

Vorlesung Software aus Komponenten

3. Komponentenmodelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2013/14

Transaktionsdienst (object transaction service, OTS)

- einer der wichtigsten Bausteine für verteilte Anwendungen
- standardisiert seit 1994
- wird von den meisten ORB-Produkten und J2EE-Servern unterstützt
- **Eingebettete Transaktionen** nur optional
 - Transaktionshülle um Folge von Operationen
 - erforderlich zur unabhängigen Entwicklung auf verschiedenen Hierarchie-Ebenen
 - (noch) nicht standardisiert, weil kaum eines der heute ex. Transaktionssysteme so etwas vorsieht
- Verwaltung eines (objektspez.) aktuellen Tr.-**kontexts** durch OTS
 - Objekte müssen dazu die Schnittstelle *TransactionalObject* implementieren
 - Methoden *begin*, *commit*, *rollback* operieren auf dem Kontext
- Objekte unter Transaktionskontrolle registrieren sich beim OTS-Koordinator-Objekt

Transaktionsdienst (Fortsetzung)

- Ressourcen müssen die Schnittstelle *Resources* implementieren
 - Koordinator wickelt darüber 2-Phasen-commit-Protokoll ab
 - bekanntes Problem der Deadlock-Gefahr
 - 3-Phasen-Protokoll vermeidet diese, ist aber teurer
- heute weit verbreitet: Transaktionskontrolle nicht als separater Dienst, sondern als Kontextkontrolle **innerhalb** eines Anwendungsservers
 - Diese Abstraktion wird vom CCM abgedeckt

Sicherheitsdienst (security service)

- erforderlich, wenn sich verteilte Anwendung über mehrere Vertrauensbereiche (trusted domains) erstreckt
- spezifiziert in **CORBAsecurity**
- Authentifizierung, sichere Kommunikation, Zertifizierung
- volles Spektrum wird derzeit von kaum einem Produkt unterstützt
 - meist nur SSL-basierte Sicherheit
 - unterstützt einfache Sicherheit, aber keine Zertifikate

Lizenzdienst (licensing service)

- Verwaltung von Objektlizenzen, Abrechnung von Gebrauchsgebühren für Objekte
- Unterstützung verschiedener Lizenzmodelle
- 2 Schnittstellen: *Lizenzdienst-Manager* (LDM) und *Lizenzdienst*
- Objekt unter Lizenz (OL) erfährt über LDM, unter welchen Bedingungen seine Nutzung legitimiert ist
 - OL fordert vom LDM Referenz auf entsprechendes (hersteller-spezifisches) Lizenzdienst-Objekt (LDO) an
 - OL informiert LDO über **Kontext** der Lizenzanforderung
 - LDO prüft, welche Nutzung des OL in dem Kontext legitim ist
 - LDO veranlasst Übergang von OL in erlaubten Zustand (ggf. Demo-Modus, Probe-Modus)
 - OL informiert LDO über Ende der Nutzung
- aktuelle Lizenzgestaltung also gekapselt zwischen OL und LDO
- zwischen beiden kann auch statistisch relevante Information ausgetauscht werden
 - Nutzerprofile, Lizenzdauer und -ablauftermine

Lebenszyklusdienst (lifecycle service)

- Verwaltung von Objekten (Erzeugen, Kopieren, Löschen, Verschieben) oder Gruppen von Objekten
- unterstützt Objekterzeugung durch Factory-Objekte
 - Registrierung, Wiederverwendung letzterer
- Objektverwaltung mit Referenzzählern in verteilten Anwendungen oder mit verteiltem garbage collection wird nicht unterstützt
 - Grund: verteiltes garbage collection in fehleranfälliger Umgebung (Maschinen- oder Netzwerkausfall) ist sehr kompliziert, braucht Transaktionskontext
 - kein Problem beim Einsatz von CORBA als Kommunikations-Middleware, da dort Objekte gewöhnlich Serverobjekte mit unbegrenzter Lebensdauer

Beziehungsdienst (relationship service)

- Erzeugen, Löschen und Verwalten von Beziehungen zwischen Objekten, Navigation über Beziehungen

Persistenz-Dienst (persistent state service, PPS)

- Persistenz = Eigenschaft eines Objekts, das Programmende zu überleben
- CORBA 2: Persistenzobjekt-Dienst (persistence object service, POS)
 - seit 1994, erste Implementierungen 1996
 - unterspezifiziert: konkrete Speicheranforderung war anwendungsspezifisch gelöst
- CORBA 3: Ablösung durch Persistenzzustands-Dienst
- Grundlegender Ansatz: Trennung von persistentem Objekt und Persistenzmechanismus
 - Dateien, Datenbanken
 - strukturierte Speicher (Containerdokumente)
- sehr einfache Schnittstelle: Speichern und Laden eines Objekts
- drei problematische Objekteigenschaften:
 1. Objekte haben Identität, sind nicht referenziell transparent
 - Problem beim mehrfachen Speichern / Laden

Persistenz-Dienst (Fortsetzung)

2. Objekte können sich aufeinander beziehen (Objekt-Web)
 - Beziehungen müssen mit gespeichert werden
 - wesentliche und flüchtige Beziehungen
 - Probleme beim Mehrfachspeichern (RAID etc.)
 3. Objekte sind Einheiten der Datenkapselung
 - Sicherung der Integrität von Objekten auf dem Speichermedium
 - Schutz vor Manipulation unter Umgehung der Objekt-Schnittstelle
- POS löste Probleme durch Kooperation zwischen Objekt und Persistenzdienst über ein Protokoll
 - PSS: explizite Deklaration, welche Objektteile wie zu speichern sind
 - neue OMG Beschreibungssprache für solche Deklarationen (**persistent state description language, PSDL**)
 - Spezifikation verschiedener abstrakter und konkreter Speichertypen (analog Schnittstellen und Klassen in Java)
 - Spezifikation entsprechender Factories

Auslagerungsdienst (externalization service)

- Linearisierung / Delinearisierung von Objekten
 - zueinander invers (erzeugt Objektkopie)
 - keine referenzielle Integrität
 - Wertkopie von Teilobjekten
 - Referenzen **nur** über ORB Referenzmechanismus
- zum Datenexport von Objekten in Dateien und Streams
- Schnittstelle *Streamable* des auszulagernden Objekts (AO)
- wird von Strom-Objekt gerufen, das selbst Schnittstelle *Stream* implementiert
 - über *externalize_to_stream* Methode des AO
 - erzeugt daraus ein lineares Objekt (LO), das Schnittstelle *StreamIO* implementiert
- es können ganze Graphen von Objekten ausgelagert werden.

Zeitdienst (time service)

- Synchronisierung der Uhren in verteilten Systemen
- Korrelation innerhalb sinnvoller Fehlerschranken, um zeitliche Kausalitäten über Systemgrenzen hinweg zu erhalten

Eigenschaftendienst (properties service)

- dynamisches Binden von Eigenschaften an Objekte
- keine semantische Interpretation der Eigenschaften
- Schnittstelle *PropertySet* mit Methoden *add*, *modify*, *delete*
- diese können normal, read-only (löschar, schreibgeschützt), fixed-normal (nicht löschar) oder fixed-read-only sein

Anfragedienst (object query service)

- Dienst zum Auffinden von Objekten nach Attributen
- ähnlich Händlerdienst, sucht aber Objektinstanzen
- Unterstützt Object Query Language (OQL-93) der Object Database Management Group) sowie SQL mit Objekterweiterungen
- Definiert Schnittstelle eigener *Sammeldienst*-Objekte
 - Semantik geordneter Mengen (add, remove, enumerate)
 - spezielle Schnittstelle *Iterator* zur Auswertung solcher Objekte
- Anfrage-Objekt kapselt die Anfrage, welche in zwei Schritten beantwortet wird: Vorbereitung und Abarbeitung der Anfrage

Anfragedienst (Fortsetzung)

- Vier Objekttypen:
 - Anfrage-Objekte (query object, QO) und Sammelanfragen (querable collections, QC)
 - Anfrage-Auswerter (query evaluator, QE) wertet QO oder QC aus und erzeugt Ergebnis-Sammelobjekt
 - Anfrage-Manager (query manager, QM) erzeugt QO oder QC und schickt sie an QE zur Beantwortung
- Das Objekt, das Anfrage generiert, benutzt *Iterator*-Schnittstelle zur Auswertung der Antwort

Sammeldienst (object collections service)

- Möglichkeit zum Bilden von Sammeltypen verschiedener Topologien, z.B. Mengen (bags, sets), Schlangen (queues), Listen (lists) oder Bäume (trees), entsprechend der Smalltalk-Klassifikation
- unklar, ob das nicht lieber auf Objektebene realisiert sein sollte
 - existieren effiziente Implementierungen dieser Datentypen auf Bibliotheksebene

Standardkomponenten (CORBAfacilities)

Standardisierung von häufig benötigten Anwendungsbestandteilen

- Komponentenrahmen zur einfachen Integration von Anbieterlösungen

Abgrenzung von Bereichen horizontal oder vertikal

- horizontal: Fokus auf generellem Anwendungsmodell
 - Standards für Nutzerschnittstellen, System- und Aufgabenverwaltung
- vertikal: Focus auf bereichsspezifischen Einsatzfeldern
 - im Rahmen von OMG SIG's oder Domain Task Forces

Standards zur Integration häufig benötigter Dienste als „Plugins“ in bestehende Komponentenrahmen (component frameworks)

- vereinfacht und standardisiert das Vorgehen bei der Integration von Komponenten verschiedener Anwender

Horizontale (generale) Standardkomponenten

Fokus auf generellem Anwendungsmodell. OMG hatte hier ursprünglich Standards für folgende Rahmen im Auge

- **Benutzerschnittstelle** (user interface)
- **Informationsverwaltung** (information management)
- **Systemverwaltung** (system management)
- **Aufgabenverwaltung** (task management)

Wird heute nur noch wenig vorangetrieben und stärker auf übergreifende Dienste konzentriert wie

- Internationalisierung, mobile Agenten, Zeit- und Druckdienst-Standards
- Fokus heute nicht auf Standard-Komponenten (ready to use), sondern auf Komponenten-Standards (Architektur)

Standardisierungsbemühungen der ursprünglichen Bereiche spielen heute praktisch so gut wie keine Rolle mehr.

Vertikale Standardkomponenten

Ursprünglich lag hier der Fokus auf Basisfunktionalitäten für unterschiedliche Marktsegmente. Ergebnisse bekommen zunehmend segmentüberschreitende Bedeutung

- Auch hier Komponenten-Standards statt Standard-Komponenten

Ausgehandelt werden dieses Standards in Aktivitäten verschiedener Domain Task Forces

- Business Enterprise Integration
- Command, Control, Communications
- Finanzbereich
- Bereich Gesundheitsvorsorge
- Lebenswissenschaften
- Produktionsstrukturen
- Telekommunikation usw.

Sun und Java

- Java: Geschichte und Konzepte
- Wichtige von Java unterstützte Grundkonzepte
- Die J2EE-Architektur
- Java Komponentenmodelle
- Java Servlets / Java Server Pages (JSP)
- Enterprise Java Beans
- Ein Beispiel

Java: Geschichte und Konzepte

große Erfolgsstory

- 1995/96 erste Anfänge (Sun Microsystems)
- Um 2000 eines der am häufigsten gebrauchten Schlagworte

Zwei Ansätze, womit Java wirklich zur Killerapplikation wurde

1. Das Sicherheitskonzept

- Applet wird doppelt geprüft (Übersetzungszeit und Ladezeit)
- strenge Sicherheitsregeln (security policies), die mit keiner anderen Programmiersprache erreicht werden
- Sicherheit auch im compilierten Code eines JIT-Compilers
- Sicherheitsaussagen durch Erzeugerzertifikate möglich
 - „signed applets“ (unter Nutzerkontrolle)

3.4. Java

Java: Geschichte und Konzepte

2. Die Java virtual machine

- Plattformunabhängigkeit
- großer Vorteil für Applikationen, die übers Web verteilt werden
- Vorteil vor allem im Standard
 - Java class-File Format und Java JAR-Archiv-Format
- beides nicht neu, jedoch in der Kombination und zu diesem Zeitpunkt durchschlagend

Java 2 (seit 1998)

- Fokus auf Applets aufgegeben
- Applets heute nur noch marginal
- Plattform-Editionen – Funktionalitätsbündel für verschiedene Klassen von Nutzern
 - J2SE als Plattform für Einzelanwendungen
 - J2EE als Serverplattform (seit Ende 1999)
 - J2ME für mobile und eingebettete Anwendungen

Java 2 (Fortsetzung)

- Formalisierung der Bezeichnungen Laufzeitumgebung (JRE), Entwicklungsumgebung (JDK) und Referenzimplementierung
- Referenzimplementierung der J2SE von Sun/Oracle auf der Basis der HotSpot-JVM
- J2EE als Standard mit Implementierungen von verschiedenen (unabhängigen) Anbietern
 - (nicht laufzeitoptimierte) Referenzimplementierung von Sun/Oracle als Beispiel-Implementierung im Quellcode verfügbar
- Prüfung der Unterstützung von Standards durch Kompatibilitätstest-Reihen
- Java BluePrints als Sammlung von Design-Richtlinien und Mustern, um spezielle Technologien zu unterstützen.