Einleitung: Integrationsvorhaben

Anwendungslandschaften

- historisch gewachsene Strukturen, unkontrollierte Redundanz, Heterogenität
- Vision: integriertes Informationssystem

Anforderungen an einen Lösungsansatz

- wirksame Reduktion von Komplexität
  - Fokussierung auf wesentliche Integrationsaspekte
  - weitgehende Vernachlässigung spezieller, zeitlich variierender Eigenschaften (Technologie …)
  - übersichtliche Darstellung auch komplexer Zusammenhänge
- Unterstützung einschlägiger Analysen
  - Bereitstellung geeigneter Konzepte und damit korrespondierender Vorgehensweisen

Skizze einer dedizierten Modellierungsmethode

- Modellierungssprachen/Diagrammarten
  - Sprache zur statischen Modellierung von IT-Landschaften
  - korrespondierende Sprache zur Modellierung von GPs
- Vorgehensmodelle für unterschiedliche Arten von Integrationsprojekten, z.B.
  - kontextunabhängige oder prozessbasierte Analyse der IT-Landschaft
- Unterstützung bei der Auswahl geeigneter Technologien

exemplarisches Vorgehensmodell

1) Klärung der IT-Strategie
2) Modellierung der relevanten Prozesse
3) Beschreibung der beteiligten Systeme
4) Analyse des aktuellen Integrationsniveaus
5) Analyse des Integrationsbedarfs
6) Erstellung eines Integrationsplans

Grundlagen serviceorientierter Architekturen (SOA)

Motivation: idealtypische Szenarien

- Integration existierender Altanwendungen
  - Altanwendungen müssen an neue Geschäftsideen angepasst werden
  - Neuentwicklungen unterliegen wirtschaftlichen Restriktionen
  - dynamische Integration mittels Prozessmodellen
- dynamische Adaption volatiler GPs
  - Ausführung von GPs wird durch IS unterstützt
  - wechselnde Anforderungen an GPs müssen sich in IS widerspiegeln

Integration von Anwendungen – Zusammenfassung

- Kontext: existierende Anwendungen
- Ziel: Integration der Anwendungen im Hinblick auf GPs
- Restriktionen
  - Neuentwicklung aller Anwendungen zu aufwändig
  - vollständiger Ersatz durch Standardsoftware nicht realistisch
- Integration mittels Schnittstellenbeschreibungen für Dienste, ggfs. Ablaufbeschreibungen

volatile GPs – Zusammenfassung

- Kontext: Prozesse eines Unternehmens verändern sich ständig
- Ziel: IT muss an geänderte Anforderungen angepasst werden
- Restriktionen
  - hoher Aufwand für Anpassungen klassischer SW-Systeme
  - Entwurfsmodelle für SW nicht vollständig kompatibel mit GPs

Idee serviceorientierter Architekturen

- Komponentenorientierung
  - Komponenten kommunizieren über definierte (standardisierte) Schnittstellen (Service)
  - Unabhängigkeit von der Implementierung, d.h. verschiedene Anwendungen können den gleichen Service anbieten
- lose Kopplung: asynchrone (robuste) Kommunikation
- dadurch möglich: adaptierbare Ablaufbeschreibungen

Web Services

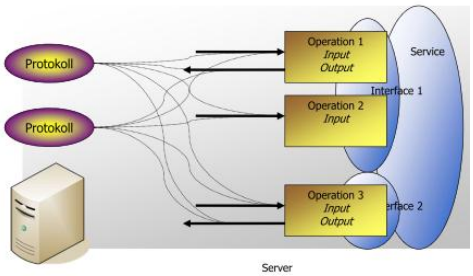
- Umsetzung der Idee eines Service
- mit Hilfe von Web-Technologien (TCP/IP, HTTP, XML, …)
- Implementierungsperspektive

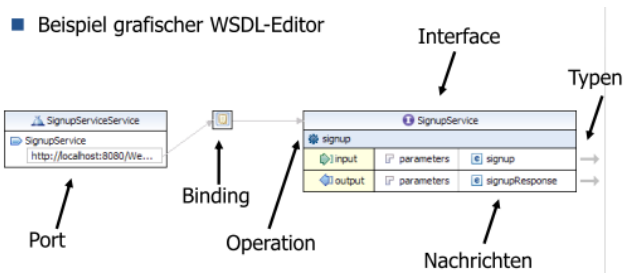
Web Service Implementierung

- Fragen, die sich bei Analyse/Design stellen
  - Welche Services gibt es? (Namen der Services)
  - Welche Operationen bieten sie an? (Namen der Operationen)
  - Welche Nachrichten erhalten und senden Operationen? (Typen der Nachrichten)
- wesentlich für die technische Implementierung: maschinenverständliche Nachrichten (→ Datenformat)
- davon konnte auf Analyse/Design-Ebene abstrahiert werden, für Implementierung aber zentral
- Service muss für Implementierung formalsprachlich beschrieben werden
- ein „Verständnis“ von der Arbeit eines Service kann nicht maschinenlesbar vermittelt werden
  - Semantik ist interpretationsbedingt, Maschinen interpretieren aber nicht
- aber zumindest möglich: detaillierte Formalisierung des Nachrichtenaustauschs
- auf der Ebene der Implementierung wird die Frage, was ein Service ist, durch Spezifikation der Nachrichten beantwortet, die er empfangen kann und sendet

Modell eines Web Service

- Konzeptionelle Komponente eines Service („Idee“ des Service)
  - Port-Type/Interface, Operation, Input/Output, Protokoll-Binding, Nachricht, Typ eines Nachrichten-Elements





pragmatische Einordnung von SOA „top-down“

- Entwurf und Strukturierung von Anwendungslandschaften
- Ausrichtung der Anwendungslandschaft auf das Geschäft, d.h. auf die GPs → Brücke zwischen Geschäft und IT

pragmatische Einordnung von SOA „bottom-up“

- SOA Kernelemente: Dienste/Services, Nachrichten/Messages
- Fokus liegt auf softwaretechnischen Diensten, deren Schnittstellen idealerweise durch Web Services beschrieben sind
- relevant sind mehrere XML-basierte Standards im Kontext von Web Services
  - Schnittstellendefinition: WSDL, Nachrichtendefinition: SOAP, Verzeichnisdienst: UDDI

Konzeptionalisierung des Servicebegriffs

- Service
  - keine überzeugende Definition
  - abstrahiert auf Schnittstellen und Parameter
  - abstrahiert von technischen Implementierungen
- Web Service (in Anlehnung an das W3C)
  - eindeutig identifizierbares Software System (URI)
  - veröffentlichte Schnittstelle, beschrieben in XML
  - nachrichtenbasierte Kommunikation mit anderen Software Systemen durch i.d.R. standardisierte Protokolle
  - ein Web Service ist doch einen sog. Agenten implementiert
  - ein Agent ist eine austauschbare SW oder HW

weitere Charakteristiken

- Seiteneffekte
- lose Kopplung
  - es liegen lediglich Informationen über Schnittstellen sowie über Aufruf- und Rückgabeparameter vor
  - Kommunikationspartner sind austauschbar
  - Kommunikation geschieht asynchron über Nachrichten

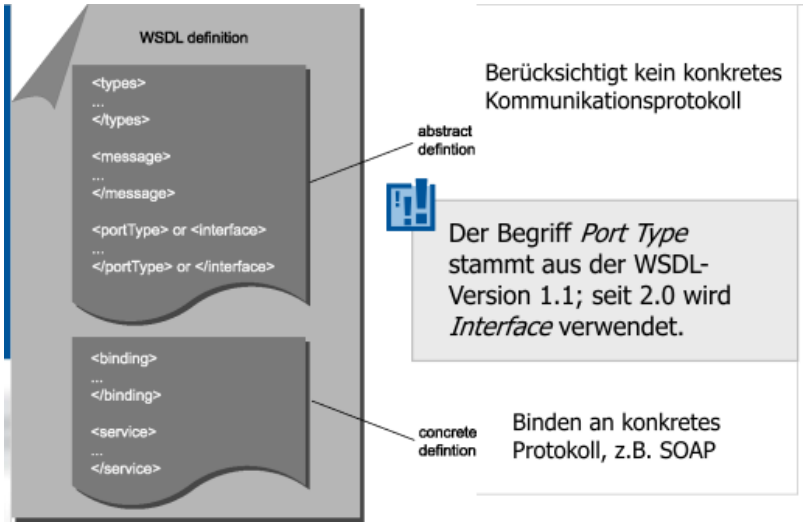
ESB – exemplarische Funktionalitäten

- Routing: Weiterleitung von Nachrichten an Empfänger
- Persistenz: Nachrichten gehen nicht verloren
- Transformation: Bruch zwischen unterschiedlichen Begriffswelten und unterschiedlichen Kommunikationstechniken wird vor den Anwendungen verborgen
- Workflow: ESB übernimmt Ablaufsteuerung
- Fehlerbehandlung und Monitoring, Lastverteilung, Ausfallsicherheit und Skalierbarkeit

Vorgehensmodell in der Übung

1) Modellierung der relevanten Prozesse
2) Beschreibung der beteiligten Prozesse
3) Identifikation von Services
4) Generierung und Verfeinerung von Web Services und Ablauflogik
5) Deployment und Test
6) Analyse des erreichten Integrationsniveaus

Schnittstellendefinitionen und Nachrichtenaustausch



WSDL – Wurzelelement

- oberstes Element ist *definitions*, listet im wesentlichen Namensräume

```
<definitions name="Employee"
    targetNamespace="http://www.xmltc.com/tls/employee/wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    ...
</definitions>
```

WSDL – Typdefinitionen

- Definition mittels XML-Schema

```
<types>
    <schema
       xmlns=http://www.w3.org/2001/XMLSchema
       targetNamespace="http://.../transform/schema/">
       <complexType name="ReturnCodeType">
           <sequence>
              <element name="Code" type="xsd:integer"/>
              <element name="Message" type="xsd:string"/>
           </sequence>
        </complexType>
    </schema>
</types>
```

WSDL – Nachrichten

- Definition aller zu empfangenden und sendenden Nachrichten
- werden mittels Operationen ausgetauscht

```
<message name="getEmployeeWeeklyHoursRequestMessage"
    <part name="RequestParameter" element="act:EmployeeHoursRequestType"/>
</message>
<message name="getEmployeeWeeklyHoursResponseMessage"
    <part name="ResponseParameter" element="act:EmployeeHoursResponseType"/>
</message>
<message name="getId"
    <part type="xsd:integer"/>
</message>
```

WSDL – Dienstoperationen

- Operationen werden in einem Port Type definiert
- Input- und Output-Elemente charakterisieren die Operation

```
<portType name="EmployeeInterface">
   <operation name="UpdateHistory">
      <input message="tns:updateHistoryRequestMessage"/>
      <output message="tns:updateHistoryResponseMessage"/>
   </operation>
   <operation name="Submit">
      <input message="tns:receiveSubmitMessage"/>
   </operation>
</portType>
```

WSDL – Binden einer Operation an SOAP

- Verbinden der abstrakt deklarierten Operationen und Nachrichten mit konkretem Übertragungsprotokoll

```
<binding name="EmployeeBinding" type="tns:EmployeeInterface">
   <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="UpdateHistory">
      <soap:operation soapAction="..."/>
      <input>
         <soap:body use="literal"/>
      </input>
      <output>
         <soap:body use="literal"/>
      </output>
   </operation>
</binding>
```

WSDL – Kommunikationsendpunkte

- Web-Adresse, Zugriff über verbundene Protokolle
- *ports* heißen seit WSDL 2.0 *endpoints*

```
<service name=„EmployeeService"
   <port binding="tns:EmployeeBinding" name="EmployeePort">
      <soap:address location="http://www.xmltc.com/tls/employee/"/>
   </port>
</service>
```

WSDL – weitere Konzepte

- *import*-Element
  - mittels des Schlüsselwort *import* können andere WSDL- oder XMLSchema-Dokumente importiert werden
- *documentation*-Element
  - Annotation von Kommentaren
  - textuelle, von Maschinen lesbare Formen
  - kann beliebigen Elementen zugeordnet werden

Binding

- „anbinden“ an Implementierung
- konkret (nur): wie wird implementiert?
  - Datenformate, Übertragungsprotokolle, …
- viele technische Kleinigkeiten
- vieles automatisierbar

- 2 Binding-Varianten üblich: SOAP, HTTP
  - mehr Varianten denkbar (z.B. RPC, JSON), aber nicht üblich
- SOAP
  - (ältere) Technik u.a. zum Aufruf von Objekt-Methoden via XML
  - „not really simple“, „no object access“, „not a protocol”
  - Ein- und Ausgabedokumente eingebettet in SOAP-Umschlägen
- HTTP
  - Bereitstellen von dynamischen parametrisierbaren Ressourcen via Internet
  - andere Idee als SOAP, zum Implementieren von Web Services aber auch geeignet
  - Eingabe muss für Implementierung als HTTP-Parameter codiert werden, Ausgabe als XML

SOAP: Struktur einer Nachricht

- Header
  - Correlation Information: eindeutige ID, welche ein *response* einem *request* zuordnet → Simulation von Verbindungen über verbindungsloses HTTP
  - außerdem möglich: Information über Routing oder Verarbeitung einer Nachricht
- Body
  - Nachrichteninhalt
  - Fehlernachrichten im *fault*-Element

Nachrichtenaustausch SOA: Muster

- Message Exchange Patterns (MEP)
  - request-response operation pattern (requestor → provider → requestor)
  - one way operation pattern (requestor → provider)
  - solicit-response operation pattern (provider → requestor → provider)
  - notification operation pattern (provider → requestor)
  - komplex: publish-and-subscribe (request-response + notifications)
- beschreiben typische Kommunikationsstrukturen zwischen Diensten
  - als Vorlage für den Entwurf der Kommunikation in einer SOA
  - allerdings keine Synchronisation
- basieren auf
  - Austausch zwischen Dienstnehmer und -anbieter
  - Versenden von Nachrichten
- höheres Abstraktionsniveau
  - zur Beantwortung der Frage „wie kommunizieren Services?“
  - aber nicht aufgegriffen in WSDL

UDDI: Universal Description Discovery and Integration

- Standard für die Beschreibung und das Auffinden von Web Services → „Registry“
  - Beschreibungsstruktur für Web Services
  - API für die Verwaltung und das Retrieval von Schnittstellen
- UDDI ist selbst wiederum als Web Service konzipiert
  - ein konkreter UDDI-Dienst kann ebenfalls in einer Registry verwaltet werden

WSDL & SOAP: kritische Würdigung

- WSDL & SOAP sind etablierte Standards im SOA-Umfeld
  - dennoch keine prinzipielle Einschränkung auf diese Protokolle

  - Standards werden in einer großen Gemeinschaft evaluiert und definiert
- Betonung von loser Kopplung und asynchronem Nachrichtenaustausch
  - erleichtert Integration neuer Dienste
  - fördert verteilte Systeme auf der Basis unterschiedlicher Softwaremodule
- aber:
  - niedriges semantisches Niveau bei der Integration
  - Tendenz zu Redundanzen

Web Service Engineering

- top-down
  - Service-Beschreibung → SW-Funktionen (z.B. WSDL → Java)
  - unabhängige Planung einer SOA: welche Dienste existieren, welche Funktionen bieten sie an?
  - bei Neuentwicklungen oder „theoretisch sauberem“ Interfacing zu Legacy-Systemen
- bottom-up
  - SW-Funktionen → Service-Beschreibung (z.B. Java → WSDL)
  - „Export“ von bestehender Legacy-Funktionalität
  - Vorsicht: Dienste-Orientierung nicht immer direkt auf bestehenden APIs aufsetzbar
- Entscheidungen auf Analyse-Niveau
  - Wird eine Anwendungslandschaft neu entwickelt? → top-down
  - Sollen bestehende Legacy-Anwendungen integriert werden? → (tendenziell) bottom-up
  - ganze Dienste sollten nicht bottom-up generiert werden, höchstens einzelne Operationen
  - die Frage, welche Dienste im Rahmen einer SOA sinnvoll sind, ist Kernfrage bei der Analyse und ergibt sich nicht von selbst aus den Legacy-Komponenten

Web Service Deployment

1) WSDL-Beschreibung verfügbar machen
   - im einfachsten Fall als statische Webseite, z.B. *http://mysite.org/files/service.wsdl*
2) SW des Dienstes auf Server installieren
   - z.B. Java-Programm/-Komponente:
   - im einfachsten Fall CLASSPATH ergänzen
   - größere Projekte: Enterprise Java Beans installieren
   - alternativ: bestehendes Legacy-System, proprietäre Wrapper
3) als Web Service aufrufbar machen
   - üblicherweise als dynamische Webseite via HTTP, z.B. *http://mysite.org/servlet/MyService*
   - Web Service-Handler vermittelt zwischen Protokoll und Programm-Komponente
   - Übergabe und Rückgabe von Parametern wählbar, je nach Protokoll-Binding
   - üblich: SOAP (Header, Envelope, …)
   - auch: simple HTTP-Parameter, z.B. *http://mysite.org/servlet/MyService?id=0815&order=1*

zustandsbehaftete Services

- Services müssen i.d.R. Datenhaltung betreiben können
- insbesondere betriebswirtschaftlich / organisatorisch motivierte Services
  - Stammdaten, Transaktionsdaten, …
- Thema Datenhaltung unabhängig von (Web) Services, betrifft SW allgemein
- technische Umsetzungsmöglichkeiten
  - Textdateien, RDBMS, Objekt-Zustände über Persistenz-Framework

Beschreibung von Geschäftslogik in SOA

konkurrierende Technologien (Auswahl)

- WfMS: BPEL vs. XPDL
- Kommunikationsmiddleware: CORBA, .NET, J2EE
- entfernte Funktionsaufrufe: RMI, Distributed Computing Environment (DCE)

SOA – Verheißungen

- Potentiale durch standardisierte Infrastruktur
  - Kommunikationsinfrastruktur
  - standardisierte Schnittstellen
  - ggfs. standardisierte Beschreibung von Diensteigenschaften
- Wiederverwendung
  - SW kann durch Rückgriff auf existierende Dienste realisiert werden
  - ggfs. weltweit wachsende Bibliothek von Daten
- Integration
  - existierender Anwendungen
  - vor allem: unternehmensübergreifende Integration zur Unterstützung von Kooperationsprozessen (z.B. AmazonWebServices)

Choreography & Orchestration – Überblick

- populäre Begriffe im SOA-Umfeld
- Bedeutung in Alltagssprache annähernd synonym
- Differenzierung in SOA eher subtiler Art

Choreography	Orchestration
▪ Beschreibung „globaler“ Prozesse ▪ Koordination relativ autonomer Partner ▪ Schnittstellendefinition	▪ Steuerung „lokaler“ Prozesse ▪ setzt lokale Workflow-Engine voraus ▪ tendenziell detaillierter als Choreography („message level“)

Choreography

- Kontrollfluss zwischen Web Services mehrerer Partner
- lokale Aktivitäten werden nicht detailliert betrachtet
- Darstellung von Prozessen mit folgenden Eigenschaften: mehrere Partner einbeziehend, meist lang andauernd, zustandsbehaftet, Konversations-orientiert
- Komposition mehrerer Web Services
- angehender Standard: WS-CDL
  - Standardisierungsorganisation W3C, XML-basiert, Einflüsse weiterer Sprachvorschläge
- möglicher Fokus: B2B

Orchestration

- Choreography kann als Rahmen für die Definition von Schnittstellen eines Partners dienen → Kollaboration verschiedener Organisationen
- Orchestration fokussiert auf die lokalen Abläufe innerhalb einer Organisationseinheit (z.B. Konzern, Unternehmen, Abteilung)
- Sprache: WS-BPEL
- Gemeinsamkeiten mit Choreography
  - Webservices als Schnittstellen (lose Kopplung)
  - asynchroner Nachrichtenaustausch

WS-BPEL

- BPEL: Business Process Execution Language
  - „high-level programming language“
  - Block-Modellierung („if“ oder „while“)
  - XML-basiert
  - keine einheitliche Notation
- Grundkonzepte aus zwei verschiedenen Sprachen: Web Service Flow Language & XLANG
- Partner Links zur Integration externer Web Services

```
<partnerLink
    myRole=„Provider"
    name="CustomerWS„
    partnerRole="Customer"
    partnerLinkType="ns:CustomerWS_PL"/>
```

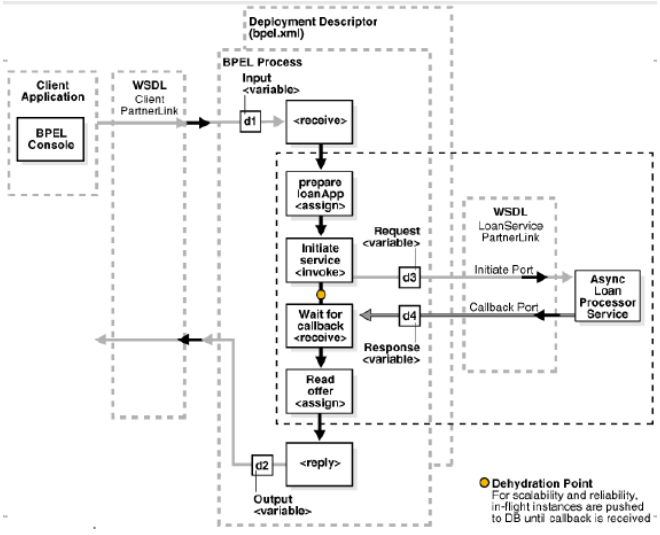
- Variablen bilden den Datenfluss innerhalb eines Workflows ab
  - Variablen beinhalten WSDL-Nachrichten
  - repräsentieren den Kontext einer Aktivität
  - Kontext kann persistent gespeichert oder zwischen PartnerLink ausgetauscht werden

```
<bpel:Variables>
   <bpel:variable name="output"
      messageType="tns:HelloWorldResponseMessage"/>
</bpel:variables>
```

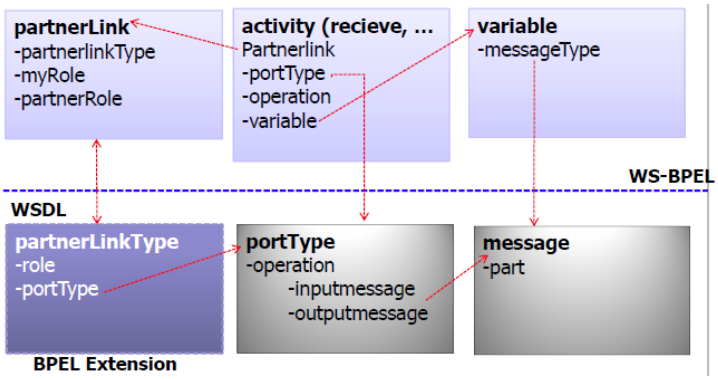
WS-BPEL Grundkonzepte

- Arten von Aktivitäten
  - *receive*: Empfangen einer Nachricht vom Partner
  - *invoke*: Senden einer Nachricht an einen Partner (mit *request*- und *response*-Variablen)
  - *reply*: Antwort auf ein *receive*
  - *assign*: Kopieren oder Manipulieren von Variablen
  - *pick*: erwartet eines von mehreren Ereignissen (z.B. *onMessage*, *onAlarm*)
- Kontrollstrukturen
  - *sequence*: sequentielle Ausführung von Aktivitäten
  - *switch*: eine von mehreren Aktivitäten wird ausgeführt (XOR)
  - *flow*: nebenläufige Ausführung
  - Schleifen (loops): *while*, *foreach*, …

WSDL als Briefumschlag für WS-BPEL



Abhängigkeiten zentraler Elemente und Attribute



Schnittstellen und Kontrollfluss

- Schnittstellen zwischen Partnern
  - *Port Type* (Bestandteil der Web Services): definiert Schnittstelle eines Web Service-Anbieters
  - *Partnerlink Type* (WS-BPEL): legt Beziehungen zwischen BPEL Aktivitäten und Port Types fest
- lokaler Kontrollfluss „block-structured“ oder „graph-oriented“

Korrelation von Nachrichten

- eingehende Antworten müssen zugehörigen Anfragen zugeordnet werden (Korrelation)
- Zuordnung zu Elementen in Nachrichten mithilfe von initialisierten Variablen

Transformation von BPMN nach BPEL

- BPMN fokussiert die Analyse und Kommunikation von Abläufen
- BPMN bietet keine formale Semantik zur Ausführung von Workflows
- BPEL fokussiert technische Implementierung, d.h. die formale Abbildung von
  - Prozessfluss über „activities“
  - Datenfluss über Variablen
  - Web Service Integration („partnerLink“)
  - sonstigen Implementierungsdetails, z.B. Java Script

Verheißung – BPEL Schemata als ausführbare BPMN Modelle

- aber: Transformation von BPMN nach BPEL stellt eine Herausforderung dar, da unterschiedliche Paradigmen verwendet werden
  - BPMN: Graph-orientierter Kontrollfluss
  - BPEL: „vorwiegend“ Block-basierter Kontrollfluss
- optionale Umsetzung in Anlehnung an Ouyan et al. (2009): „well-structured“ → direktes Mapping
  - Zerlegung eines BPMN-Diagramms in Komponenten
  - Mapping der Komponenten auf BPEL-Blöcke
  - z.B. Abbildung von BPEL Variablen auf BPMN Data Objects, aber dafür müssen alle relevanten Data Objects im BPMN-Diagramm modelliert werden

WS-BPEL – Zusammenfassung

- WS-BPEL dient der Orchestrierung von Web Services
- umfassende Kontrollstrukturen
- aber: Einschränkung des Anwendungsbereichs auf Web Services und somit keine dedizierte Berücksichtigung von sonstigen Ressourcen und Informationsobjekten
- keine sonstigen Schnittstellenarten (z.B. CORBA, RPC, RMI)

WS-BPEL – kritische Würdigung

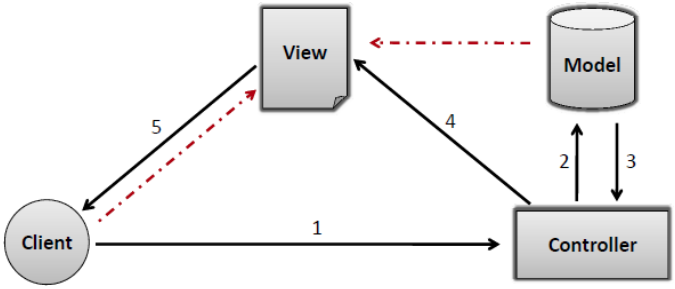
- Integrationsbasis für etablierte Technologien (Web Services)
- Abbilden von Kontrollflüssen
- Unterstützung unternehmensübergreifender GPs
- aber: keine graphische Notation definiert
- und: zwar eine höhere Programmiersprache, aber immer noch eine Programmiersprache

Vorgehen zur Erstellung einer Orchestrierung

1) Eclipse um BPEL und Orchestrierungs-Engine erweitern
2) BPEL-Projekt erzeugen
3) BPEL-Prozess erzeugen: BPEL und eine korrespondierende WSDL-Datei wird generiert
4) Kontrollfluss beschreiben und ggfs. externe Web Services integrieren
5) Orchestrierung auf Server zur Verfügung stellen, d.h. Kopieren der .bpel- und .wsdl-Dateien sowie eines Deployment Descriptors in den entsprechenden Ordner des Servers
6) Test durch Aufruf der entsprechenden URL: localhost:Portnummer/ode/processes/ServiceName/Operation?Inputvariable

Persistenztechnologien

Exkurs: Model-View-Controller (MVC)



3-Schichten-Architektur innerhalb von SW-Systemen

- Model
  - unabhängig von Präsentation
  - repräsentiert Daten der Anwendung
  - Implementierung der Geschäftslogik (Datenmanipulation, Validierung von Eingaben)
  - Anbindung an unterschiedliche Datenquellen (relationale DB, XML-Dateien, …)
  - oftmals in Kombination mit einer DB-Abstraktion
- View
  - Darstellung der Daten für den Benutzer = Präsentationsschicht
  - häufig Nutzung von Templates, d.h. Definition von Platzhalter, die durch Daten ersetzt werden
  - Entwickler von Views muss von der zugrunde liegenden Applikation nur sehr wenig wissen
- Controller
  - kontrolliert Benutzerinteraktion
    - Welche Aktionen darf der Benutzer durchführen?
    - Weiterleitung auf entsprechende Ergebnisseiten (Views)
  - regelt Zusammenarbeit zwischen Model und View
  - wertet aufgerufene Aktion und übergebene Parameter aus

Warum Persistenz-Mechanismen?

- standardisiertes Laden und Speichern von Objekten durch fertigen Code
  - Interoperabilität, weniger Entwicklungsaufwand, weniger Fehler
- Auslagern, um bei großen Datenmengen nicht alle Objekte im RAM halten zu müssen
- bei SW, die mit verbindungslosen Kommunikations-Architekturen operiert:
  - wiederholtes Laden / Speichern von Zuständen
  - manche Architekturen machen Persistenz besonders sinnvoll (z.B. Web-Anwendungen via HTTP)

Persistenz

- Zustände von Objekten (=Belegung der Variablen) sollen
  - speicherbar sein
    - nach Neustart von Programm / System / HW
    - zum Auslagern von großen Datenmengen aus dem RAM
  - über die Zeit erhalten bleiben
- Möglichkeiten
  - (De-)Serialisieren von Objekten
  - Rückgriff auf Datenbanken

Persistenz in Datenbanken: allgemeiner Ansatz

- Klassen werden DB-Tabellen zugeordnet
- Attribute von Klassen werden DB-Spalten zugeordnet
- Funktionen wie *loadObject()* und *saveObject()* werden vom Persistenz-Framework zur Verfügung gestellt
  - greifen auf die DB zu
  - verstecken SQL und andere DB-Details

DB-Schema generieren

- implizit aus Klassen-Beschreibungen und/oder Objekten zur Laufzeit (wie oben beschrieben)
  - problematisch: nicht unterstützte Datentypen, Vererbung, Beziehungen
- explizit aus zusätzlichen Angaben speziell für Persistenz
  - Konfigurationsdateien, Annotations im Code von Klassen
- ggfs. Mischform aus Modellen

OR-Mapping

- Strukturbruch, genannt „Impedance-Mismatch“
  - Objekte = beliebig vernetzte Strukturen, Hierarchien, Vererbungs- und enthalten-Beziehungen
  - Relationen = Tabellen, Beziehungen zwischen Tabellen über Fremdschlüssel
- Übersetzung zwischen Objektstrukturen und Relationen nicht eindeutig
  - mehrdeutiges Entwurfs- und Entscheidungsproblem

Vererbung – unterschiedliche Ansätze

- je 1 Tabelle pro Klasse
  - auch für (abstrakte) Oberklassen, geerbte Felder werden nicht übernommen
- 1 Tabelle pro konkreter Klasse
- 1 Tabelle aller Datenbankspalte (Nachteil: „sparse“ Belegung)
- 1:1-Verweis auf Zeilen in Tabelle einer Oberklasse

Persistenz-Manager vs. Persistenz-Methoden

- Persistenz-Manager Beispiel

```
PersistenceManager pm =
new PersistenceManager(); // singleton
pm.configure(...);
myObject = pm.load(...);
...
myObject.setValue(42);
...
pm.save(myObject);
```

- Persistenz-Methoden an Klassen / Objekten

```
myObject = new MyType(); // MyType extends PersistenceBase ...
myObject.load(...);
...
myObject.setValue(42);
...
myObject.save();
```

Laden eines Objekts

1) üblicherweise explizit zu Beginn eines Programms
   - z.B. `$myInstance = PersistenceManager::get($type,$id);`
   - Anfrage muss bestimmen, welches Objekt geladen werden soll
   - Objekt kann nicht aus dem Nichts kommen
   - Laden einzelner Objekte vs. Menge von Objekten
2) aber implizit bei Life-Cycle-Modell möglich (z.B. JavaBeans Activation Framework)
   - Objekt „wacht auf“

Speichern eines Objekts

1) explizit zum Ende eines Programms
   - z.B. `PersistenceManager::saveInstance($myObj);`
   - oder `$myInstance->save();`
   - Stichwort: “Database Facade”
2) explizit immer nach Änderungen am Objekt
   - ```
     $myInstance->variable = „a value“;
     $myInstance->save();
     ```
3) implizit nach Änderungen an einem Objekt
   - z.B. innerhalb einer Setter-Methode, transparent für den Anwender des Objekts
     ```
     $myInstance->setVariable(„a value“);
     ```
   - Implementierung z.B.
     ```
     class TheClass {
       function setVariable(String s) {
         if (s != $this->variable) {
            $this->variable = s;
            $this->save();
     } } }
     ```
   - Nachteil: hoher Ressourcenbedarf
4) implizit nach Ende des Programms bzw. bei Destruktion des Objekts
   - Destruktor-Methode
     ```
     function __destruct() {
       if ($this->isDirty()) {
         $this->save();
       }
     }
     ```
     ```

manuelle Persistenz

- Laden und Speichern als explizit programmierte Funktionen einer Klasse
  - z.B. `public static Collection loadAllInstancesFromDB()`
  - `public void saveInstanceToDB()`
- Datenbankanbindung muss selbst programmiert werden
- zwar sehr flexibel, aber sehr aufwändig und kaum Wiederverwendung

Persistenz-Frameworks – Hauptfunktionen

1) Datenbank-Schema generieren
2) Laden und Speichern von Objekten
   - meist in relationalen Datenbanken
   - z.B. via Funktionen `loadObject()` / `saveObject()`

implizite Schema-Generierung

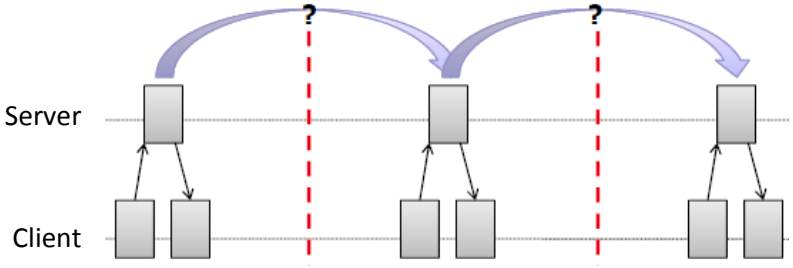
- kann im Idealfall „out-of-the-box“ auf bestehendem Code angewendet werden
- auf Informationen über Klasse / Instanz wird über Reflection-Mechanismen zugegriffen

Mischform: Persistenz-Framework aus Modell

- Hinweise zur Persistenz-Behandlung als Teil eines Modells
- Generierung von DB-Schema kann implizit aus Modell geschehen
- möglich, denn Modellierungssprache enthält semantikreichere Sprachelemente als Programmiersprache

Persistenz bei Web-Applikationen

- verbindungsloses HTTP-Protokoll präformiert die Architektur von Web-Anwendungen
- üblicherweise: Request-Action-Response
- Generierung dynamischer Web-Seiten
  - HTML-Output generieren: kurz ein Programm starten und wieder beenden
  - in der Zwischenzeit liegt Zustand der Web-Applikation als persistente Objekte in DB



Was geschieht mit den Objekten der Applikation in der Zwischenzeit?

Beispiel-Framework PHPersistence

- „phpersistence is an object oriented persistence layer for php, using transparent object relational mapping”

```
$persistence = new Persistence();
$persistence->save($user);
```

Persistenz in Web/MVC-Frameworks

- OR-Mapper integraler Bestandteil (und Erfolgsfaktor) aktueller Web-Frameworks
  - Ruby on Rails: ActiveRecord, Symfony: Doctrine oder Orpel, CakePHP: eigener Ansatz
- Convention over Configuration (Namenskonventionen): Abstraktion über DB folgt i.d.R. gewissen Namenskonventionen, d.h. Zusammenhang zwischen Objekten der Programmiersprache und Benennung der Tabellen und Attribute (insb. Schlüssel)

Beispiel-Framework: CakePHP

- MVC-Framework (= „Code-Gerüst“) für PHP
  - Anlehnung an Ruby on Rails (Ruby) & Struts (Java)
- Grundprinzipien
  - DRY – Don’t Repeat Yourself
  - CRUD – Create Read Update Delete
  - KISS – Keep It Simple Stupid
  - Rapid Development
    - try, change, try again
    - scaffolding, bake
- Unterstützung für Ajax, Sessions, Rechte-Verwaltung (ACL), Internationalisierung, TDD, etc.

von DB zur Anwendung: Scaffolding & Bake

- Analyse der Tabellen und Assoziationen als Ausgangspunkt
- Scaffolding
  - erstellt automatisch ein „Gerüst“ auf Basis einer DB-Tabelle
  - dienst als erste funktionsfähige Dummy-Anwendung (CRUD-Operationen)
- Bake
  - interaktives Shell-Skript
  - Model, View, Controller oder ganzes Projekt anlagen
  - DB-Konfiguration anpassen / erstellen
  - …

Vorgehen für einen ersten Prototypen

1) Relationen in DB anlegen (ggfs. + Testdaten)
2) Models definieren
3) Model-Relationen definieren
4) Controller anlegen
5) Scaffolding aktivieren

} über Bake Script

Besonderheiten & Gefahren

- Fremdschlüssel: Konsistenzsicherung im Framework und nicht auf der DB
- Limitation durch Konventionen?
- Nachvollziehbarkeit?
- Möglichkeiten zur Optimierung?

Java Persistence API

- Teil der Java EE Spezifikation
- Konfiguration bei Web-Applikationen und –Services über META-INF/persistence.xml
- Java-Annotations: Metadaten direkt im Sourcecode hinterlegen, können dann zur Compile- oder zur Laufzeit ausgelesen werden
- Annotationen können sich auf verschiedenste Java-Sprachelemente beziehen

Zusammenfassung

- Persistenz notwendig und wichtig
- Bruch zwischen OO-Welt und DB
- OR-Mapper weit verbreitet und ausgereift
- Funktionsumfang unterschiedlich und vom Einsatzgebiet abhängig