

TKeasy - 10 Jahre produktives Java Enterprise in Hamburg

Arbeitskreis Objekttechnologie, 9.5.2011, Hamburg
Ralf Degner, Techniker Krankenkasse



Überblick

- Voraussetzungen, Ziele, Entscheidungen
- Architektur: Logische Schichten und Business-Objects
- Das unternehmensweite OO-Modell
- Integration von Fremdsystemen und Batch - SOA
- Erfahrungen und Erfolgsfaktoren

Voraussetzungen

- EDV in der gesetzlichen Krankenkasse (GKV)
 - Keine „normale“ Versicherung, fast alles ist anders
 - Elektronischer Datenaustausch mit diversen Stellen
 - viele gesetzliche Regelungen, viele Anwendungen, häufige Änderungen
- Systeme bei Projektstart
 - Datenhaltung zentral auf dem Großrechner (DB2 / IMS-DB)
 - 3270-Masken (COBOL)
 - 2-Schicht-Anwendungen (Centura)
- 11.500 Mitarbeiter, davon ca. 7.500 parallel online
- ca. 230 Lokationen (1 bis 350 Mitarbeiter)
- ca. 7,5 Mio. Versicherte

Ziele der Anwendungsarchitektur

Fachlich

- Optimale Unterstützung der Prozesse der TK
- Komfortable und effiziente Bedienung
- Schnelle Realisierung von gesetzlichen Änderungen und optimierten Geschäftsprozessen

Technisch

- Moderne und langfristig tragfähige Architektur
- Gute Wartbarkeit
- Einbindung der Alt-Systeme
- Minimierung technologischer Abhängigkeiten

Entscheidungen

- Möglichst durchgängiger Einsatz von **objektorientierten Technologien und Methodiken**: Architektur soll dies ermöglichen
- Implementierung der Geschäftslogik im Application-Server
 - kein Masken/Unterprogramm-Wrapping
 - logisch „dünner“ Client
- Vollwertige GUI-Anwendung (kein HTML)
- Kein Big-Bang: schrittweise Ablösung aller Alt-Anwendungen
- Eigenentwicklung von Framework und Application-Server (1998 am Markt: BEA Iceberg, IBM Component Broker)
- Basistechnologien: Java und SWING

TKeasy in Zahlen

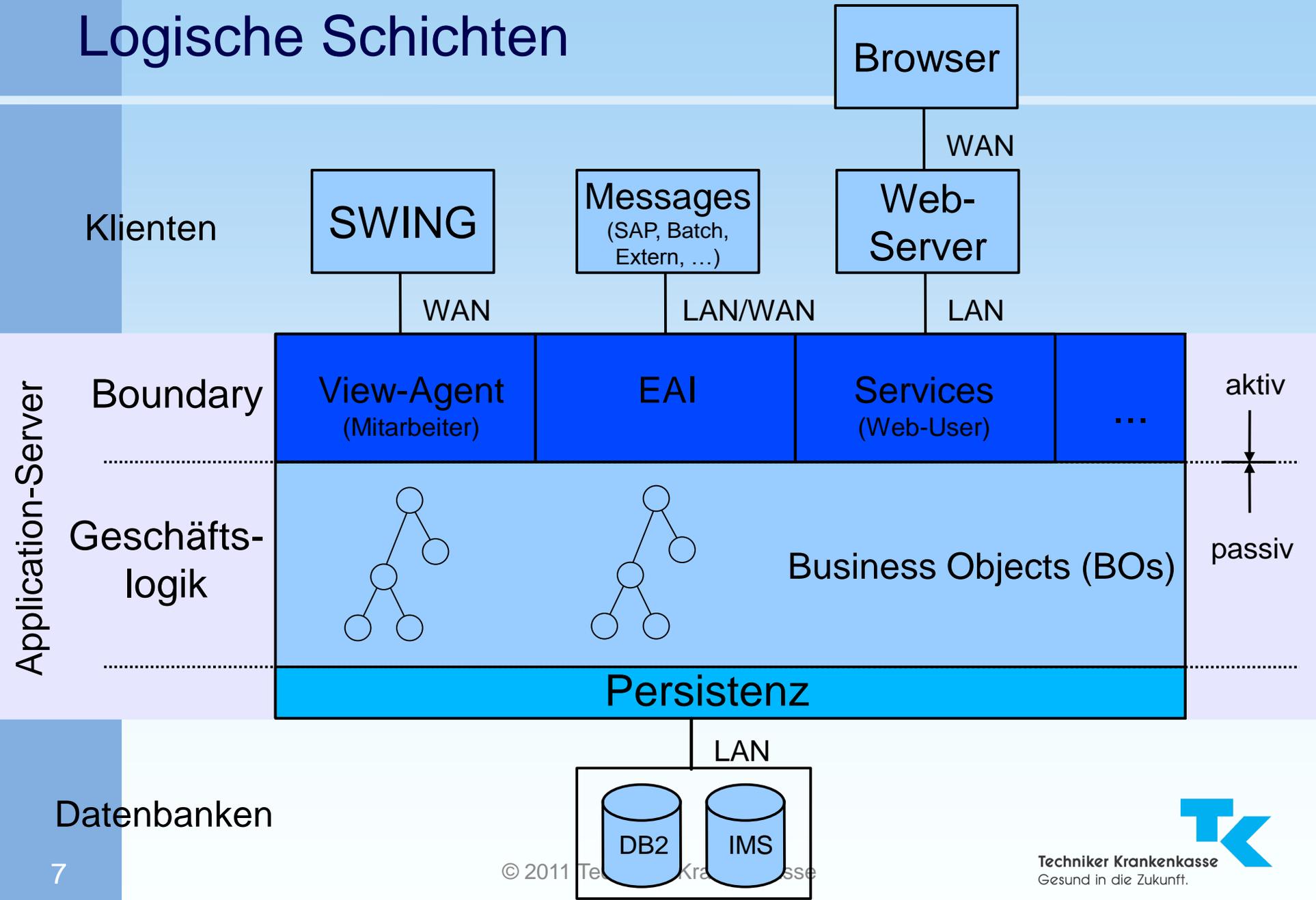
Zahlen pro Tag:

- 2.100.000 Anwendungen (Fenster)
- 6.300.000 C/S-Kommunikationen
- bis zu 2.000.000 Messages
- 5.500.000 Objekt-Transaktionen
- 48.000.000 Großrechner-Transaktionen (DB-Transaktionen)
- 410.000.000 BO-Instanzen
- 17.000.000.000 Feldzugriffe auf BOs (geschützt und transaktional)

Typische Antwortzeit : 0,42 Sekunden

Sehr viele Zugriffe auf viele BOs mit wenigen DB-Transaktionen durch sehr wenige C/S-Kommunikationen.

Logische Schichten

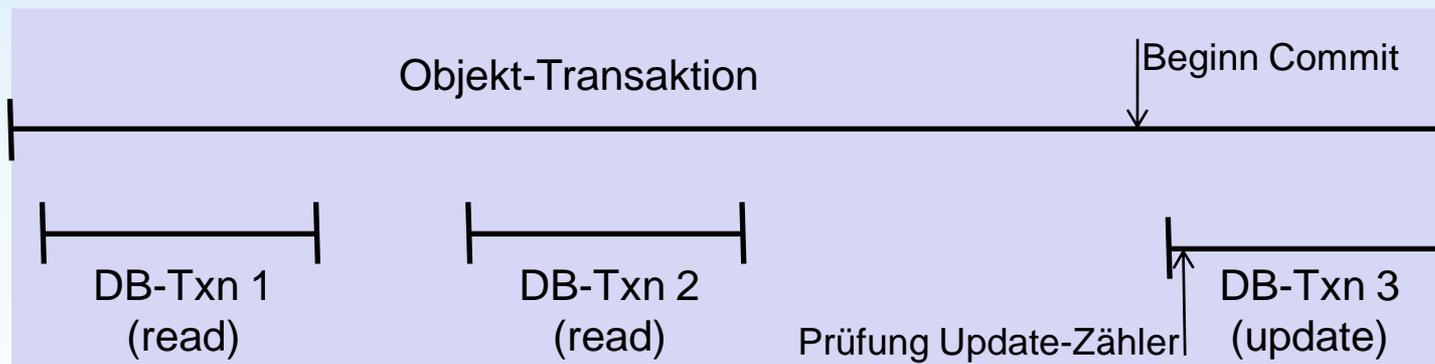


Datenbanken

Framework für Business Objects

Laufzeitumgebung für BOs

- Leichte BOs (POJOs) => feingranulare Modellierung möglich
- „Lange“ Objekt-Transaktion fasst mehrere kurze DB-Transaktionen zusammen
- Optimistisches Locking: Prüfung Update-Zähler vor Änderung in DB
- Verantwortung für ACID bei der Datenbank
- Angelehnt an Java Data Objects (JDO, ähnlich EJB3)



Framework für Business Objects - Persistenz

- Mapping für DB2 und IMS-DB – „alte“ Tabellen in „neue“ Objekte
- Endgültige Ablösung von IMS-DB
- Viele Optimierungsmöglichkeiten (insbesondere Prefetching)
- Generische Referenz für persistente Objekte – Typ „Object“ in DB
 - einheitliche Objekt-ID
 - Beispiel: viele konkrete Ausprägungen von „Rolle“ (Versicherter, Interessent, Arbeitgeber, ...) werden von Basis-Systemen (Partner, Postkorb, Vorgang, Schriftgut, ...) als „Rolle“ gespeichert

Performance der Persistenz sehr kritisch!

Das unternehmensweite OO-Modell (1)

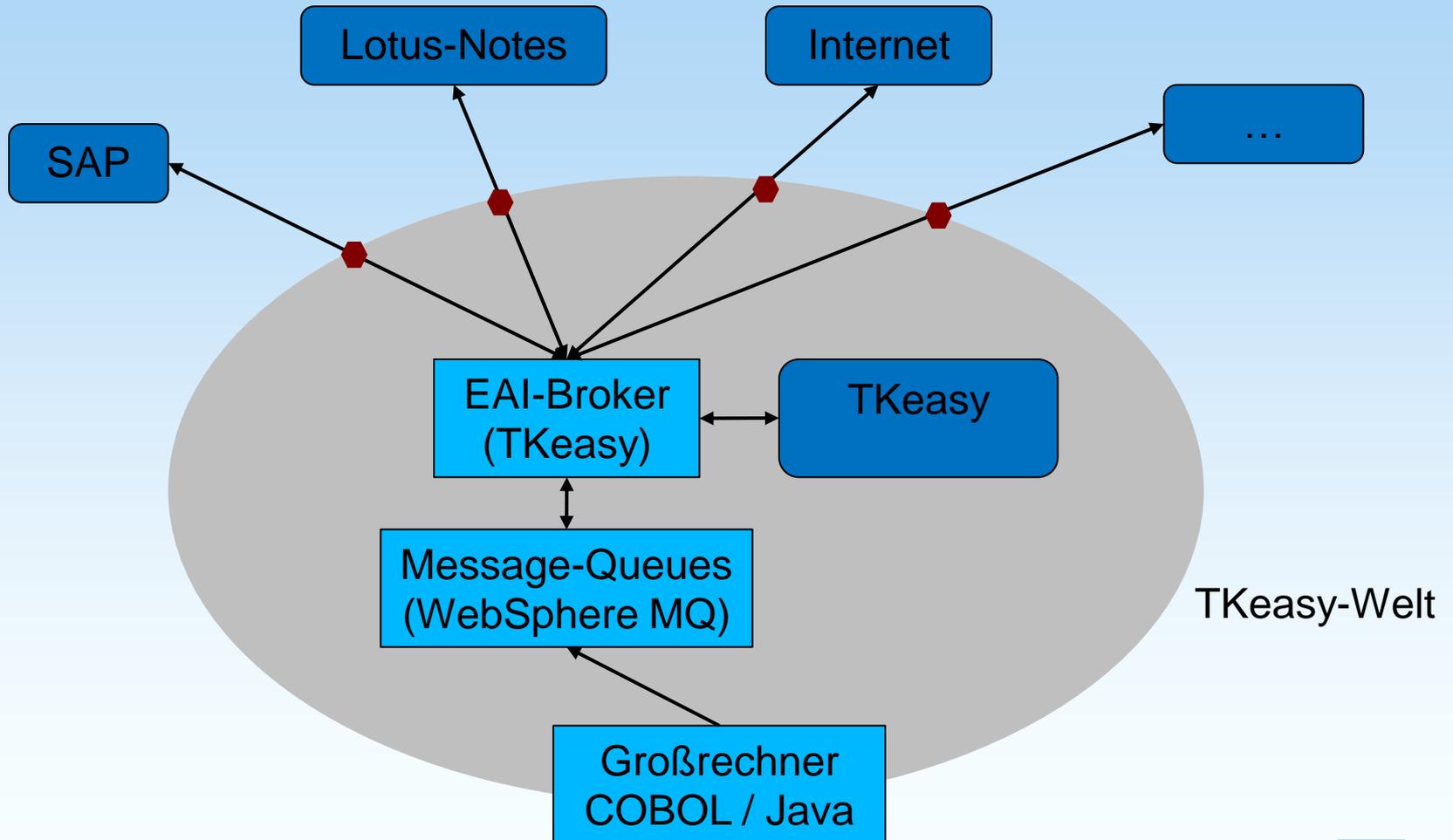
- Kein zentral entworfenenes OO-Modell
 - Modell wächst mit den Projekten
 - Architekten koordinieren
- Abhängigkeitsmanagement: WER darf WEN nutzen
 - definieren und prüfen
 - unterstützt auch gutes Design
- Fachliche Querschnittssysteme
 - Partner, Vorgänge, ...
- Fachliche Modelle „konkret“, nicht „generisch“
- „Harte“ Typisierung
 - Compile-Time-Safety: verhindert viele Fehler
 - Abhängigkeiten sichtbar

Das unternehmensweite OO-Modell (2)

- In Schnittstellen (Interfaces) denken
 - So lokal wie möglich entwerfen
 - Implementierung ist eine Black-Box, „darf“ Nutzer nichts angehen
 - Java-Klassen/Interfaces als Schnittstellen (keine technische Abstraktion)
- Viele Werttypen und Aufzählungstypen (unveränderbar)
 - Helfen BOs schlank zu halten
 - Einfache Wiederverwendung
 - Beispiele: Anrede, Geschlecht, Bankverbindung, BLZ, Geldbetrag, ...
- Generische Referenzen ermöglichen flexible Querschnittssysteme
- Keine „echte“ Vererbung: Interface-Erweiterung und Delegation

Resümee: Realisierung von gutem fachlichen Design ist eine komplexe Aufgabe

Integration von Fremdsystemen



SOA – serviceorientierte Ansätze

- EAI-Broker: Verschiedene Systeme geben Messages ab, die an andere Systeme weitergeleitet werden
- Kein transaktionale Sicherheit über Systemgrenzen
- aber keine Message geht verloren
- „Zwischen“ fachlichen Teilen von TKeasy kein SOA
 - Transaktionale Sicherheit notwendig
 - Direkte Nutzung der Java-Schnittstellen der Komponenten
- 7 * 24 für Internet notwendig (zumindest lesend)

Massenverarbeitung (Batch)

- Selektion nah an der Datenbank (COBOL / Java auf Großrechner)
- Für jeden relevanten Datensatz Erzeugung einer Message
- Abarbeitung im Application-Server mit hoher Parallelität
 - Damit Geschäftslogik nur in BOs
 - Parallelität muss klappen (Klassiker: Schlüsselvergabe)

Massenverarbeitung frühzeitig bedenken!

Erfahrungen mit Basistechnologien

- SUN Java (J2SE) auf Windows und Solaris
 - Stabil für hochlastige Enterprise Systeme
 - Konfiguration des Garbage-Collectors schwierig (GC-Durchsatz pro Server und Tag: ca. 5 Tera-Byte)
 - 64-Bit: Speicherverbrauch hoch
- SWING
 - Für große Anwendungen mit hohen Ansprüchen an die GUI geeignet
 - Viele Möglichkeiten: für effiziente Entwicklung beschränken
 - Performance gut, Speicherverbrauch nicht klein

Erfolgsfaktoren (I) - Vorgehen

- Kein Big-Bang: Lernen durch schrittweise Umstellung
- Frühe Produktivität = frühes Lernen
- Bereitschaft zu Re-Designs – Erfahrungen auch umsetzen
- Vorgehensmodell als Richtschnur, nicht als Korsett
- Markttrends sehr kritisch würdigen
 - „Bringt es mir wirklich etwas?“
 - „Kann es die Versprechen halten?“
 - „Wird es die nächsten Jahre überleben?“
- Geringe Abhängigkeit von Produkten/Herstellern
- Integration: Anwendungen und Systeme

Erfolgsfaktoren (II) - Technik

- Schlanke Architektur
 - Nur notwendiges tun, keine unnötigen Abstraktionen
 - Wenn Performanceprobleme nicht in DB liegen, stimmt etwas mit der Architektur nicht
- Stabilität durch Architektur sicherstellen
 - Anwendungen dürfen Systemstabilität nicht gefährden können
- Flexibler und performanter Persistenzdienst
- Automatisierte Qualitätssicherung (auf Source- und Byte-Code)
- Leistungsfähige Massenverarbeitung (Batch)
- Gute Technik ist notwendig, gutes fachliches Design auch!

Vielen herzlichen Dank für Ihre Aufmerksamkeit!

Ralf.Degner@tk.de



TK
Techniker Krankenkasse
Gesund in die Zukunft.