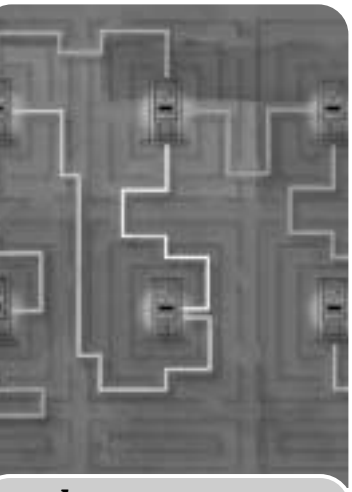


thema

De laatste jaren zijn patterns, met name design patterns, sterk in populariteit gegroeid. Inmiddels zijn er al verschillende soorten patterns ontstaan: architectural patterns, system patterns, anti-patterns. In artikelen en boeken wordt verwezen naar design patterns zoals beschreven in het boek van Erich Gamma (et al.) Vaak wordt er vanuit gegaan dat iedereen volledig vertrouwd is met de concepten hierachter. In deze reeks artikelen worden design patterns behandeld aan de hand van voorbeelden uit het dagelijkse leven die in Java geprogrammeerd zouden kunnen worden. De doelstelling is om design patterns inzichtelijk en begrijpelijk te maken en zo de mystiek weg te nemen en er zelf mee aan de slag te kunnen.

Het zijn maar patronen (2)

Toepassing van proxy pattern in Java-applicatie

Wie geregeld in de bioscoop of thuis een film bekijkt zal zien dat menig hoofdpersoon halsbrekende toeren uit moet halen. Van een klif afspringen, een achtervolging tijdens spitsuur met hoge snelheden en gevechtsscenes zijn voor de meeste 'normale' mensen niet weggelegd. De acteurs weten het geheel overtuigend neer te zetten alsof het de normaalste zaak van de wereld is. Wie echter op de aftiteling let, zal zien dat in dergelijke films een lijst aan stuntpersonen mee doet. Op de gevaarlijke momenten zullen zij de plaats innemen van de acteurs zodat de acteurs geen verwondingen oplopen. Als de stuntpersoon zijn werk goed doet, zie je geen verschil en zal op het juiste ogenblik de acteur weer in beeld zijn.

STAND-IN Binnen een Java applicatie kunnen er verschillende redenen zijn om het Proxy pattern toe te passen. Dit vertaalt zich naar drie verschillende variaties die allemaal hun eigen context hebben waarbij ze van waarde zijn. In algemene zin is het doel van een Proxy pattern om een plaatsvervanger of 'stand-in' te zijn voor een ander object. De Proxy zal wel altijd een referentie hebben naar het object dat hij vervangt alhoewel deze nog niet op elk moment geinstantieerd hoeft te zijn. Om als plaatsvervanger dienst te kunnen doen zullen zowel de vervanger (proxy) als het object dat vervangen wordt dezelfde interface implementeren. Voor de rest van de applicatie zal het geen verschil maken welk van de twee (proxy of origineel) zij aanroept. Uiteraard verschilt de uitwerking wel.

> Het doel van een Proxy Pattern is om een plaatsvervanger te zijn van een ander object

VARIANTEN

De drie varianten van de Proxy zijn:

- Virtual Proxy
- Remote Proxy
- Protection Proxy

De eerste variant is vergelijkbaar met de stand-in van de film. Als een object relatief veel resources in beslag neemt terwijl dat niet noodzakelijk is voor het normale gebruik van de instantie is dit pattern geschikt. Voor het gebruik van een adressensysteem is het niet altijd noodzakelijk ook werkelijk alle gegevens tot je beschikking te hebben om er toch gebruik van te kunnen maken. Vaak is het genoeg als je weet hoeveel contacten er in het bestand staan. Ook als je een nieuw adres toe wil voegen hoef je niet eerst het hele systeem te creëren. De virtual proxy bevat de logica om te kunnen bepalen wanneer het systeem geheel of gedeeltelijk moet worden geinstantieerd. De virtual

proxy zal het origineel in een keer in zijn geheel instantiëren of stukje bij beetje het origineel opbouwen. Zolang de proxy voldoende informatie heeft, kan hij de creatie van het origineel uitstellen en zelf de gegevens aan de client geven.

RETURNWAARDE De remote proxy is verantwoordelijk voor het afhandelen van het netwerk verkeer met het origineel. De proxy zal alle argumenten bij het aanroepen van een methode netjes inpakken en versturen naar het origineel. Vervolgens moet ook de returnwaarde uitgepakt worden om weer terug te kunnen geven aan de client die de methode in eerste instantie aanriep op de proxy. Voor deze client is het netwerk onzichtbaar geworden tenzij er iets mis gaat in de communicatie tussen de twee genetwerkte objecten.

Dit is de manier waarop Remote Method Invocation (RMI) werkt. Van elk object dat een remote interface implementeert wordt een stub (andere naam voor een proxy) gegenereerd die verantwoordelijk is voor de communicatie. Het inpakken en uitpakken van de argumenten heet binnen rmi marshall en unmarshall. Daarbij worden de argumenten geserialiseerd en voorzien van een codebase property. Dit laatste om ervoor te zorgen dat de ontvanger weet welke class hij moet gebruiken en waar hij deze kan downloaden. Deze complexe handelingen zijn nu voor de gebruiker van de proxy verbergen; het netwerk wordt nu alleen nog zichtbaar indien er problemen zijn met de communicatie.

> De virtual proxy bevat de logica om te kunnen bepalen wanneer het systeem moet worden geïnstantieerd

TOEGANGSCONTROLE De laatste variant, de protection proxy, dient om de toegang tot specifieke zaken te bewaken. Op basis van door de programmeur opgestelde eisen wordt er gekozen of de aanroeper van de methode wel of geen toestemming heeft om deze methode te gebruiken.

Indien de client rechtstreeks bij het object (het origineel) zou kunnen, heb je als programmeur die mogelijkheid niet. Alle toegangscontrole vindt nu plaats op een plek. Een protection proxy is als een agent voor een acteur; de agent handelt allerlei zakelijk details af en alleen als het noodzakelijk is verschijnt de acteur (handtekening zetten, acteren).

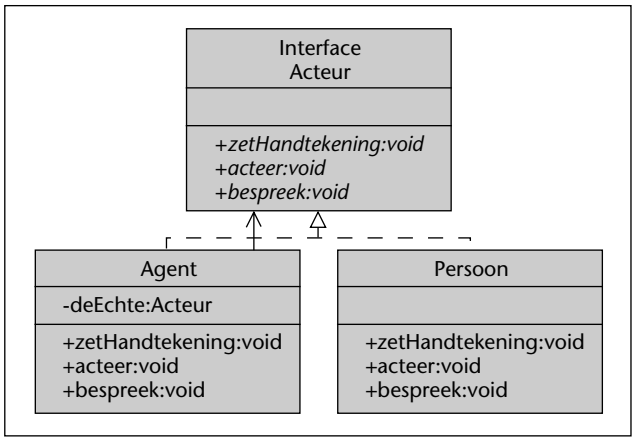
TOEPASSEN EN HERKENNEN Voor het toepassen van het Proxy design pattern is het volgende noodzakelijk (zie class diagram 2)

```
package studio.film;

public class Persoon implements Acteur {
  public void zetHandtekening() {
    // doe allerlei fan zaken
  }

  public void acteer() {
    //acteer in film
  }

  public void bespreek(Contract contract) {
    // doe niks
  }
}
```

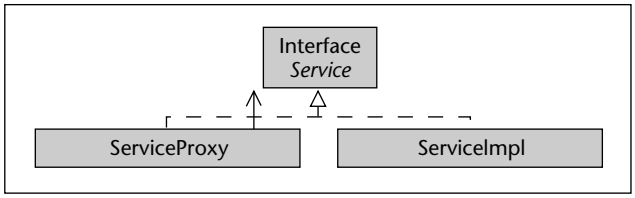
CODE VOORBEELD 1: Java uitwerking van het Proxy pattern

```
package studio.film;

public interface Acteur {
  public void zetHandtekening();
  public void acteer();
  public void bespreek(Contract contract);
}
```

CODE VOORBEELD 2: De interface van de acteur en de agent



UML Diagram 1



UML Diagram 2

```
package studio.film;

public class Agent implements Acteur {
  private Acteur deEchte;

  public void zetHandtekening() {
    deEchte.zetHandtekening();
  }

  public void acteer() {
    deEchte.acteer();
  }

  public void bespreek(Contract contract) {
    //handel deze bespreking zelf af
  }
}
```

CODE VOORBEELD 3: De implementatie van de Proxy

- Service - interface met daarin de definitie van de te gebruiken methoden
- ServiceProxy - de proxy class die Service implementeert en op de juiste momenten de methode aanroepen doorstuurt naar de volledige implementatie van Service
- ServiceImpl - De volledige implementatie van de Service, waarbij ServiceProxy als representant van deze class optreedt

Voordelen Proxy

- Verbergen van complexiteit en netwerk zaken door de complexiteit in de proxy af te handelen
- Betere controle over wie toegang heeft tot bepaalde functionaliteit
- Er is een plaatsvervanger van een object aanwezig waardoor je het werkelijke achterliggende object nog niet hoeft te instantiëren

TENSLOTTE De Proxy is het tweede pattern welke beschreven is in deze reeks over design patterns. De meeste patterns in deze reeks zijn net als de Proxy voor het eerst beschreven in "Design Patterns" van de Gang of Four (1). Volgende keer gaat het over HOPP, een pattern welke niet beschreven staat in het GoF boek.

LITERATUUR

(1) Design Patterns, Erich Gamma, et. al., Addison-Wesley, 1995

(2) Applied Java Patterns, Stephen Stelting & Olav Maassen, Prentice Hall, 2002

Olav Maassen is senior Java developer bij ITIS J-solutions B.V. te Maarsbergen en co-auteur van het boek "Applied Java Patterns", samen met Stephen Stelting.

PATCHES Patches PATCHES Patches PATCHES Patches PATCHES

Ontwikkelomgeving voor COBOL-applicaties

Op het GigaWorld IT Forum maakte Micro Focus International Ltd. de nieuwe release van zijn Net Express ontwikkel- en uitvoeringsomgeving bekend. Met Net Express 4.0 kunnen bedrijven de ontwikkeling en uitvoering van hun bestaande COBOL-applicaties ter hand nemen. Dit is mogelijk door ze uit te breiden naar Windows- en UNIX-omgevingen. De verbeteringen aan Net Express omvatten directe COBOL Web Services, J2EE-connectiviteit en XML-ondersteuning, waarmee bedrijven COBOL-applicaties opnieuw kunnen gebruiken en integreren in strategische architecturen. De introductie van Net Express 4.0 bouwt voort op de strategie van Micro Focus om de inzet van COBOL-applicaties te verbreden. Ook met deze nieuwe release richt het bedrijf zich er op om COBOL het belangrijkste keuzeplatform te maken voor bedrijven waar flexibiliteit in IT-beslissingen van groot belang is.

Micro Focus introduceert ook Enterprise Server, een nieuwe uitvoeringsomgeving voor Net Express. Enterprise Server breidt COBOL uit tot een onafhankelijk platform met een schaalbare, betrouwbare en transactionele COBOL-uitvoeringsomgeving voor two-tier client/server- en three-tier samengestelde COBOL-applicaties die zijn ontwikkeld met Net Express. Enterprise Server omvat ondersteuning voor directe webservices en een COBOL resource adapter voor J2EE-verbindingen. Hiermee kan COBOL worden geïntegreerd met alle veelvoorkomende run-time technologieën. Enterprise Server, in combinatie met Net Express, maakt een kosteneffectieve ontwikkeling en inzet mogelijk van nieuwe bedrijfsapplicaties. Tegelijkertijd worden bestaande bedrijfsprocessen en expertise opnieuw gebruikt.

Met Net Express kunnen de miljarden COBOL-coderegels zoals die worden gebruikt in de huidige mainframe-systemen, worden benut door ze direct om te zetten in COBOL-webservices en platformonafhankelijke applicaties. Deze zijn direct voor het gehele bedrijf of het internet beschikbaar. Met de functionaliteit voor directe webservices kan een programma in COBOL worden ontwikkeld, ingezet en gebruikt, waardoor bedrijven hun zakelijke activiteiten naar het internet kunnen uitbreiden en de kosten voor de roll-out op het intranet kunnen verlagen. Met webservices is het mogelijk Java en .NET te gebruiken voor toegang tot COBOL.

Tevens ondersteunt Net Express samenwerking tussen ontwikkelaars met verschillende expertisegebieden. Meer informatie vindt u op: www.microfocus.com.