Open Suse Buildservice

Software für Millionen

von Klaas Freitag
Erschienen im Linux-Magazin 2008/01

Open Suses freier, übers Web zugänglicher Buildservice nimmt jedem interessierten Entwickler Arbeit und Ärger ab. Er kann sich aufs Programmieren konzentrieren statt für diverse Plattformen Compiler-Sessions abzuhalten, Abhängigkeiten nachzuhecheln und Pakete für Dutzende Distributionen zu schnüren.

Vor gut zwei Jahren hat die Firma Novell, Mutter des deutschen Linux-Distributors Suse, die Entwicklung der Distribution komplett geöffnet, um Suse Linux zu einem echten Community-Projekt werden zu lassen, dessen Entwicklung nicht mehr abgeschlossenen in den eigenen Labors stattfindet.

Das Ergebnis der Entwicklerarbeit ist der Sourcecode, der zwar ein wesentlicher Teil, aber nicht die einzige Voraussetzung für ein erfolgreiches Open-Source-Projekt ist. Der Buildservice soll eine weltweite und effiziente Zusammenarbeit aller Entwickler ermöglichen. Die Suse-Distributionsentwickler arbeiten als aktiver Teil der Community an Open Suse, da es die Basis für die Suse Enterprise Platform darstellt.

Das Projekt bietet neben Kommunikationsplattformen wie einem mehrsprachigen Wiki unter [1] und einer leistungsfähigen Infrastruktur zum Software-Download zusätzlich einen freien Service, der Software übersetzt und in Pakete verpackt, den Open Suse Buildservice (Abbildung 1). Dieser Onlineservice überführt die Quellpakete der Software in ein Binärformat.

Abbildung 1: Das Webinterface des Open Suse Buildservice im Browser.

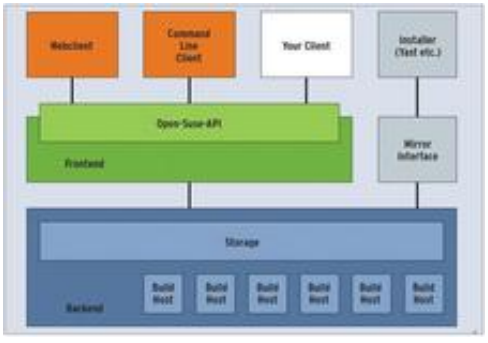

Abbildung 2: Eine Übersicht über die einzelnen Teile des Open Suse Buildservice. Frontend und Backend kommunizieren über eine sichere Netzverbindung.

Programmieren und Packen

Nach dem Schreiben des Code folgt der mühsame und ungeliebte, deshalb auch oft vernachlässigte Arbeitsschritt, die Quellen zu übersetzen und für die Distribution zu paketieren, auf der die Software laufen soll. Das ermöglicht es Benutzern, das Programm über das Softwaremanagement der Linux-Distribution zu installieren. Es kann eventuell vorhandene Abhängigkeiten auflösen und so sicherstellen, dass die Software ordnungsgemäß funktioniert.

Da in der Welt der freien und Open-Source-Software (FOSS) die Vielfalt regiert, ist es für den Autor eines Projekts praktisch unmöglich, für die Vielzahl der gängigen Distributionen und deren aktuelle Releases Binärpakete zu erstellen. Die Systeme unterscheiden sich und es gibt keine einheitliche Packaging-Methode. Oft lässt sich nur sinnvoll auf den Systemen selbst bauen, was bedeutet, dass der Autor diese zur Verfügung haben muss. Ähnliches gilt für verschiedene Hardwareplattformen.

Nicht nur RPM, sondern auch Debian

Mit Hilfe des Open Suse Buildservice lassen sich Pakete für die Hardwareplattformen I-586, x86_64 und in Zukunft wahrscheinlich auch PowerPC bauen. Entwickler können Pakete auf Basis aller gängigen Suse-Distributionen wie Open Suse, aber auch für die Businessvarianten SLES und SLED verschnüren. Darüber hinaus bietet der Buildservice nicht nur die Möglichkeit, für andere RPM-basierte Distributionen wie Red Hat, Fedora und Mandriva zu bauen, sondern auch für Dpkg-basierte Systeme wie Debian und damit (K)Ubuntu. Das geschieht alles aus einer Quelle, die mit den entsprechenden Bauvorschriften zu kombinieren ist.

Ändern sich Pakete, die als Grundlage zum Bau der eigenen Software dienen, baut der Buildservice die abhängigen Pakete automatisch neu. Damit ist auf einem System, das sich noch in Entwicklung befindet, immer gewährleistet, dass die Binärpakete zusammenpassen und aktuell sind. Für den Autor eines Projekts wird die Welt damit einfach: Er stellt sein Paket mit den entsprechenden Bauvorschriften in den Buildservice ein und erhält automatisch die Binärpakete für alle gewünschten Systeme.

Doch es geht nicht nur um Pakete, also die atomaren Einheiten für die Paketmanagement-Systeme heutiger Linux-Distributionen. Im Buildservice lassen sich Pakete zu Projekten zusammenfassen, die er dann zusammen verwaltet und baut. Das eröffnet eine ganze Reihe Möglichkeiten, vor allem, da Projekte aufeinander aufbauen können.

Natürlich ist das alles nur von Wert, wenn die Benutzer die Software auch herunterladen und installieren können. Dafür ist der Open Suse Buildservice mit einer leistungsfähigen Softwaresuche und einer Redirector-Infrastruktur ausgestattet, um die Benutzer automatisch auf leistungsfähige Mirrors umzuleiten. Yast und andere Softwaremanagement-Systeme können die Projektverzeichnisse als Installationsquellen sowie seit Open Suse 10.3 auch einfach zur One-Click-Installation verwenden.

Architektur

Der Buildservice besteht aus vier wesentlichen Teilen, die logisch voneinander getrennt sind und über klare Schnittstellen miteinander kommunizieren. Die Schnittstellen sind alle nach REST-Art (Representational State Transfer) ausgeführt, die zu transportierende Information ist in XML-Dokumente verpackt, die die Systemteile über die üblichen HTTP-Requests »GET«, »PUT« oder »DELETE« untereinander austauschen. Das betrifft sowohl Daten wie die Quellpakete oder Specfiles als auch Metadaten, die die Verarbeitung der Daten festlegen. Zum Beispiel sorgen die HTTP-»POST«-Kommandos für die Übertragung von Kommandos und auch fürs Anlegen von neuen Dateien.

Die Entscheidung fiel auf die REST-Methode, weil sie ein Protokoll verwendet, das durch eine Vielzahl von Sprachen und Entwicklungstools unterstützt wird. Es handelt sich um eine klare, einfache und gut nachvollziehbare Architektur, mit der elegante Interfaces zu designen sind, die sich einfach in verschiedenste Systeme integrieren.

Der Open Suse Buildservice ist vom Internet aus über das so genannte Frontend zu steuern. Es ist das Programmierinterface, auf das Clients zugreifen. Das API bildet Basisrequests wie zum Beispiel das Anlegen von Projekten und Paketen, das Editieren von Repositories und das Verändern von Metadateien ab. An persistenten Daten hält das Frontend nur wenige Benutzerdaten und gecachte, daher reproduzierbare Daten vor. Da das Frontend im Internet erreichbar ist, sind dort keine wichtigen Daten gespeichert, damit im Fall eines Angriffs insbesondere Cracker keine Sourcen unbemerkt verändern können.

Das Frontend wurde mit Hilfe des Framework Ruby on Rails entwickelt. Für diese Art von Systemen bietet Ruby on Rails einige bestechende Vorteile: Es erlaubt eine schnelle und elegante Entwicklung und profitiert von der großen und dynamischen Community. Dazu kommt die innovative Umsetzung von Webtechnologien in Ruby on Rails.

Backend

Das Open-Suse-Buildserver-Backend ist als Überbegriff jener Teile des Systems zu verstehen, die nicht direkt über das Internet zu erreichen sind, aber wichtige Daten halten und die eigentlichen Paketbauvorgänge steuern. Zwischen Frontend und Backend besteht ebenfalls nur eine wohldefinierte REST-Schnittstelle, die zum Beispiel Load Balancing und Mirroring der Daten ermöglicht.

Ein Teil ist der Source-Server, der alle paket- und projektrelevanten Daten verwaltet. Das sind Sourcen und Baubeschreibungen genauso wie Patches. Alle diese Daten legt der Server versioniert ab und zeichnet Änderungen in Form einer Historie auf.

Der Repository-Server verwaltet die Binärpakete, die zum Aufbau der Build-Umgebungen dienen. Ein so genannter Publisher organisiert die Weitergabe von Binärpaketen in Download-Bereiche und erzeugt auch die nötigen Metadaten, zum Beispiel Repository-Daten in verschiedenen Formaten zur Integration in die jeweiligen Installer.

Ein Scheduler organisiert durch Analyse von Abhängigkeiten, wann und in welcher Reihenfolge Projekte neu gebaut werden müssen. Wenn sich zum Beispiel ein Basispaket, etwa der GCC, geändert hat, heißt das für einen gerade in Entwicklung befindlichen Codestand, dass die davon abhängigen Pakete neu gebaut werden müssen, um die Änderungen am Basisteil mit einzubeziehen.

Der Scheduler organisiert über einen Worker-Prozess die Kommunikation mit den Build-Hosts, auf denen das Paket gebaut wird. Dieser Bauvorgang findet auf einem anderen Rechner statt, der über das Netzwerk als Build-Host am Backend angemeldet ist. Auf dem Build-Host läuft per Xen-Virtualisierung eine sichere Umgebung für diesen Bau. Als Alternative lässt sich auch eine Chroot-Umgebung konfigurieren.

Die Build-Umgebung stellt jene für die Übersetzung nötigen Dateien bereit, die das System über das Specfile und aufgelöste Abhängigkeiten herausfindet. Nach der Beendigung des Baujobs bekommt das Backend das fertige RPM zurück und der Job wird als fertig markiert. Im Fehlerfall steht das Logfile mit Debug-Information zur Verfügung.

Clients für Packager

Der Paketbauer interagiert mit dem Frontend über einen Client, der die wichtigen Funktionen benutzerfreundlich kapselt. Da die Arbeitsweise von Programmierer zu Programmierer stark variiert, ist die Entwicklung von Clients so flexibel und standardisiert wie irgend möglich gehalten. Clients kommunizieren mit der REST-Schnittstelle des Frontend und steuern so indirekt alle Vorgänge im Backend des Buildservice.

Gegenwärtig existieren drei Clients: »osc« ist ein in Python geschriebenes Kommandozeilenwerkzeug, das in seiner Bedienung stark an Subversion erinnert. Es stellt die nötigen Mittel zur Verfügung, um alle Fähigkeiten des Open Suse Buildservice auszureizen, erfahrene Benutzer schätzen seine Effizienz auf der Kommandozeile. Ganz im Stil eines Quelltext-Verwaltungssystems wie SVN können Entwickler OSC-Projekte und Pakete auschecken, verändern, sowohl lokal als auch auf dem Server bauen und die Änderungen wieder einchecken. Metadaten lassen sich durch Editieren von XML-Dateien ändern. OSC ist modular aufgebaut und zusätzlich durch Plugins erweiterbar.

Außerdem gibt es den in Abbildung 1 gezeigten Webclient im Browser. Er ermöglicht es dem Anwender, Projekte und Pakete auf dem Server komfortabel in einer durch Ajax-Technologie schnell bedienbaren Oberfläche zu bearbeiten. Gerade wenn viele Pakete im Spiel sind, behält man damit leicht den Überblick. Insbesondere Anfänger können sich mit dem Webclient einen guten Überblick über die Möglichkeiten des Buildservice verschaffen und leicht neue Pakete und Projekte anlegen.

Eher experimentell ist zurzeit noch ein KDE-Client, den der Entwickler wie OSC lokal installiert und der das Bearbeiten von Paketen komfortabel in einer KDE-Oberfläche gestattet, wobei die Kommunikaton mit dem Server komplett gekapselt ist. Das Projekt war Teil von Googles Summer of Code.

Ärmel hochkrempeln

Um zu zeigen, wie Benutzer den Open Suse Buildservice in der Praxis verwenden können, soll ein fiktiver Autor freier Software im Folgenden ein Paket im Buildservice entwickeln. Er möchte für sein Projekt Suse-Pakete, zusätzlich aber auch Binärpakete für möglichst viele andere Distributionen anbieten.

Um den Open Suse Buildservice zu verwenden, ist ein Account erforderlich. Es ist derselbe, den auch das Wiki und der Bugzilla-Bugtracker [http://bugzilla.novell.com] verwenden. Er lässt sich leicht auf der Startseite des Wiki [http://en.opensuse.org] anlegen. Wer sich einloggt, gelangt automatisch auf eine Seite, auf der er auch sein Home-Projekt findet (Abbildung 3). Dabei handelt es sich um eine Spielwiese, auf der die Benutzer nach Herzenslust

experimentieren können, ohne dabei etwas Wichtiges kaputt zu machen.

Abbildung 3: Das Home-Projekt des Buildservice-Benutzers, noch sind keine Pakete angelegt.

Prinzipiell sind Home-Projekte ebenbürtig zu den großen Projekten: Sie lassen sich linken und es können mehrere Personen Schreibrechte darauf bekommen. Lediglich bei der Suche des Softwareportals [http://software.opensuse.org/search] sind sie dadurch etwas zurückgesetzt, dass die Treffer in Home-Projekten stets unter denen regulärer Projekte erscheinen.

Als Erstes gibt der Entwickler dem Home-Projekt einen Namen und beschreibt kurz, was in diesem Projekt passieren wird. In dem Beispielprojekt namens "Kraft" sollen Binärpakete entstehen. Natürlich soll der Buildservice auch seine Abhängigkeiten, das heißt die Pakete, die für den Betrieb von Kraft dringend nötig sind, bauen und zum Download anbieten, sofern sie nicht Teil des Basissystems sind.

Struktur von Projekten und Paketen

Ist zum Beispiel ein erfahrener Paketbauer dazu eingeladen, ein Auge auf das Projekt zu werfen, kann er als Maintainer zum Projekt hinzukommen. Ein Klick auf den Link »Add User« öffnet ein entsprechendes Formular für die Eingabe der User-ID. Jetzt besteht ein Projekt als Rahmen für Pakete, die innerhalb des Projekts entstehen.

Alle Eigenschaften vererbt das Projekt an die Pakete, solange sie nicht im Paket anders gesetzt werden. Zum Beispiel haben alle in dem Projekt liegenden Pakete den zusätzlichen Maintainer »bauersman«, ohne dass dieser explizit im einzelnen Paket gesetzt ist. Der Link »add Package« legt ein neues Paket an, was einen Systemnamen, einen gut lesbaren Titel und eine Beschreibung des Pakets erfordert. Eine zusätzliche Option legt ein Specfile-Template in dem Paket an.

Repositories, für die Pakete entstehen sollen, lassen sich ebenfalls im Projekt festlegen. Das sind im Normalfall stabile Distributionen wie Open Suse 10.3 oder Suse Linux Enterprise Platform 10. Im Einzelfall können das auch spezielle Repositories sein, die dadurch entstehen, dass Basissysteme in bestimmten Aspekten verändert und dann ebenfalls als Grundlagen-Repositories verwendbar sind. Zum Beispiel kann der Entwickler entscheiden, dass sein Paket nicht auf einem Standard Open Suse 10.3, sondern auf einer 10.3-Version mit besonders optimiertem KDE 3 basiert.

Das Build-Beispiel des Artikels nimmt einige wichtige, RPM-basierte Repositories hinzu, etwa die letzten stabilen Open-Suse-Versionen 10.2 und 10.3 sowie Fedora und Mandriva. Eine besondere Stellung nimmt Open Suse Factory ein. Das ist die noch nicht stabile nächste Open-Suse-Release. Hier finden die meisten Änderungen statt und es sind aufgrund der sich ständig ändernden Basispakete häufige Rebuilds zu erwarten.

Die Benutzer des Buildservice stört das glücklicherweise nicht, denn die Rebuilds laufen ja vollständig automatisch ab. Dafür ist das eigene Paket an der Spitze der Distributionsentwicklung dabei und kann sich rasch an inkompatible Bibliotheken oder Ähnliches anpassen.

Abbildung 4: Anlegen eines neuen Pakets im Buildservice.

Command Line Tool

Nun geht es darum, die Pakete im Beispielprojekt zu füllen. Der Kommandozeilen-Client »osc« erledigt das schnell und unkompliziert. Dafür checkt der Entwickler als Erstes das gesamte Projekt auf die lokale Workstation aus:

```
> osc co home:krafti
A    home:krafti
A    home:krafti/kraft
A    home:krafti/kraft/kraft.spec
```

Dann gibt er die nötigen Paket-Sourcen ein und ergänzt die Specdatei um die für das spezifische Paket nötige Bauvorschrift. Um zu prüfen, ob beim Paketbau Fehler auftreten, kann der Entwickler es zum Test lokal bauen. Der Aufbau der Build-Umgebung erfolgt automatisch mit den entsprechenden Paketen von Open Suse Server. Der Entwickler muss im Projekt angeben, für welche Hard- und Softwareplattformen der Buildservice bauen soll. Eine Übersicht liefert:

```
> osc repos
Fedora_7         i586
Fedora_7         x86_64
Mandriva_2007    i586
openSUSE_10.2    i586
openSUSE_10.2    x86_64
...
```

Obwohl die eigentliche Basis Open Suse ist, soll testweise auch ein Paket für Fedora entstehen:

```
> osc build Fedora_7 i586 kraft.spec
Getting buildinfo from server
...
```

Das Kommando benötigt etwas Zeit, denn es holt ein komplettes Basissystem für Fedora 7 vom Server und installiert eine Chroot-Baustelle auf der lokalen Maschine. Es sind sicherlich einige Durchläufe erforderlich, bis das Paket für alle gewünschten Basissysteme funktioniert. Dabei ist es sehr hilfreich, dass der Entwickler direkt in der Chroot-Umgebung debuggen kann, um zu erkennen, warum ein Build fehlschlägt. Die typischen Fehler reichen von unvollständigen Bauumgebungen über Portabilitätsprobleme (Stichwort Lib64) bis zu Dateilistenfehlern.

Im Allgemeinen ist es möglich, aus nur einem Specfile für alle RPM-basierten Distributionen Pakete zu bauen, weil es im Buildservice für die offenbar unvermeidlichen Unterschiede Makrodefinitionen gibt, die die jeweiligen Systemspezifika abfangen oder durch geschickte If-Abfragen je nach Basissystem Kommandos während des Baus unterschiedlich ausführen (siehe den Kasten "RPM-Specs"). Wenn das Specfile nach einiger Arbeit einwandfrei für alle gewünschten Basissysteme funktioniert, überträgt »osc« die Änderungen auf den Open Suse Server:

```
> osc add kraft-0.20.tar.bz2
```

```
> osc add kraft.spec
> osc commit
```

Der Entwickler darf dem Projekt jetzt, egal ob mit OSC oder Webclient, weitere Pakete hinzufügen. Die Pakete können auch innerhalb des Projekts Abhängigkeiten untereinander erfüllen. Wann immer der Buildservice Änderungen an seinen Paketen vornimmt, merkt er sie auf dem Server zum Rebuild vor.

Abbildung 5: Projektansicht mit dem Baustatus der jeweiligen Basisrepositories.

RPM-Specs

Das Übersetzen und Paketieren der Quellen auf RPM-basierten Systemen erfolgt gemäß der Specdatei, die im Grunde genommen Shellkommandos enthält, die der Buildservice nacheinander abarbeitet. Leider unterscheiden sich die Systeme im Detail, sodass zum Bauen für verschiedene Distributionen aus einem einzigen Specfile einige Tricks erforderlich sind. Beispielhaft hier ein Ausschnitt aus einer Specdatei, der den richtigen Configure-Aufruf ermittelt:

```
%if 0%{?fedora_version} >= 5
source "%{_sysconfdir}/profile.d/qt.sh"
%configure
--disable-dependency-tracking
--with-xinerama
--with-extra-libs=%{_libdir}
%else
export CFLAGS="$RPM_OPT_FLAGS"
export CXXFLAGS="$RPM_OPT_FLAGS"
./configure
--prefix=/opt/kde3
--with-qt-libraries=/usr/%_lib/qt3/%_lib
%ifarch x86_64 ppc64 s390x
  --enable-libsuffix=64
%endif
%endif
```

Ebenso unterscheiden sich die Distributionen in den Paketnamen, was bei der Auswahl der Pakete für die Build-Umgebung Schwierigkeiten macht, weil dafür nur eine einzige Liste im Specfile genügen soll. Das löst der Buildservice, indem er Mapping-Tabellen führt, die passende Paketnamen auf Nicht-Suse-Systemen enthalten.

Quelle des Glücks

Wie können interessierte Linux-Anwender die im Buildservice gebaute Software nun herunterladen und ausprobieren? Alle Projekte landen automatisch in der Softwareverteilung von Open Suse. Viele Mirrors weltweit spiegeln die Binärpakete, um schnelle Downloads zu ermöglichen. Der Benutzer muss sich auch nicht darum kümmern, von welchem Mirror er herunterlädt, das geschieht über einen ausgefeilten Weiterleitungsmechanismus [2], der den besten

Mirror für den Anwender bestimmt und transparent zu diesem vermittelt.

Die Softwaresuche ist Sache von [http://software.opensuse.org/search]. Nach Eingabe eines Suchworts erscheint die Liste der Projekte, in denen das Wort vorkommt. Um die gefundene Software zu installieren, genügt ein Klick auf den »1-Click-Install«-Button. Damit wird automatisch in Yast das entsprechende Software-Repository angemeldet, dann die gewünschte Software mit allen abhängigen Paketen installiert.

Der Buildservice kennt noch mehr Möglichkeiten zur Verteilung der Software. Zum Beispiel kann der Entwickler ein Live-System von Open Suse 10.3 erstellen, das neben einem Basissystem das Paket "Kraft" zur Demonstration enthält. Open Suse bietet mit KIWI [3] ein Open-Source-System zum Zusammenstellen von Live-Images. Die Images lassen sich zum Beispiel in Xen- oder VMware-Instanzen verwenden. Aber auch die offiziellen Open-Suse-Live-DVDs entstehen damit. Daher kann die bestehende offizielle Konfiguration der Open-Suse-10.3-Live-DVD als Basis dienen. Der folgende Befehl checkt die offizielle Konfiguration aus dem Repository aus:

```
svn co https://forgesvn1.novell.com/svn/Uopensuse/trunk/distribution/images/
```

Ein neuer Abschnitt fügt das Repository des Open Suse Buildservice der Datei »config.xml« hinzu:

```
<repository type="rpm-md">
  <source path="opensuse:KDE:KDE4/openSUSE_10.3"/>
</repository>
```

Dieses Beispiel nimmt zusätzlich das Projekt »KDE:KDE4« auf. KIWI wird die Quelle später über [http://download.opensuse.org/repositories] anmelden. Außerdem muss der Entwickler im Abschnitt »packages« noch die gewünschten Pakete hinzufügen. Eine neue Live-CD erstellen dann folgende Befehle:

```
kiwi --root $HOME/mydvd --prepare $PATH_TO_CONFIG.XML
kiwi --create $HOME/mydvd -d $HOME
```

Das Resultat ist gleich nach Abschluss des Mastering im Unterverzeichnis »mydvd« im Homeverzeichnis des Benutzers zu finden.

Das Open-Suse-Projekt ist mit dem Anspruch angetreten, eine wirklich offene Linux Distribution zu schaffen - bis dahin ist es noch ein weiter Weg. Der Open Suse Buildservice ist das Vehikel, um viele Probleme auf transparente, gleichberechtigte und effiziente Art zu lösen [4]. Als Softwareservice im Netz soll der Buildservice die Entwickler vom Ballast des Paketbaus für verschiedene Distributionen befreien.

Abbildung 6: Bei der Suche gefundene Pakete lassen sich mit einem Klick installieren.

Wohin die Reise geht

Der nächste große Schritt ist die Organisation der Zusammenarbeit: Wie kann jemand ein Paket anpassen, bei dem er nicht Maintainer ist? Der Buildservice wird es ermöglichen, dem Maintainer Patches anzubieten und ihre Anwendung zu diskutieren [5]. Ein weiterer Schritt bei der Verteilung des Service wird mehrere Buildservice-Installationen miteinander kommunizieren lassen, um zum Beispiel Repositories untereinander auszutauschen. *(ofr)*

Infos

[1] Buildservice-Wiki: [http://de.opensuse.org/Build_Service]

[2] Open Suse Buildservice Redirector: [http://en.opensuse.org/Build_Service/Redirector]

[3] KIWI: [http://en.opensuse.org/Build_Service/KIWI]

[4] Buildservice-Tutorial: [http://en.opensuse.org/Build_Service_Tutorial]

[5] Buildservice-Roadmap: [http://en.opensuse.org/Build_Service/Roadmap]

Der Autor

Klaas Freitag ist Entwickler und arbeitet als Software-Architekt in der Abteilung bei Suse/Novell, die die Buildservice-Entwicklung vorantreibt.