



Kombination textueller und grafischer Editoren

TMF meets GMF

Während mit GMF schon seit Langem ein De-facto-Standard für grafische Modellierung im Eclipse-Umfeld existiert, hat sich in den letzten Jahren mit TMF/Xtext ebenfalls ein Framework zur textuellen Modellierung etabliert. Trotz ihrer sich gut ergänzenden Vorteile werden diese beiden Frameworks aufgrund ihrer Eigenständigkeit und individuellen Komplexität aber nur selten kombiniert. Da bislang keine etablierten Integrationsansätze existieren, diskutieren wir in diesem Artikel einige Möglichkeiten, die wir in unserer Projektarbeit kennen gelernt und eingesetzt haben.

von Andreas Müller und Alexander Nyßen

Die modellbasierte Entwicklung ist mittlerweile fest in der industriellen Praxis etabliert, nicht zuletzt, da durch Eclipse und insbesondere das Eclipse Modeling Project (EMP) das Erstellen von Modellierungswerkzeugen kein Hexenwerk mehr darstellt. Während bis vor wenigen Jahren noch stark auf – mitunter etwas schwergewichtige – grafische Modellierungsansätze gesetzt wurde, findet man aktuell, insbesondere beflügelt durch Xtext, vor allem leichtgewichtige, oft rein textuelle Ansätze. Das ist wohl vor allem damit zu erklären, dass sich besonders lineare Zusammenhänge in Form von Text einfach und klar abbilden lassen und eine grafische Modellierung im Fall relativ „linearer“ Strukturen als recht umständlich

empfunden wird. Da grafische Modellierungssprachen aber im Allgemeinen gegenüber textuellen Sprachen den Vorteil bieten, („nicht lineare“) Beziehungen zwischen Modellelementen übersichtlicher darzustellen, gibt es nach wie vor auch vielfältige Einsatzmöglichkeiten für grafische Modellierungsansätze. Oft ist wohl ein integrierter Ansatz, bei dem die innere Struktur eines Modellelements textuell beschrieben wird, während die externen Beziehungen eines Elements grafisch notiert werden, die geeignetste Modellierungsform. Ohne hier auf die detaillierte Gestaltung einer Modellierungssprache näher einzugehen (dem interessierten Leser empfehlen wir [1] zur weiteren Lektüre), stellt sich jedoch die grundlegende Frage, ob das gesamte Modell oder nur bestimmte Teile textuell repräsentiert werden sollen.

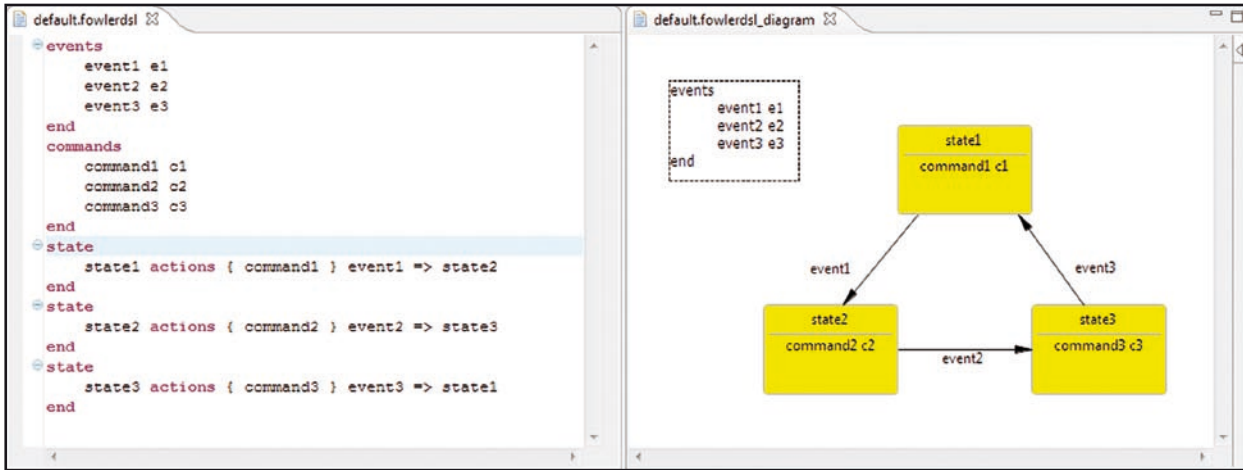


Abb. 1: Simultanes Editieren einer Xtext-Ressource

Für die Kombination textueller und grafischer Editoren ist das entscheidend, da beide Möglichkeiten aus technischer Sicht grundlegend unterschiedliche Ansätze verfolgen. Während im ersten Fall das parallele Editieren einer Ressource mit textuellem und grafischem Editor ermöglicht wird, besteht im zweiten Fall (nur) die Möglichkeit, die bekannten Features von Xtext (Linking und Scoping, Syntax-Highlighting, Validierung und Autovervollständigung) innerhalb eines GMF-Editors nutzbar zu machen. Im Folgenden diskutieren wir beide Möglichkeiten anhand des Beispielprojekts Fowler DSL. Dieses Projekt basiert auf dem mit Xtext mitgelieferten State-machine-Beispiel, das wir zur besseren Integration angepasst und um einen grafischen Editor erweitert haben, sodass hiermit alle vorgestellten Integrationsansätze demonstriert werden können. Die Quellen stehen unter [2] zum Download zur Verfügung.

Paralleles Editieren einer Xtext-Ressource mit Xtext- und GMF-Editor

Um das Erstellen mehrere Diagramme für ein einzelnes semantisches Modell realisieren zu können, werden bei einem GMF-Editor die grafischen Layoutinformationen in einem eigenen Modell, dem so genannten Notationsmodell, abgelegt. Das Synchronisieren grafischer und semantischer Modellelemente geschieht dabei in GMF über so genannte *CanonicalEditPolicies*, Strategien, die auf semantische Modelländerungen reagieren und das Notationsmodell entsprechend aktualisieren, indem sie bei Bedarf grafische Modellelemente neu erzeugen oder bestehende entfernen. Da sich ein GMF-Editor für beliebige EMF-basierte Modelle erstellen lässt und sich Xtext-Dokumente über den EMF-Resource-Mechanismus transparent laden lassen, können sie ebenfalls als semantisches Modell in einem GMF-Editor bearbeitet werden. Damit ist es sogar möglich, ein Modell parallel mittels Xtext- und GMF-Editor zu bearbeiten, wie in **Abbildung 1** am State-machine-Beispiel dargestellt.

Obwohl EMF sogar das gleichzeitige Editieren eines im Speicher geladenen Modells über eine geteilte *Editing-Domain* erlaubt, bietet sich das für die Integration von Xtext- und GMF-Editoren nicht an. Der wesentliche

Grund hierfür ist, dass der Xtext-Editor bei jeder Änderung am Dokument ein erneutes Parsen der geänderten Textregion vornimmt, sodass Teile des abstrakten Syntaxbaums (AST), falls z. B. zwischenzeitlich ungültige Ausdrücke eingegeben werden, zeitweise wegfallen können. Die *CanonicalEditPolicies* des GMF-Editors würden bei einer solchen Änderung die grafischen Notationselemente augenblicklich ebenfalls entfernen, sodass die entsprechenden Layoutinformationen für ein semantisches Modellelement auch im Fall der späteren vollständigen Wiederherstellung des AST verloren wären.

Daher ist es sinnvoll, eine Synchronisierung von GMF- und Xtext-Editor rein auf Dateiebene zu realisieren, sodass Änderungen am Textdokument erst nach dem Speichern im grafischen Editor sichtbar werden, analog mittels grafischem Editor vorgenommene Änderungen erst beim Speichern im Xtext-Editor. Auch wenn wir die Umsetzung einer solchen Integration im beiliegenden State-machine-Beispiel realisiert haben, möchten wir an dieser Stelle technisch nicht tiefer ins Detail gehen. Stattdessen verweisen wir auf die Xtext-Hilfe [3], wo ein Beispiel für das parallele Editieren mit GMF- und Xtext-Editoren anhand eines Entity-Editors dokumentiert ist.

Einbettung von Xtext in einen GMF-Editor

Während sich Xtext-basierte Editoren auf das reine Editieren von Text beschränken, findet man in grafischen, GMF-basierten Editoren häufig eine Vermischung textueller und grafischer Artefakte. So werden zum Bei-

Listing 1

```
Injector injector = Guice.createInjector(module);
IAntlrParser parser = injector.getInstance(IAntlrParser.class);
IParseResult parseResult = parser.parse(parserRuleToApply, new StringReader(editString));
```

Listing 2: Anbindung des Xtext Serializers

```
Injector injector = Guice.createInjector(module);
Serializer serializer = injector.getInstance(Serializer.class);
String editString = serializer.serialize(eObject);
```

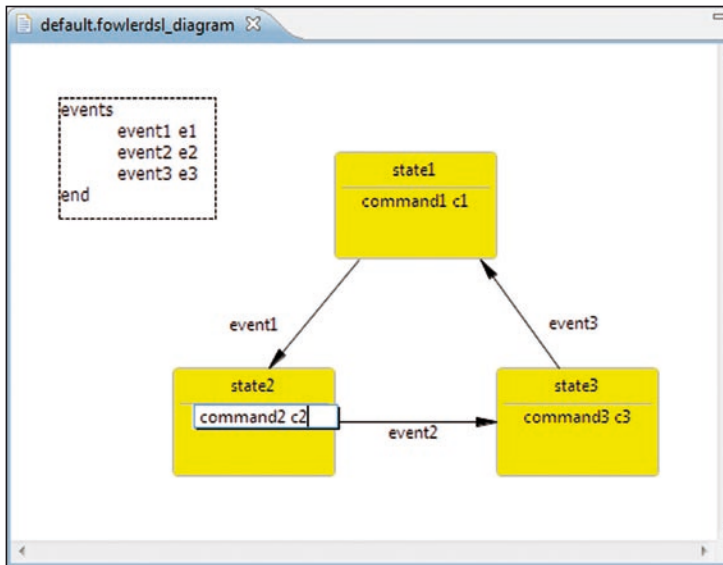


Abb. 2: Xtext-Parser-Anbindung via GMF-IParser-Schnittstelle

spiel häufig Labels oder Compartments mit textuellen Inhalten in Diagrammen dargestellt. Das Editieren solcher Textartefakte wird bei GMF, neben der indirekten Möglichkeit über die Properties View, auf der Basis des so genannten Direct-Editing-Mechanismus durch einen temporär eingeblendeten JFace *TextCellEditor* ermöglicht. Das Prüfen der syntaktischen Korrektheit des eingegebenen Textes wird hierbei über die von GMF spezifizierte *IParser*-Schnittstelle abgewickelt. Darüber hinaus ist diese Schnittstelle auch dafür zuständig, den angezeigten Text für ein semantisches Element zu ermitteln (Edit und Print String) sowie Unterstützung für eine mögliche Autovervollständigung zur Verfügung zu stellen.

Neben der Option, einen solchen *IParser* vollständig frei zu implementieren, bietet GMF die Möglichkeit, einen *MessageFormat*-basierten Parser von einer abstrakten Oberklasse abzuleiten; mithilfe von GMF Tooling

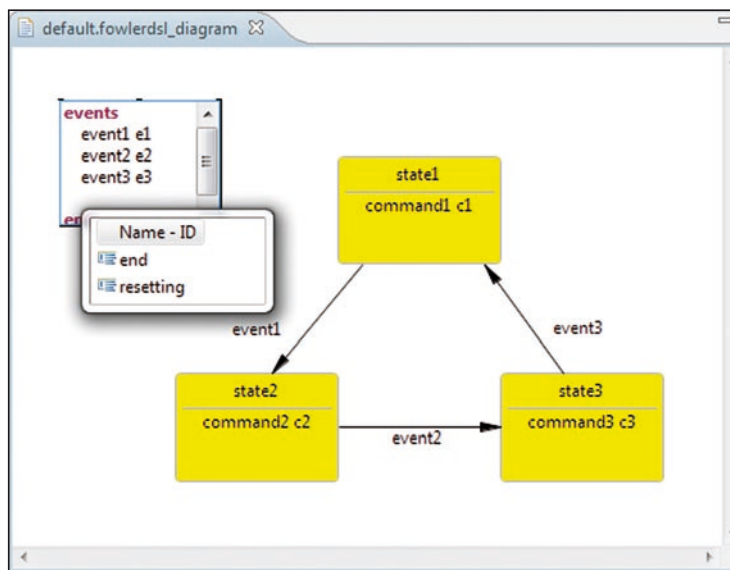


Abb. 3: XtextCellEditor integriert in GMF-Editor

lässt sich ein solcher *MessageFormat*-basierter Parser sogar relativ einfach generieren. Leider lassen sich auf dieser Basis komplexere Ausdrücke aber nur eingeschränkt spezifizieren, sodass es oft zu einer manuellen *IParser*-Implementierung keine echte Alternative gibt.

Da Xtext es erlaubt, auf der Basis einer Grammatik sowohl Parser und Serializer als auch Unterstützung für Autovervollständigung und Cross Referencing sehr einfach generativ zu entwickeln, liegt der Wunsch nahe, entsprechende Xtext-Artefakte im Kontext von GMF nutzen zu können. Hierfür haben wir grundlegend zwei mögliche Integrationspunkte identifiziert: zum einen die Anbindung des Xtext Parsers und Serializers an die GMF *IParser*-Schnittstelle und zum anderen die direkte Integration mittels eines angepassten Cell-Editors. Wir beleuchten beide Möglichkeiten im Folgenden anhand des State-Machine-Beispiels.

Anbindung von Xtext Parser und Serializer via IParser

Das Anbinden des durch das Xtext Tooling generierten Parsers an die von GMF bereitgestellte *IParser*-Schnittstelle stellt die wohl einfachste Kopplung der beiden Frameworks dar. Da keine der weitergehenden Xtext-Mechanismen zur Auflösung von Crossreferenzen oder zur Unterstützung von Autovervollständigung eingesetzt werden, ist damit allerdings auch nur eine rein syntaktische Prüfung der eingegebenen Texte umsetzbar. Voraussetzung ist, dass ein im Diagramm zu editierender Text einer Parser-Regel der Xtext-Grammatik zugeordnet werden kann, da der durch Xtext generierte ANTLR Parser dann mit der entsprechenden Parser-Regel als Einstiegspunkt aufgerufen werden kann, wie in Listing 1 dargestellt.

In unserem State-Machine-Beispiel ist das exemplarisch für die in Zuständen als Actions enthaltenen Commands umgesetzt. Hier werden, wie im Screenshot in **Abbildung 2** dargestellt, für ein Command-Objekt die Werte der beiden *name*- und *id*-Attribute in einem Text-String editiert.

Im Fall des textuellen Editierens eines gesamten *EObjects* kann eine weitere Zuständigkeit der *IParser*-Schnittstelle, nämlich das Ermitteln eines Print und Edit Strings zur Anzeige im Diagramm und während des Editierens, mithilfe des Xtext Serializers – wie in Listing 2 dargestellt – recht einfach umgesetzt werden.

Neben dem Parsen und Serialisieren ist die *IParser*-Schnittstelle auch dafür zuständig, die durch das Direct Editing bedingten Modelländerungen in einem GMF Command zu kapseln, mit dem die Änderungen schließlich auf dem semantischen Modell appliziert werden können. Auch beim Direct Editing werden die Änderungen nicht direkt, sondern erst durch das Ausführen eines GMF Commands vorgenommen. Dadurch wird sichergestellt, dass Änderungen auch wieder verworfen werden können.

Wird das semantische Modell nun vollständig in einer Xtext *Resource* abgelegt, kann man sich hier ebenfalls entsprechender Xtext-Mechanismen bedienen. Eine re-

lativ generische Lösung, bei der mithilfe des Xtext *Serializers* die textuelle Repräsentation eines Modellelements erzeugt und die entsprechende Textpassage in der Xtext *Resource* ersetzt wird, ist anhand des vorher schon erwähnten Entity-Beispiels in der Xtext-Hilfe [2] ebenfalls skizziert. Ist das Modell nicht vollständig in einer Xtext *Resource* abgelegt, so kann mithilfe des beim Parser-Aufruf zurückgegebenen *IParseResults* (Listing 1) auf den abstrakten Syntaxbaum für das entsprechende Textfragment zugegriffen werden. Da dieser bei Xtext schon aus *EObjects* aufgebaut ist, lassen sich hierbei über den Vergleich mit dem semantischen Modell Änderungen recht leicht bestimmen. Eine entsprechende Lösung haben wir exemplarisch im StateMachine-Beispiel für das Editieren der Commands umgesetzt.

Anbindung von Xtext via CellEditor

Um die von Xtext bekannten Komfortfunktionalitäten wie Syntax-Highlighting, Validierung und Autovervollständigung nutzen zu können, ist eine Adaption an die *IParser*-Schnittstelle von GMF nicht ausreichend, da sich hierüber das beim Direct Editing angebotene User Interface nicht anpassen lässt. Wie schon skizziert, verwendet GMF standardmäßig JFace-basierte *TextCellEditoren* als User Interface, die für das Direct Editing von Text temporär eingeblendet werden.

Listing 3: Anpassung des semantischen Parsers

```
public class FowlerDslEventParser extends FowlerDslParser {
    @Override
    protected String getDefaultRuleName() {
        return "EventContainer";
    }
}
```

Listing 4

```
public class FowlerDslEventParser extends FowlerDslParser {
    @Override
    protected Collection getFollowElements(AbstractInternalContentAssistParser
        parser){
        InternalFowlerDslParser typedParser = (InternalFowlerDslParser) parser;
        // specify parser entry rule
        typedParser.entryRuleEventContainer();
        return typedParser.getFollowElements();
    }
}
```

Listing 5

```
public class FowlerDslEventContainerRuntimeModule extends FowlerDslRuntimeModule {
    @Override
    public Class<? extends IAntlrParser> bindIAntlrParser() {
        return FowlerDslEventParser.class;
    }
}
```

Listing 6: Erzeugen eines gepatchten Modules

```
public Module getEventContainerScopedModule() {
    return Modules.override(
        Modules.override(new FowlerDslEventRuntimeModule()).with(
            new FowlerDslEventUiModule(this))).with(
        new SharedModule());
}
```

meat Motion®



- Videocasts
- Messefilme
- Firmenvideos
- Recruitingvideos
- Videointegration
in sozialen Netzwerken



Jetzt kostenlose
Beratung vereinbaren!

info@meat-frankfurt.com



Möchte man die in einem Xtext-Editor verfügbaren Komfortfunktionen auch während des Direct Editings in einem GMF-Editor unterstützen, so ist das nur über die Einbindung eines entsprechend angepassten *CellEditors* möglich. In unserem Beispielprojekt [2] haben wir eine Implementierung eines solchen *XtextCellEditors* realisiert. Das *CellEditor* API ist ein Bestandteil von JFace, und es gibt keine Abhängigkeiten zum Eclipse Editor API. Somit kann eine solche Klasse theoretisch in jeden beliebigen *StructuredViewer* als *CellEditor* eingebunden werden, z. B. in den *TableViewer* der Property Pages.

Abbildung 3 zeigt seine Integration anhand des State-machine-Beispiels. Innerhalb des grafischen Editors befindet sich dort ein *Compartment*, das dem Benutzer erlaubt, Ereignisse textuell zu spezifizieren. Beim Direct Editing dieses *Compartment*s kommt der realisierte *XtextCellEditor* zum Einsatz. Somit sind dann sowohl Syntax-Highlighting als auch Autovervollständigung und Validierung während des Direct Editing möglich.

Um den *XtextCellEditor* in das User Interface einzubinden, muss lediglich im *EditPart* die Methode *performDirectEditRequest* überschrieben und der für das Direct Editing zuständige *DirectEditManager* ausgetauscht werden, damit anstelle eines einfachen *TextCellEditors* der *XtextCellEditor* eingebunden werden kann. Standardmäßig ist ein solcher *XtextCellEditor* nicht auf einen bestimmten Teilbereich der Grammatik eingeschränkt, sodass z. B. bei der Autovervollständigung alle möglichen Parser-Regeln in Betracht gezogen werden. Wie im Beispiel zuvor möchten wir jedoch hier explizit eine Parser-Regel (*EventContainer*) als Einstiegspunkt verwenden. Die *EventContainer*-Parser-Regel gibt es im Xtext-FowlerDSL-Beispielprojekt nicht. Sie wurde eingefügt, da die hier beschriebene Integrationslösung eine eindeutige Einstiegsregel zwingend erfordert. Da der Xtext Parser im Unterschied zur vorstehend beschriebenen *IParser*-Lösung hier aber nicht direkt eingebunden ist, können wir den Aufruf des Parsers nicht entsprechend parametrisieren und müssen ihn durch eine angepasste Version ersetzen. Listing 3 zeigt die notwendigen Anpassungen am generierten Parser.

Neben diesem semantischen Parser generiert Xtext noch einen weiteren Parser für die Autovervollständigung. Auch dieser muss an die gewünschte Einstiegsregel angepasst werden, wie Listing 4 zeigt.

Der Austausch der Parser-Implementierungen lässt sich dank des von Xtext eingesetzten Google Guice Dependency Injection Frameworks recht einfach realisieren, indem die entsprechenden Module überschrieben und die Parser Bindings angepasst werden, wie in Listing 5 für das Runtime Module dargestellt.

Der *XtextCellEditor* muss dann nur noch mit einem entsprechend gepatchten Module, das wie in Listing 6 erzeugt werden kann, parametrisiert werden.

Um die Änderungen innerhalb des *XtextCellEditors* ins Modell zu überführen, müssen diese wie auch bei der Anbindung an die *IParser*-Schnittstelle in GMF Commands gekapselt werden. Während das im Normalfall

durch eine *DirectEditPolicy* an die *IParser*-Schnittstelle von GMF weiterdelegiert wird, muss dieser „Umweg“ hier nicht zwangsläufig genommen werden. Stattdessen kann durch Austausch der *DirectEditPolicy* das reine Erstellen eines solchen GMF-Commands auch losgelöst von der *IParser*-Schnittstelle realisiert werden.

Zum Ableiten eines Edit Strings für das Modellelement kann wie im Fall der *IParser*-Anbindung auf den Xtext Serializer zurückgegriffen werden. Hier reicht es ebenfalls aus, wenn der *DirectEditManager* diese Zuständigkeit nicht an die *IParser*-Schnittstelle, sondern direkt an das entsprechende *EditPart* delegiert und dort der Xtext Serializer, wie in Listing 2 dargestellt, eingebunden wird.

Fazit

Wie wir anhand unseres State-machine-Beispiels demonstriert haben, bestehen schon jetzt einige Möglichkeiten zur Integration von GMF- und Xtext-basierten Editoren, auch wenn die zugrunde liegenden Frameworks diese nicht explizit vorsehen. So lassen sich neben dem parallelen Editieren eines Modells sowohl ein durch Xtext generierter Parser als auch ein *CellEditor* mit den von Xtext bekannten UI-Features wie Syntax-Highlighting und Autovervollständigung schon jetzt in einem GMF-Editor nutzbar machen. Alle Ansätze erfordern aktuell jedoch viel Handarbeit und relativ umfangreiche Kenntnisse der jeweiligen Frameworks. Zudem sind sie zum Teil nur unter bestimmten Voraussetzungen anwendbar. Für eine wirklich nahtlose Integration textueller und grafischer Modellierung liegt daher noch ein gewisser Weg vor uns. Betrachtet man aber, mit welchem Engagement die Modeling-Community ihre Entwicklungen vorantreibt, so sind wir zuversichtlich, dass hier in Kürze tragfähige Lösungen entstehen werden.



Andreas Müller (andreas.mueller@itemis.de) arbeitet als Software Engineer bei der itemis AG in Lünen. Dort beschäftigt er sich schwerpunktmäßig mit der Entwicklung von textuellen und grafischen Modellierungswerkzeugen auf Basis von Eclipse. Außerdem ist er Committer des Open-Source-Projekts YAKINDU.



Alexander Nyßen (alexander.nyssen@itemis.de) ist Software Engineer bei der itemis AG in Lünen. Er beschäftigt sich dort vor allem mit Entwicklung und Beratung im Bereich grafischer Modellierungswerkzeuge und Eclipse-basierter Werkzeugketten. Darüber hinaus ist er als Committer im Eclipse Graphical Editing Framework (GEF) sowie in weiteren Open-Source-Projekten engagiert.

Links & Literatur

- [1] Daniel L. Moody (2009): „The ‚Physics‘ of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering.“ IEEE Transactions on Software Engineering. Vol. 35. No. 6, S.756-779
- [2] http://ftp.itemis.de/eclipse_magazin/tmfgmf
- [3] http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.xtext.doc/help/gmf_integration.html

27.– 29. JUNI 2011
HOLIDAY INN, MÜNCHEN



Java EE Summit

The Ultimate Java-EE-Event



Nicht verpassen!
Mit den Early-Bird-Preisen
bis 28. Mai 100 € sparen!

The Ultimate Java EE Event

Die Entwickler Akademie präsentiert Ihnen zusammen mit dem Java Magazin den ersten Java EE Summit! Das große Trainingsevent vermittelt Ihnen in drei Tagen die wichtigsten Java-EE-Themen in kompakter Form. Dabei können Sie Ihre individuellen Themenschwerpunkte aus insgesamt 18 Power Workshops wählen. Erleben Sie auch die beiden Speaker Panels am ersten und zweiten Abend, die Ihnen zusätzlich wertvolle Informationen bieten.

In drei parallel laufenden Tracks – Technology, Best Practices und Architecture – lernen Sie unter anderem, wie Sie eine Java-EE-basierte Anwendung optimal planen, realisieren und zu einem erfolgreichen Abschluss bringen, wel-

che Stärken und Schwächen die verschiedenen Technologien haben oder wie Sie bei der Wahl einer geeigneten Architektur vorgehen sollten. Bei all dem profitieren Sie vom geballten Wissen und der Praxiserfahrung von sechs der bedeutendsten deutschsprachigen Java-EE-Experten. Alle sind bekannt als Autoren des Java Magazins oder als Top-Speaker der JAX-Konferenzen. Nutzen Sie auch die einmalige Chance, Ihre individuelle Fragen und Herausforderungen mit den hochkarätigen Experten aus der Praxis intensiv zu diskutieren.

Dieses einzigartige Java-EE-Event sollten Sie auf keinen Fall verpassen!

TRAINER



Adam Bien



Thilo Frotscher



Arne Limburg



Peter Roßbach



Lars Röwekamp



Jens Schumann

Alle Infos auf www.java-ee-summit.de

Präsentiert von



Java magazin

Powered by



Veranstalter



ÜBERSICHT ÜBER DIE POWER WORKSHOPS

Montag, 27. Juni 2011

	Track Technology	Track Best Practices	Track Architecture
09.00 – 12.30 Uhr	Java EE 5/6 für Einsteiger, Umsteiger und Aufsteiger <i>Lars Röwekamp, Programm Chair</i>	Modellierung im Java-EE-Umfeld (Domain-driven Design) <i>Arne Limburg</i>	Java-EE-Architekturen im Überblick <i>Jens Schumann</i>
13.30 – 17.00 Uhr	Java EE Core: JPA 2.x <i>Arne Limburg</i>	Messaging und Events mit Java EE <i>Jens Schumann</i>	„Good old“ Schichtenarchitektur <i>Lars Röwekamp, Programm Chair</i>
17.30 – 18.30 Uhr	Speaker Panel: Java EE – back in Town		

Dienstag, 28. Juni 2011

	Track Technology	Track Best Practices	Track Architecture
09.00 – 12.30 Uhr	Java EE Core: EJB 3.x <i>Jens Schumann</i>	JAVA EE Server auswählen und betreiben <i>Peter Roßbach</i>	Java-EE-Patterns & J2EE-Anti-Patterns <i>Adam Bien</i>
13.30 – 17.00 Uhr	Java EE Core: JSF 2.0 <i>Lars Röwekamp, Programm Chair</i>	Wie testet man Java EE 6 – Real World Unit, Integration, Acceptance und Stress Tests <i>Adam Bien</i>	Mit Java EE große Webarchitekturen gestalten <i>Peter Roßbach</i>
18.00 – 19.00 Uhr	Speaker Panel: Java EE in der Cloud		

Mittwoch, 29. Juni 2011

	Track Technology	Track Best Practices	Track Architecture
09.00 – 12.30 Uhr	Java EE Core: Web Services & Co. <i>Thilo Frotscher</i>	Wieviel Code kann man mit Java EE 6 löschen – die pragmatischen Best Practices, Teil 1 <i>Adam Bien</i>	Viele Wege führen zu Java EE <i>Jens Schumann</i>
13.30 – 17.00 Uhr	Java EE Core: CDI <i>Lars Röwekamp</i>	Wieviel Code kann man mit Java EE 6 löschen – die pragmatischen Best Practices, Teil 2 <i>Adam Bien</i>	Java EE Integration Architecture <i>Thilo Frotscher</i>

LEISTUNGEN IM ÜBERBLICK

- 18 intensive Power Workshops mit klarem Praxisbezug.
- Sechs der bekanntesten Java-EE-Experten vor Ort erleben.
- Eine ideale Plattform für Erfahrungsaustausch und Networking.
- Zwei spannende Speaker Panels am Montag- und Dienstagabend inkl. Snacks und Freibier.
- Genießen Sie die All-inclusive-Verpflegung mit Erfrischungen und Snacks in den Pausen sowie ein leckeres Mittagsbuffet.
- Sie erhalten die Materialien der Workshops in elektronischer Form.
- Ihr persönliches Zertifikat, Gratismagazine, kostenloser Internetzugang u.v.m.

POWER WORKSHOPS

Montag, 27. Juni 2011

Java EE 5/6 für Einsteiger, Umsteiger und Aufsteiger

Lars Röwekamp (open knowledge GmbH)

Die Java Enterprise Edition erlebt derzeit eine wahre Renaissance – und dies zurecht. Und auch das viel gescholtene Komponentenmodell EJB ist spätestens seit der Version 3.0 wieder salonfähig, bekommt mit CDI aber Konkurrenz aus den eigenen Reihen. Der Workshop zeigt, dass Java EE dank der Umorientierung hin zu „Ease of Development“ und „Flexibility“ deutlich mehr zu bieten hat als nur eine lose Sammlung von Spezifikationen. Es werden dabei die wichtigsten Neuerungen und Kernkonzepte von Java EE 5/6 erläutert und verdeutlicht, warum genau jetzt der richtige Zeitpunkt ist, auf Java EE zu setzen.

Modellierung im Java-EE-Umfeld (Domain-driven Design)

Arne Limburg (open knowledge GmbH)

Der Ansatz des Domain-driven Designs stellt die Modellierung der Fachdomäne in den Mittelpunkt der Anwendungsentwicklung. Aber was bedeutet das für den Softwareentwicklungsprozess? Und wie kann das Ziel eines fachlichen Domänenmodells im Umfeld von Java EE mit all seinen Design Patterns erreicht werden? In dem Workshop werden Vorgehensmodelle, Designpraktiken, -techniken und -prinzipien vorgestellt, um ein „Rich Domain-Model“ zu erstellen und weiterzuentwickeln, das die Fachlichkeit adäquat abbildet.

Java-EE-Architekturen im Überblick

Jens Schumann (open knowledge GmbH)

Enterprise Java liefert ein Rahmenwerk zur Umsetzung von Anwendungen aus dem Spektrum klassischer „synchroner“ Client-Server Anwendungen bis hin zu vollständig lose gekoppelter, nachrichtenbasierter Systeme. Im Rahmen dieses Workshops werden die unterschiedlichen, mit der Java-EE-Plattform realisierbaren architektonischen Ansätze vorgestellt. Dabei werden die von der Plattform vorgesehenen Technologien und Kommunikationsarten zugeordnet, ableitbare Deployment-Szenarien erörtert sowie die Wartbarkeit und Komplexität derartiger Architekturen diskutiert.

Java EE Core: JPA 2.x

Arne Limburg (open knowledge GmbH)

Mit JPA wurde in Java EE ein Standard geschaffen, um Object-Relational Mapping leichtgewichtig durchführen zu können. Aber wie kann dieser sinnvoll in „Real-Life-Szenarien“ eingesetzt werden? Neben dem Zusammenspiel mit anderen Java-EE-Technologien betrachtet dieser Workshop erweiterte Problemstellungen, wie Datenintegrität, Transaktionsgrenzen, Lazy Initialization und Datensichtbarkeit in Mehr-Benutzer-Applikationen. Des Weiteren wird ein Ausblick auf andere Persistentechnologien gegeben mit Schwerpunkt auf NoSQL und Persistence in der Cloud.

Messaging und Events mit Java EE

Jens Schumann (open knowledge GmbH)

Mit Java EE 6 wurde die technologische Basis für Message- bzw. Eventorientierte Architekturen nochmals erheblich erweitert. Doch was bedeutet das für Java-EE-Anwendungen? Dieser Workshop erläutert diese neuen Java EE 6 Features und erörtert, wann es sich heute lohnt, Use Cases oder gesamte Java-EE-Anwendungen vom klassischen synchronen Kommunikationsmodell auf lose gekoppelte Interaktion umzustellen. Neben der reinen technologischen Betrachtung werden klassische, aber auch auf den ersten Blick atypische Message- bzw. Eventorientierte Use Cases diskutiert.

„Good old“ Schichtenarchitektur

Lars Röwekamp (open knowledge GmbH)

Macht das gute, alte Schichtenmodell aus J2EE auch in Java EE noch Sinn? Und wenn ja, wie sehen dann die einzelnen Schichten aus und wie werden sie optimal miteinander verbunden? Der Workshop zeigt an einem praktischen Beispiel, welchen Einfluss die konzeptionellen Neuerungen in Java EE auf die klassische Schichtenarchitektur haben. Neben der Erläuterung etlicher Best Practices wird natürlich auch auf typische Pitfalls eingegangen.

Speaker Panel: „Java EE – back in Town“

Der Enterprise-Java-Standard scheint zurück. Aber ist die aktuelle Spezifikation wirklich so stark wie von vielen behauptet? Wo liegen die Grenzen? Was gilt es für die Zukunft noch in den Standard mit aufzunehmen? Und wann sollte man auf „fremde Welten“ zurückgreifen?

Dienstag, 28. Juni 2011

Java EE Core: EJB 3.x

Jens Schumann (open knowledge GmbH)

EJBs haben sich wieder zu einem zentralen Baustein einer Enterprise-Java-Architektur entwickelt, und das, nachdem diese lange Zeit als technologische Sackgasse galten. Dabei kommt ihnen zugute, dass neben der Vereinfachung des Entwicklungsprozesses auch Features wie Interceptors, Web Archive Deployment und Testkonfigurationen verständlich und benutzbar eingeführt wurden. Dieser Workshop widmet sich den grundlegenden und weiterführenden EJB-3.x-Features, wobei Transaktions- und Concurrency-Management, die Vorzüge von Stateful-Komponenten und EJB 3.x Unit Testing nicht zu kurz kommen.

JAVA-EE-Server auswählen und betreiben

Peter Roßbach (Freiberufler)

Seit einiger Zeit gibt es verschiedene Container, die Java EE 6 implementieren. Das Angebot der neuen Server ist verführerisch. Welche Vor- und Nachteile stehen uns in Entwicklung und Produktion bevor? Welche neuen Herausforderungen stellt sich im Deployment und dem Betrieb? Es wird ein Überblick über den Status der Java EE 6 Container gegeben. Mar-

kante Eigenschaften werden erläutert. Auch exotische Möglichkeiten, wie die Integration von Apache OpenWebBeans in den Apache Tomcat oder dem Betrieb in einer Cloud werden vorgestellt.

Java EE Patterns & J2EE Anti-Patterns

Adam Bien (adam-bien.com)

Transaktionale Konfiguration, Publish-/Subscribe ohne JMS, Integration von Legacy POJOs, asynchrone Batch-Verarbeitung, Plugins, syntaktische und semantische Validierung von Rich-Domain-Modellen und typische Erweiterungspunkte sind nur wenige Beispiele für Patterns, die sich ohne wesentlichen Mehraufwand mit Java EE 6 implementieren lassen. In diesem Workshop werden sowohl die neuen Patterns der Java EE 6 diskutiert als auch die überflüssigen DAOs, ServiceLocators, SessionFacades, DTOs der antiken J2EE 1.4 eliminiert.

Java EE Core: JSF 2.0

Lars Röwekamp (open knowledge GmbH)

JavaServer Faces 2.0 heißt das Zauberwort, wenn es um die Entwicklung webbasierter Anwendungen im Java-EE-Umfeld geht. Der Workshop führt in die wesentlichen Konzepte der JavaServer Faces ein und zeigt anhand praktischer Beispiele, wie sich mit JSF 2.0 moderne und skalierbare Webanwendungen auf Basis von wiederverwendbaren Komponenten realisieren lassen.

Wie testet man Java EE 6 – Real World Unit, Integration, Acceptance und Stress Tests

Adam Bien (adam-bien.com)

Man freut sich bereits über eine 80% Unit-Test-Abdeckung, dabei wurde lediglich sichergestellt, dass während der Tests 80% des Codes in einem einzigen Thread ausgeführt wurde. Nach der Implementierung einer Java-EE-6-Anwendung mit Ressourcen Ihrer Wahl (z.B. abhängige Komponenten, Entity Manager, JMS etc.), werden wir Unit-, Integration-, Akzeptanz-, Funktional- und Stress-Tests diskutieren, realisieren und in eine CI-Umgebung integrieren.

Mit Java EE große Webarchitekturen gestalten

Peter Roßbach (Freiberufler)

Mit dem neuen Servlet API 3.0 gibt es einen Aufbruch in eine neue Modularisierung und einfache Verarbeitung asynchroner Anfragen. Dies bietet neue Chancen für die Gestaltung großer Websystemen mit Java. Die Session erläutert den Servlet-3.0-Standard. Daraus resultieren erprobte und neue Konzepte, um skalierbare und hochverfügbare Webarchitekturen zu gestalten.

Speaker Panel: „Java EE in der Cloud“

„Software as a Service“ und die „Cloud“ sind derzeit zwei nahezu magische Ausdrücke. Wie lässt sich das Ganze sinnvoll mit Java EE vereinbaren? Wie können dabei reale Architekturen im produktiven Umfeld aussehen?

POWER WORKSHOPS

Mittwoch, 29. Juni 2011

Java EE Core: Web Services & Co.

Thilo Frotscher (Freiberufler)

Die Umsetzung von Integrationsanforderungen ist nach wie vor ein hochaktuelles Thema für viele Entwickler. Dieser Workshop beleuchtet, welche Bordmittel Java EE 6 hierfür mitbringt und wie sie in der Praxis angewendet werden. Daneben werden wir uns auch der Frage widmen, in welchen Fällen man mit Java EE 6 an Grenzen stößt und welche Alternativen jeweils zur Verfügung stehen.

Java EE Core: CDI

Lars Röwekamp (open knowledge GmbH)

Dank CDI (JSR 299) hat ein extrem leichtgewichtiger Dependency-Injection-Mechanismus Einzug in Java EE gehalten, der deutlich über die Injection von Infrastruktur hinaus geht. Der Workshop zeigt neben den Grundlagen von CDI, wie mithilfe Event-orientierter Kommunikation, Producern und Interceptoren

das fachliche Design der Anwendung in den Vordergrund rücken kann.

Wie viel Code kann man mit Java EE 6 löschen – die pragmatischen Best Practices, Teil 1 + 2

Adam Bien (adam-bien.com)

Java EE 6 ist schlanker als POJOs (diese Behauptung wird ausführlich in diesem Workshop diskutiert), dennoch werden immer noch unnötige Patterns, Schichten und Indirektionen entworfen, dokumentiert und dann auch noch realisiert. In diesem Workshop werden wir gemeinsam leichtgewichtige Java-EE-6-Architekturen diskutieren und dabei eine End-to-end-Anwendung entwickeln.

Viele Wege führen zu Java EE

Jens Schumann (open knowledge GmbH)

Bei allem Sex Appeal einer Technologie stellt sich immer wieder die Frage, wie man diese im konkreten Projektumfeld einführen kann. Ausgehend vom idealen „Grüne Wiese“-Projekt stellt dieser Workshop Strategien vor, wie

bestehende Enterprise-Java-Anwendungen (EJB 2.x, Spring, Struts etc.) schrittweise zum Java-EE-6-Technologie-Stack migriert werden können. Neben unterschiedlichen Ansätzen für die technologischen Bausteine der Java-EE-6-Plattform erarbeitet dieser Workshop auch Kriterien, die eine Migration erschweren bzw. verhindern oder kaum sinnvoll erscheinen lassen.

Java EE Integration Architecture

Thilo Frotscher (Freiberufler)

Ein Service ist mit Java EE 6 und aktuellen IDEs im Handumdrehen erstellt. Jedoch ergeben sich darüber hinaus eine Vielzahl architektonischer Fragen. Wie strukturiert man die Implementierung eines Service am besten? Wie sollten eintreffende Nachrichten verarbeitet werden? Wann ist asynchrone Verarbeitung über Message Queues sinnvoll und wie lassen sich Systeme wirklich lose koppeln? In diesem Workshop werden Alternativen für eine Integrationsarchitektur diskutiert und typische Fallen aufgezeigt.

TRAINER



Adam Bien *adam-bien.com*

Dipl.-Inf. Adam Bien (Java Champion, EJB 3, JPA 2, Java EE 6 Expert Group Member) arbeitet als freiberuflicher Berater und Dozent im Enterprise-Java-Bereich. Seine praktische Erfahrung stammt aus seiner Mitarbeit in vielen Java- und Java-EE-Projekten. Er entwickelt bereits seit JDK 1.0 mit Java und hat die Bücher „Enterprise Java Frameworks“, „J2EE Patterns“, „J2EE HotSpots“, „Enterprise Architekturen“ und „Java EE 5 Architekturen“ sowie zahlreiche Fachartikel zur verteilten Java-Programmierung verfasst. Kontakt: blog.adam-bien.com.



Thilo Frotscher *Freiberufler*

Thilo Frotscher ist freiberuflicher Softwarearchitekt und Trainer. Als Experte für Enterprise Java, Web Services und XML unterstützt er seine Kunden in Projekten und mit Schulungen. Thilo ist (Ko-)Autor der Bücher „Java Web Services mit Apache Axis“, „Web Services mit Apache Axis2“ und „SOA-Expertenwissen“ sowie zahlreicher Artikel für verschiedene Fachzeitschriften. Darüber hinaus wirkt er als Fachgutachter für Publikationen im Bereich Web Services sowie XML und berichtet regelmäßig auf internationalen Fachkonferenzen über seine Erfahrungen (<http://www.frotscher.com>).



Arne Limburg *open knowledge GmbH*

Dipl.-Inf. Arne Limburg ist Enterprise-Developer bei der OpenKnowledge GmbH in Oldenburg. Er verfügt über langjährige Erfahrungen als Entwickler, Architekt und Trainer im Enterprise-Umfeld (EJB und Spring-Framework). In diesem Bereich führt er auch regelmäßige Workshops durch, u.a. zu den Themen Domain-driven Design, zum Java Persistence API und zum Testen. Darüber hinaus ist er im Open-Source-Bereich aktiv, unter anderem als Urheber und Projektleiter von JPA-Security.



Peter Roßbach *Freiberufler*

Peter Roßbach ist freiberuflicher Systemarchitekt und Coach zahlreicher Java-EE-Anwendungen. Sein Schwerpunkt liegt in der Entwicklung von komplexen Informationssystemen, einschließlich der Realisierung testgetriebener Softwareprozesse. Seit 1997 wirkt Peter im Bereich „Java Server und Servlets“ und veröffentlichte das gleichnamige Buch und zahlreiche Fachartikel. Mit dem Buch „Tomcat4x“ und als Autor der Tomcat@-Kolumne möchte er Sie für das Apache-Tomcat-Projekt begeistern. Er ist aktiver Entwickler des Tomcats und Mitglied der Apache Software Foundation.



Lars Röwekamp, *Program Chair open knowledge GmbH*

Lars Röwekamp, Gründer des IT-Beratungs- und Entwicklungsunternehmens OpenKnowledge GmbH, beschäftigt sich als „CIO New Technologies“ mit der Analyse und Bewertung neuer Software- und Technologietrends. Ein besonderer Schwerpunkt seiner Arbeit liegt derzeit auf Enterprise und Mobile Computing, wobei neben Design- und Architekturfragen insbesondere die Real-Life-Aspekte im Fokus seiner Betrachtung stehen. Er ist Autor vieler Fachartikel und -bücher und beschäftigt sich seit der Geburtsstunde von Java mit dieser Programmiersprache.



Jens Schumann *open knowledge GmbH*

Jens Schumann, Softwarearchitekt bei dem IT-Beratungs- und Entwicklungsunternehmen OpenKnowledge GmbH sowie freier Autor, beschäftigt sich seit vielen Jahren mit dem Design und der Umsetzung komplexer Lösungen im Enterprise-Computing-Umfeld. Seine praktische Erfahrung sammelte Jens Schumann in diversen großen, internationalen Projekten. Er ist regelmäßiger Sprecher auf Fachkonferenzen und engagiert sich in diversen Open-Source-Projekten. Seine Schwerpunkte bilden derzeit vornehmlich die Aspekte des serverseitigen Java.

PREISE

	Early Bird (inkl. Rabatt ab 3 Kollegen)	Early Bird (bis 28. Mai)	Standard (inkl. Rabatt ab 3 Kollegen)	Standard
Eintages-Pass	549 €	599 €	599 €	699 €
Zweitäges-Pass	849 €	949 €	949 €	1049 €
Dreitäges-Pass	1199 €	1299 €	1299 €	1399 €

alle Preise zzgl. ges. MwSt.



Haben Sie Fragen zum Java EE Summit? Gerne beraten wir Sie persönlich!

Kontakt: Telefon: +49 (0)331 282-2225

E-Mail: info@java-ee-summit.de