

Orientierung im Java-Dschungel

Java hat sich heute als betriebssystemunabhängige und strategische Entwicklungsplattform etabliert. Doch was versteht man heute eigentlich unter Java? Aufgrund der sehr aktiven Java-Community gibt es viele Vorstellungen darüber, was Java ist. *Philipp H. Oser*

Wenig Firmen setzen Java oder die für sie gedachte Java-Enterprise-Edition im Rohzustand ein, sondern nutzen darauf basierende Frameworks, die Java erweitern und seine Komplexität verstecken. Da es eine unüberschaubare Anzahl an solchen Frameworks gibt, ist deren Auswahl nicht trivial. Es gibt etliche Motive für solche Frameworks. Ein Ziel ist es, das Programmiermodell zu vereinfachen und fehlende Features nachzurüsten. Im reinen Java fehlt beispielsweise Unterstützung für lange andauernde Verarbeitungen (Batch Jobs), Unittests, automatische Builds oder Workflows.

Der grosse Konkurrent von Java, das .Net-Framework bietet in dieser Hinsicht einen klaren Vorteil: Fast die ganze Technologie stammt aus einem Haus, nämlich von Microsoft.

Wieso dieser Framework-Wildwuchs?

Was läuft in der Java-Welt anders? Unterschiedlich ist sicher die Popularität von quelloffener Software. Während es unzählige Open-Source-Projekte für Java gibt und diese Bewegung von Sun gefördert wird, ist diese Community bei .Net viel weniger bedeutend und Microsoft scheint Open Source im .Net-Umfeld weniger zu unterstützen. Dies

geschah beispielsweise beim offenen NUnit Framework, das Microsoft durch ein fast gleiches, im Basisframework enthaltenes Tool überflüssig machte. Demgegenüber wurden einige Java-Technologien von der Community schlecht aufgenommen oder haben an Rückhalt verloren, zum Beispiel das Pre-EJB-3-Programmiermodell oder JDO. Parallel dazu hat die Community neue Technologien hervorgebracht, die wiederum diese bemängelten Technologien ablösen (z.B. Spring anstatt dem alten EJB-Programmiermodell oder Hibernate anstatt JDO). Gleichzeitig inspirierten diese und viele andere Open-Source-Frameworks

- ▶ dem Benutzer nach erfolgter Autorisierung seine Services zur Verfügung. Um an die entsprechende Autorisierung in Form einer Assertion zu gelangen, gibt es die Rolle des Identity Providers (IDP), der für die Verwaltung der Identitäten, die Authentisierung und die Ausgabe entsprechender Berechtigungen in Form von Assertions verantwortlich ist. Die einzelnen Parteien ermöglichen durch diese konsequente Trennung die funktional gewünschte Föderierung von Identitäten.

Identity Federation mit SAML

Für Identity Federation wird heute hauptsächlich das SAML Webbrowser SSO Profile verwendet (siehe Grafik). Beim Einsatz dieses Profiles versucht der Benutzer auf einen bestimmten Service des SP zuzugreifen (1). Wenn beim SP noch kein entsprechender Sicherheitskontext vorhanden ist, muss dieser erstellt werden. Dazu ermittelt der SP den verantwortlichen IDP und erzeugt einen Authentication Request. Dieser Request kann entweder vollständig über den Browser des Benutzers mittels HTTP Post Binding (2a) oder über eine Referenz, dem sogenannten Artifact, mittels HTTP Artifact Binding (2b) zum IDP transferiert werden. Bei Letzterem erfolgt die Bindung zwischen SP und IDP über einen zusätzlichen direkten Kanal (SAML über SOAP). So kann vermieden werden, dass die Assertion über den Benutzer transferiert wird, was eine

XML-Verschlüsselung der sensitiven Daten erfordern würde. Der direkte Kanal wird dann über den Transport Layer gesichert. In diesem Fall sieht der Benutzer nur eine Referenz auf die entsprechende Assertion. Im nächsten Schritt muss der IDP den Benutzer authentifizieren, falls dies nicht bereits im Rahmen eines SSO erfolgt ist. Diese Authentifizierung ist vollkommen unabhängig vom SAML-Standard. Nach einer erfolgreichen Identifizierung erstellt der IDP die Authentication Response mit der dazugehörigen Assertion. Auch hier kann die Übertragung zum SP vollständig über den Browser des Benutzers (3a) oder über den direkten Kanal (3b) erfolgen. Der SP kann nun die Assertion überprüfen und dem Benutzer Zugriff auf den gewünschten Service gewährleisten (4).

SAML 2.0 im Einsatz

Heute unterstützen die meisten kommerziellen Java EE Container neueren Datums SAML 2.0. Im Einsatzfall lohnt es sich, die genaueren Details zu studieren. Oft werden gewisse Konfigurationsparameter erwartet, die über das Notwendige hinausgehen oder es sind nur Teilbereiche des Standards umgesetzt. Eine baldige Verbesserung in dieser Hinsicht – auch dank der OpenSAML-2.0-Implementierung – ist jedoch absehbar.

Generell muss die Skalierbarkeit des IDP gewährleistet sein, wobei gewisse Faktoren die

Leistungsfähigkeit entscheidend beeinflussen: Durch die Verwendung von XML und das damit verbundene grössere Token entsteht ein Mehraufwand für das Erstellen, Parsen und Encoding (Base-64-Kodierung von Feldern) von Identitätsinformationen, was entsprechend berücksichtigt werden muss. Zusätzlicher Rechenaufwand kann auch daher resultieren, dass sowohl die Response wie auch einzelne Assertions innerhalb der Response signiert werden können. Somit kann ein Teil der gesamten Information doppelt signiert sein. Dies wird unter Umständen sogar vom Container des SP verlangt. Die Signierung einer einfachen Assertion kann seine Grösse verdoppeln – bei einer zusätzlichen Verschlüsselung sogar verdreifachen. Der Standard erlaubt es weiter, neben ganzen Assertions nur bestimmte IDs und Attribute zu verschlüsseln. Auch dies führt zu zusätzlichem Aufwand.

Sind diese Hürden genommen, überwiegen die Vorteile von SAML 2.0, um die produktunabhängige Föderierung von Service-Infrastrukturen zu ermöglichen. Der Einsatz von SAML hat für Serviceanbieter auch den Vorteil, dass bestehende Benutzerpopulationen wiederverwendet und Kosten für die Registrierung und Aktivierung gesenkt werden können. Und nicht zuletzt lassen sich Legacy-Systeme mit proprietärer Benutzerverwaltung über eine adäquate Sicherheitsinfrastruktur in eine SAML-basierte Umgebung integrieren.