Taratatata – The Tool Talk

Hallo, mein Name ist Thomas Ronzon. Bei meiner Arbeit beschäftige ich mich seit fast zwanzig Jahren mit den unterschiedlichsten Facetten der Softwareentwicklung. Als Architekt weiß man, dass ein Tool nicht für sich allein steht, sondern stets ein Werkzeug zur Lösung eines Problems darstellt.

In dieser neuen Kolumne möchte ich ausgehend von einer Problemstellung Tools vorstellen, welche bei der Lösung helfen. Dabei können diese Tools (Open-Source) Software, Vorgehensweisen oder auch „mentale Tools" wie Prinzipien zur Softwareentwicklung sein. Gerade die Kombination verschiedener Tools und Prinzipien erlaubt oft eine Mächtigkeit, welche kein Tool alleine für sich leisten kann. Ich denke hier beispielsweise an die diversen Unix-Tools, welche erst durch das Prinzip der Pipes mächtig werden.

Ich möchte Sie aber auch dazu einladen, über die Lösung zu diskutieren!

Bis dahin
Thomas Ronzon .-)

Altenpflege mit Java

Lose Kopplung für stabilen Code

Thomas Ronzon

Wer mich kennt, weiß, dass ich mich oft mit alter Software „herumschlage". Dabei muss alt aber nicht gleich schlecht sein. Auch wenn die Techniken sich in den letzten Jahren weiterentwickelt haben, so hat manche alte Software durchaus noch ihre Berechtigung. Sei es, weil sie einfach unauffällig ihren Dienst tut (die Fehler sind im Laufe der Laufzeit alle behoben), oder sei es auch, weil soviel undokumentierte Logik in dem Code steckt, dass eine Neuimplementierung ziemlich aufwendig und risikoreich wäre.

Problembeschreibung

Wir haben ein Altsystem, welches spezielle Buchungen in ANSI-C geschriebenem Code durchführt. Diese Buchungen sollen auch vom neuen, in Java geschriebenen System ausgeführt werden und sind nicht trivial in Java umzusetzen.

Lösungsvorschlag

Nun, wir könnten den Code als native Code, wie Christian Robert in seinem Artikel [Rob14] beschreibt, per Java Native Interface (JNI) integrieren. Leider ist damit eine sehr enge Kopplung der Teile verbunden. Zusätzlich müssen die Teile auf der gleichen Hardware laufen. Schöner wäre es, diese verteilt laufen zu lassen und eine lose Kopplung anzustreben. Nach einer kurzen Überlegung kommt man schnell auf die Idee, dass ein Webservice hier eine tolle Sache wäre [WEBS].

Aber mal ehrlich, wer hat schon Lust dazu, eine WSDL-Datei zu schreiben, welche die Schnittstelle definiert, und dann auch noch Wrapper-Klassen für den Java- und für den C-Teil?

Wer jedoch C kann, wird sich vielleicht an die „include-Files" [INCLUDE] erinnern. Diese beinhalten die Funktionsdeklarationen aller öffentlichen Funktionen – eigentlich das Gleiche, was wir auch in der WSDL-Datei benötigen. Der Trick ist nun, diese Header-Dateien umzuwandeln!

Hier bietet sich gsoap [gSOAP] an. Bei gsoap handelt es sich um ein Paket für Webservices in C und C++. Schön ist, dass es hier genau für diesen Zweck das Tool *soapcpp2* im bin-Verzeichnis gibt.

Ein Aufruf von

```
soapcpp2 -c <INCLUDE-FILE>
```

generiert aber nicht nur die WSDL-Datei, sondern die Stubs für den C-Teil gleich mit. Leider ist ein wenig Arbeit nötig, da die aufzurufenden Funktionen den Namespace (im Beispiel `rot`) vorangestellt haben müssen. Also beispielsweise

```
int rot__plus (int a, int b, int &ergebnis );
```

Nun zum Java-Teil: Ein bekanntes Tool für Webservices im Java-Umfeld stellt Apache CXF da [CXF]. In diesem Paket ist auch ein interessantes Tool enthalten: *wsdl2java*. Es generiert aus einer WSDL-Datei die nötigen Java-Klassen für den Zugriff.

Ein Aufruf von

```
wsdl2java rot.wsdl
```

und schon habe ich auch den Java-Teil inklusive einer Klasse `Plus()` generiert, welche den Aufruf komplett kapselt.

Abb. 1: Ablauf: vom C-Header zur Java-Klasse

Fazit

Durch eine geschickte Kombination aus den Paketen *gsoap* und *Apache CXF* ist es vergleichsweise einfach, eine lose Kopplung von native Code zu bewerkstelligen. Möchte man Code in einer anderen Sprache anbinden, so kann anstelle von gsoap die WSDL-Datei natürlich auch mit jedem anderen Tool generiert werden.

Links

[CXF] `http://cxf.apache.org/`
[gSOAP] `http://www.cs.fsu.edu/~engelen/soap.html`
[INLCUDE] Sun Studio 12: C User's Guide, How to Specify Include Files, `http://docs.oracle.com/cd/E19205-01/819-5265/bjadq/index.html`
[Rob14] Ch. Robert, Java an C++, bitte melden! JNI – Java Native Interface, in: JavaSPEKTRUM, 4/2014, s. a.: `http://www.sigs-datacom.de/fachzeitschriften/javaspektrum/archiv/artikelansicht.html?tx_mwjournals_pi1[pointer]=0&tx_mwjournals_pi1[mode]=1&tx_mwjournals_pi1[showUid]=7700`
[WEBS] `http://de.wikipedia.org/wiki/Webservice`

Thomas Ronzon ist seit mehr als zehn Jahren bei der w3logistics AG in Dortmund als Projektleiter bei diversen Logistik-Projekten beschäftigt.
E-Mail: ronzon@w3logistics.de