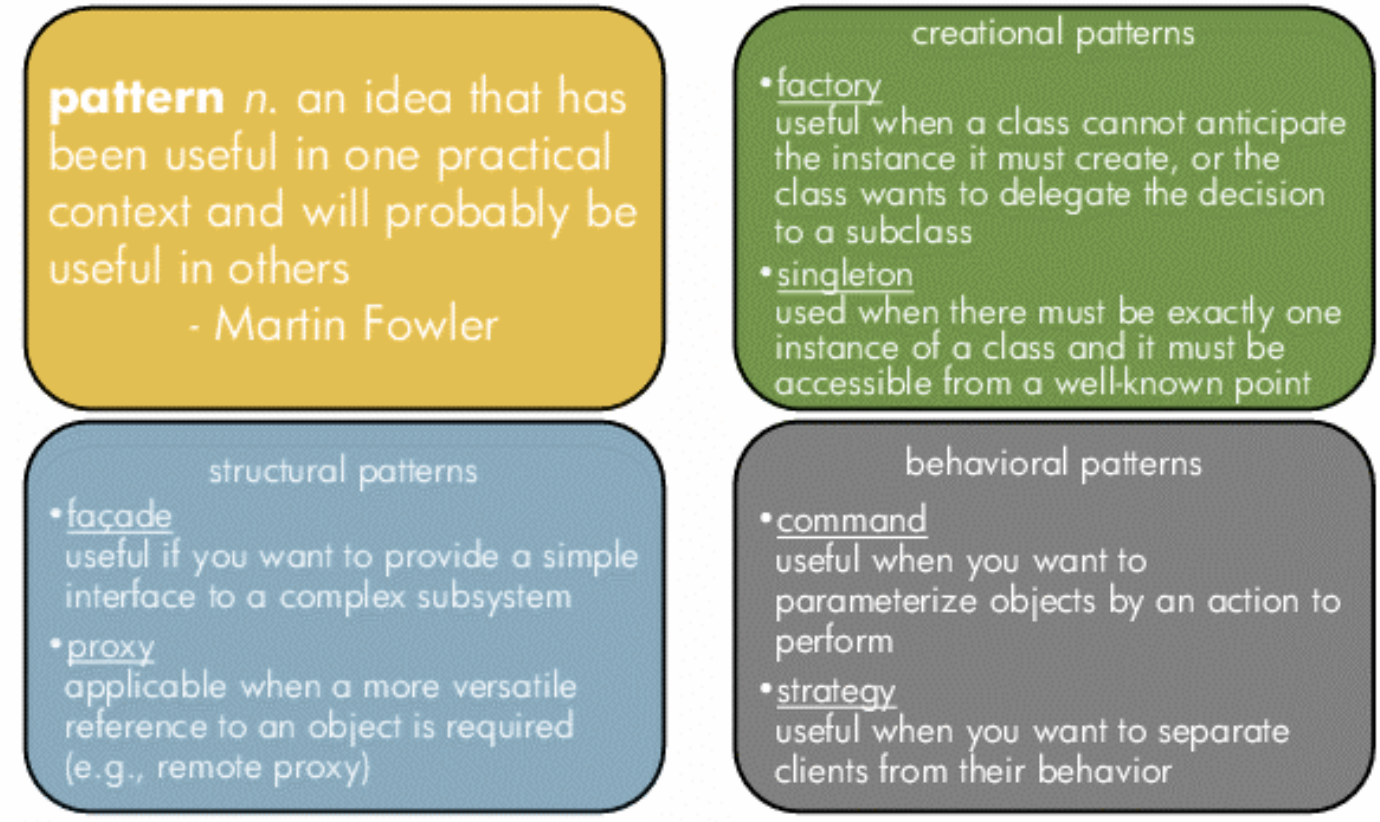
EJB Patterns

Architectural Patterns



What is a “pattern”?

“The best solution to a recurring problem”

Recurring software design problems

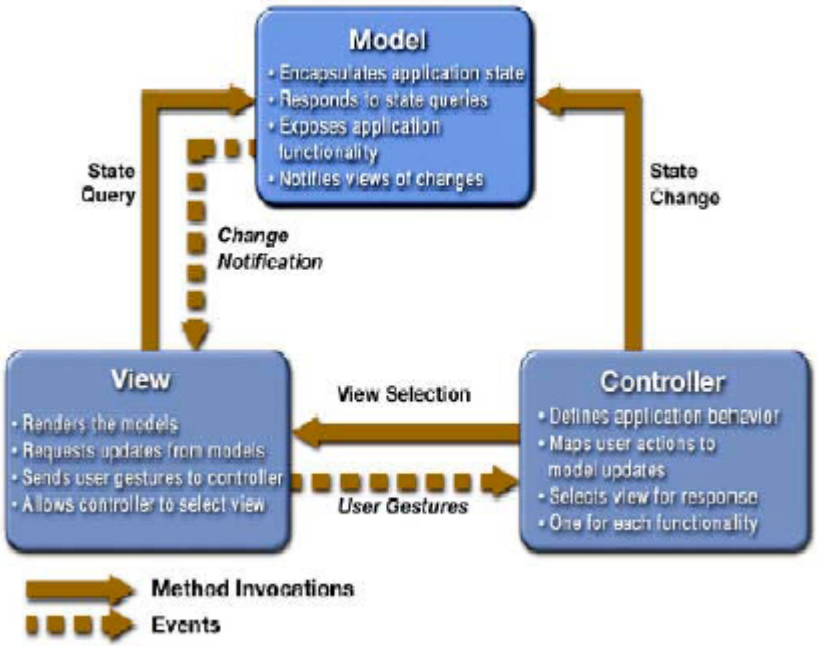
identified and catalogued in a standard way ao as to be accessibile to everybody and usable in any programming language.

What is a J2EE pattern?

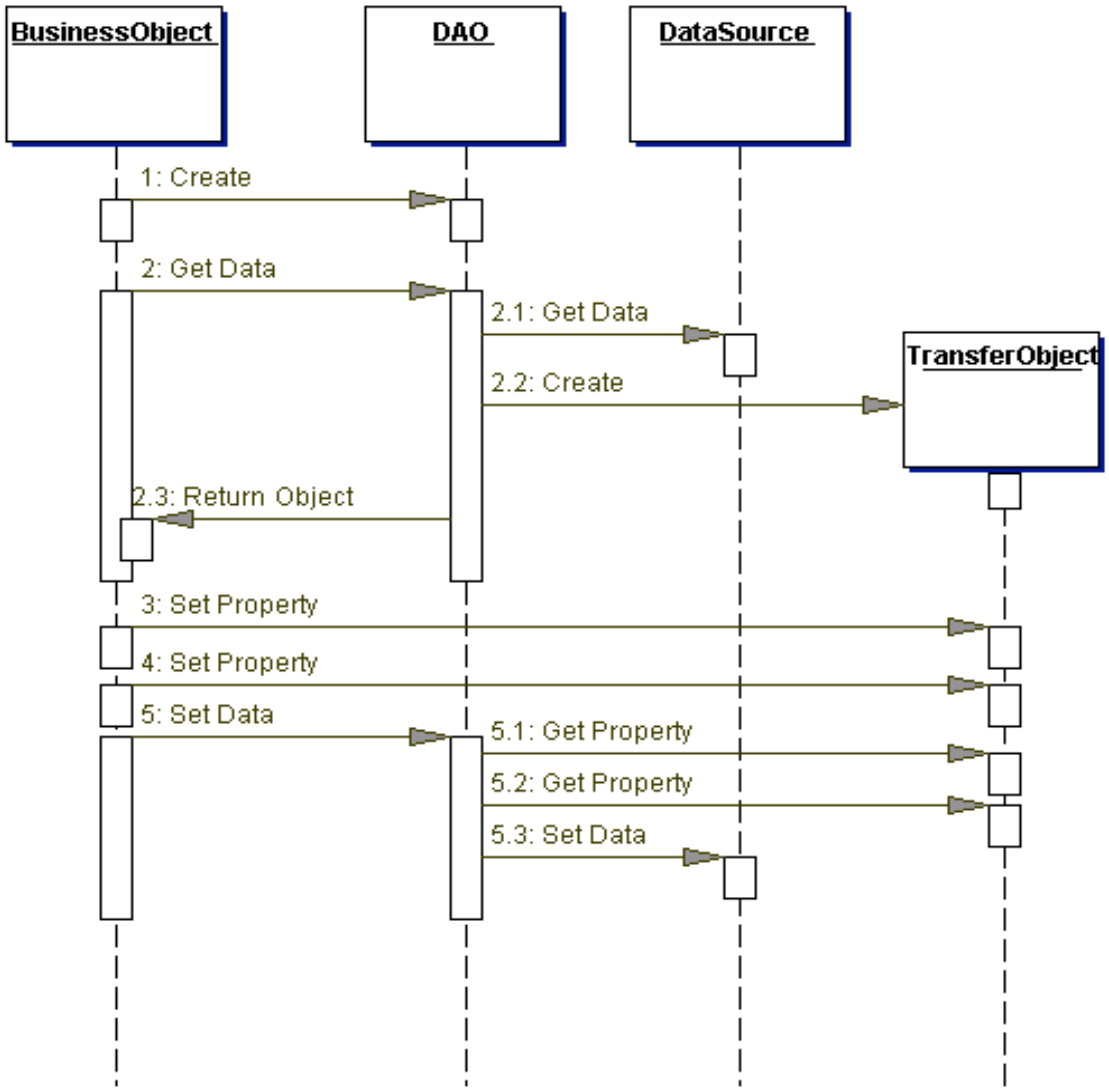
- A J2EE design pattern is essentially any pattern that utilizes J2EE technology to solve a recurring problem.
- Patterns are typically classified by logical tier:
  - presentation tier
  - business tier
  - integration (EIS) tier

Architecture decisions
Use MVC architecture

- Clear separation of design concerns
- Easier maintenance
  - decreased code duplication
  - better isolation and less interdependence
- Easy to add new client types or adjust to different client requirements
- A common approach in J2EE
  - **Model** = EJBs
  - **View** = JSPs
  - **Controller** = servlet



DAO Pattern
Data Access Object



Data Transfer Objects (**DTO**)

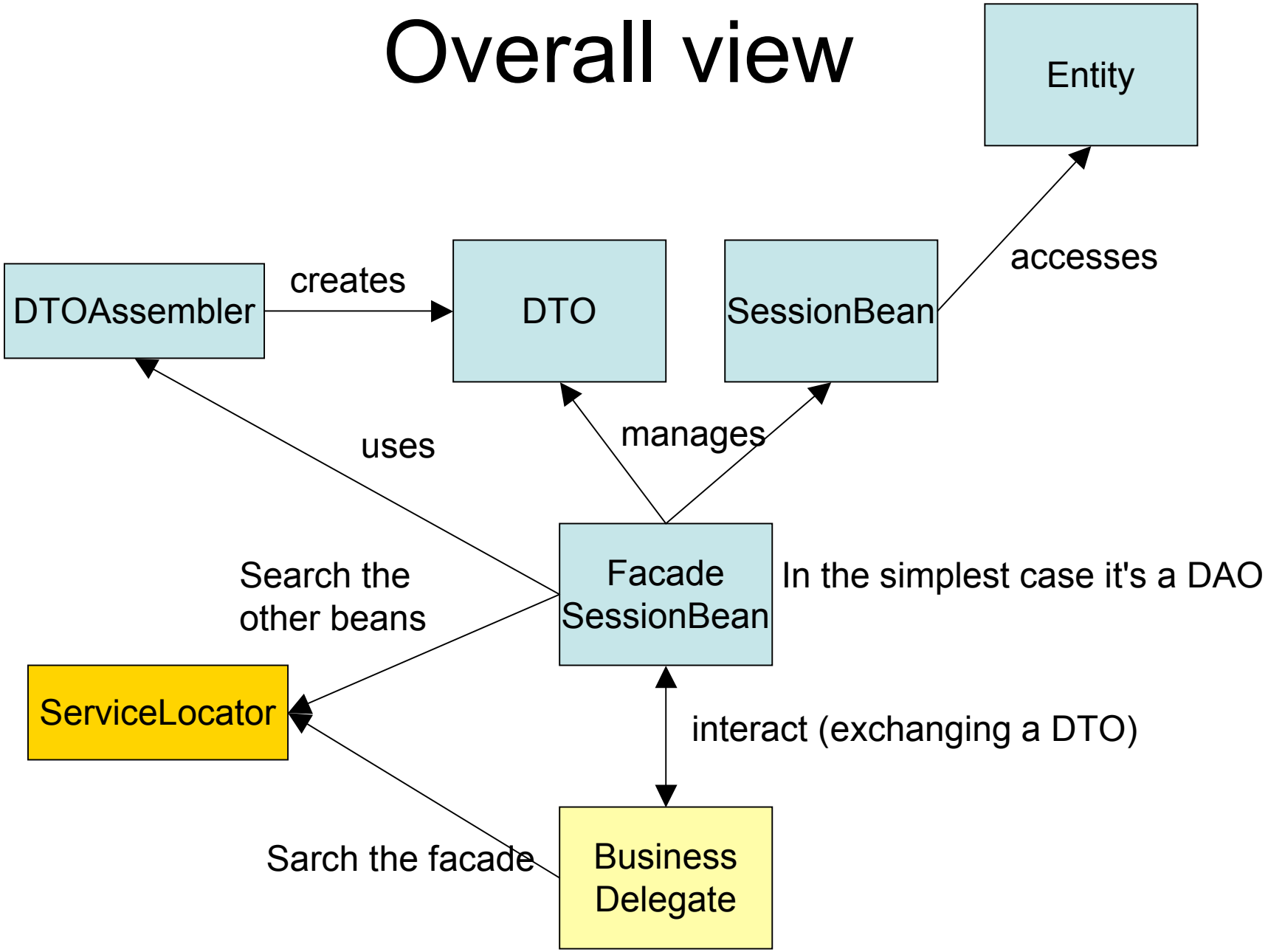
also known as **Value Objects**or **VO**,

used to transfer data between software application subsystems.

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

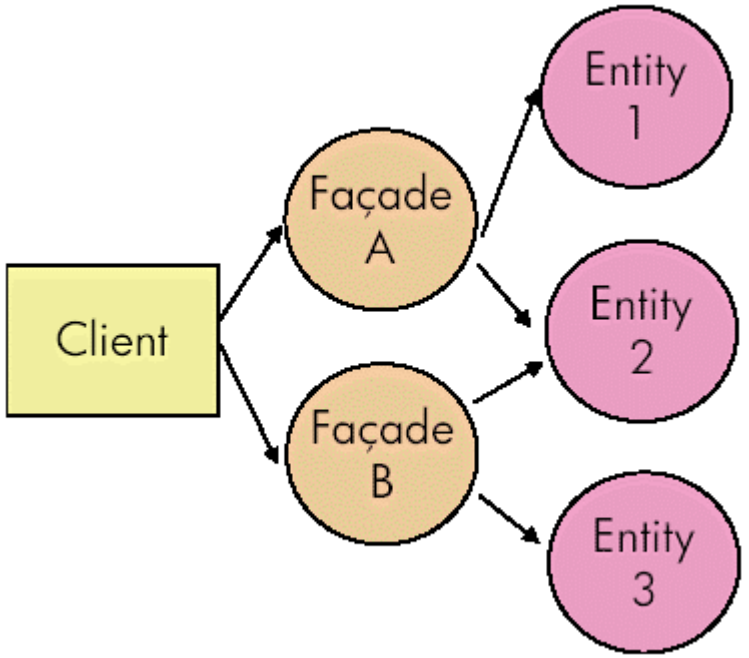
DTOs do not have any behaviour except for storage and retrieval of its own data (mutators and accessor).

Overall view



The session facade Pattern

- 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 service
  access layer to clients

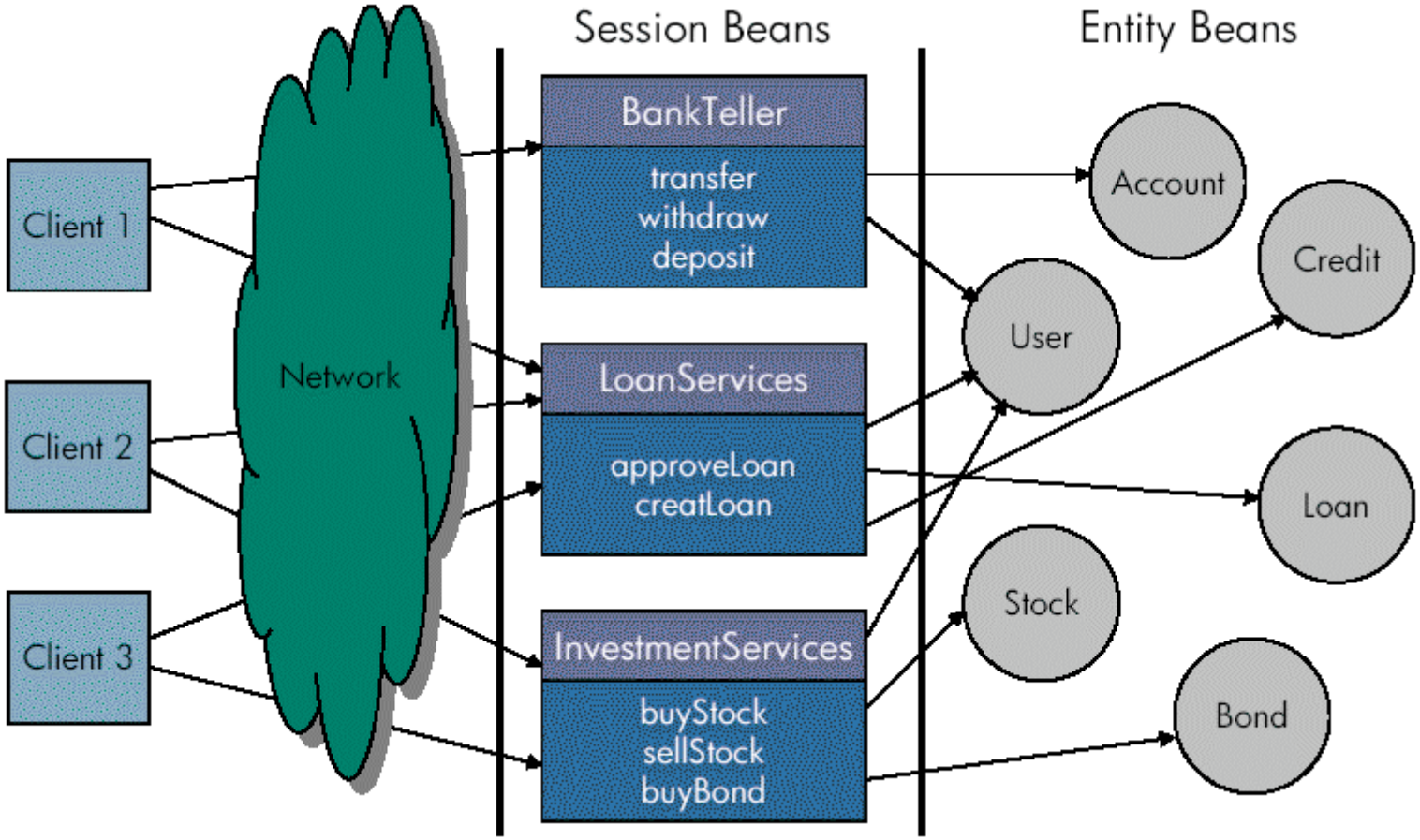


Business Delegate Pattern

- Use a Business Delegate 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

Mapping Session Facade on use cases



Singleton Pattern

```
public class MySingleton {
    private static  MySingleton _instance;

    private MySingleton() {
    // construct object . . .
    }

    // For lazy initialization
    public static synchronized MySingleton getInstance() {
    if (_instance==null) {
    _instance = new MySingleton();
    }
    return _instance;
    }
    // Remainder of  class definition . . .
    }
```

Cache to itself

if the cache is empty
it creates an istance

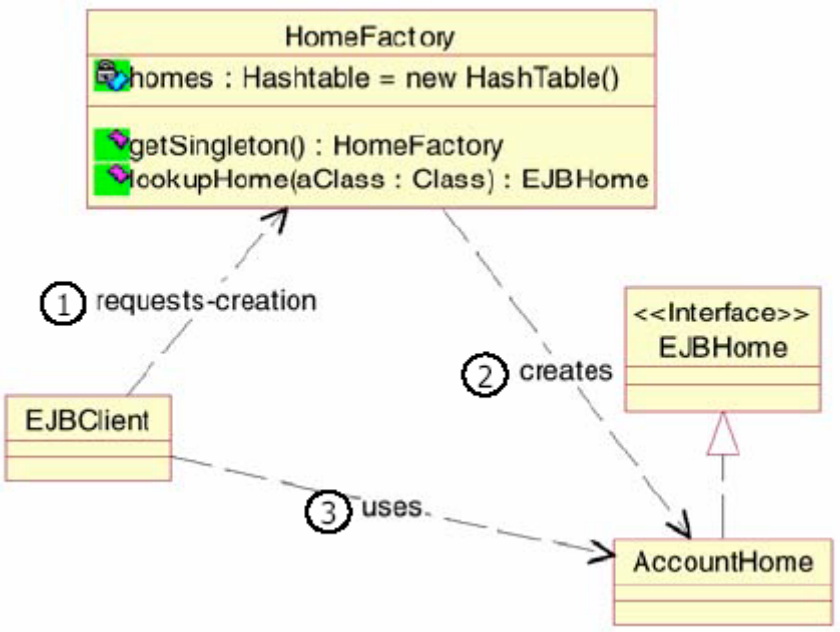
Home Factory Example

```
public final class MyBeanHomeFactory {
  private static MyBeanHomeFactory factory = null;
  private static MyBeanHome handle = null;
  public final MyBeanHome getHome() {
     return handle;   }
  public static MyBeanHomeFactory getSingleton() {
     if (factory == null) {factory = new CoursesEJB(); }
     return factory ;
  }
  private MyBeanHomeFactory () {
    try {
      Object result = DirectoryAccess.lookup(" MyBean");
      MyBeanHome handle = (MyBeanHome)
      PortableRemoteObject.narrow(result, MyBeanHome.class);
    }
    catch (RemoteException ex1) { ex1.printStackTrace(); }
    catch (CreateException ex1) { ex1.printStackTrace();  }
  }
}
```

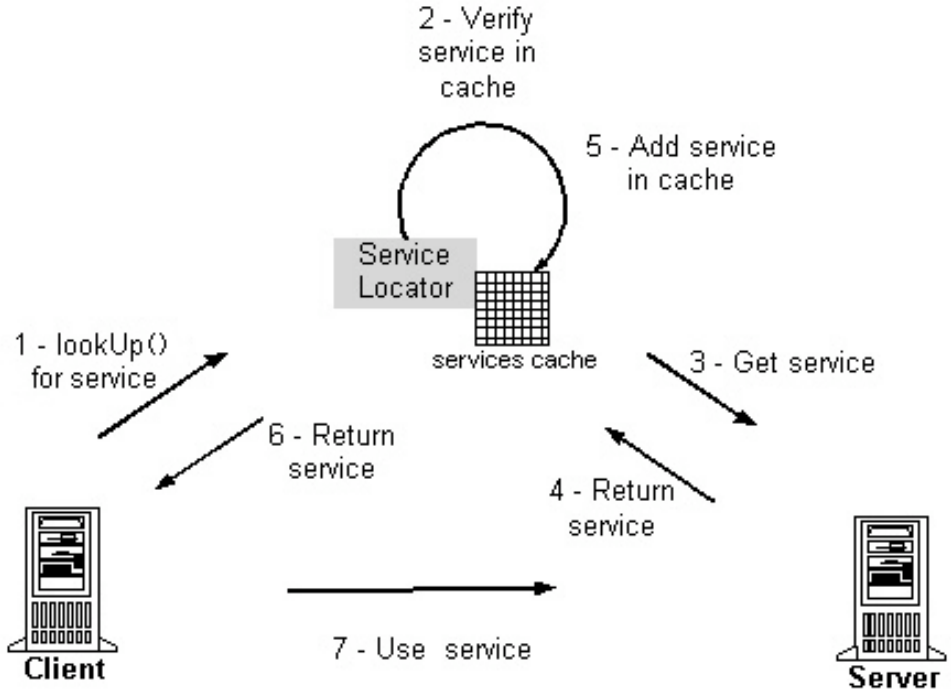
```
MyBeanHome mbh=mBeanHomeFactory.getSingleton().getHome();
```

The Home Factory Pattern

- Insulates clients from the naming service caches lookup for better performance
- Can be used for EJB home and JNDI lookups
- Uses the singleton pattern to ensure only one instance of the factory class is created



Service Locator Pattern



…to acquire an EJB Home object for the first time, Service Locator first creates a JNDI initial context object and performs a lookup on the EJB Home object.

But not only for EJB Home also for other kinds of services…an enhanced Service Locator can have a verifier mechanism that checks the cached services' validity by monitoring them during a cycle of a specified frequency.

Service Locator Pattern

- Use a Service Locator to
  - Abstract naming service usage
  - Shield complexity of service lookup and creation
  - Promote reuse
  - Enable optimize service lookup and creation functions
- Usually called within Business Delegate or Session Facade object

Service Locator Pattern

```
package ...;
import ...;

public class ServiceLocator {
  private static ServiceLocator serviceLocator;
  private static Context context;

  protected ServiceLocator() throws Exception {
     context = getInitialContext();
  }

 public static synchronized ServiceLocator getInstance() throws Exception {
    if (serviceLocator == null) {
      serviceLocator = new ServiceLocator();
    }
    return serviceLocator;
  }
```

Service Locator Pattern

```
  private Context getInitialContext() throws NamingException {
    Hashtable environment = new Hashtable();
    environment.put(Context.INITIAL_CONTEXT_FACTORY,
      "org.jnp.interfaces.NamingContextFactory");
    environment.put(Context.URL_PKG_PREFIXES,
      "org.jboss.naming:org.jnp.interfaces");
    environment.put(Context.PROVIDER_URL, "jnp://localhost:1099");
    return new InitialContext(environment);
  }
}
```

Service Locator Pattern

```
public static EJBHome getEjbHome(String ejbName, Class ejbClass)
      throws Exception {
      Object object = context.lookup(ejbName);
      EJBHome ejbHome = null;
      ejbHome = (EJBHome) PortableRemoteObject.narrow(object,ejbClass);
      if (ejbHome == null) {  throw new Exception(
                        "Could not get home for " + ejbName);
      }
      return ejbHome;
    }
}

public static EJBLocalHome getEjbLocalHome(String ejbName)
throws Exception {
    ...
  }
```

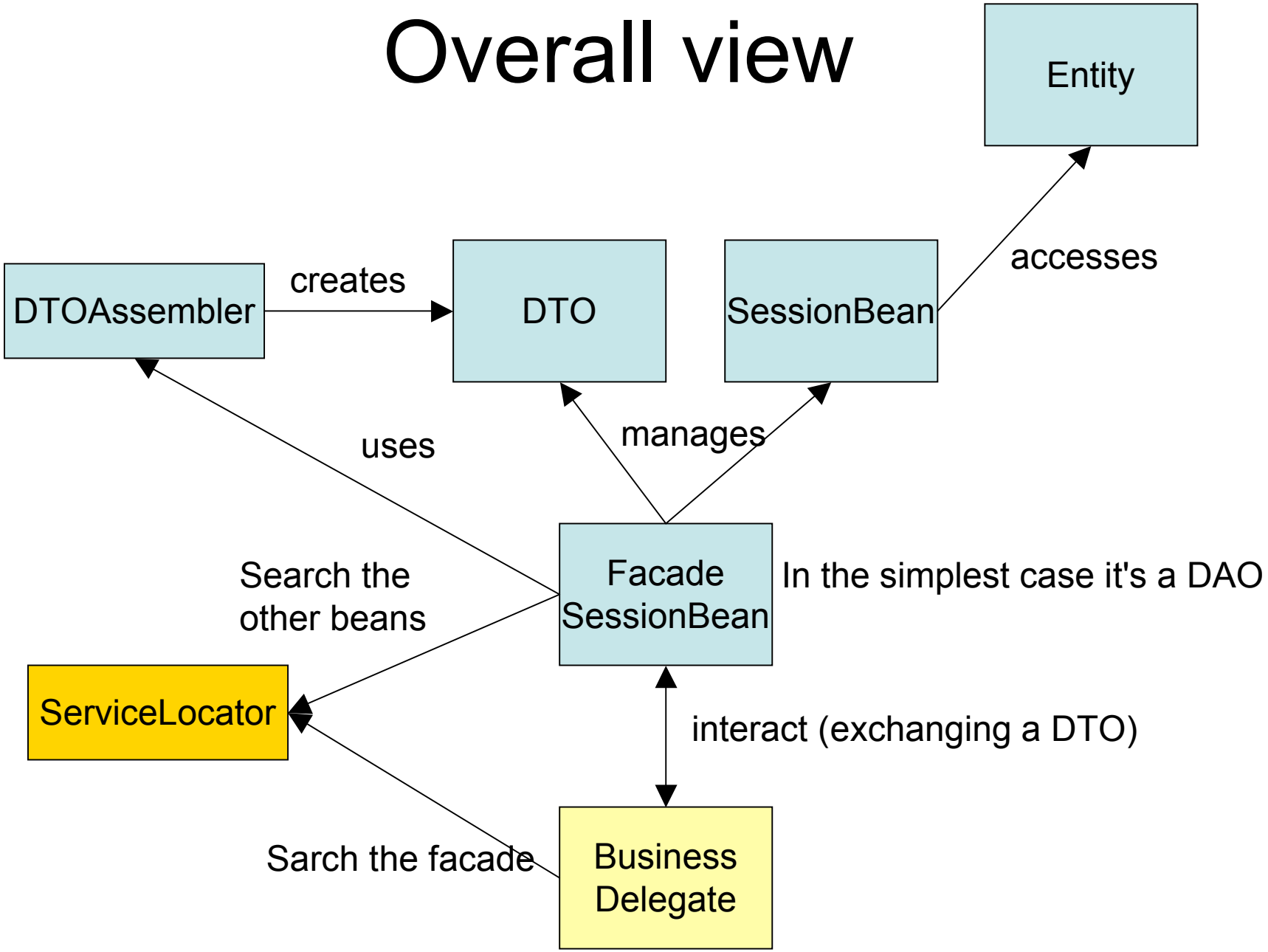
Using the Service Locator

```
private void findEmployeeHome() throws EJBException {
   final String ENTITY_NAME = "java:comp/env/ejb/employee";
   if (employeeHome == null) {
     try {
       ServiceLocator locator = ServiceLocator.getInstance();
       employeeHome = (EmployeeHome)
       locator.getEjbLocalHome(ENTITY_NAME);
     }
     catch (Exception e) {
       ...
     }
   }
 }
```

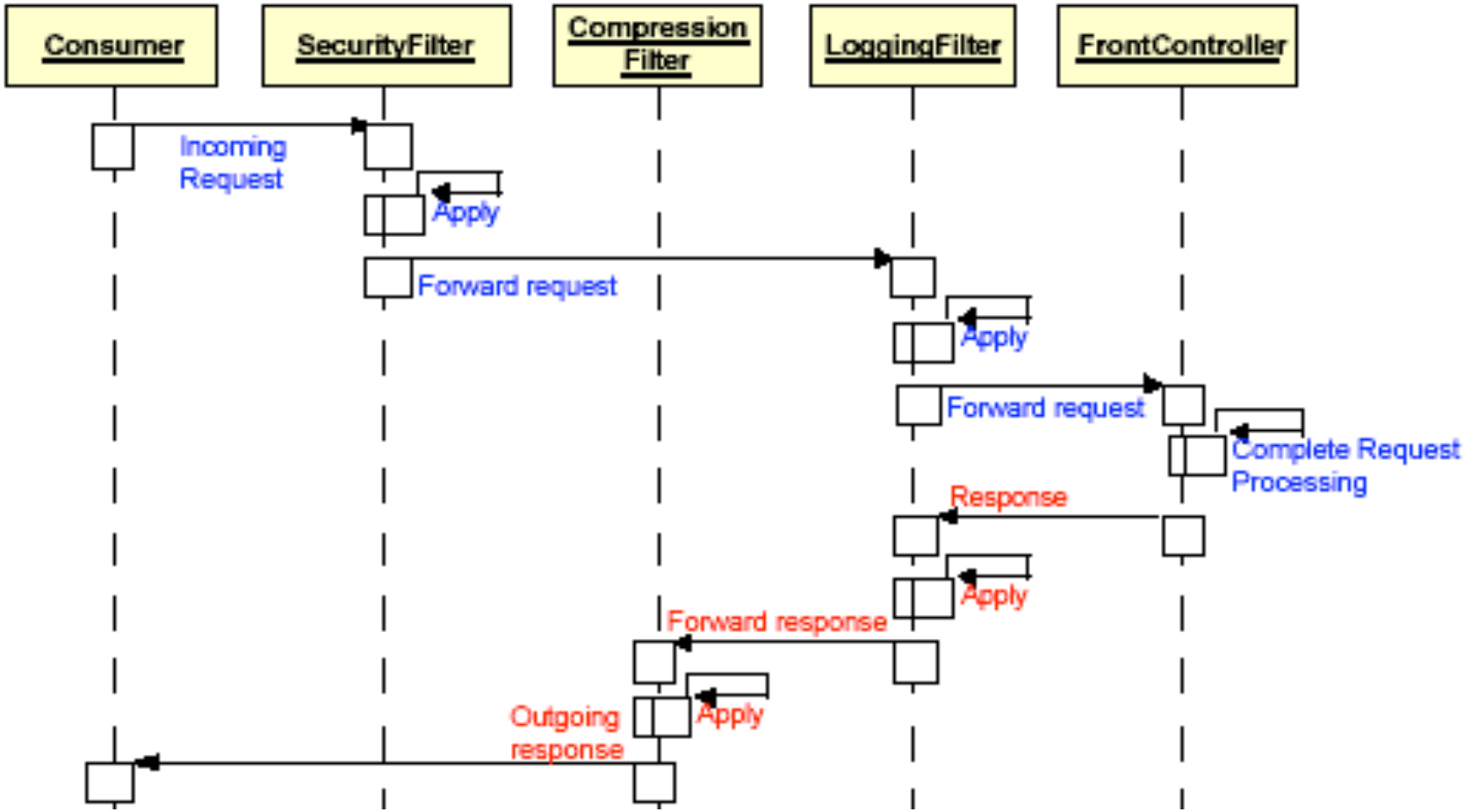
Other Patterns

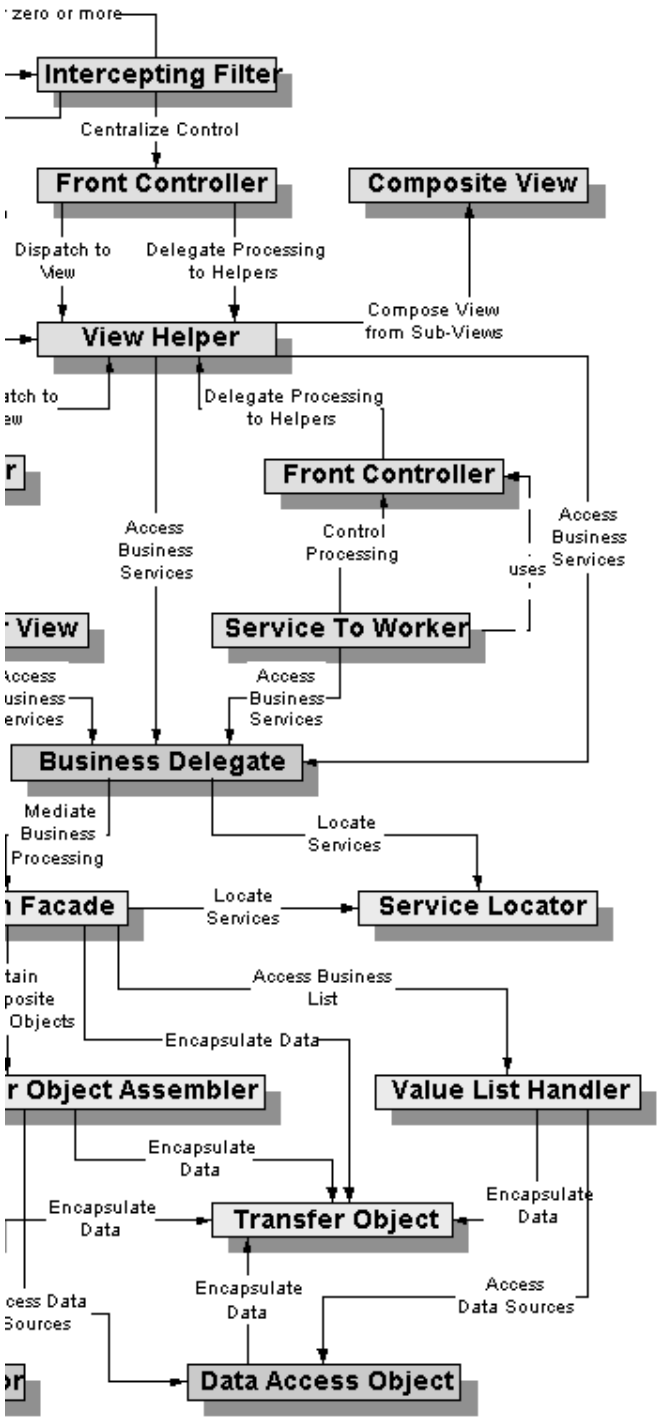
- – Version Numbering
- – Sequence Blocks

Overall view



Intercepting Filter Pattern





http://java.sun.com/blueprints/corej2eepatterns/Patterns/

Conclusion

" There are many decisions that must be made for a successful J2EE implementation "

Key patterns for J2EE:

- session façade
- service locator
- value object / data transfer object
- data access object

Use a combination of time tested patterns and best practices