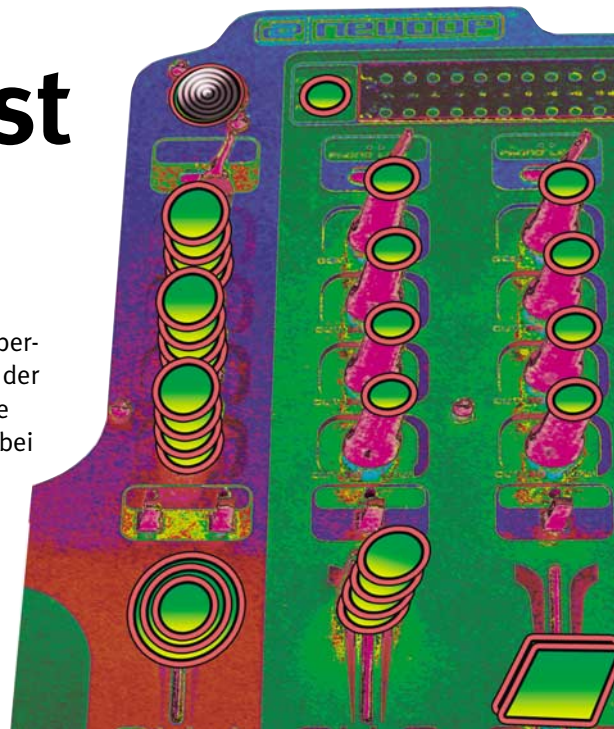


Programmierung von Rich Thin Clients mit der Thinlet-Technologie

von Marc Teufel

Dünn geschlagen ist bald geschliffen

Bei Thinlet handelt es sich um ein Toolkit zur Programmierung von Benutzeroberflächen. Die Kernkomponente verwaltet hierbei Hierarchie und Eigenschaften der Benutzeroberfläche, reagiert auf Eingaben bzw. Events und ruft schließlich die eigentliche Business-Logik auf. Die Definition der Benutzeroberfläche erfolgt bei einer Thinlet-Applikation in einer XML-Datei, die Programmierung der Business-Logik dagegen in einer Java-Klasse. Anhand eines Beispielprogramms erfolgt mit diesem Artikel ein erster Kontakt mit der Thinlet-Technologie, außerdem werden alle wichtigen Konzepte, die hinter Thinlet stehen, aufgegriffen und näher erläutert.



Kaum ein neues, größeres Projekt, das heutzutage in Java programmiert wird, folgt noch dem klassischen Client/Server-Prinzip. Mindestens dreischichtig muss es sein, da man aber für die Zukunft gewappnet sein will und von den Vorteilen der Web-schicht einer 4-Tier-Architektur profitieren möchte, realisiert man neue Projekte heute meist als Webapplikation. Als Benutzeroberfläche fungiert hierbei oft ein Thin Client. Vorteil und gleichzeitig Nachteil von Thin Clients ist, dass auf dem Rechner des Anwenders keine zusätzliche Software installiert werden muss, da die Oberfläche in reinem HTML dargestellt wird. Das größte Manko solcher Clients ist hier vor allem in den eingeschränkten Oberflächenelementen zu sehen. Die Programmierung von komplexeren Baumstrukturen oder sortierbaren Tabellen kann mit HTML durchaus etwas mühsam werden.

Dieses Problem vor Augen, machten sich schlaue Softwarearchitekten Gedanken und kreierten schließlich die so ge-

nannten Rich Thin Clients, in dem sie die HTML-Oberfläche entweder durch ein spezielles Java-Applet ersetzen oder sehr stark mit clientseitigem JavaScript erweitern. Klassische Vertreter solcher Rich-Thin-Client-Architekturen sind Canoo ULC, Oracle Forms (ab Version 6i, aktuell 10g), Casabac GUI Server (kein Applet auf dem Client, baut auf (D)HTML und JavaScript

auf) und natürlich Thinlet. Die Thinlet-Technologie hebt sich in vielerlei Hinsicht von der Konkurrenz ab. Hervorzuheben ist hier vor allem die XML-getriebene Definition von Benutzeroberflächen, was Thinlet zu einem Vertreter der neuen XUL-Tools macht (siehe Kasten und Artikel ab Seite 46). Ein weiterer Vorteil ist der bewusste Verzicht auf Swing zugunsten von

Architekturen zur Realisierung von Benutzeroberflächen

- Thin Client: Schlanke Webanwendung basierend auf HTML, DHTML und JavaScript. Zur Ausführung genügt ein einfacher Webbrowser, im Normalfall keine Zusatzinstallation erforderlich. Kompliziertere Oberflächenelemente wie z.B. Bäume, interaktive Tabellen oder Benutzervalidierung sind unter Umständen etwas aufwendiger zu realisieren.
- Fat Client: Desktopapplikation, die auf dem Client installiert werden muss. Früher meist in Form einer Client/Server-Architektur verwendet, heutige Fat Clients folgen oft einer dreischichtigen Architektur und agieren entweder über RMI oder Web Services mit der Geschäftslogik auf dem Application Server.
- Rich Thin Client: Mix aus Thin und Fat Client, d.h. Fat Client (meist spezielles Java-Applet

mit serverseitiger Oberflächensteuerung. Oft eine Implementierung des HOPP Pattern (Half-Object Plus Protocol) mit Server-Side Swing und entsprechenden Proxies sowohl im Client als auch im Server, die miteinander kommunizieren. Das clientseitige Applet fungiert hier oft als Übermittler und delegiert Events, wie beispielsweise das Klicken auf einen Button, an den Server, auf dem schließlich die Verarbeitung erfolgt. Canoo ULC ist zum Beispiel ein Vertreter einer solchen Technologie.

- Thinlet macht hier eine Ausnahme, hier findet die komplette Verarbeitung aller Events innerhalb des Applet auf dem Client statt (wobei auch hier die Möglichkeit besteht, auf Business-Logik im Application Server zuzugreifen).



Quellcode auf CD!

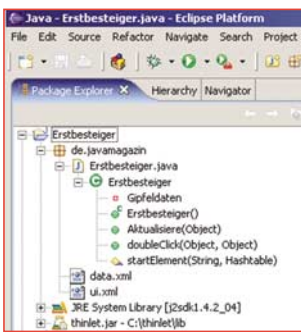


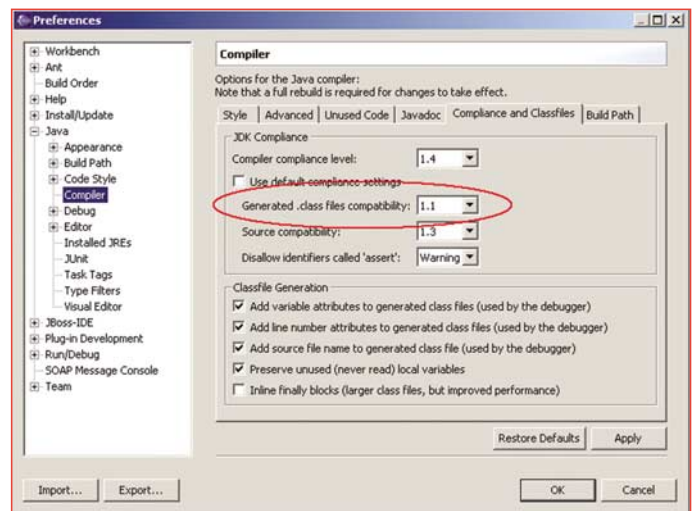
Abb. 1: Die Thinlet-Applikation *Erstbesteiger* als Java-Projekt in Eclipse 3.0

AWT im clientseitigen Applet. Damit können Thinlet-Applikationen in Webbrowsern, die mindestens Java 1.1 unterstützen, auch ohne Java-Plug-in problemlos betrieben werden. Zu guter Letzt bleibt noch zu erwähnen, dass es sich bei Thinlet um ein Open-Source-Projekt handelt.

Gipfelstürmer

Zum Einstieg sind an dieser Stelle alle Leser herzlich eingeladen, das folgende Beispielprogramm in der Praxis nachzuvollziehen. Auf diese Weise erfolgt ein erster Kontakt mit der interessanten Thinlet-Technologie, außerdem werden alle wichtigen Konzepte, die hinter Thinlet stehen, aufgegriffen und im weiteren Verlauf des Artikels näher erläutert. Als Beispielprogramm soll ein Frontend zur Anzeige von Gipfelerstbestigungen entstehen. In einer echten, produktiven Umgebung würden die Daten wahrscheinlich aus einer Datenbank kommen, da in diesem Artikel allerdings die Programmierung der Oberfläche im Vordergrund

Abb. 2: Java 1.1-Kompatibilität für *.class*-Files unter Eclipse einstellen



steht, beschränkt sich das Beispiel auf eine XML-Datei als Datenquelle.

Zu Beginn nimmt man sich zunächst die aktuelle Thinlet-Distribution [1] und sorgt dafür, dass die Datei *thinlet.jar* (im Unterverzeichnis *lib*) in den Klassenpfad der Java-Umgebung aufgenommen wird. Arbeitet man mit Eclipse, so ist zunächst ein neues Java-Projekt mit dem Namen *Erstbesteiger* anzulegen und die Bibliothek *thinlet.jar* in den *Build Path* aufzunehmen (Abb. 1).

Tipp: Um eine Thinlet-Applikationen ohne Java Plug-in lauffähig zu machen, muss der Quellcode mit Java 1.1-Kompatibilität kompiliert werden. Unter Eclipse stellt man dies in den Compiler-Einstellungen der Java-Umgebung unter WINDOW | PREFERENCES ein (Abb. 2). Auf der Kommandozeile übergibt man dem Compiler *-target 1.1* als zusätzlichen Parameter.

Nachdem das Projekt angelegt wurde, ist im nächsten Schritt die Benutzeroberfläche zu definieren. Da es sich bei Thin-

Anzeige

XUL und XAML

- XUL ist die Abkürzung für XML User Interface Language und stellt eine auf XML basierende Sprache dar, mit der grafische Benutzeroberflächen definiert werden können. Ursprünglich für das Mozilla-Projekt entwickelt, kommt XUL inzwischen in mehreren Open-Source-Projekten wie beispielsweise Thinlet zum Einsatz. Vorteile von XUL: Layout und Implementierung einer Benutzeroberfläche sind getrennt, XML-konforme Beschreibungssprache und entschlackter, sauberer Quellcode.
- XAML steht für eXtensible Application Markup Language und ist Microsofts Konzept zur Beschreibung von Benutzeroberflächen über XML. Kommende Windows- und Visual Studio .NET-Versionen werden XAML unterstützen.

Anzeige



Abb. 3: Die Oberfläche in Aktion, nachdem Thinlet die XUL-Definition geparkt hat

let um ein XUL-Werkzeug handelt, erfolgt diese Definition in Form von XML. Thinlet-Oberflächen sind durch ihren XML-Charakter hierarchisch aufgebaut. Das Wurzelement ist dabei immer ein Desktop-Element. Auf einem Desktop liegen entweder Panel- oder Dialog-Container, die wiederum die eigentlichen GUI-Komponenten (Widgets) aufnehmen. Panels nehmen immer den komplett verfügbaren Platz ein, Dialog-Container dagegen werden als eigenständige Fenster dargestellt (ähnlich Internal Frames in Swing). Die XUL-Oberflächendefinition für die Erstbesteiger-Applikation ist in Listing 1 ersichtlich. In Abbildung 3 kann man sehen, was Thinlet schließlich aus dieser XUL-Definition macht: Ein Fenster mit zwei Buttons, einer Table und einem Listenfeld. Zieht man das Fenster auseinander, werden die Widgets automatisch an die neue Fenstergröße angepasst. Möglich wird dies durch das hier zum Einsatz kommende Panel, welches einen Layoutmanager ähnlich dem *GridBagLayout* verwendet. Durch das Attribut *columns* wird die Anzahl der Spalten definiert, jedes Widget, das sich nun innerhalb dieses Panels befindet, wird der Reihe nach in einer separaten Spalte untergebracht. Betrachtet man Listing 1 etwas genauer, wird man feststellen, dass für jedes Widget – welches Bestandteil der Oberfläche werden soll – ein eigenes Tag existiert.

Verfügbare Oberflächenelemente

Thinlet baut zwar auf AWT auf, bietet aber durch eigene Implementierung dennoch eine beeindruckende Auswahl komplexer Oberflächenelemente (Widgets) an. Folgende Widgets stehen dem Entwickler zur Verfügung: *Label*, *Button*, *Checkbox*, *ToggleButton*, *ComboBox*, *TextField*, *PasswordField*, *TextArea*, *TabbedPane*, *SpinBox*, *ProgressBar*, *Slider*, *SplitPane*, *List*, *Table*, *Tree*, *Separator* und *MenuBar*.

tiert. Konfiguriert werden diese Widget-Komponenten letztendlich durch Attribute. So steuert beispielsweise das Attribut *weightx* bei einem Button, ob dieser auf Größenveränderungen des Panels bzw. Desktops reagiert oder ob er in der Breite immer fix bleibt.

Es wächst zusammen ...

Wie bringt man nun XUL-Definition und Anwendungslogik zusammen? Listing 2 zeigt, wie das funktioniert. Alles beginnt mit der Erzeugung eines Objekts *Erstbesteiger* in der *Main*-Methode. Im Konstruktor der *Erstbesteiger*-Klasse wird daraufhin die Oberfläche erzeugt. Dazu wird über die Methode *parse* (vererbt von *Thinlet*) die XML-Datei mit der Oberflächenbeschreibung eingelesen und eine Datenstruktur erzeugt, welche die Maske im Speicher darstellt. Diese Struktur (vom Typ *Object*) wird schließlich über die *add*-Methode zur Anzeige gebracht. Programmiert man mit *Thinlet*, kann die eigene Programmentwicklung wahlweise als allein stehende Desktopanwendung (Abb. 3) oder als Applet im Web (Abb. 4) zur Verfügung gestellt werden. Hierzu bringt *Thinlet* entsprechende Launcher-Klassen mit. In Listing 2 wird in der *Main*-Methode das Programm über einen *FrameLauncher* als Desktop-Applikation gestartet. Um das Ganze als Applet zu starten, würde man sich eines *AppletLauncher* bedienen, hierzu später mehr.

... was zusammengehört

Und wie reagiert das Programm nun auf Benutzereingaben? Hier lohnt es sich, noch einmal kurz einen Blick auf die XUL-Definition der Oberfläche in Listing 1 zu werfen. Bei der Definition des Buttons zur Aktualisierung der Gipfelliste wird über das Attribut *action* festgelegt, welche Methode im Falle eines Klicks ausgeführt werden soll. Listing 2 enthält demnach auch eine öffentliche Methode *Aktualisiere*, die von *Thinlet* immer dann aufgerufen wird, wenn auf den besagten Button geklickt wird. Gleiches gilt für die Table mit der Gipfelliste. Über das Attribut *perform* wird die Methode definiert, die *Thinlet* bei einem Doppelklick in die Table aufzurufen hat. Analog hierzu könnte man auch über das ebenfalls bei Table Widgets verfügbare At-

tribut *action* eine Methode ansteuern, die immer dann aufgerufen wird, wenn der Benutzer eine Zeile in der Tabelle selektiert.

Von Eventhandlern und Properties

Die XML-Ressource und die Java-Klasse mit der Implementierung werden wie bereits erwähnt über die *parse*-Methode verbunden. Die Steuerung sämtlicher Events übernimmt hierbei die eigene Java-Klasse, die von *Thinlet* abgeleitet ist. Jedes Widget auf einer Oberfläche kann verschiedene Events auslösen. Über spezielle Attribute, sog. Listener, werden diese Events in der XML-Definition der Oberfläche mit öffentlichen Methoden in der Java-Klasse verbunden. So gibt es bei fast allen Widgets einen Listener namens *action*. Komplexere Widgets, die mehr als ein Event auslösen können, besitzen unter Umständen mehrere Listener. In der XML-Datei kann beim Festlegen der aufzurufenden Methode in den Listener-Attributen außerdem optional ein oder mehrere Parameter zur Übergabe an die Methode in der Java-Klasse definiert werden. Durch die Verwendung von *this* als Parameter wird das aktuelle Widget (vom Typ *Object*) an die Methode übergeben, alternativ kann auch ein Widget durch seinen Namen referenziert und als Parameter an eine Methode übergeben wer-

Listing 1

Definition der Oberfläche (XUL)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<panel columns="2" top="4" left="4" bottom="4"
       right="4" gap="4">
  <button text="Gipfelliste aktualisieren"
    action="Aktualisiere(tableGipfel)"/>
  <button text="Programm beenden" />
  <label text="Gipfelliste:" colspan="2" font="bold"/>
  <table name="tableGipfel" selection="single" colspan="2"
    perform="doubleClick(this, listErstbesteiger)">
    <header>
      <column text="Gipfel" width="100" />
      <column text="Höhe" width="50" />
      <column text="Land" />
    </header>
  </table>
  <label text="Erstbesteiger:" colspan="2" font="bold"/>
  <list name="listErstbesteiger" colspan="2" />
</panel>
```

titelthema

Rich Thin Clients mit Thinlet

den (ebenfalls vom Typ *Object*). In der Methode kann dann auf das übergebene Widget zugegriffen und Eigenschaften gezielt ausgelesen und manipuliert werden. Die Methode *doubleClick* in Listing 2 demonstriert dies: Der Methode (aufgerufen durch Verwendung des *perform* Listener im Falle eines Doppelklick auf die Table) werden das komplette Tabellenobjekt (*this*) und das Widget zugeführt, in dem später die Erstbesteiger dargestellt werden sollen (*list-Erstbesteiger*). Aus der Tabelle werden zunächst die angeklickte Zeile und danach der Wert der ersten Spalte in dieser Zeile er-

mittelt. Dieser Wert dient als Schlüssel, mit dem die Erstbesteigerdaten aus der Datenquelle gelesen und schließlich in das übergebene Listenelement eingetragen werden. Das Eintragen der Daten erfolgt durch gezieltes Manipulieren der Eigenschaften des Listenelements. Jedes Widget besitzt eine Fülle von Eigenschaften (oder auch Properties), die über Attribute in der XML-Oberflächenbeschreibung gesetzt und bei Bedarf über Code manipuliert werden können. Dabei gibt es verschiedene Typen von Eigenschaften, für die Thinlet jeweils eine Getter- und eine Setter-Methode zur Verfü-

Listing 2

Verschmelzung von XUL mit Code

```
package de.javamagazin;
import thinlet.*;
import java.util.Enumeration;
import java.util.Hashtable;

public class Erstbesteiger extends Thinlet {
    private Hashtable Gipfeldaten;

    public Erstbesteiger() {
        try {
            add(parse("ui.xml"));
        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Erstbesteiger eb = new Erstbesteiger();
        FrameLauncher frame = new FrameLauncher
            ("Erstbesteiger", eb, 275, 275);
    }

    protected void startElement(String name,
        Hashtable attributlist) {
        if (name.equals("Gipfel")) {
            Gipfeldaten.put(attributlist.get("Name"), attributlist);
        }
    }

    public void Aktualisiere(Object table) {
        try {
            Gipfeldaten = new Hashtable();
        }

        parseXML(getClass().getResourceAsStream("data.xml"));

        removeAll(table);
        Enumeration en = Gipfeldaten.keys();
        while (en.hasMoreElements()) {
            Object row = Thinlet.create("row");
            Hashtable Gipfel = (Hashtable) Gipfeldaten.get
                ((String)en.nextElement());

            Object cell_name = Thinlet.create("cell");
            setString(cell_name, "text", Gipfel.get
                ("Name").toString());
            add(row, cell_name);

            Object cell_hoehe = Thinlet.create("cell");
            setString(cell_hoehe, "text", Gipfel.get
                ("Hoehe").toString());
            add(row, cell_hoehe);

            Object cell_land = Thinlet.create("cell");
            setString(cell_land, "text", Gipfel.get
                ("Land").toString());
            add(row, cell_land);

            add(table, row);
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }

    public void doubleClick(Object table, Object liste) {
        removeAll(liste);

        Object row = getSelectedItem(table);
        Object col = getItem(row, 0);
        String Gipfelname = getString(col, "text");

        Hashtable Gipfel = (Hashtable) Gipfeldaten.
            get(Gipfelname);

        Object listitem = Thinlet.create("item");
        setString(listitem, "text",
            Gipfel.get("Besteiger1").toString());
        add(liste, listitem);
        listitem = Thinlet.create("item");
        setString(listitem, "text", Gipfel.get("Besteiger2").
            toString());
        add(liste, listitem);
    }
}
```

Anzeige



Abb. 4: Die Erstbesteiger-Applikation läuft als Applet innerhalb des Internet Explorer

gung stellt: *string, choice, boolean, integer, icon, method, component, property, font, color, keystroke* und *bean*. In der Methode *doubleClick* wird demnach mittels *setString* die Texteigenschaft der neuen Zeile eingestellt, bevor diese in die eigentliche Liste eingefügt wird. Eine tabellarische Darstellung aller verfügbaren Listener für das Eventhandling und Properties für jedes Widget kann bei Bedarf in der Thinlet-Dokumentation nachgesehen werden.

XML-Parser inklusive

Zur Verarbeitung von XML bringt Thinlet einen einfach zu bedienenden XML Parser mit, der sowohl DOM als auch SAX Parsing größtenteils beherrscht. Die Methode *Aktualisiere* in Listing 2 bedient sich hierzu des internen SAX Parser zum Einlesen der Gipfeldaten aus der XML-Quelle

Listing 3

data.xml – Gipfelstürmerdatenbank in XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Gipfelstuermer>
  <Gipfel Name="Mount Everest"
    Hoeh="8848m"
    Land="Nepal"
    Besteiger1="Hillary, Edmund"
    Besteiger2="Norgay, Tenzing" />
  <Gipfel Name="Ortler"
    Hoeh="3905m"
    Land="Italien"
    Besteiger1="Pichler, Josef" />
  <Gipfel Name="Zugspitze"
    Hoeh="2968m"
    Land="Deutschland"
    Besteiger1="Naus, Josef" />
</Gipfelstuermer>
```

(Listing 3). Durch Aufruf der *parseXML*-Methode beginnt der Parser die übergebene XML-Datei (aus einem *InputStream*) einzulesen. Bei jedem neuen Element auf das der SAX Parser stößt, wird die von Thinlet zur Verfügung gestellte Methode *startElement* aufgerufen, die durch Überschreiben an eigene Bedürfnisse angepasst werden kann. Im Listing reagiert die überschriebene Methode auf jedes XML-Element mit der Bezeichnung „Gipfel“ und speichert alle Attribute dieses Elements in einer internen *HashTable*, die im weiteren Verlauf der Methode *Aktualisiere* zum Füllen der Table mit der Gipfelliste verwendet wird, aber auch in *doubleClick* zur Anwendung kommt.

Deployment im Web

Wie bereits erwähnt, machen solche Rich Thin Clients vor allem dann Sinn, wenn sie im Kontext einer Webanwendung laufen, da so der komplette Installationsaufwand auf dem Client entfällt. Eine Thinlet-Applikation in Form eines Applet in eine Webseite zu integrieren, ist nicht schwer. Hierzu ist ein JAR File zu erzeugen, das die vollständige Applikation mit allen zugehörigen Ressourcen, wie kompilierten Klassen, XML-Definition der Oberfläche, Daten, Icons usw. enthält. Das eigentliche Deployment geschieht, indem man das JAR File zusammen mit der Thinlet-Bibliothek (*thinlet.jar*) und einer HTML-Seite in den Webordner ablegt. Die HTML-Seite ist für das Starten der Applikation als Applet über den bereits weiter oben erwähnten *AppletLauncher* verantwortlich.

```
<html>
<head>
  <title>Thinlet im Web</title>
</head>
<applet code="thinlet.AppletLauncher" archive=
  "thinlet.jar, erstbesteiger.jar"
  width="275" height="275">
  <param name="class" value="de.javamagazin.
  Erstbesteiger"></param>
</applet>
</html>
```

Wichtig ist an dieser Stelle zu verstehen, dass für ein Thinlet Deployment ein einfacher Webserver ausreicht. Auch wenn ein Deployment in Webcontainern wie Tomcat möglich ist und in Einzelfällen auch durchaus Sinn machen kann (zum Beispiel

wenn die Applikation mit Servlets kommuniziert), so ist ein Webcontainer zur Ausführung von webbasierten Thinlet-Applikationen nicht zwingend erforderlich.

Fazit

Mit Thinlet liegt eine leistungsfähige, universelle und einfach zu bedienende Bibliothek zur Erstellung von Rich Thin Clients vor. Durch den bewussten Einsatz von AWT anstelle von Swing sind Thinlet-Applikationen auch ohne Java-Plug-in lauffähig und lassen sich auf einer Vielzahl von JVMs ausführen. Die strikte Trennung von Applikationslogik und Oberfläche mit XML machen Thinlet zu einem echten XUL-Tool und einer zukunftsweisenden Technologie, mit der viele Entwickler (vor allem im Microsoft-Umfeld) in Zukunft konfrontiert werden dürften. Im Gegensatz zu anderen Rich-Thin-Client-Tools verwendet Thinlet nicht das HOPP-Protokoll, die Verarbeitung findet vollständig auf dem Client statt, ist aber nicht auf diesen beschränkt. Die vorbildliche Dokumentation und die guten Beispiele (hier ist vor allem die Drafts-Anwendung zu erwähnen) erleichtern die tägliche Arbeit mit Thinlet ungemein. Fragen, die nicht durch die Dokumentation oder in den Beispielen beantwortet werden können, finden in der dem Projekt zugehörigen, äußerst aktiven Mailing-List [3] meist eine Antwort. ■

Marc Teufel arbeitet als Softwareentwickler bei einem führenden deutschen Großhandelsunternehmen und beschäftigt sich hier vorwiegend mit der Programmierung von mehrschichtigen Anwendungen im Oracle- und J2EE-Umfeld.

Links & Literatur

- [1] www.thinlet.com
- [2] Thinlet auf Sourceforge: sourceforge.net/projects/thinlet/
- [3] groups.yahoo.com/group/thinlet/
- [4] Offizieller Thinlet Blog mit Tutorials: thinlet.blog-city.com
- [5] Tool zur grafischen Erzeugung von Thinlet-Oberflächen: www.carlsbadcubes.com/theodore/
- [6] Canoo ULC: www.canoo.com/ulc/
- [7] Oracle Forms: www.oracle.com/technology/products/forms/
- [8] Casabac GUI Server: www.casabac.com
- [9] XUL Project: www.mozilla.org/projects/xul/
- [10] XAML: www.xaml.net