Integration von Webservices in Applikationen

Inhalt

1. Servicemodelle
    1. Utility-Service
    2. Business-Service
    3. Controller-Service
2. Modellierung Serviceorientierter Komponenten und Schnittstellen
    1. Probleme in vielen Projekten
    2. XWIF-Prozess für Komponenten Design
    3. XWIF-Prozess für Service-Interface Gestaltung

Inhalt

3. Strategien zur Integration von SOA
    1. Definition Konsistenter Kriterien für Kapselung- und Schnittstellengranularität
    2. Parameter vs. Methodengesteuerte Schnittstellen
    3. Design gemischter Granularitäten
    4. Separate Standards für Interne und Externe Services
    5. Webservices Dritter

Inhalt

4. Modellierung von Servicekompositionen
5. Erweiterungen der Servicefunktionalität
6. Integration von SOAP-Nachrichten

Servicemodelle

1.

- Utility-Service
- Business-Service
- Controller-Service

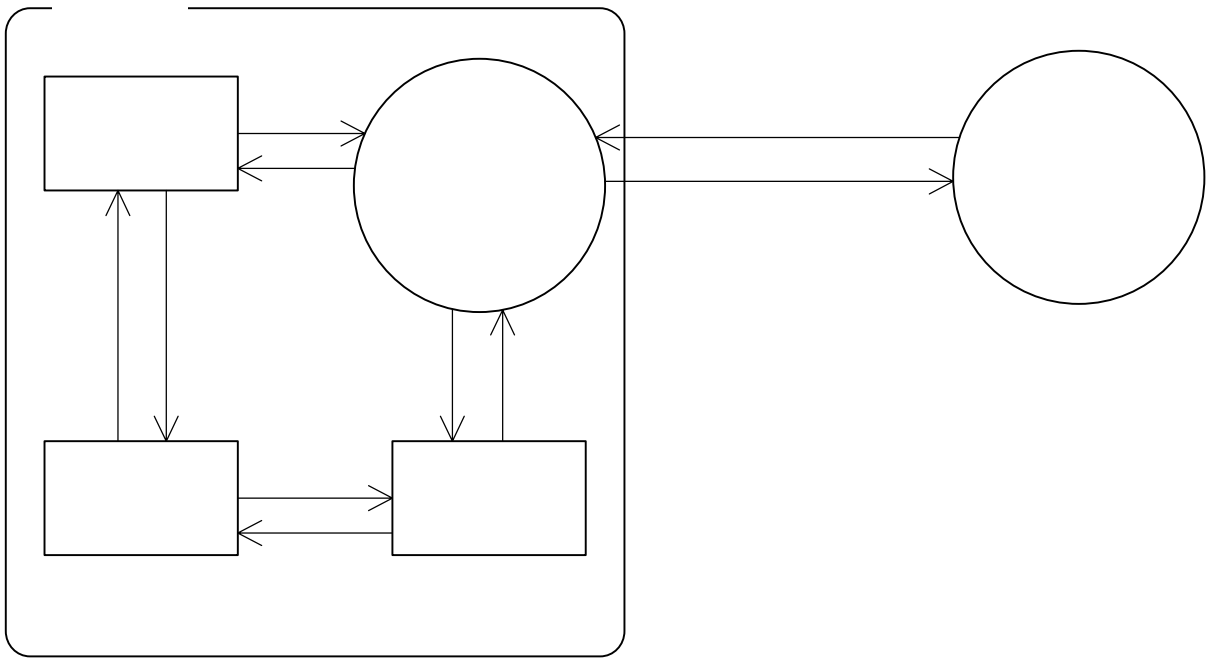
Servicemodelle

1.

- Utility-Service
  - Ein WebService mit Charakter einer widerverwendbaren Komponente
  - Ein allgemeiner Service als WS ohne Spezielle Geschäftslogik
    - Transformation von Daten
    - Verwalten von Daten
    - Berechnungen

Utility-Service

1.1.



Utility-Service

1.1.

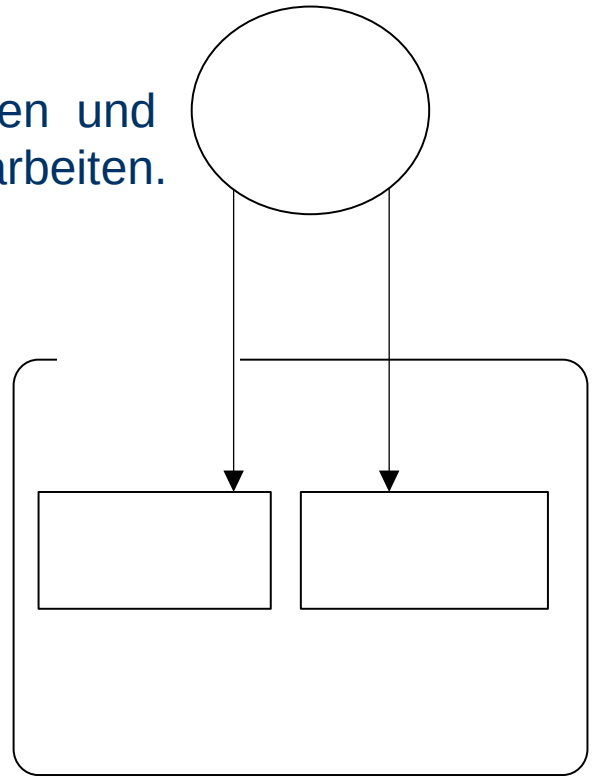
- Eigenschaften

Zugriffsvolumen	Sehr hohe Zugriffsraten. Gerade durch dritte können solche Services sehr stark in Anspruch genommen werden.
Schnittstelleneigenschaften	Normalerweise grobe Schnittstellen mit mehreren Parametern. Da der Service meist von außerhalb aufgerufen wird, sollte Kommunikationshäufigkeit beschränkt werden. Möglichst viel Information pro Transport.
Einsatzvorrausetzung	Auf Grund der hohen Nachfrage bei Utilityservices, sollten diese besser auf eigenen Servern angesiedelt sein. Somit können sie an einer Stelle angesiedelt werden, und trotzdem volle Arbeit leisten.

Servicemodelle

1.

- Business-Service
  - Stellt Services dar, die auf den Daten und Komponenten der Geschäftslogik arbeiten.



Business-Service

1.2.

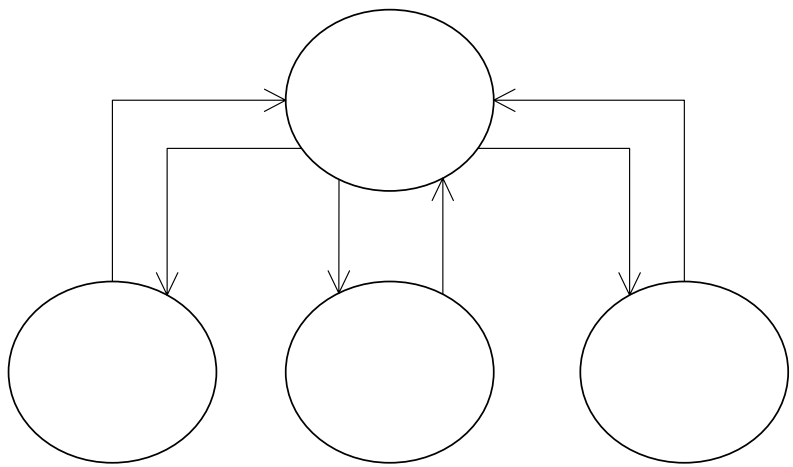
- Eigenschaften

Zugriffsvolumen	Mittelmass bis Hoch. Hängt davon ab wie oft der Geschäftsvorfall im Unternehmen benötigt wird
Schnittstelleneigenschaften	Detailliert bis grob. Die Schnittstellen müssen genau auf den Vorfall zugeschnitten sein. Somit hängt es vom Vorfall selber ab. Es gib auch Redundante Schnittstellen. Feine für Interne und Grobe für Externe Aufrufe.
Einsatzvorrausetzung	Aus Sicherheits- und Performancegründen sollten die Business-Services immer möglichst na an den Anwendungskomponenten gelagert sein.

Servicemodelle

1.

- Controller-Service
  - Führt mehrere Web-Services zu einem zusammen.
  - Kann durch Kombination von Business-Services einen weitern Business-Service darstellen



Controller-Service

1.3.

- Eigenschaften

Zugriffsvolumen	Mittelmass bis Hoch. Hängt davon ab wie oft der Geschäftsvorfall im Unternehmen benötigt wird. Aber als Repräsentant für mehrere Geschäftsvorfälle, ist das Volumen hier höher als beim Business-Service
Schnittstelleneigenschaften	Grobe Schnittstellen. Controller-Services vereinen existierende Serviceschnittstellen in kleinere Businessservice umfassende Operationen.
Einsatzvorrausetzung	Je nach dem, wie groß die Menge der Businessservices ist, die der Controllerservice vertritt und wo sich deren Daten befinden, kann ein eigener eine Vorraussetzung für Controller-Services sein.

Modellierung Serviceorientierter Komponenten und Schnittstellen

2.

Um die Schicht eines Webservices mit möglichst wenig Schwierigkeiten hinzufügen zu können, müssen die Komponenten der Software gut durchdacht werden.

Je mehr Gedanken man sich beim erstellen der

Basiskomponenten macht und sie dahingehend, dass ein Webservice nachgerüstet werden soll, modelliert, desto problemloser lässt sich zu einem späteren Zeitpunkt ein Webservice-Layer adaptieren.

Modellierung Serviceorientierter Komponenten und Schnittstellen

2.

Es ist wichtig, dass man sich Gedanken darüber macht, welche Services möglicher Weise in der Zukunft benötigt werden könnten.

- Welches Ausmaß soll der Webservice erreichen.
- Welche Geschäftsabläufe wird von den einzelnen Services benötigt.
- Wie viele Geschäftsabläufe sollte solche ein Service umfassen.
- Welche Funktionalitäten soll ein Webservice liefern?
- Wo liegen die technischen Grenzen meines Systems?
- Wie kann man Komponenten für die Serviceunterstützung optimieren?
- Wie sollte eine Service Schnittstelle designed sein?

Probleme in vielen Projekten

2.1.

In der Praxis ist die SOA unterschiedlich stark vertreten. Viele Projekte berücksichtigen die Nachrüstung von Webservices nur eingeschränkt, oder sogar gar nicht.

- Das kann unterschiedliche gründe haben, aber Thomas Erl führt die folgenden drei Gründe an.
  - SOA wir als neue Technologie gesehen, die nicht in die Software rein wächst, sondern vielmehr die jetzigen Technologien ersetzen wird.
    Somit ist es nicht relevant sich Gedanken über SOA zu machen, wenn man nach der alten Technologie vorgeht.
  - Die Modellierung von SOA und WS ist zu Komplex. Es wäre zu teuer sich in die Methodik einzuarbeiten.
  - Webservices sind sowieso nicht relevant.

Probleme in vielen Projekten

2.1.

Daraus resultieren aber zwei ganz entscheidende Probleme.

- Es ist sehr aufwändig Serviceschnittstellen an eine Software zu adaptieren, wenn sie nicht von vornherein darauf ausgelegt ist.
- Unter Service unfreundlichen Komponenten leidet die Performance sehr stark. Es entstehen zusätzliche Kosten um dieses
  Performanceproblem zu kompensieren.

Probleme in vielen Projekten

2.1.

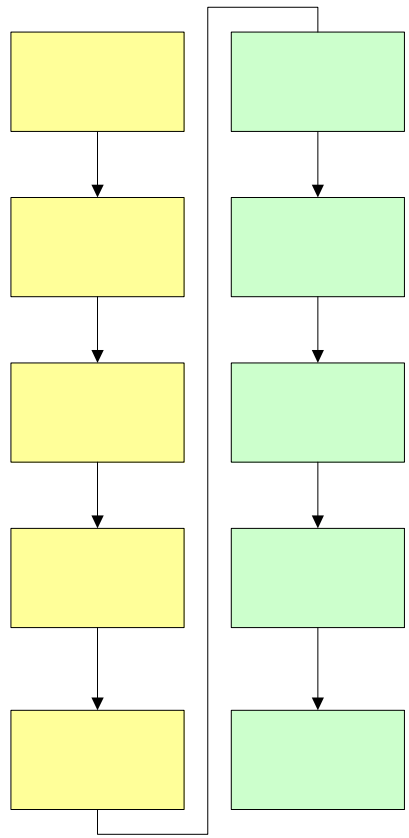
Um diesen Problemen vorzubeugen, hilft ein vom XWIF beschriebene Prozess, durch Spekulation, die Komponenten im vornherein SOA freundlich zu gestallten.

- für den Spekulativen Ansatz ist das Ausmaß des Webservices nicht wichtig. Es geht nur darum sich einen Überblick
- Sollen die Schnittstellen hier schon voll optimiert werden, ist auch die exakte Anforderung an den Webservice erforderlich.

XWIF-Prozess für Komponenten-Design

2.2.

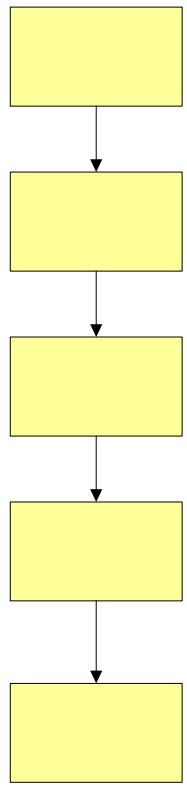
- Für das Desingnen der Komponenten beschreibt der XWIF-Prozess folgende Punkte.
  - 1-5 Erstellen und Analysieren der traditionellen Komponentenbasierten Klassen.
  - 6-10 Designen Serviceorientierter Komponenten & Klassen



Erstelle und analysiere traditionelle Komponentenbasierte Klassen

2.2.

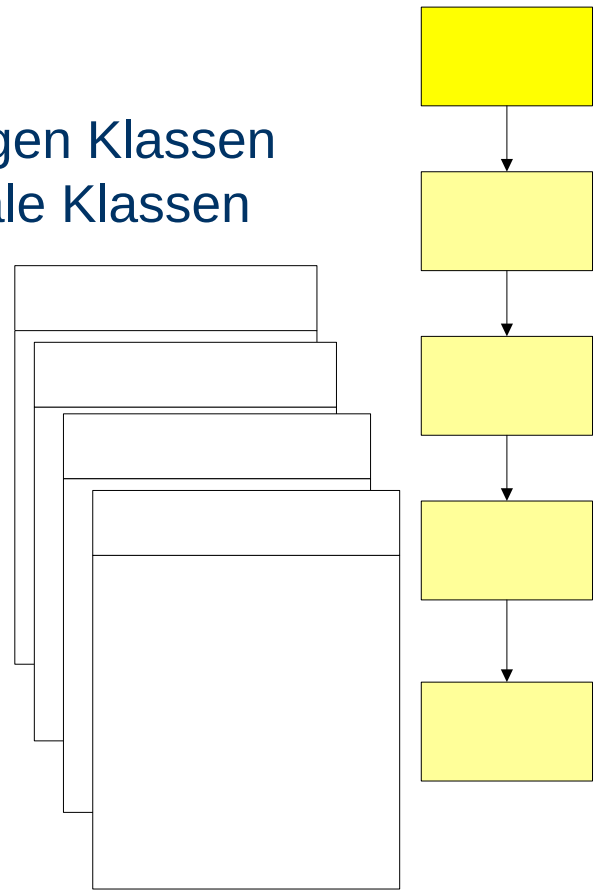
1. Business-Komponenten zusammen tragen
2. Identifizieren der Business-Logic für die Webservices nur beschränkt nutzbar sind
3. Identifizieren von Widerverwendbarkeiten
4. Identifizieren von Interface-Abhängigkeiten
5. Bestimmen von Ebenen der Zusammenarbeit



Businessklassen zusammentragen

2.2.1.

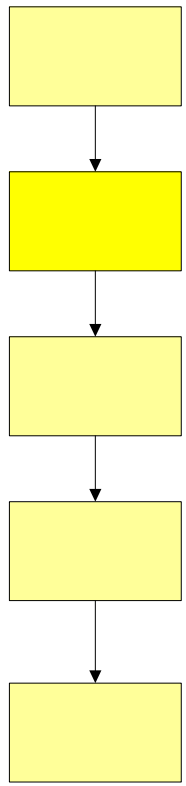
- Erstellen der Basisklassen
  - Es ist zu Anfang einfacher von fertigen Klassen auszugehen, anstelle gleich Optimale Klassen zu erstellen
- Bei den folgenden fünf Schritten werden die Methoden der Klassen in Gruppen mit bestimmten Merkmalen eingeteilt.



Welche Methoden sind nur begrenzt nutzbar

2.2.2.

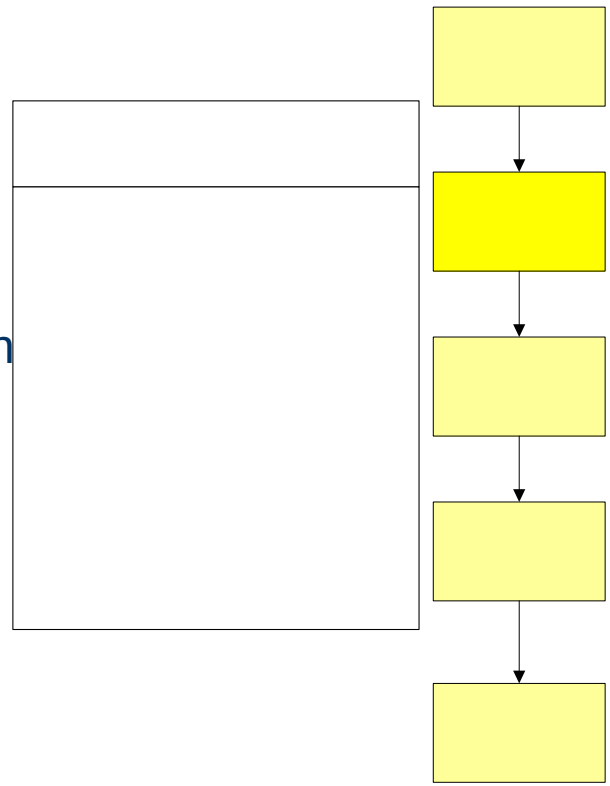
- Nicht alle Methoden bzw. Funktionalitäten sind zwingend für Webservices geeignet.
- Es ist nötig diese Funktionalitäten schon hier zu erkennen und zu markieren, da sie in der Modellierung gesondert behandelt werden.



Welche Methoden sind nur begrenzt nutzbar

2.2.2.

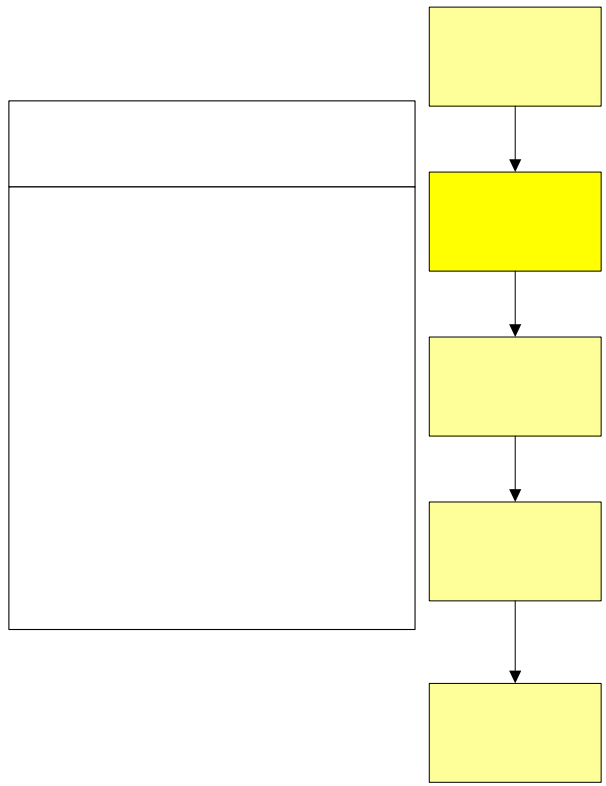
- updateInvoice oder submitInvoice sind zur Zeit nicht für WS geeignet
  - Wir gehen hier mal davon aus, dass die Implementierung von Transaktionen in diesem Fall nicht in Frage kommt.



Welche Methoden sind nur begrenzt nutzbar

2.2.2.

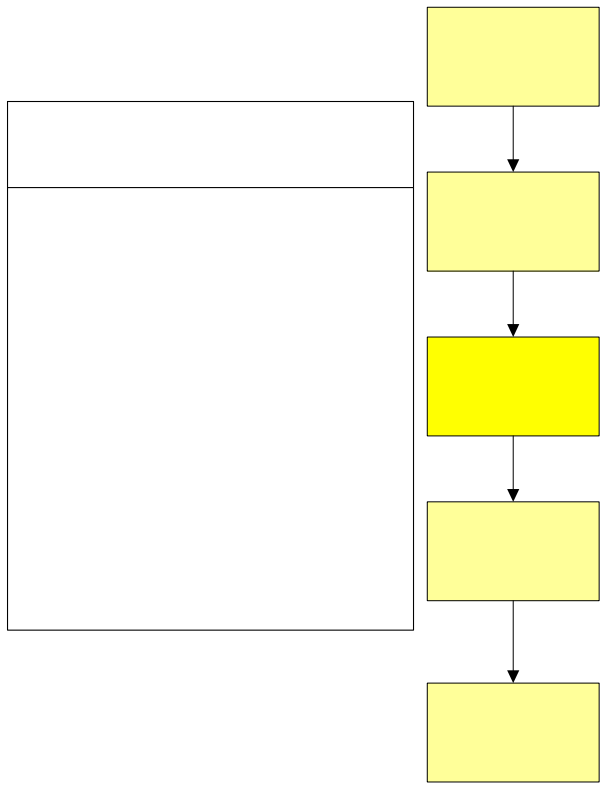
- getInvoiceHistory ist nur beschränkt nutzbar.
  - Es würde ein zusätzliches Authentifizierungskonzept benötigen



Identifiziere Widerverwendbarkeiten

2.2.3.

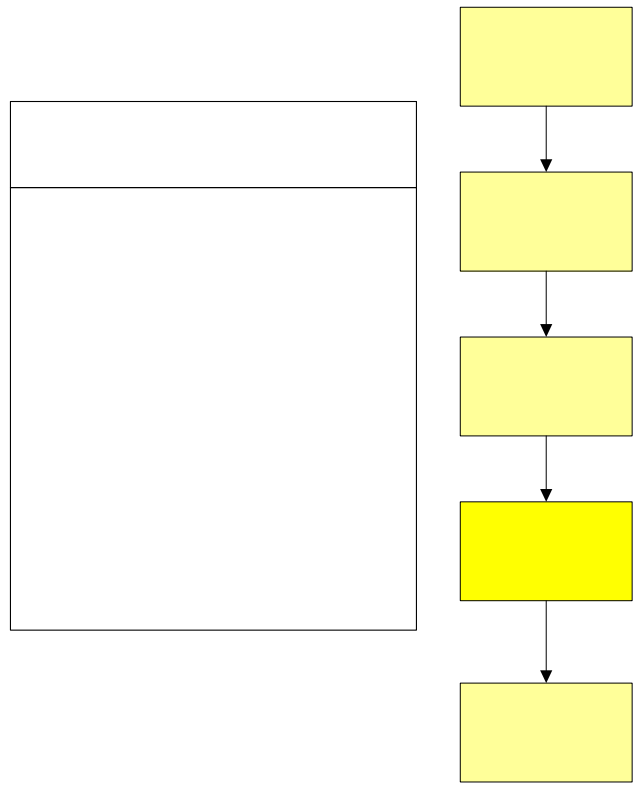
- Markieren der Funktionalitäten die für anderen Applikationen widerverwendbar sein können.
  - Die Erstellung von PDF-Dokumenten wird in der Regel nach dem gleichen Schema ablaufen.
  - Wozu also in jedem Prozess wieder aufführen?



Identifizieren der Abhängigkeiten in Methoden

2.2.4.

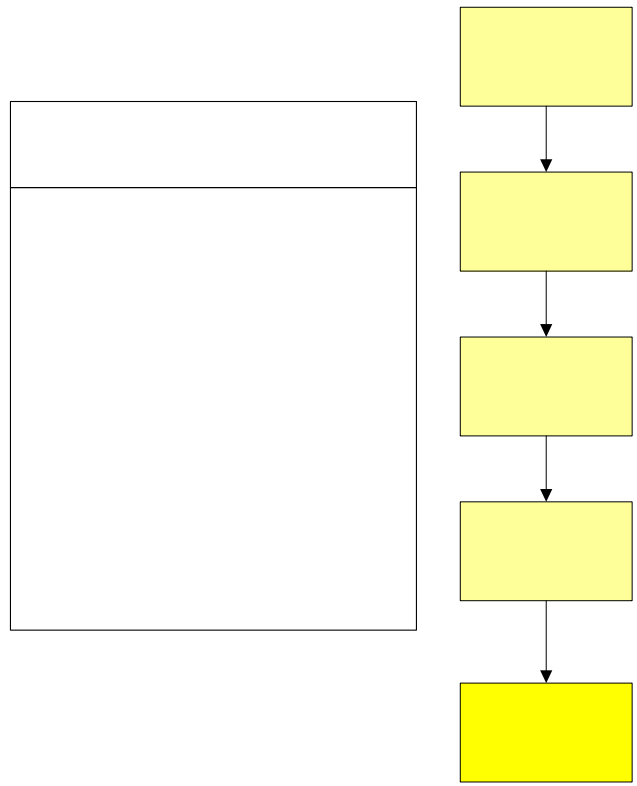
- Welche Methoden greifen auf die selben Hilfsmethoden zurück?
- Methoden die die selben Funktionen benutzen, sollten nicht getrennt werden.



Identifizieren der Abhängigkeiten in Methoden

2.2.5.

- Ein wichtiger Gesichtspunk bei der SOA ist auch das Level der Zusammenarbeit.
  - interne Anfrage
  - Externe Anfrage
- getInvoiceX12 ist mit großer Wahrscheinlichkeit eine Methode die von Außenstehenden Applikationen aufgerufen wird.

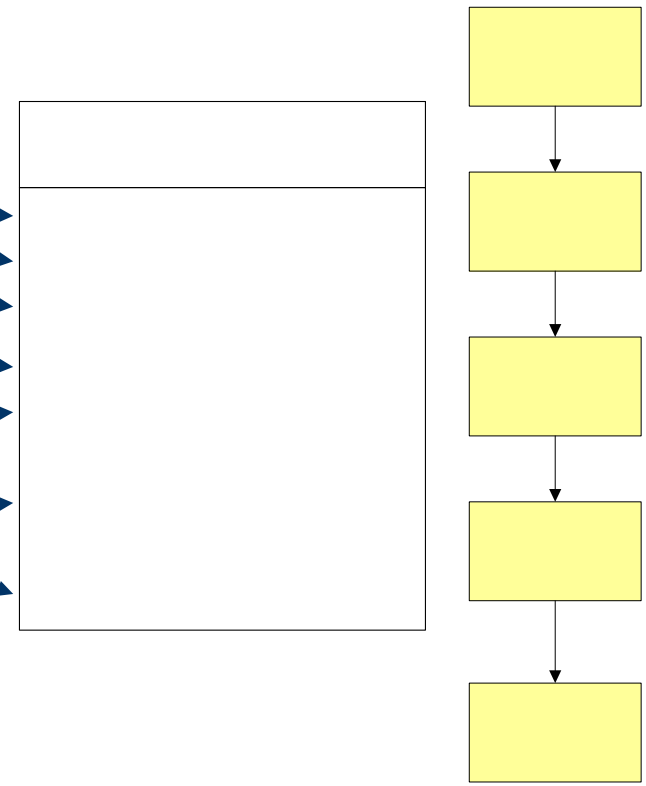


Identifizieren der Abhängigkeiten in Methoden

2.2.

Wir haben nun folgende Eigenschaften zusammengetragen

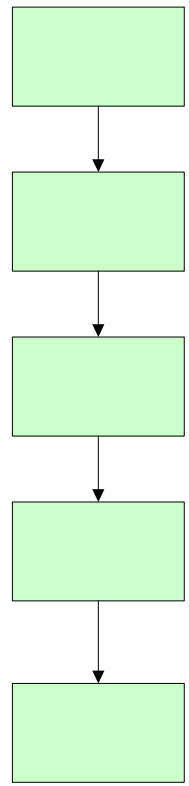
- Abhängigkeit
- Widerverwendbarkeit
- Externe Aufrufe
- Spezielle Implementierung nötig (Authentifizierung)
- Nicht ohne weiteres umsetzbar.



Design Serviceorientierter Komponenten & Klassen

2.2.

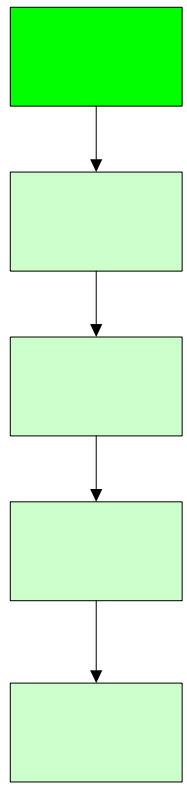
6. Erstellen von feineren Aufgabenbezogenen Komponenten
7. Gruppieren der Methoden auf Grund Ihres Typs
8. Bestimme Kandidaten die für SOA relevant sind
9. Kontrolliere nicht Serviceorientierter Komponenten
10. Suche Möglichkeiten der Zusammenführung



Erstellen von feineren Aufgabenbezogenen Komponenten

2.2.6.

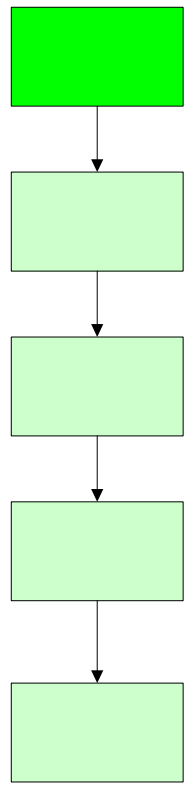
- Wir haben nun die Klasse ProcessInvoice, die eine ziemlich Grobe Schnittstelle darstellt
- In der Klasse befinden sich jedoch noch Methoden, die wir für den jetzigen Zeitpunkt als nicht relevant für Webservices eingestuft haben.
  - UpdateInvoice()
  - UpdateInvoiceStatus()
  - SubmitInvoice()
- Ausgliedern dieser Methoden in eine neue Klasse

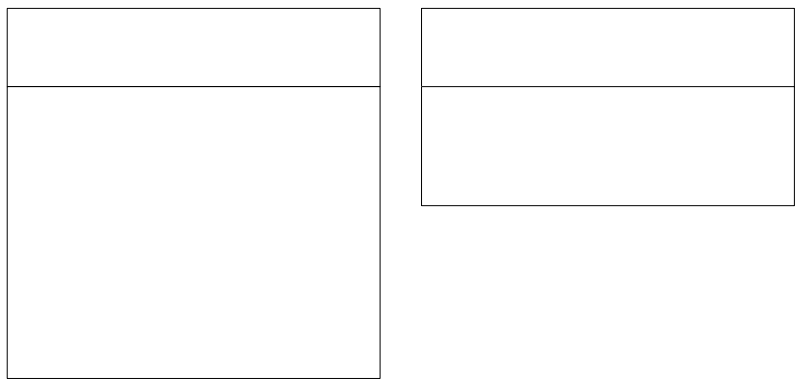


Erstellen von feineren Aufgabenbezogenen Komponenten

2.2.6.

- Nach dem Ausgliedern dieser Methoden haben wir Zwei klassen.
- Jedoch ist die Klasse mit den Servicerelevanten Methoden immer noch recht grob als Schnittstelle.

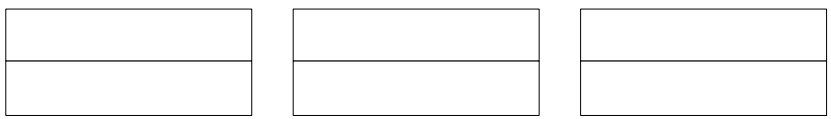


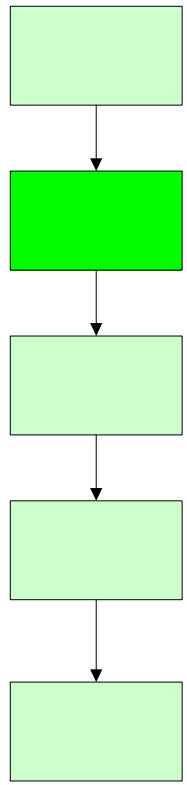


Gruppieren der Methoden auf Grund Ihres Typs

2.2.7.

- Drei Methoden hatten noch extra Eigenschaften
  - getInvoiceHistory()
  - getInvoicePDF()
  - getInvoiceX12()

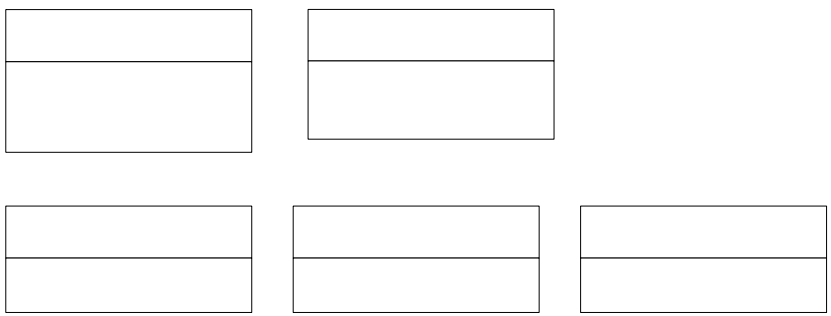


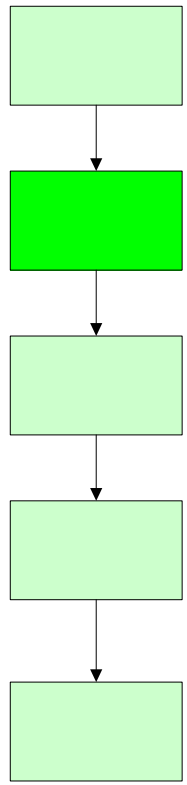


Gruppieren der Methoden auf Grund Ihres Typs

2.2.7.

- Daraus entstehen die folgenden fünf Klassen.

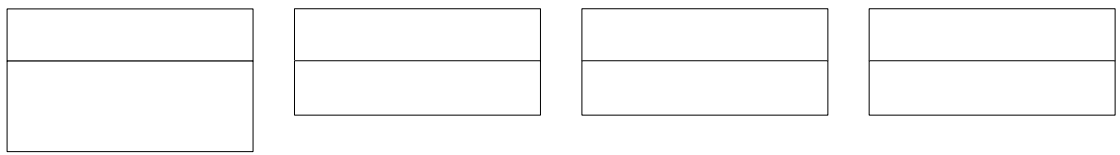




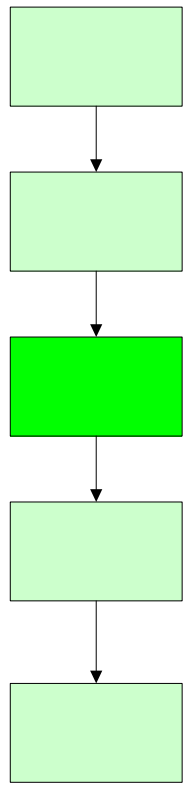
Bestimme Kandidaten die für SOA relevant sind

2.2.8.

- Relevant für Webservices sind nach dieser Analyse:



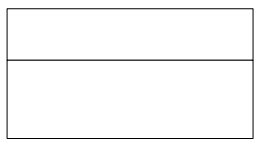
- Aus diesen Klassen können nun die einzelnen Services bedient werden, ohne dass zu grobe Schnittstellen die Performance belasten.

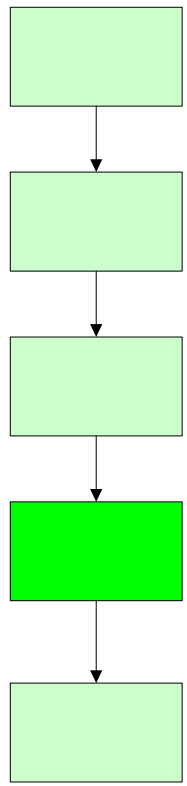


Kontrolliere nicht servicerelevante Komponenten

2.2.9.

- Im Zehnten Schritt können Klassen, die nicht Servicerelevant sind wieder zu einer größeren Klasse zusammengesetzt werden.
- Wir haben sie gar nicht erst getrennt gehabt.

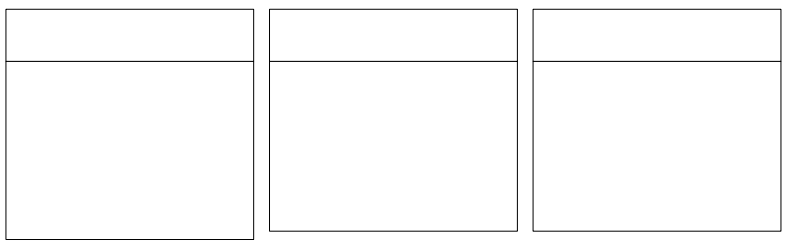


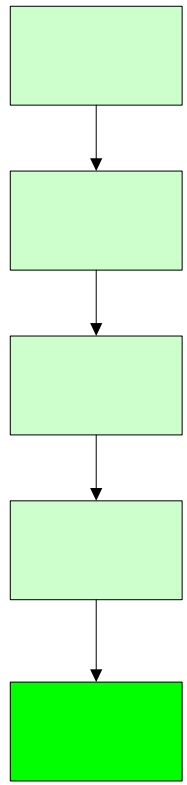


Identify cross-task consolidation opportunities

2.2.10.

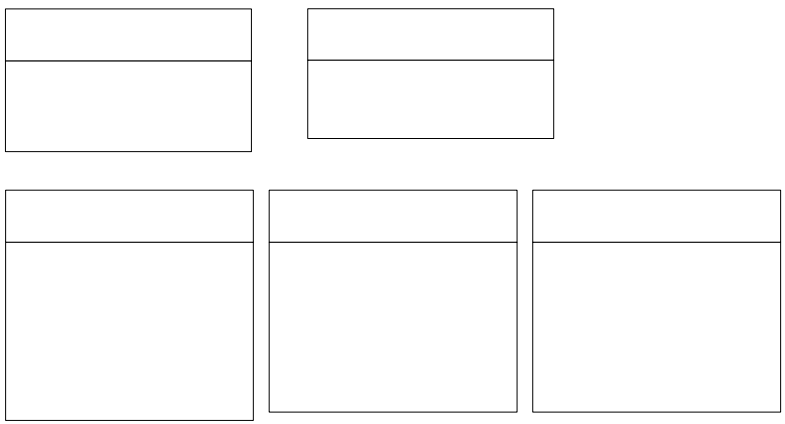
- Der letzte Schritt wird eigentlich erst relevant, wenn man vorher schon Klassen abgearbeitet hatte.
  - Methoden aus anderen Objekten, die gleiche Bedingungen an die Serviceumgebung haben wie z.B. getHistory() die Authentifizierung oder getPDF() die PDF-Anweisungen, können zusammengelegt werden.

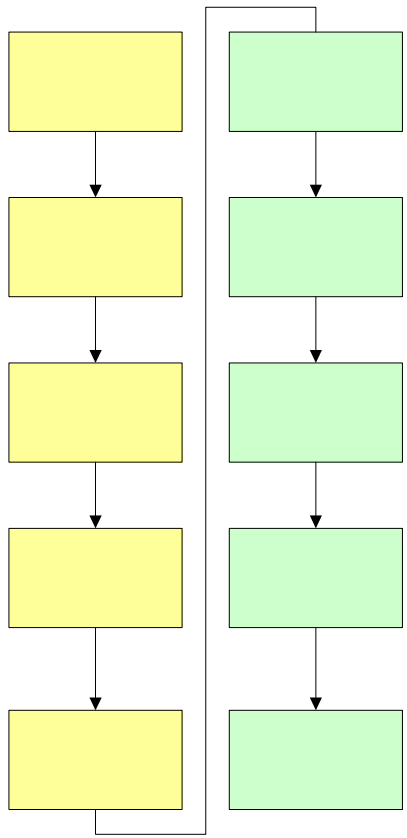




XWIF-Prozess für Komponenten-Design

- Mit den letzten 10 Schritten pro Klasse haben wir ein Klassenmodell erstellt, welches eine gute Serviceorientierung aufweist.

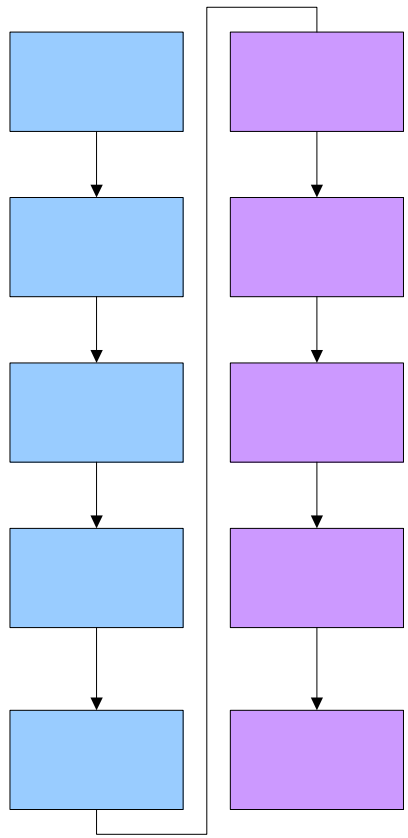




Designen der Webservice Interfaces

2.3.

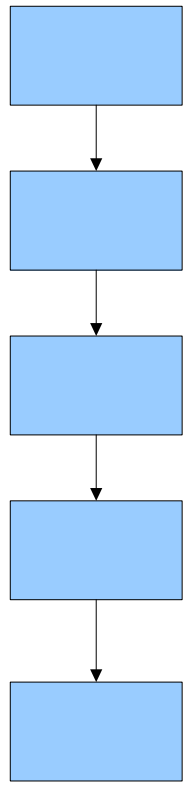
- Für das Desingnen der Webservice Interfaces hat der XWIF wieder einen Prozess beschrieben.
  - 1-5 Zusammensammeln der für einen bestimmten Service nötigen Klassen
  - 6-10 Das erstellen der Schnittstellen zu dem Service



Zusammensammeln der für einen bestimmten Service nötigen Klassen

2.3.

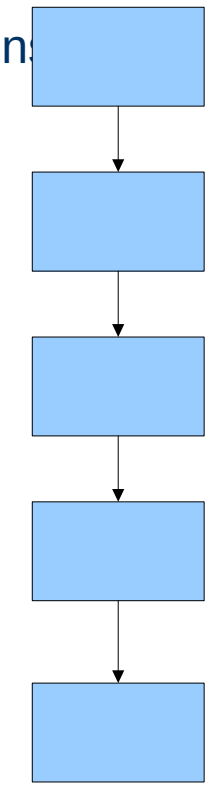
1. Servicemodel wählen
2. Bestimmen des Anwendungsbereiches der Geschäftslogik
3. Bestimmen potentieller Klienten
4. Bestimmen der benötigten Daten
5. Prüfen der gegenseitigen Aufrufe



Zusammensammeln der für einen bestimmten Service nötigen Klassen

2.3.

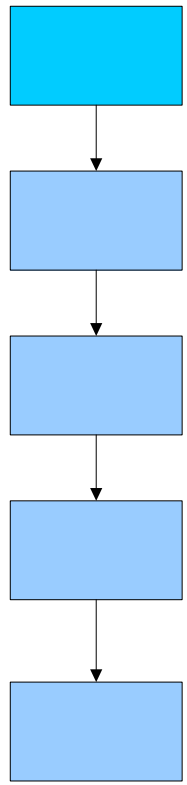
- Bevor wir die Klassen zusammenstellen können müssen wir uns
  Webservices entscheiden die wir anbieten wollen.
  - GetInvoice
  - GetReports



Servicemodel Wählen

2.3.1.

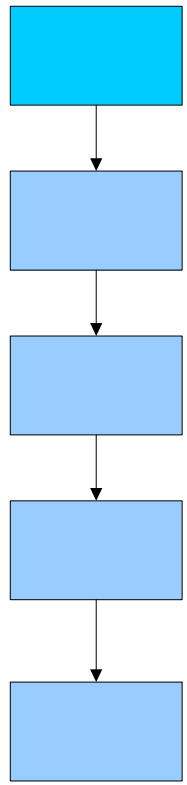
- Die Wahl des Servicemodells ist wichtig, da jedes Modell eine anderes Nutzungs- und Anwendungsverhalten hat.
- ABER: Services können auch mehreren Modellen angehören.
- „Utility-Service ist die beste Klassifikation für Services, sobald bestätigt wurde, dass ihre Hauptaufgaben allgemein genutzt und wieder verwändet werden.“ (Thomas Erl)



Servicemodel Wählen

2.3.1.

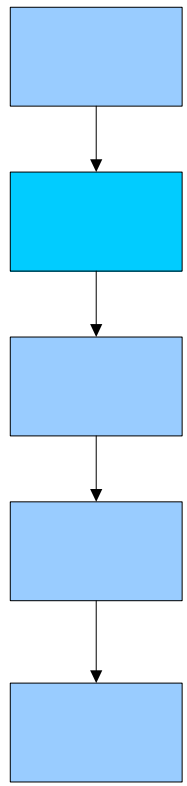
- Welches Modell beschreibt den Service der zu erstellen ist am besten?
  - Utility service GetReports
  - Business service GetInvoice
  - Controller service
  - Proxy service
  - Wrapper service
  - Coordination service für atomare Transaktionen
  - Process service
  - Coordination service für Geschäftsaktivitäten



Bestimmen der genauen Funktion eines Services

2.3.2.

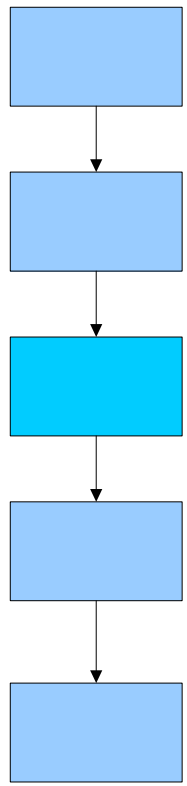
- Es ist wichtig die genauen Aufgaben und Funktionen eines Services zu bestimmen.
- Services die auf Grund schlechter Abgrenzungen in den Aufgabenbereich anderer geraten, können dazu führen, dass
  - Sie die Funktion von einem Service in einen anderen umpflanzen müssen
  - die Funktion in beiden Services vorkommt, was eine unnötige Coderedundanz darstellt.



Bestimmen Potentieller Clienten

2.3.3.

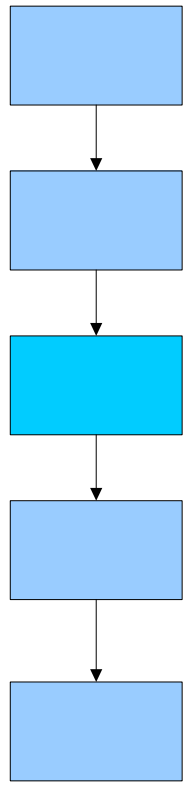
- In den meisten Fällen werden Services für eine bestimmte Aufgabe in einer bestimmten Applikation erstellt.
- ABER:
  - Gerade bei WebServices ist es aber wichtig, auch darauf zu achten, dass der Service auch für andere Interessenten in Frage kommen kann.



Bestimmen Potentieller Clienten

2.3.3.

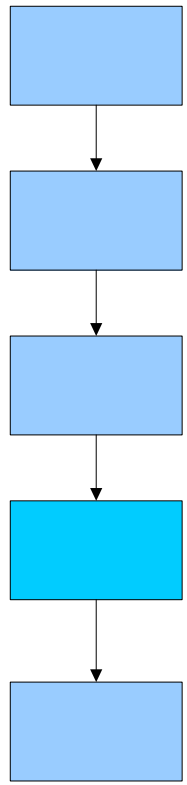
- Über folgende Clienten sollte man sich Gedanken über deren zukünftige Relevanz machen.
  - Teile der Eigenen Applikation die in der Zukunft in Services gepackt werden können.
  - Andere Applikationen die in der Zukunft Servicefähig werde können.
  - Services die den eigenen Service Kapseln wollen.
  - Externe Organisationen die auf Öffentliche Teile der Applikation, repräsentiert durch einen Service, zugreifen dürfen.



Bestimmen der benötigten Daten

2.3.4.

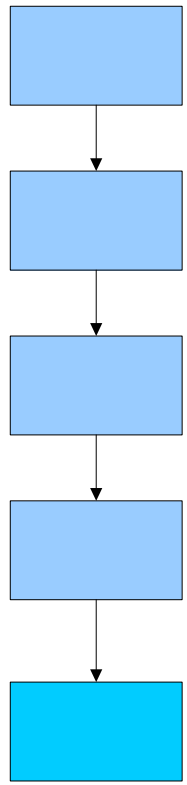
- Auflisten der Daten die für die Verarbeitung der Anfrage benötigt werden.
  - Daten zur Validierung von Übermittelten Daten
  - Daten die für die Antwort auf die Anfrage benötigt werden.
- Die Art der Daten sollte hier getrennt werden.
  - Als Parameter übergebene Daten
  - Rein intern verarbeitete Daten



Prüfen der gegenseitigen Aufrufe

2.3.5.

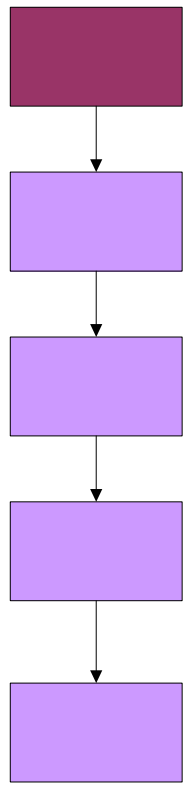
- Zu diesem Zeitpunkt sollte die Übersicht über die Funktionen und Daten die von einem Service genutzt werden komplett sein.
- Das Prüfen der gegenseitigen Aufrufe soll eventuelle Alternativen aufweisen, die eine effektivere und schnellere Datenverarbeitung ermöglichen.
- Bei Neuentwicklungen mag dieser Schritt nicht zwingende zu sein, aber beim Adaptieren der Services an ein bestehendes System, können an einigen Stellen bessere Wege zum Gewinn der Daten gefunden werden.



Bestimmen der benötigten Komponenten

2.3.6.

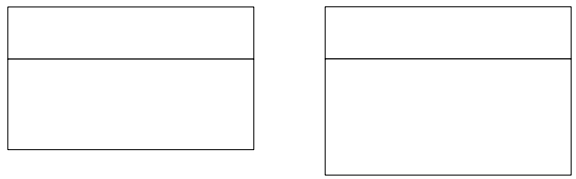
6. Bestimmen des Zusammenhanges der benötigten Klassen.
7. Modellieren des Serviceinterfaces
8. Visualisieren der Interaktions-Szenarien
9. Design der Nachrichtenstruktur
10. Servicemodel verfeinern



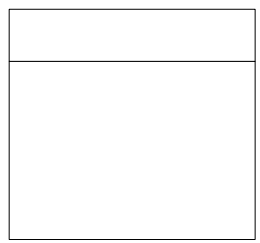
Bestimmen der benötigten Komponenten

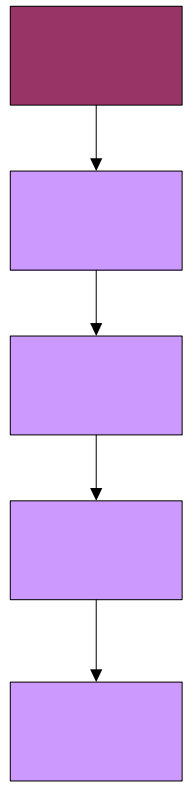
2.3.6.

- Erstellen einer Übersicht, welche Komponenten von den einzelnen Services benötigt werden.
  - Für GetInvoice



  - Für GetReports diese:

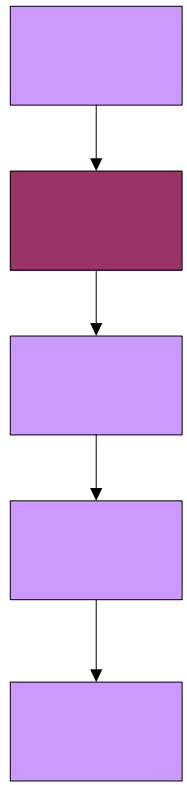




Modellieren des Serviceinterfaces

2.3.7.

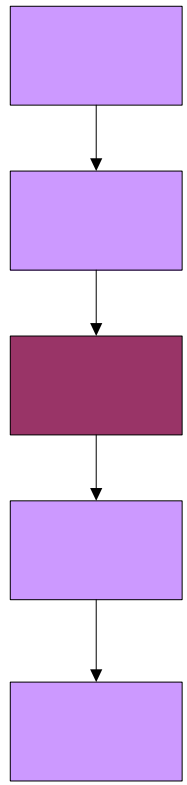
- Hier geht es hauptsächlich um die öffentlichen Schnittstellen.
- Die in den vorhergegangenen Schritten gesammelten Informationen stellen eine gute Hilfe für das erstellen der Interfaces dar.



Visualisiere Interaktionsszenarien

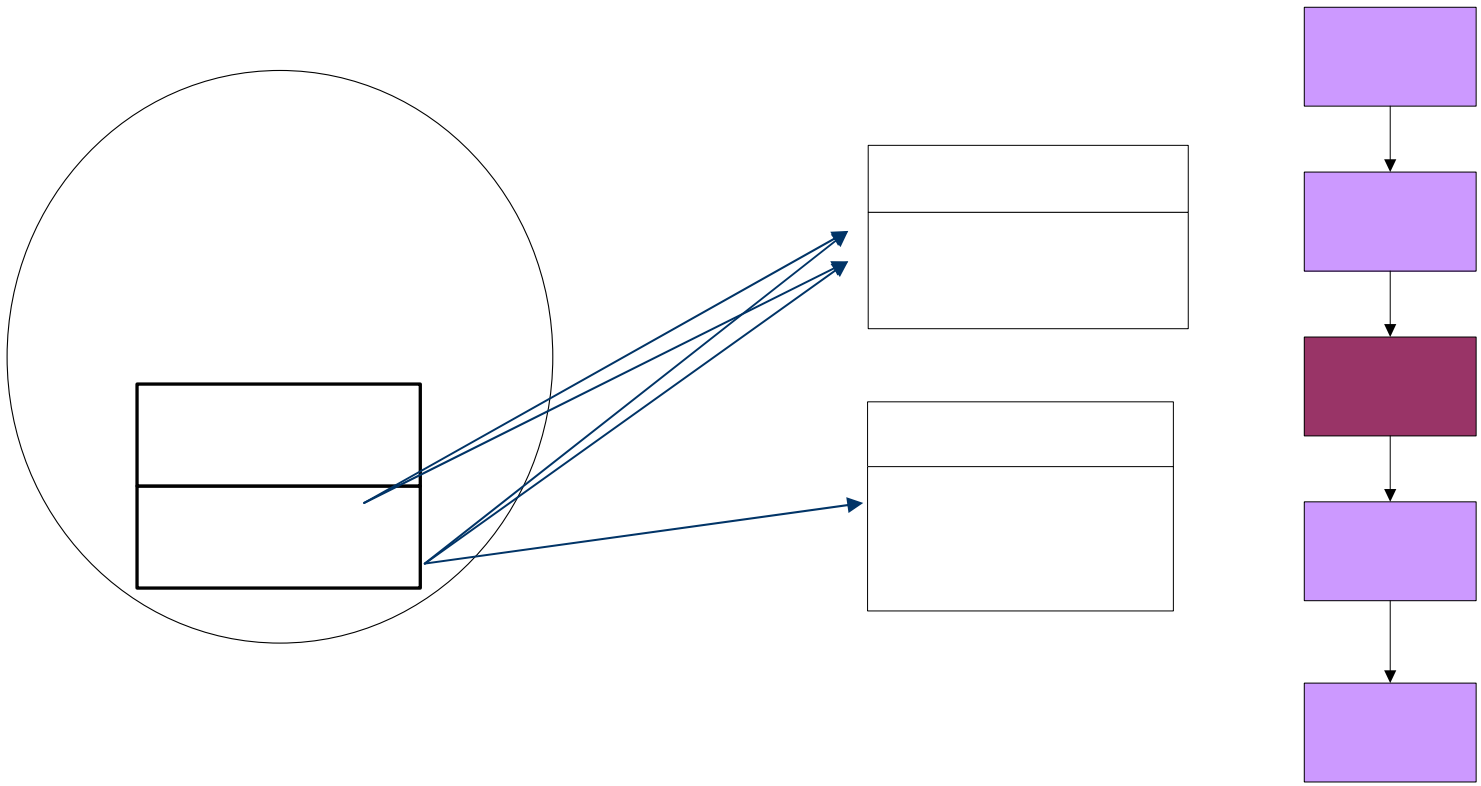
2.3.8.

- Je nachdem wie groß die Gruppe der benötigten Komponenten ist, desto mehr unterschiedliche Komponenten werden in den einzelnen Methoden benötigt.
- Das aufzeichnen dieser einzelnen Aufrufe zu den einzelnen Methoden hilft nicht nur dem Verständnis des eigenen Systems, es kann auch Grundlagen für ein vernünftiges Testsystem werden.



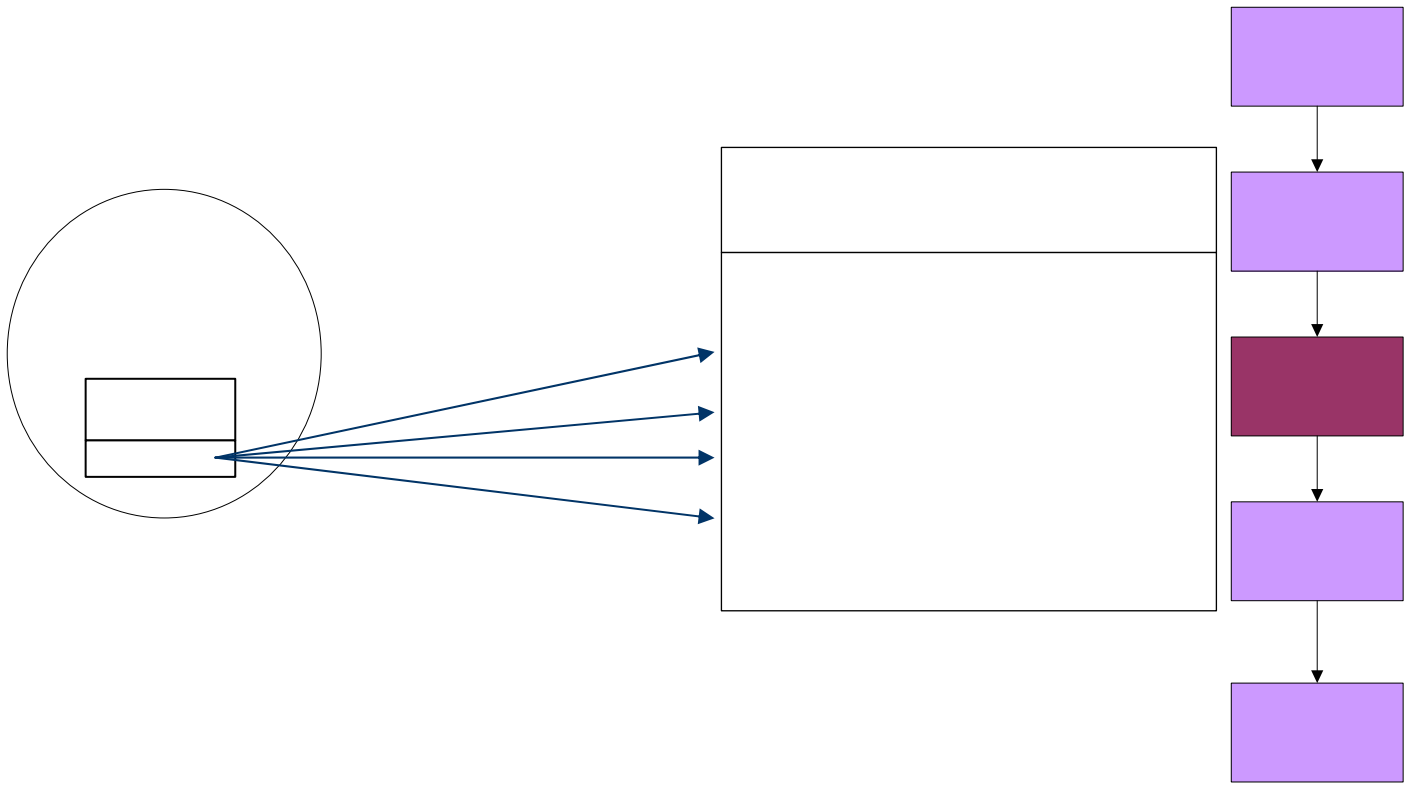
Visualisiere Interaktionsszenarien

2.3.8.



Visualisiere Interaktionsszenarien

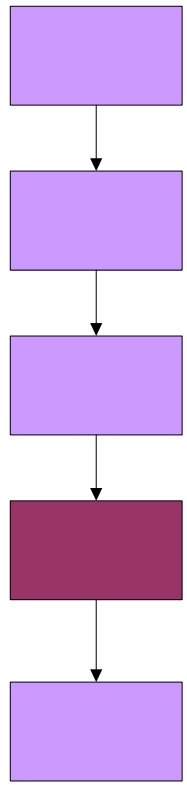
2.3.8.



Design der Nachrichtenstruktur

2.3.9.

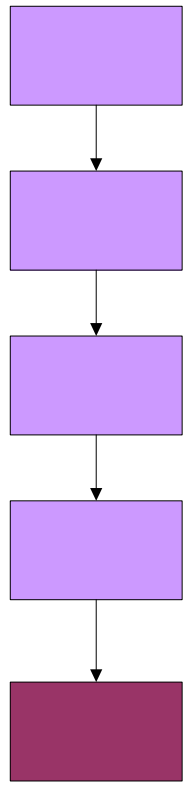
- Modellieren der XML-Nachrichten in beiden Richtungen
  - Inbound
  - Outbound



Servicemodell verfeinern

2.3.10.

- Prüfen, ob die Servicemodelle der Services im ersten Schritt wirklich korrekt gewählt wurden.
- Weiteres verfeinern über die folgenden Punkte



Strategien für Integration
SOA-Encapsulation

3.

- Definition konsistenter Kriterien für Kapselungs- und Schnittstellengranularität
- Parameter- vs. Methoden gesteuerte Interfaces
- Design mit diversen Granularitäten
- Konsistentes Anwenden gewöhnlicher Services
- Webservices Dritter

3.1. Definition Konsistenter Kriterien für Kapselungs- und Schnittstellengranularität

- Verwendung der immer gleichen Modellierungsmethode
- Bei nicht einheitlicher Modellierung entsteht schnell Chaos
- Die Granularität hangt von der Aufgabe des Services ab.
- XWIF

Parameter- vs. Methodengesteuerte Schnittstellen

- Methoden in der Form Verb-Nomen
  - Einfach und schnell zu verstehen, um welche Methode bzw. Aufgabe es sich handelt.
- Parametergesteuert
  - Methode hat immer den selben Namen
  - Der Parameter bestimmt, welche Funktion im innern des Services ausgeführt wird.
  - Hohe Erweiterbarkeit.

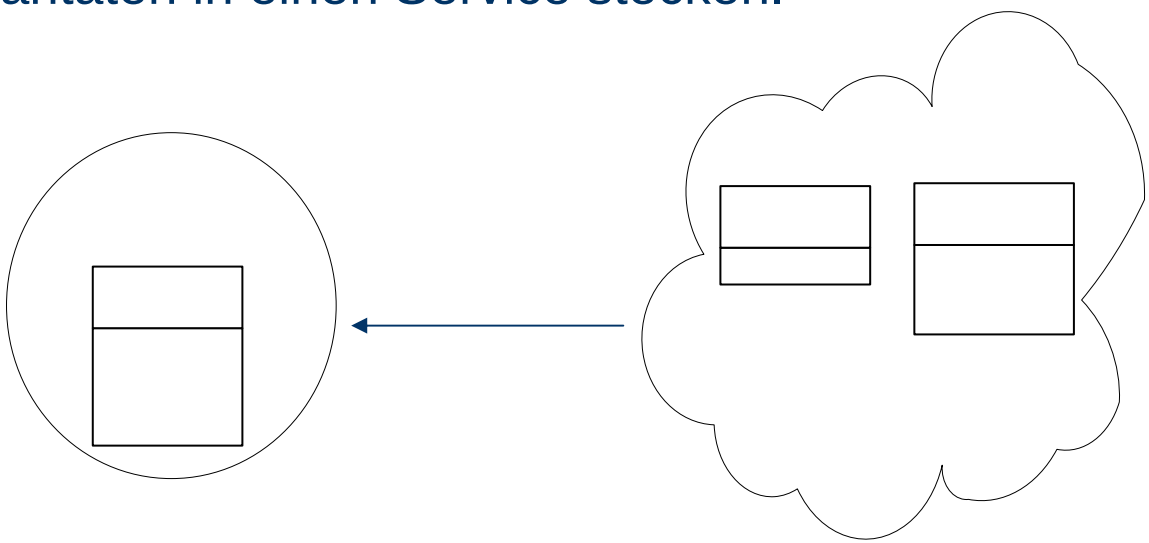
Design gemischter Granularitäten

- Grobe Granularität
  - Minimierung der Methodenaufrufe
  - Aber auch immer viel Information die übermittelt wird
- Feine Granularität
  - Durch viel kleine Aufgaben flexibler
  - Zwar viel Aufrufe für größere Aufgaben, aber das Datenvolumen pro Aufruf ist kleiner

Design gemischter Granularitäten

3.3.

- Um eine Alternative zu bilden könnte man auch beide Granularitäten in einen Service stecken.



- Das erzeugt womöglich Redundanz, aber nur in der Verfügbarkeit von Methoden durch Schnittstellen.
  Die Implementierung bleibt die gleiche

Separate Standards für Interne und Externe Services

3.4.

- Die beste Variante ist natürlich für Interne und Externe Services die selben Standarte zu haben.
  - Die Portierung von Intern zu extern oder anders herum wäre dadurch wesentlich einfacher.

Separate Standards für Interne und Externe Services

3.4.

- Jedoch gibt es einige Praktische Unterschiede zwischen Internen und Externen Services die zu beachten sind.
  - Usagevolume
    - Interne Aufrufe können wesentlich besser vorhergesagt werden.
    - Externe Aufrufe sind weniger berechenbar und müssen abgeschätzt werden.
      Bei deren Implementierung muss wesentlich mehr auf Skalierbarkeit geachtet werden.

Separate Standards für Interne und Externe Services

3.4.

- UDDI
  - Externe Services sollten möglichst UDDI-Server publiziert werden um dynamisch eingebunden werden zu können
  - Interne Services können auch manuell eingebunden werden und benötigen somit die UDDI-Registrierung nicht.
- Externe Services müssen Informationen über Ihre Schnittstellen zur Verfügung stellen
- Externe Services benötigen ein ausgeprägteres Exception-Handling

Separate Standards für Interne und Externe Services

3.4.

- Um für Externen gebrauch erstellte Services Intern zu nutzen gibt es zwei Möglichkeiten:
  - Nutzen der Externen Schnittstelle Intern und Inkonsistenz in Kauf nehmen.
  - Hinzufügen einer Internen Schnittstelle zum Externen Service.

Webservices Dritter

3.5.

- Bevor man selber Services Implementiert sollte auch in Betracht gezogen werden, ob man die Anforderungen auch durch Externe Services Drittanbieter erfüllen kann.

Webservices Dritter

3.5.

- Jedoch sollten folgende Dinge beachtet werden:
  - Jeder kann Webservices erstellen und zur Verfügung stellen. Ist der Angebotene Services Qualitativ wirklich ausreichend für die eigene Anwendung
  - Ist die Verfügbarkeit des Services gegeben?
    - Viele können diesen Service nutzen. Ist er für diesen Andrang ausgelegt?
    - Es sollte versucht werden, mit dem Anbieter ein Service Level Aggreement zu vereinbaren.
  - Nicht länger Suchen, als man für die eigene Implementierung brauche würde ;-)

Modellieren von Servicekompositionen

4.

- Kompositionen sind am effizientesten wenn:
  - Sie aus kleinen,
  - agilen und
  - unabhängigen Komponenten

  Bestehen.
- Sogar zusammenhängende Geschäftslogiken werden so in einzelne Teilservices zerstückelt.

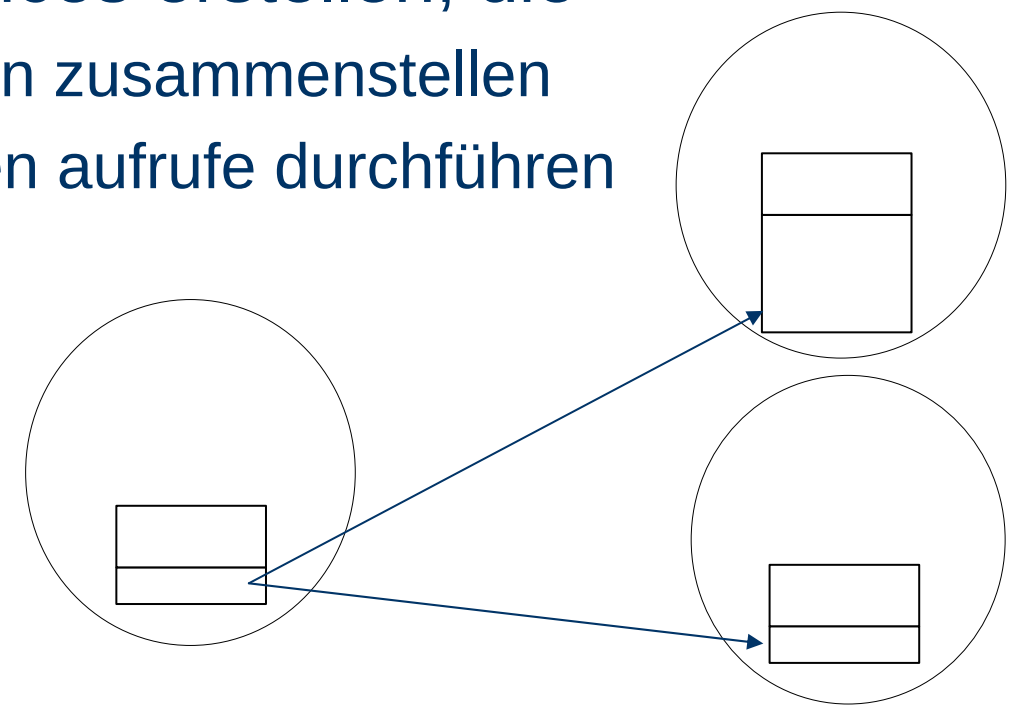
Modellieren von Servicekompositionen

4.

- Dieses Aufteilen ist etwas typisches für eine SOA, hat aber zwei Nachteile
  - Das dynamische zusammensetzen der Services zu dem gewünschtem Businessprozess kostet viel Zeit.
  - Der Service-Client muss jeden Teilservice einzeln aufrufen.

Modellieren von Servicekompositionen

- Um dieses Problem zu umgehen, kann man Controller-Services erstellen, die
  - die Komposition zusammenstellen
  - und die internen aufrufe durchführen



Modellieren von Servicekompositionen

4.

- Jetzt können wir über einen Service auf alle Daten der Rechnung zugreifen.
  - Um aber auf die einzelnen Daten zugreifen zu können, müssen wir
    - Entweder die SubServices auch zugänglich machen,
    - oder die Abfrage am Controllerservice Parametergesteuert durchführen.

Strategien zum Integrieren von Service Kompositionen

4.

- Nichts sollte übertrieben werden.
- Probleme mit:
  - Abhängigkeiten
    - Servicekompositionen die aus zu vielen einzelnen Services bestehen, haben oft so viele Abhängikeiten mit denen
      - Grenzen entstehen,
      - und Flexibilität schwindet.

Strategien zum Integrieren von Service Kompositionen

4.

- Probleme mit:
  - Performance
    - Jeder Webservice hat einen Overhead. Z.B. den SOAP-Envelope
    - Dieser Overhead, multipliziert mit der Anzahl der in der Komposition vorhanden Services, kann durchaus zu Performaceproblemen führen.
- Es ist Sinnvoll alle Kompositionen auf eine Serviceanzahl festzusetzen.

Erweiterungen der Servicefunktionalitäten

5.

- Services müssen nicht immer nur für Serverseitige Programmierlogik stehen.
- Informationen für die GUI
  - Datenformulare können mit übergeben werden um das Userinterface vorzugeben.
  - Daten können in HTML vorformatiert werden, damit der Client das nicht machen muss.
  - Die Darstellungen können Versioniert werden, sodass das Userinterface nicht immer mit übergeben werden muss.
- Es müssen nicht immer nur Textuelle Daten sein die gecached werden. Auch Bilder die auf der GUI erscheinen sind prädestiniert zum Cachen.

Erweiterungen der Servicefunktionalitäten

5.

- Verhaltensweisen der Clients analysieren und verwänden
  - Vorrausschauendes Cachen der Daten die der Client anfragen könnte
  - Frühzeitige Abbrüche analysieren um Service zu verbessern
  - Nicht nur die Verhaltensweisen der User sind wichtig. Auch das Verhalten von Servicen, die unseren Service nutzen, sind interessant.

Integration von SOAP Nachrichten

- SOAP Nachrichten sind ziemlich groß.
- Jede Nachricht sollte möglichst viel Information enthalten, um den Overhead zu minimieren.
- Gibt es Wege Nachrichten zusammen zu fassen?

Integration von SOAP Nachrichten

- SOAP und Kompression
  - Beim anwenden von Kompression muss die Dauer der Kompression und Dekompression berücksichtig werden.
  - Kompression ist nur dann sinnvoll, wenn die Nachrichten groß sind.
  - Bei kurzen Nachrichten ist es wahrscheinlich, daß die Kompression länger dauert, als die Kurze Information einfach los zu schicken.

Integration von SOAP Nachrichten

6.

- Sicherheit
  - Gerade durch die Einfachheit von XML und dessen Darstellungen, gibt es viele Sicherheitsfragen die man sich stellen muss.
  - Gute e-business Lösungen verschlüsseln SOAP Nachrichten immer. Aber dadurch kommt wieder ein Overhead hinzu.
  - Ein Vorteil von SOAP ist, dass es über den Webserver und Port 80 läuft. Die Firewall muss somit keinen weitern Port öffnen.
  - HTTP-Content kann aber durch „Man in the Middle“ attacken verfälscht werden.