KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

07/2012

Jahrgang 5

KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

Sie können die elektronische Form des KaffeeKlatschs
monatlich, kostenlos und unverbindlich
durch eine E-Mail an

abo@bookware.de

abonnieren.

Ihre E-Mail-Adresse wird ausschließlich für den Versand
des KaffeeKlatschs verwendet.

Editorial

Entmottung

Während meiner Studienzeit habe ich einmal den unvergesslichen Satz gehört, dass nur derjenige sein Programm *debuggt*, der es nicht verstanden hat.[1] Bisher hatte ich den Satz immer benutzt, um rechtzufertigen, dass ich mich mit dem jeweils aktuell verfügbaren *Debugger* nicht auskenne. Die Crux ist aber, dass man es eben all zu oft nicht mit dem eigenen Programm zu tun hat.

Beobachtet man Anfänger dabei wie sie eine nicht mehr ganz so triviale Übung programmieren, dann stellt man immer wieder fest, dass sie ihr Programm mehr oder weniger geschickt mit Bildschirmausgaben bestücken. Diese sollen anscheinend dabei helfen, den Ablauf ihres Programms nachvollziehbarer zu machen. Das ist die einfachste Art des *Debugging*, die ohne Zuhilfenahme von Werkzeugen möglich ist.

Hat man es mit fremden Programmen zu tun, die verstanden werden wollen, bedient man sich typischerweise eines Debuggers, dessen wichtigste Eigenschaft es ist ebenfalls an Bildschirmausgaben zu kommen, allerdings ohne dabei die Quellen verändern zu müssen (oder überhaupt einen Zugriff auf diese zu haben). Das Debuggen hilft dann dabei, die Abläufe, Änderungen und Strukturen besser zu verstehen.

Ob man ein Programm mit der einen oder anderen Motivation debuggt, ein Bekannter von mir hat das neulich sehr schön formuliert: Das Debuggen erlaubt das Verifizieren des Gedankenmodells, das man sich über das Programm gemacht hat. So führen die Ausgaben entweder zu einer Bestätigung dieses Modells oder machen eine Verfeinerung bzw. Anpassung des Modells nötig – bis das Programm ausreichend verstanden ist oder in Kombination mit Änderungen den gewünschten Zweck erfüllt.

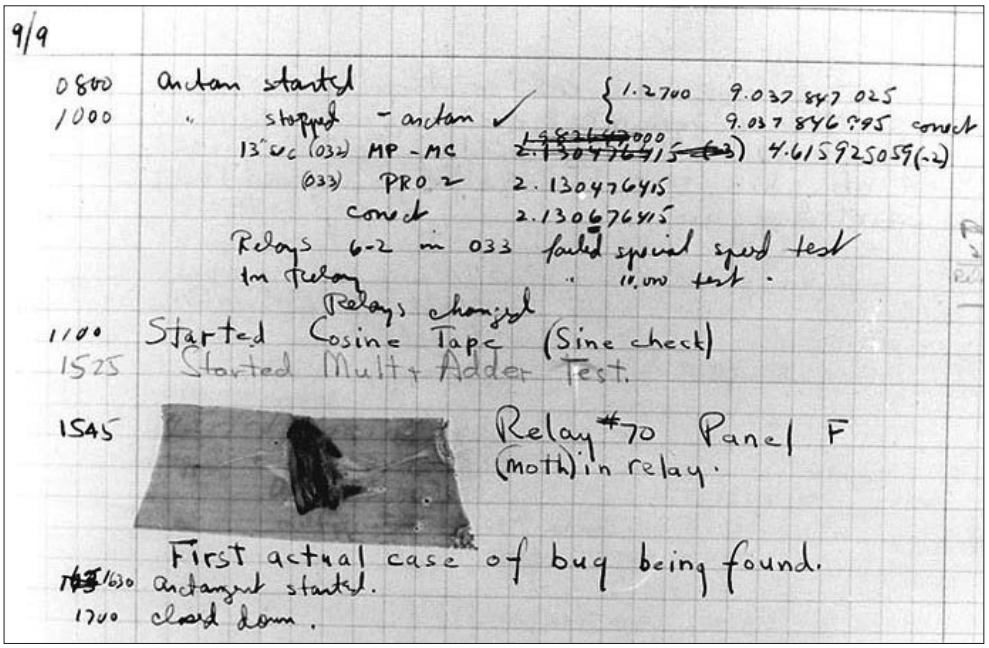
Eine sehr feinsinnige Beobachtung über die schwierige, anspruchsvolle und allgegenwärtige Aufgabe unserer täglichen Arbeit.

So wünsche ich einen möglichst Debug-freien Restsommer,

Ihr Michael Wiedeking
Herausgeber

PS: Es sei noch wegen der Überschrift erwähnt, was der Duden zum Thema entmotten sagt: „vor der Ingebrauchnahme an etwas das Mottenschutzmittel entfernen“ – was etwas völlig anderes ist.

[1] Angeblich stammt der Satz von Nikolaus Wirth, dem Erfinder von *Pascal*, was ich über all die Jahre leider nicht verifizieren konnte.



Quelle: U.S. Naval Historical Center Online Library Photograph NH 96566-KN

Erster echter *Computer Bug* – eine Motte, die sich in einem *Relais* verfangen hatte und am 9. September 1947 artgerecht dokumentiert wurde.

Beitragsinformation

Der KaffeeKlatsch dient Entwicklern, Architekten, Projektleitern und Entscheidern als Kommunikationsplattform. Er soll neben dem Know-how-Transfer von Technologien (insbesondere Java und .NET) auch auf einfache Weise die Publikation von Projekt- und Erfahrungsberichten ermöglichen.

Beiträge

Um einen Beitrag im KaffeeKlatsch veröffentlichen zu können, müssen Sie prüfen, ob Ihr Beitrag den folgenden Mindestanforderungen genügt:

- Ist das Thema von Interesse für Entwickler, Architekten, Projektleiter oder Entscheider, speziell wenn sich diese mit der Java- oder .NET-Technologie beschäftigen?
- Ist der Artikel für diese Zielgruppe bei der Arbeit mit Java oder .NET relevant oder hilfreich?
- Genügt die Arbeit den üblichen professionellen Standards für Artikel in Bezug auf Sprache und Erscheinungsbild?

Wenn Sie uns einen solchen Artikel, um ihn in diesem Medium zu veröffentlichen, zukommen lassen, dann übertragen Sie Bookware unwiderruflich das nicht exklusive, weltweit geltende Recht

- diesen Artikel bei Annahme durch die Redaktion im KaffeeKlatsch zu veröffentlichen
- diesen Artikel nach Belieben in elektronischer oder gedruckter Form zu verbreiten
- diesen Artikel in der Bookware-Bibliothek zu veröffentlichen
- den Nutzern zu erlauben diesen Artikel für nicht-kommerzielle Zwecke, insbesondere für Weiterbildung und Forschung, zu kopieren und zu verteilen.

Wir möchten deshalb keine Artikel veröffentlichen, die bereits in anderen Print- oder Online-Medien veröffentlicht worden sind.

Selbstverständlich bleibt das Copyright auch bei Ihnen und Bookware wird jede Anfrage für eine kommerzielle Nutzung direkt an Sie weiterleiten.

Die Beiträge sollten in elektronischer Form via E-Mail an redaktion@bookware.de geschickt werden.

Auf Wunsch stellen wir dem Autor seinen Artikel als unveränderlichen PDF-Nachdruck in der kanonischen KaffeeKlatsch-Form zur Verfügung, für den er ein unwiderrufliches, nicht-exklusives Nutzungsrecht erhält.

Leserbriefe

Leserbriefe werden nur dann akzeptiert, wenn sie mit vollständigem Namen, Anschrift und E-Mail-Adresse versehen sind. Die Redaktion behält sich vor, Leserbriefe – auch gekürzt – zu veröffentlichen, wenn dem nicht explizit widersprochen wurde.

Sobald ein Leserbrief (oder auch Artikel) als direkte Kritik zu einem bereits veröffentlichten Beitrag aufgefasst werden kann, behält sich die Redaktion vor, die Veröffentlichung jener Beiträge zu verzögern, so dass der Kritisierte die Möglichkeit hat, auf die Kritik in der selben Ausgabe zu reagieren.

Leserbriefe schicken Sie bitte an leserbrief@bookware.de. Für Fragen und Wünsche zu Nachdrucken, Kopien von Berichten oder Referenzen wenden Sie sich bitte direkt an die Autoren.

Werbung ist Information

Firmen haben die Möglichkeit Werbung im KaffeeKlatsch unterzubringen. Der Werbeteil ist in drei Teile gegliedert:

- Stellenanzeigen
- Seminaranzeigen
- Produktinformation und -werbung

Die Werbeflächen werden als Vielfaches von Sechsteln und Vierteln einer DIN-A4-Seite zur Verfügung gestellt.

Der Werbeplatz kann bei Frau Natalia Wilhelm via E-Mail an anzeigen@bookware.de oder telefonisch unter 09131/8903-16 gebucht werden.

Abonnement

Der KaffeeKlatsch erscheint zur Zeit monatlich. Die jeweils aktuelle Version wird nur via E-Mail als PDF-Dokument versandt. Sie können den KaffeeKlatsch via E-Mail an abo@bookware.de oder über das Internet unter www.bookware.de/abo bestellen. Selbstverständlich können Sie das Abo jederzeit und ohne Angabe von Gründen sowohl via E-Mail als auch übers Internet kündigen.

Ältere Versionen können einfach über das Internet als Download unter www.bookware.de/archiv bezogen werden.

Auf Wunsch schicken wir Ihnen auch ein gedrucktes Exemplar. Da es sich dabei um einzelne Exemplare handelt, erkundigen Sie sich bitte wegen der Preise und Versandkosten bei Alexandra Specht via E-Mail unter alexandra.specht@bookware.de oder telefonisch unter 09131/8903-14.

Copyright

Impressum

KaffeeKlatsch Jahrgang 5, Nummer 7, Juli 2012
ISSN 1865-682X
Bookware – eine Initiative der MATHEMA Software GmbH

Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0
Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de
Internet: www.bookware.de

Herausgeber/Redakteur: Michael Wiedeking
Anzeigen: Natalia Wilhelm
Grafik: Nicole Delong-Buchanan

Inhalt

Artikel

Kolumnen

Ein Fall für REST

REST hat schon den Gipfel der überzogenen Erwartungen in den Hype-Zyklus hinter sich. Weil der Gipfel aber noch nicht so fern zurückliegt, findet man immer noch widersprüchliche Informationen und nicht genügend Musterlösungen im Netz. Dieser Artikel befasst sich mit der Theorie von REST und demonstriert bestimmte *RESTful HTTP*-Lösungsmuster an einem Beispiel aus der Praxis.

Der Super-Duper-Happy-Pfad

Eine Web-Applikation zu schreiben sollte purer Spaß sein. Ein neues Projekt aufzubauen sollte reibungslos funktionieren ohne Gänsehaut beim Gedanken, verschiedene Frameworks und Tools zusammen zu führen. Die *.NET*-Welt außerhalb von Microsoft wuchert nicht so wild wie bei anderen Technologien, aber im Open-Source-Universum erscheinen doch ab und zu beeindruckende Projekte: *Nancy* und *Simple.Data* sind zwei davon. Werden diese zusammen und in Kombination mit *MongoDB* genutzt, entsteht bald der „Super-Duper-Happy-Pfad".

Hüpfprävention

Bedingte Sprünge scheinen immer dann unangenehme Auswirkungen auf die Performanz zu haben, wenn die Bedingung nicht vorhersehbar ist. Prozessorhersteller sind deswegen bemüht, die Entwickler – wenn es denn wirklich darauf ankommt – darauf hinzuweisen, wie man Sprünge vermeiden kann.

Ein Fall für REST

Einführung ins RESTful HTTP mit Fallstudie

von Róbert Bräutigam

REST hat schon den Gipfel der überzogenen Erwartungen in den Hype-Zyklus[1] hinter sich. Weil der Gipfel aber noch nicht so fern zurückliegt, findet man immer noch widersprüchliche Informationen und nicht genügend Musterlösungen im Netz. Dieser Artikel befasst sich mit der Theorie von *REST* und demonstriert bestimmte *RESTful HTTP*-Lösungsmuster an einem Beispiel aus der Praxis.

Glossar

Die Wichtigkeit Namen und Wörter richtig zu benutzen wird meistens unterschätzt, und nicht nur wenn es um Variablen oder Methoden [1] sondern überraschenderweise auch wenn es um Architektur und Kommunikation von Anforderungen geht. Fangen wir deshalb mit dem Glossar an.

REST steht für „REpresentational State Transfer" und stammt aus der Dissertation von Roy T. Fielding aus dem Jahre 2000 [2]. Es ist ein „Architekturstil", den man benutzen kann um bestimmte architekturelle Anforderungen (funktionale und nicht-funktionale eingeschlossen) zu erfüllen. Es ist aber abstrakter als die Architektur selbst, da es meistens nur abstrakte Bausteine oder Randbedingungen definiert. Zum Beispiel sind die *Client-Server*- und *Pipe-and-Filter*-Muster Architekturstile. Architekturen, spezifische Implementationen oder sogar APIs sind daher nicht REST.

Architekturen, Implementationen oder APIs können aber *RESTful* sein. Das steht umgangssprachlich für etwas, dass den REST-Stil verfolgt und dessen Randbedingungen erfüllt.

RESTful HTTP oder *RESTful Web-Services* sind Synonyme und bezeichnen eine Lösung die RESTful ist und zusätzlich HTTP als einheitliche Schnittstelle benutzt (die allerdings nicht vorgeschrieben ist).

Was ist REST?

Es gibt sechs verschiedene Stile, die REST kombiniert. Sie alle definieren mehrere Randbedingungen, die zu erfüllen sind, um etwas RESTful zu nennen. Die folgende Liste ist eine kurze Zusammenfassung:

- *Client-Server*: Stellt sicher, dass bestimmte Belange getrennt sind und etabliert Anfrage-Antwort-Kommunikation. Es definiert allerdings nicht wo Zustände gehalten werden.
- *Stateless (zustandslos)*: In jeder Anfrage muss alles vorhanden sein, was der Server braucht um die Anfrage zu verstehen, ohne frühere Anfragen oder Antworten zu berücksichtigen. Diese Bedingung hilft z. B. bei der Skalierbarkeit.
- *Cache*: Jede Anfrage und Antwort muss *Cache*-Metadaten beinhalten, die es für andere Komponenten zwischen Client und Server möglich machen, Teile der Kommunikation potenziell zu *cachen*.
- *Einheitliche Schnittstelle*: Die wichtigsten Bedingungen sind hier die einheitliche Identifizierung von *Ressourcen*, der Umgang mit Ressourcen durch Repräsentationen, selbst-beschreibende Nachrichten und *Hypermedia* als Antrieb für Zustandsveränderung[2].
- *Mehrschichtiges System*: Es definiert weitere Zwischenkomponenten wie *Gateway* und *Proxy*, die in einer mehrschichtigen Hierarchie funktionieren können. Diese Komponenten sind für Clients unsichtbar oder sehen wie normale Server aus.
- *Code on demand (optional)*: Der Server kann bei Bedarf den Client mit weiteren Funktionalitäten erweitern.

Warum?

Die Randbedingungen, die REST definiert, sind nicht rein akademischer Natur. Der korrekte Einsatz ermöglicht es, bestimmte Qualitätsattribute wie Skalierbarkeit, inkrementelle und unabhängige Entwicklung von Komponenten, Beständigkeit u.s.w., die in der Dissertation von Fielding [2] im Detail beschrieben sind, zu erreichen.

[1] Neue Technologien werden manchmal überbewertet und gleichzeitig nicht immer richtig eingesetzt: http://de.wikipedia.org/wiki/Hype-Zyklus.

[2] Im Original: Hypermedia as the engine of application state. („HATEOAS")

Natürlich ist es möglich in einem Projekt bestimmte Bedingungen wegzulassen. Allerdings müssen wir in diesem Fall sicher sein, dass die Qualitätsattribute, die positiv oder negativ beeinträchtigt werden, mit unseren Zielen übereinstimmen, und wir dürfen die resultierende Architektur oder API nicht RESTful nennen, da es die nötigen Randbedingungen nicht mehr erfüllt und wir Missverständnisse vermeiden wollen.

Das Beispielproblem

Mit den folgenden vereinfachten, aber existierenden *SOAP Services* aus dem Bankwesen, werden wir demonstrieren, wie man eine gleichwertige RESTful HTTP-Schnittstelle definieren kann.

Anstatt *WSDLs* zu nutzen, werden die Services hier mit Beispielnachrichten (ohne SOAP-Umschlag) definiert, da sie sowieso voneinander völlig unabhängig sind und ihren eigenen *Namespace* haben.

Der Service *KundenSuche* kann Kunden anhand von Name oder Kontonummer suchen.

```
<request xmlns="http://bank.de/ws/kundensuche/v1_0">
  <accountnoOrUsername
    >111222333</accountnoOrUsername>
</request>
<response xmlns="http://bank.de/ws/kundensuche/v1_0">
  <customer>
    <userName>Udo Schmidt</userName>
    <customerNo>567567</customerNo>
    <account>
      <accountNo>111222333</accountNo>
      <accountType>Konto</accountType>
      <customerRole>Inhaber</customerRole>
    </account>
  </customer>
  <customer>
    <userName>Stefanie Schmidt</userName>
    <customerNo>567568</customerNo>
    <account>
      <accountNo>111222333</accountNo>
      <accountType>Konto</accountType>
      <customerRole>Berechtigte</customerRole>
    </account>
  </customer>
</response>
```

Der Service *KundeDetails* listet die Konten eines Kunden auf.

```
<request xmlns="http://bank.de/ws/kundendetails/v1_0">
  <customerNo>567567</customerNo>
</request>
<response xmlns="http://bank.de/ws/kundendetails/v1_0">
  <userName>Udo Schmidt</userName>
  <customerNo>567567</customerNo>
  <account>
    <accountNo>111222333</accountNo>
    <accountBalance>123.45</accountBalance>
    <blockingStatus>0</blockingStatus>
    <accountRoleId>Inhaber</accountRoleId>
    <accountType>Konto</accountType>
    <ownerName>Udo Schmidt</ownerName>
  </account>
  <account>
    <accountNo>111222334</accountNo>
    <accountBalance>234.50</accountBalance>
    <blockingStatus>0</blockingStatus>
    <accountRoleId>Inhaber</accountRoleId>
    <accountType>Tagesgeld</accountType>
    <ownerName>Udo Schmidt</ownerName>
  </account>
</response>
```

Der Service *ÜberweisungAnlegen* legt eine Überweisung an.

```
<request xmlns=
  "http://bank.de/ws/überweisunganlegen/v1_0">
  <customerNo>567567</customerNo>
  <tan>123456</tan>
  <accountNo>111222334</accountNo>
  <accountNoOpponent
    >22233344</accountNoOpponent>
  <bankCodeOpponent
    >71010010</backCodeOpponent>
  <accountingOpponent
    >Stephan Krueger</accountingOpponent>
  <amount>10.00</amount>
  <purpose>Wette verloren</purpose>
</request>
<response xmlns=
  "http://bank.de/ws/überweisunganlegen/v1_0">
  <transactionDate>2012-06-18</transactionDate>
</response>
```

Der Service *TerminÜberweisungAnlegen* ist fast derselbe wie *ÜberweisungAnlegen*, nur hat er zusätzlich noch ein *dateFirstExecution*-Feld und liefert eine ID, die wir später benutzen können um diesen Eintrag zu modifizieren oder zu löschen.

```
<request xmlns=
  "http://bank.de/ws/terminüberweisunganlegen/v1_0">
  <customerNo>567567</customerNo>
  <tan>123456</tan>
  <accountNo>111222334</accountNo>
  <accountNoOpponent
    >22233344</accountNoOpponent>
  <bankCodeOpponent
    >71010010</backCodeOpponent>
  <accountingOpponent
    >Stephan Krueger</accountingOpponent>
  <amount>10.00</amount>
  <purpose>Ich werde die Wette verlieren</purpose>
  <dateFirstExecution>2012-07-01</dateFirstExecution>
</request>
```

```
<response xmlns=
  "http://bank.de/ws/terminüberweisunganlegen/v1_0">
  <transactionDate>2012-06-18</transactionDate>
  <jobnumber>12</jobnumber>
</response>
```

Der Service *TerminÜberweisungÄndern* ähnelt dem Service *Anlegen*, er benutzt nur das Feld *jobnumber* um zu spezifizieren, welcher Eintrag geändert werden muss.

```
<request xmlns=
  "http://bank.de/ws/terminüberweisungaendern/v1_0">
  <customerNo>567567</customerNo>
  <jobnumber>12</jobnumber>
  <tan>123456</tan>
  <accountNo>111222334</accountNo>
  <accountNoOpponent
    >22233344</accountNoOpponent>
  <bankCodeOpponent
    >71010010</backCodeOpponent>
  <accountingOpponent
    >Stephan Krueger</accountingOpponent>
  <amount>20.00</amount>
  <purpose>Ich werde die Wette verlieren</purpose>
  <dateFirstExecution>2012-07-01</dateFirstExecution>
</request>
<response xmlns=
  "http://bank.de/ws/terminüberweisungaendern/v1_0">
  <transactionDate>2012-06-18</transactionDate>
</response>
```

TerminÜberweisungLöschen:

```
<request xmlns=
  "http://bank.de/ws/terminüberweisungloeschen/v1_0">
  <customerNo>567567</customerNo>
  <jobnumber>12</jobnumber>
  <accountNo>111222334</accountNo>
  <tan>123456</tan>
</request>
<response xmlns=
  "http://bank.de/ws/terminüberweisungloeschen/v1_0">
  <transactionDate>2012-06-18</transactionDate>
</response>
```

TerminÜberweisungListen listet alle Einträge zu einem Konto.

```
<request xmlns=
  "http://bank.de/ws/terminüberweisunglisten/v1_0">
  <customerNo>567567</customerNo>
  <accountNo>111222334</accountNo>
</request>
<response xmlns=
  "http://bank.de/ws/terminüberweisunglisten/v1_0">
  <transfer>
    <jobnumber>12</jobnumber>
    <accountNoOpponent
      >22233344</accountNoOpponent>
    <bankCodeOpponent
      >71010010</backCodeOpponent>
    <accountingOpponent
      >Stephan Krueger</accountingOpponent>
    <amount>20.00</amount>
    <purpose>Ich werde die Wette verlieren</purpose>
    <dateFirstExecution
      >2012-07-01</dateFirstExecution>
  </transfer>
  <transfer>
    <jobnumber>14</jobnumber>
    <accountNoOpponent
      >999888777</accountNoOpponent>
    <bankCodeOpponent
      >71010010</backCodeOpponent>
    <accountingOpponent
      >Hans Gartner</accountingOpponent>
    <amount>100.00</amount>
    <purpose>Charter</purpose>
    <dateFirstExecution
      >2012-09-01</dateFirstExecution>
  </transfer>
</response>
```

Ressourcen

Da REST nicht Service-orientiert sondern Ressourcen-orientiert ist – was zuerst ungewöhnlich erscheinen mag –, besteht die erste Aufgabe darin, die Ressourcen zu definieren (Objekte die wir identifizieren oder referenzieren wollen).

- Jeder Kunde muss eine Ressource sein, da wir auf Kunden zugreifen wollen.
- Jede Terminüberweisung (für einen bestimmten Kunden und Konto) ist eine Ressource, da wir diese referenzieren und nutzen. Es ist wichtig zu erwähnen, dass die Ressource von einem Kunden und einem Konto abhängt; sie ist also spezifisch für nur einen Kunden und ein Konto.
- Alle Terminüberweisungen (für einen bestimmten Kunden und Konto) bilden auch eine Ressource, da wir alle Terminüberweisungen gleichzeitig referenzieren wollen.
- Alle Überweisungen (für einen bestimmten Kunden und Konto) bilden ebenfalls eine Ressource.

Das Definieren von Ressourcen erfordert eine andere Denkweise als bei SOA. Wir definieren nicht Services sondern Ressourcen. Eine Ressource kann alles sein, von statischen Seiten (z. B. Web-Seiten) über dynamische und subjektive Listen (z. B. Meine Überweisungen), bis zu zeitabhängigen Informationen (z. B. aktuelle Zeit).

Hier muss man aufpassen, die URLs auf keinen Fall festzulegen. Es gibt APIs, die die URLs für die Ressourcen gleich am Anfang definieren und erwarten, dass die Clients dies irgendwoher wissen. Zum Beispiel wäre es ein Fehler zu sagen, dass alle Kunden mit der

URL */customer/{customerNo}* erreichbar sind, da diese Information eigentlich durch Links (HATEOAS) dynamisch kommuniziert werden müsste.

Mime-Typ

In REST ist das interne Server- und Schnittstellenmodell getrennt, also können Clients nur durch vorgegebene Repräsentationen mit Ressourcen interagieren. Diese Repräsentationen sind in HTTP mit *Mime-Typen* beschrieben, die wir als API-Designer verwenden oder definieren müssen.

Tatsächlich ist dies unser wichtigstes Werkzeug um die Schnittstelle zu definieren. Außer Mime-Typen dürfen wir den Clients eigentlich nur ein paar initiale URLs angeben, sonst nichts. Falls wir doch andere Informationsquellen benutzen (z. B. E-Mail-Vereinbarungen, statische URL-Vorlagen), riskieren wir implizites Wissen in unser System einzuführen, das bestimmte Qualitätsattribute wie Beständigkeit und Veränderbarkeit negativ beeinflusst.

Falls wir keinen standardisierten Mime-Typ benutzen können (wie in diesem Fall), müssen wir einen selbst definierten, mit dem dazu reservierten Präfix *vnd* (kurz für *Vendor*), nutzen, zum Beispiel *application/vnd.bank.accounts-v1+xml*. *Application* ist die oberste Kategorie für Anwendungen [3], und *vnd.bank* bedeutet, dass der Typ spezifisch für die Firma namens „Bank“ ist. Das *+xml-Postfix* ist auch standardisiert [4] und bedeutet, dass XML-Nachrichten verwendet werden. Man könnte hier auch mehrere Mime-Typen für die verschiedenen Ressourcen definieren. Was den Vorteil hätte, dass man die Ressourcen unabhängig voneinander weiterentwickeln könnte, aber es hätte auch den Nachteil, dass Clients dann mehrere Mime-Typen verfolgen müssen.

Jedes Mal wenn ein Client den Server aufruft, liefert er alle Mime-Typ-Namen, die er als Antwort verstehen würde und wählt den bestmöglichen Typ aus. Dieser Mechanismus nennt sich Inhalts-Aushandlung und kann auch für Versionierung verwendet werden. Zum Beispiel kann ein Client *application/vnd.bank.accounts-v1+xml* anfordern, wenn er nur Version 1 von den *accounts* Mime-Typ versteht, oder er kann gleichzeitig auch *application/vnd.bank.accounts-v2+xml* anfordern, falls er es bearbeiten kann. Damit kann man leicht rückwärtskompatible Client- und Server-Applikationen implementieren.

Repräsentation

Obwohl wir hier nur einen Mime-Typ definieren, kann dieser auch mehrere Nachrichtenformate beinhalten. Wir müssen nur aufpassen, dass alle Nachrichten „selbstbeschreibend“ sind und wir nicht implizites Wissen benutzen, wie zum Beispiel von welcher URL die Nachricht kommt.

Der Einfachheit halber definieren wir hier die Nachrichten nur durch Beispiele ohne die exakten *XSDs* zu listen. Der Namensraum kann zum Beispiel *http://bank.de/schema/account/v1* sein, wobei die Versionierung die Versionsnummer in dem *Media-Typ* verfolgen muss.

Die erste Repräsentation definieren wir für die Startressource *Kundensuche*.

```
<customers xmlns=
  “http://bank.de/schema/account/v1“
    xmlns:xlink=
      "http://www.w3.org/1999/xlink">
  <customer xlink:href=
    “http://bank.de/customer/567567“>
    <userName>Udo Schmidt</userName>
    <customerNo>567567</customerNo>
    <account>
      <accountNo>111222333</accountNo>
      <accountType>Konto</accountType>
      <customerRole>Inhaber</customerRole>
    </account>
  </customer>
  <customer xlink:href=
    “http://bank.de/customer/567568“>
    <userName>Stefanie Schmidt</userName>
    <customerNo>567568</customerNo>
    <account>
      <accountNo>111222333</accountNo>
      <accountType>Konto</accountType>
      <customerRole>Berechtigte</customerRole>
    </account>
  </customer>
  <search xlink:href=“http://bank.de/customersearch“/>
</customers>
```

Alle Clients müssen von dieser Ressource aus ihren *Workflow* starten und dafür müssen sie auch deren URL kennen, z. B. *http://bank.de/customersearch*. Weitere URLs müssen in den Nachrichten geliefert werden, was im diesen Fall mit *XLink*-Attributen gemacht wird. XLink ist eine standardisierte Methode [5], um von XML auf andere Ressourcen zu verlinken, im einfachsten Fall mit einem *href*-Attribut, wie wir es von HTML kennen. In dieser Nachricht wird es an die zwei Detailansichten von Kunden gelinkt, aber auch der Suchlink ist am Ende erwähnt. Da wir explizit definieren wollen wie man suchen kann, müssen wir das innerhalb der Definition des Mime-Typs machen und damit zu einem Teil des Nachrichtenformats.

Wir definieren deshalb das *search*-Element so, dass die verlinkte Ressource mit dem *accountnoOrUsername*-Parameter aufgerufen werden muss, also z. B. so:

```
http://bank.de/
customersearch?accountnoOrUsername=111222333.
```

Aber Achtung: damit haben wir eine neue Ressource definiert.

Warum definieren wir diese Anfrage nicht wie bei dem Originalbeispiel als XML-Nachricht, die wir zu der angegebenen URL *POST*en? Weil *GET* mit Parametern viel mehr Garantien und Funktionalitäten liefert: Clients und Zwischenkomponenten wie *Cache* oder *Proxys* wissen, dass *GET* eine „sichere" Methode ist. Mit anderen Worten: es verändert nichts (man kann es mehrmals ohne Nebeneffekte aufrufen), und man kann die Antwort auch *cachen*, falls die Metadaten richtig gesetzt sind.

Um die Detailansicht für einen Kunden aufzurufen, kann der Client die Links in dem *customer*-Element verfolgen. Die Antwort könnte so aussehen:

```
<customer xmlns="http://bank.de/schema/account/v1"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <userName>Udo Schmidt</userName>
  <customerNo>567567</customerNo>
  <account>
    <accountNo>111222333</accountNo>
    <accountBalance>123.45</accountBalance>
    <blockingStatus>0</blockingStatus>
    <accountRoleId>Inhaber</accountRoleId>
    <accountType>Konto</accountType>
    <ownerName>Udo Schmidt</ownerName>
    <transfer xlink:href=
      "http://bank.de/cashtransfer/567567/111222333"/>
    <appointedTransfer xlink:href=
      "http://bank.de/appointedtransfer/567567/111222333"/>
  </account>
  <account>
    <accountNo>111222334</accountNo>
    <accountBalance>234.50</accountBalance>
    <blockingStatus>0</blockingStatus>
    <accountRoleId>Inhaber</accountRoleId>
    <accountType>Tagesgeld</accountType>
    <ownerName>Udo Schmidt</ownerName>
    <transfer xlink:href=
      "http://bank.de/cashtransfer/567567/111222334"/>
    <appointedTransfer xlink:href=
      "http://bank.de/appointedtransfer/567567/111222334"/>
  </account>
</response>
```

Diese Antwort benutzt dasselbe Element *customer* wie *customers*, nur liefert es mehr Informationen. Die Links die hier definiert sind, kann der Client benutzen, um die restliche (kein Wortspiel beabsichtigt!) Funktionalität (eine *ÜberweisungAnlegen* und vier *Terminüberweisung-Funktionen*) aufzurufen.

Wir müssen nur in unserem Media-Typ definieren, dass die Elemente *transfer* und *appointedTransfer* zu „alle Überweisungen" und „alle Terminüberweisungen" Ressourcen verlinken, die spezifisch für diesen Kunden und dieses Konto sind. Wie diese angelegt, verändert, gelöscht oder gelistet werden, definiert netterweise HTTP für uns. *GET* auf die „alle Terminüberweisungen"-Ressource sollte eine Liste von Terminüberweisungsressourcen ausgeben (mit URLs), *POST* sollte eine Neue anlegen, *PUT* auf eine Terminüberweisungsressource sollte sie ersetzen und *DELETE* löschen [6]. In der Theorie könnte die „alle Überweisungen"-Ressource auch Listen unterstützen, aber das ist nicht nötig. Es ist auch nicht nötig dies zu spezifizieren, weil das dynamisch vom Client mit der *OPTIONS*-Methode abgefragt werden kann [6].

Für den *transfer*-Link unterstützt unser Server also nur *POST* mit der folgenden Nachricht.

```
<cashtransfer xmlns=
  "http://bank.de/schema/account/v1">
  <tan>123456</tan>
  <accountNoOpponent
    >22233344</accountNoOpponent>
  <bankCodeOpponent
    >71010010</backCodeOpponent>
  <accountingOpponent
    >Stephan Krueger</accountingOpponent>
  <amount>10.00</amount>
  <purpose>Wette verloren</purpose>
</castransfer>
```

Da wir unsere Ressource so definiert haben, dass sie von einem Kunden und einem Konto abhängig ist, können wir diese Information jetzt einfach weglassen. Achtung: Die Nachricht ist damit noch immer selbstbeschreibend für die angegebene Ressource.

Wir brauchen auch das Antwortfeld *transactionDate* nicht, da es in HTTP ein eingebautes Antwortfeld *Date* gibt, das den selben Zweck erfüllt: die Serverzeit anzuzeigen. Ein Aufruf zwischen Client und Server würde also wie folgt ablaufen.

```
POST /account/567567 /111222333 /cashtransfer HTTP/1.1
Host: bank.de
User-Agent: Java/1.7
Content-Length: 234
Content-Type: application/vnd.bank.accounts-v1+xml

<cashtransfer xmlns="http://bank.de/schema/account/v1">
...
</castransfer>

HTTP/1.1 204 No Content
Date: Fri, 22 Jun 2012 23:59:59 GMT
Content-Type:  application/vnd.bank.accounts-v1+xml
Content-Length: 0
```

Für die Terminüberweisungen ist das Anlegen sehr ähnlich, mit dieser Repräsentation:

```
<appointedcashtransfer xmlns=
  "http://bank.de/schema/account/v1">
  <tan>123456</tan>
  <accountNoOpponent
    >22233344</accountNoOpponent>
  <bankCodeOpponent
    >71010010</backCodeOpponent>
  <accountingOpponent
    >Stephan Krueger</accountingOpponent>
  <amount>10.00</amount>
  <purpose>Ich werde die Wette verlieren</purpose>
  <dateFirstExecution>2012-07-01</dateFirstExecution>
</appointedcastransfer>
```

Hier brauchen wir aber irgendetwas, damit wir den kreierten Eintrag später identifizieren können (zum Ändern oder Löschen). Glücklicherweise haben wir Ressourcen für Terminüberweisungen definiert, also wird, falls dieser Aufruf erfolgreich ist, eine neue Ressource mit einer URL angelegt, die wir zum Referenzieren verwenden können. HTTP definiert das *Location*-Feld für genau diesen Zweck. Die Antwort vom Server sieht im diesen Fall also so aus:

```
HTTP/1.1 201 Created
Date: Fri, 22 Jun 2012 23:59:59 GMT
Content-Type:  application/vnd.bank.accounts-v1+xml
Content-Length: 0
Location:
  http://bank.de/appointedtransfer/567567 /111222334 /12
```

Die Repräsentation „alle Terminüberweisungen“ kann das vorher definierte *appointedcashtransfer*-Element wiederverwenden.

```
<appointedtransfers xmlns="http://bank.de/schema/
  account/v1" xmlns:xlink="http://www.w3.org/1999/xlink">
  <appointedcashtransfer xlink:href=
    "http://bank.de/appointedtransfer/567567 /111222334 /12">
    <accountNoOpponent
      >22233344</accountNoOpponent>
    <bankCodeOpponent
      >71010010</backCodeOpponent>
    <accountingOpponent
      >Stephan Krueger</accountingOpponent>
    <amount>20.00</amount>
    <purpose>Ich werde die Wette verlieren</purpose>
    <dateFirstExecution
      >2012-07-01</dateFirstExecution>
  </appointedcashtransfer>
  <appointedcashtransfer xlink:href=
    "http://bank.de/appointedtransfer/567567 /111222334 /14">
    <accountNoOpponent
      >999888777</accountNoOpponent>
    <bankCodeOpponent
      >71010010</backCodeOpponent>
    <accountingOpponent
      >Hans Gartner</accountingOpponent>
    <amount>100.00</amount>
    <purpose>Charter</purpose>
    <dateFirstExecution
      >2012-09-01</dateFirstExecution>
  </appointedcashtransfer>
</appointedtransfers>
```

Die Links in dieser Nachricht sind die selben wie bei dem Anlegen, und weil die Repräsentation für eine Terminüberweisung schon festgelegt ist, müssen wir nicht weiter spezifizieren, dass man eine Änderung mit *PUT* erreichen oder mit *DELETE* Terminüberweisungen löschen kann.

Zusammenfassung

Mit einer SOAP-Beispielapplikation haben wir demonstriert, wie man eine gleichwertige RESTful-Schnittstelle spezifizieren kann.

Im ersten Schritt sind die Ressourcen festzulegen, die auf dem Server aufzufinden sind. Hier muss man darauf achten, dass nicht die URLs sondern die Semantik zu definieren ist.

Der zweite Schritt ist die Repräsentation(en) für die oben erstellten Ressourcen mit Media-Typen zu bestimmen. Besonders wichtig ist es implizites Wissen zu vermeiden und selbstbeschreibende Nachrichten mit Links für Zustandsveränderungen zu benutzen.

Implementation

Wie man für die oben erstellte Spezifikation eine Server- oder Client-seitige Applikation baut, z.B. mit *Jersey* oder *Spring*, ist an dieser Stelle dem geschätzten Leser überlassen. Für die meisten großen Frameworks existieren gute Einführungen und Tutorien im Netz.

Hier möchten wir nur auf ein paar nicht-technische Fehlerquellen hinweisen:

- Media-Typ muss immer von Client und Server spezifiziert werden (*Accept-* und *Content-Type-Header* Felder). Wenn kein oder ein generischer Media-Typ (z.B. *text/html*) angegeben ist, sollte der Server annehmen, dass er eine menschenlesbare Repräsentation wählen soll.
- Der Server sollte keine Session beibehalten. Für Funktionen, die so etwas normalerweise benötigen (wie Authentifizierung), sollte höchstens *Cache* verwendet werden, aber der Client sollte trotzdem alle Informationen in jedem Aufruf mitschicken.
- Der Server sollte explizit die *Caching*-Metadaten pro Ressource steuern. Hierfür gibt es viele Möglichkeiten und man sollte die *Caching*-Strategie sorgfältig planen (und nicht unterschätzen).
- Der Client sollte niemals mit festen URLs arbeiten, außer ein paar *Bookmark-URL*s, und auch die sollte er ändern, wenn er die richtigen *Response-Codes* erhält (z. B. 301 – *Moved Permanently*). Allerdings kann er

URLs für Ressourcen, die er schon gesehen hat, als Bookmarks speichern, da sich URLs für Ressourcen kaum (und nur kontrolliert mit Umleitungen) ändern.

- Der Client sollte niemals annehmen, dass bei einem URL-Aufruf ein bestimmtes XML-Format zurückkommt. Das Format hängt von dem Media-Typ ab, und dieser wird wiederum dynamisch ausgehandelt.

Fazit

Es ist nicht schwierig REST richtig einzusetzen, es benötigt aber ein Umdenken von konkurrierenden Technologien und Stilen wie *RPC* und *SOAP*.

Wir haben demonstriert, dass eine RESTful HTTP-Schnittstelle um vieles einfacher zu verstehen und zu implementieren sein kann, und man implizites Wissen leicht vermeiden kann, was die Verständlichkeit und auch Beständigkeit erhöht.

Wenn Sie auch an einer RESTful HTTP-Schnittstelle arbeiten und ein interessantes Problem (ob klein oder groß) mit uns teilen wollen, zögern Sie nicht, uns an redaktion@bookware.de zu schreiben.

Referenzen

[1] Martin, Robert C. *Clean Code: a handbook of agile software craftmanship*, May 2011, Chapter 2

[2] Fielding, Roy T. *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation University of California, Irvine, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[3] IETF *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, RFC 2046, http://www.ietf.org/rfc/rfc2046.txt

[4] IETF *XML Media Types*, RFC 3023, http://www.ietf.org/rfc/rfc3023.txt

[5] W3C *XML Linking Language (XLink) Version 1.0*, 27 June 2001, http://www.w3.org/TR/2001/REC-xlink-20010627

[6] IETF *Hypertext Transfer Protocol -- HTTP/1.1*, RFC 2616, http://www.ietf.org/rfc/rfc2616.txt

Weiterführende Literatur

- Fielding, Roy T. *Untangled Blog* http://roy.gbiv.com/untangled
- Java The Java EE 6 Tutorial http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html
- Spring 3.0.0M1 REST Dokumentation http://static.springsource.org/spring/docs/3.0.0.M3/spring-framework-reference/html/ch18s02.html

Kurzbiographie

Robert Bräutigam ist als Senior-Consultant für die MATHEMA Software GmbH tätig. Er ist seit 1999 als Entwickler und Architekt im Java Enterprise-Umfeld beschäftigt und interessiert sich für leichtgewichtige Lösungen in Technologien sowie im Projektmanagement.

Der Super-Duper-Happy-Pfad

2 1/2 Blicke auf großartige Open-Source-Frameworks für .NET

von Timothée Bourguignon

Eine Web-Applikation zu schreiben sollte purer Spaß sein. Ein neues Projekt aufzubauen sollte reibungslos funktionieren ohne Gänsehaut beim Gedanken, verschiedene Frameworks und Tools zusammen zu führen. Die *.NET*-Welt außerhalb von Microsoft wuchert nicht so wild wie bei anderen Technologien, aber im Open-Source-Universum erscheinen doch ab und zu beeindruckende Projekte: *Nancy*[1] und *Simple.Data*[2] sind zwei davon. Werden diese zusammen und in Kombination mit *MongoDB*[3] genutzt, entsteht bald der „Super-Duper-Happy-Pfad".

Erstes Date mit Nancy

Nancy gehört zu der Familie der *Micro-Frameworks* und ist inspiriert vom *Ruby*-Framework *Sinatra* [4]. Nancy ist ein leichtgewichtiges Framework mit dem man Web-Seiten entwickeln kann. Und nur das! Anstatt gebündelt zusammen mit zig Tools zu kommen – wie *ASP.NET MVC* oder *Ruby on Rails* – bringt Nancy nur genügend Features mit um sich selber aufzubauen, die Möglichkeit auf *HTTP-Requests* zu reagieren und Daten zu und von *Views* zu bewegen.

Simple.Data ist ein ganz kleiner *Object Relational Mapper* (ORM), der keine Objekte benutzt, auch mit nicht-relationalen Datenbanken funktioniert und kein Mapping braucht. Also ein ORM ohne O und nicht unbedingt R oder M. Er wurde geschrieben um *SQL-Injections* zu vermeiden. Er wurde von *ActiveRecord* und *DataMapper* aus Ruby on Rails inspiriert und basiert auf dem „dynamic" Schlüsselwort unter dem .NET-Framework 4.

MongoDB ist eine dokumentenorientierte *NoSQL*-Datenbank. Anstatt Tabellen zu verwenden, benutzt MongoDB *JSON*-artige Dokumente mit dynamischem Schema, dem *BSON*. Dies hat viele Vorteile, insbesondere ein sehr einfaches Speichern von komplexen Objekten, die in einer relationalen Datenbank mehrere Tabellen brauchen würden und bietet dadurch, da sie Schemalos sind, eine riesige Flexibilität.

Alle drei Tools sind Open-Source und werden durch aktive Communities unterstützt.

Liebe auf den ersten Blick

Nancy wurde auf dem *.NET Client Profile* gebaut. Es braucht also sehr wenig zum Laufen. Es funktioniert auch unter *Mono* [5] und kann auf verschiedenen Web-Servern laufen, inkl. *IIS* und *OWIN* [6], kann sich aber auch selber „hosten".

Um ein Nancy-Projekt anzufangen, baut man am besten ein leeres *ASP.NET*-Projekt in Visual Studio und installiert das *Nancy.Hosting.Aspnet Nuget* [7] Package. Damit referenziert Nancy zwei *DLLs* und fügt ihre *HTTP-Handler* an die *web.config*-Datei hinzu. Das war's auch schon!

Nancys Herz

Nancys zentraler Baustein ist das *Modul*. Es ist ein bisschen wie ein ASP.NET MVC-Kontroller und ist das einzige Teil, um das man nicht herum kommt. Module sind die Stellen, an denen man die unterstützten *Routen* und das Verhalten der Applikation definiert.

Ein Modul muss von der *NancyModule* Basisklasse erben. Neben der Definition von Verhalten, bringen Module Informationen über aktuelle Anfragen, Kontext der Anfragen, *Response Helpers* und viel mehr mit. Endlich werden damit alle Module dynamisch „gefunden"; eine Registrierung ist nicht nötig.

Routen werden im Konstruktor eines Moduls definiert und mit *Lambda-Expressions* vom Typ *Func<dynamic, Response>* geschrieben. Eingabe ist ein dynamisches *Nancy-Dictionary* und Antwort ein *Nancy-Response*-Objekt. Das eingegebene Dictionary enthält alle zusätzlichen Informationen über die Routen, wie zum Beispiel Parameter.

Aber genug, hier nun Nancys „Hallo Welt". Wir schreiben unsere Module in eine neue *C#*-Klasse:

```
using Nancy;
public class GraffitiModule : NancyModule {
  public GraffitiModule() {
    Get["/"] = _ => {
      return "Hello World";
    };
  }
}
```

Dieses Stück Code erklärt Nancy, dass die Antwort zu einem *HTTP Get Request* auf den *Root*-Pfad *"/"* der Text „Hello World" ist. Einfacher geht's nicht.

Nancy pudert sich die Nase

Einen "Hello World"-String zu liefern ist schön, aber manchmal will man „komplexere" Daten darstellen; dafür braucht man einen View. Nancy kommt mit einer *Super-Simple-View-Engine*, die ähnlich wie *Razor*, die *Standard-ASP.NET-MVC3-View-Engine* von Microsoft, funktioniert.

Für das jetzige Beispiel legt man einfach eine *index.cshtml*-Datei mit dem folgenden Inhalt an.

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body>
<h1>Graffiti</h1>
</body>
</html>
```

Zurück im Modul, halten sich die Änderungen in Grenzen:

```
public GraffitiModule() {
  Get["/"] = _ => {
    return View["index"];
  };
}
```

Nancy, Zeit zum Ausgehen

Weiß man, wie Nancy grob funktioniert, ist es höchste Zeit Simple.Data und MongoDB zu erkunden und unsere Applikation mit einer echten Datenbank zu verbinden. Aber erstmal müssen wir ein paar zusätzliche *Nuget Packages* installieren:

```
Install-Package Simple.Data.MongoDB
Install-Package Nancy.Viewengines.Razor
```

Hinweis: das *Razor Package* braucht nur, wer Razor verwenden will. Man könnte mit der Super-Simple-View-Engine weiterarbeiten, oder auch mit *Spark* [8], *NDjango* [9] oder *dotLiquid* [10].

Es sei noch darauf hingewiesen, dass das Simple.Data-Package hier mit dem MongoDB-Adapter installiert wird. Es könnte auch separat via *Install-Package Simple.Data.Core* installiert werden.

Falls ein anderer Datenbank-Typ verwendet werden soll, bedarf es natürlich eines anderen Adapters.

Zurück im Modul, werden ein paar *private* Felder angelegt, um die Datenbankverbindung aufzubauen:

```
usingSimple.Data;
usingSimple.Data.MongoDB;

public class GraffitiModule : NancyModule {
  private const string ConnectionString =
  @"mongodb://localhost:27017/graffiti";
  private readonly dynamic db =
  Database.Opener.OpenMongo(ConnectionString);
  public GraffitiModule() {
    [...]
  }
}
```

Die „Graffiti"-Datenbank existiert noch nicht. Sie wird aber automatisch angelegt, sobald ein Objekt geschrieben wird. Das war's! Simple.Data ist nun bereit mit einer Datenbank zu sprechen.

Wenn noch nicht vorhanden, muss noch das letzte MongoDB-Package von der offiziellen Web-Seite geladen und installiert werden. Danach kann der Datenbank-*Service* in einem *Command*-Fenster gestartet werden:

```
mongod.exe --dbpath <path to the db>
(z. B. mongod.exe --dbpath C:\data\db)
```

Und in einem anderen *Command*-Fenster startet man die *MongoDB Interactive Shell*:

```
mongo.exe localhost/<dbname>
(z. B. mongo.exe localhost/graffiti)
```

Bedarf es einiger Testdaten, so fügt man ein „Graffiti"-Objekt mit einer einfachen *Text-Property* hinzu. In der MongoDB Interactive Shell tippt man dazu Folgendes.

```
db.Graffiti.save({Text: "Comin' from the DB"});
```

Um zu prüfen ob die Daten richtig gespeichert wurden, kann man die Datenbank *Collection* Statistik abfragen:

```
db.printCollectionStats()
```

Und falls alles gut ging, bekommt man folgendes Ergebnis.

```
system.indexes
{
"ns" : "graffiti.system.indexes",
"count" : 1,
"size" : 72,
"avgObjSize" : 72,
"storageSize" : 4096,
"numExtents" : 1,
"nindexes" : 0,
"lastExtentSize" : 4096,
"paddingFactor" : 1,
"flags" : 0,
"totalIndexSize" : 0,
"indexSizes" : {
},
"ok" : 1
}
```

Dabei ist die Zeile *"count" : 1* am wichtigsten. Alles hat geklappt!

Am Rande sei erwähnt, dass die Hilfe der MongoDB Interactive Shell via *.help*, wie z. B. *db.help* oder *db.graffiti.help*, aufgerufen werden kann.

Es wird ernst

Jetzt, wo das *Setup* vollständig ist, ist man in der Lage, die Daten der Datenbank anzuzeigen. Dafür ändert man die *Get["/"]*-Expression, um die Daten zu holen und dem View ein Model zu reichen.

```
Get["/"] = _ => {
  var scrawls = db.Graffiti.All();
  return View["index", scrawls];
};
```

Ein Simple.Data-Aufruf funktioniert nach folgendem Muster: *Datenbank.Objekt.Methode().Optionen()*. In diesen Fall wird die Datenbank nach allen "Graffiti"-Objekten gefragt, ohne spezifische zusätzliche Informationen (z. B. *OrderBy*). Diese Information wird zunächst an den View gereicht.

Im *cshtml*-View wird der *Body* geändert, um über die Objekt-Kollektion zu iterieren:

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body>
<h1>Graffiti</h1>
@foreach(var scrawl in @Model)
{
<p>@scrawl.Text</p>
}
</body>
</html>
```

Damit werden die „Graffiti"-Objekte angezeigt.

Obwohl hier Razor-Syntax verwendet wird, ist die Model-Definition einfacher geworden; leider verliert man damit aber die *Intellisense-Auto-Complete*-Hilfe in den Views.

Gemeinsame Zukunft?

Um Daten hinzuzufügen braucht man noch ein Formular:

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body>
<h1>Graffiti</h1>
<form method="POST">
Write something: <input type="text" name="Text"/>
<input type="submit" value="Scrawl!" />
</form>
@foreach(var scrawl in @Model)
{
<p>@scrawl.Text</p>
}
</body>
</html>
```

Dies ist ein Standard-HTML-Formular mit einem *Submit*-Knopf, der eine *POST*-Aktion auslösen wird.

Zurück in unserem Modul müssen wir uns um diese *POST*-Aktion auf *"/"* kümmern:

```
usingNancy.Responses;

Post["/"] = _ =>
{
db.Graffiti.Insert(Text: Request.Form.Text.Value);
return new RedirectResponse("/");
};
```

Dazu sind noch vier Bemerkungen zu machen!

- Das Einfügen von Daten via Simple.Data ist sehr einfach und folgt dem Muster *Datenbank.Objekt.Insert()*.
- Nach Ausführung der *HTTP-POST*-Aktion wird der Benutzer weiter zur *GET["/"]*-Aktion via einem neuen *RedirectResponse("/")*-Objekt geleitet.
- Das Präfix *Text*: wird verwendet, um Simple.Data zu sagen, dass ein *on the fly*-Objekt benutzt werden soll. Ansonsten müsste ein "myGraffiti"-Objekt mit einer Text-Property definiert, die Property ausgefüllt und *db.Graffiti.Insert(myGraffiti)* aufgerufen werden.
- Durch *Request.Form.Text.Value* wird das View-Model repräsentiert.

Damit ist die Applikation vollständig und wir können Text hinzufügen und wieder lesen.

Der Abschiedskuss

Es wurde gezeigt, wie man Nancy, Simple.Data und MongoDB kombinieren, konfigurieren und miteinander verbinden kann. Dabei konnte man sehen, wie Nancy auf eine bestimmte Route reagiert, einen View anzeigt, View-Daten von und zum View schickt. Ebenso wurde gezeigt, wie man Simple.Data verwendet, um Daten zu schreiben bzw. von einer Datenbank zu lesen. Dazu wurde MongoDB gestartet und demonstriert, wie über die *Interactive Shell* Objekte hinzugefügt und gelesen werden können. Das Ganze innerhalb zehn *C#*-Linien und kaum mehr *cshtml*-Linien. Das ist ein erster Blick in den Super-Duper-Happy-Pfad.

Natürlich gibt es viel mehr zu erfahren, aber dafür braucht es einen weiteren Artikel.

Referenzen

[1] NancyFx *Leightweight Web Framework for .NET*, http://nancyfx.org

[2] github *Simple.Data*, https://github.com/markrendle/Simple.Data/wiki

[3] MongoDB *Agile and Scalable* http://www.mongodb.org

[4] Sinatra http://www.sinatrarb.com

[5] Mono http://www.mono-project.com

[6] OWIN *Open Web Interface for .NET* http://owin.org

[7] Nuget gallary *home* http://nuget.org

[8] Spark View Engine *Html friendly. Less is more.* http://sparkviewengine.com

[9] Ndjango http://ndjango.org

[10] DotLiquid *A safe templating system for .NET* http://dotliquidmarkup.org

Weiterführende Literatur

- Håkansson, Andreas *Persönliches Blog* http://thecodejunkie.com/
- Rendle, Mark *Persönliches Blog* http://blog.markrendle.net
- Rendle, Mark *Introduction to Nancy and Simple.Data* http://skillsmatter.com/podcast/open-source-dot-net/introduction-to-nancy-and-simple-data

Kurzbiographie

Timothée Bourguignon ist als Senior Developer für die MATHEMA Software GmbH tätig. Sein Spezialgebiet ist Desktop- und Web-Programmierung mit dem .NET-Framework. Hierbei setzt er auf die Philosophie des Software Craftsmanships. Daneben beschäftigt er sich mit den Neuerungen der .NET-Welt und deren Communities. Als Certified Scrum Master liegt ein weiterer Schwerpunkt auf agilen Methoden.

Des Programmierers kleine Vergnügen

Hüpfprävention

von Michael Wiedeking

Bedingte Sprünge scheinen immer dann unangenehme Auswirkungen auf die Performanz zu haben, wenn die Bedingung nicht vorhersehbar ist. Prozessorhersteller sind deswegen bemüht, die Entwickler – wenn es denn wirklich darauf ankommt – darauf hinzuweisen, wie man Sprünge vermeiden kann.

Liest man beispielsweise die Optimierungsempfehlungen von Intel für die 64- und 32-Bit-Architekturen [1], erfährt man tatsächlich, dass bedingte Sprünge wann immer möglich zu vermeiden sind. Und das lässt sich leicht an einem einfachen Beispiel demonstrieren.

Stellen sie sich vor, Sie müssen in Abhängigkeit einer nicht vorhersehbaren Bedingung *b()* (wie beispielsweise $a < b$) eine Variable v mit dem konstanten Wert X oder dem Wert Y initialisieren.

```
if (b()) {
   v = X;
} else {
   v = Y;
}
…
```

Das lässt sich in den *C*-orientierten Sprachen auch etwas kompakter schreiben:

```
v = (b()) ? X : Y
```

So naheliegend diese Implementierung auch sein mag, so problematisch ist sie auch. Die Schwierigkeit liegt hier nämlich darin, dass diese Implementierung immer einen Sprung erforderlich macht: Ist die Bedingung nicht erfüllt, so muss zwar initial nicht gesprungen werden, dafür muss aber der *else*-Fall übersprungen werden.

In einer geeigneten *Assembler*-Sprache könnte das wie folgt umgesetzt werden:

```
   r ← b()
   jumpIfFalse(r, L1)
   v ← X
   goto L2
L1:
   v ← Y
L2:
   …
```

Die Frage, die sich stellt, ist also, inwiefern die Sprünge vermieden werden können. Unabhängig von der Wahl der konkreten Operation, die angewendet werden soll (und ohne wieder Gefahr zu laufen, negativen Einfluss auf die Performanz zu haben), lässt sich die grundsätzliche Idee wie folgt skizzieren:

```
r ← b()          // 0 oder 1
r ← −r           // 0…0 oder 1…1
r ← r & (X − Y)  // 0 oder (X − Y)
r ← r + Y        // X oder Y
```

Durch die Negation des Ergebnisses aus der Bedingung (wobei hier angenommen wird, dass *true* dem Wert 1 und *false* dem Wert 0 entspricht), erhält man eine Maske, die entweder im Falle von *false* vollständig aus Nullen oder im Falle von *true* aus Einsen besteht. Verknüpft man nun diese Maske durch ein Bit-weises Und mit $(X - Y)$, das sich ja schon zur Übersetzungszeit übersetzen lässt, so erhält man entweder den Wert 0 oder eben diese Differenz. Addiert man nun Y hinzu, so ist das Ergebnis entweder Y oder $(X - Y + Y)$, also X.

Ein Prozessor kümmert sich aber noch um unglaublich viel mehr. Deshalb muss man z. B. innerhalb einer Schleife zusätzlich versuchen, die verbleibenden Sprünge so zu reduzieren, dass sie eine gewisse Anzahl (beispielsweise 16) nicht überschreiten, damit die Sprunghistorie gespeichert und so innerhalb der Schleife für eine Vorhersage genutzt werden kann.

Auch wenn an dieser Stelle *eine* Idee zur Eliminierung eines bedingten Sprungs aufgezeigt werden konnte, erspart es einem im konkreten Fall nicht, die Eigenschaften der einzelnen Instruktionen sehr sorgfältig zu prüfen und zwischen deren Vorteilen und Nachteilen abzuwägen. Beispielsweise könnte der Performanzvorteil verschwinden, wenn X und Y nicht wie angenommen konstant sind.

Aber zum Glück muss man sich in der Regel um Optimierungen dieser Art nicht kümmern, weil es die meisten Compiler schon korrekt für einen machen.

Referenzen

[1] Intel *Intel 64 and IA-32 Architectures Optimization Reference Manual*, April 2012, http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html

Deutsch für Informatiker

Durchgebeugt

von Michael Wiedeking

Das Problem mit dem Duden ist, dass er einem zwar sagt, wie ein Wort grundsätzlich geschrieben wird, überlässt es dann aber der Phantasie, was man weiter damit macht. Zugegeben wird bei Substantiven noch der Genitiv und der Plural beschrieben, aber bei den Verben ist man doch mehr oder weniger auf sich selbst gestellt.

Das ist mir deshalb wieder einmal aufgefallen, weil ich im Editorial wegen des „Zitats“ das Verb *debuggen* beugen musste. Immerhin kennt der Duden *de·bug·gen*. Das Deutsche Universalwörterbuch sagt dazu: „[engl. to debug, zu: bug, Bug] (EDV): *ein Debugging vornehmen*“ [1]. Freundlicherweise wird auch beschrieben, was ein *Bug* ist: „[engl. bug = Fehler, Macke, eigtl. = Wanze; (lästiges) Insekt]: *Fehler in einem Computerprogramm*“ [1]. Das Fremdwörterbuch weiß das sogar noch zu präzisieren und definiert genauer: „einen Programmfehler in einem Softwareprogramm beseitigen“ [2].

Was mir der Duden nicht offenbart – da verlässt er sich offensichtlich ganz auf meine eigenen Deutschkenntnisse –, ist, wie ich das Verb *debuggen* zu beugen habe. Insbesondere habe ich Probleme mit dem Doppel-*g*, weil mir auf die Schnelle kein einziges deutsches Wort eingefallen ist, das ein solches enthält – wenngleich ich bedingt durch meinen Wohnort in Franken bestimmt auch irgendein Wort mit *ck* benutzen könnte.

Geholfen wurde mir dann beispielsweise von Reverso [3], ein Web-Auftritt, der sich neben Übersetzungen und Wörterbüchern auch als Anbieter von Konjugation versteht. Dort findet man alle Formen, unter der Annahme, dass *debuggen* ein deutsches Wort ist. Damit ist klar, dass *ich debugge* und *du debuggst*, *wir gedebuggt haben* und *gedebuggt haben werden*. Ob *debuggend* oder *gedebuggt*, wenn *ihr debugget*, *Du debuggest* oder *ich debugge*, oder gar *du debuggtest* – für jede Form ist etwas dabei.

Beim Konjuktiv allerdings würde ich eine andere Form bevorzugen, die diese klarer hervorheben würde. Inspiration findet man in solchen Fällen immer bei der Gesellschaft zur Stärkung der Verben [4]. Um beispielsweise den Indikativ Präsens *ich debugge* besser vom gleich lautenden Konjunktiv I Präsens zu unterscheiden, würde mir *ich debügge* besser gefallen. Aber das würde vielleicht dann doch zu weit gehen.

Übrigens hilft einem Reverso auch bei nicht deutschen Wörtern. Möchte man beispielsweise *transmogrifizieren* konjugieren, so weist die Seite freundlich darauf hin, dass es sich dabei nicht um ein deutsches Wort handelt, konjugiert es aber trotzdem.

Referenzen

[1] Duden *Deutsches Universalwörterbuch*, 6. Aufl. Mannheim 2006 [CD-ROM]

[2] Duden *Das Fremdwörterbuch*, 7. Aufl. Mannheim 2001 [CD-ROM]

[3] Reverso *Übersetzung, online Übersetzer, Wörterbuch, Konjugation*, http://www.reverso.net/text_translation.aspx?lang=DE

[4] GSV *Gesellschaft zur Stärkung der Verben*, http://verben.texttheater.net/Startseite

Kurzbiographie

Michael Wiedeking (michael.wiedeking@mathema.de) ist Gründer und Geschäftsführer der MATHEMA Software GmbH, die sich von Anfang an mit Objekttechnologien und dem professionellen Einsatz von Java einen Namen gemacht hat. Er ist Java-Programmierer der ersten Stunde, „sammelt“ Programmiersprachen und beschäftigt sich mit deren Design und Implementierung.

Kaffeesatz

Flash-Back

von Michael Wiedeking

Wenn man durch das Silicon Valley fährt, ist das wie eine kleine Entdeckungsfahrt. An jeder Ecke findet man jemanden aus der IT-Branche, der Rang und Namen hat. Biegt man dort rechts ab, so steht man beispielsweise vor Yahoo, kurz dahinter ist dann Oracle und Google ist auch nicht weit. Und PerkinElmer, die ich ganz vergessen hatte.

Gerade als ich mit dem Studieren angefangen hatte, musste ich auch einen Programmierschein machen, den POP-Schein fürs problemorientierte Programmieren. Die Programmiersprache war *Pascal* und programmiert wurde auf einer *PerkinElmer*.

Die *PerkinElmer* war – wenn ich das richtig in Erinnerung habe – eine *Unix*-Maschine und zeichnete sich dadurch aus, dass sie bestimmt fünfzig Terminals versorgen konnte. Die Maschine war damals ein Schmuckstück des Informatiklehrstuhls, denn der Jahrgang vor uns „durfte“ noch mit Lochkarten programmieren. Die Terminals waren also echter Luxus.

Die „Erstsemester“ durften allerdings nur den Zeileneditor *ed* aufrufen, der richtige Editor – der *vi* – war den höheren Semestern vorbehalten. Den *ed* gibt es auch heute noch auf jedem *Unix*-System und er leistet gelegentlich auch gute Dienste, wenn alles andere versagt.

Ich kann mich nur noch wage erinnern, aber irgendwie konnte man, hatte man den Editor mit einem Dateinamen aufgerufen, die korrespondierende Datei zeilen- oder blockweise anschauen und dann einzelne Zeilen ändern. Irgendwie hatte das was mit regulären Ausdrücken zu tun und mit deren Hilfe konnte man einzelne Zeichen oder Fragmente einfügen, löschen oder austauschen.

Ach was war das eine Wohltat, den *vi* zu nutzen. Alleine die Tatsache, dass man anhand des Cursors die aktuelle Zeile sehen konnte, war Gold wert. Den Cursor über die Tasten *h*, *j*, *k* und *l* zu bewegen, war ein bisschen abenteuerlich, hat mir aber in den vergangenen Jahrzehnten (sic!) so manches Mal, wenn ein Terminal falsch konfiguriert war und deshalb die Cursor-Tasten nicht funktionierten, sehr geholfen.

Für mich war PerkinElmer immer so etwas wie Pan Am, etwas was früher einmal wichtig war, aber dann von der Zeit eingeholt und wegoptimiert wurde. Ach, das ist ja schön, dass es den Laden noch gibt. Zumindest deswegen, weil er mich an die alten, zum Glück vergangenen Zeiten erinnert hat.

Lektüre

Praxis der User Interface-Entwicklung
Informationsstrukturen, Designpatterns, Vorgehensmuster
Taschenbuch: 209 Seiten, Deutsch
Vieweg+Teubner Verlag, 2011
ISBN 3834807281 (978-3834807281)

rezensiert von Frank Goraus

Der Titel dieses Buches hatte mein Interesse geweckt. Ich beschäftige mich gerne auch mit dem Design von Anwendungen und *User-Interfaces*, und weiss ein optisches Zuckerl oder winzige durchdachte Features sehr zu würdigen. Darum kam mir das Buch gerade recht um meine Kenntnisse diesbezüglich noch weiter zu vertiefen. Im Folgenden ist nun meine Meinung zu dem genannten Buch zu lesen.

Für alle etwas

Eines vorne weg: Ich finde das Buch sehr gut, doch konnte es meine Erwartungen leider nicht erfüllen. Entweder ist mein Vorwissen doch größer als ich gedacht habe oder es gibt einfach gar nicht soviel zu beachten.

Warum mich das Buch dennoch begeistert hat, lag wiederum in der Art wie der Autor alle Aspekte, die bei der *User-Interface*-Entwicklung anfallen, beleuchtet und aufgezeigt hat, wie man sie angehen kann. Allerdings weniger als eine Art Cookbook mit vielen konkreten „Für Problem A kann man folgende Lösung B anwenden."-Beispielen oder Best Practices sondern viel mehr mit verschiedensten Techniken rund um Entwurf und Dokumentation.

Der Autor gibt sich auch viel Mühe den größtmöglichen Zielgruppen-Bereich abzudecken. Dies wird einerseits auf dem Rückentext deutlich, in dem als Zielgruppen UI-Designer, Consultants, Spezifikateure (Requirements Engineers), Answendungsprogrammierer, Projektleiter und IT-Abteilungsleiter angegeben wurden. Anderseits auch wenn man dann das erste einleitende Kapitel liest. Hier wird viel mit Beispielen gearbeitet und viel erklärt, um auch möglichst jeden mit unterschiedlichem Vorwissensstand einzufangen und in das Thema zu bringen. Die Beispiele beleuchten hier viele recht einfache und simple Gegenstände und zeigen daran den Sinn eines guten *Interfaces* und welche Teile wie dazu beitragen. Als Beispiel wäre da zum Beispiel Besteck ohne Griffe oder ein Hammer mit zwei spitzen Kopfenden. Einfach aber effektiv, denn so versteht es jeder.

Auch sehr effektiv sind die *Road Checks*, eine Art kleiner Fragenkatalog mit Testfragen ob man das zuvor gelesene auch wirklich verstanden hat. Etwas irritierend in diesem Zusammenhang war jedoch, dass die Erläuterungen, was dieses Buch für die einzelnen Zielgruppen bringt, wie der rote Faden ausschaut und insbesondere was die Road Checks sind und wie sie funktionieren, erst am Ende von Kapitel 1 im Abschnitt „Quid pro quo" zu finden sind. Nachdem man in den drei vorherigen Abschnitten bereits damit konfrontiert wurde. Zumindest war dies in der mir vorliegenden Erstauflage so.

Inhalt

Nach dieser recht ausführlichen Einleitung folgt das Buch dann dem roten Faden der Projektphasen. In der Analyse wird aufgezeigt wie man u. a. mit *Mindmaps* oder *UML*-Diagrammen die Anforderungen möglichst effektiv erfassen kann.

Danach geht es um die Konzeption und das Design, wo beschrieben wird wie man am besten das entworfene Konzept dokumentiert und schon eigene *Styleguide*-Definitionen erfasst.

Im nächsten Kapitel geht es um die Verwendung und Auswahl von Input- und Interaktionselementen sowie dem möglichst sinnvollen Aufbau von Dialogseiten.

Den Abschluss macht dann ein Kapitel über Design und Redesign. In dem es vor allem um die sinnvolle Herangehensweise an Entwürfe und Entwurfsänderungen geht. Hier wird u. a. kurz auf CRs (*Change Requests*), *Scrum* oder *Extreme Programming* eingegangen.

Persönliches Fazit

Wie eingangs erwähnt, finde ich dieses Buch sehr gut, da es einen umfassenden Überblick und Einblick gibt, über alles was einem beim Entwurf eines UIs begegnen wird oder man im Zweifelsfall einsetzen sollte. Auch konkrete UI-Elemente werden hier aufgezeigt und wie man sie sinnvoll einsetzt. Man bekommt also alle praktischen Werkzeuge zur Entwicklung von UIs vorgestellt, womit sich das Buch im Zweifelsfall auch gut als Nachschlagewerk nutzen lässt.

Persönlich hatte ich jedoch gehofft noch ein wenig mehr Einblick zu bekommen, wie man am besten herangeht, wenn man den „Wunschkatalog“ des Kunden vorliegen hat und dann daraus ein möglichst praktisches und schickes UI entwerfen soll. Das soll nicht heißen, dass die entsprechenden Werkzeuge und Methodiken nicht beschrieben worden wären. Doch wo fange ich am besten an? Was hat sich bewährt? Man kann Eingabeformulare auf viele Arten gestalten. Als mehrere Prozessschritte mit ein- und ausklapp- oder -blendbaren Blöcken, oder als scrollbaren Monolithen. Was davon ist am sinnvollsten? Solche Fragen sind für mich leider noch offen, aber dafür ist das Buch wohl auch nicht ausgelegt und liegt eher in einer falschen Erwartungshaltung meinerseits.

Wer also einen guten Überblick über die Herangehensweisen der UI-Entwicklung sucht, dem ist dieses Buch wärmstens zu empfehlen. Wen jedoch ähnliche Fragen wie mich quälen, der sollte zumindest nicht enttäuscht sein, wenn das Buch sie nicht beantwortet.

Kurzbiografie

Frank Goraus (frank.goraus@mathema.de) ist Senior Developer bei der MATHEMA Software GmbH in Erlangen. Seit 2006 beschäftigt er sich bereits mit der Entwicklung von JEE-Anwendungen, u. a. in Verbindung mit einem Portal-Server. Seine Liebe zum Detail verwirklicht er mit seinen Web-Design-Kenntnissen. In seiner Freizeit beschäftigt er sich außerdem mit Android-Entwicklung, verschiedensten Web-Frameworks und einem eigenen Projekt für eine Sammlungsverwaltung.

User Groups

Fehlt eine User Group? Sind Kontaktdaten falsch?
Dann geben Sie uns doch bitte Bescheid.

Bookware
Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0
Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de

Java User Groups

Deutschland

JUG Berlin Brandenburg
http://www.jug-bb.de
Kontakt: Herr Ralph Bergmann (orga@jug-bb.de)

Java User Group Saxony
Java User Group Dresden
http://www.jugsaxouy.de
Kontakt: Herr Torsten Rentsch (torsten@jugsaxony.de)
Herr Falk Hartmann (falk@jugsaxony.de)
Herr Kristian Rink (kristian@jugsaxony.de)

rheinjug e.V.
Java User Group Düsseldorf
Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1, 40225 Düsseldorf
http://www.rheinjug.de
Kontakt: Herr Heiko Sippel (info@rheinjug.de)

ruhrjug
Java User Group Essen
Glaspavillon Uni-Campus
Universitätsstr. 12, 45127 Essen
http://www.ruhrjug.de
Kontakt: Herr Heiko Sippel (heiko.sippel@ruhrjug.de)

JUGF
Java User Group Frankfurt
http://www.jugf.de
Kontakt: Herr Alexander Culum
(alexander.culum@web.de)

JUG Deutschland e.V.
Java User Group Deutschland e.V.
c/o asc-Dienstleistungs GmbH
Ehrengard-Schramm-Weg 11, 37085 Göttingen
http://www.java.de (office@java.de)

JUG Hamburg
Java User Group Hamburg
http://www.jughh.org

JUG Karlsruhe
Java User Group Karlsruhe
Universität Karlsruhe, Gebäude 50.34
Am Fasanengarten 4, 76131 Karlsruhe
http://jug-karlsruhe.de
jug-karlsruhe@gmail.com

JUGC
Java User Group Köln
http://www.jugcologne.org
Kontakt: Herr Michael Hüttermann
(michael@huettermann.net)

jugm
Java User Group München
Jupiterweg 8, 85586 Poing
http://www.jugm.de
Kontakt: Herr Andreas Haug (ah@jugm.de)

JUG Münster
Java User Group für Münster und das Münsterland
http://www.jug-muenster.de
Kontakt: Herr Thomas Kruse (tkjugi@sforce.org)

JUG MeNue
Java User Group der Metropolregion Nürnberg
c/o MATHEMA Software GmbH
Henkestraße 91, 91052 Erlangen
http://www.jug-n.de
Kontakt: Frau Alexandra Specht
(alexandra.specht@jug-n.de)

JUG Ostfalen
Java User Group Ostfalen
(Braunschweig, Wolfsburg, Hannover)
Siekstraße 4, 38444 Wolfsburg
http://www.jug-ostfalen.de
Kontakt: Uwe Sauerbrei (info@jug-ostfalen.de)

JUGS e.V.
Java User Group Stuttgart e.V.
c/o Dr. Michael Paus
Schönaicherstraße 3, 70597 Stuttgart
http://www.jugs.org
Kontakt: Herr Dr. Micheal Paus (mp@jugs.org)
Herr Hagen Stanek (hs@jugs.org)

Schweiz

JUGS
Java User Group Switzerland
Postfach 2322, 8033 Zürich
http://www.jugs.ch (info@jugs.ch)
Kontakt: Frau Ursula Burri

.Net User Groups

Deutschland

.NET User Group OWL
http://www.gedoplan.de/cms/gedoplan/ak/ms_net
c/o GEDOPLAN GmbH
Stieghorster Str. 60, 33605 Bielefeld

.Net User Group Bonn
.NET User Group "Bonn-to-Code.Net"
Langwartweg 101, 53129 Bonn
http://www.bonn-to-code.net (mail@bonn-to-code.net)
Kontakt: Herr Roland Weigelt

Die Dodnedder
.NET User Group Franken
http://www.dodnedder.de
Kontakt: Herr Bernd Hengelein
Herr Thomas Müller, Udo Neßhöver
(dodned@googlemail.com)

.net Usergroup Frankfurt
c/o Thomas Sohnrey, Agile IService
Mendelssohnstrasse 80, 60325 Frankfurt
http://www.dotnet-ug-frankfurt.de
Kontakt: Herr Thomas 'Teddy' Sohnrey
(thomas.sohnrey@gmx.de)

.NET DGH
.NET Developers Group Hannover
Landwehrstraße 85, 30519 Hannover
http://www.dotnet-hannover.de
Kontakt: Herr Friedhelm Drecktrah
(friedhelm@drecktrah.de)

INdotNET
Ingolstädter .NET Developers Group
http://www.indot.net
Kontakt: Herr Gregor Biswanger
(gregor.biswanger@web-enliven.de)

DNUG-Köln
DotNetUserGroup Köln
Goldammerweg 325, 50829 Köln
http://www.dnug-koeln.de
Kontakt: Herr Albert Weinert (info@der-albert.com)

.Net User Group Leipzig
Brockhausstraße 26, 04229 Leipzig
http://www.dotnet-leipzig.de
Kontakt: Herr Alexander Groß (agross@dotnet-leipzig.de)
Herr Torsten Weber (tweber@dotnet-leipzig.de)

.NET Developers Group München
http://www.munichdot.net
Kontakt: Hardy Erlinger (hardy.erlinger@hotmail.com)

.NET User Group Oldenburg
c/o Hilmar Bunjes und Yvette Teiken
Sachsenstr. 24, 26121 Oldenburg
http://www.dotnet-oldenburg.de
Kontakt: Herr Hilmar Bunjes
(hilmar.bunjes@dotnet-oldenburg.de)
Yvette Teiken (yvette.teiken@dotnet-oldenburg.de)

.NET User Group Paderborn
c/o Net at Work Netzwerksysteme GmbH,
Am Hoppenhof 32, 33104 Paderborn
http://www.dotnet-paderborn.de
(raacke@dotnet-paderborn.de)
Kontakt: Herr Mathias Raacke

.Net Developers Group Stuttgart
Tieto Deutschland GmbH
Mittlerer Pfad 2, 70499 Stuttgart
http://www.devgroup-stuttgart.de
(GroupLeader@devgroup-stuttgart.de)
Kontakt: Frau Catrin Busley

.net Developer-Group Ulm
c/o artiso solutions GmbH
Oberer Wiesenweg 25, 89134 Blaustein
http://www.dotnet-ulm.de
Kontakt: Herr Thomas Schissler (tschissler@artiso.com)

Österreich

.NET Usergroup Rheintal
c/o Computer Studio Kogoj
Josefgasse 11, 6800 Feldkirch
http://usergroups.at/blogs/dotnetusergrouprheintal/default.aspx
Kontakt: Herr Thomas Kogoj (thomas@kogoj.com)

.NET User Group Austria
c/o Global Knowledge Network GmbH,
Gutheil Schoder Gasse 7a, 1101 Wien
http://usergroups.at/blogs/dotnetusergroupaustria/default.aspx
Kontakt: Herr Christian Nagel (ug@christiannagel.com)

Software Craftsmanship Communities

Deutschland

Softwerkskammer – Mehrere regionale Gruppen unter einem Dach, http://www.softwerkskammer.de

MATHEMA

meet the experts of enterprise infrastructure

Refactoring
Verbesserung von Design und Quellcode bestehender Software,
16. – 17. August 2012, 25. – 26. Oktober 2012,
925,– € (zzgl. 19 % MwSt.)

Neues in Java 7
Die nächste Java Generation
20. – 21. August 2012, 5. – 6. November 2012
835,– € (zzgl. 19 % MwSt.)

Web-Anwendungen mit JavaServer Faces (JSF)
Komponentenbasierte Web-Entwicklung mit JavaServer Faces (JSF)
29. – 31. August 2012, 21. – 23. November 2012,
1.180,– € (zzgl. 19 % MwSt.)

Sicherheitskonzepte unter Java
Mechanismen für Datenschutz und Datensicherheit in Netzwerken
30. – 31. August 2012, 29. – 30. November 2012
925,– € (zzgl. 19 % MwSt.)

Objektorientierte Analyse und Design mit UML und Design Patterns
Methoden und Prinzipen für die Entwicklung von OO-Modellen und die Dokumentation mit der UML
15. – 17. 10. 2012,
1.180,– € (zzgl. 19 % MwSt.)

Lesen bildet. **Training macht fit.**

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de

MATHEMA

Software-Entwickler (m/w)
Software-Architekt (m/w)

Arbeiten Sie gerne selbstständig, motiviert und im Team? Haben Sie gesunden Ehrgeiz und Lust, Verantwortung zu übernehmen?

Wir bieten Ihnen erstklassigen Wissensaustausch, ein tolles Umfeld, spannende Projekte in den unterschiedlichsten Branchen und Bereichen sowie herausfordernde Produktentwicklung.

Wenn Sie ihr Know-how gerne auch als Trainer oder Coach weitergeben möchten, Sie über Berufserfahrung mit verteilten Systemen verfügen und Ihnen Komponenten- und Objektorientierung im .Net- oder JEE-Umfeld vertraut sind, dann lernen Sie uns doch kennen.

Wir freuen uns auf Ihre Bewerbung!

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de

Das Allerletzte

Dies ist kein Scherz!
Diese Anzeige wurde tatsächlich in der freien
Wildbahn angetroffen.

Ist Ihnen auch schon einmal ein Exemplar dieser
Gattung über den Weg gelaufen?
Dann scheuen Sie sich bitte nicht, uns das mitzuteilen.

Der nächste KaffeeKlatsch erscheint im August.

Herbstcampus

Wissenstransfer par excellence

3.– 6. September 2012

in Nürnberg