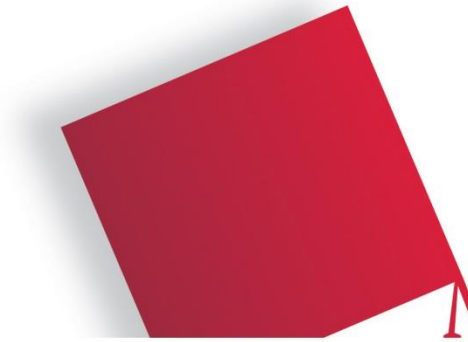


HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

Grundbausteine Java-basierter Web-Applikationen

FWP Aktuelle Technologien zur Entwicklung verteilter Java-Anwendungen

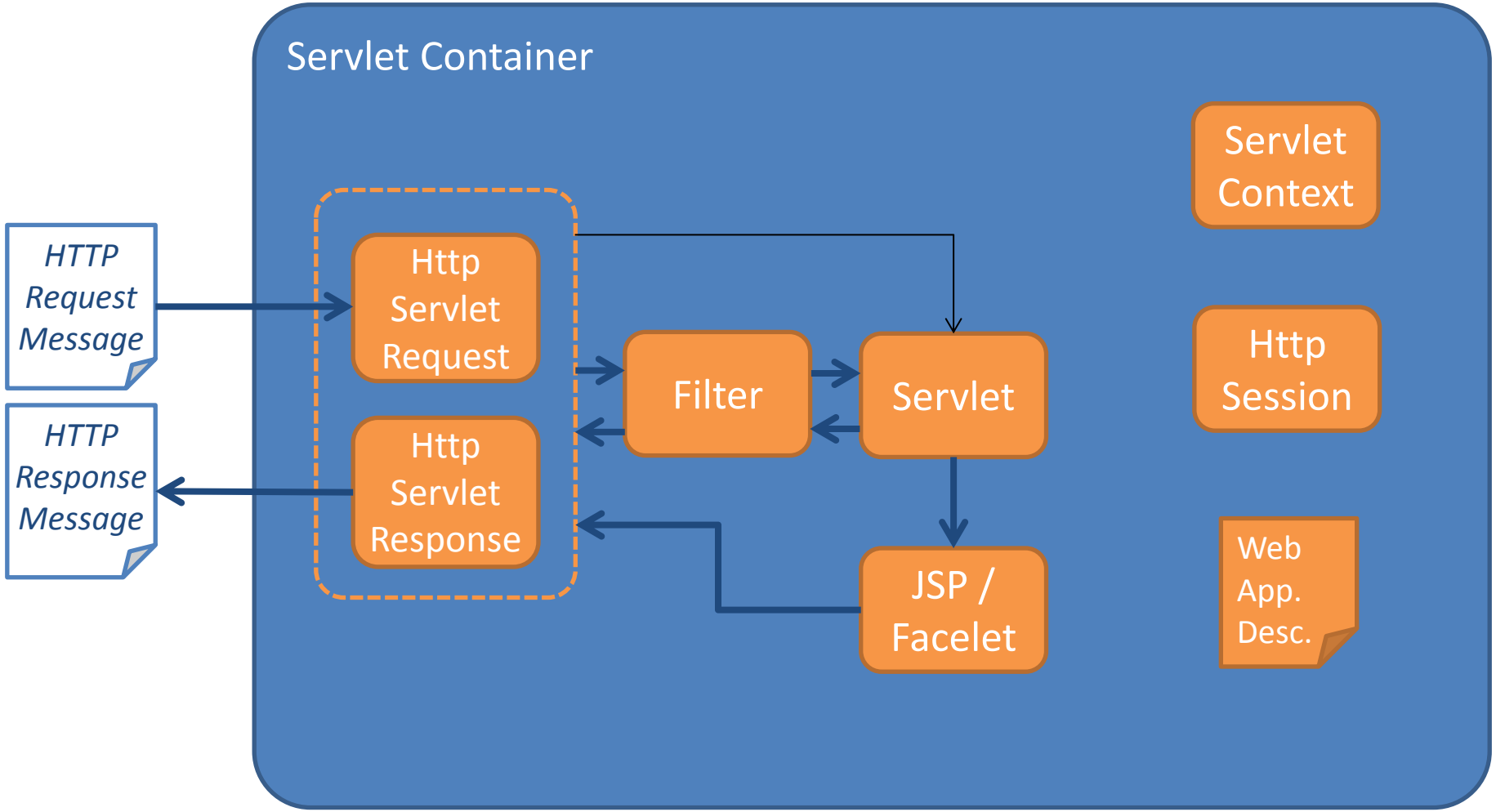
Michael Theis, Lehrbeauftragter

GRUNDBAUSTEINE JAVA-BASIERTER WEB-APPLIKATIONEN

Web Applikationen mit Java

- Werden als *Web Application Archive (WAR)* verpackt
- Können eigenständig oder als Teil einer *Enterprise Application* installiert werden
- Ihre Komponenten werden über den *Web Application Descriptor (web.xml)* oder über *Annotations* konfiguriert
- *Kontextpfad* in der URL identifiziert eine Web-Applikation: *http://localhost:12345/**jeetrain**/...*

Servlet API auf einen Blick



Web Application Descriptor

- XML-Konfigurationsdatei /WEB-INF/web.xml
- Definiert alle Komponenten und Artefakte:
  - Servlet Kontext Parameter
  - Servlets und Filter und deren URLs
  - Servlet Context Listener
  - Security Constraints, Security Roles, Art der Anmeldung
- Kommt in zwei Ausprägungen:
  - hersteller-neutral: /WEB-INF/web.xml
  - hersteller-spezifisch: (z.B.) /WEB-INF/glassfish-web.xml

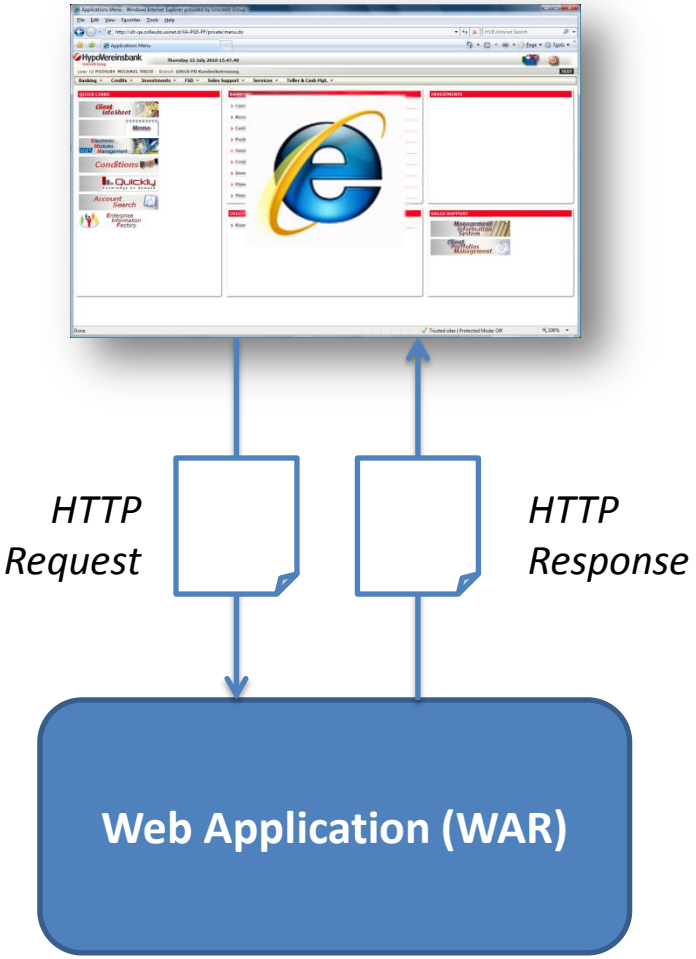
Servlet

- Endpunkte für HTTP-Nachrichten
  - Konsumieren HTTP-Requests
  - Produzieren HTTP-Responses
- Konfiguration über */WEB-INF/web.xml* oder *@WebServlet*
  - Servlet-Name, Servlet-Klasse, Servlet-URL-Mapping
- Eigene Servletklassen müssen entweder Interface *javax.servlet.Servlet* oder *javax.servlet.http.HttpServlet* implementieren

Filter

- Ermöglichen das Abfangen und Manipulieren von HTTP-Request- und HTTP-Response-Nachrichten
- Unterstützen beim Einbringen von Aspekten auf HTTP-Request- und HTTP-Response-Ebene
- Konfiguration über */WEB-INF/web.xml* oder *@WebFilter*
  - Filter-Name, Filter-Klasse, Filter-URL-Mapping
- Eigene Filterklassen müssen Interface *javax.servlet.Filter* implementieren

Request und Response



- Browser und Web-Applikation tauschen HTTP Nachrichten aus:
  - HTTP Requests lösen spezifische Aktionen innerhalb der Web-Applikation aus
  - HTTP Responses enthalten das Ergebnis der Aktion (im Allgemeinen als HTML-Dokument)
- Text-basierte HTTP-Nachrichten werden durch Objekte repräsentiert:
  - HTTPServletRequest
  - HTTPServletResponse
- *HTTPServletRequest*
  - Bestimmt Art des Zugriffs (GET/POST/…)
  - Definiert welche Aktion ausgelöst werden soll (URL)
  - Bietet Zugriff auf übergebene Parameter
- *HTTPServletResponse*
  - Bietet einen Ausgabestrom, in den (unter anderem) HTML Code geschrieben werden kann

Session

- Repräsentiert durch *javax.servlet.http.HttpSession*
- Identifizieren einen Benutzer über mehrere Requests hinweg
- Informationen zum Benutzer können als Sessionattribute in der Session gespeichert werden
- HTTP-Sessions sind gebunden an einen bestimmten Benutzer und eine bestimmte Web-Applikation

Lebenszyklus einer HTTP Session

- Neue Sessions werden erzeugt beim ersten Aufruf von *HttpServletRequest.getSession()*
- Servlet Container identifiziert existierende Sessions über ein Cookie identifiziert
- Session existiert
  - entweder bis zur expliziten Freigabe über *HttpSession.invalidate()*
  - oder bis zur impliziten Freigabe durch den Container nach Ablauf des HTTP-Session-Timeouts

Möglichkeiten der Datenhaltung

- Web-Applikationen müssen Daten über einen bestimmten Zeitraum halten
  - Zwischen aufeinanderfolgenden HTTP-Requests
  - Solange ein Benutzer mit der Web-Applikation arbeitet
- Es gibt drei Möglichkeiten zur Datenspeicherung:
  - Request-Attribute / request scope
  - Session-Attribute / session scope
  - Servlet-Context-Attribute / application scope
- Manche Frameworks bieten erweiterte Scopes

Gedanken zur Datenhaltung

- Sinnvolle Datenhaltung ist Voraussetzung für skalierbare Applikationen
  - Halten Sie Daten so kurz wie möglich vor
  - Denken Sie darüber nach, was Sie in Sessions speichern wollen und wie lange die Daten dort bleiben sollen
  - Benutzer verlassen Web-Applikationen u.U. unerwartet
- Nutzen Sie Frameworks mit Datenmanagement
- Seien Sie darauf vorbereitet, dass mehrere Browserfenster die gleiche Session nutzen

Java Server Pages (JSP)

- Bringen HTML und Java zusammen
  - HTML-Code und Java-Code lassen sich in einer Datei mischen
- Aus JSPs werden on-demand von einem JSP Compiler Java Sourcen generiert und kompiliert
- Generierte JSP-Klassen entsprechen einem Servlet

JSP Tags

- Erweitern den Sprachraum von HTML
- Werden bei jedem Anzeigen einer JSP durchlaufen
- Ermöglichen die Ausführung komplexer Präsentations-Logik
- Können dynamisch HTML-Code in eine Seite einfügen
- Werden zusammengefasst in JSP Tag Libraries
- Pro Tag ein Eintrag im Tag Library Deskriptor (TLD) und eine Tag Handler Klasse

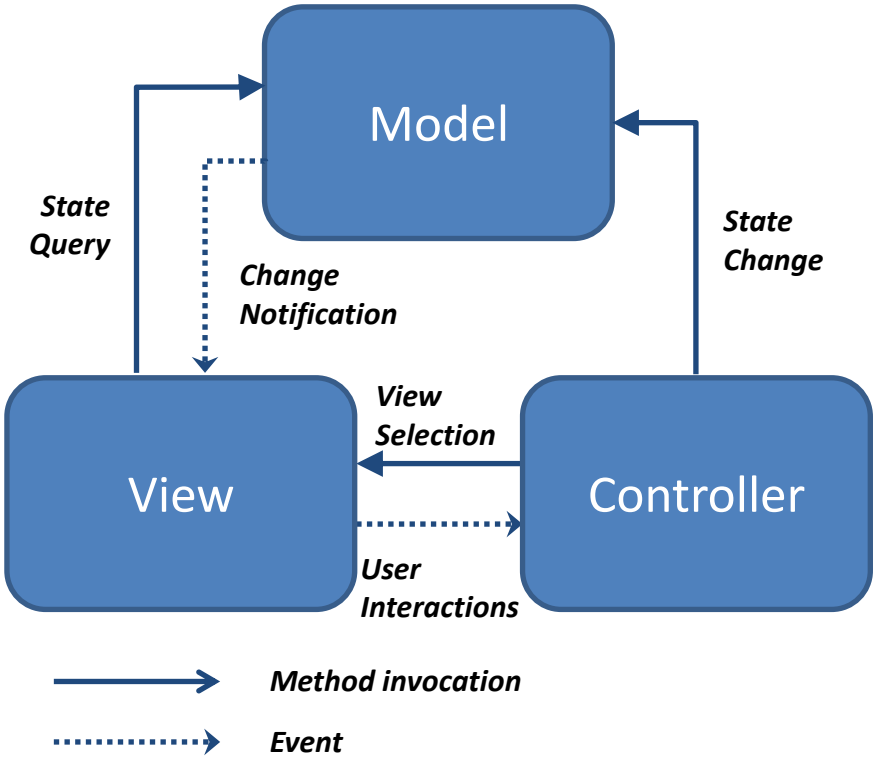
JSP Standard Tag Library (JSTL)

- Stellt immer wieder benötigte Basis-Funktionalität in Form einfacher Tags zur Verfügung:
  - Ausgabe von Werten aus Request und Session: *c:out*
  - Iteration über Arrays oder Collections: *c:forEach*
  - Bedingte Sprünge: *c:if* *c:choose*/*c:when*/*c:otherwise*
  - Internationalisierung: *fmt:message*, *fmt:format**
  - Manipulation von XML-Dokumenten: *x:**

BASISARCHITEKTUR WEB-BASIERTER APPLIKATIONEN

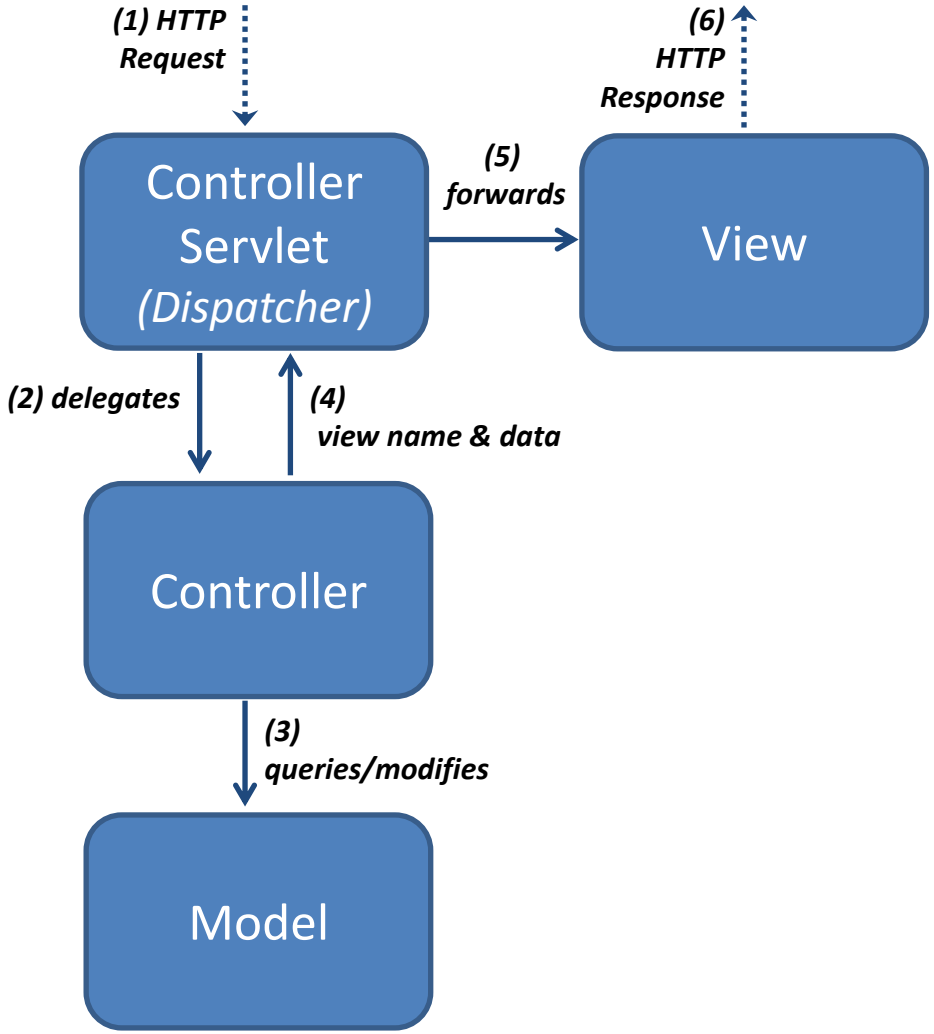
Model-View-Controller

- Regelt die Gliederung einer Web-Applikation in Artefakte mit wohl-definierten Rollen und Verantwortlichkeiten
- ***Model***: Repräsentiert die Daten inkl. der Regeln, nach denen auf diese Daten zugegriffen werden darf bzw. wie diese Daten verändert werden dürfen.
- ***View***: Stellt die im Model enthaltenen Daten für einen menschlichen Benutzer lesbar dar. Ermöglicht dem Benutzer Interaktion mit dem Model.
- ***Controller***: Übersetzt die Interaktionen des Benutzers mit dem View in Aktionen, die durch das Model ausgeführt werden sollen und dabei ggf. den Zustand des Models ändern.



Model 2 Architektur

- **Motivation**: Ereignisbasierte Benachrichtigungen funktionieren im Web nicht besonders gut
- Die Model 2 Architektur versucht dieses Problem mit einer ver-einfachten MVC-Adaption zu lösen:
  - Ein ***Controller Servlet*** leitet eingehende Requests basierend auf deren URL und Parametern an einen Controller weiter.
  - Der ***Controller*** übernimmt den gewünschten Zugriff auf das Model, fügt dem aktuellen Request die anzuzeigenden Daten hinzu und liefert den nächsten View zurück.
  - Das Controller Servlet leitet den Request auf den nächsten View um.
  - Der ***View*** stellt die anzuzeigenden Daten dar.



SECURITY IN WEB-APPLIKATIONEN

Security Constraints

- Stellen sicher, dass geschützte Bereiche nur von Benutzern mit den passenden Rechten betreten werden
- Deklarative Sicherheit über *web.xml* oder *@ServletSecurity*
- Nur authentisierte und autorisierte Benutzer können geschützte Bereiche betreten
- Nicht authentisierte Benutzer werden automatisch zur Anmeldung gezwungen

Security Roles

- Repräsentieren logische Zugriffsrechte einer Anwendung
- Müssen auf physische Rollen/Gruppen abgebildet werden („Admin“ entspricht „JTR00123“)
- Bestimmen in Verbindung mit Security Constraints die erwarteten Zugriffsrechte
- Können programmatisch über *HttpServletRequest.isUserInRole(String)* abgefragt werden

Basic Authentication

- Einfachster Weg der Authentisierung

```
<security-constraint>
  <display-name>ProtectedPages</display-name>
  <web-resource-collection>
    <web-resource-name>ProtectedPages</web-resource-name>
    <url-pattern>/ping.jsp</url-pattern>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint><role-name>Authenticated</role-name></auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<login-config><auth-method>BASIC</auth-method></login-config>
<security-role><role-name>Authenticated</role-name></security-role>
```

Was will ich schützen?

Erforderliche Rolle

Verschlüsselte Kommunikation

Basic authentication

- Browser übernimmt Abfrage der Logindaten über Popup

Form-based Login

- Anmeldung über ein selbst erstelltes Formular

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/login/login.jsp</form-login-page>
    <form-error-page>/login/loginFailed.jsp</form-error-page>
  </form-login-config>
</login-config>
```

- Anmeldung wird vom Servletcontainer über konfigurierte Login-Seite erzwungen
- Login-Seite muss eine vordefinierte Aktion auslösen und die Eingabefelder müssen bestimmte Namen haben

Aufbau eines Login-Formulars

- Um korrekt zu funktionieren, muss die Login-Seite den folgenden Namenskonventionen genügen:
  - Ausgelöste Aktion muss *j_security_check* heißen
  - Eingabefeld für den Benutzernamen muss *j_username* heißen
  - Eingabefeld für das Passwort muss *j_password* heißen

```
<form id="loginForm" name="loginForm" action="j_security_check"
method="post">
    <label id="userNameLabel" for="j_username"> User name:</label>
    <input id="j_username" type="text" name="j_username" size="16" />
    <label id="passwordLabel" for="j_password"> Password:</label>
    <input id="j_password" type="password" name="j_password" size="16" />
    <input id="loginButton" type="submit" name="loginButton" value="Login"/>
</form>
```

Programmatische Sicherheit

- Über *javax.servlet.HttpServletRequest*:
  - *getUserPrincipal()* liefert einen *java.security.Principal* zurück, der den angemeldeten Benutzer repräsentiert:

```
Principal principal = request.getUserPrincipal();
if (principal != null) {
    String authenticatedUserId = principal.getName();
}
```

  - *isUserInRole()* prüft, ob der angemeldete Benutzer eine bestimmte Rolle besitzt:

```
if (request.isUserInRole("Authenticated")) {
    // do something only accessible to authenticated users
} else {
    // throw exception indicating that authentication is required
    throw new IllegalStateException("Requires authentication!");
}
```

Fragen?

ANHANG

Quellen

- Eric Jendrock et. al.
  The Java EE 7 Tutorial Part III Kapitel 6, 17
  http://docs.oracle.com/javaee/7/tutorial/partwebtier.htm
  Oracle September 2014

Kontakt

Michael Theis
Lehrbeauftragter Hochschule München

email michael.theis@hm.edu
mobile + 49 170 5403805
web http://www.tschutschu.de/Lehrauftrag.html