TKeasy:
Die unternehmensweite Java-Enterprise-Architektur der Techniker Krankenkasse

Universität Hamburg, 8.11.2011

Ralf Degner, Techniker Krankenkasse

Techniker Krankenkasse
Gesund in die Zukunft.

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-Wrapping / COBOL-Programm-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

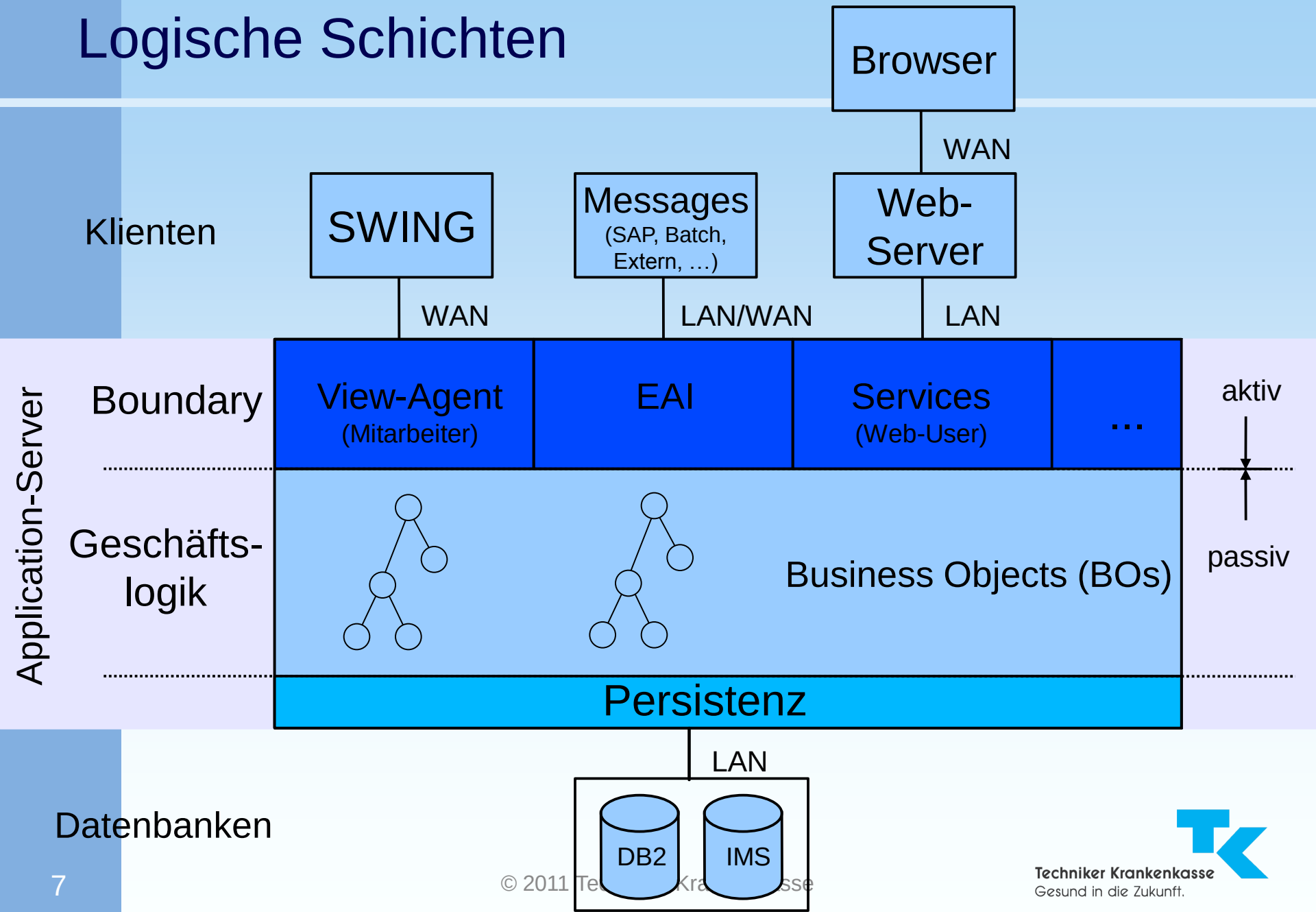
Zahlen pro Tag:

- 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.

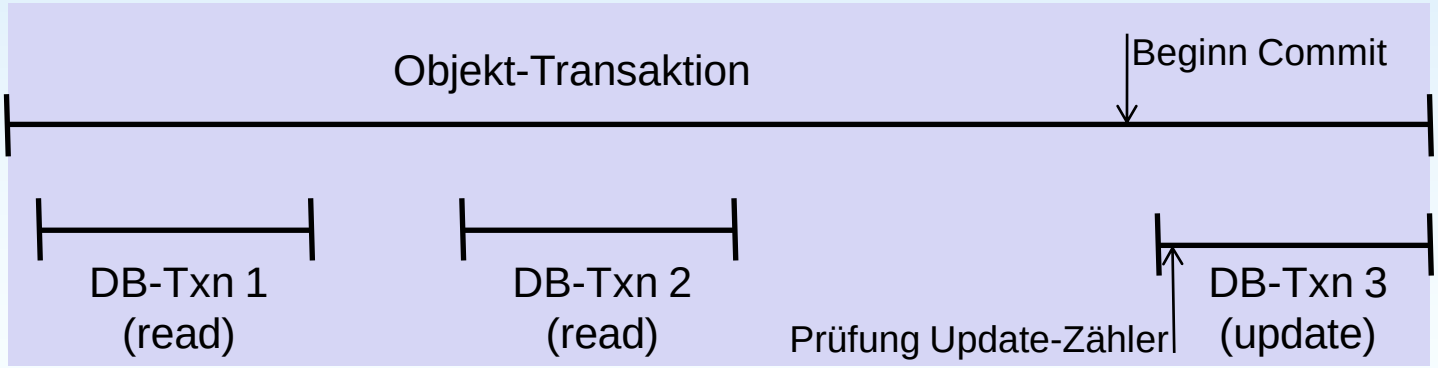
Logische Schichten



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 erfolgt gerade
- 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 entworfenes 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) Werttypen in TKeasy

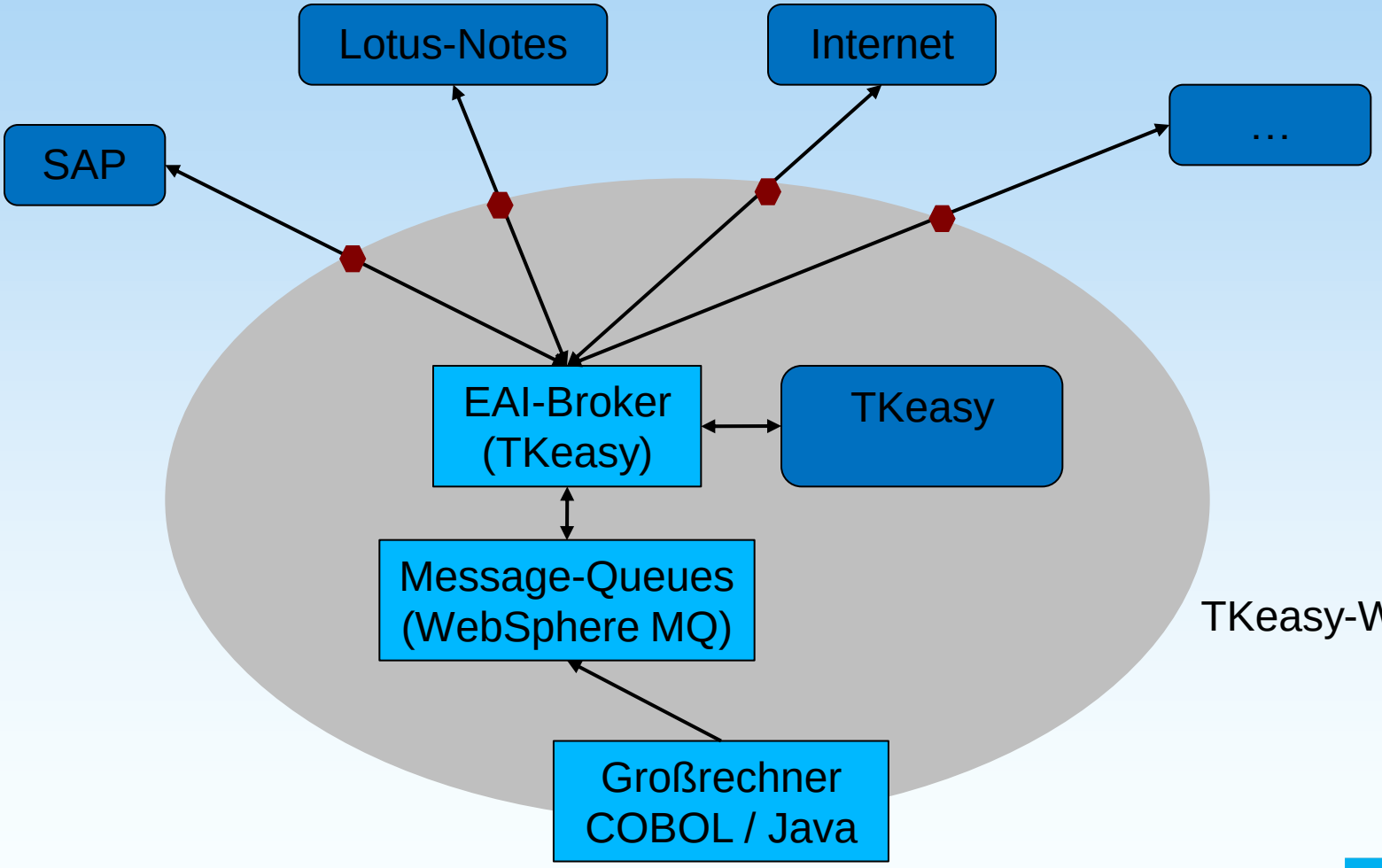
- Viele Werttypen
  - Sind „unkompliziert“, einfache Wiederverwendung
  - Helfen BOs schlank zu halten
- Beispiele: Anrede, Geschlecht, Bankverbindung, Geldbetrag, …
- Eigenschaften
  - Unveränderbar (Tag-Interface de.tk.util.Immutable)
  - Referenzieren Werttypen und „echte“ Identitätstypen (Identität ist unveränderbar)
- Unterstützung durch das Framework
  - Basisklasse: vereinfacht Implementierung von equals
  - Maschinelle Prüfung (beim Build und zur Laufzeit)
  - ImmutableCollections
- Spezialisierungen: TkEnum (Aufzählungen), TkPrimitive (einfach)

Das unternehmensweite OO-Modell (3)

- 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)
- Generische Referenzen ermöglichen flexible Querschnittsysteme
- 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 (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!

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

Techniker Krankenkasse
Gesund in die Zukunft.