Konfigurieren statt Programmieren: Data Driven Java Development

Thomas Haskes
Triestram & Partner GmbH
Bochum

Schlüsselworte

rapid.Java, Data Driven Java Development, UI-Konfiguration, Eclipse, SWT, Oracle, Datenbank

Einleitung

In Java UI Frameworks wie SWT oder Swing, aber auch in Forms werden die Eigenschaften eines jeden UI-Elements als Properties definiert. So kann z.B. die Anzahl Zeichen, die in ein Textfeld eingegeben werden dürfen, mit Hilfe eine Setter-Methode festgelegt werden. Das hat jedoch bei einer Änderung dieser Eigenschaft zur Folge, dass Code neu kompiliert und die Anwendung neu ausgeliefert (deployed) werden muss.

Legt man diese Eigenschaften jedoch nicht im Code, sondern in der Datenbank ab, entsteht eine Anwendung, die leicht an verschiedene Benutzer-, Projekt- oder Produktanforderungen angepasst werden kann, ohne dafür den Sourcecode ändern zu müssen. So beschränkt sich das Ändern der oben genannten Eigenschaft auf das Ändern eines Wertes in der Datenbank.

Natürlich sind die meisten Anforderungen viel komplexer als das genannte Beispiel:

- Bereits in der Anwendung abgebildete Geschäftsfälle ändern sich kurzfristig, sodass Validierungen angepasst oder zusätzliche Daten erfasst bzw. angezeigt werden müssen.
- Je Anwendergruppe werden verschiedene Sichten auf die verfügbaren Daten benötigt.
- Wird die Organisationsstruktur des Unternehmens geändert, ändern sich häufig auch die Berechtigungen bzw. Zuständigkeiten der Anwender.

Diese und viele weitere Anforderungsbeispiele verdeutlichen die Notwendigkeit, eine Anwendung in möglichst vielen Aspekten konfigurierbar zu halten. Andererseits erhöht sich mit steigender Flexibilität das Datenaufkommen für die Konfiguration in der Datenbank.

Der Vortrag zeigt am Beispiel des rapid.Java Frameworks von T&P, wie vielfältig die Möglichkeiten der Konfiguration einer Anwendung sein können und stellt Konzepte vor, mit dem erhöhten Datenaufkommen in der Oracle Datenbank und den damit verbundenen Herausforderungen sinnvoll umzugehen.

Ein Beispiel

Angenommen, für die Eingabe von Daten in ein Formular soll ein Textfeld verwendet werden, dass einen grauen Hintergrund hat und dessen Eingabelänge auf 200 Zeichen begrenzt sein soll. Als Schriftfarbe soll rot verwendet werden.

Programmieren

Wird ein solches Textfeld programmatisch - z.B. in Eclipse SWT - erzeugt, werden die oben genannten Einstellungen im Programmcode am erzeugten Textfeld gesetzt:

```
Text inputText = new Text(parent, SWT.NONE);
inputText.setBackground(new Color(Display.getCurrent(), 120, 120, 120));
inputText.setForeground(new Color(Display.getCurrent(), 255, 0, 0));
inputText.setTextLimit(200);
```

Soll nun die Eingabelänge des Textfeldes verändert werden, muss die letzte Zeile des Codes in obigem Listing geändert werden. Das hat zur Folge, dass der geänderte Quellcode neu kompiliert und ausgeliefert werden muss – ziemlich viel Aufwand für eine solch kleine Änderung. Dabei wird der eigentliche Sinn des Quellcodes gar nicht geändert, lediglich der an das Textfeld übergebene Wert für das Textlimit.
An diesem einfachen Beispiel zeigt sich schon, dass Konfigurationsdaten im Programmcode nicht gut aufgehoben sind. In einem realistischeren Anwendungsfall sind die Konfigurationsdaten für ein Textfeld zudem oft noch wesentlich umfangreicher. So werden über ein Textfeld häufig nicht nur Daten eingegeben, sondern auch bereits eingegebene Daten angezeigt, sodass das erzeugte Textfeld mit dem in der Anwendung verwendeten Datenmodell verknüpft werden muss („Databinding“). Handelt sich um ein Pflichtfeld, muss geprüft werden, dass das Textfeld nicht leer ist, usw. Die Liste der denkbaren Konfigurationen ist lang, allein die SWT API für das Text Control enthält 38 Setter Methoden. Schließlich ist das Textfeld nur eines von vielen verfügbaren Widgets, die für die Erstellung eines Formulars eingesetzt werden können, womit die Anzahl der Konfigurationsdaten, die sich im Code befinden, wenn alle Einstellungen lediglich programmatisch vorgenommen werden, weiter wächst.

Konfigurieren

Angenommen, die zu verwendenden Konfigurationsdaten für das obige Beispiel, also die Hintergrundfarbe, die Schriftfarbe und die Eingabelänge, wären in einer Datenbanktabelle in folgender Form abgelegt:

Ui Element ID	Eigenschaft	Wert
inputTextId	BackgroundColor	120,120,120
inputTextId	ForegroundColor	255,0,0
inputTextId	TextLimit	200

Die Spalte UI Element ID enthält dabei den Schlüssel, anhand dessen die Einstellungen für das beispielhafte Textfeld zu finden sind, die Spalte Eigenschaft beschreibt, auf welche Eigenschaft des Textfeldes sich das Datum in der Spalte Wert bezieht.

Nehmen wir weiter an, es existiere eine Framework.-Klasse, die die Einstellungen für das Textfeld aus der Datenbank auslesen und an einer übergebenen Referenz auf das Textfeld setzen kann. Da die Klasse diese Funktionalität für viele in SWT verfügbaren Widgets bereithält, nennen wir sie „SWTWidgetToolkit“. Der Anwendungscode für die Erstellung eines Textfeldes mit den beschriebenen Eigenschaften könnte dann so aussehen:

```
SWTWidgetToolkit toolkit = new SWTWidgetToolkit();
Text inputText = new Text(parent, SWT.NONE);
toolkit.adapt(inputText, "inputTextId");
```

Die folgende Abbildung veranschaulicht den Vorgang.

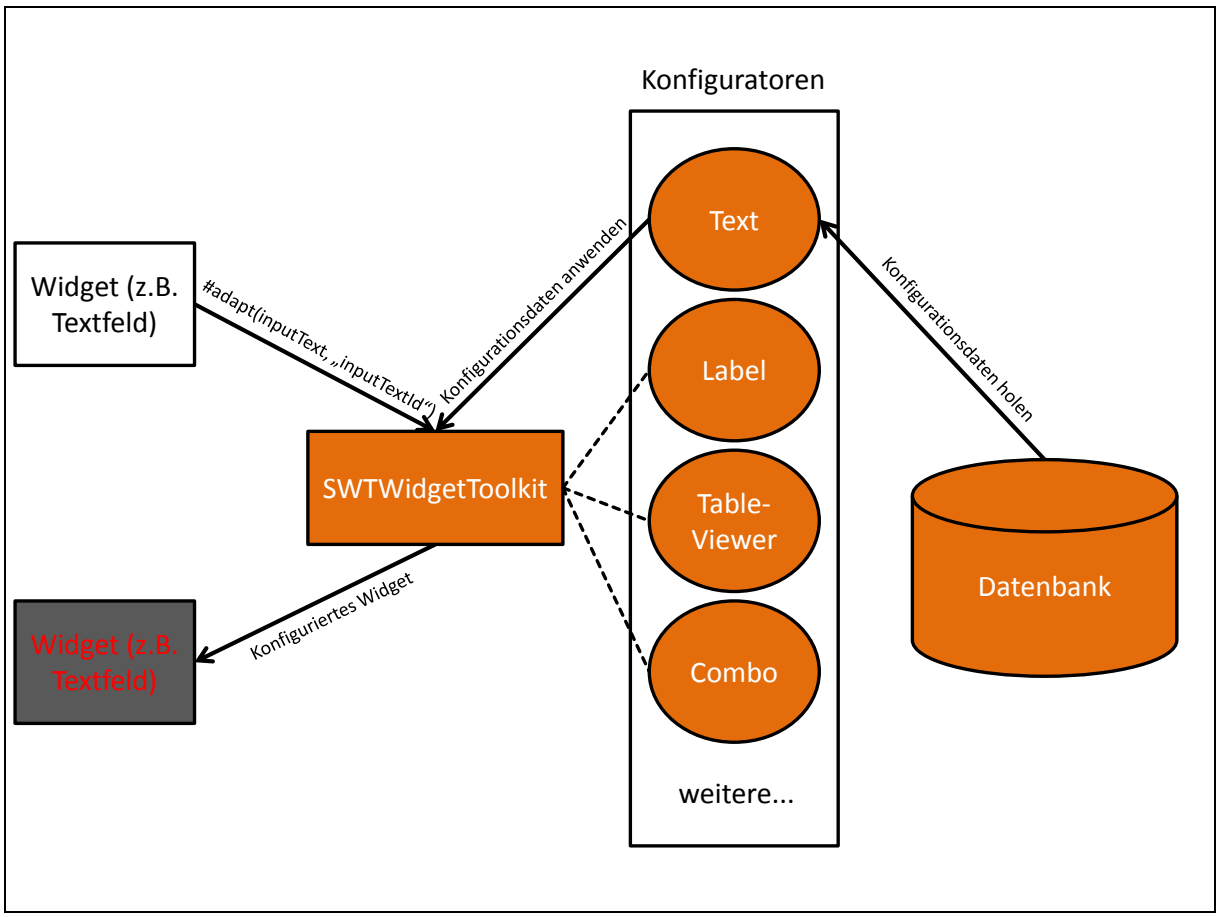

Abb. 1: Konfigurationsvorgang mit Hilfe von Konfigurator-Klassen und Daten aus der Datenbank

Dieses Vorgehen beschreibt im Wesentlichen das Konzept, nach dem im rapid.Java Framework von T&P Konfigurationsdaten verwaltet werden. Für nahezu jedes Widget, das die Eclipse SWT Bibliothek anbietet, steht im rapid.Java Framework ein Konfigurator zur Verfügung, der durch das SWTWidgetToolkit verwendet werden kann. Auf diese Art und Weise kann ein Großteil der Konfigurationsdaten in die Datenbank ausgelagert, und somit auch noch nachträglich verändert werden, ohne dafür den Anwendungscode ändern zu müssen.

Durch dieses Vorgehen wird ein hohes Maß an Flexibilität erreicht, mit Hilfe dessen eine Anwendung, die auf rapid.Java basiert, sehr gut an sich ändernde Bedingungen angepasst werden kann.

Widget vs. UI-Element

Die scheinbar synonymen Bergiffe „Widget“ und „UI-Element“ werden im Folgenden so verwendet:

- Wird der Begriff „Widget“ verwendet, so ist damit ein Element in der Oberfläche der Anwendung gemeint, also z.B. ein Textfeld, Combobox oder Tabelle.

- Der Begriff „UI-Element“ bezeichnet einen Satz von Konfigurationsdaten in der Datenbank, der einem bestimmten Widget über seine ID zugeordnet ist.

Datenaufkommen

Weiter oben wurde bereits erwähnt, dass für jede Art von Widget eine Vielzahl von Eigenschaften gesetzt werden muss. Nicht selten besteht eine Anwendung aus vielen tausend Widgets, die allesamt konfiguriert werden müssen. Befinden sich alle diese Konfigurationsdaten in der Datenbank, ergibt sich hieraus ein beträchtliches Datenaufkommen. Hier bedarf es weiterer Konzepte, die zum Einen die Organisation der Konfigurationsdaten betreffen, um deren Pflege zu erleichtern und die Übersicht zu behalten, zum Anderen deren Verarbeitung, sodass sich durch das erhöhte Datenaufkommen kein Performance-Nachteil ergibt.

Organisation

Im obigen Beispiel wurde die zum Textfeld gehörigen Konfigurationsdaten lediglich mit einer Id versehen und zusammen mit Eigenschaft und Wert in einer einfachen Tabelle abgelegt. Sobald Konfigurationsdaten vieler tausender UI-Elemente verwaltet werden müssen, ist dieses Vorgehen nicht sinnvoll. Oft stehen UI-Elemente in einer Beziehung zueinander, die sich aus deren Verwendung für den jeweilig umgesetzten Anwendungsfall ergibt. So befinden sich meist Gruppen von Widgets auf einem Karteireiter oder auf einem Dialog. In rapid.Java sind die Dialoge nach Anwendungsfällen, sogenannten Perspektiven organisiert; für den Anwendungsfall „Kundenverwaltung“ existiert also eine gleichnamige Perspektive. Diese Perspektiven sind wiederrum in sogenannte „Views“ unterteilt.

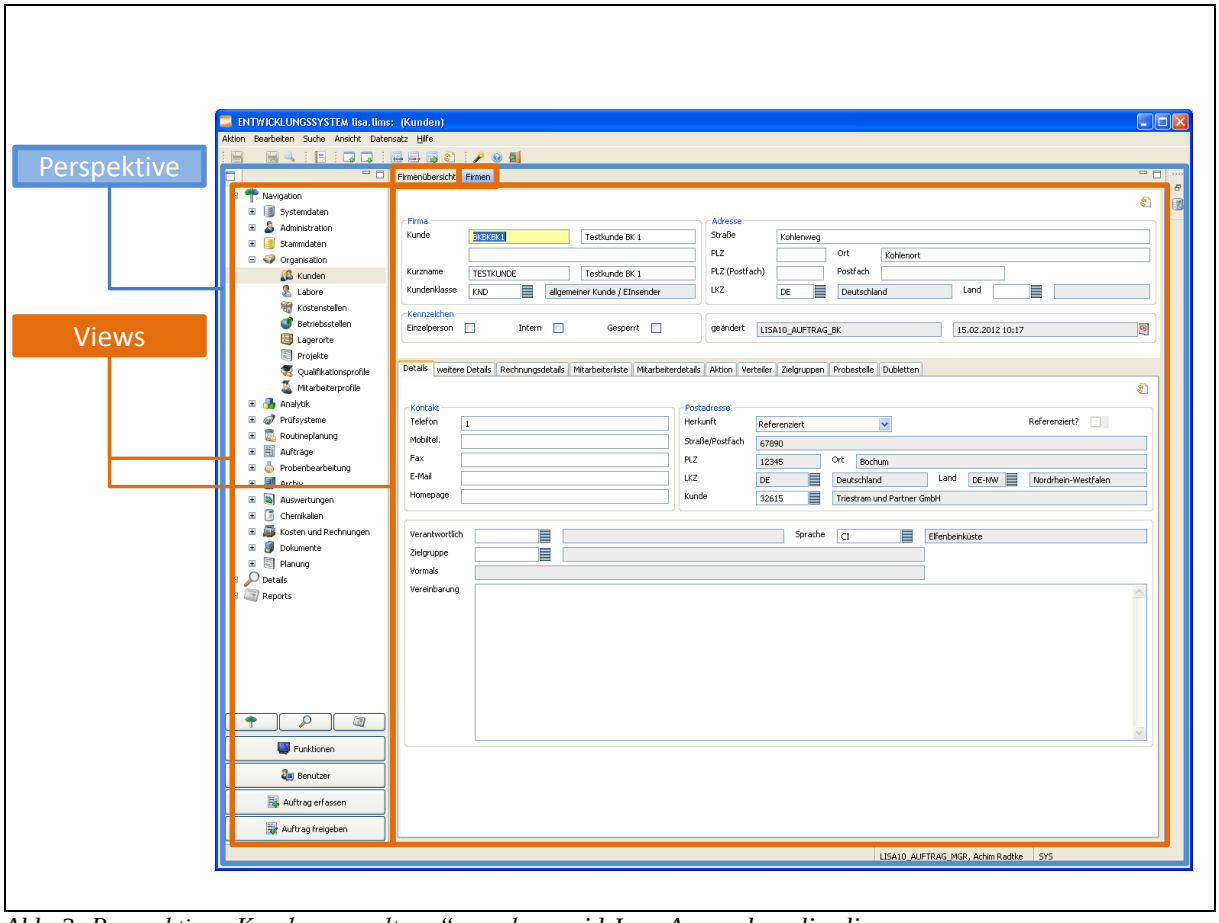

Abb. 2: Perspektive „Kundenverwaltung“ aus der rapid.Java Anwendung lisa.lims

Da die Widgets in rapid.Java Anwendungen bereits nach Views gruppiert sind, stellt die im Eclipse RCP System verwendete View-Id ein gutes Kriterium zur Gruppierung der in der Datenbank gespeicherten UI-Elemente dar. Aus diesem Grund haben die Konfigurationsdaten eines jeden UI-Elements in rapid.Java einen zweiwertigen Schlüssel, der einerseits aus der View-Id besteht, auf dem sich das zugehörige Widget befindet, andererseits einen sprechenden Element-Namen enthält. Mit Hilfe von Namenskonventionen bei der Benennung der Views und der UI-Elemente können die Elemente weiter gruppiert und damit schnell in der Datenbank wiedergefunden werden.

Propertysets

Nicht nur die UI-Elemente selbst, sondern auch deren Eigenschaften lassen sich nach bestimmten Kriterien gruppieren. Werden mehrere Eigenschaften zu einem Satz an Eigenschaften zusammengefasst und dieser Satz mit einem Namen versehen, lassen sich diese Eigenschaften wiederverwenden, indem man den UI-Elementen ihre Eigenschaften nicht direkt, sondern durch die Zuordnung eines Eigenschaftensatzes (Propertyset) zuweist. Eine mögliche Gruppierung von Eigenschaften ergibt sich etwa durch den Typ des Widgets. So ergibt die Zuordnung von Tabellenspaltennamen zu einem Textfeld keinen Sinn, da ein Textfeld keine Tabellenspalten hat. Ebenso ist die Eigenschaft „TextLimit“ bei einer Checkbox sinnlos.

Um eine erste Gruppierung von Eigenschaften vorzunehmen, bietet sich also der Typ eines Widgets an. In einem solchen „Default“-Propertyset lassen sich dann alle Eigenschaften zusammenfassen, die einem bestimmten Widget-Typen sinnvoll zugeordnet werden können.

Eine weitere Möglichkeit der Gruppierung von Eigenschaften ergibt sich, wenn der Einsatzzweck eines Widgets betrachtet wird. Wird z.B. die Anforderung formuliert, dass Auftragsnummern in einem Informationssystem immer in Fettschrift darzustellen sowie immer 10 Zeichen lang sind – unabhängig davon, ob die Auftragsnummer in einer Tabelle, einem Textfeld oder einer Combobox angezeigt wird – ergibt sich für den Kontext „Auftragsnummer“ ein Propertyset mit den Eigenschaften „Fettschrift“ und Textlimit 10. Hierfür lässt sich ein Propertyset definieren, dass – diesmal unabhängig vom Typ des Widgets – alle möglichen Einstellungen enthält, die zur Definition einer fettgedruckten Schrift gebraucht werden und die Eingabelänge auf 10 Zeichen begrenzt. Diese Einstellungen können dann bei der Definition von UI-Elementen, die eine Auftragsnummer darstellen, komfortabel wiederverwendet werden.

Schließlich gibt es Konfigurationen, die einzigartig für ein bestimmtes Widget sind, und nur für dieses spezielle Widget gelten sollen. Propertysets einer solchen Art könnte man als „lokale“ Propertysets (ähnlich einer lokalen Variable im Quellcode) bezeichnen. Soll z.B. ein bestimmtes Widget, dass eine Auftragsnummer anzeigt, vom Kontext „Auftragsnummer“ abweichen und eine gelbe Hintergrundfarbe haben, so kann diese Eigenschaft in einem lokalen Propertyset definiert werden, welches dann lediglich diesem einen, besonderen UI-Element zugewiesen wird.

Diese drei Gruppierungen werden im rapid.Java Framework von T&P für die Zuordnung von Eigenschaften zu UI-Elementen verwendet. Jedem UI-Element werden drei verschiedene Propertysets zugeordnet. Die Eigenschaften aller zugeordneten Propertysets werden dann bei der Konfiguration eines Widgets durch einen Konfigurator zusammengeführt. Wird eine Eigenschaft durch mehrere Propertysets definiert, die dem UI-Element zugeordnet sind, wird der konfigurierte Wert hierarchisch überschrieben. Dabei überschreibt eine im Kontext Propertyset gesetzte Eigenschaft diejenige aus dem Default Propertyset, während eine Eigenschaft aus einem lokalen Propertyset wiederrum Eigenschaften aus dem Kontext Propertyset überschreibt. Alle übrigen Eigenschaften werden zusammengeführt. Unten stehende Abbildung veranschaulicht das Vorgehen.

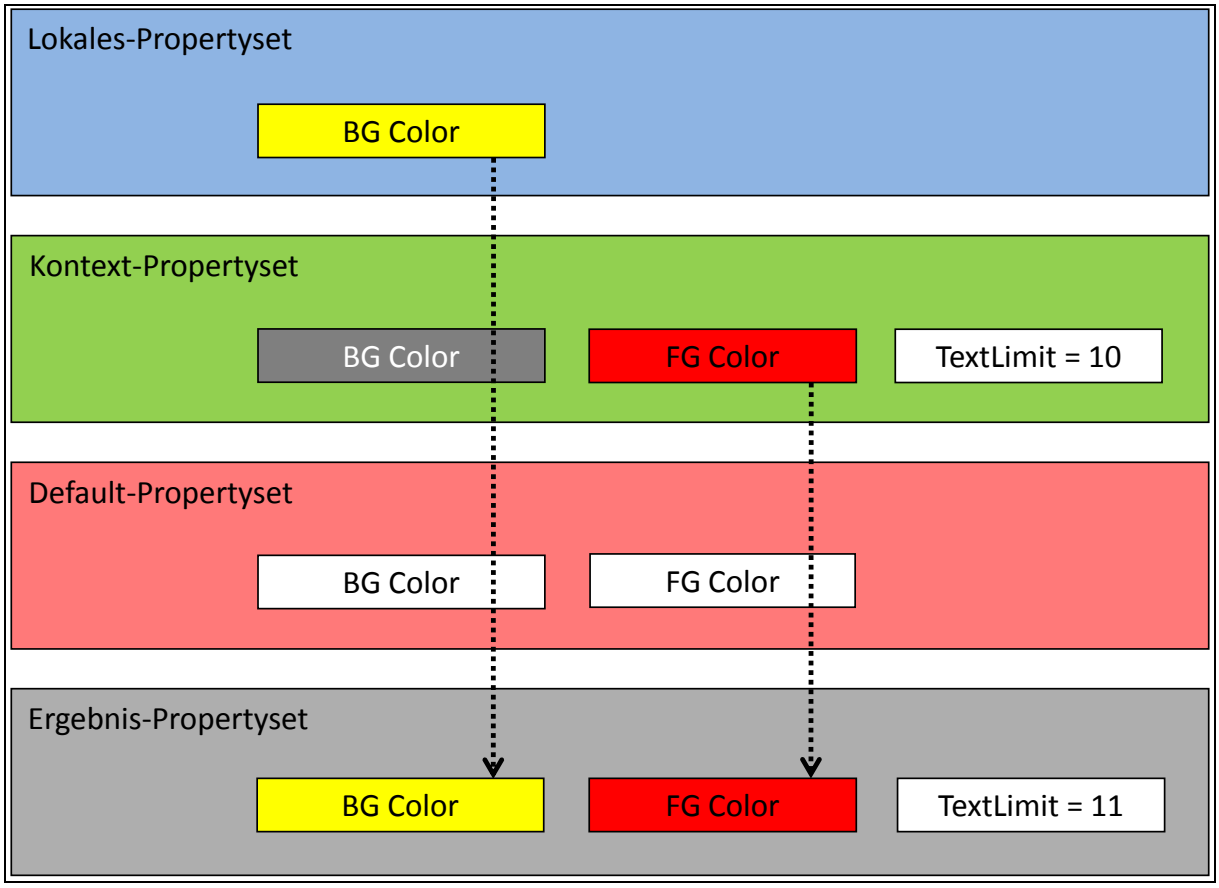

Abb. 3: Hierarchisches Zusammenführen von Eigenschaftswerten

Durch diese Art der Verwaltung von Eigenschaftswerten können einerseits häufig verwendete Eigenschaftswerte wiederverendet werden und müssen nicht für jedes neue UI-Element mehrfach gespeichert werden. Außerdem wird dadurch die Pflege der Eigenschaften erleichtert. Wird z.B. festgelegt, dass fortan alle Textfelder einen blauen Hintergrund haben sollen, genügt die Änderung der Eigenschaft im Default Propertyset. Jene Widgets, für die über das Kontext Propertyset bewusst eine andere Hintergrundfarbe definiert wurde, bleiben von dieser Änderung unberührt.

Performance

Durch den Umstand, dass bei der Erzeugung eines Widgets dessen Konfigurationsdaten aus der Datenbank abgerufen werden müssen, werden bei Verwendung einer solchen Anwendung vermehrt Daten aus der Datenbank abgefragt. Um die Konfigurationsdaten nicht unnötig oft aus der Datenbank anzurufen, bedarf es u.U. weiterer Konzepte wie Lazy-Loading, Caching und Komprimierung. Diese Konzepte werden im Vortrag vorgestellt.

Kontaktadresse:

Thomas Haskes

Triestram & Partner GmbH

Kohlenstraße 55

D-44795 Bochum

Telefon:	+49 (0) 234-94375-0
Fax:	+49 (0) 234-452206
E-Mail	t.haskes@t-p.com
Internet:	www.t-p.com