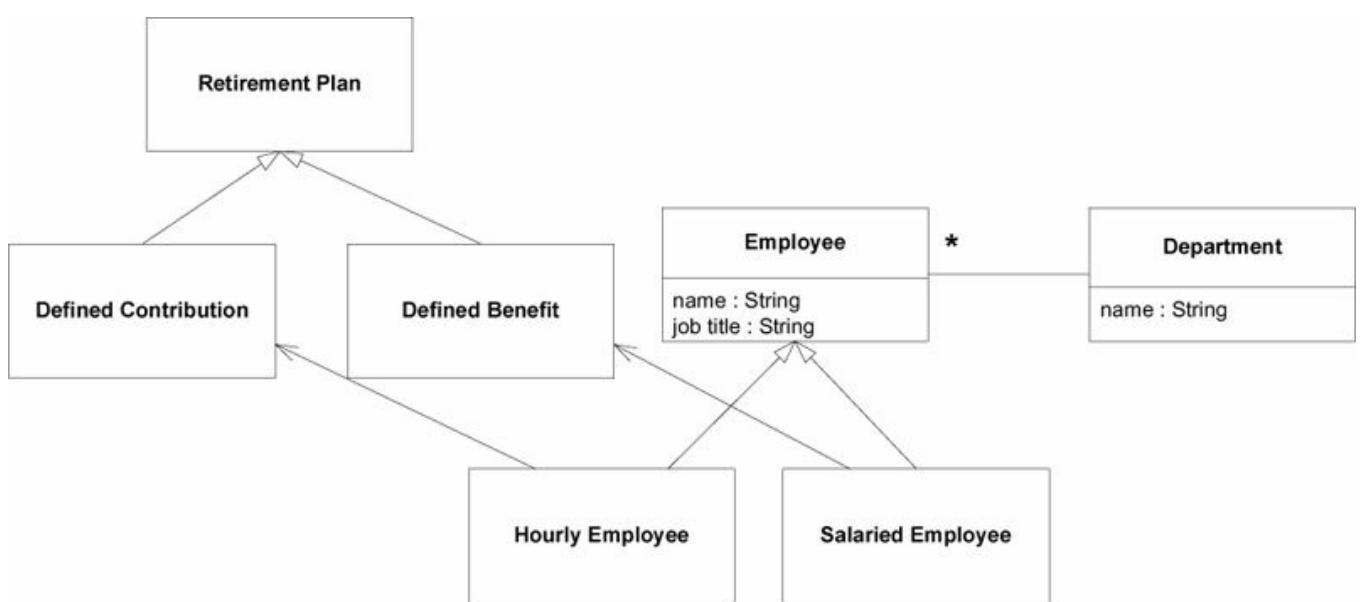
dependency

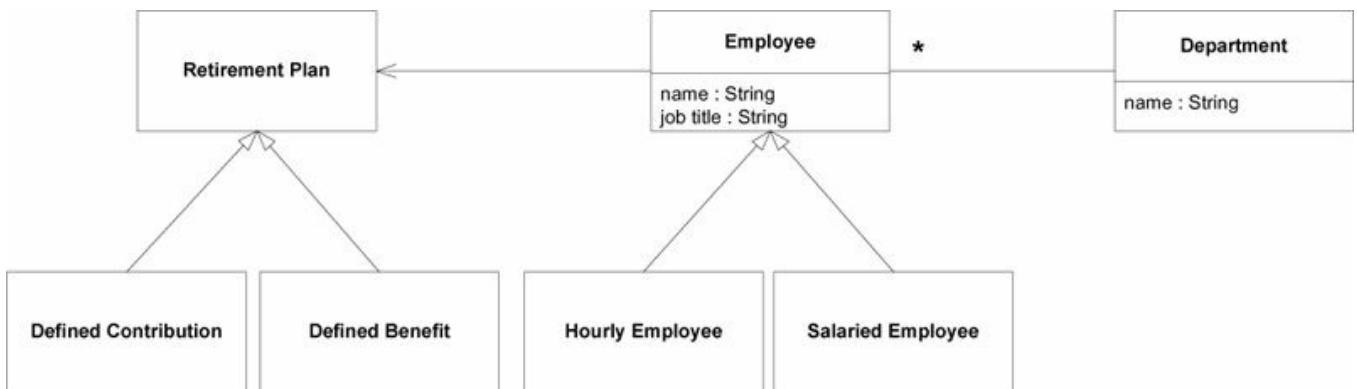
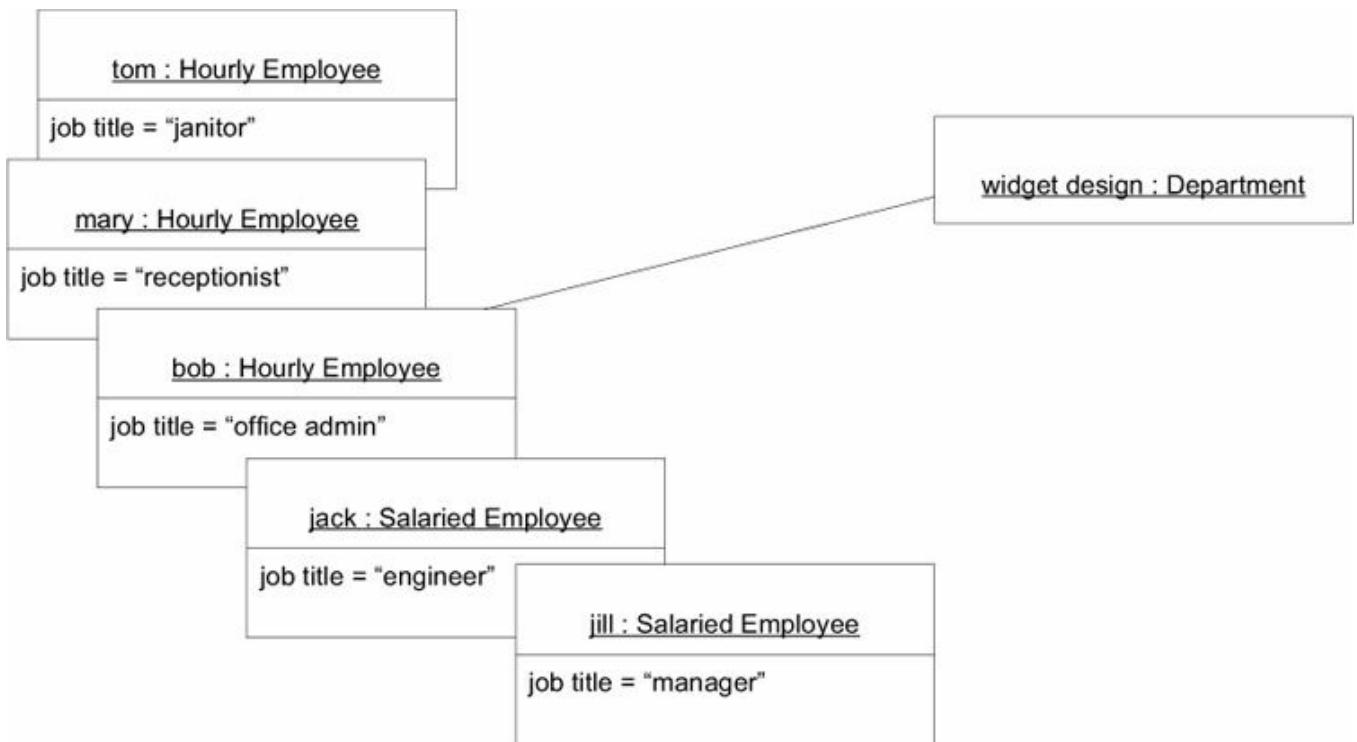


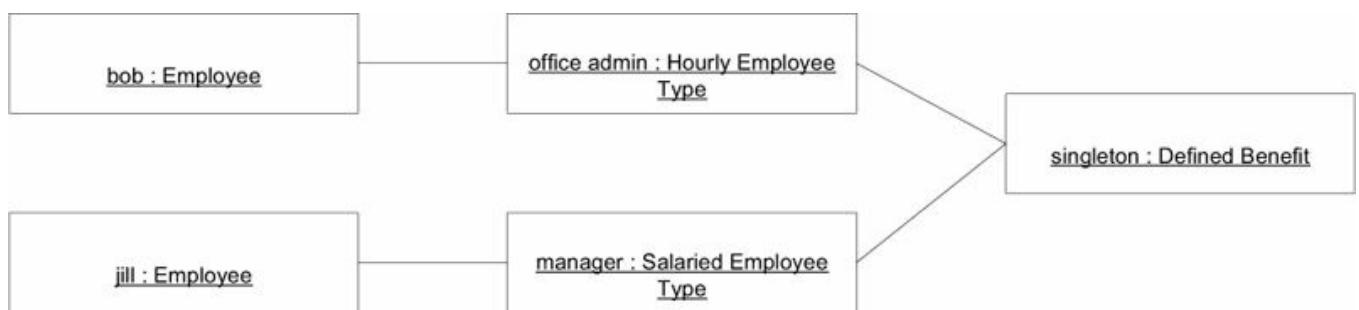
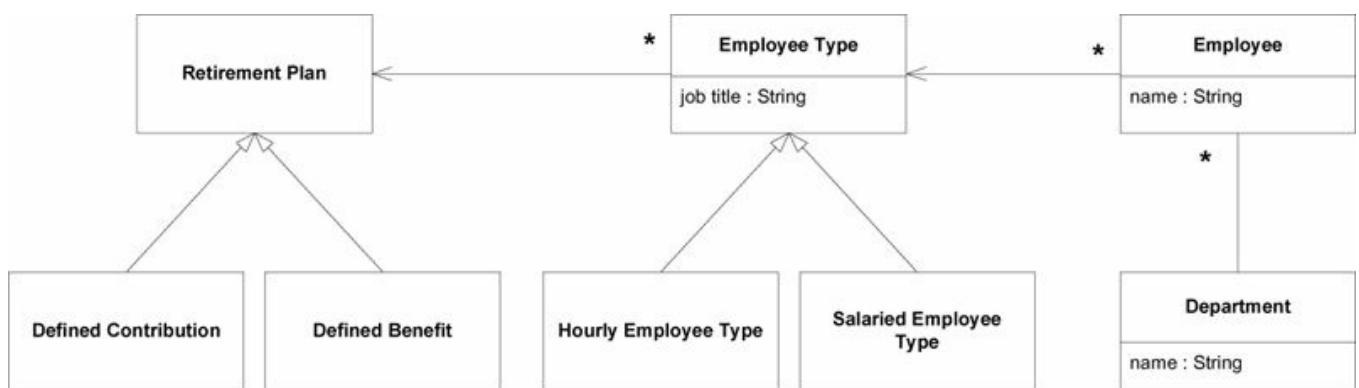
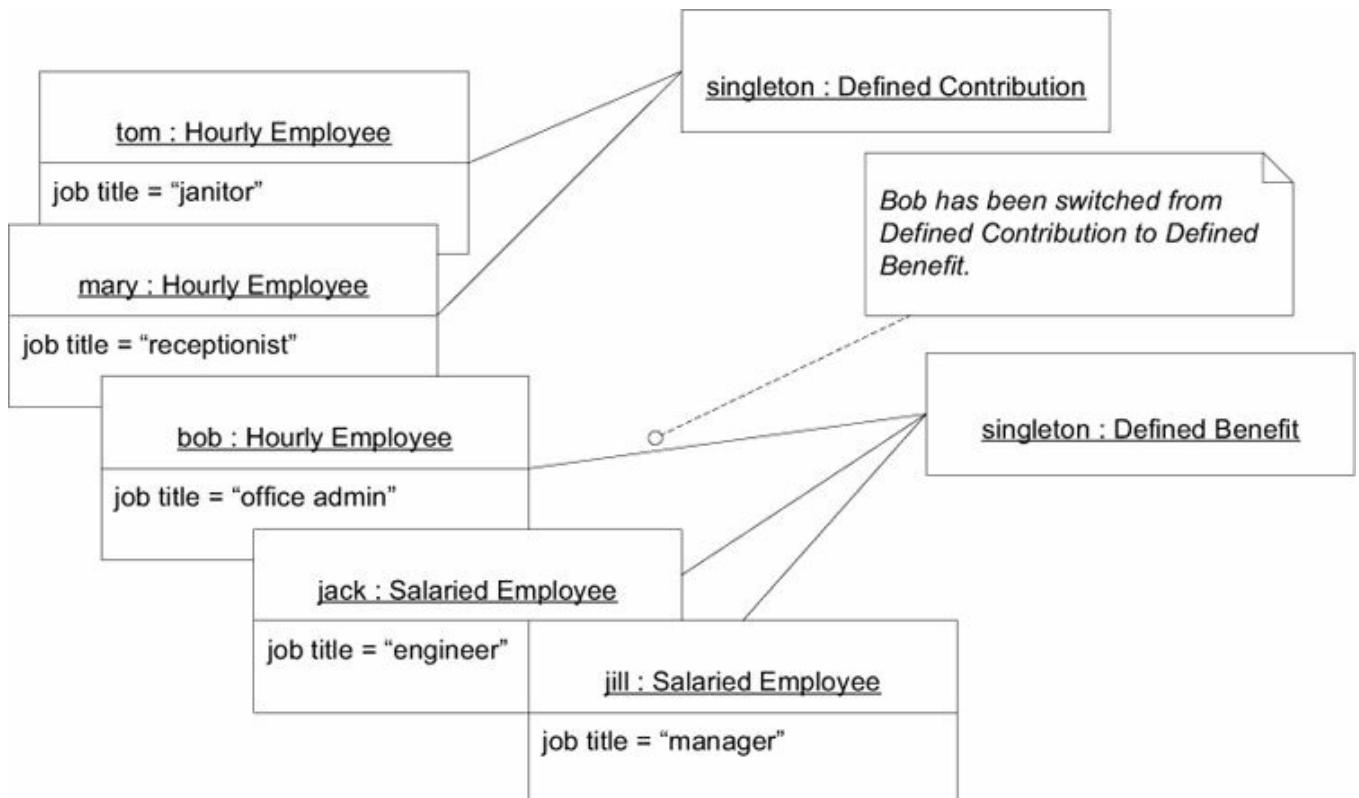
Decision	Analytical mechanisms	Very little state, so little change	Risk analysis Portfolio analysis Negotiation tools ...
Policy	Strategies Constraints (based on business goals or laws)	Slow state change	Reserve limits Asset allocation goals ...
Commitment	State reflecting business deals and contracts with customers	Moderate rate of state change	Customer agreements Syndication agreements ...
Operation	State reflecting business reality (of activities and plans)	Rapid state change	Status of outstanding loans Accruals Payments and distributions ...

↓
dependency

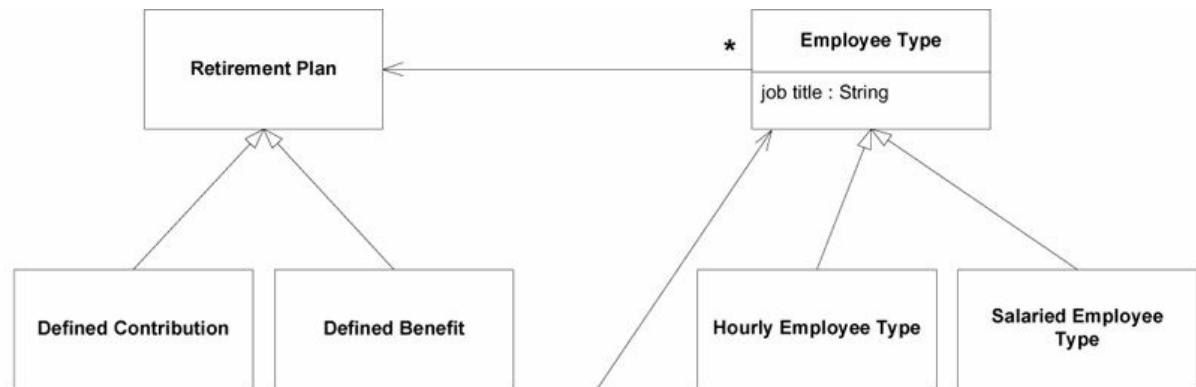


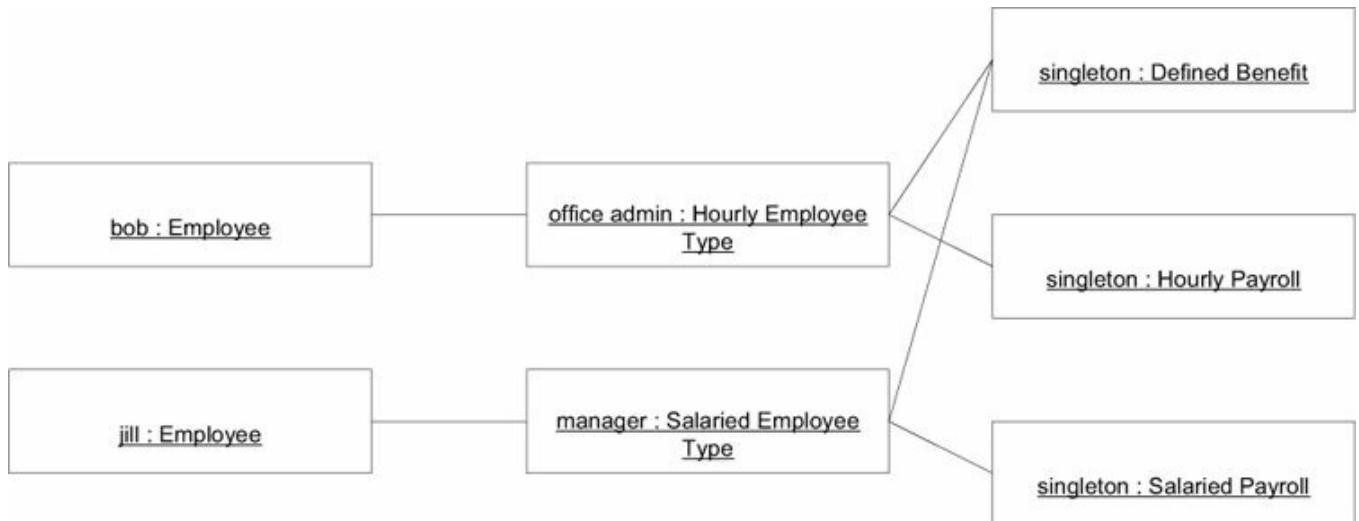
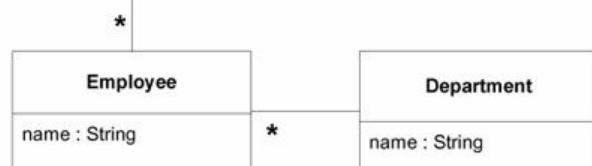
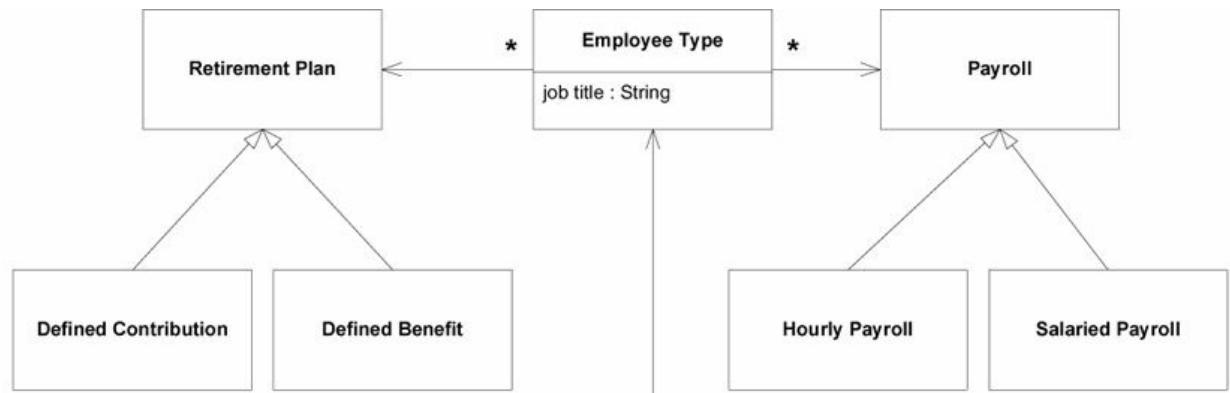




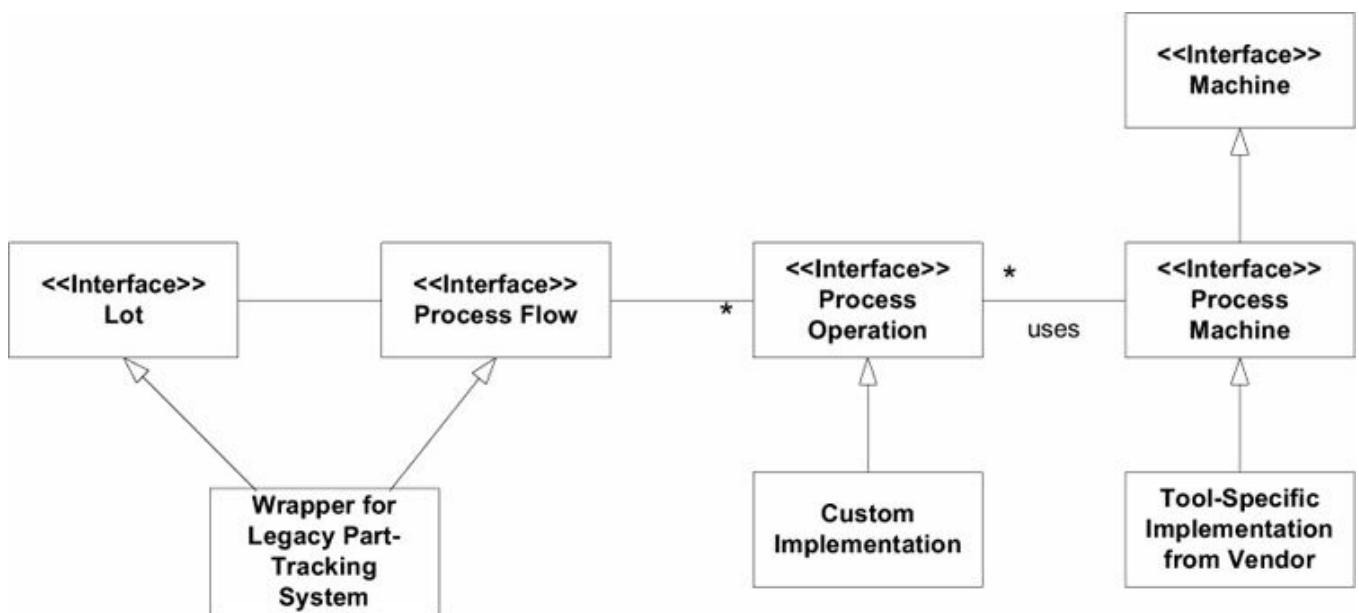


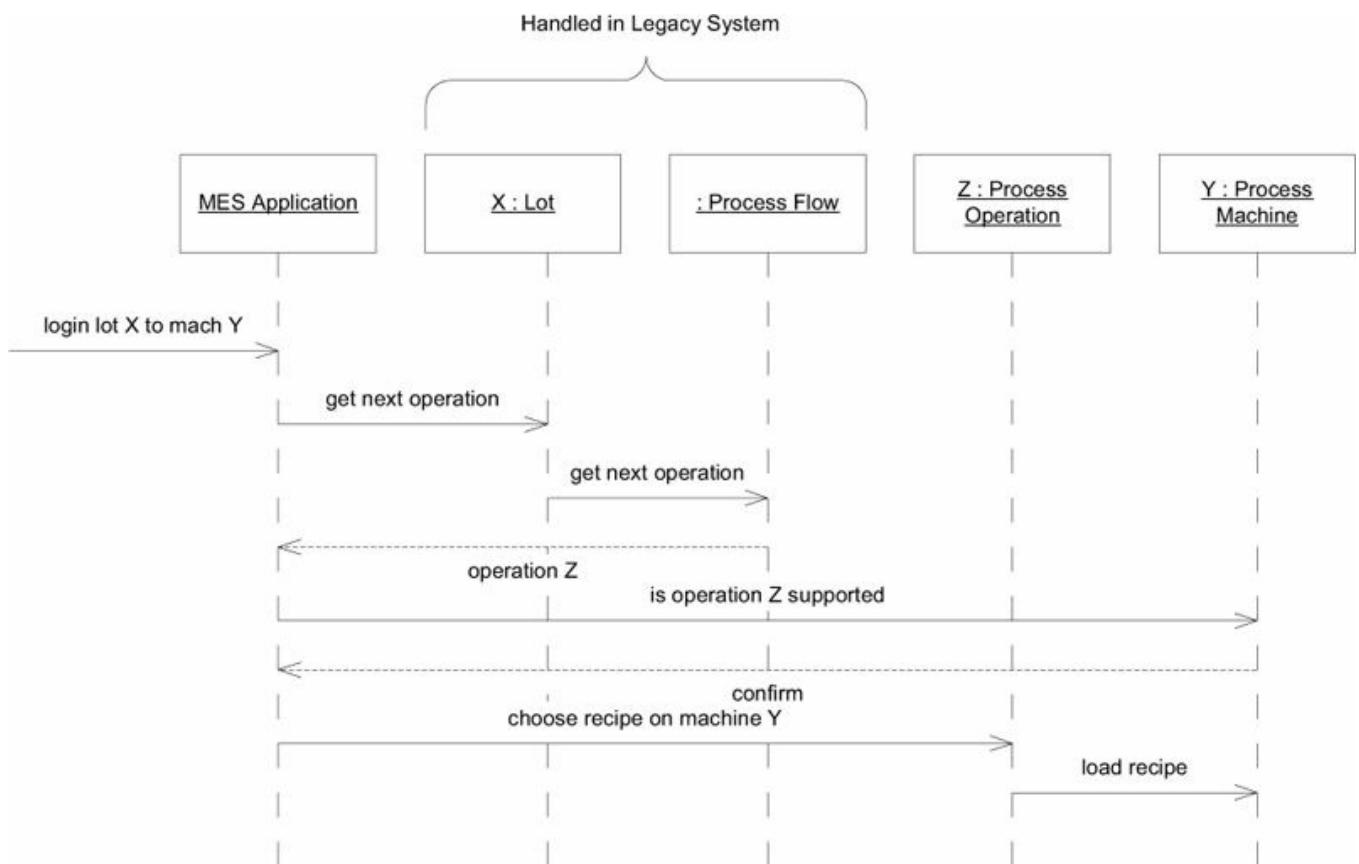
Fowler Terminology	POSA Terminology ²
Knowledge Level	Meta Level
Operations Level	Base Level





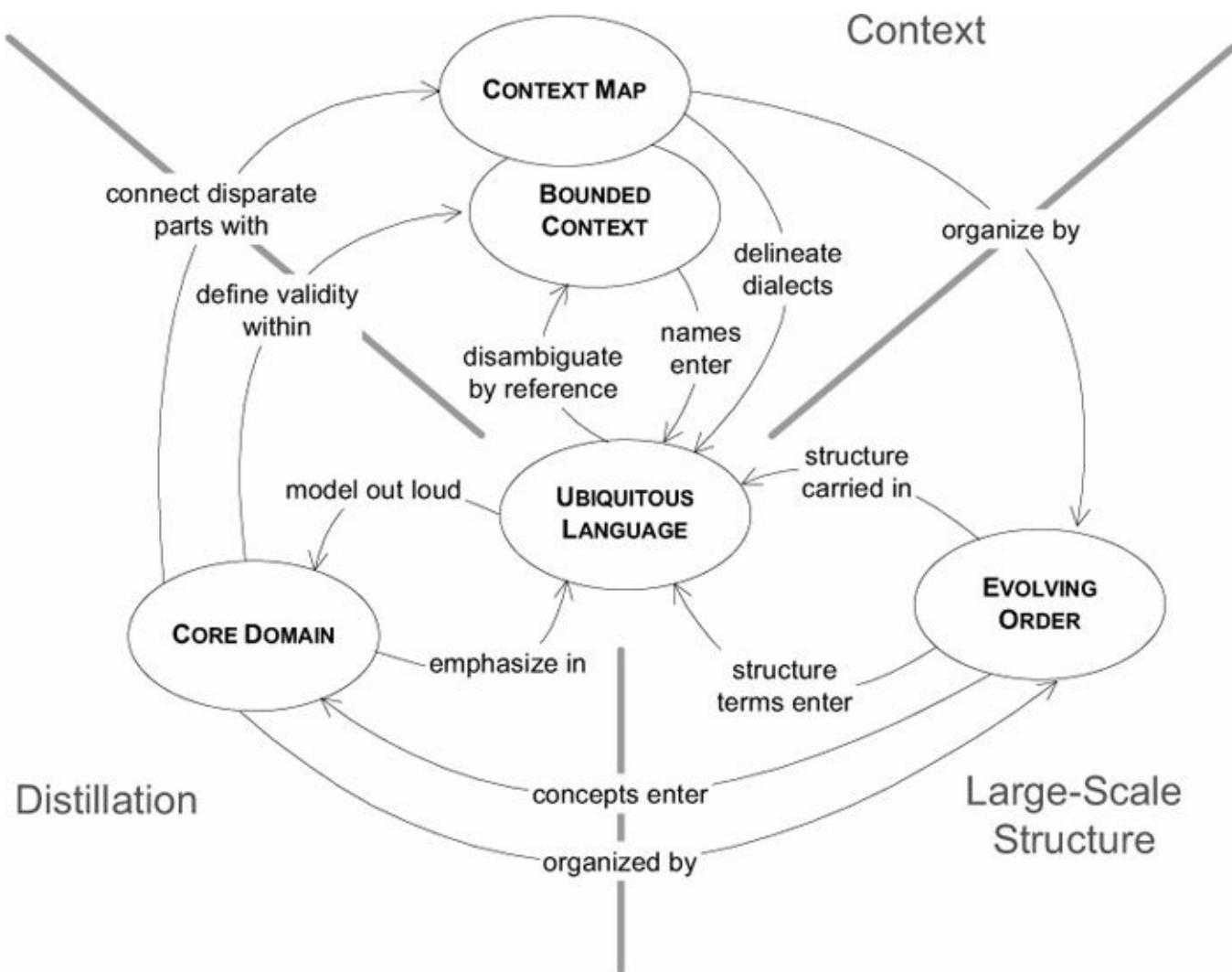
* * *

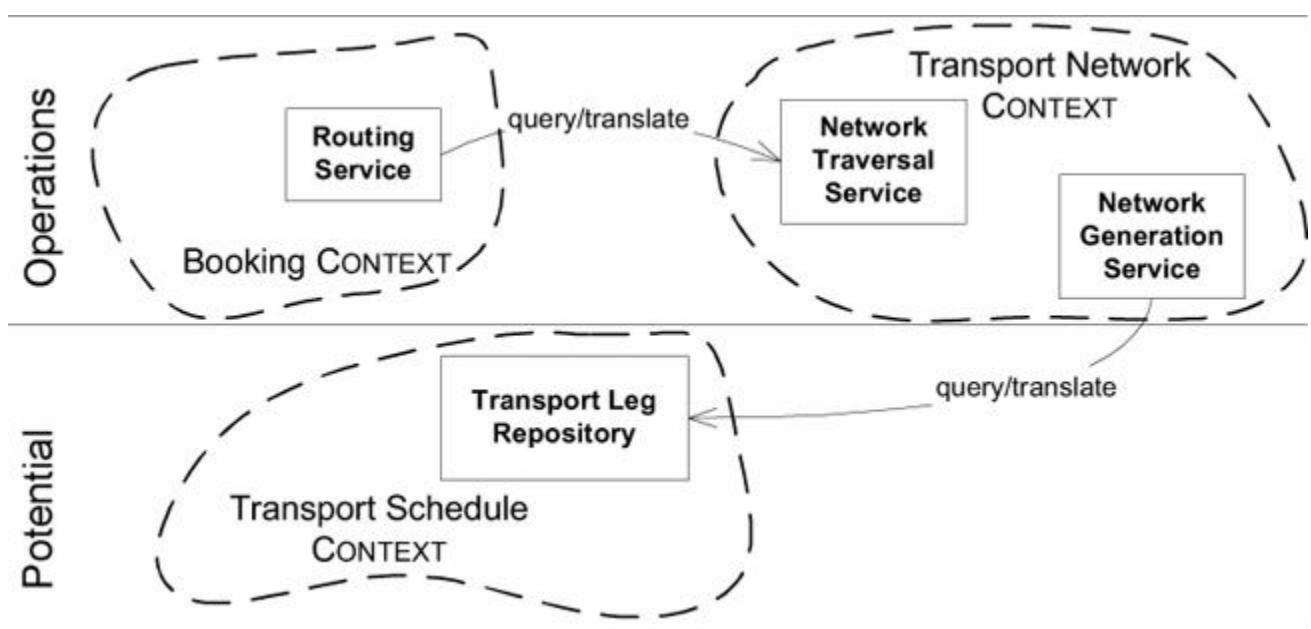
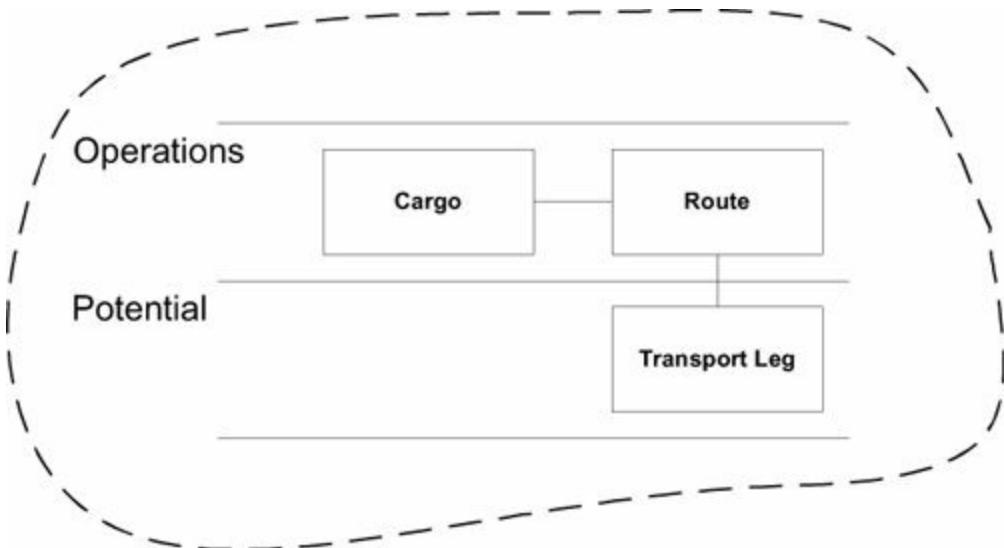


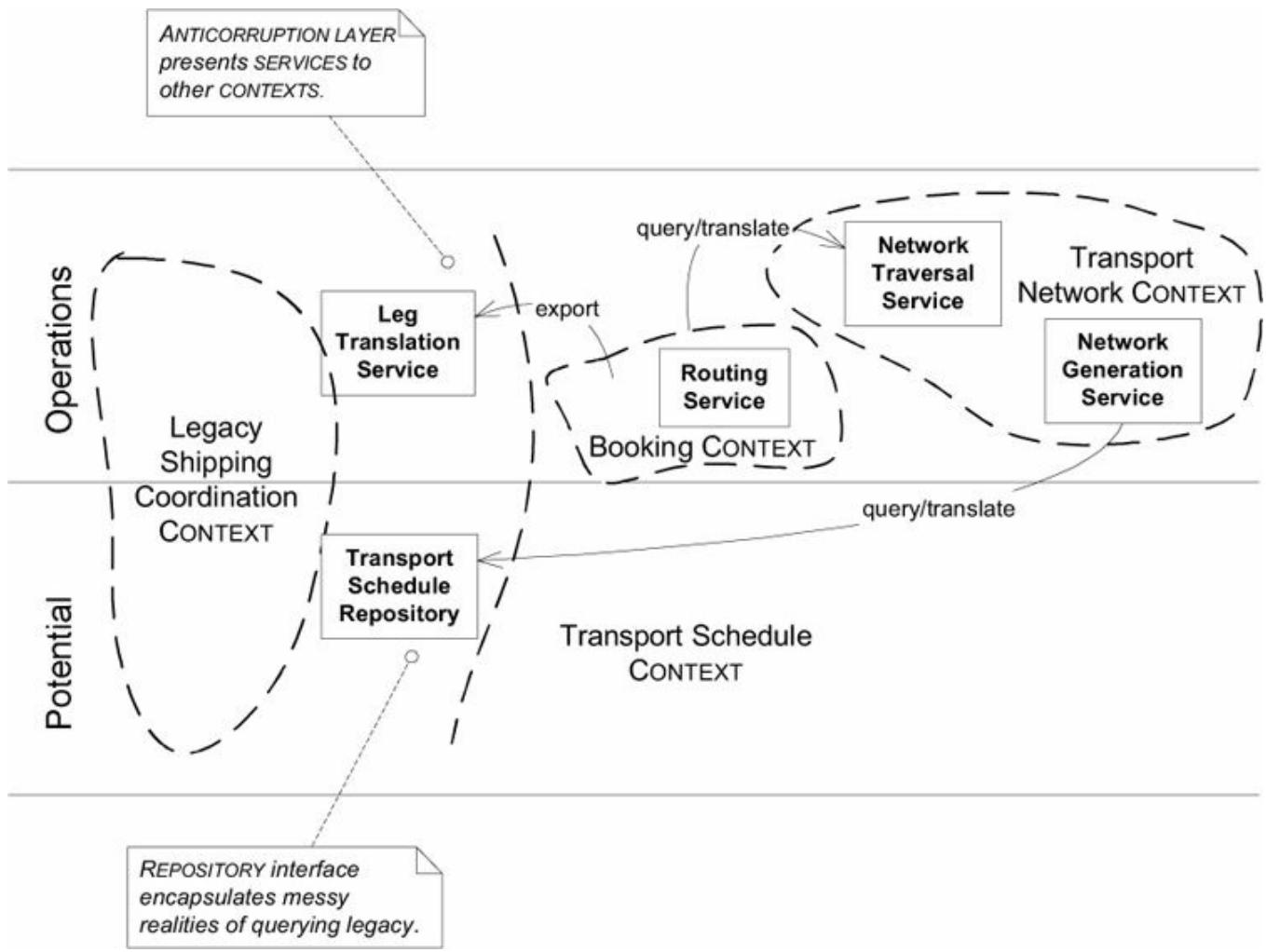


* * *

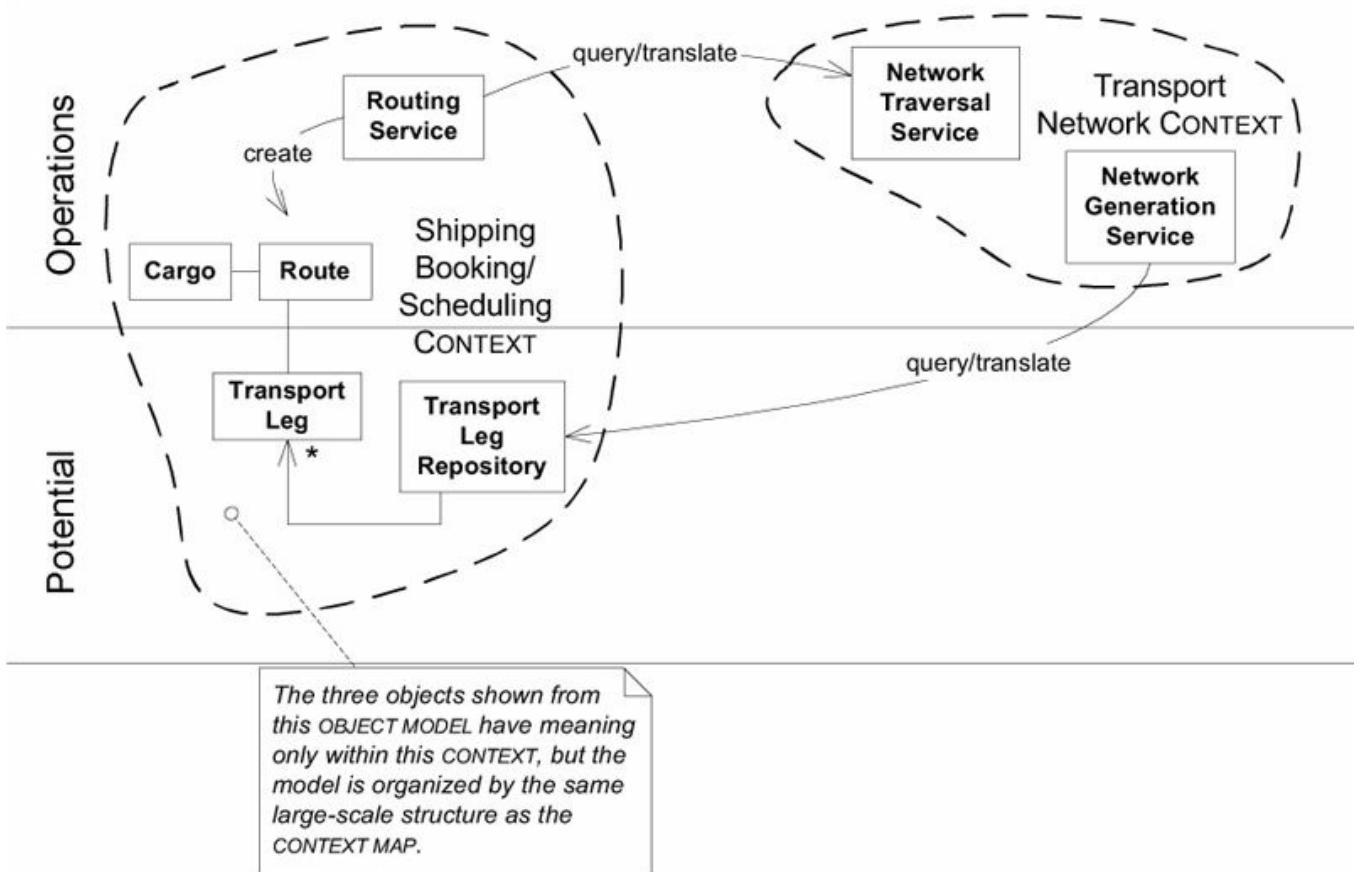




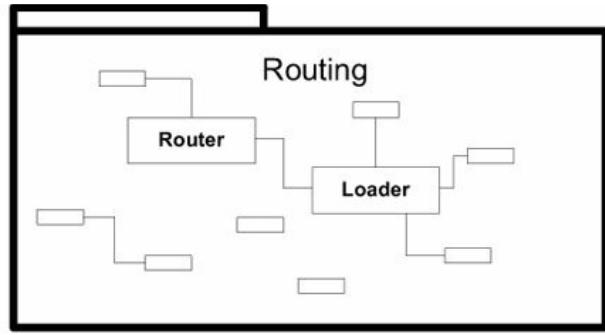




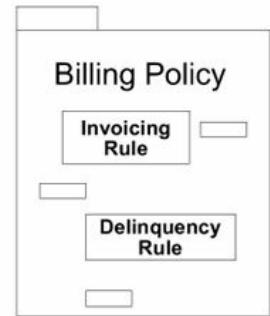
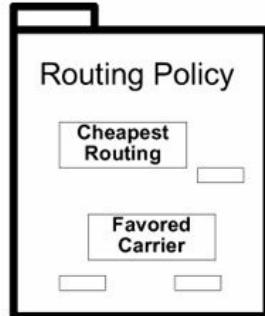
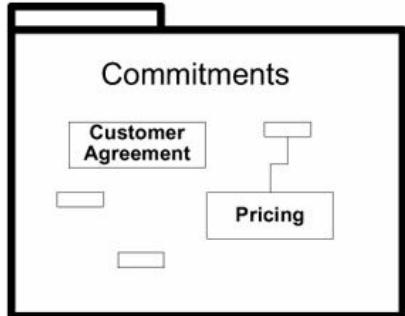
REPOSITORY interface
encapsulates messy
realities of querying legacy.



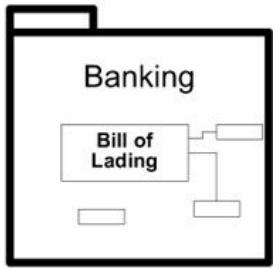
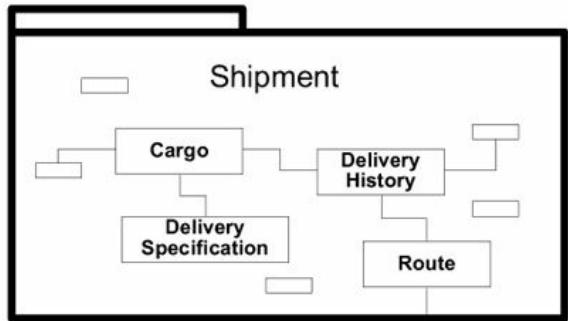
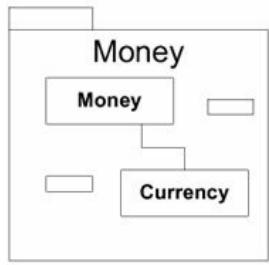
Decision



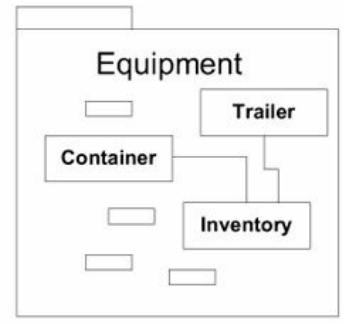
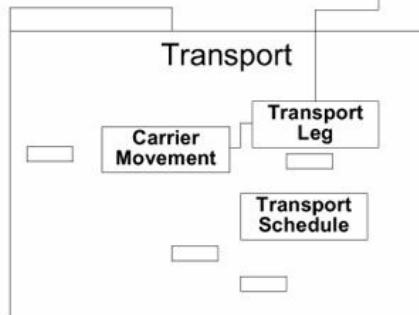
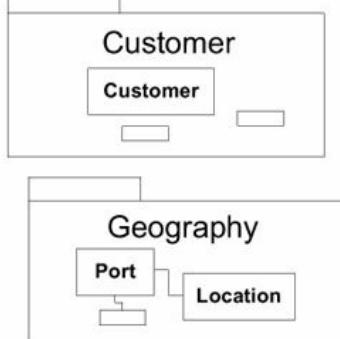
Policy



Operation



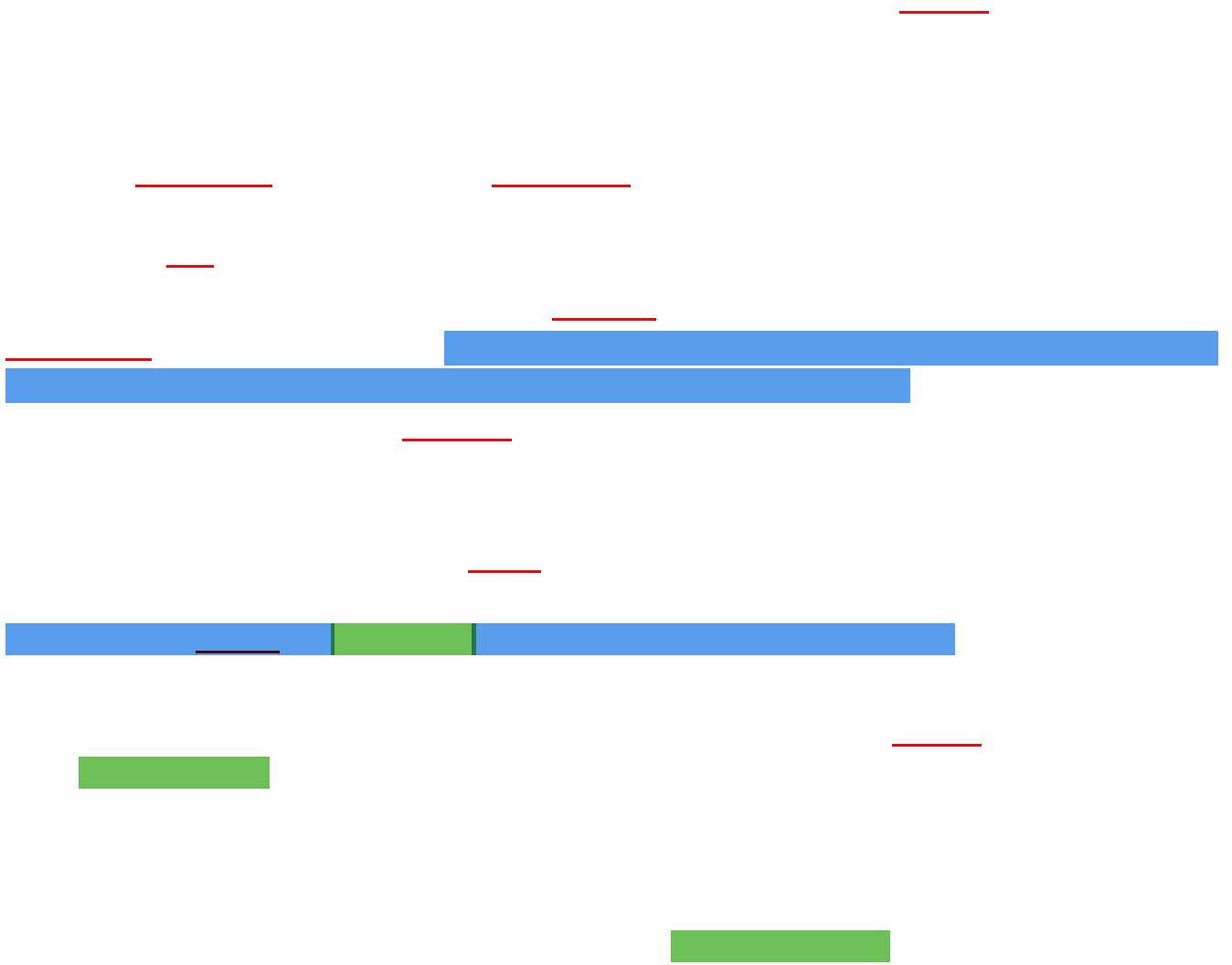
Potential

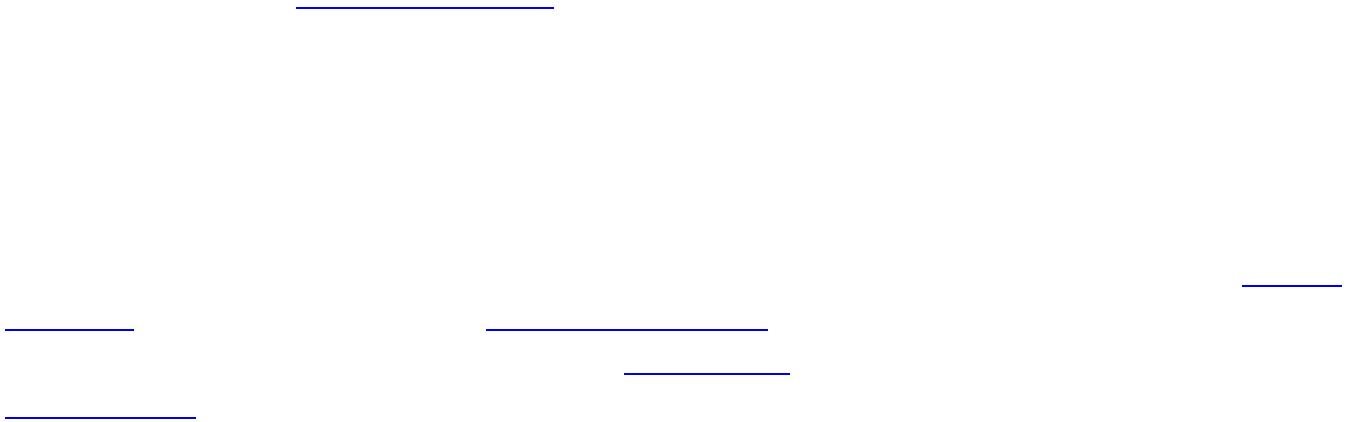












* * *

* * *

