Architecture of Enterprise Applications XV
Architectural Patterns – Service Design and State Management

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

Service Design and State Management

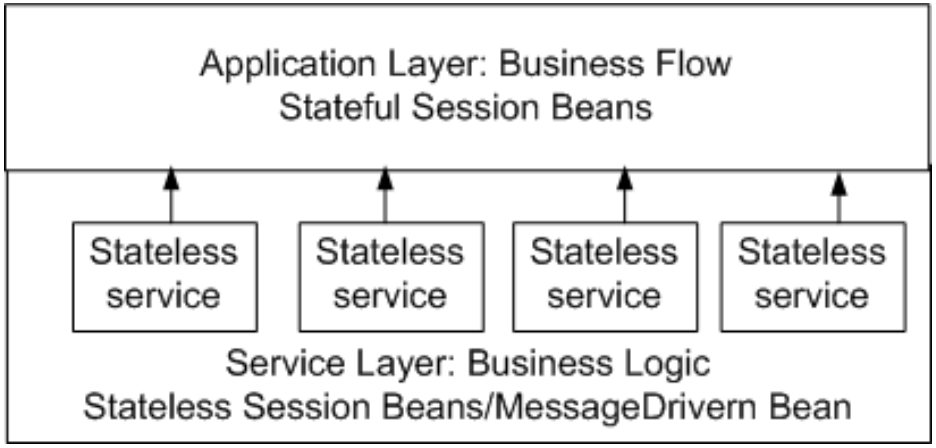
- System Requirements
  - E-learning
  - To manage all kinds of teaching activities of one of the top universities, including enrollment, courses selection, score management, assignment management, and so on.
  - This university has over 30,000 undergraduate students and graduate students, and over 4,000 teachers.
  - This university opens over 4,000 courses every semester.
  - We should keep all data for a long period.
  - It should be integrated with other e-campus systems, such as Financial System, Research Projects Management Systems.

Stateful or Stateless

- Services are stateful or stateless?
  - Up to 30,000 users
  - Stateful
    - Have to maintain states for each user
    - Maintain the states between requests
  - Stateless
    - Multiple users share the same instance of service
    - Can not maintain the states between requests
  - Prefer stateless services to stateful services

Stateful or Stateless

- Services are stateful or stateless?
  - How to hold the personal information about authentication and authority ?
    - We have to use stateful services
  - What should we design if all of the services need personal information?
    - Mediator Pattern



Synchronous or Asynchronous?

- How to communicate with services?
  - Synchronous
    - RPC
    - Programming is simple, but performance maybe low
  - Asynchronous
    - Messaging
    - Programming is complex, but performance is improved
  - Which kind of mechanism should we use if the jitter of concurrent users is great?
    - Asynchronous is better

Synchronous or Asynchronous?

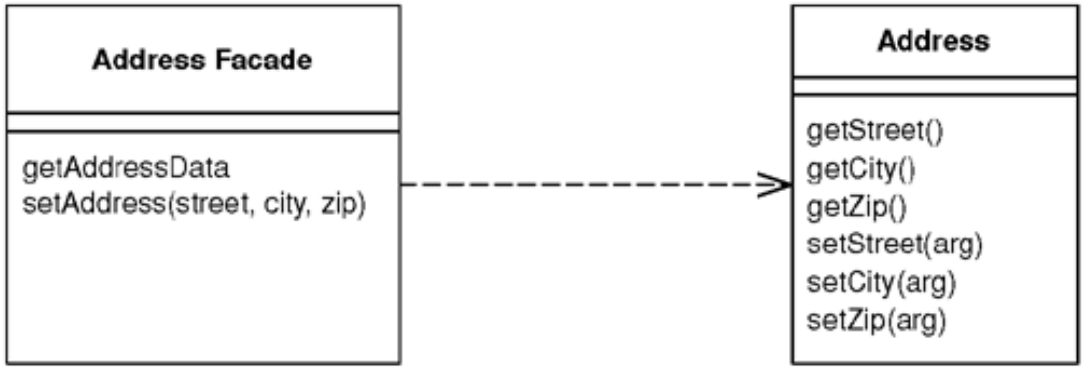
- How to communicate with services?
  - How can we get the response of asynchronous service?
    - Messaging again
    - Call backs
  - Can asynchronous service always improve the performance and shorten response time?
    - It COULD be
    - When the number of requests is always great, and its jitter is not great, messaging can not bring any improvement on performance

Granularity of Services

- Coarse-grained or Fine-grained?
  - Coarse-grained
    - Decrease the number of invocations between client and service
    - Encapsulate the business logic
    - Weaken the control on business flow
    - Make it simple to write client program
  - Fine-grained
    - Have more control on transaction, security, and so on
    - Increase the number of invocations between client and service
    - Need clients to implement business flow
    - The performance can be very low when the services are remote for client

Remote Facade

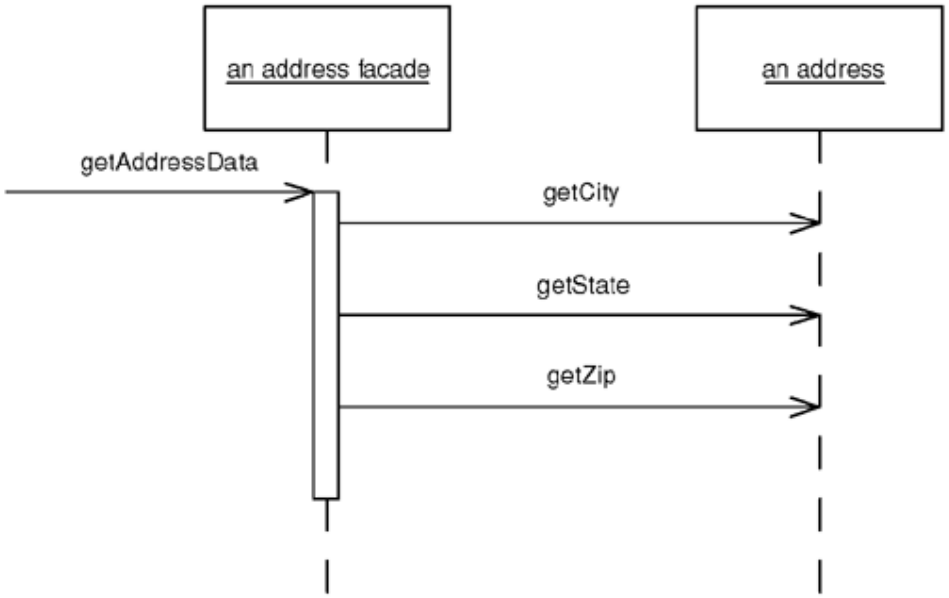
- Provides a coarse-grained facade on fine-grained objects to improve efficiency over a network.



- Any inter-process call is orders of magnitude more expensive than an in-process call.
- A Remote Facade is a coarse-grained facade over a web of fine-grained objects.

Remote Facade-How It Works

- Remote Facade tackles the distribution problem which the standard OO approach of separating distinct responsibilities into different objects; and as a result it has become the standard pattern for this problem.
  - the facade is merely a thin skin that switches from a coarse-grained to a fine-grained interface.



Remote Facade-How It Works

- In a more complex case a single Remote Facade may act as a remote gateway for many fine-grained objects.
  - For example, an order facade may be used to get and update information for an order, all its order lines, and maybe some customer data as well.
- Granularity is one of the most tricky issues with Remote Facade.
  - The facade is designed to make life simpler for external users, not for the internal system.
- Remote Facade can be stateful or stateless.

Remote Facade-How It Works

- As well as providing a coarse-grained interface, several other responsibilities can be added to a Remote Facade.
  - For example, its methods are a natural point at which to apply security. An access control list can say which users can invoke calls on which methods.
  - The Remote Facade methods also are a natural point at which to apply transactional control.

Remote Facade-How It Works

- Remote Facade and Session Façade
  - There‘s a crucial difference. Remote Facade is all about having a thin remote skin， hence my diatribe against domain logic in it.
  - In contrast, most descriptions of Session Facade involve putting logic in it, usually of a workflow kind.
  - A large part of this is due to the common approach of using J2EE session beans to wrap entity beans. Any coordination of entity beans has to be done by another object since they can't be re-entrant.

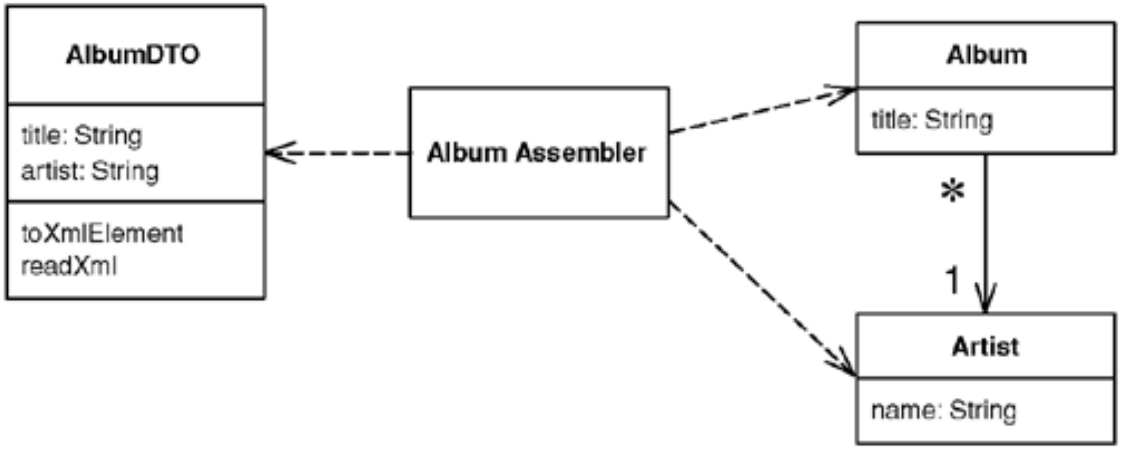
- Service Layer
  - A concept familiar to facades is a Service Layer. The main difference is that a service layer doesn't have to be remote and thus doesn't need to have only fine-grained methods.

Remote Facade-When to Use It

- Use Remote Facade whenever you need remote access to a fine-grained object model.
- Most often you run into this with different processes on different machines, but it turns out that the cost of an inter-process call on the same box is sufficiently large that you need a coarse-grained interface for any inter-process communication regardless of where the processes live.
- Remote Facades imply a synchronous--that is, a remote procedure call--style of distribution.

Data Transfer Object

- An object that carries data between processes in order to reduce the number of method calls.



- When you're working with a remote interface, each call to it is expensive.
- The solution is to create a Data Transfer Object that can hold all the data for the call.
  - It needs to be serializable to go across the connection.
  - Usually an assembler is used on the server side to transfer data between the DTO and any domain objects.

Data Transfer Object-How It Works

REliable, INtelligent & Scalable Systems

- Serializing the Data Transfer Object

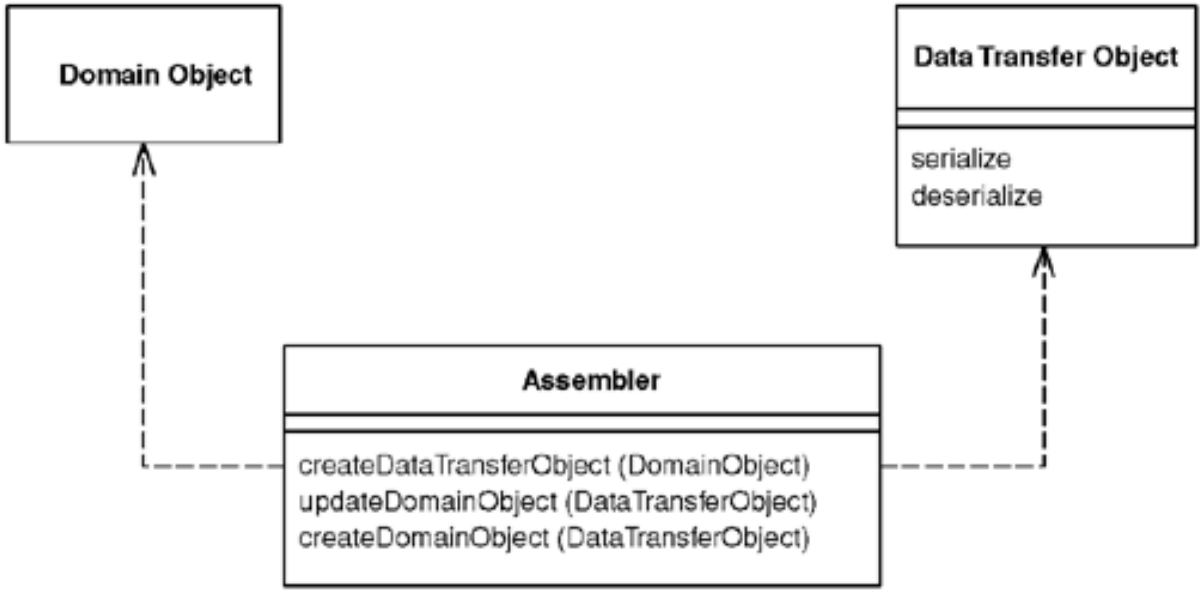
- Other than simple getters and setters, the Data Transfer Object is also usually responsible for serializing itself into some format that will go over the wire.
  - Which format depends on what's on either side of the connection, what can run over the connection itself, and how easy the serialization is.

- You have to choose a mechanism that both ends of the connection will work with.
  - If you control both ends, you pick the easiest one; if you don't, you may be able to provide a connector at the end you don't own.
  - Then you can use a simple Data Transfer Object on both sides of the connection and use the connector to adapt to the foreign component.

- An important factor for serialization is the synchronization of the Data Transfer Object on each side of the wire.
  - In theory, whenever the server changes the definition of the Data Transfer Object, the client updates as well but in practice this may not happen.

Data Transfer Object-How It Works

- Assembling a Data Transfer Object from Domain Objects



Data Transfer Object-When to Use It

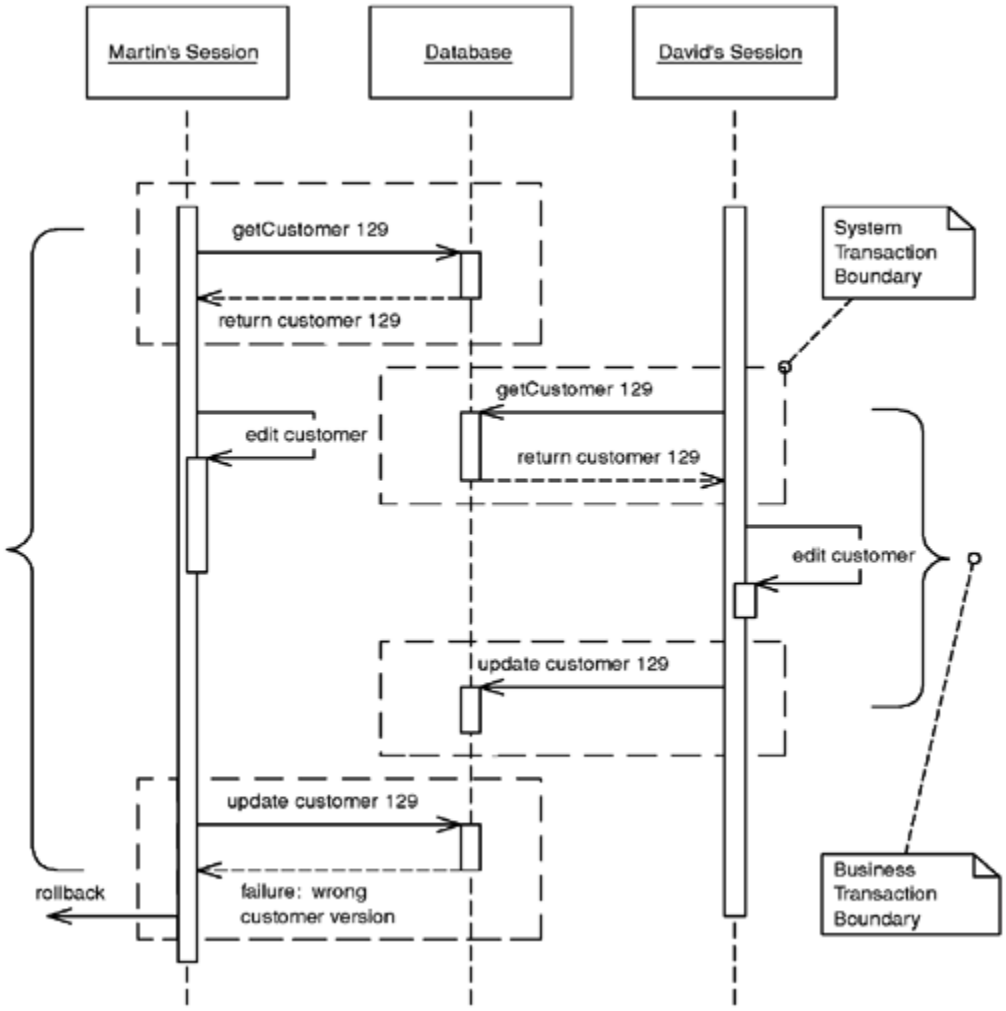
- Use a Data Transfer Object whenever you need to transfer multiple items of data between two processes in a single method call.
- There are some alternatives to Data Transfer Object.
  - One is to not use an object at all but simply to use a setting method with many arguments or a getting method with several pass-by reference arguments.
  - Another alternative is to use a some form of string representation directly, without an object acting as the interface to it.
- Another common purpose for a Data Transfer Object is to act as a common source of data for various components in different layers.

Concurrency

- Since we use O/R mapping, and it is an offline way to access database, we should manage the concurrent access to data.
- The core is how to prevent confliction between business transactions.
- Locking is the solution
  - What kind of lock?

Optimistic Offline Lock

- Prevents conflicts between concurrent business transactions by detecting a conflict and rolling back the transaction.
- Often a business transaction executes across a series of system transactions.
  - Once outside the confines of a single system transaction, we can't depend on our database manager alone to ensure that the business transaction will leave the record data in a consistent state.
- Optimistic Offline Lock solves this problem by validating that the changes about to be committed by one session don't conflict with the changes of another session.



Optimistic Offline Lock-How It Works

REin
REliable, INtelligent & Scalable Systems

- An Optimistic Offline Lock is obtained by validating that, in the time since a session loaded a record, another session hasn't altered it.
  - It can be acquired at any time but is valid only during the system transaction in which it is obtained.
  - Thus, in order that a business transaction not corrupt record data it must acquire an Optimistic Offline Lock for each member of its change set during the system transaction in which it applies changes to the database.
- The most common implementation is to associate a version number with each record in your system.
  - With an RDBMS data store the verification is a matter of adding the version number to the criteria of any SQL statements used to update or delete a record.
- In addition to a version number for each record, storing information as to who last modified a record and when can be quite useful when managing concurrency conflicts.
- In an alternative implementation the where clause in the update includes every field in the row.

Optimistic Offline Lock-When to Use It

- Optimistic concurrency management is appropriate when the chance of conflict between any two business transactions is low.

- As optimistic locking is much easier to implement and not prone to the same defects and runtime errors as a Pessimistic Offline Lock, consider using it as the default approach to business transaction conflict management in any system you build.

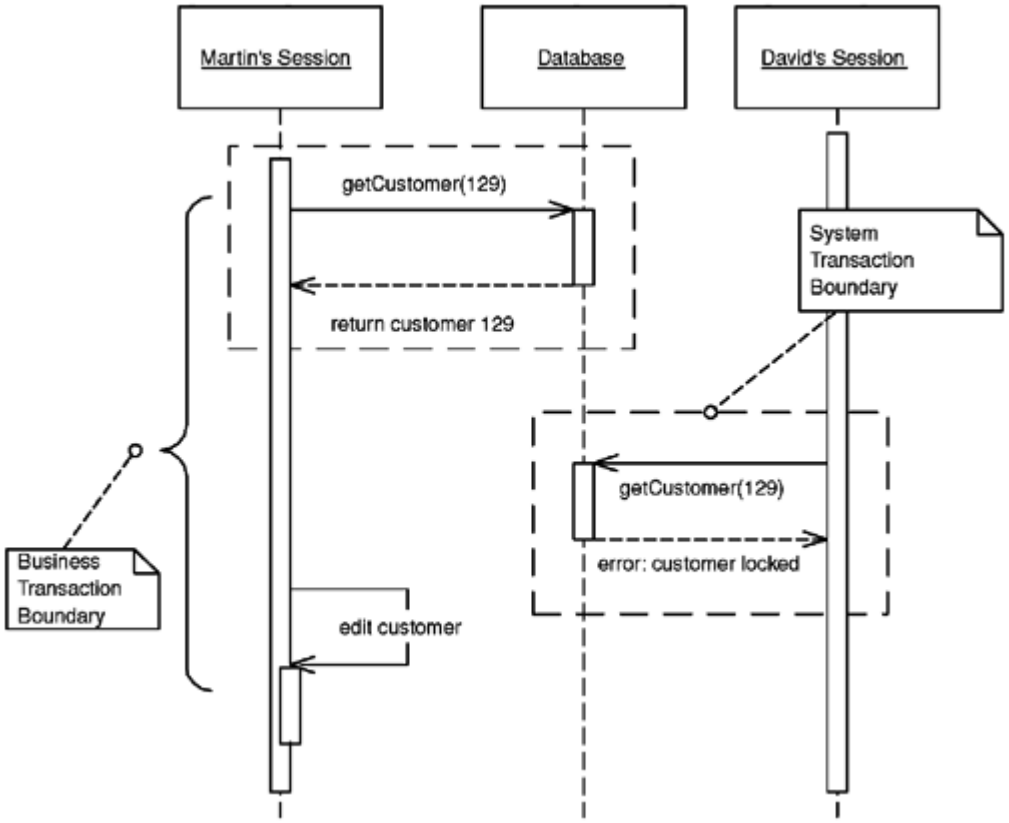
Optimistic Offline Lock-Example

class DomainObject...
```
private Timestamp modified;
private String modifiedBy;
private int version;
```

table customer...
```
create table customer(id bigint primary key, name varchar, createdby varchar, created
    datetime, modifiedby varchar, modified datetime, version int)
```
SQL customer CRUD...
```
INSERT INTO customer VALUES (?, ?, ?, ?, ?, ?, ?)
SELECT * FROM customer WHERE id = ?
UPDATE customer SET name = ?, modifiedBy = ?, modified = ?, version = ?
   WHERE id = ? and version = ?
DELETE FROM customer WHERE id = ? and version = ?
```

Pessimistic Offline Lock

- Prevents conflicts between concurrent business transactions by allowing only one business transaction at a time to access data.
- Pessimistic Offline Lock prevents conflicts by avoiding them altogether.
  - It forces a business transaction to acquire a lock on a piece of data before it starts to use it, so that, most of the time, once you begin a business transaction you can be pretty sure you'll complete it without being bounced by concurrency control.



Pessimistic Offline Lock-How It Works

- You implement Pessimistic Offline Lock in three phases: determining what type of locks you need, building a lock manager, and defining procedures for a business transaction to use locks.

- Lock types:
  - exclusive write lock
  - exclusive read lock
  - read/write lock
    - Read and write locks are mutually exclusive.
    - Concurrent read locks are acceptable.

- In choosing the correct lock type think about maximizing system concurrency, meeting business needs, and minimizing code complexity.
  - Also keep in mind that the locking strategy must be understood by domain modelers and analysts.

- Once you've decided upon your lock type, define your lock manager. The lock manager's job is to grant or deny any request by a business transaction to acquire or release a lock.
  - The lock, whether implemented as an object or as SQL against a database table, should remain private to the lock manager.
  - Business transactions should interact only with the lock manager, never with a lock object.

Pessimistic Offline Lock-When to Use It

- Pessimistic Offline Lock is appropriate when the chance of conflict between concurrent sessions is high. A user should never have to throw away work.
- If you have to use Pessimistic Offline Lock, you should also consider a long transaction.
- Don't use these techniques if your business transactions fit within a single system transaction.

Pessimistic Offline Lock-Example

- Simple Lock Manager (Java)

```
interface ExclusiveReadLockManager...
 public static final ExclusiveReadLockManager INSTANCE = (ExclusiveReadLockManager)
     Plugins.getPlugin(ExclusiveReadLockManager.class);
 public void acquireLock(Long lockable, String owner) throws ConcurrencyException;
 public void releaseLock(Long lockable, String owner);
 public void relaseAllLocks(String owner);
```

Pessimistic Offline Lock-Example

- We'll write a lock manager that interacts directly with a lock table in our database rather than with a lock object.

table lock... create table lock(lockableid bigint primary key, ownerid bigint)

class ExclusiveReadLockManagerDBImpl implements ExclusiveLockManager...

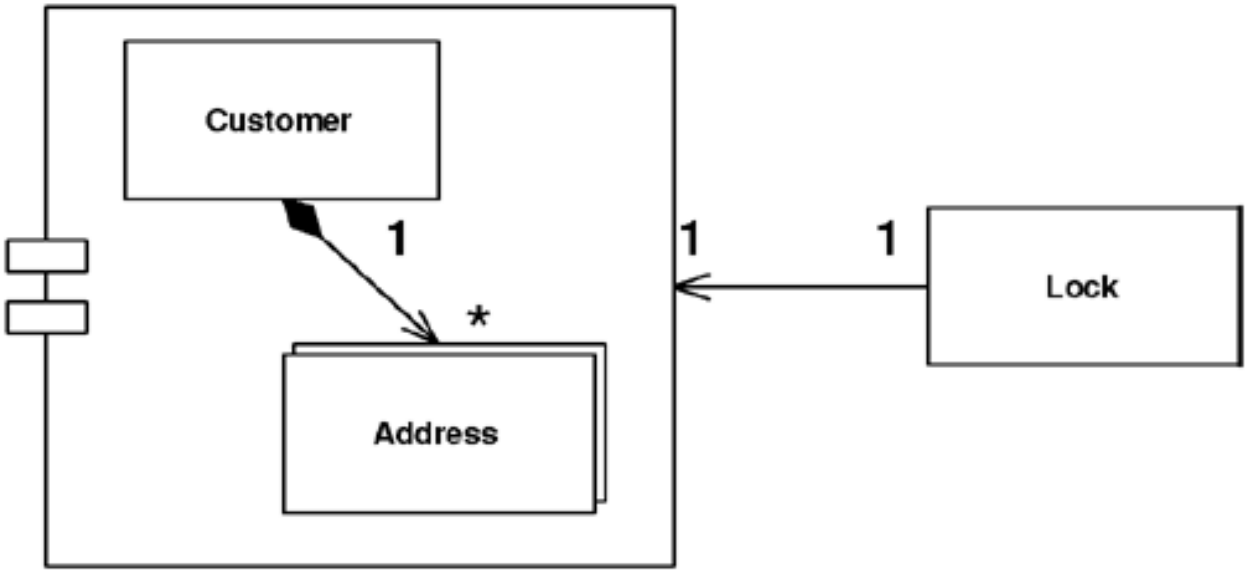
```
private static final String INSERT_SQL = "insert into lock values(?, ?)";
private static final String DELETE_SINGLE_SQL = "delete from lock where
    lockableid = ? and ownerid = ?";
private static final String DELETE_ALL_SQL = "delete from lock where ownerid
    = ?";
private static final String CHECK_SQL = "select lockableid from lock where
    lockableid = ? and ownerid = ?";
```

Pessimistic Offline Lock-Example


```
public void acquireLock(Long lockable, String owner) throws ConcurrencyException {
  if (!hasLock(lockable, owner)) {
   Connection conn = null;
   PreparedStatement pstmt = null;
   try {
     conn = ConnectionManager.INSTANCE.getConnection();
     pstmt = conn.prepareStatement(INSERT_SQL);
     pstmt.setLong(1, lockable.longValue());
     pstmt.setString(2, owner);
     pstmt.executeUpdate();
   } catch (SQLException sqlEx) { throw new ConcurrencyException("unable to lock " + lockable);
   } finally { closeDBResources(conn, pstmt); }
 }
}
 public void releaseLock(Long lockable, String owner) {
  Connection conn = null;
  PreparedStatement pstmt = null;
  try {
   conn = ConnectionManager.INSTANCE.getConnection();
   pstmt = conn.prepareStatement(DELETE_SINGLE_SQL);
   pstmt.setLong(1, lockable.longValue());
   pstmt.setString(2, owner);
   pstmt.executeUpdate();
  } catch (SQLException sqlEx) { throw new SystemException("unexpected error releasing lock on " + lockable);
  } finally { closeDBResources(conn, pstmt); }
 }
```

Coarse-Grained Lock

- Locks a set of related objects with a single lock.



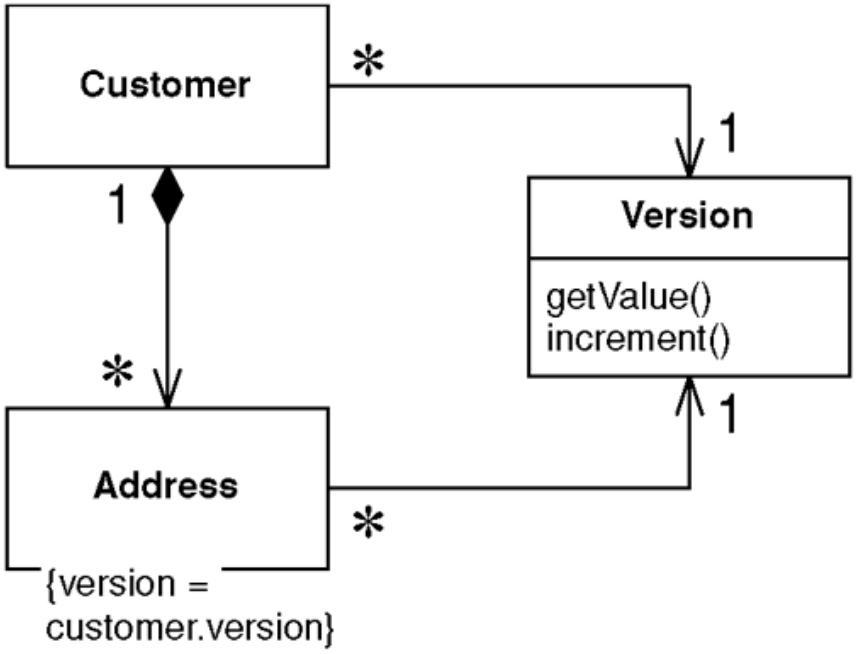
- A Coarse-Grained Lock is a single lock that covers many objects. It not only simplifies the locking action itself but also frees you from having to load all the members of a group in order to lock them.

Coarse-Grained Lock-How It Works

- The first step in implementing Coarse-Grained Lock is to create a single point of contention for locking a group of objects.
  - This makes only one lock necessary for locking the entire set.
  - Then you provide the shortest path possible to finding that single lock point in order to minimize the group members that must be identified and possibly loaded into memory in the process of obtaining that lock.

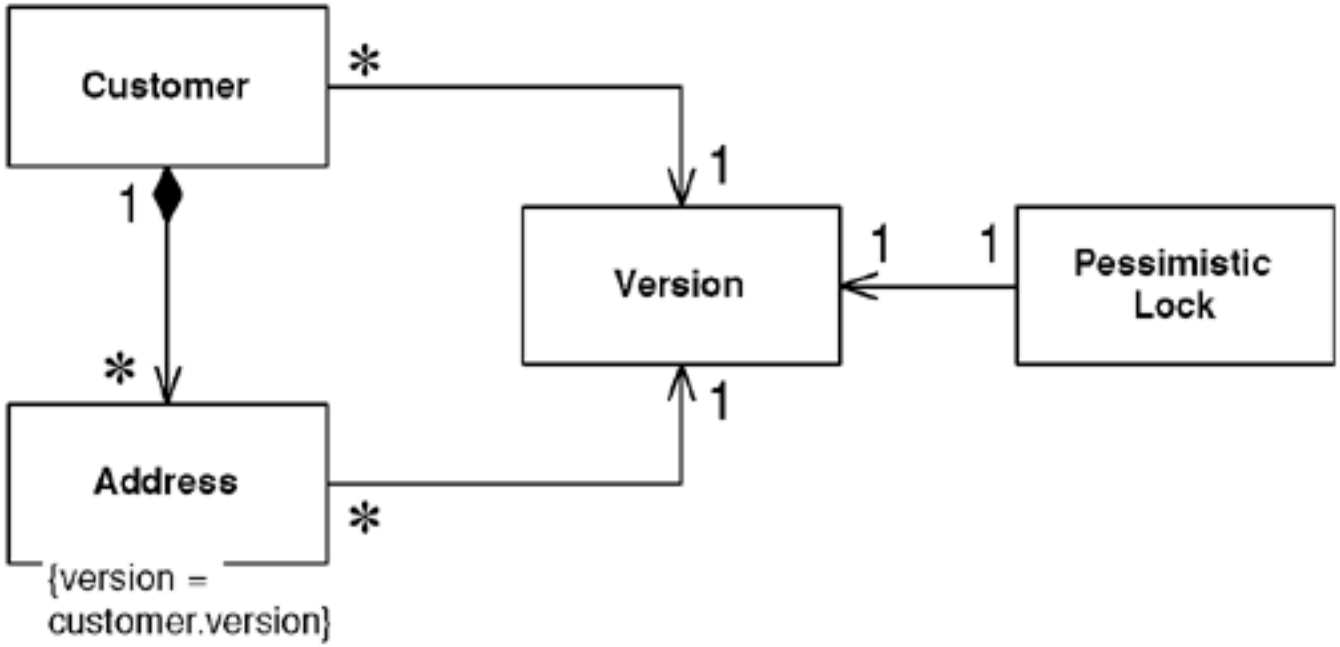
Coarse-Grained Lock-How It Works

- With Optimistic Offline Lock, having each item in a group share a version creates the single point of contention, which means sharing the same version, not an equal version.
  - Incrementing this version will lock the entire group with a shared lock.
  - Set up your model to point every member of the group at the shared version and you have certainly minimized the path to the point of contention.



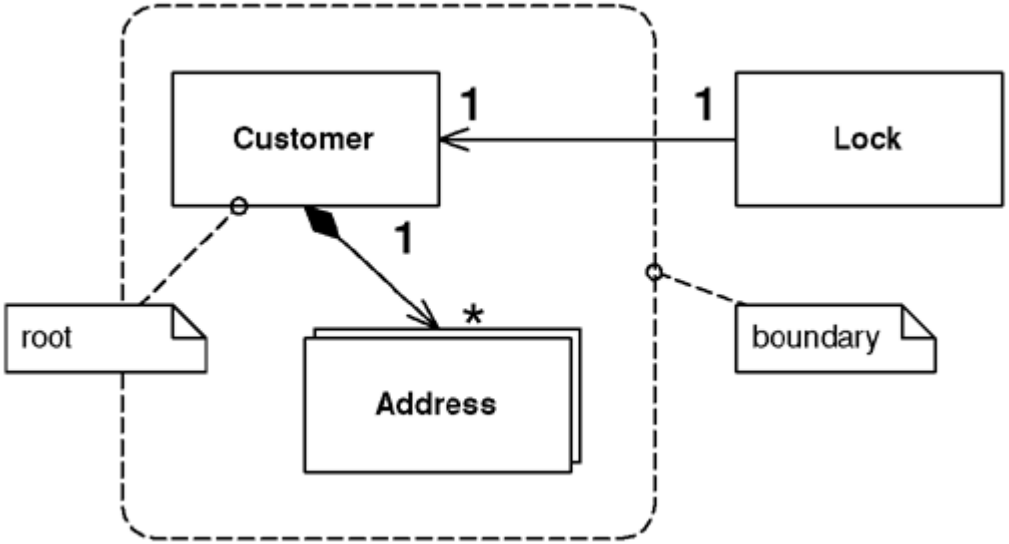
Coarse-Grained Lock-How It Works

- A shared Pessimistic Offline Lock requires that each member of the group share some sort of lockable token, on which it must then be acquired.
  - As Pessimistic Offline Lock is often used as a complement to Optimistic Offline Lock, a shared version object makes an excellent candidate for the lockable token role.



Coarse-Grained Lock-How It Works

- Eric Evans and David Siegel define an aggregate as a cluster of associated objects that we treat as a unit for data changes.
  - Each aggregate has a root that provides the only access point to members of the set and a boundary that defines what's included in the set.
  - The aggregate's characteristics call for a Coarse-Grained Lock, since working with any of its members requires locking all of them.
  - By definition locking the root locks all members of the aggregate. The root lock gives us a single point of contention.



Coarse-Grained Lock-How It Works

- Using a root lock as a Coarse-Grained Lock makes it necessary to implement navigation to the root in your object graph.

- The shared lock and root lock implementations of Coarse-Grained Lock both have their trade-offs.
  - When using a relational database the shared lock carries the burden that almost all of your selects will require a join to the version table.
  - But loading objects while navigating to the root can be a performance hit as well.
  - The root lock and Pessimistic Offline Lock perhaps make an odd combination. By the time you navigate to the root and lock it you may need to reload a few objects to guarantee their freshness.

- Locking implementations abound, and the subtleties are even more numerous. Be sure to arrive at an implementation that suits your needs.

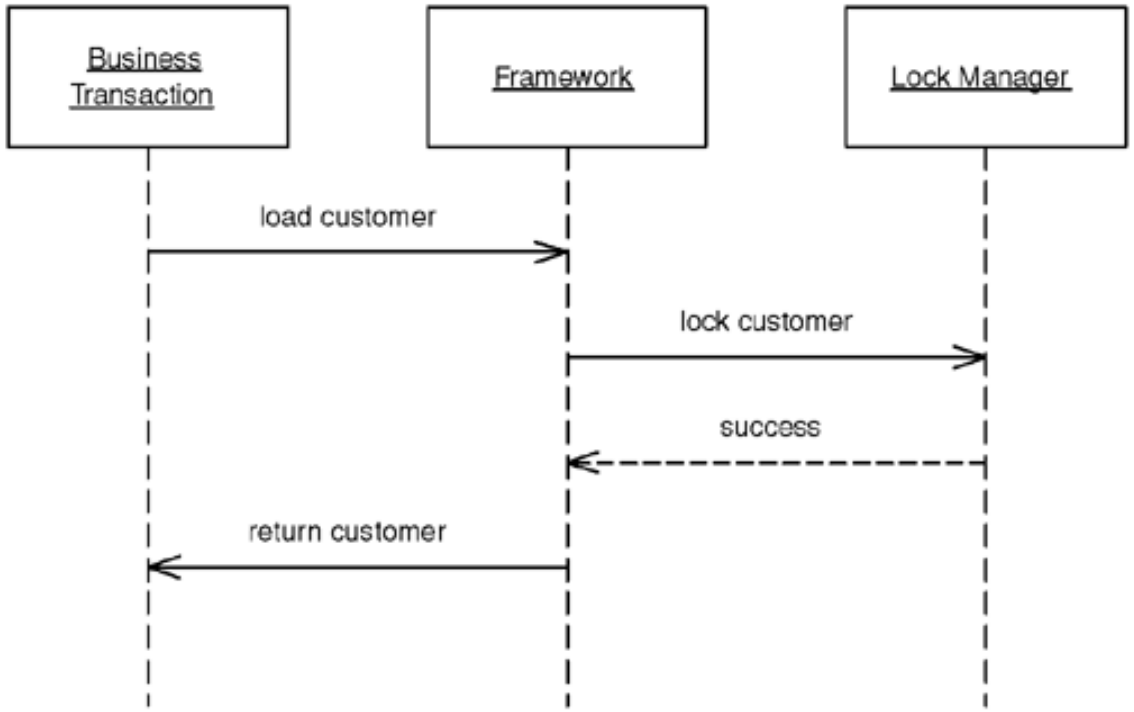
Coarse-Grained Lock-When to Use It

- The most obvious reason to use a Coarse-Grained Lock is to satisfy business requirements.
  - This is the case when locking an aggregate.
  - Consider a lease object that owns a collection of assets. It probably doesn't make business sense for one user to edit the lease and another user to simultaneously edit an asset. Locking either the asset or the lease ought to result in the lease and all of its assets being locked.

- A very positive outcome of using Coarse-Grained Locks is that acquiring and releasing lock is cheaper.
  - This is certainly a legitimate motivation for using them.
  - The shared lock can be used beyond the concept of the aggregate, but be careful when working from nonfunctional requirements such as performance.
  - Beware of creating unnatural object relationships in order to facilitate Coarse-Grained Lock.

Implicit Lock

- Allows framework or layer supertype code to acquire offline locks.



- Because offline concurrency management is difficult to test, errors might go undetected by all of your test suites.
- One solution is to not allow developers to make such a mistake. Locking tasks that cannot be overlooked should be handled not explicitly by developers but implicitly by the application.

Implicit Lock-How It Works

- Implementing Implicit Lock is a matter of factoring your code such that any locking mechanics that absolutely cannot be skipped can be carried out by your application framework.

- The first step is to assemble a list of what tasks are mandatory for a business transaction to work within your locking strategy.
  - For Optimistic Offline Lock that list will contain items such as storing a version count for each record, including the version in update SQL criteria, and storing an incremented version when changing the record.
  - The Pessimistic Offline Lock list will include items along the lines of acquiring any lock necessary to load a piece of data---typically the exclusive read lock or the read portion of the read/write lock---and releasing all locks when the business transaction or session completes.

- Second, and just as important, is that these lock types most greatly limit system concurrency.
  - Making locking work is a matter of determining the best way to get your framework to implicitly carry out the locking mechanics.

Implicit Lock-When to Use It

- Implicit Lock should be used in all but the simplest of applications that have no concept of framework. The risk of a single forgotten lock is too great.

Implicit Lock-Example

- Implicit Pessimistic Offline Lock (Java)
- We'll write a locking mapper that acquires a lock before attempting to find an object.

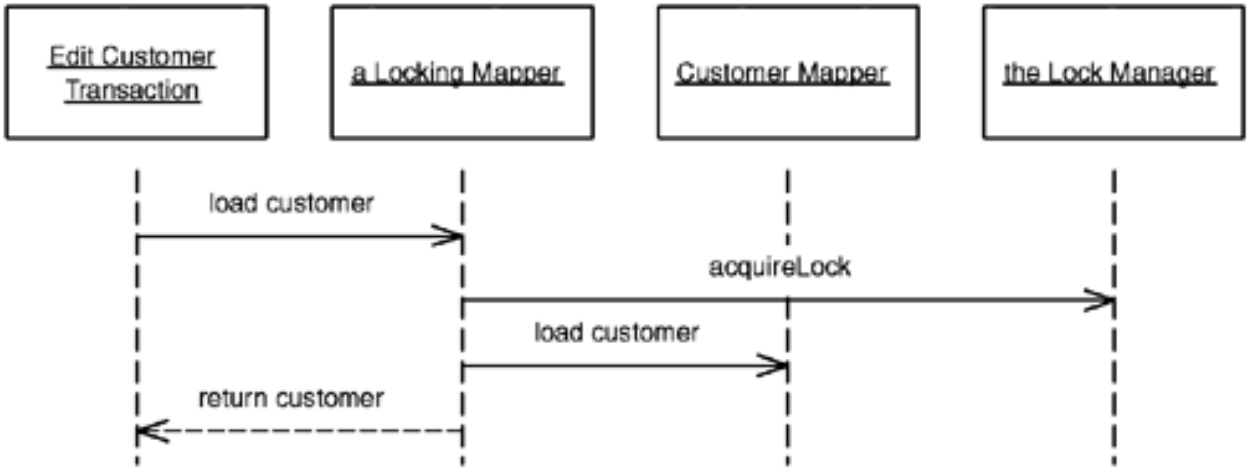
```
interface Mapper...
 public DomainObject find(Long id);
 public void insert(DomainObject obj);
 public void update(DomainObject obj);
 public void delete(DomainObject obj);

class LockingMapper implements Mapper...
 private Mapper impl;
 public LockingMapper(Mapper impl) { this.impl = impl; }
 public DomainObject find(Long id) {
   ExclusiveReadLockManager.INSTANCE.acquireLock( id,
      AppSessionManager.getSession().getId());
 return impl.find(id);
 }
 public void insert(DomainObject obj) { impl.insert(obj); }
 public void update(DomainObject obj) { impl.update(obj); }
 public void delete(DomainObject obj) { impl.delete(obj); }
```

Implicit Lock-Example

- One of the nice things about decorators is that the object being wrapped doesn't even know that it's functionality is being enhanced. Here we can wrap the mappers in our registry:

```
LockingMapperRegistry implements MappingRegistry...
 private Map mappers = new HashMap();
 public void registerMapper(Class cls, Mapper mapper) {
  mappers.put(cls, new LockingMapper(mapper));
 }
 public Mapper getMapper(Class cls) {
  return (Mapper) mappers.get(cls);
 }
```



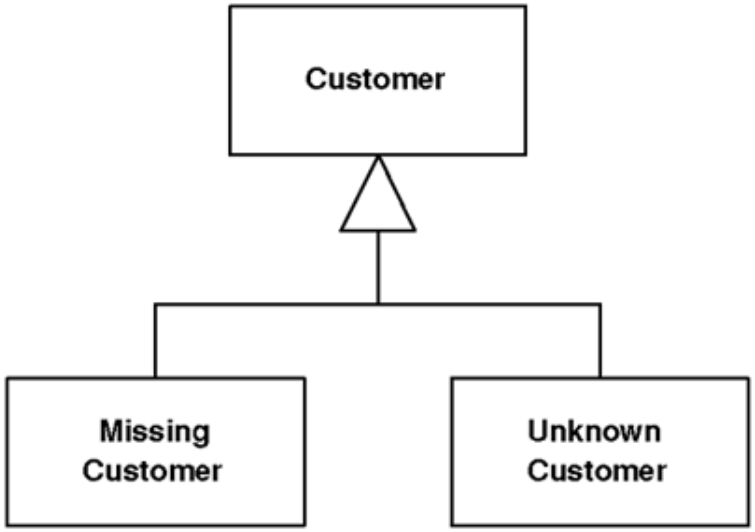
Integrate with other systems

- Requirements:
  - Should be integrated with other systems
  - Some of them are heterogeneous systems

- Web services:
  - Access via web
  - Access via SOAP
  - But how to test?

Special Case

- A subclass that provides special behavior for particular cases.



- If it's possible for a variable to be null, you have to remember to surround it with null test code so you'll do the right thing if a null is present. Often the right thing is same in many contexts, so you end up writing similar code in lots of places--committing the sin of code duplication.
- Instead of returning null, or some odd value, return a Special Case that has the same interface as what the caller expects.

Special Case-How It Works

- The basic idea is to create a subclass to handle the Special Case.
  - Thus, if you have a customer object and you want to avoid null checks, you make a null customer object.
  - Take all of the methods for customer and override them in the Special Case to provide some harmless behavior.
  - Then, whenever you have a null, put in an instance of null customer instead.
- There's usually no reason to distinguish between different instances of null customer, so you can often implement a Special Case with a flyweight .
- A null can mean different things.
  - A null customer may mean no customer or it may mean that there's a customer but we don't know who it is.
  - Rather than just using a null customer, consider having separate Special Cases for missing customer and unknown customer.
- A common way for a Special Case to override methods is to return another Special Case.
  - so if you ask an unknown customer for his last bill, you may well get an unknown bill.

Special Case-When to Use It

- Use Special Case whenever
  - you have multiple places in the system that have the same behavior after a conditional check for a particular class instance,
  - or the same behavior after a null check.

Special Case-Example

- Here's a simple example of Special Case used as a null object.
- We have a regular employee.

```
class Employee...
 public virtual String Name {
   get {return _name;}
   set {_name = value;}
 }
 private String _name;
 public virtual Decimal GrossToDate {
   get {return calculateGrossFromPeriod(0);}
 }
 public virtual Contract Contract {
   get {return _contract;}
 }
 private Contract _contract;
```

Special Case-Example

- The features of the class could be overridden by a null employee

```
class NullEmployee : Employee, INull...
  public override String Name {
   get {return "Null Employee";}
   set {}
  }
  public override Decimal GrossToDate {
    get {return 0m;}
  }
  public override Contract Contract {
    get {return Contract.NULL;}
  }
```

- Notice that when you ask a null employee for its contract you get a null contract back.

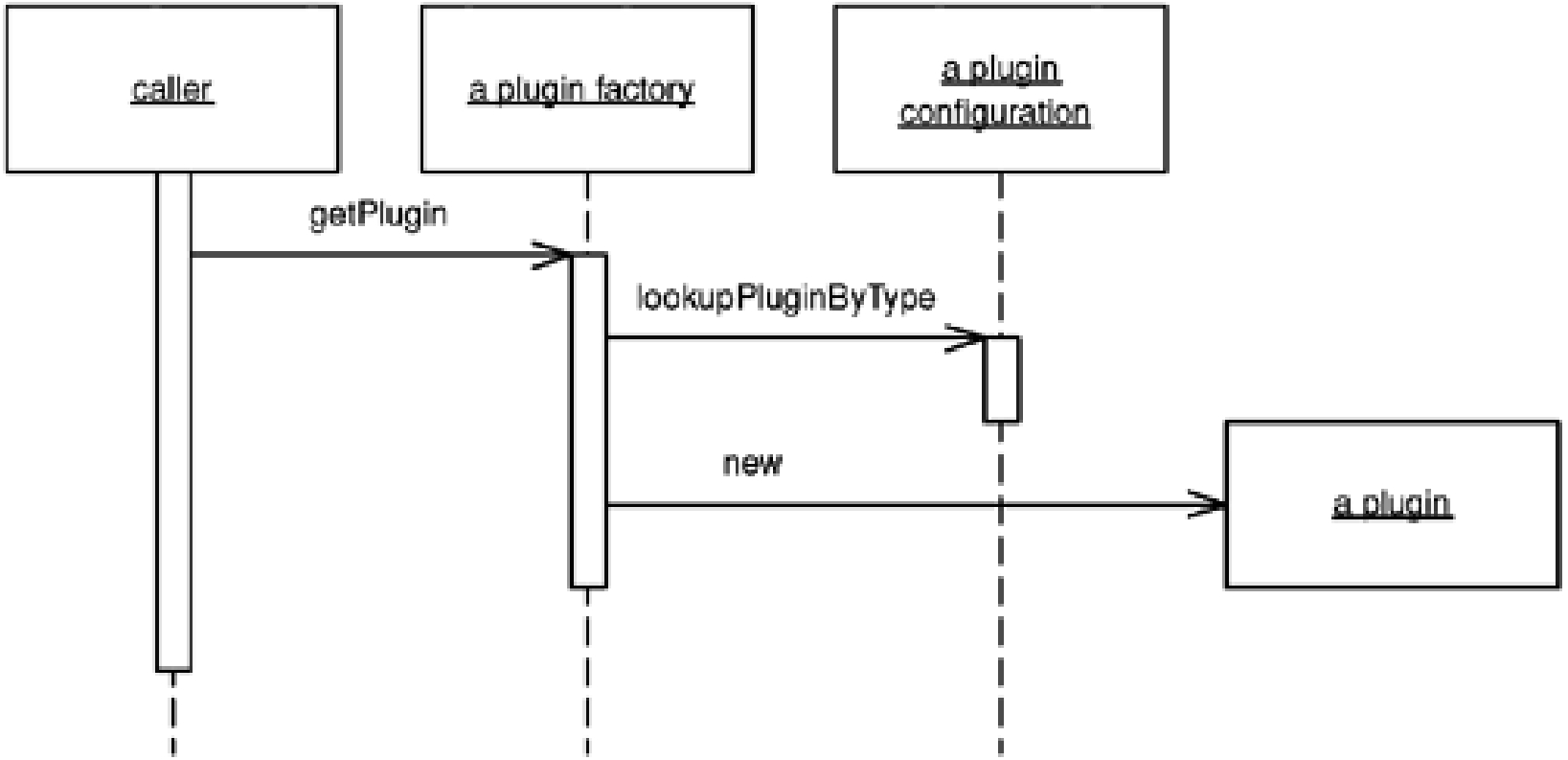
Plugin

- Links classes during configuration rather than compilation.

- Configuration shouldn't be scattered throughout your application, nor should it require a rebuild or redeployment. Plugin solves both problems by providing centralized, runtime configuration.

Plugin-How It Works

REin
REliable, INtelligent & Scalable Systems



Plugin-How It Works

- A text file works quite well as the means of stating linking rules.
  - The Plugin factory will simply read the text file, look for an entry specifying the implementation of a requested interface, and return that implementation.

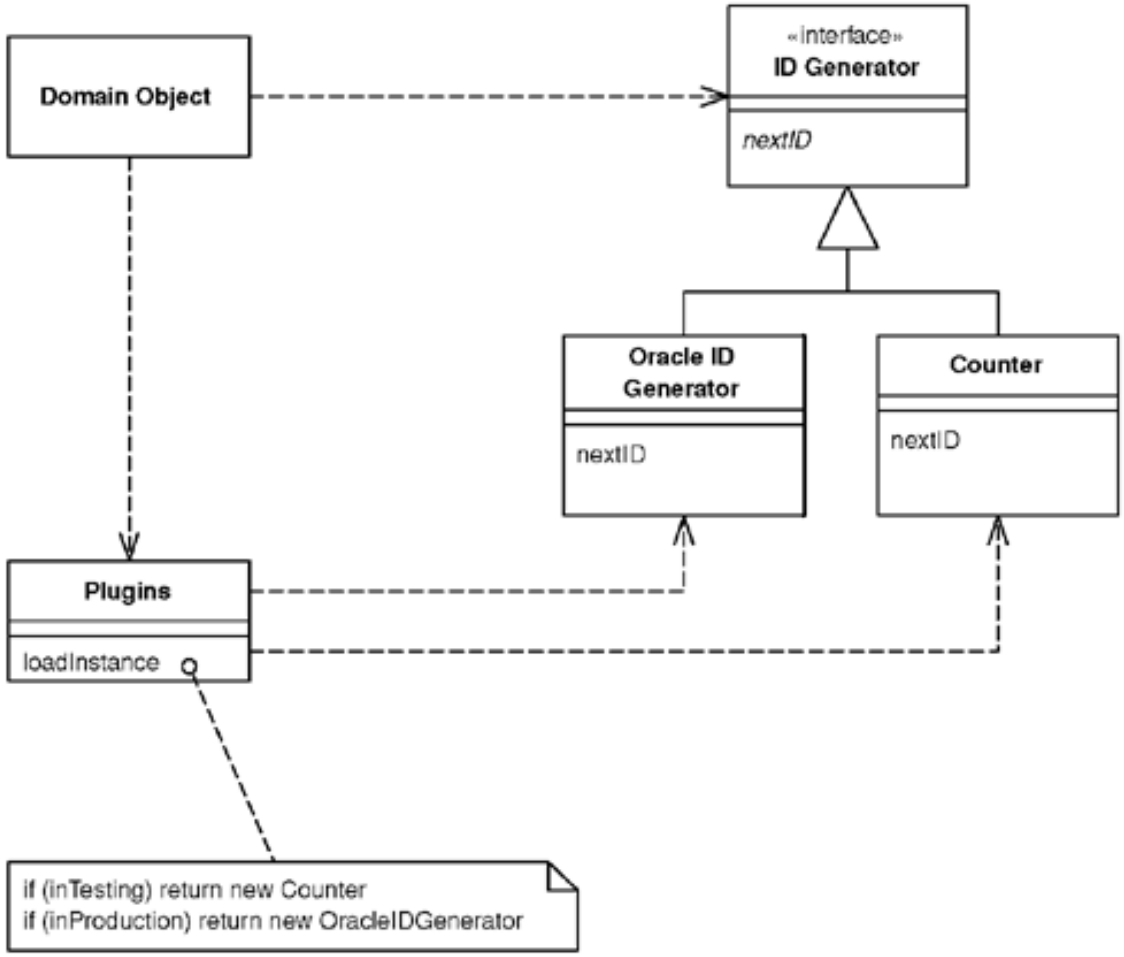
- Plugin works best in a language that supports reflection because the factory can construct implementations without compile-time dependencies on them.
  - When using reflection, the configuration file must contain mappings of interface names to implementation class names.
  - The factory can sit independently in a framework package and needn't be changed when you add new implementations to your configuration options.

Plugin-When to Use It

- Use Plugin whenever you have behaviors that require different implementations based on runtime environment.

Plugin-Example

- key, or ID, generation is a task whose implementation might vary between deployment environments.



Plugin-Example

- First we'll write the IdGenerator Separated Interface as well as any needed implementations.

```
interface IdGenerator...
   public Long nextId();

class OracleIdGenerator implements IdGenerator...
   public OracleIdGenerator() {
      this.sequence = Environment.getProperty("id.sequence");
      this.datasource = Environment.getProperty("id.source");
}
```

- In the OracleIdGenerator, nextId() select the next available number out of the defined sequence from the defined data source.

```
class Counter implements IdGenerator...
  private long count = 0;
  public synchronized Long nextId() {
    return new Long(count++);
  }
```

Plugin-Example

- Now that we have something to construct, let's write the plugin factory that will realize the current interface-to-implementation mappings.

```
class PluginFactory...
 private static Properties props = new Properties();
 static {
   try {
     props.load(PluginFactory.class.
        getResourceAsStream("/plugins.properties"));
   } catch (Exception ex) {
     throw new ExceptionInInitializerError(ex);
   }
 }

 public static Object getPlugin(Class iface) {
  String implName = props.getProperty(iface.getName());
  if (implName == null) {
    throw new RuntimeException("implementation not specified for " +
                                    iface.getName() + " in PluginFactory propeties.");
  }
  try {
   return Class.forName(implName).newInstance();
  } catch (Exception ex) {
   throw new RuntimeException("factory unable to construct instance of " + iface.getName());
  }
}
```

Plugin-Example

- Here's how two different configuration files, one for test and one for production, might look:

config file test.properties...

```
# test configuration
IdGenerator=TestIdGenerator
```

config file prod.properties...

```
# production configuration
IdGenerator=OracleIdGenerator
```

- Let's go back to the IdGenerator interface and add a static INSTANCE member that's set by a call to the Plugin factory. It combines Plugin with the singleton pattern to provide an extremely simple, readable call to obtain an ID.

class IdGenerator...

```
public static final IdGenerator INSTANCE =
  (IdGenerator) PluginFactory.getPlugin(IdGenerator.class);
```

Plugin-Example

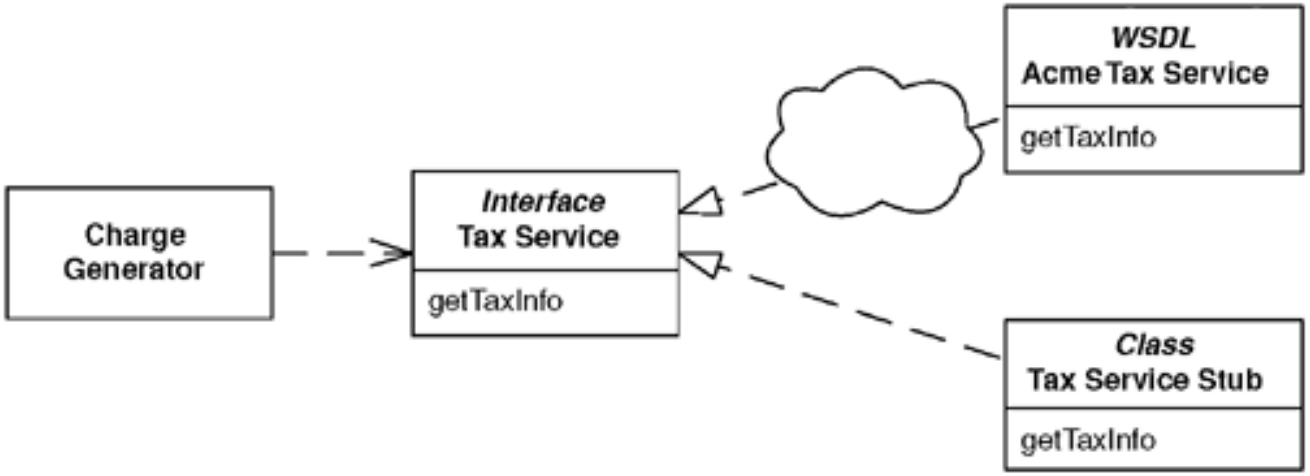
- We can now make that call knowing that we'll get the right ID for the right environment.

class Customer extends DomainObject...

```
private Customer(String name, Long id) {
 super(id);
 this.name = name;
}
public Customer create(String name) {
 Long newObjId = IdGenerator.INSTANCE.nextId();
 Customer obj = new Customer(name, newObjId);
 obj.markNew();
 return obj;
}
```

Service Stub

- Removes dependence upon problematic services during testing.



- Enterprise systems often depend on access to third-party services such as credit scoring, tax rate lookups, and pricing engines.
- Feature delivery is unpredictable, and as these services are often remote reliability and performance can suffer as well.
- Replacing the service during testing with a Service Stub that runs locally, fast, and in memory improves your development experience.

Service Stub-How It Works

- The first step is to define access to the service with a Gateway.
  - The Gateway should not be a class but rather a Separated Interface so you can have one implementation that calls the real service and at least one that's only a Service Stub.
- The desired implementation of the Gateway should be loaded using a Plugin.
  - The key to writing a Service Stub is that you keep it as simple as possible--Complexity will defeat your purpose.

Service Stub-When to Use It

REin
REliable, INtelligent & Scalable Systems

- Use Service Stub whenever you find that dependence on a particular service is hindering your development and testing.

- Many practitioners of Extreme Programming use the term Mock Object, for a Service Stub. We've stuck with Service Stub because it's been around longer.

Service Stub-Example

- Our application uses a tax service that deployed as a Web service.

- The first item we'll take care of is defining a Gateway so that our domain code isn't forced to deal with the wonders of Web services.
  - The Gateway is defined as an interface to facilitate loading of any Service Stubs that we write. We'll use Plugin to load the correct tax service implementation.

```
interface TaxService...
  public static final TaxService INSTANCE = (TaxService)
        PluginFactory.getPlugin(TaxService.class);
  public TaxInfo getSalesTaxInfo(String productCode, Address addr,
        Money saleAmount);
```

Service Stub-Example

- The simple flat rate Service Stub would look like this:

```
class FlatRateTaxService implements TaxService...
  private static final BigDecimal FLAT_RATE = new BigDecimal("0.0500");
  public TaxInfo getSalesTaxInfo(String productCode, Address addr, Money
        saleAmount) {
    return new TaxInfo(FLAT_RATE,
            saleAmount.multiply(FLAT_RATE));
}
```

Service Stub-Example

- Here's a Service Stub that provides tax exemptions for a particular address and product combination:

```
class ExemptProductTaxService implements TaxService...
  private static final BigDecimal EXEMPT_RATE = new BigDecimal("0.0000");
  private static final BigDecimal FLAT_RATE = new BigDecimal("0.0500");
  private static final String EXEMPT_STATE = "IL";
  private static final String EXEMPT_PRODUCT = "12300";
  public TaxInfo getSalesTaxInfo(String productCode, Address addr, Money
      saleAmount) {
    if (productCode.equals(EXEMPT_PRODUCT) &&
        addr.getStateCode().equals(EXEMPT_STATE)) {
      return new TaxInfo(EXEMPT_RATE,
          saleAmount.multiply(EXEMPT_RATE));
    } else {
      return new TaxInfo(FLAT_RATE,
          saleAmount.multiply(FLAT_RATE));
    }
}
```

Service Stub-Example

- Now here's a more dynamic Service Stub whose methods allow a test case to add and reset exemption combinations.

```
class TestTaxService implements TaxService...
 private static Set exemptions = new HashSet();
 public TaxInfo getSalesTaxInfo(String productCode, Address addr, Money saleAmount) {
    BigDecimal rate = getRate(productCode, addr);
    return new TaxInfo(rate, saleAmount.multiply(rate));
 }
 public static void addExemption(String productCode, String stateCode) {
    exemptions.add(getExemptionKey(productCode, stateCode));
 }
 public static void reset() { exemptions.clear(); }
 private static BigDecimal getRate(String productCode, Address addr) {
    if (exemptions.contains(getExemptionKey(productCode, addr.getStateCode())))
     { return EXEMPT_RATE; }
    else { return FLAT_RATE; }
 }
```

Service Stub-Example

- Finally, any caller to the tax service must access the service via the Gateway. We have a charge generator here that creates standard charges and then calls the tax service in order to create any corresponding taxes.

```
class ChargeGenerator...
 public Charge[] calculateCharges(BillingSchedule schedule) {
  List charges = new ArrayList();
  Charge baseCharge = new Charge(schedule.getBillingAmount(), false);
  charges.add(baseCharge);
  TaxInfo info =
        TaxService.INSTANCE.getSalesTaxInfo( schedule.getProduct(),
        schedule.getAddress(), schedule.getBillingAmount());
  if (info.getStateRate().compareTo(new BigDecimal(0)) > 0) {
    Charge taxCharge = new Charge(info.getStateAmount(), true);
    charges.add(taxCharge);
  }
  return (Charge[]) charges.toArray(new Charge[charges.size()]);
}
```

References

- Martin Fowler's Patterns of Enterprise Application Architecture

Thank You!