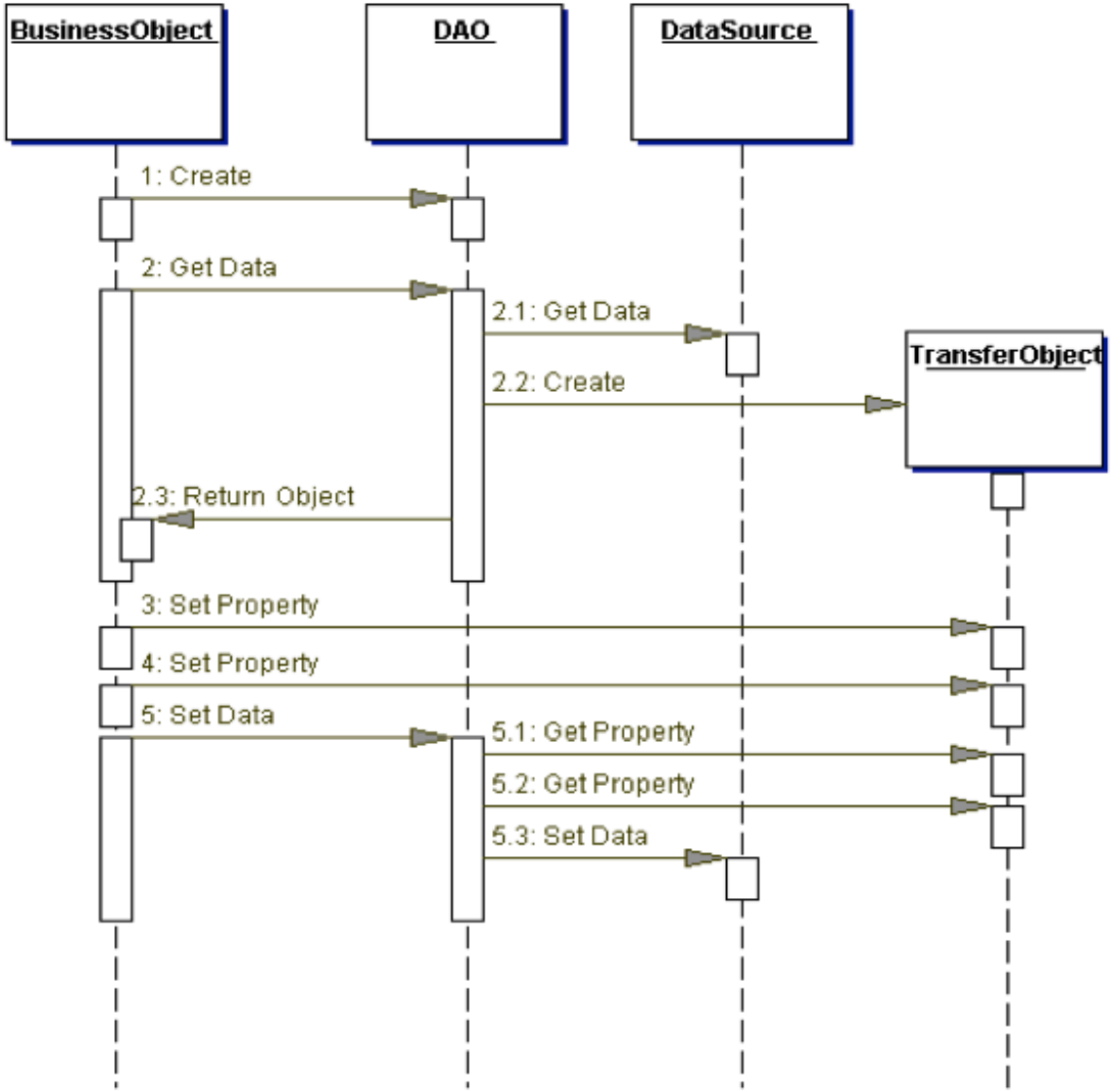
DAO – Data Access Object



DTO – Data Transfer Object

also known as **Value Object** or **VO**,

used to transfer data between software application subsystems.

DTO's are often used in conjunction with DAOs to retrieve data from a database.

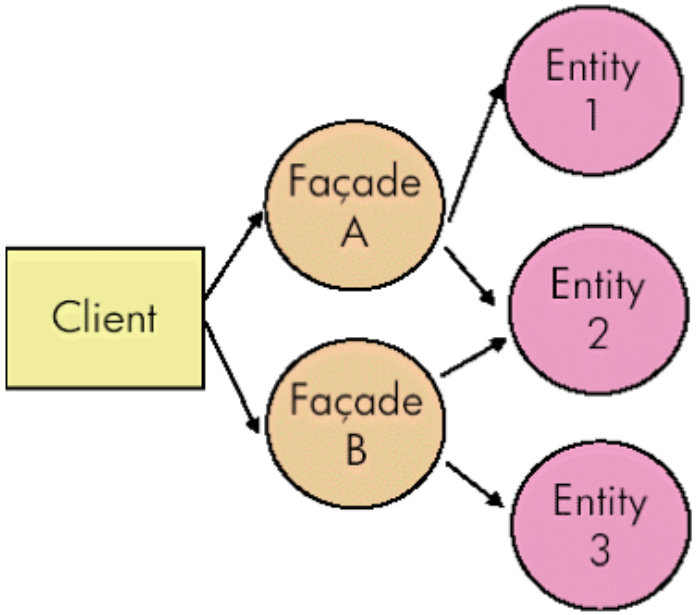
DTOs do not have any behaviour except for storage and retrieval of its own data (mutators and accessor).

Session Facade

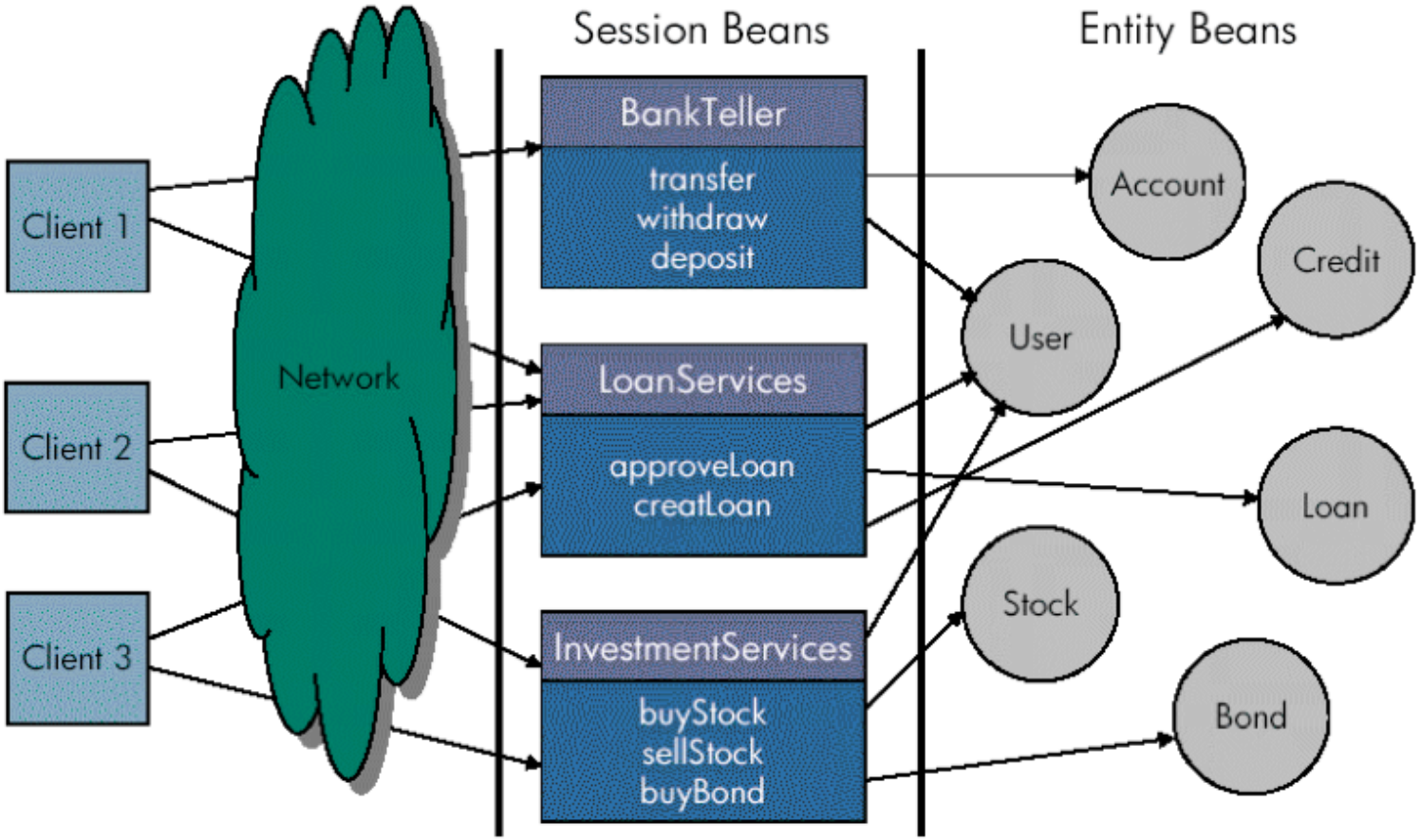
Uses a session bean to encapsulate the complexity of interactions between the business objects participating in a workflow.

Manages the business objects, and provides a uniform coarse-grained servic

access layer to clients



Mapping Session Facade on use cases



Business Delegate Pattern

Use a BusinessDelegate to

– Reduce coupling between presentation-tier and business service components

– Hide the underlying implementation details of the business service components

– Cache references to business services components
– Cache data
– Translate low level exceptions to application level exceptions – Transparently retry failed transactions
– Can create dummy data for clients

Business Delegate is a plain java class

Service Locator

Have an object that knows how to get hold of all of the services that an application might need.

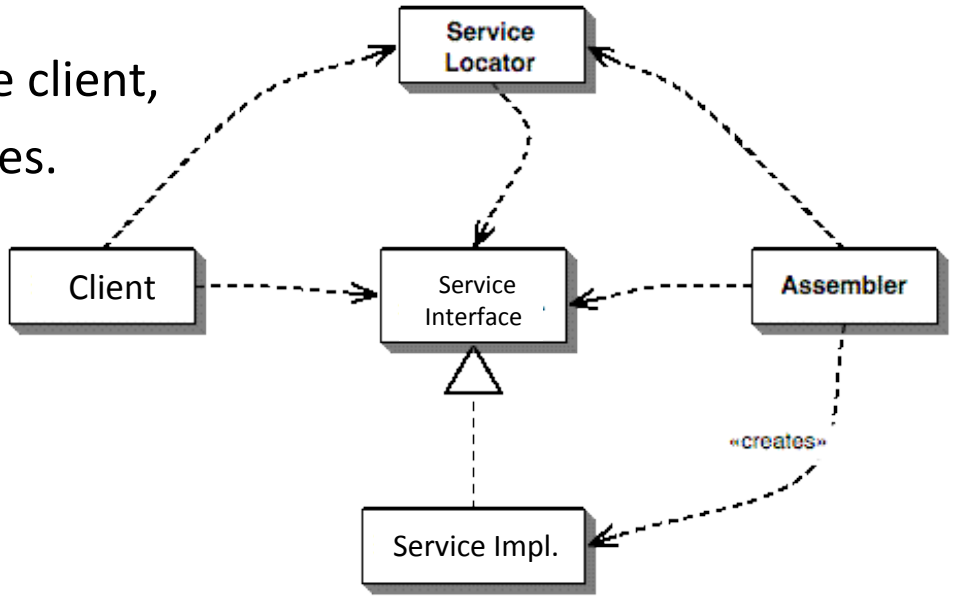
A service locator has a method that, given a key value, returns the implementation of a service when one is needed.

Of course this just shifts the burden:

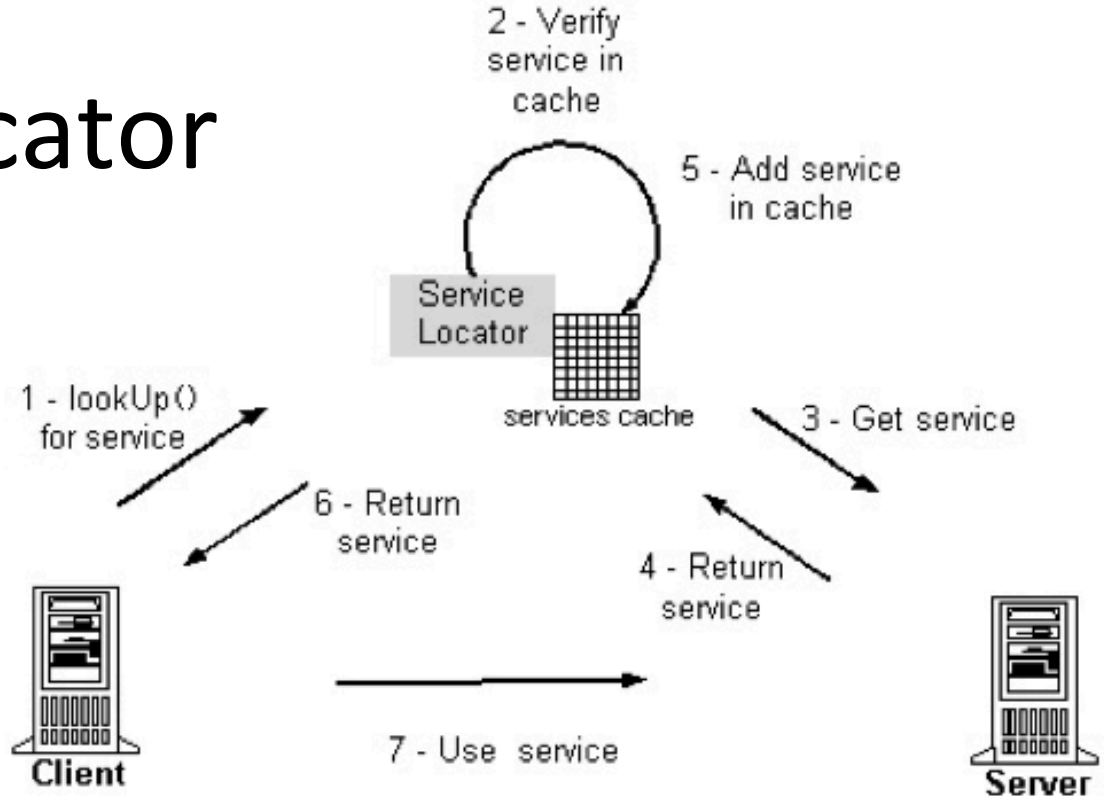
we still have to get the locator into the client,

but this scales well for multiple services.

Example: the rmi registry



Service Locator



Use a ServiceLocator to

– Abstract naming service usage

– Shield complexity of service lookup and creation

– Promote reuse

– Enable optimize service lookup and creation functions

• Usually called within BusinessDelegate or Session Facade object

Service Locator

```
package ...; import ...;
public class ServiceLocator throws Exception {
   private static ServiceLocator serviceLocator;
   private static Context context;
   private ServiceLocator() { context =  getInitialContext(); }
   private Context getInitialContext(){
      Hashtable environment = new Hashtable();
      environment.put(..);
      return new InitialContext(environment);
   } public static synchronized ServiceLocator getInstance(){
     if (serviceLocator == null) {
         serviceLocator = new ServiceLocator(); }
     return serviceLocator;
   }
   public Object getBean(…) {return context.lookup(…)}
}
```

Overall view