

# J2EE en Jini 'pluggable' gemaakt middels Aspect Oriented Programming

## aEOS case study

**Jeroen Borgers**

Nedap IDEAS security management R&D

## Inhoud van de presentatie

- **Introductie**
  - Spreker, Nedap, aEOS
- **Probleemstelling**
- **Stappen naar eenvoudiger EJB**
- **AOP toegepast**
- **Integratie EJB, JNDI-services en Jini**
- **Conclusies**

## Introductie spreker

- **1994: Afgestudeerd op OO technieken in rekenintensieve natuurkunde**
- **1996-2002: werkzaam bij Atos Origin, in diverse Java projecten bij klanten en trekker van de Java kennis groep**
- **2002-heden: werkzaam bij Nedap, aan het aEOS toegangscontrolesysteem**
- **Expertises: Java/J2EE, Jini, performance, reduceren van complexiteit.**

## Introductie Nedap

- **Nederlandse apparaten fabriek levert diverse producten aan diverse markten, typisch bestaand uit een combinatie van elektronica en software.**
- **Centraal staat identificatie en registratie, bijv.:**
  - Stemmachines
  - Anti winkeldiefstalsysteem (detectiepoortjes)
  - Toegangscontrolesysteem aEOS
  - Thuiszorg tijdsregistratie
  - Koeien identificatie
  - Spijbel chip.



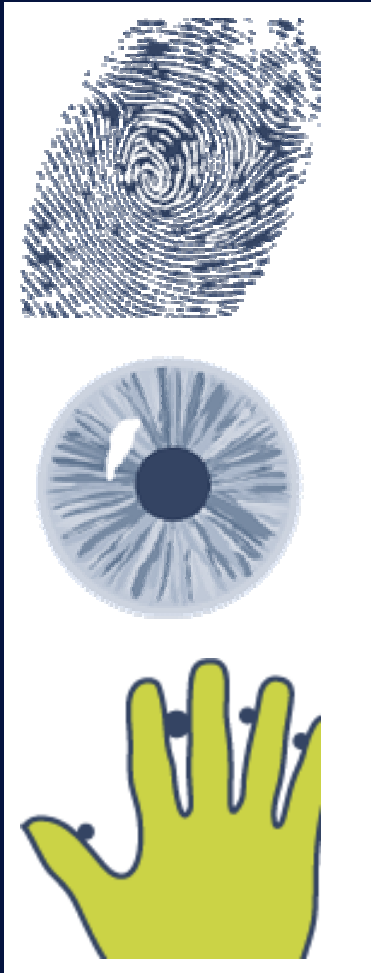
## Introductie aEOS

- **Toegangscontrole en beveiligingssysteem voor gebouwen, raffinaderijen, luchthavens, parkeergarages, etc.**
- **Basisfunctie: drager van de pas mag door een deur of niet**
- **Internet technologie: gebruik van bestaand IP netwerk dus veel minder kabels**
- **Gedistribueerd: 1 server en veel controllers (aEPU's)**
- **Functioneert ook bij afwezigheid server & netwerk**
- **Flexibiliteit**

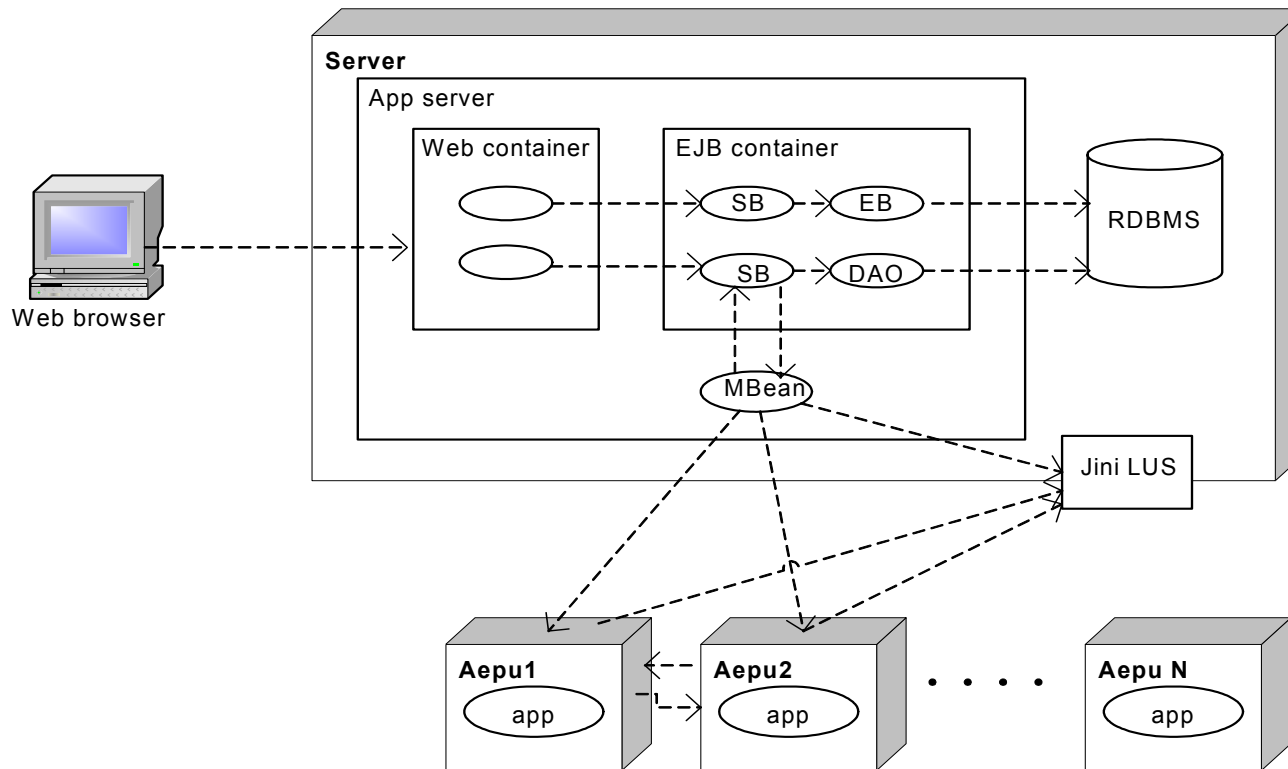


## Flexibiliteit van aEOS

- **Locatie van front-end gebruiker (browser)**
- **Middel van identificatie en verificatie (RFID pas, streepjescode, paspoort, vinger, iris, gezicht,..)**
- **Type autorisatie (tijd, tellen,..)**
- **Drager: werknemer, bezoeker, auto, etc.**
- **Inputs: sensor, antenne,.. en outputs: deur, tourniquet, slagboom, ..**
- **Configuratie in software, integratie met alarmcentrales, video beveiliging etc.**

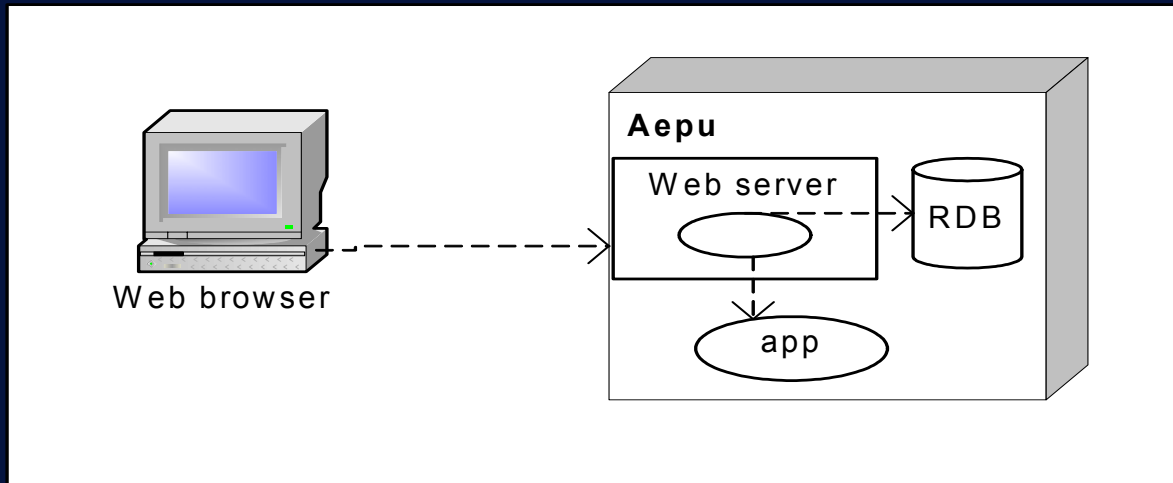


# aEOS architectuur



## Nieuw product: aEOS stand-alone

- **J2ME omgeving: geen app server; web server & DB op aEPU**

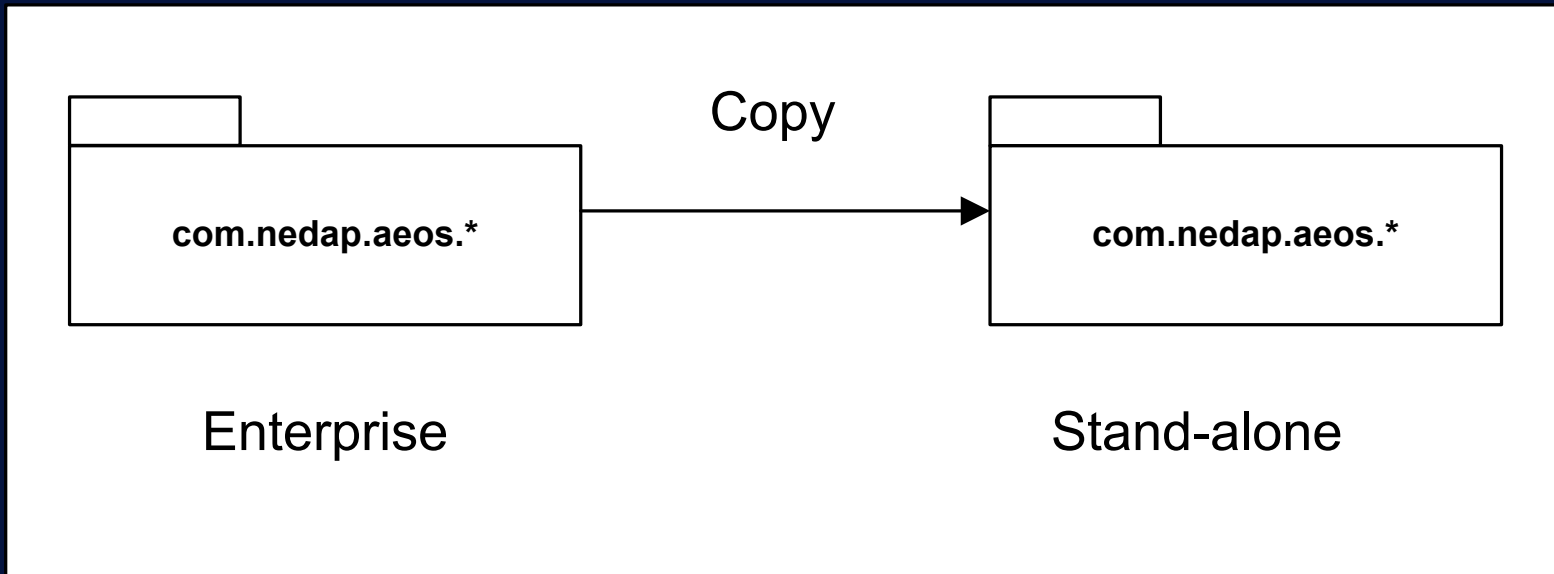


- **Kleinschalig: max. 1000 (50.000+) dragers, 16 (100+) deuren**
- **Beperkt tot basis functionaliteit**
- **Business logica is deels hetzelfde, deels anders**
- **Algemene applicatie logica is hetzelfde**

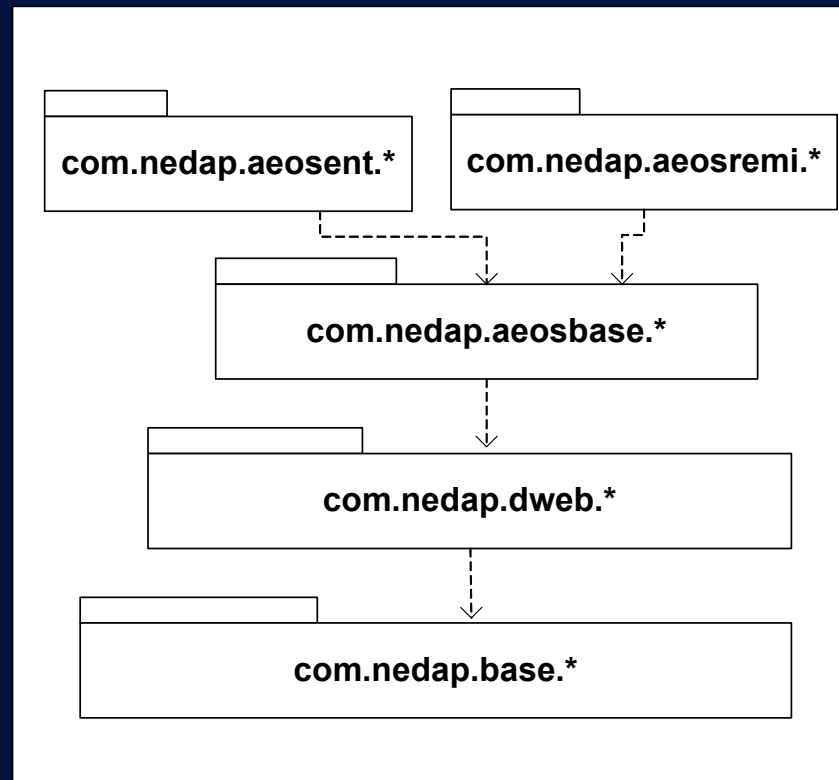
## Probleemstelling: hergebruik

- **Hoe is de business en applicatie logica te hergebruiken in stand-alone aEOS (Remi)?**
- **Problemen:**
  - Business logica zit in enterprise beans: EJB container nodig
  - Afhankelijkheden van J2EE container services als JNDI, transactie management, datastore, etc.
  - Gedistribueerd karakter (RMI) van EJB's (1.1)
  - Afhankelijkheid van Jini: lookup & registratie van Jini services
- **Reden: J2EE/EJB & Jini zijn opdringerige, complexe frameworks**

## Hergebruik: hoe het niet moet



## Hergebruik: hoe het wel moet



## Eenvoudiger EJB stap 1: design pattern

- **Standaard EJB home lookup (client code):**

```
personHome = null;
try {
    InitialContext ctx = new InitialContext();
    Object ref = ctx.lookup(JndiNames.EJB_HOME_PERSON);
    personHome = (PersonHome)PortableRemoteObject.narrow(ref,
        PersonHome.class);
}
catch (NamingException nex) {
    Logger.log(...);
    throw new SystemException(nex.toString());
}
```

- **Met EJBHomeFactory:**

```
personHome = (PersonHome)EJBHomeFactory.getHome(PersonHome.class);
```

## Stap 2: gebruik reflectie

- **Opzoeken van een SessionBean (client code)**

```
announcer = null;
```

```
AnnouncerHome announcerHome =
```

```
    (AnnouncerHome)EjbUtil.getEJBHome(AnnouncerHome.class);
```

```
try {
```

```
    announcer = announcerHome.create();
```

```
} catch (CreateException ex) {
```

```
    EjbUtil.handleAsSystemException(ex, "Could not create the Announcer");
```

```
} catch (RemoteException ex) {
```

```
    EjbUtil.handleAsSystemException(ex, "Could not create the Announcer");
```

```
}
```

- **Na toepassing van reflectie**

```
announcer = (Announcer)EjbUtil.createSessionBean(AnnouncerHome.class);
```

## Aspect Oriented Programming (AOP)

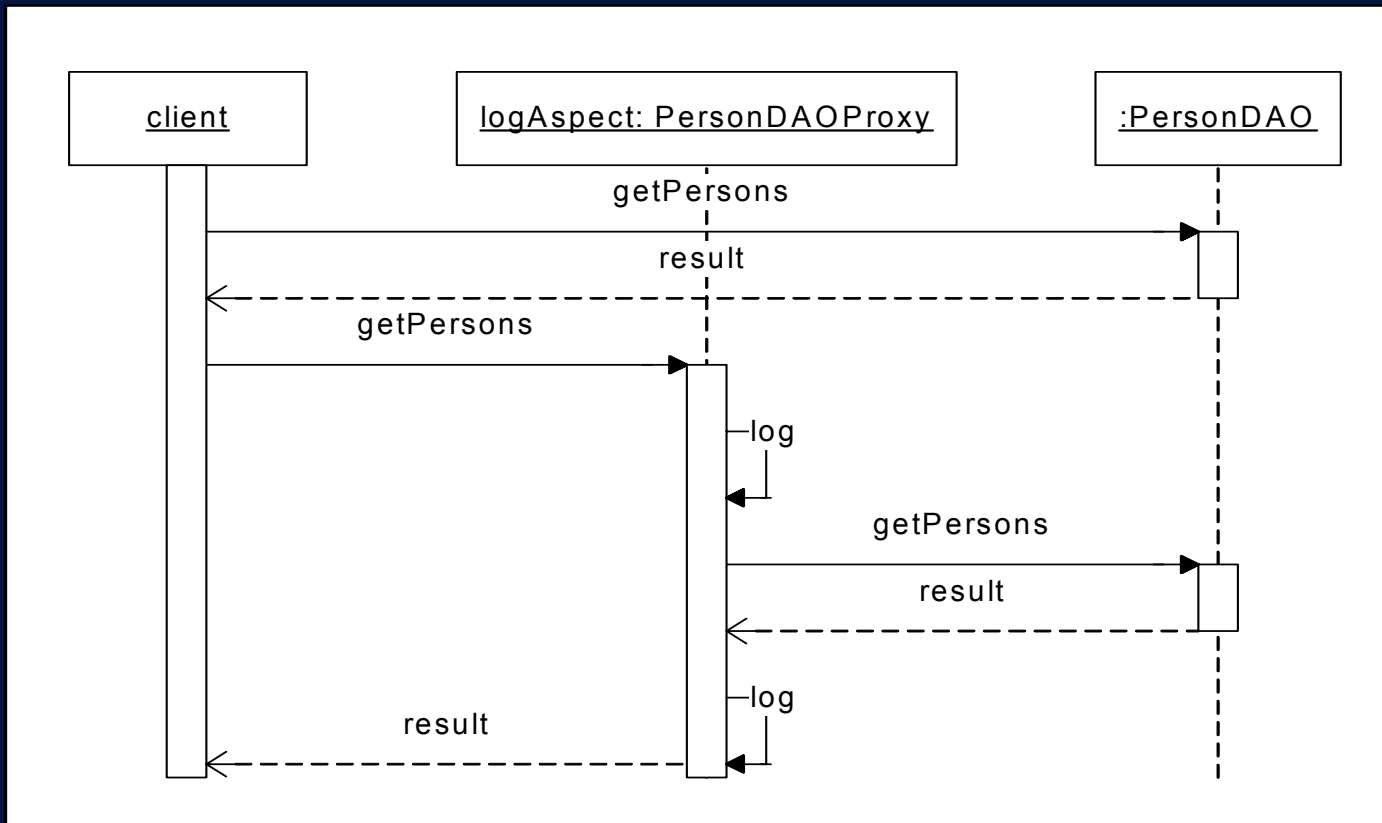
- **Logging voorbeeld:**

```
public Collection getPersons(Collection personIds, int max) {  
    log("getPersons: personIds=" + personIds + ", max=" + max);  
    // Collection result = ...(haal persoonsgegevens uit DB)  
    log(result);  
}
```

(Ook voor getVehicles(..), createPerson(..), etc.)

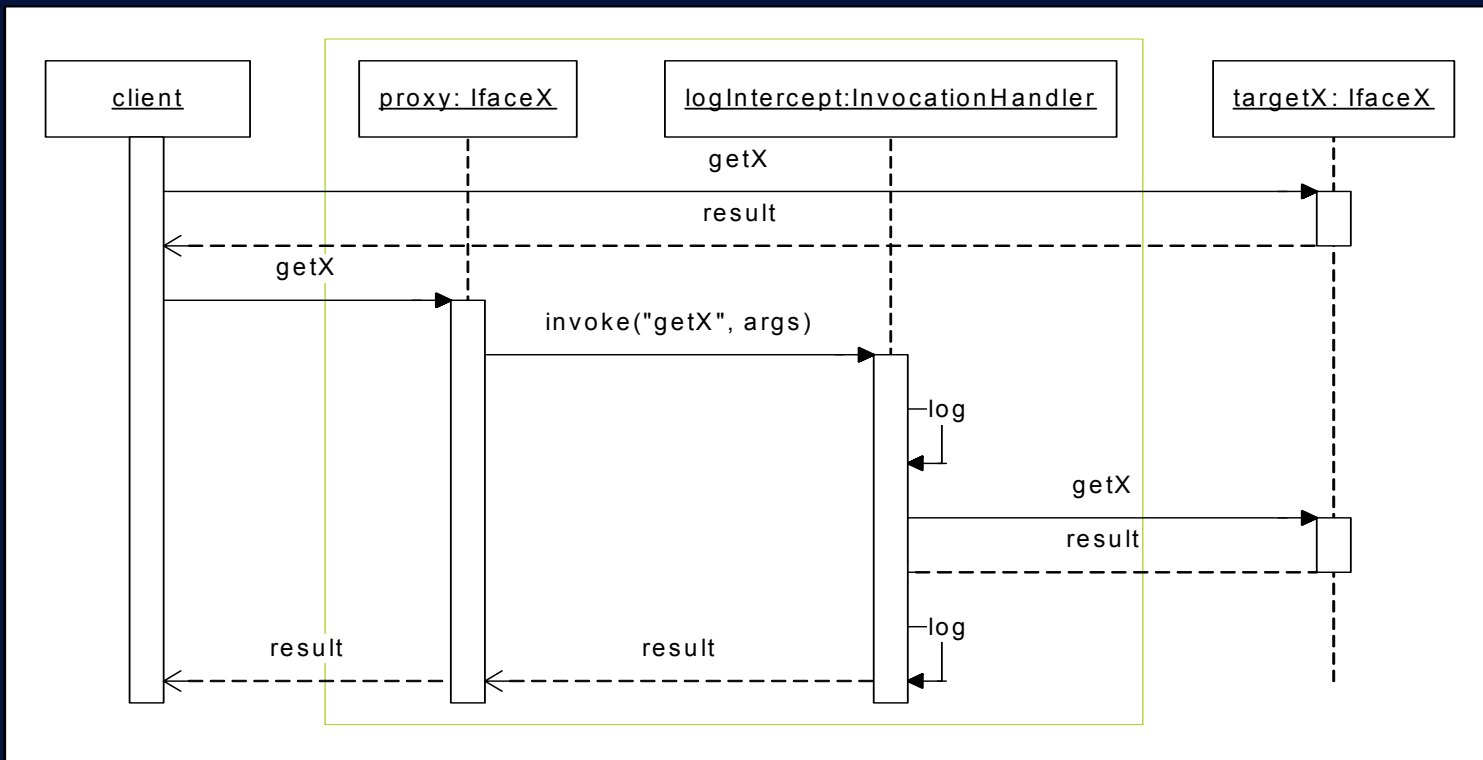
- **Method in/uit logging als aspect heeft als voordeel:**
  - op 1 plek: aan/uit, formattering, uitbreidingen
- **Aspect is een “cross-cutting concern”**

## Logging aspect sequence diagram



## AOP middels dynamische proxies

- `java.lang.reflect.Proxy.newProxyInstance(IfaceX.class, invocationHandler)`



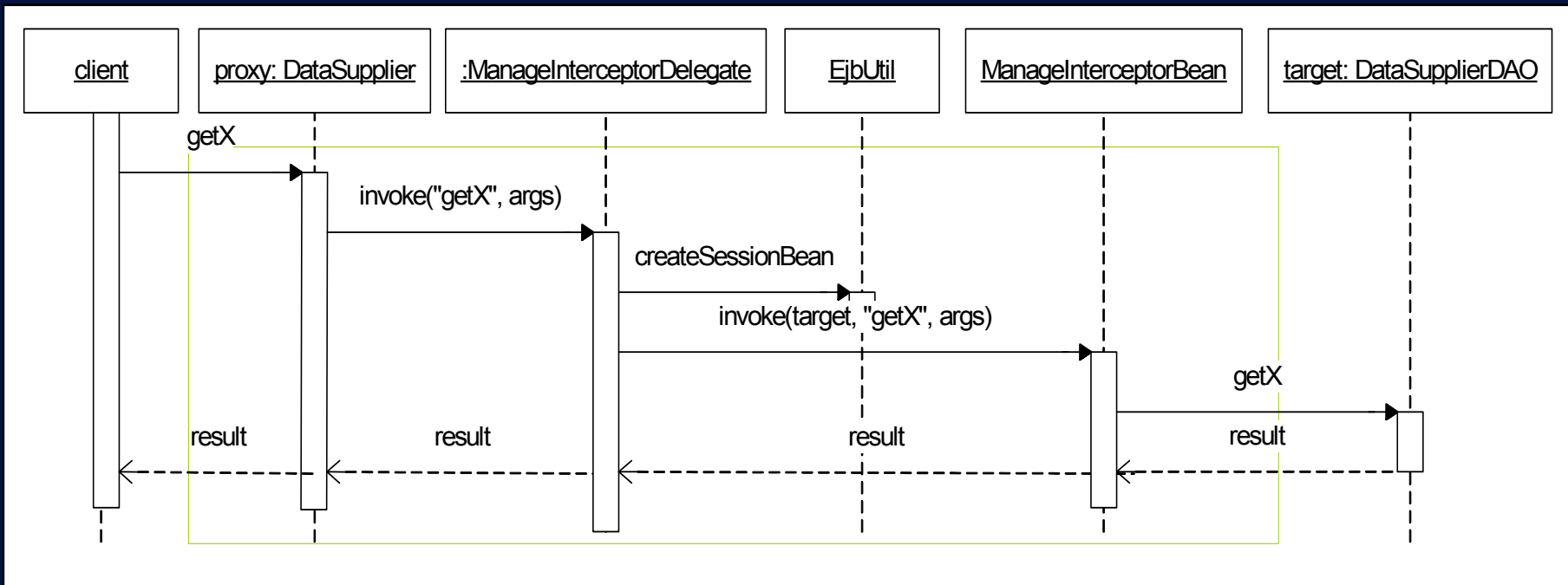
## AOP - transactie voorbeeld

```
dbConnection.setAutoCommit(false);  
...  
public void createPerson(PersonDetails personDetails) {  
    try {  
        s = dbConnection.createStatement();  
        s.executeUpdate("INSERT INTO person ...");  
        s.executeUpdate("INSERT INTO profile ...");  
        dbConnection.commit();  
    }  
    catch(SQLException e) {  
        dbConnection.rollback();  
    }  
}
```

## Stap 3: Maak SessionBean een aspect

- **Veel bestanden per SessionBean: Home interface, Remote interface, ejb-jar.xml, jaws.xml, jboss.xml**
- **Maar bijna altijd dezelfde eigenschappen (SL, TX, DB, JNDI)**
- **En bij iedere call:** `try{} catch(RemoteException e) {handleAsSystemException(e);}`
- **Oplossing: Op 1 plek: SL SessionBean als aspect**  
`supplier = (DataSupplier)EjbUtil.createResourceManaged(DataSupplierDAO.class);  
supplier.getAccessIdentifiers();`
- **Nu enkel nog nodig:**
  - POJI: DataSupplier
  - POJO: DataSupplierDAO

## SessionBean als aspect – sequence diagram



## SessionBean als aspect – code

- **EjbUtil**

```
public Object createResourceManaged(Class targetClass, Class iface) {  
    return Proxy.newProxyInstance(iface, new ManageInterceptorDelegate(targetClass));  
}
```

- **ManageInterceptorDelegate implements InvocationHandler**

```
In constructor: targetInstance = targetClass.newInstance();  
public Object invoke(Method method, Object[] args) throws Throwable {  
    try { ManageInterceptor interceptor = EjbUtil.  
        createSessionBean(ManageInterceptorHome.class);  
        result = interceptor.invoke(targetInstance, method, args);  
    } catch (RemoteException e) { ErrorUtil.handleError(e); }  
    return result;  
}
```

- **ManageInterceptorBean**

```
public Object invoke(Object targetInstance , Method method, Object[] args) {  
    return method.invoke(targetInstance , args);  
}
```

## Stap 4: Configureren van aspecten en implementatie: in Lookup class

- **Je ‘kiest’ het aspect en implementerende class elke keer bij het verkrijgen van referentie**

```
dataSupplier = (DataSupplier)EjbUtil.createResourceManaged(DataSupplierDAO.class);
```

- **De Lookup class biedt configuratie en opzoeken via interface (Registry/ServiceLocator pattern)**

```
Lookup.registerManagedService(DataSupplier.class, DataSupplierDAO.class);
```

**Client:**

```
dataSupplier = (DataSupplier)Lookup.getService(DataSupplier.class);
```

- **“Naming pattern” voor default configuratie: Met interface DataSupplier en implementatie DataSupplierObject in zelfde package, dan zelfs geen registratie nodig**

## Andere technologieën ook als aspect: Integratie via de Lookup

- Zoals resource managed services (SessionBean), ook bij:
- ‘Singleton’ services (gestart in JMX MBean, geregistreerd bij JNDI)

```
Lookup.registerSingletonService(ACSynchronizer.class, ACSynchronizerImpl.class);  
synchronizer = Lookup.getService(ACSynchronizer.class);
```

- **Network services (Jini)**
  - Verschillende instanties, typisch 1 per host (aEPU)
  - Remote exceptions moeten wel afgehandeld: standaard strategieën

```
acServices = Lookup.getAllServices(AccessControlNetService.class);  
acService = Lookup.getService(serviceKey, RetryPolicy.timeout(7000));
```

## Configuratie voor aEOS stand-alone: 'Unpluggen' van EJB, JNDI en Jini

- **Configuratie van de aspecten: "XXXInterceptedFactories"**

### **aEOS Enterprise:**

```
Lookup.setResourceManagedFactory(new SessionEJBInterceptedFactory());  
Lookup.setSingletonFactory(new JNDIInterceptedFactory());  
Lookup.setNetworkedFactory(new JiniInterceptedFactory());
```

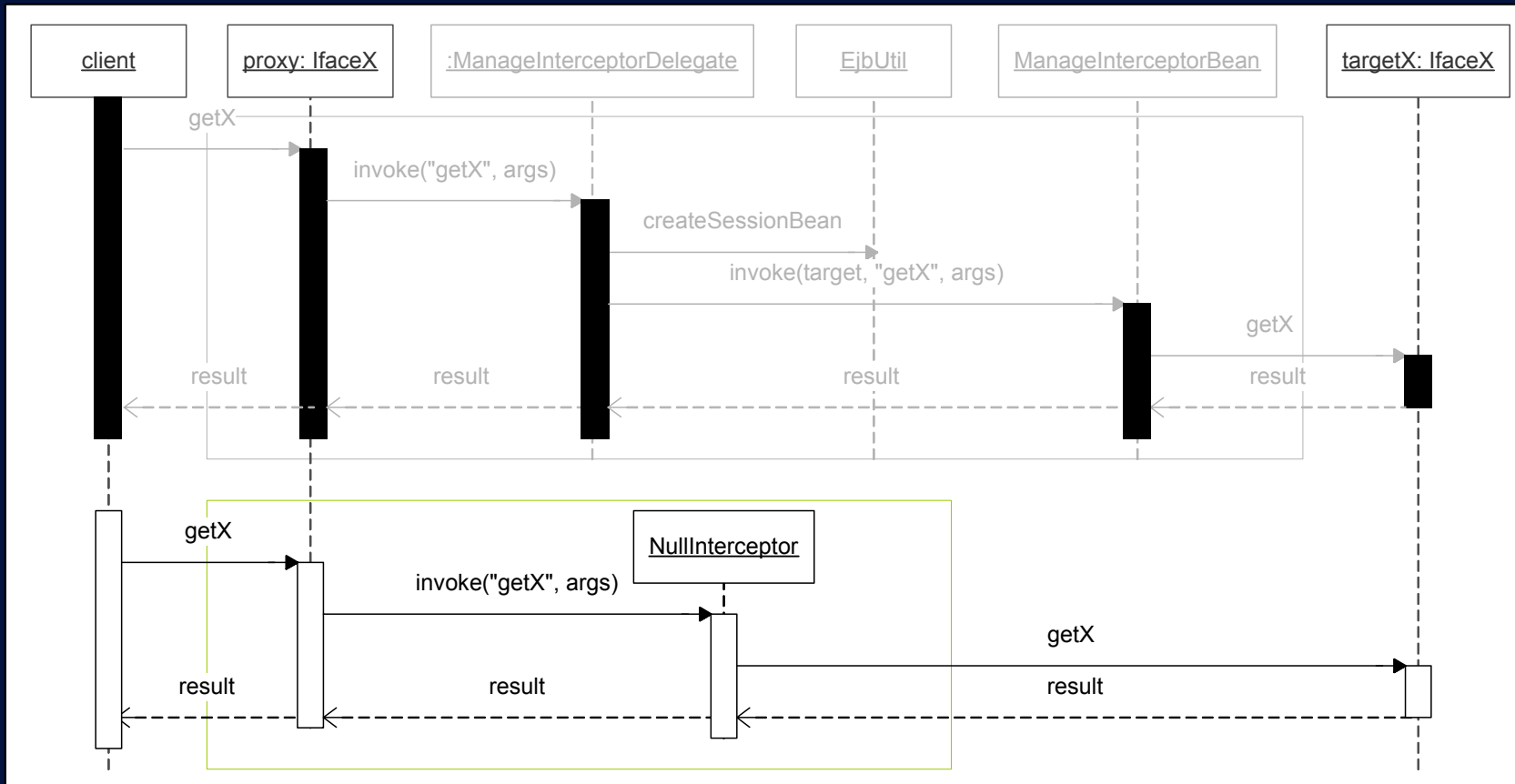
### **aEOS Stand-alone:**

```
Lookup.setResourceManagedFactory(new NullInterceptedFactory());  
Lookup.setSingletonFactory(new NullInterceptedFactory());  
Lookup.setNetworkedFactory(new NullInterceptedFactory());
```

- **Dit voor hergebruik van dezelfde logica**
- **Configuratie van implementatie: Verschillende logica achter dezelfde interface**

```
Lookup.registerManagedService(PersonAnnouncer.class, PersonAnnouncerRemi.class);
```

# 'Unpluggen' van EJB met de NullInterceptor



## Conclusies voor deze toepassing van AOP

- **Opdringerigheid van frameworks succesvol ingekapseld**
  - Technologie is 'pluggable' geworden
  - Hergebruik van logica tussen aEOS enterprise en stand-alone mogelijk
  - Reductie van vervuiling en complexiteit
  - Logische OO model staat centraal i.p.v. de gebruikte technologie
  - Veel minder afhankelijk van technologie & versies
  - Testbaarheid aEOS Enterprise onderdelen buiten de app server

## Referentias

- “J2EE Design Patterns” – John Crupi et al.
- “EJB design patterns” – Floyd Marinescu
- “J2EE Development and Design” – Rod Johnson
- “Effective Java” – Joshua Bloch
- [www.martinfowler.com/articles/injection.html](http://www.martinfowler.com/articles/injection.html)
- [java.sun.com/j2ee](http://java.sun.com/j2ee)
- [www.sun.com/software/jini](http://www.sun.com/software/jini)
- [www.nedap.com](http://www.nedap.com)
- [www.nedap-aeos.com](http://www.nedap-aeos.com)

**Vragen?**