Quelle: SBB

RCP im täglichen Einsatz bei den Schweizerischen Bundesbahnen

Eclipse bei der Eisenbahn

>> DANIEL MEIER, HUGO GRAF, MARC HOFFMANN, SIMON EGGLER UND GILLES IACHELINI

Die Schweizerischen Bundesbahnen (SBB) betreiben eines der meistgenutzten Schienennetze der Welt. Als Netzbetreiber bietet die SBB-Infrastruktur nationalen und internationalen Eisenbahnverkehrsunternehmen ihre Dienste an und garantiert, dass deren Züge sicher und pünktlich ankommen. Das 2004 gestartete Projekt „Rail Control System" (RCS) ist die künftige Softwaregrundlage für die Wartung und den Betrieb des Schienennetzes und basiert in weiten Teilen auf Eclipse-RCP-Technologien.

Wer denkt bei der Schweizer Eisenbahn nicht an historische Brückenbauten, die in waghalsigen Konstruktionen Flüsse und enge Bergtäler überwinden? Oder die sprichwörtliche Pünktlichkeit der Züge? Die Geschichte der Schweizer Eisenbahn geht ins Jahr 1845 zurück und mündet zunächst in vielen einzelnen, regionalen Eisenbahngesellschaften. Diese wurden 1902 zu den Schweizerischen Bundesbahnen (SBB) zusammengeführt, die heute für den größten Teil des Schweizer Schienennetzes verantwortlich sind.

Die Aufgaben eines Schienennetzbetreibers

Wie in den meisten Staaten Europas ist das Schienennetz organisatorisch vom Zugverkehr getrennt, der das Netz befährt: Ein Unternehmen – in unserem Fall die SBB-Infrastruktur – betreibt das Schienennetz und bietet die Nutzung des Netzes verschiedenen anderen Eisenbahnverkehrsunternehmen an. Dabei handelt es sich um nationale sowie internationale Unternehmen, die das Netz der SBB mit Personen- und Güterzügen befahren. Im Prinzip kann jeder eine so genannte „Trasse" bei der SBB-Infrastruktur buchen, also eine bestimmte Strecke zur Benutzung durch einen Zug zu einem bestimmten Zeitpunkt. Zu den Aufgaben des Netzbetreibers gehören neben dem Ausbau und dem Unterhalt des Schienennetzes die langfristige Planung sowie die kurzfristige Steuerung des Zugverkehrs auf dem Netz. Eine große Herausforderung, da es sich im Fall der SBB um eines der meistgenutzten Eisenbahnnetze der Welt handelt. Abbildung 1 verdeutlicht, was täglich auf den Schienen los ist: Die zurückgelegte Strecke aller Züge an einem durchschnitt-

lichen Tag ist mehr als neunzigmal länger als das Netz selbst. Im Vergleich zu anderen nationalen Eisenbahnen sind die SBB damit absoluter Spitzenreiter.

Trotz der bereits enormen Auslastung besteht weiterer Bedarf nach zusätzlichen Kapazitäten. Diese werden zum einen durch zusätzliche Strecken geschaffen, was allerdings aufgrund der Geografie der Schweiz nur noch bedingt möglich ist und teure Baumaßnahmen erfordert. Die Planungs- und Bauzeiten für neue Streckenprojekte betragen typischerweise Jahrzehnte. Also gilt es, das bestehende Netz so effizient wie möglich zu nutzen. Das beginnt bei der Planung fein abgestimmter Fahrpläne und geht bis zur sekundengenauen Abwicklung des Zugverkehrs. Auch bei Störungsfällen muss man versuchen, das Netz so wenig wie möglich aus dem Takt zu bringen und den verschiedenen Eisenbahnunternehmen trotzdem eine pünktliche Nutzung ihrer Trassen zu ermöglichen. Hier ist der Ansatzpunkt, an dem sich Jahrhunderte alte Technik und moderne Softwaresysteme ergänzen. Das 2004 gestartete Projekt Rail Control System (RCS) schafft eine einheitliche und landesweite Plattform für verschiedene Aspekte des Schienenbetriebs – von der langfristigen Planung bis hin zum Livebetrieb.

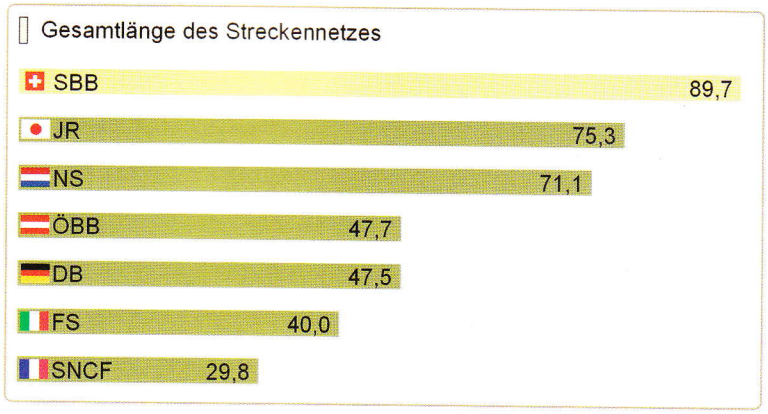


Abb. 1: Auslastung des Schienennetzes: Zurückgelegte Strecke aller Züge im Verhältnis zur Gesamtlänge des Netzes (Quelle: SBB Jahresbericht 2007)

Abb. 2: Grundlage aller RCS-Projekte ist eine detaillierte Datenbasis des Gleisnetzes (Quelle: Google Earth, SBB)

Das Projekt

Das Rail Control System gliedert sich in mehrere Teilprojekte, die zusammengefügt eine optimale Unterstützung des Schienennetzbetreibers SBB-Infrastruktur gewährleisten müssen. Die Grundlage für alle Projekte ist eine detaillierte Datenbasis der Eisenbahninfrastruktur der SBB. Diese wird mithilfe des Teilprojekts Unified Network Objects (UNO) erstellt und laufend gepflegt. Unter anderem sind hier sämtliche Gleise, Weichen und Signale mit ihren technischen Eigenschaften und geografischen Positionen erfasst. Die Einbindung von Google Earth zur Darstellung des Gleisnetzes auf Satellitenbildern erlaubt eine visuelle Qualitätskontrolle der Datenbasis (Abb. 2).

Eine effiziente Nutzung des Streckennetzes beginnt mit der Planung des Zugverkehrs. Das Projekt Netzweites Trassen-System (NeTS) ist das Werkzeug für die netzweite, gleisgenaue Trassenplanung auf den Strecken und Knoten. Diverse grafische Werkzeuge auf Basis der Eclipse Workbench unterstützen dabei die Anwender bei der Fahrplanerstellung. Trotz aller Planung wird der Verkehr auf dem Netz rund um die Uhr von Disponenten überwacht, die bei allen Unregelmäßigkeiten in den Ablauf eingreifen müssen. Deren Arbeitsgrundlage ist RCS Disposition (RCS-D). In den Betriebsleitzentralen sind RCP-basierte Arbeitsplätze mit bis zu acht Monitoren installiert, die ein exaktes grafisches Abbild der aktuellen Betriebslage sowie eine Prognose für die nächsten zwei Stunden bieten (Abb. 3).

Um im Störungsfall eine schnelle und zielgerichtete Kommunikation zwischen den Beteiligten zu gewährleisten, steht der Alamierungs- und Ereignisassis-

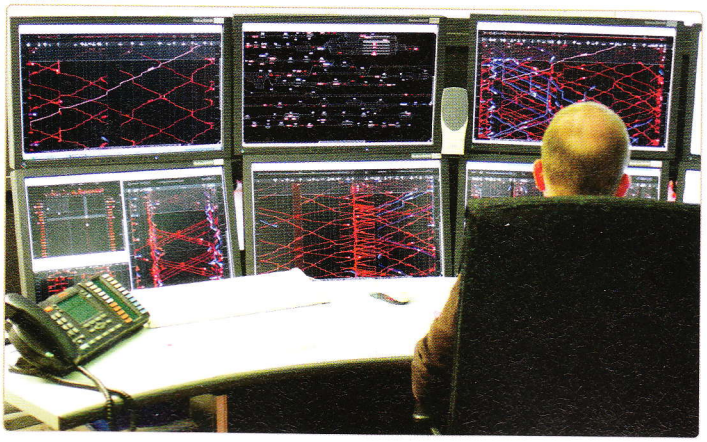


Abb. 3: Die Anwendung RCS-Dispo zur Anzeige der aktuellen Betriebslage (Quelle: SBB)

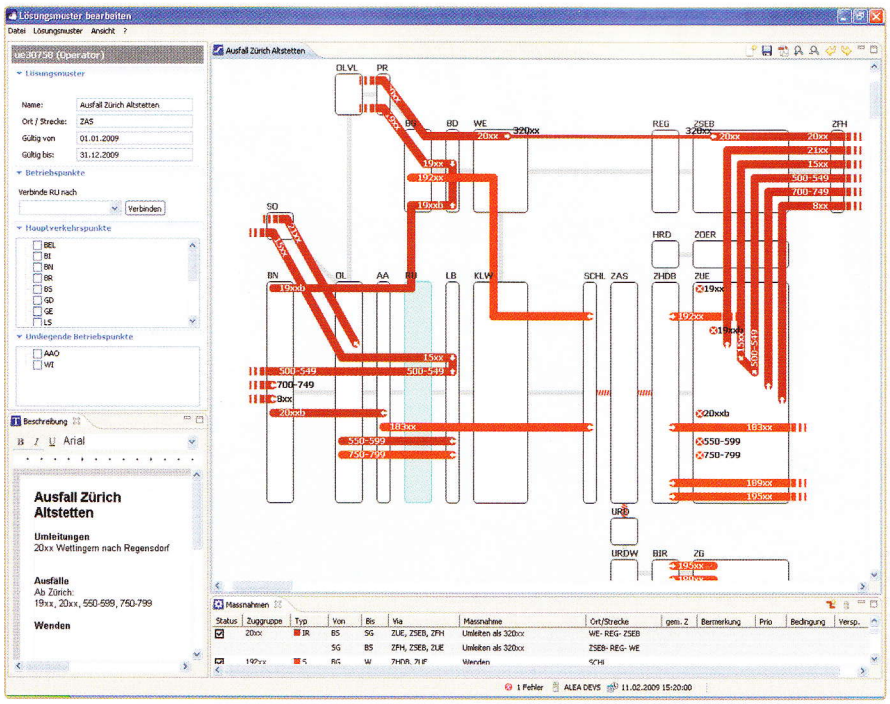


Abb. 4: Grafisch aufbereitete Notfallszenarien für die rasche Behebung von Störungen

tent (ALEA) zur Verfügung. Über ALEA werden alle betroffenen Stellen über Störungen informiert und die notwendigen Maßnahmen zur Behebung untereinander abgestimmt. Grafisch aufbereitete Notfallszenarien helfen z. B. bei der plötzlichen Sperrung einer Strecke, Züge auf alternative Routen umzuleiten (Abb. 4).

Einsatz der RCP Workbench

In jedem der zuvor genannten Teilprojekte wird für die Umsetzung der Clientanwendung die Eclipse RCP Workbench eingesetzt. Deren Konzepte eignen sich hervorragend zur Realisierung von technisch hochstehenden und benutzerfreundlichen Fachanwendungen. Ein Eclipse Plug-in stellt für uns genau die Granularität einer Softwarekomponente dar, die für die Modellierung unserer technischen und fachlichen Module notwendig gewesen ist. Eine Java-Klasse ist als „Komponentenmodell" definitiv nicht tragfähig, weil damit nicht ausreichend die Merkmale beschrieben werden können, die für eine Softwarekomponente in unserem Kontext von Bedeutung sind. Die Möglichkeit, Beziehungen zwischen Plug-ins explizit festzulegen, hilft uns, die notwendige Trennung zwischen technischen und fachlichen Modulen, aber auch zwischen fachlichen Modulen untereinander festzulegen. Durch das ebenfalls explizite Freigeben einer API eines Plug-ins sowie die Verwendung von Extension Points wurde die Interaktion zwischen Plug-ins realisiert. Dabei wird immer darauf geachtet, dass ein Plug-in klein genug bleibt, um es pflegen zu können, aber groß genug, um eine sinnvolle Funktionalität bereit zu stellen. Somit ist die Grundlage für eine sauber strukturierte Codebasis geschaffen, die in unserem Projekt mit einer großen Anzahl von Entwicklern notwendig ist.

Darüber hinaus skaliert eine derartig strukturierte Codebasis gut, weil sie es ermöglicht, dass viele Entwickler nebeneinander (und nicht aneinander vorbei) an getrennten Modulen arbeiten und gleichzeitig ein hohes Maß an gegenseitiger Integration genießen können, weil die Module durch Extensions (Points) miteinander lose gekoppelt sind. Das auf OSGi basierende Komponentenmodell der Eclipse-Plattform unterstützt uns, über den gesamten Softwareentwicklungsprozess die ursprünglich vorgegebene Architektur einzuhalten. Sie wird nicht durch unerwünschte Querbeziehungen verwässert. Die technischen und fachlichen Module bleiben über den gesamten Entwicklungszeitraum stets für Erweiterungen offen, aber immer geschlossen für Änderungen.

Viele Konstrukte und Konzepte, die wir aus dem täglichen Gebrauch der Eclipse IDE kennen, können ebenso für die Umsetzung von Businessanforderungen zum Einsatz kommen. Natürlich werden Züge, Signale, Bahnschranken, Gleise und Bahnhöfe nicht kompiliert oder refaktorisiert, doch so können sie in bestimmte Handlungsabläufe eingebettet werden, die sonst Java-Ressourcen vorbehalten sind. Während mit der Outline View die fachlichen Attribute eines im Editor geöffneten Geschäftsobjekts dargestellt werden, können via Kontextmenü auf einem selektierten Geschäftobjekt fachliche Operationen ausgeführt werden. Zudem werden Themes verwendet, um Farben und Fonts zu verwalten, und es gibt fachliche Property Pages (z. B. für die Anzeige von Ankunfts-, Abfahrtzeiten sowie Verspätungsinformationen eines Zuges).

Die Herausforderungen

Die grafische Aufbereitung des gesamten Schweizer Zugverkehrs stellt nicht nur hohe Anforderungen an die Datenqualität, Übertragungsgeschwindigkeit der Daten und deren Darstellung, sondern auch an den Arbeitsplatz des Bahndisponenten. So besteht ein normaler Arbeitsplatz aus einem PC mit acht Bildschirmen (Abb. 3). Auf diesen acht Bildschirmen wird ein Ausschnitt aus dem Schweizer Eisenbahnnetz dargestellt, was die Aufgabe genau einer Eclipse-Workbench-Instanz ist. Die Herausforderung liegt nun in der Ressourcenverwaltung: Sämtliche 8 x 1600 x 1280 dargestellten Pixel müssen kontinuierlich aktualisiert werden, wobei darunter die Reaktionszeit der Anwendung gegenüber dem Anwender gleichbleibend schnell sein muss.

Die Workbench liefert hier unter Windows XP eine tadellose Perfor-

Das Netz der SBB in Zahlen

- 3011 Kilometer Schienennetz, davon 1751 Kilometer mehrgleisig
- 804 Bahnhöfe und Stationen
- 727 Stellwerke zur Bedienung von 13 642 Weichen und 31 231 Signalen
- 307 Tunnel auf einer Länge von insgesamt 259 Kilometern
- 5873 Brücken mit einer Gesamtlänge von 87 Kilometern

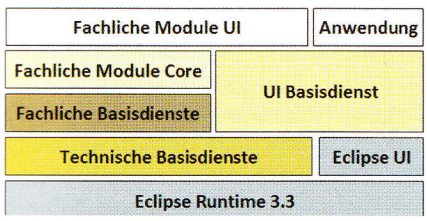


Abb. 5: Schematische Darstellung der Architektur

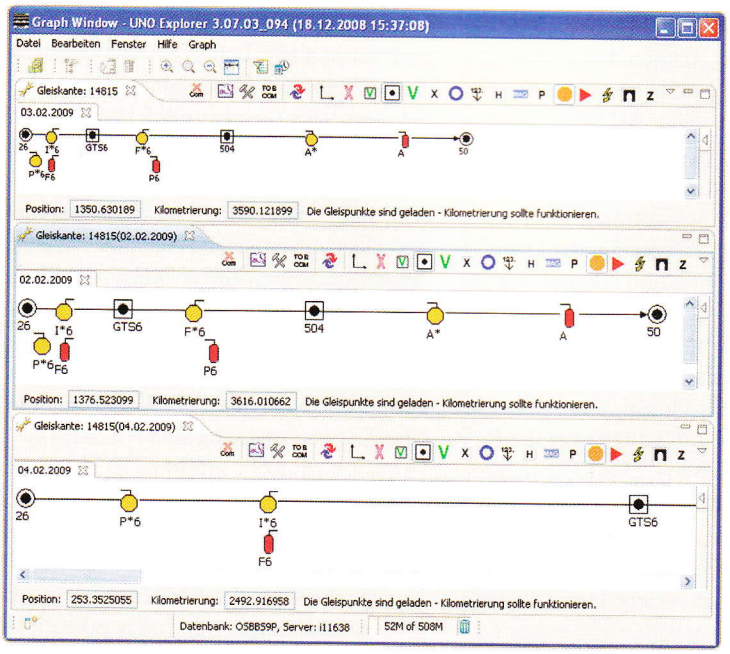


Abb. 6: Zoomen in GEF

mance. Wichtig ist alleine, dass langlaufende Operationen im Hintergrund ablaufen, und dass im SWT Render Thread die Zeit zum Zeichnen minimiert wird. Da die Implementierung des Graphics Context unter Windows auf das äußerst langsame GDI API zugreift, muss bei grafischen Darstellungen darauf besonderer Wert gelegt werden. Um Operationen nebenläufig auszuführen, bietet das Job-API die ideale Infrastruktur. Eine andere Möglichkeit zur Ausführung längerer Operationen bietet die Klasse *org.eclipse.jface.operation.ModalContext*.

Ein weiterer Knackpunkt war die Benutzeranforderung einer bearbeitbaren Tabelle mit 50 000 und mehr Datensätzen. Die Erstellung der Tabellenzellen für zigtausend Zeilen dauert Ewigkeiten und der Ressourcenverbrauch steigt sehr hoch an. Glücklicherweise kann in SWT zu diesem Zweck eine Tabelle mit dem Parameter *SWT.VIRTUAL* instanziiert werden, womit das Problem bereits gelöst ist. Dass der virtuelle Modus nur das Problem der vielen Zeilen löst, nicht aber das mögliche Problem einer großen Anzahl von Spalten, war für uns ohne Relevanz. Ab Eclipse Version 3.3 funktionieren im virtuellen Modus sogar Filter und Sortierung wie erwartet. Allerdings können diese Funktionen bei derart vielen Datensätzen sehr viel Zeit in Anspruch nehmen, weshalb wir Sortier- und Filterfunktionen nebenläufig implementiert haben. Somit hat der Anwender die Möglichkeit, parallel in verschiedenen Editoren zu arbeiten, ohne blockiert zu sein.

Ein kleines Manko sei auch noch genannt: Seit Eclipse 3.2 können mit den Custom Draw Events für Tabellen und Trees mehrzeilige Zellen in einer Tabelle realisiert werden. Leider kann aber der oftmals gewünschte Effekt der platzsparenden mehrzeiligen Tabellen nicht produziert werden, da pro Tabelle nur eine Zeilenhöhe möglich ist.

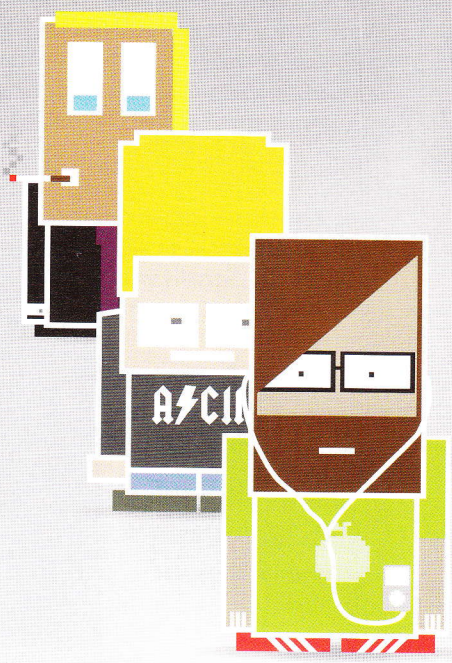
Mit genügend Fantasie konnte auch eine Lösung für das Kopieren von Bereichen in einer Tabelle gefunden werden. Es handelt sich nun nicht um vollständige Zeilen, sondern um einen zusammenhängenden Bereich von Zellen aus mehreren Zeilen und Spalten. Dies war nötig, damit der Anwender die Möglichkeit hat, nur bestimmte Informationen (z. B. ohne Statusinformationen) zu kopieren. Mit *SWT.MeasureItem*, *SWT.EraseItem* und *SWT.PaintItem* lässt sich eine solche Funktionalität umsetzen. Hinderlich bei der Umsetzung war einzig der zugleich aktive TableCursor. Durch einfache Deaktivierung konnte aber verhindert werden, dass dieser für die Tabelle bestimmte Events konsumiert.

Auch das Zoomen mit GEF hat seine Tücken. In der in Abbildung 6 dargestellten Gleisansicht kann der Benutzer die Elemente (Figuren) einsehen, die einen Bezug zu einem Gleis haben. Das Standardverhalten von GEF skaliert grundsätzlich alle dargestellten Elemente gleich. Dies macht in unserem Fall ab Zoomlevel

EARLY BIRD SPECIAL
Jetzt anmelden und sparen!
Melden Sie sich bis zum 23. April 2009 an und sparen Sie 150 € für Ihre Teilnahme!

webinale 09
the holistic web conference

webinale 2009
25. bis 27. Mai 2009
Berliner Congress Center, Alexanderplatz, Berlin



www.webinale.de

1 aber keinen Sinn, da sonst die Figuren mit den zugehörigen Bezeichnungen zu groß werden und somit eine übersichtliche Darstellung verhindern. Dass GEF die Skalierung nach den Bedürfnissen unserer Anwender vornimmt, konnte durch einfaches Erweitern des ZoomManagers erreicht werden. Dort wird nun ab dem Zoomlevel 1 das Layout jeweils neu berechnet und die Figuren werden entsprechend positioniert. Allgemein gilt für GEF: Ist das Konzept der Figuren, Modelle und EditParts einmal durchschaut, lassen sich einfache grafische Editoren in der Regel schnell umsetzen.

Konkrete Anwendungsfälle

RCS ist eine grafikintensive Anwendung. Um einerseits den hohen Anforderungen einer Echtzeitabbildung des Zugverkehrs gerecht zu werden und anderseits die Fähigkeiten der Workbench voll ausnutzen zu können, wird für das Rendering der Zugbewegungsdaten ein eigener JFace Viewer eingesetzt. Der ContentProvider versorgt den Viewer mit den zu rendernden Objekten. Ein LabelProvider (Listing 1) übernimmt das Zeichnen dieser Objekte. Durch die Verwendung des JFace Viewer Patterns erhalten wir volle Unterstützung für Selektion und die damit verbundenen Annehmlichkeiten wie Kontextmenüs, Tooltips, Statuszeileneinträge und Property Pages. Zusätzlich wird die strukturelle Trennung von Inhalt und Darstellung erzwungen.

Um das fachliche Objektmodell so aufzubereiten, dass es unterschiedlich dargestellt werden kann, wird es in ein für die jeweilige Darstellung optimiertes Modell transformiert. Dieser Schritt erfolgt durch eine POJO/POJO-Transformation. Damit liegen den verschiedenen grafischen Repräsentationsformen von Zugdaten jeweils andere Java-Objekte zugrunde.

Um bei der Selektion eines Zugs auf dessen fachliche Attribute zugreifen zu können und diesen Zugriff über sämtliche grafischen und tabellarischen Repräsentationsarten zu vereinheitlichen, wird das Adapter-Pattern verwendet. So wird z. B. für den Zug mit der Nummer 2530 ein und dieselbe Property Page geöffnet (Abb. 7). Für jedes Objekt, das einen Zug repräsentiert (Listing 2), ist eine Adapter Factory definiert (Listing 3 und 4). So kann die Property Page für den Typ *IZuglauf* definiert werden (Listing 5). Um einen Menüeintrag aufgrund der Selektion eines Objekts vom Typ *IZuglauf* einzublenden, wird die Core Expression verwendet (Listing 6).

Der Werkzeugkasten

Jedes Softwareentwicklungsprojekt benötigt eine Sammlung von Werkzeugen, die dem Team die Arbeit erleichtern (prozessunterstützende Tools) oder diese erst ermöglichen (IDE, Compiler). RCS bildet hier keine Ausnahme. Während allerdings bei einem monolithischen Großprojekt ein einheitliches Toolset ein absolutes Muss ist, ergab sich bei RCS ein Freiraum für die einzelnen Teilprojekte, sodass nach den jeweiligen Anforderungen und Bedürfnissen das „beste Tool

Listing 1

```
public interface IShapeProvider extends
                                IBaseLabelProvider {

 /**
 * Zeichnet das gegebene Modelelement einmal in
                                den gegebenen
 * {@link GC}.
 */
 public void render(Object element, GC gc);

}
```

Listing 2

```
//Fachliches Interface: definiert die fachlichen
//Attribute zu einem Zug
public final interface IZuglauf{
 ...
}

//Klasse in Abb. 1 verwendet
public final class GZuglauf{
 ...
}

//Klasse in Abb. 2 verwendet
public final class GZugkasten{
 ...
}
```

Listing 3

```
...
<extension
  point="org.eclipse.core.runtime.adapters">
 <factory
   adaptableType="GZuglauf"
   class=" GZuglaufAdapterFactory">
  <adapter
    type="IZuglauf">
  </adapter>
  ...
 </factory>
</extension>
...
```

Listing 4

```
public class GZuglaufAdapterFactory implements
                                IAdapterFactory {

 private static final Class<?>[] CLASSES = new Class[]
                                { IZuglauf.class };

 public Object getAdapter(Object adaptableObject,
                                Class adapterType) {
  GZuglauf zuglauf = (GZuglauf) adaptableObject;
  if (adapterType == IZuglauf.class) {
   ...
   //fachlicher Gibberish: return Adaption auf
  // fachliches Zugobjekt
   ...
  }
  return null;
 }

 public Class[] getAdapterList() {
  return CLASSES;
 }
}
```

Listing 5

```
...
<extension
  point="org.eclipse.ui.propertyPages">
 <page
   class="ZuglaufPropertyPage"
   id="zuglaufPropertyPage „
   name="Zugdaten Details">
  <enabledWhen>
   <adapt
     type="IZuglauf">
   </adapt>
  </enabledWhen>
 </page>
</extension> ...
```

Listing 6

```
...
<extension point="org.eclipse.core.expressions.
definitions">
 <definition
  id="zuglaufSelektiert">
  <with variable="selection">
   <iterate operator="or">
    <adapt type="IZuglauf">
    </adapt>
   </iterate>
  </with>
 </definition>
</extension>
...
```

für den Job" verwendet werden konnte. Nachfolgend werden Gemeinsamkeiten und Unterschiede zwischen den einzelnen Teilprojekten aufgezeigt.

Nachdem bei allen Projekten Eclipse RCP zum Einsatz kommt, stand die zu verwendende IDE von Anfang an fest: Nur die Eclipse IDE bietet Entwicklern die nötige Unterstützung. Abgesehen davon gibt es aus unserer Sicht aber noch andere Gründe, die für Eclipse sprechen: Erstens die zahlreich verfügbaren Plug-ins, die dem Entwickler die notwendige Funktionalität oder Information direkt in der IDE anbieten. Solche Plug-ins decken bei uns viele Bereiche ab – von der Prüfung der Codequalität (PMD, Checkstyle) und Testabdeckung (EclEmma) über die Taskverwaltung und die Unterstützung des taskbasierten Arbeitens (Mylyn) bis hin zur Kommunikation innerhalb und zwischen den Teams mit einem in Eclipse integrierten Intranet-Chat, um z. B. die Entwickler über den bevorstehenden Neustart des Testsystems zu informieren. Als unerlässlich erwies sich zweitens auch der Memory Analyzer (MAT). Mit seiner Hilfe wurden nicht nur Memory Leaks aufgespürt, allgemein wurde auch der Memory Footprint der Projekte drastisch reduziert (bei einem Teilprojekt auf ca. 25 %). Die Analyse verläuft dabei ohne Laufzeit-Overhead und funktioniert auch unter erschwerten Bedingungen (Heapdumps von über 1 GB, Post-Mortem-Analyse). Und nicht zuletzt machen der Debugging-Support mit Hot Code Replacement und die mächtigen Refactoring-Möglichkeiten Eclipse zu einem Tool, auf das man ungern verzichtet.

Die Schweiz ist wahrscheinlich die kleinste Industrienation mit mehreren offiziellen Landessprachen. Sobald hier ein Softwareprodukt landesweit eingesetzt werden soll, wird Mehrsprachigkeit unausweichlich zu einem Thema. Das gilt auch für die RCS-Teilprojekte. Es wurde dabei auf eine reine Eclipse-Lösung gesetzt: Die Externalisierung der benutzersichtbaren Strings in Property-Dateien (Internationalisierung) erledigt Eclipse, und für deren Übersetzung (Lokalisierung) sorgt das Babel-Projekt, das für die Eclipse-RCP-eigenen Plug-ins fertig übersetzte Sprachpakete bereitstellt. Doch damit nicht genug: Wir betreiben für unsere Projekte einen eigenen Babel-Server. Der Babel-Server liest die Property-Dateien

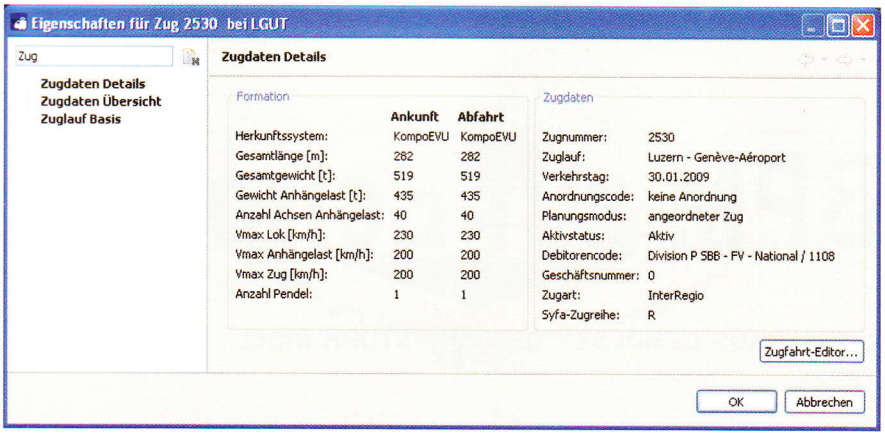


Abb. 7: Property Page für die Darstellung der Attribute eines Zugs

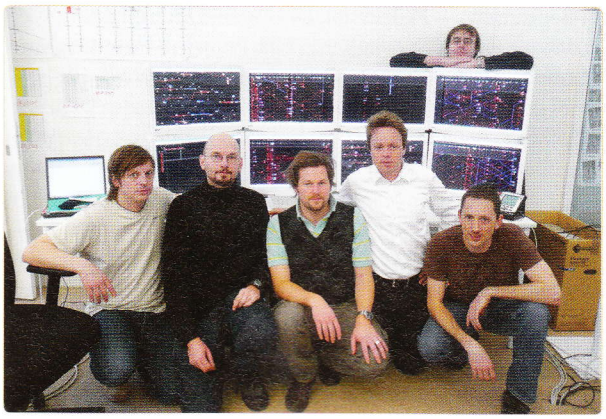
direkt aus dem Subversion Repository ein und legt sie in einer Datenbank ab. Die Übersetzer pflegen über die Babel-Webapplikation die Übersetzungen ein, und unser Babel-Server erzeugt daraus bei Bedarf die entsprechenden Sprachpakete. Es war zwar einiger Initialaufwand notwendig, um den nur mäßig dokumentierten Babel-Server – nebenbei bemerkt eine PHP-Webapplikation – zum Laufen zu bringen, aber dieser Aufwand hat sich gelohnt. Beim Thema Build und Release setzen die Projekte auf unterschiedliche Ansätze. Das Spektrum reicht vom manuellen Release aus Eclipse heraus bis hin zum komplett automatisierten Release mit PDE-Build und Continuous Integration Server. Als erschwerend erweist sich, dass Client- und Serverkomponenten in einem gemeinsamen Build- und Releaseprozess integriert werden sollen. Für diese Problematik gibt es aus unserer Sicht noch keine zufriedenstellende Lösung.

Fazit

Eclipse RCP war aus unserer Sicht ein Quantensprung in der Geschichte der Rich-Client-Entwicklung. Mit den Konzepten der Eclipse Workbench können fast alle möglichen Anforderungen effizient umgesetzt werden. Natürlich gibt es, wie überall, auch hier noch Potenzial für Verbesserungen. Dieses ist aber weitgehend erkannt und wird durch die Eclipse Community schrittweise und kontinuierlich in Verbesserungen umgesetzt.

Zusammenfassend können wir sagen, dass der Einsatz von Eclipse RCP in unseren Projekten eine gute Entscheidung war. Wurde früher eine Rich-Client-Anwendung realisiert, so war der Aufbau eines GUI-Grundgerüsts mit allen dazugehörenden Grundfunktionalitäten im Wesentlichen ein zeitlicher sowie finanzieller Faktor in der Umsetzung der Anwendung.

Nimmt man sich heute zu Beginn die Zeit, sich mit der Materie Eclipse RCP im Detail auseinander zu setzen und die verschiedenen Konzepte sowie deren Einsatzarten zu verstehen, so wird man dafür mit einer Fülle von Erleichterungen belohnt.



Das Autorenteam
dieses Artikels setzt sich aus Entwicklern der Projekte RCS-D, UNO und ALEA zusammen. Von rechts: Daniel Meier (SBB Informatik), Hugo Graf (hinten, Fincons Group AG), Marc Hoffmann (Mountainminds GmbH & Co.), Simon Eggler (softbee GmbH) und Gilles Iachelini (CSC Switzerland GmbH, ganz links).