


Catalysts Components

DI Klaus Gieber
Catalysts GmbH



Geschichte

- Seit April 2008 arbeiten wir an taskmind (www.taskmind.net)
 - Gewisse Architekturüberlegungen haben wir aus den Projekten davor übernommen
 - Klaus Gieber und Christian Federspiel von großen Systemen bei der Siemens/VAI
 - Harald Radi von www.modernfamilies.net
 - Christoph Steindl von großen Kundenprojekten
 - Vieles haben wir seither verbessert
- Die wiederverwendbaren Teile haben wir herausgezogen von taskmind in die Catalysts Components
 - als Basis für taskmind, aber auch
 - als Basis für diverse Kundenprojekte

© www.catalysts.cc, 2009 Catalysts Components 2

Was sind Catalysts Components? Catalysts

- **Catalysts Components** bauen auf bewährten **Open Source-Frameworks** auf – wir entwickeln nichts neu, was es schon gibt.
- Wir haben **die richtigen Open Source-Bausteine ausgewählt** und sie **richtig zusammengesetzt** – damit man sich darauf verlassen kann und alles funktioniert.
- Wir haben **wesentliche Funktionen hinzugefügt** – damit man sich aufs Wesentliche fokussieren kann.

Client (C#, Flex, Air)	Server (Java)	Business Code
SmartListView, Context Menu, Connection Handling, Auto-Reconnect, Event Queuing & Data Synchronization, Proxy & Cache	Session Handling, Authentication, Scheduling, Validation, Natural Language Extraction, Testable DAO Layer, API Layer	Catalysts Components
Red5, Hessian, Serialization, Server Push		
PureMVC	Web Server, Spring, Hibernate, JPA, Doclets	Open Source

© www.catalysts.cc, 2009 Catalysts Components 3

Vorteile Catalysts

- Die Catalysts Components sind
 - erprobt – in www.taskmind.net und in etlichen Projekten von Catalysts im Einsatz
 - verfügbar
 - modular einsetzbar – eine, mehrere oder alle Komponenten
 - testgetrieben entwickelt und ausführlich getestet
 - flexibel – die Catalysts Components können mit diversen Open-Source- und Drittprodukten verwendet werden
- Man muss nicht mehr selbst
 - alle Open Source Frameworks evaluieren
 - die richtigen auswählen
 - sie integrieren
 - sie in den Entwicklungsprozess einbetten
 - die Verwendungsvorgaben für die Entwickler erstellen
- Man kann sich auf die Geschäftslogik konzentrieren und muss sich nicht mehr so stark mit der Plattform / den Frameworks beschäftigen.

© www.catalysts.cc, 2009 Catalysts Components 4

Open Source Fallen

Catalysts

- Wenn man mehrere Open Source Frameworks (OSF) gemeinsam einsetzt, treten im Zusammenspiel oft „unerwartete“ Szenarien auf
 - Wenn man ein einzelnes OSF einsetzt, tritt die Situation nicht auf
 - Im Zusammenspiel mehrerer OSFs aber schon
- In diesen unerwarteten Szenarien
 - Werden oft Methoden aufgerufen, die zwar vorgesehen sind, aber noch nicht implementiert sind
 - Treten aber auch einfach Fehler auf, die noch nicht entdeckt wurden
- So, wie wir die Open Source Frameworks zusammengefügt haben, funktionieren sie sehr gut.
- Bis dorthin war es aber viel Arbeit, weil wir etliche Fehler in Open Source Frameworks aufgezeigt, selbst behoben und zurückgemeldet haben.
- Wenn man bei Null beginnt, wird man auch in diese Open Source Fallen tapen.

Funktionaler Vorteil

Catalysts

- Wertvolle Controls für Flex
- Programmiersprachenunabhängige Kommunikation (C#, Actionscript, Java)
- Generierung der (clientseitigen) Schnittstelle in Actionscript und C# aus der (serverseitigen) Java-Schnittstelle (inkl. Kommentare)
- Authentifizierungs-Framework
- Automatischer Verbindungsaufbau (mit automatischem Reconnect)
- Server-Push
- Validierungs-Framework
- Unterstützung für Laufende Integration
- Testfälle
 - JUnit am Server
 - Datenbank-Testfälle
 - JUnit, FUnit, NUnit am Client
- Beispielprogramme

Features: Client Catalysts

- Die Catalysts Components unterstützen am Client:
 - Browserfähige RIA-Anwendungen
 - Installierbare Clients (AIR)
 - Mobile Endgeräte (Windows Mobile)
 - Objektbasierte Kommunikation mit dem Server (Übertragung von IDs statt ganzer Objekte, wo möglich)
 - Automatisches Wiederherstellen der Verbindung zum Server
 - Notifizierungen vom Server („server push“ wie bei Blackberry)
 - Ausnahmebehandlung (auch mit Weiterleitung der Exceptions zwischen Client und Server)
 - Klare Vorgaben für Model-View-Controller

© www.catalysts.cc, 2009 Catalysts Components 7

Features: Server Catalysts

- Die Catalysts Components unterstützen am Server (erfordern aber nicht):
 - Klare Architektur-Vorgaben für den serverseitigen Code
 - Persistierung mit JPA und Hibernate
 - Transaktionen mit Spring
 - Validierung
 - Dependency Injection mit Spring
 - Security und Logging mit AOP und Spring
 - Server Push (Event-Queuing und Daten-Synchronisierung)
 - Mediendaten wie Fotos und Videos (Download und Streaming in beide Richtungen)
- Voraussetzung:
 - für Flex-Clients genügt ein beliebiger Servlet Container
 - für Server Push zu C#-Clients wird Apache Tomcat 6.0 benötigt

© www.catalysts.cc, 2009 Catalysts Components 8

Was war uns wichtig? (1/2) Catalysts

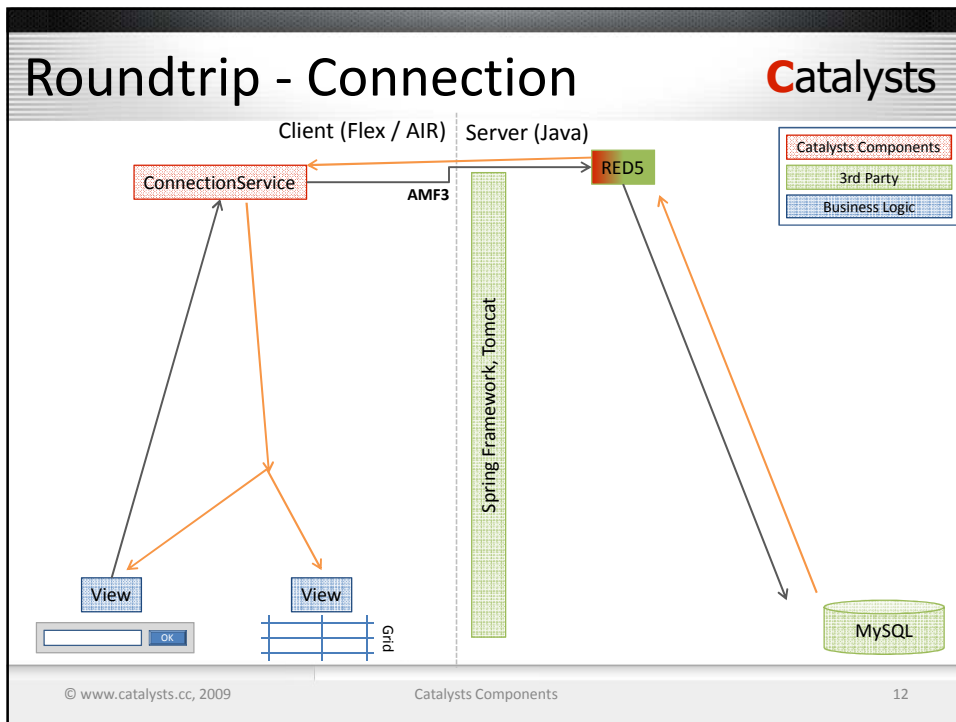
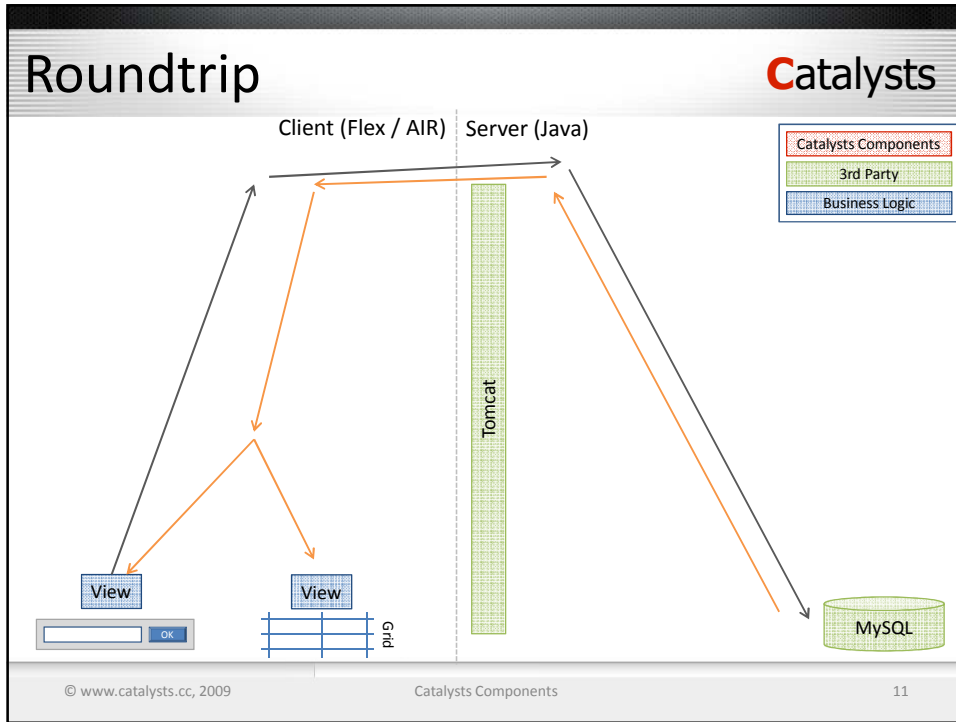
- Klare Trennung zwischen Client und Server → Integrierbarkeit, Portabilität, Ersetzbarkeit des Clients
 - Möglichst schlanke Clients
 - Möglichst viel Logik am Server (für alle Clients gleich), dadurch Konsistenz über alle Clients hinweg
 - Clients in mehreren Programmiersprachen möglich (Java, C#, Flex/Actionscript)
 - Client-Updates während laufendem Server möglich
 - Mit typisierter Schnittstelle – sauberes API mit Methoden und Value-Objects (auch über Programmiersprachen hinweg)
 - Effiziente Kommunikation zwischen Client und Server (mit Delta-Übertragung)
- Saubere Software-Architektur → Erlernbarkeit, Erweiterbarkeit, Wartbarkeit
 - Am Client und am Server
 - Genaue Vorgaben an Entwickler (welcher Code wo hin gehört)
 - Client-Entwickler brauchen kein Server-Wissen
 - Server-Entwickler brauchen kein Client-Wissen
 - Dadurch leichtere Skalierung des Teams
- Testgetriebene Entwicklung → Qualität
 - Serverseitige Testfälle (in jeder Schicht der Server-Architektur)
 - Clientseitige Testfälle (in jeder Schicht der Client-Architektur)

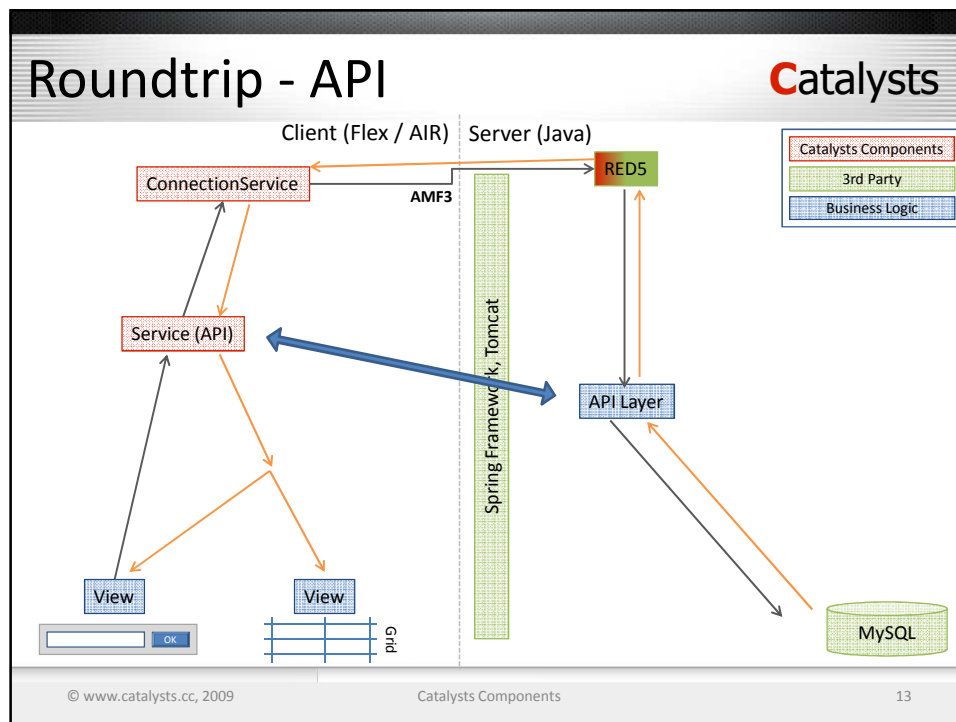
© www.catalysts.cc, 2009 Catalysts Components 9

Was war uns wichtig? (2/2) Catalysts

- Flexibilität → Unabhängigkeit
 - Unterstützung verschiedener Entwicklungsumgebungen (IntelliJ, Eclipse, Visual Studio), dennoch automatischer Build
 - Minimale Anforderungen an Container (einfacher Servlet-Container und einfache Datenbank reichen aus)
 - Austauschbarkeit einzelner Komponenten
 - Modulares System
- Auswertungen und Monitoring → Qualitätskontrolle und fokussierte Qualitätsverbesserung
 - Unterstützung für Continuous Build
 - Automatische Regressionstests
 - Automatische Code-Analysen und Architektur-Analysen
- Offline-Fähigkeit der Clients → zwischenzeitliche Verbindungsprobleme überleben
 - Automatischer Re-Connect
 - Lokale Ereignis-Warteschlange
 - Automatische Datensynchronisierung
- Server-Push → alle Clients zeigen „ohne Refresh“ immer die aktuellen Daten an

© www.catalysts.cc, 2009 Catalysts Components 10





Generierung von Bridge-Code Catalysts

- Daten werden bei den Catalysts Components zwischen Server und Client als „Value Objects“ übertragen.
- Dadurch bleibt auch über das Netz die Typsicherheit erhalten; Änderungen an der (serverseitigen) Schnittstelle führen sofort zu Kompilierfehlern bei Client-Code.
- Problem: Der Server läuft in Java, die Clients jedoch in ActionScript oder C#
- Lösung: Aus den Java-Klassen erzeugen wir automatisch ActionScript und C#-Klassen
- Das funktioniert auch für Interfaces; Schnittstellenbeschreibungen (Javadoc) werden auch automatisch in die Zielsprache portiert.

© www.catalysts.cc, 2009 Catalysts Components 14

Generierung von Bridge-Code Catalysts

Java

```

package cc.catalysts.cp.admin {
/**
 * @actionscript.class annotation=Bindable
 * @cs.class
 */
public class ClientSession {
    private Date since;
    private String remoteAddress;
    public ClientSession(Date since, String remoteAddress) {
        super();
        this.since = since;
        this.remoteAddress = remoteAddress;
    }
    public Date getSince() {
        return since;
    }
    public void setSince(Date since) {
        this.since = since;
    }
    public String getRemoteAddress() {
        return remoteAddress;
    }
    public void setRemoteAddress(String remoteAddress) {
        this.remoteAddress = remoteAddress;
    }
}
                
```

C#

```

namespace cc.catalysts.cp.admin {
    public class ClientSession {
        private System.DateTime? since;
        private string remoteAddress;
        public virtual System.DateTime? Since {
            get { return this.since; }
            set { this.since = value; }
        }
        public virtual string RemoteAddress {
            get { return this.remoteAddress; }
            set { this.remoteAddress = value; }
        }
    }
}
                
```

ActionScript

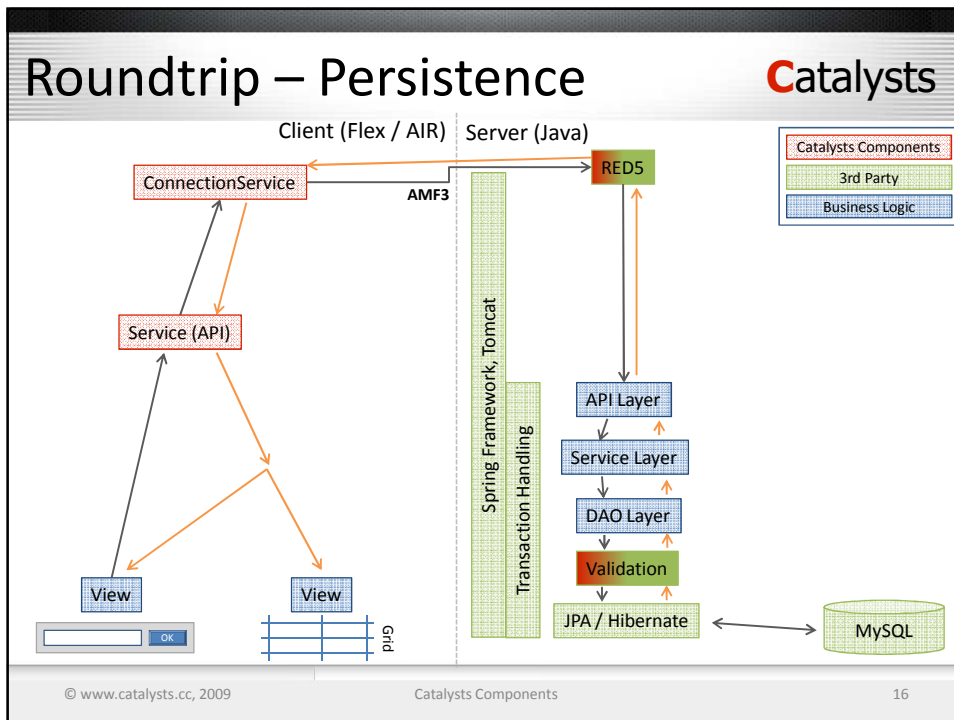
```

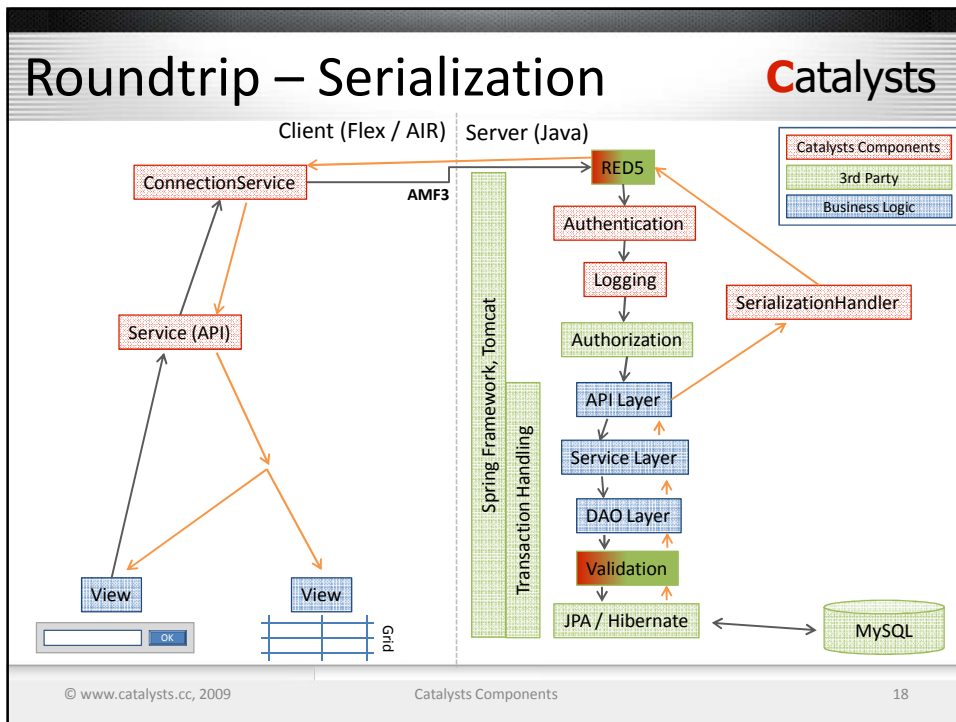
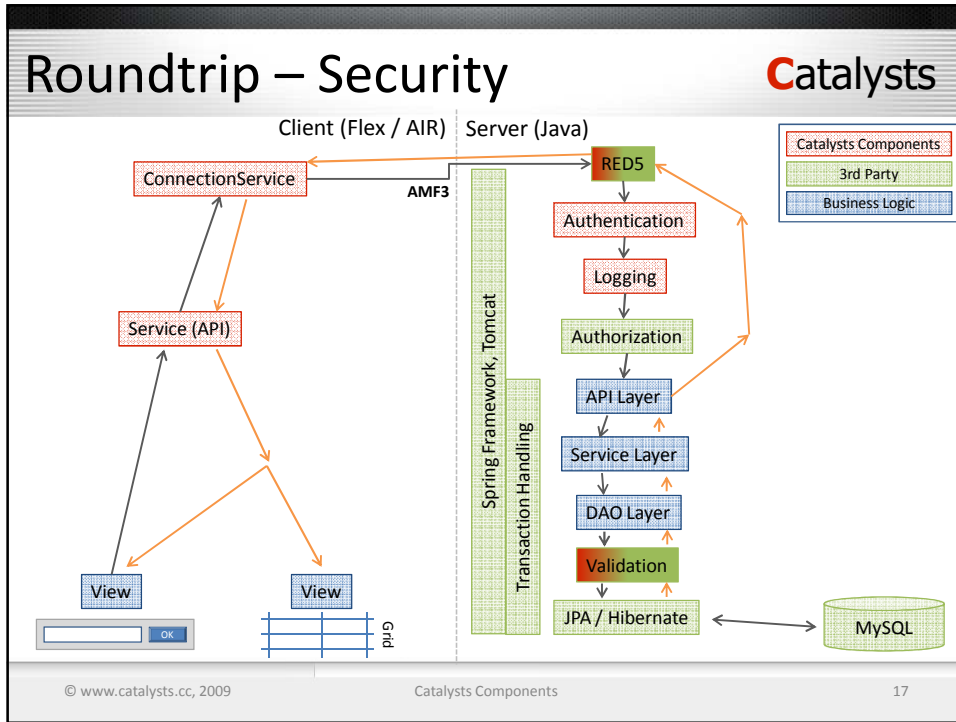
package cc.catalysts.cp.admin{
[RemoteClass(alias="cc.catalysts.cp.admin.ClientSession")]
[Bindable]
public class ClientSession {
    private var _since:Date;
    public function get since():Date {
        return _since;
    }
    public function set since(value:Date):void {
        _since=value;
    }
    private var _remoteAddress:String;
    public function get remoteAddress():String {
        return _remoteAddress;
    }
    public function set remoteAddress(value:String):void{
        _remoteAddress=value;
    }
}
                
```

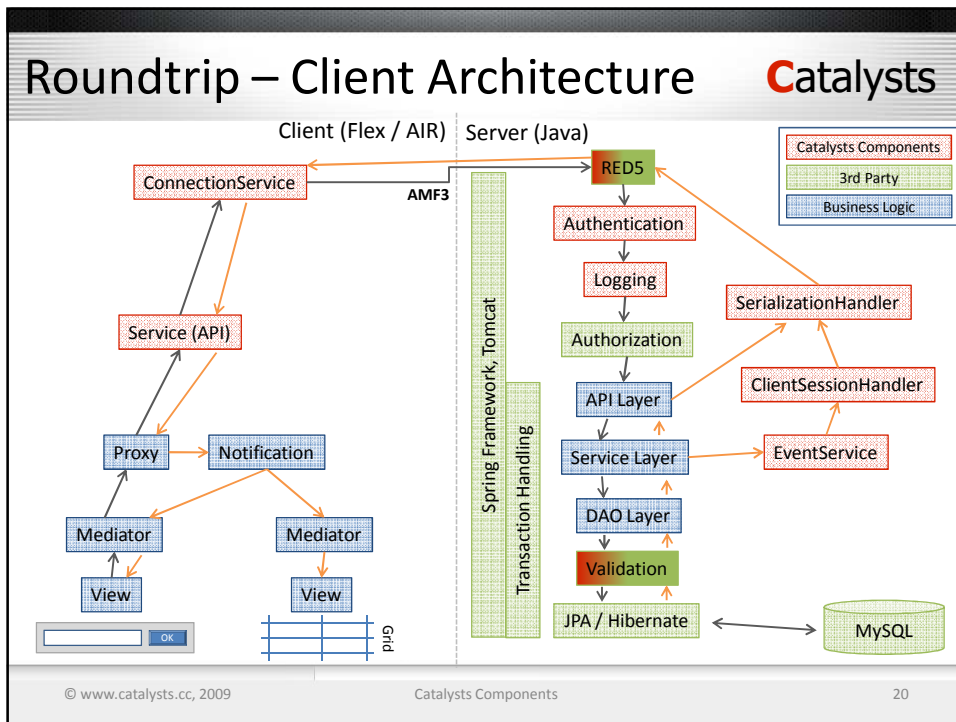
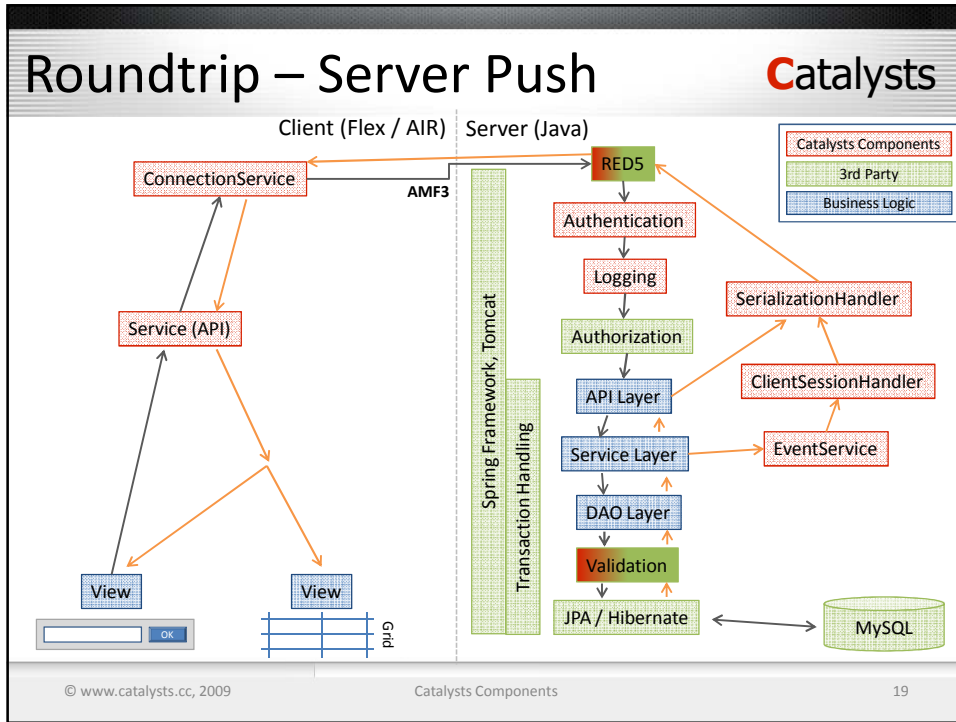
© www.catalysts.cc, 2009

Catalysts Components

15







Entwicklungsumgebung Catalysts

- Optimiert auf Flexbuilder, Eclipse, IntelliJ IDEA und MS Visual Studio.NET
- Automatische Projektgenerierung
 - Aus wenigen Inputparametern (Projektname, benötigte Module,...) wird automatisch eine vollständige Projektstruktur erstellt.
- Automatisierter Buildprozess mit ANT und Teamcity
 - Generieren der Bridge-Klassen
 - Kompilieren
 - Deployment
 - Testfälle
 - Dokumentation und Statistiken
- Verwaltung der Modulabhängigkeiten mit Apache IVY
 - Automatische Einbindung von Quellcode von Fremdbibliotheken in die Entwicklungsumgebung
- Gesamte Konfiguration im gemeinsamen Repository abgelegt.
 - Lokales Überschreiben möglich

© www.catalysts.cc, 2009 Catalysts Components 21

Zusammenfassung Catalysts

- Wichtigste verwendete Technologien:
 - Entwicklungsumgebung, Infrastruktur:
 - ANT, Ivy, Velocity, Eclipse, IntelliJ IDEA
 - Programmiersprachen:
 - Java, Actionscript, C#
 - Server:
 - Spring, Red5, Hessian, Velocity, JPA, Hibernate
 - Client:
 - PureMVC, Addin Express (Outlook), iLog (Actionscript)

© www.catalysts.cc, 2009 Catalysts Components 22