Event Services

Gregor Bättig
Fachseminar Verteilte Systeme
21.05.2002

Motivation

- Grössere Software-Systeme häufig als interagierende Komponenten konstruiert
- Viel versprechender Ansatz zur Modellierung der Interaktion ist Verwendung von Events
- Komponenten generieren Events um
  - andere Komponenten über interne Zustandsänderungen zu informieren
  - Services von anderen Komponenten zu verlangen
- Bei Detektion eines oder einer Kombination von Events
  - Ausführen bestimmter Aktionen
  - Generierung weiterer Events
- Event Service ist Infrastruktur, welche solche Interaktionen ermöglicht

Motivation (2)

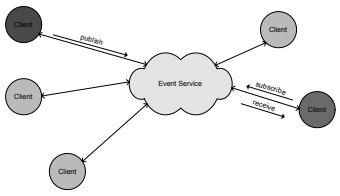
- Komponenten nicht selbst für Übermittlung von Events an andere Komponenten verantwortlich
  - Schicken generierte Events an Event Service
  - Melden Interesse an bestimmten Events an
  - Bekommen "interessante" Events von Event Service zugestellt
- Vorteile gegenüber traditionellem Client/Server Modell
  - Lose Kopplung
    - Eliminiert statische Abhängigkeiten
    - Verbessert Interoperabilität
  - Asynchron
  - Kommunikation 1 : n

Beispiel: Active Badge

- Vor einer Woche Besprochen
- Aufenthaltsort von Personen wird mithilfe von Active Badges registriert
- Bei Entdecken einer Person Event generieren
- System reagiert darauf
  - Aufenhaltsort merken, wenn nötig Telefonanrufe umleiten

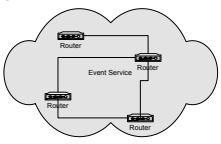
Definition

- Infrastruktur für asynchrones und implizites Verteilen von Informationen von Event-Produzenten an interessierte Konsumenten
- Clients kommunizieren nur mit logisch zentraler Komponente
- Produzenten publishen Events
- Konsumenten subscriben



Skalierbarkeit vs. Ausdrucksstärke

- Skalierbarkeit
  - Fähigkeit eines Event Services, auch in einem WAN mit vielen verteilten Clients und vielen Events zu funktionieren
  - Grössere Distanz zwischen Komponenten als im LAN
    - Bandbreite nimmt ab
    - Latenz wird spürbar
    - Zuverlässigkeit des Netzwerks nimmt ab
  - Heterogene Umgebung, welche sich ständig verändert
  - Dezentralisierte Struktur nötig
- Ausdrucksstärke
  - Strukturierte Events
  - Event Filter
  - Patterns
  - Effizienz nimmt ab
- Beides zusammen schwierig zu erreichen



Event Struktur

- Generisch
  - Data-Blob ohne echte Struktur
  - Schwierig zu filtern
- Typisiert
  - Events mit von Event Service erkennbarer Struktur gelten als typisiert
    - Event Name als String
    - Menge von Strings (Event Name, mehrere Parameter)
    - Menge von Attributen mit Name, Typ, Wert
    - Objektorientiert
  - Pattern matching

Event Filter

- Channel-based Subscription (auch Topic-based)
  - Einfachste Art, Events zu filtern
  - Channel entspricht einem Thema
  - Publisher senden Event an Channel
  - Event Service erkennt am Event bloss Channel Id
  - Subscriber melden sich für einen (oder mehrere) Channel an
  - Event Service leitet Events, die auf Channel ankommen, an alle Subscriber weiter
  - Entspricht dem Abonnieren einer Mailing-Liste

Event Filter (2)

- Subject-based Subscription
  - flexibler als Channel-based
  - Event enthält bekanntes Subject, welches von Event Service erkannt wird und Adresse darstellt
  - Eine Subscription kann sich auf viele Subjects beziehen
    - Ausdruck, der auf die Subjects ausgewertet wird
    - z.B. eingeschränkte Form von Regular Expressions angewandt auf Liste von Strings
      - foo.bar.*.bla*
  - Publisher müssen “mitdenken”
    - Information, die zum Filtern gebraucht werden könnte, muss in Subject stehen

Event Filter (3)

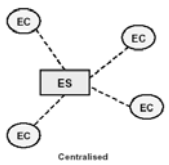
- Content-based Subscription
  - Erweiterung von Subject-based
  - Bezieht sich auf gesamten Inhalt des Events
  - Mit typisierten Events äusserst flexibel
  - Subscriber beschreiben, woran sie interessiert sind
    - bloss vom Inhalt des Events abhängig, nicht davon, was Publisher als Subject definiert

Patterns

- Patterns stellen eine Kombination von Filtern dar
  - AND: zwei Events müssen beide eintreten, damit Subscriber informiert wird
  - OR: einer von zwei Events muss eintreten
  - THEN: temporal, erst der eine Event, dann der andere muss eintreten

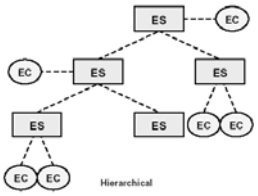
Service Architektur

- Event Server Topologie ist entweder zentralisiert oder verteilt
- Immer logisch zentralisiert
  - Braucht globales Wissen über Clients
- Zentralisiertes Modell
  - Gesamte Kommunikation läuft über via einen Server
  - Skaliert schlecht
  - Single Point of Failure



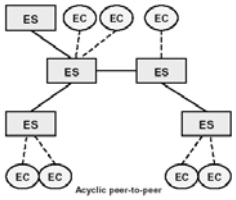
Service Architektur (2)

- Hierarchisches Modell
  - Verteilt
  - Jeder Server ist mit Clients verbunden
    - Publisher/Subscriber
    - anderer Server
    - wird nicht unterschieden
  - Hat Wurzel, durch die aller Verkehr geht
  - Keine Redundanz
    - Steigt ein Server aus ist das System nicht mehr zusammenhängend



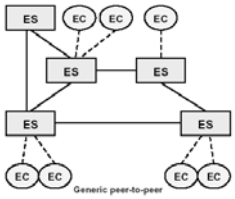
Service Architecture (3)

- Azyklisches Peer-to-Peer Modell
  - Verteilt
  - Server unterscheiden zwischen anderen Servern und Clients
  - Keine Wurzel, durch die aller Verkehr geleitet wird
  - Ebenfalls Baumstruktur
    - Keine Redundanz
    - Störungsanfällig



Service Architektur (4)

- Generisches Peer-to-Peer Modell
  - Verteilt
  - Mit Zyklen
  - Robuster gegen Ausfall einzelner Server
  - Braucht spezielle Algorithmen, um besten Pfad zu finden und Zyklen zu meiden
  - Routen entlang minimalen Spannbäumen

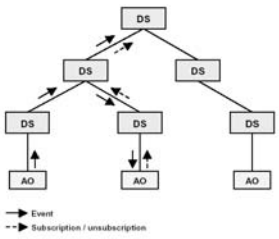


Routing

- Verteilte Topologien
- Ziel: Möglichst wenige Nachrichten verschicken
- Strategien:
  - Event Filter und Event Patterns physisch möglichst nahe beim Publisher anwenden
    - Upstream Monitoring
    - Unnötige Events möglichst nicht weiterleiten
  - Events physisch möglichst nahe bei den Subscribern multicasten
    - Downstream Duplication
    - Solange wie möglich mit einem Event auskommen

Routing (2)

- Beispiel mit hierarchischer Topologie:



Routing (3)

- Interessantes Konzept: Advertisements
  - Erweiterung von Publish/Subscribe
  - Wer Events publishen will, muss diese Absicht erst ankündigen
    - Advertise
    - Definiert die Art Events, die generiert werden
  - Eine Subscription muss dann bloss in die Richtungen von Publishern geroutet werden, welche passende Advertisements verschickt haben
  - Senkt die Anzahl Nachrichten im Allgemeinen, da Advertisement nur einmal verschickt wird

Siena

- Politechnico di Milano
- Unterstützt WAN's
- Terminologie:
  - Publishers = objects of interest
  - Subscribers = interested parties
- Advertise/Publish/Subscribe Modell
  - Subscribe (interested party, pattern)
  - Unsubscribe (interested party, pattern)
  - Publish (event)
  - Advertise (object of interest, filter)
  - Unadvertise (object of interest, filter)
- Funktioniert auch ohne Advertisements

Siena (2)

- Typisierte Events als Menge von Attributen mit Typ, Name und Wert
  - string event = account/debit
  - time date = 21.05.2002
  - float amount = 8430.50
- Notation erinnert an Programmiersprachen
- Event Filter implementiert als Menge von Attribut Namen und Typen plus Einschränkungen bezüglich Wert
  - string event == account/*
  - time date >= 19.12.1977
  - float amount < 3000.00
- Event Patterns evenfalls implementiert

Siena (3)

- Server Topologien
  - Die vier zuvor beschriebenen (logisch zentralisierten) Topologien
    - Zentralisiert
    - Hierarchisch
    - Azyklisch Peer-to-Peer
    - Generisch Peer-to-Peer
  - wurden alle implementiert
  - Bei vielen Clients bewähren sich die verteilten Topologien besser