J Design Patterns für nebenläufige und verteilte Objekte

J.1 Überblick

- Motivation
- Standardprobleme
- Design Patterns für nebenläufige und verteilte Objekte

J.2 Literatur

SSRB04. Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann: Pattern-Oriented Software Architecture. Wiley, Chichester, 2004.

. Douglas Schmidt: http://www.cs.wustl.edu/~schmidt/

J.3 Motivation

★ Typische Probleme bei nebenläufiger und verteilter Software

- Inhärente und unbeabsichtigte Komplexität
  - ➤ bedingt durch Verteilung
  - ➤ bedingt durch Einschränkungen in Werkzeugen und schlechte Unterstützung durch Sprachen
- Nicht-adäquate Methoden und Techniken
  - ➤ berücksichtigen Besonderheiten von Verteilung nicht
- Häufige Neu-Erfindung grundlegender Konzepte und Techniken

J.4 Standardprobleme

- Service Access and Configuration
  - Kommunikation (low level oder durch Middleware unterstützt)
  - Konfiguration (dynamische Änderungen eines Dienstes)
- Event Handling
  - Reaktion auf externe und interne Ereignisse
- Concurrency and Synchronization
  - Strukturierung und Synchronisation von Threads

J.5 Design Patterns

1 Service Access and Configuration Patterns

- Wrapper Facade
  - kapselt Funktionen und Daten von nicht-objektorientierten Schnittstellen in Klassen
  - Beispiel: Management von TCP-Verbindungen durch *handle*-Objekte
- Component Configurator (Service Configurator)
  - dynamisches Binden und Entfernen von Komponenten einer Anwendung
  - Realisierung über DLL und Austausch von Komponenten zur Laufzeit

1 … Service Access and Configuration Patterns

- Interceptor
  - dynamisches Hinzufügen von Diensten und automatische Benachrichtung oder Zwischenschaltung bei bestimmten Ereignissen
  - Beispiel: Security Service, Logging Service
- Extension Interface
  - mehrere Schnittstellen für eine Komponente
  - Bespiel: Standard-Schnittstelle, Debug-Schnittstelle, Administrations-Schnittstelle, ...

2 Event Handling Patterns

- Reactor (Dispatcher, Notifier)
  - Reator nimmt Ereignisse von mehreren Quellen an und verteilt sie kontext-abhängig an registrierte Event-Handler
- Proactor
  - asynchrone Behandlung lang-dauernder Aktionen gekoppelt mit anwendungs-synchronisierter Ergebnisbehandlung
  - nutzt Vorteile von Nebenläufigkeit und vermeidet die Nachteile
- Asynchronous Completion Token
  - automatische Ergebnisbehandlung asynchroner Dienstabfragen
- Acceptor-Connector
  - Entkoppeln des Verbindungsaufbaus zwischen gleichberechtigten Diensten von der nachfolgenden Interaktion

3 Synchronization Patterns

- Scoped Locking (Synchronized Block, Guard, Execute Around Object)
  - atomatisches Belegen und Freigeben eines Locks beim Betreten bzw. Verlassen eines Abschnitts – unabhängig vom Pfad beim Verlassen
- Strategized Locking
  - Parametrierbarer Synchronisationsmechanismus
- Thread-Safe Interface
  - Locking durch Schnittstellen-Methoden an der "Grenze" einer Komponente
  - Komponenten-interne Aufrufe arbeiten ohne Locking
- Double-Checked Locking Optimization
  - Thread "merkt sich" welche Locks er schon hat

4 Concurrency Patterns

- Active Object
  - entkoppelt Methodenaufruf durch Client-Thread von Methodenausführung in Server-Thread(s)
- Monitor Object
  - vermeidet Nebenläufigkeit innerhalb eines Objekts
- Half-Sync/Half-Async
  - Vermittlung zwischen synchronen Dienstaufrufen "von oben" und asynchrone Ereignissen "von unten"
- Leader/Followers
  - Menge von Threads wartet auf Ereignisse. Leader-Thread nimmt Ereignis auf und bearbeitet es, der erste der Follower wird zum neuen Leader
- Thread-Specific Storage
  - globaler Name für Thread-lokale Daten (z. B. errno)