Über Ant und Maven zu SBT und Gradle

Andreas Hartmann - Dr. Halil-Cem Gürsoy

adesso AG

Dortmund

Schlüsselworte:

Buildmanagement, Repository Server, Ant, Maven, SBT, Gradle, Ivy

Einleitung

Nach Zeiten der Stagnation im Build Tool Umfeld betreten gleich zwei neue Buildsysteme die Bühne. Mit Graddle tritt ein Produkt an, von dessen Entwickler sich sicher sind, dass es das hält, was Maven immer versprochen hatte, aber auch nie ansatzweise einhalten konnte: konsequente Umsetzung von „convention over configuration“, dabei dennoch transparent und flexibel. Und mit dem „Simple Build Tool“ schickt sich ein Anwärter aus der Scala-Welt an, das Feld von hinten aufzuräumen.

Eine kritische Bestandsaufnahme und Ausblick auf aktuelle und zukünftige Entwicklungen.

Ant-Maven-SBT-Gradle

Betrachtet man das Thema **Buildmanagement** zunächst von der konzeptionellen Seite so sind das Bauen und Deployen, das Erstellen von QS-Reports, das Erstellen von Artefakten und die Verwaltung von 3rd-Party Libraries nur einige Kernprozesse die von einem Buildmanagementwerkzeug abgebildet werden müssen.

Bis zum Jahre 1999 wurde im Rahmen des Buildmanagements häufig make eingesetzt und ist im C/C++ Umfeld immer noch weit verbreitet. Make hat neben den nicht gerade intuitiven Charakter den gravierenden Nachteil nicht auf allen Betriebssystemen verfügbar zu sein. James Duncan Davidson stellte 1999 mit **Ant** ein Buildmanagementwerkzeug zu Verfügung das nicht nur unter allen Betriebssystemen verfügbar ist, auf den Java läuft, sondern auch im Gegensatz zu make speziell für das Bauen von Java Software konzipiert wurde und somit erheblich besser hierfür geeignet ist. Ant verfolgt hierbei einen imperativen Ansatz. Die Targets repräsentieren dabei die Funktionen und die Task die einzelnen Funktionen. Neben dem klassischen javac Task steht ein umfangreicher Satz von weiteren Tasks wie z.B. delete, mkdir, junit, jar, war, etc. zur Verfügung. Neben den bereits sehr umfangreichen Satz von Tasks bietet Ant durch sein Pluginkonzept eine einfache Möglichkeit für Erweiterungen.

Mit der zunehmender Komplexität der Software und der konzeptionellen Aufteilung in Komponenten und Service sind die Anforderungen an Ant mitgewachsenen. Ant setzt auf XML als Sprachekonstrukt und gibt dabei selber keinen Buildprozess vor. Des Weiteren werden keine Vorgaben bzgl. der Verzeichnisstruktur eines Softwareartefakts sowie keine Standards für Target-Namen gemacht. Dies führt zu unübersichtlichen und schwer wartbaren Buildscripten.

Darüber hinaus fehlt ein konzeptioneller Ansatz zum Umgang mit **3thrd-Party** Libraries. So bleiben u.a. folgende Fragen unbeantwortet.

- Welche Bibliotheken werden in welcher Version wofür benötigt?

- Welche Abhängigkeiten habe ich zur Compile, Runtime und Test?

- Wie kann ich meine Abhängigkeiten effizient Verwalten?

- Wie kann ich Versionskonflikte zwischen den Bibliotheken einfacher identifizieren?

- Wie kann ich leichter Reproduzierbarkeit von Builds sicherstellen?

- Wie gestalte ich meine Buildskripte übersichtlich und wartungsfreundlich?

Zwar verfügt Ant mit dem Ivy Plugin mittlerweile über eine Anbindung an einen beliebigen Maven Reposity Server, jedoch ist hierzu eine zusätzliche Konfigurationsdatei, in Form der ivy.xml Datei notwendig.

Maven entstand inspiriert aus den mit Ant bekannten Problem bzw. den Unzulänglichkeiten. Ein hehrer Ansatz war, Metadaten eines Projektes an einer zentralen Stelle, der POM („Project Object Modell“) zu verwalten und feste Vorgaben bezüglich Strukturierung und Lebenszyklus eines Builds zu machen. Auch wurde das in Ant oft vermisste Dependency Management ein fester Bestandteil des Werkzeuges.

Allerdings holte die Realität bald die Entwicklung ein: Version 1 von Maven, erschienen im Jahre 2003, wurde alsbald durch die Version 2, die viele Verbesserungen auch auf konzeptioneller Ebene mitbrachte, ersetzt. Aber auch diese Version hatte ihre Tücken: eine unzureichende Dokumentation führte zu einiger Verzweiflung, wie auch den bekannten Autor und damaligen Architekten von JBoss, Bill Burk, der seine Erfahrungen mit Maven in seinem Blog in einem Satz subsumierte: „Ist too bad ist so freakin painful“. Weitere Schwachpunkte waren, nur um einige wenige zu nennen, instabile Repositories mit Artefakten von schlechter Qualität, nichtdeterministisches Verhalten der Abhängigkeitsauflösung und damit verbunden nicht reproduzierbare Builds. Im Jahre 2010 ist nun nach fast dreijähriger Entwicklungszeit Version 3 von Maven erschienen, das von Grund auf neu entwickelt wurde.

Nach Zeiten der Stagnation im Umfeld der Buildwerkzeuge betreten mit Gradle und SBT gleich zwei neue Build-Systeme die Bühne. Mit **Gradle** tritt ein Produkt an, von dem dessen Entwickler sicher sind, dass es das hält, was Maven immer versprochen hatte, aber nie auch nur ansatzweise einhalten konnte: Konsequente Umsetzung von „Convention over Configuration“ bei gleichzeitiger Transparenz und Flexibilität.

Die Buildsprache Gradle wurde von Hans Dockter initiiert und basiert auf Groovy, einer Sprache für die Java-Plattform. Ein einfaches Java Buildscript zum Compilieren und Erstellen eines JAR-Files besteht aus „applayplugin: 'java'“. Gradle setzte dabei auf Convention over Configuration und verwendet eine auf Maven basierende Verzeichnisstruktur. Durch das offene Pluginkonzeptist Gradle für diverse Sprachen wie z.B. Java, Groovy und Scala einsetzbar. Gradle ist Repository enabled und unterstützt filebasierte oder MavenRepositories. Beim hochladen der Artefakte wird der Maven POM dabei automatisch erstellt. Des Weiteren ist Gradle taskbasiert und leicht erweiterbar, so gibt es z.B. die Hooks doFirst/doLast bei den Tasks. Des Weiteren besteht im Rahmen des Buildprozesses die Möglichkeit auf das Gradle Objektmodell zuzugreifen und in den Buildablauf einzugreifen. Bei der Erstellung eines Buildscriptes werden vornehmlich Tasks konfiguriert, d.h. die Erstellung des Buildscriptes erfolgt deklarativ. Die zur Verfügung stehenden Tasktypen definieren dabei das wie. Um zu verhindern dass bei der Ausführung eines Buildscriptes Tasks mehrfach ausgeführt werden wird die Abhängigkeitsstruktur der Tasks als DAG aufgebaut. Dies ermöglicht die deterministische sequentielle Abarbeitung des Buildprozesses und inkrementelle Builds. Gradle ermöglicht die

Erstellung beliebig vieler Artefakte pro Projekt um z.B. bei Services gezielt zwischen Implementierung und Schnittstelle zu unterscheiden. Eine Unterstützung für Multi-Project Builds ist ebenso Bestandteil von Gradle. Des Weiteren ermöglicht die Ant Integration in Gradle den Zugriff auf eine breite Basis an hochwertigen Tasks. Darüber hinaus bietet Gradle u.a. noch weitere hilfreiche und konzeptionell interessante Ansätze um z.B. das Testen und Debuggen zu unterstützen und das eigentliche Rollout von Gradle zu ermöglichen.

Das „Simple Build Tool", kurz „**SBT**" genannt, stammt aus dem Scala-Umfeld. Dort existierte einerseits eine geringe bzw. initial nicht vorhandene Unterstützung von Scala durch die etablierten Build-Werkzeuge, was sich allerdings in der Zwischenzeit auch geändert hat. Aber auch Besonderheiten von Scala, wie die nicht vorhandene Binärkompatibilität zwischen den Versionen riefen nach einem insbesondere den Ansprüchen von Scala zugeschnittenem Werkzeug.

SBT ist in Scala insbesondere für Scala-Projekte entwickelt worden. Auch lassen sich damit auch klassische Java-Projekte bauen, aber der Schwerpunkt ist klar erkennbar. Auch SBT setzt dabei auf den Ansatz „Convention over Configuration", wobei z.B. die Standard-Verzeichnisstrukturen sich an denen von Maven (und damit auch denen von Gradle) orientieren. Die Konfiguration von SBT kann inzwischen durch zwei verschiedene Ansätze erfolgen. Eine Möglichkeit ist, die Konfiguration analog dem Ansatz in Gradle in Scala zu Programmieren. Dies hat den Reiz, dass einerseits der Entwickler in einer ihm „nativen" Sprache bewegt und nicht wie in Ant oder Maven mit teilweise sehr großen XML-Dateien herumhantieren muss. Andererseits wird diese Konfiguration während des Builds selbst auch compiliert und damit auf Richtigkeit im syntaktischen Sinne überprüft. In den neuen Versionen von SBT (0.10 und höher) wird zudem eine Konfiguration über eine Konfigurationsdatei möglich. Der Inhalt der Konfigurationsdatei ähnelt der Syntax der Scala-Klassen zur Konfiguration, bietet sich aber sehr gut für das Aufsetzen von Projekten an. Solange keine sehr großen Abweichungen vom Standard vorliegen dürften die meisten Projekte mit dieser Konfigurationsdatei auskommen.

Das Dependency Management beherrscht SBT selbstverständlich auch. Im Hintergrund werkelt dafür Ivy, so dass SBT mit Ivy und Maven-Repositorys gut klar kommt. Zudem können Abhängigkeiten einfach manuell ohne die Verwendung eines Repository Servers verwaltet werden.

Die Integration von SBT in die gängigen Entwicklungsumgebungen der Java/Scala-Welt lässt aber noch zu wünschen übrig bzw. existiert nur rudimentär. Es gibt inzwischen Plugins, um ein SBT-Projekt zusätzlich in ein Eclipse-Projekt zu überführen, allerdings sind die Roundtrips während der Entwicklung, wenn z.B. neue Abhängigkeiten hinzukommen, etwas umständlich verglichen mit der Integration von Maven in z.B. Eclipse.

SBT hat seine klaren Vorteile in reinen Scala-Projekten. Dort ist SBT inzwischen sogar ein „muss". Aber in der übrigen Java-Welt ist die Sichtbarkeit und die Community leider noch recht gering. SBT hat viel Potential, allerdings braucht es noch etwas Zeit, bis die volle notwendige Unterstützung vorhanden ist.

Zu Gradle läst sich sagen dass es bereits seinen festen Platz in der OpenSource Welt gefunden hat. So wird Gradle mittlerweile als Buildwerkzeug u.a. bei Hibernate und Spring Integration eingesetzt. Nicht zuletzt durch die einfache Erweiterbarkeit, die gut lesbaren und wartbaren Buildscripte und die gute Unterstützung für Multi-Modul Projekte hebt Gradle sich deutlich hervor. Bei Gradle handelt es sich somit um einen sehr aussichtsreichen Kandidaten der die Stärken von Ant und Maven in sich vereint und die Schwächen beseitigt.

Kontaktadresse:

Andreas Hartmann - Dr. Halil-Cem Gürsoy

adesso AG

Stockholmer Allee 24
D- 44269 Dortmund

Telefon:	+49 (0) 231 930-9330
Fax:	+49 (0) 231 930-9331
E-Mail	hartmann@adresse.de - Halil-Cem.Guersoy@adesso.de
Internet:	www.adesso.de