INF

FAKULTÄT FÜR
INFORMATIK

Driving Cloud Innovation at OvGU

Dipl.–Wirt.–Inform. Robert Neumann

INF

FAKULTÄT FÜR
INFORMATIK

Ein organischer Produktkatalog

Warum geht es?

- Produkte werden mittels Modellierungssprache abgebildet
- Produkte verfügen über dynamisches Verhalten
- Katalog ist Multi–Mandantenfähig

Was will ich euch zeigen?

- Nutzergenerierter Code in der Cloud
- Data Access Layer
- Multi-Mandantenfähigkeit

Nutzergenerierter Code in der Cloud

Nutzergenerierter Code in der Cloud

- Bibliothek mit Interface/Abstrakter Klasse für die Nutzer
- Upload in die Cloud via Webservice (DLL -> byte[])
  - Speicherung: SQL Azure (auch Blob Storage oder Table Storage möglich)
- Einbinden der DLLs zur Laufzeit

```
Assembly lib = Assembly.Load(ah.BYTECONTENT.ToArray());
```

- Erzeugen einer Instanz

```
ahInst = Activator.CreateInstance(type);
```

Vor- und Nachteile

Vorteile:

- Typisierte Parameter
- Interface ist bekannt und geprüft
- Vollständige Integration im Code (Exceptions…)

Nachteile:

- Sicherheit
- Instanziieren erfordert Zeit

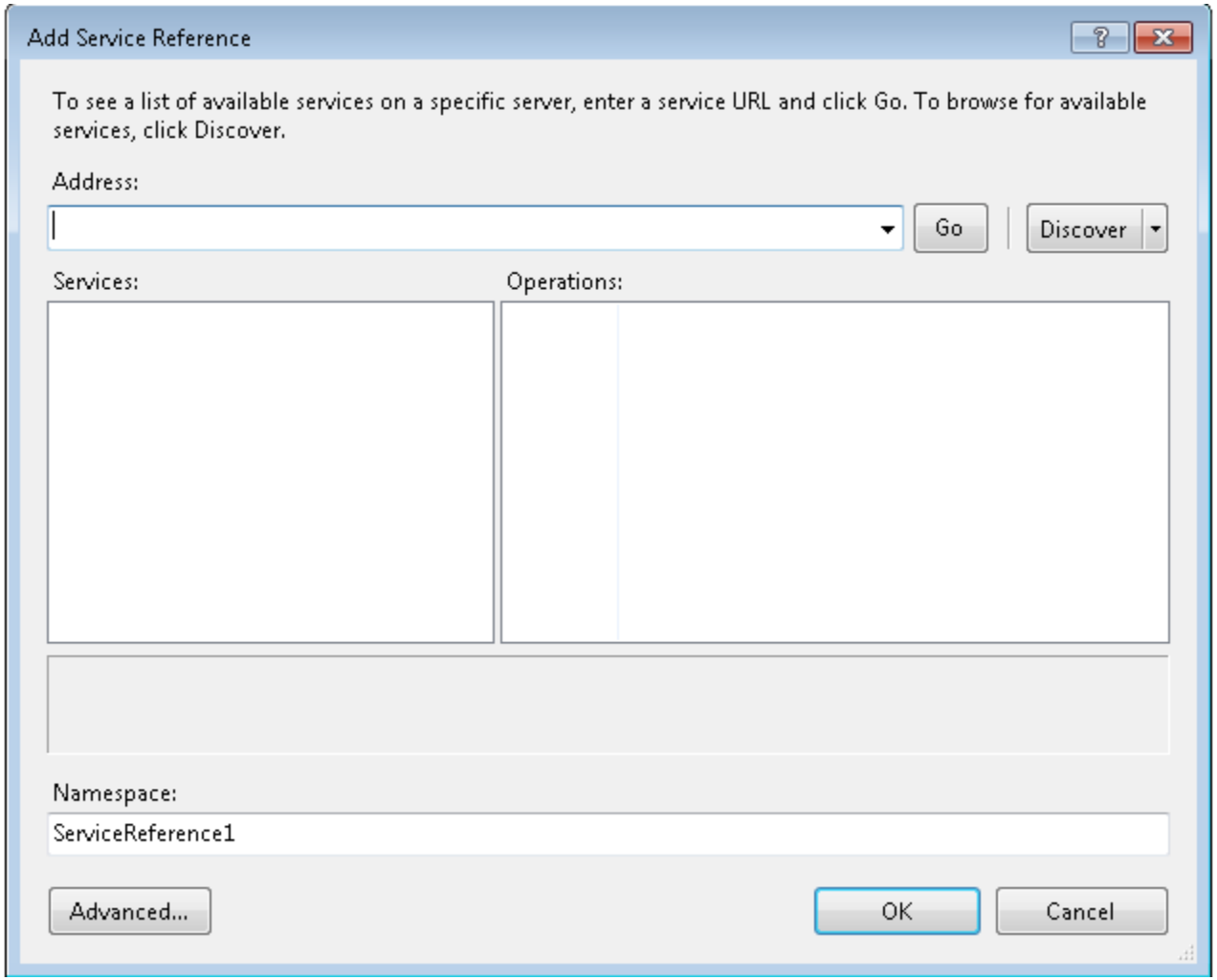
Alternativen

- Webservice–Endpunkte anstelle von DLLs
  - + Sicherheit
  - – Aufwand
  - – Performance
- Ausführung als neuer Prozess
  - +/– Sicherheit
  - – Performance
  - – Nicht–typisiert

Data Access Layer

Data Access Layer

- Ihr kennt:



Aber, große Anzahl von sich ändernden Webservice-Endpunkten

Unser Weg!

- Geteilte Bibliothek für alle Projekte, die die AO–API nutzen
  - Enthält
    - DataContract
    - ServiceContract
    - Geteilte Logik

- Erstellen unseres Proxy mittels ChannelFactory<T>

```
new AuthChannelFactory<IProductChannel>(new EndpointAddr
ess(PathSafe.BasePath + „ProdService.svc")).CreateChannel();
```

- Eigene Implementierung der ChannelFactory für Authentifizierung etc.

Authentifizierung

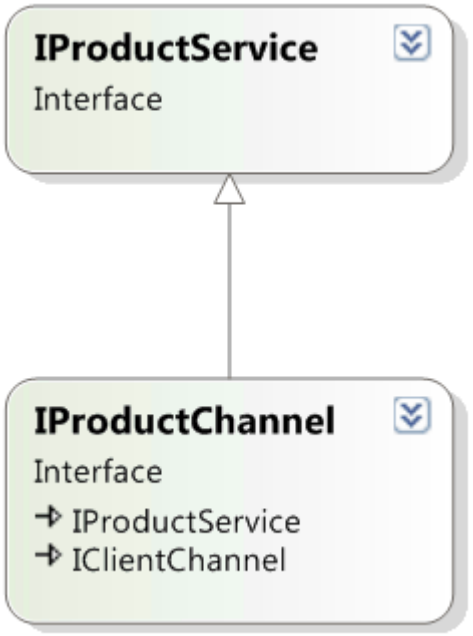
```
public AuthChannelFactory(EndpointAddress _address)
: base(binding, _address)
{
  this.Credentials.UserName.Password = _password;
  this.Credentials.UserName.UserName = _username;

  …
}

public new T CreateChannel()
 {
  T channel = base.CreateChannel();
  channel.Faulted += new EventHandler(Channel_Faulted);
  return channel;
 }
```

ChannelInterface

Kompliziert?Nein!

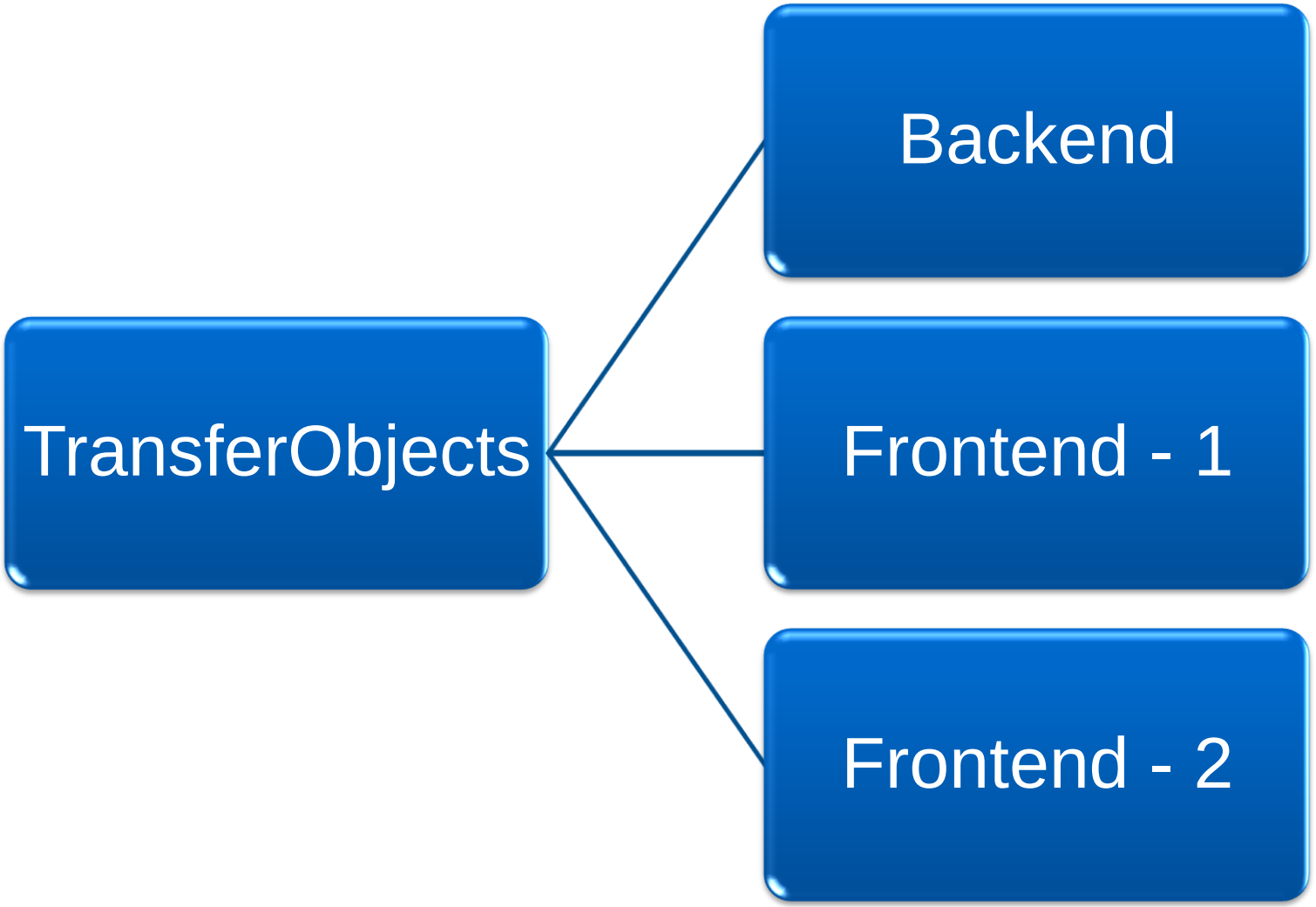



```
public interface IProductChannel
: IProductService, IClientChannel
{
}
```

Aufruf

```
ISearchService searchProxy =
        Ovgu.Cs.Goliath.Util.ServiceUtil.GetSearchService();
```

Zusammenfassend



Multi-Mandantenfähigkeit

Wie lassen sich Daten separieren?

- Jeder Mandant erhält eigene Tabellen und BlobStorage-Bereiche
  - + „echte“ Trennung der Daten
  - – Sehr aufwendig zu pflegen
  - – Overhead
- Trennung der Daten erfolgt mittels Business-Logik
  - + herkömmliche Datenhaltung möglich
  - + leichte Pflege
  - – spezielle Business-Logik möglich

Multi–Mandantenfähigkeit

- Nutzer (normale User, als auch Mandanten) sollen die Teilung nicht bemerken

➔ Einführung einer AppID
  - Jeder Aufruf des Backends muss Authentifiziert werden
  - NutzerId, des aufrufenden Users –> AppID

- Wie zustandslose Webservices authentifizieren?

- WCF hilft uns!

AppID– Schritt 1

- Authentifizierung für WCF aktivieren

```
<serviceBehaviors>
<behavior>
        <serviceCredentials>
                <serviceCertificate … />
                 <userNameAuthentication
        userNamePasswordValidationMode="MembershipProvider"
        membershipProviderName="TableStorageBackend" />
        </serviceCredentials>
         <serviceAuthorization
        roleProviderName="TableStorageRoleBackend"
        principalPermissionMode="UseAspNetRoles" />
```

AppId– Schritt 2

- Auslesen des NutzerId in WCF

```
(Guid) Membership.GetUser(_userName).ProviderUserKey
```

- Vorhalten des Wertes in einem Singleton, um mehrfaches auslesen zu vermeiden

AppId– Schritt 3

- Separieren der Daten
- Jede TableStorageEntity beginnt mit AppID –> Filterung problemlos über PK
- TableServiceContext übernimmt diese Aufgabe selbstständig
  - Parsen des Linq–Queries und generieren des PK

Zukünftige Aufgabe– ArbiterOne

Themen

- Mobil–Client, der interaktiv Produkte in der Umgebung darstellt
- Product–Matching–Engine
- UI zur Unterstützung der Konstruktion eines Queries

INF

FAKULTÄT FÜR
INFORMATIK

OvGUApp

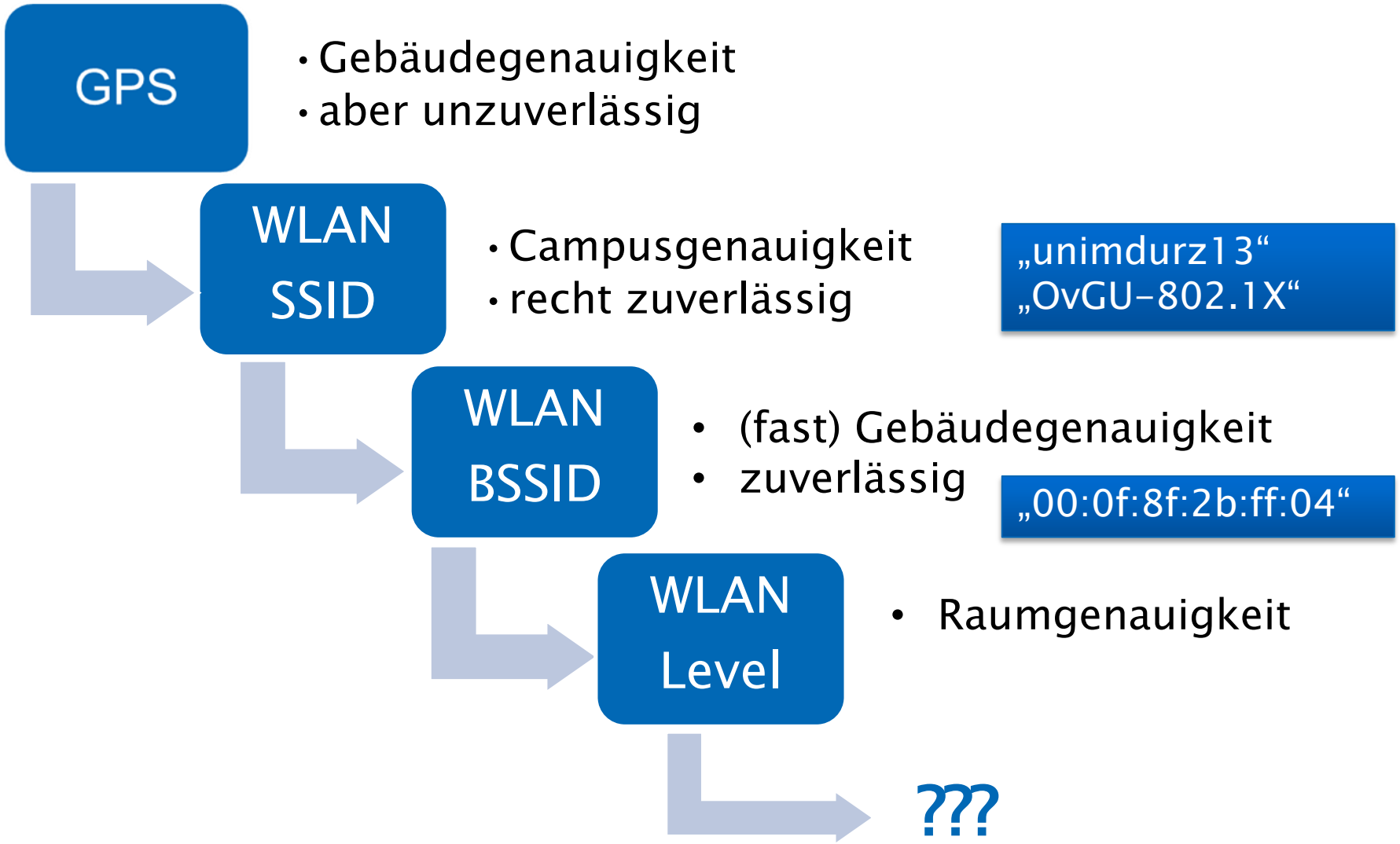
Norman Peitek und Marcus Pöhls

Agenda

1. Grundidee
2. Anwendungsmöglichkeiten
3. Frontend – Android
4. Backend – REST, JSON und Azure
5. Livedemo

Grundidee

Grundidee – Lokalisierung des Handy



Anwendungsmöglichkeiten

Sag mir, wo ich bin

- Lehrevaluation
- Foliendownload
  - www.aufobe.de – @aufobe

Sag mir, wie viele Leute an einem Ort sind

- Mensa voll?

Sag mir, wer wo ist

- Nicht in der OvGUApp!
- Bereitschaftsarzt im Krankenhaus

Frontend – Android

Warum Android

- Persönliche Präferenz
- Testgerät vorhanden

WiFi – API	Ja	Vermutlich?	Nein (private?)
Background	Ja	(noch) Nein	Teilweise → Nein
REST	Ja		
JSON	Ja		

Backend – REST, JSON und Azure

Warum REST und nicht SOAP

- Google = REST
- Android unterstützt nativ kein SOAP
    - abhängig von Third Party Library

Backend – REST, JSON und Azure

Warum JSON und nicht XML

- Für uns: kein SOAP – kein XML ☺
- Googles Gson–Library

Backend – REST, JSON und Azure

Wie bringt man dem WCF nun JSON bei

```
[OperationContract]
[WebInvoke(
    UriTemplate = "/InsertFingerprint/",
    Method = "POST",
    ResponseFormat = WebMessageFormat.Json,
    RequestFormat = WebMessageFormat.Json
    )
]
public BoolResponse InsertFingerprint(FingerprintLocation _footprint)
{
```

Backend – REST, JSON und Azure

JSON vs. XML

- Vorlesung 6 – Web Services, Folie 201

	JSON	XML	YAML
Network bandwidth	++	+ (ZIP)	+
Processing performance	-	++	+
Human readability	--	++	+
Schema validation	--	++ (XSLT)	--
Partial processing	--	++	--

Backend – REST, JSON und Azure

JSON Request-Body – 196 Zeichen

```
{
        "houseName":"String content",
        "houseNumber":"String content",
        "room":"String content",
        "wifiNetworks":[{
                "BSSID":"String content",
                "SSID":"String content",
                "frequency":2147483647,
                "level":2147483647
        }]
}
```

Backend – REST, JSON und Azure

XML Request-Body – 374 Zeichen

```
<FingerprintLocation xmlns="http://schemas.datacontract.org/2004/07/OvguAppRest">
  <houseName>String content</houseName>
  <houseNumber>String content</houseNumber>
  <room>String content</room>
  <wifiNetworks>
    <WifiClass>
      <BSSID>String content</BSSID>
      <SSID>String content</SSID>
      <frequency>2147483647</frequency>
      <level>2147483647</level>
    </WifiClass>
    <WifiClass>
      <BSSID>String content</BSSID>
      <SSID>String content</SSID>
      <frequency>2147483647</frequency>
      <level>2147483647</level>
    </WifiClass>
  </wifiNetworks>
</FingerprintLocation>
```

Backend – REST, JSON und Azure

JSON Response-Body – 18 Zeichen

```
{
        "response":true
}
```

Backend – REST, JSON und Azure

XML Request-Body – 122 Zeichen

```
<BoolResponse xmlns="http://schemas.datacontract.org/2004/07/OvguAppRest.Classes">
  <response>true</response>
</BoolResponse>
```

Backend – REST, JSON und Azure

Warum Windows Azure

- leichter Einstieg in C# – sehr ähnlich zu Java
- Know-how und Support durch Robert und Matze
- Cloud Computing mit Azure als Veranstaltung

Livedemo

Es ist soweit …

Fragen und Antworten

@OvGUApp

INF

FAKULTÄT FÜR
INFORMATIK

AzureMD Resource Market

FIN-SMK Projekt

Gliederung

1. Projektvorstellung
2. Motivation und Ziele
3. Teamvorstellung
4. Architektur
    1. Frontend
    2. Backend
    3. Beispiel: Kommunikation
5. Live–Demo
6. Ausblick

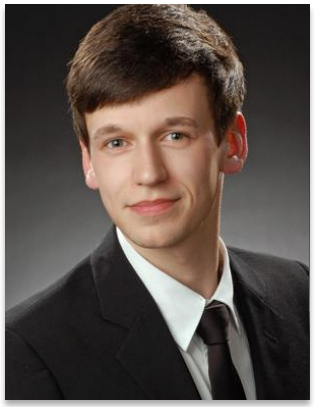
Projektvorstellung

- Marktplatz für die Vermietung von Computer–Ressourcen

Motivation und Ziele

- Ressourcen
  - Kostengünstiges Mieten
  - Zentrale Plattform zur Bereitstellung
- Unkomplizierte Installation und Verwendung
  - Keine spezielle Zielgruppe
  - Schnelle Bereitstellung
  - Einfaches Ressourcenmanagement
- Schnell, Individuell, Plattformunabhängig

Teamvorstellung



Projektmanager

Eike Kirschner

Chefentwickler

Christoph Papmeyer

Entwickler

Marco Holzknecht

Entwickler

Ronny Garz

Teamleiter Frontend, Chefentwickler

Friedrich Lüder

Entwickler

Andreas Schmückert

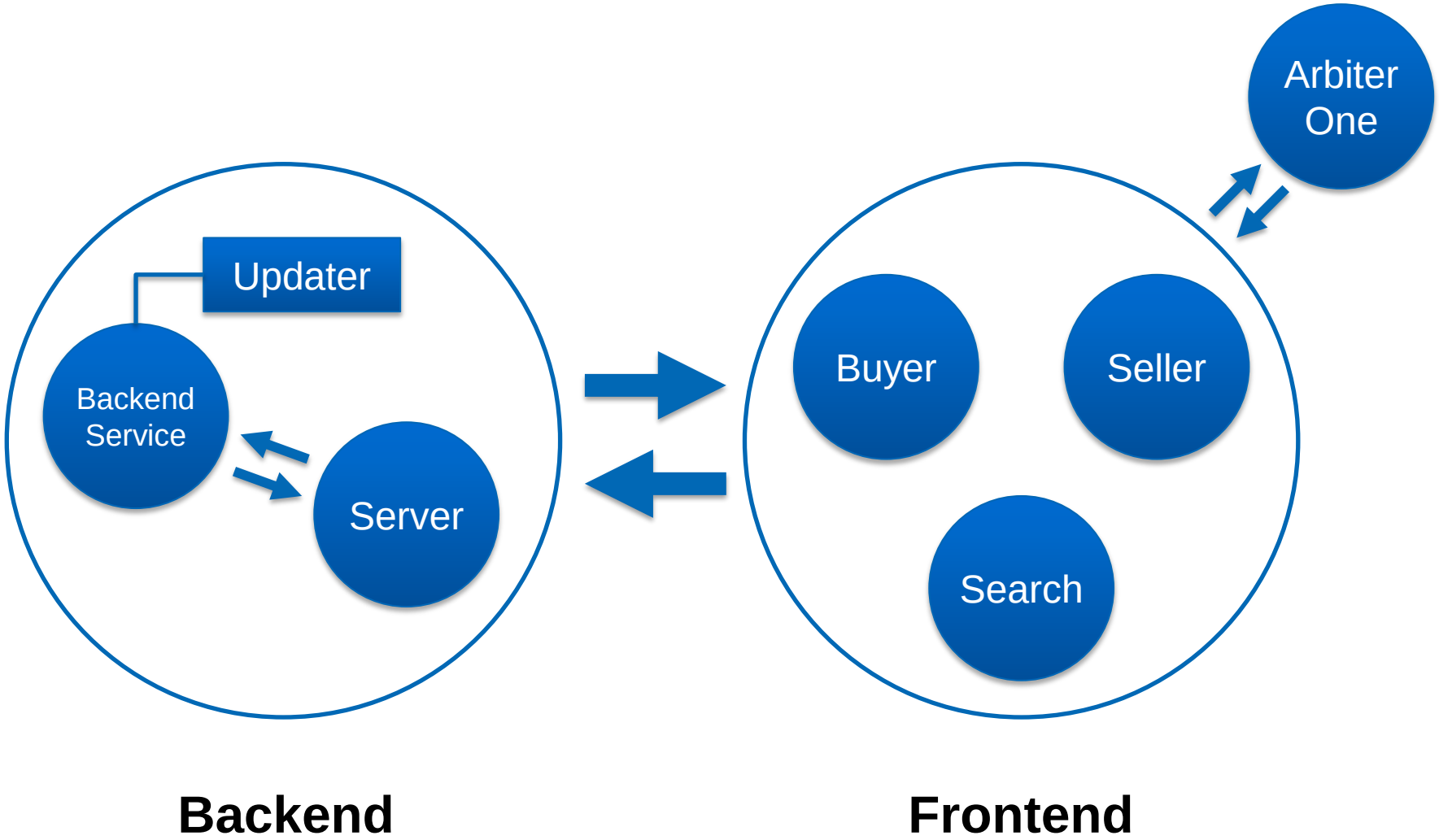
Entwickler

Olga Egorow

Qualitätsmanagement

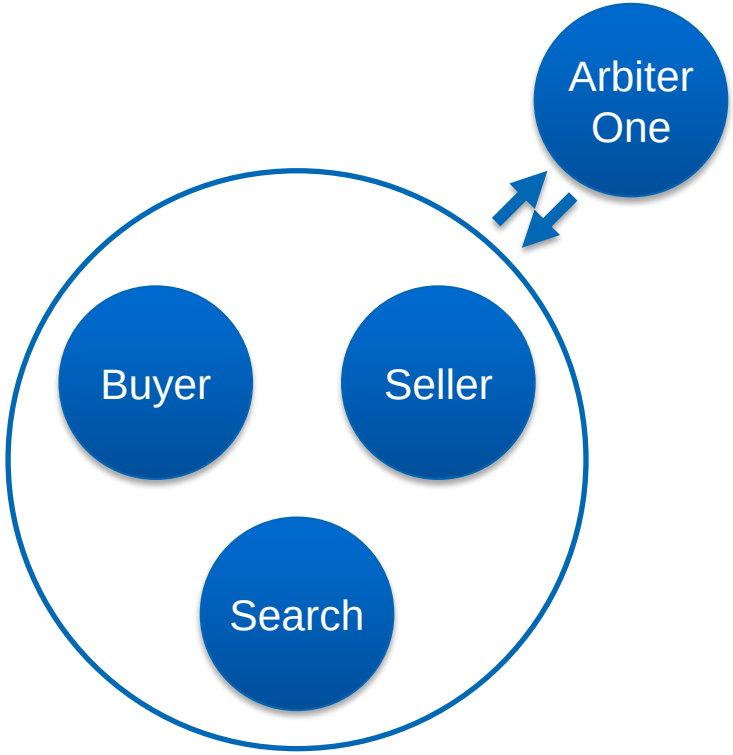
Christina Lenz

Architektur



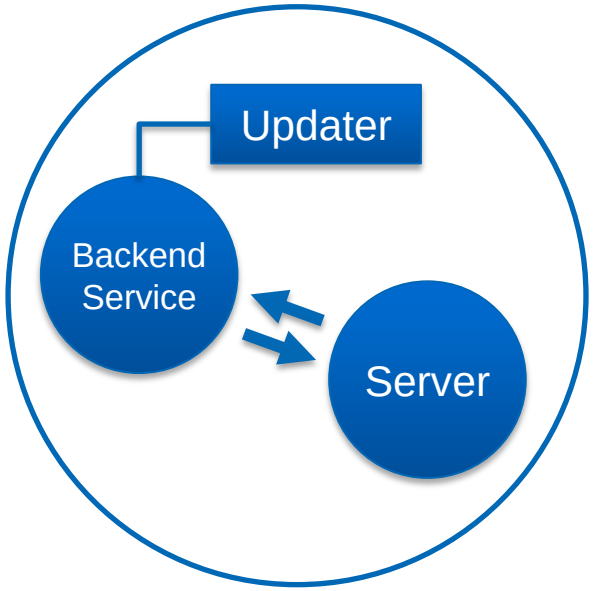
Frontend

- Ressourcen
  - Veröffentlichen
  - Zeitraumbasierte Angebote
  - Live–Suche
  - Image–Verwaltung
- User–Verwaltung
- Nutzergesteuerte Preisgestaltung
- Zentrale Steuerung des Backends
  - Statusinformationen
  - Start, Stop, Restart, Upgrade, Resize, ...

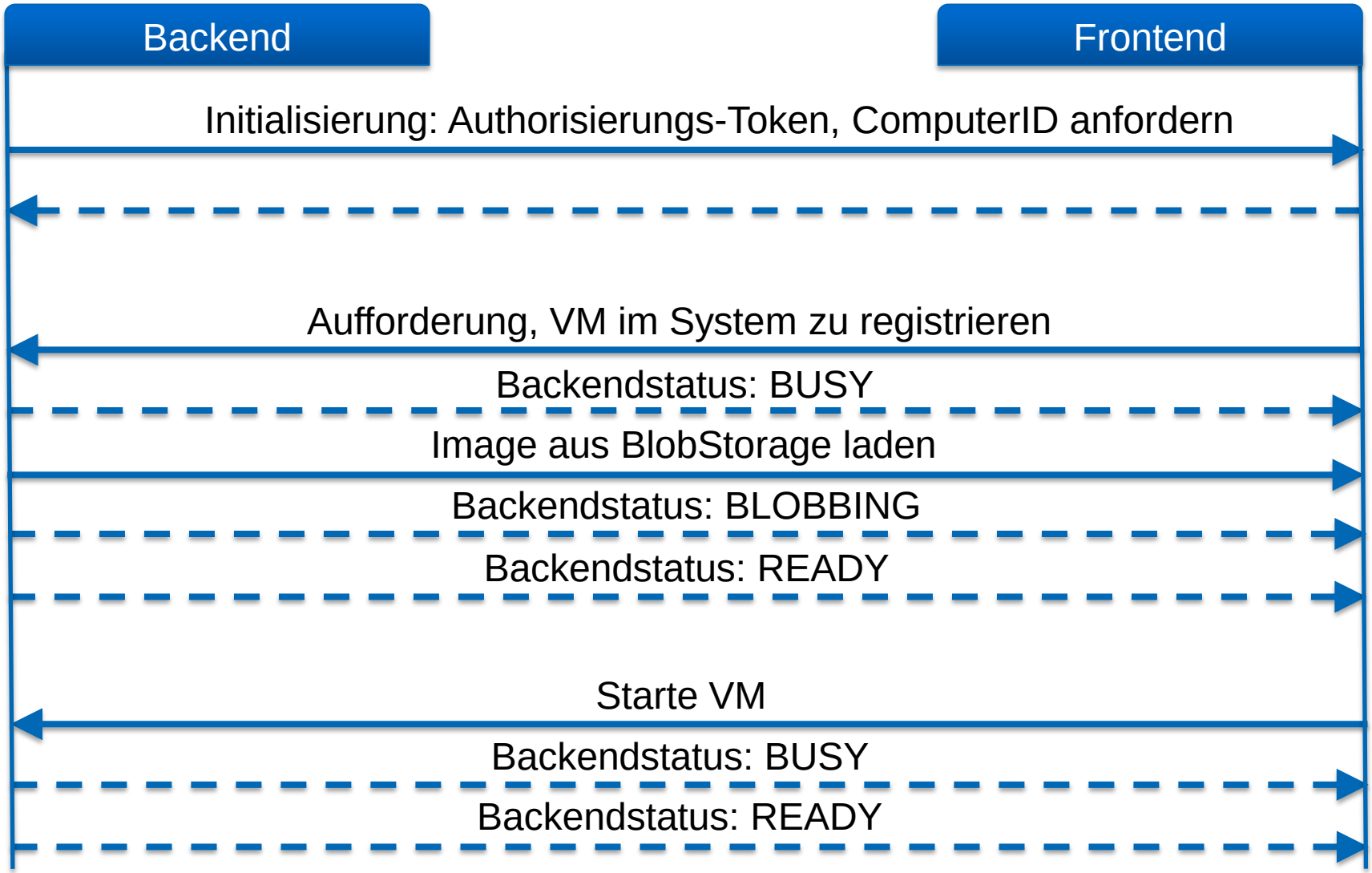


Backend

- Backend Service
  - Plattformunabhängig
  - Steuerung des Servers
  - Kommunikation mit Frontend
  - Automatisiertes Upgrade
- VM–Management
  - Statusinformationen
  - Start, Stop, Restart
  - Automatisierte Installation
  - Dynamische Ressourcenverwaltung



Beispiel: Initiale Kontaktaufnahme



Live-Demo

Ausblick

- Frontend
  - Benutzerfreundlichkeit verbessern
  - Überarbeitung des Designs
  - Kommunikation mit Backend vervollständigen
- Backend
  - Unterstützung weiterer Server
  - Dynamische Ressourcenverwaltung verbessern

Fragen?

Danke für Eure Aufmerksamkeit!