Middleware-Konzepte zur Verteilung von komponentenbasierten Anwendungen im Netzwerk

Ralf Vandenhouten, Thomas Kistel

Zusammenfassung

Dieser Beitrag untersucht Konzepte zur Verteilung von Softwareanwendungen auf Basis des OSGi-Standards im Netzwerk. Darin wird mit R-OSGi eine Lösung vorgestellt, die transparente Anwendungsentwicklung gegenüber der verwendeten Netzwerktechnologie ermöglicht. Dabei wird auch darauf eingegangen, wie netzwerkspezifische Charakteristiken (höhere Latenzzeiten, Übertragungsfehler oder Ausfall des Netzwerkes) agnostisch auf die höheren Softwareschichten abgebildet werden können.

Abstract

This article describes a concept for distributing network software applications based on the OSGi standard. The reader will be introduced into the R-OSGi solution, which allows the transparent development of applications with respect to the network technology and topology in use. The article also demonstrates how network specific characteristics (e. g. latency, transmission errors or network failure) can be mapped to the higher application layers in an agnostic way.

1 Einleitung

Wenige Softwarekonzepte haben eine solch unauffällige und gleichzeitig nachhaltige Erfolgsgeschichte vorzuweisen wie das Framework der vor 10 Jahren gegründeten OSGi Alliance (früher Open Services Gateway Initiative). Während ungezählte Hypes am IT-Horizont erschienen und zumeist schnell wieder untergingen, verrichtete OSGi unter der Motorhaube von Kraftfahrzeugen, in der Industrie- und Gebäudeautomation oder in Mobilfunkgeräten zuverlässig seinen Dienst. Bei eingebetteten Systemen gilt OSGi heute als einer der führenden Softwarestandards. Als Basis der populären Entwicklungsumgebung Eclipse, die seit Version 3.0 aus OSGi-Komponenten besteht, hat sich die OSGi-Technologie auch als stabiles Fundament einer Rich Client Platform bewährt und damit nachgewiesen, dass ihr Geltungsanspruch deutlich über Embedded Systeme hinausgeht.

Der Erfolg liegt in der Einfachheit und Erweiterbarkeit des Frameworks begründet. Das Komponentenmodell sieht vor, Software in leichtgewichtigen Modulen (so genannten Bundles) zu entwickeln. Die OSGi-Plattform steuert den gesamten Lebenszyklus dieser Bundles und ermöglicht, diese zur Laufzeit zu installieren, zu starten, anzuhalten oder zu entfernen. Das ist insbesondere für Systeme wichtig, die rund um die Uhr laufen müssen. Bundles können Standarddienste des OSGi-Frameworks oder von anderen Bundles bereit gestellte Dienste nutzen (siehe Abb. 1). Um einen solchen Service zur Verfügung zu stellen, muss ein Bundle ein entsprechendes Java-Interface beim Framework registrieren. An das Interface werden keine besonderen Anforderungen gestellt, jedes POJO (Plain Old Java Object) ist dafür geeignet. Möchte ein Bundle einen bestimmten Service nutzen, kann es bei der so genannten Service-Registry des Frameworks dessen Verfügbarkeit abfragen und sich ggf. Referenzen auf die verfügbaren Instanzen geben lassen.

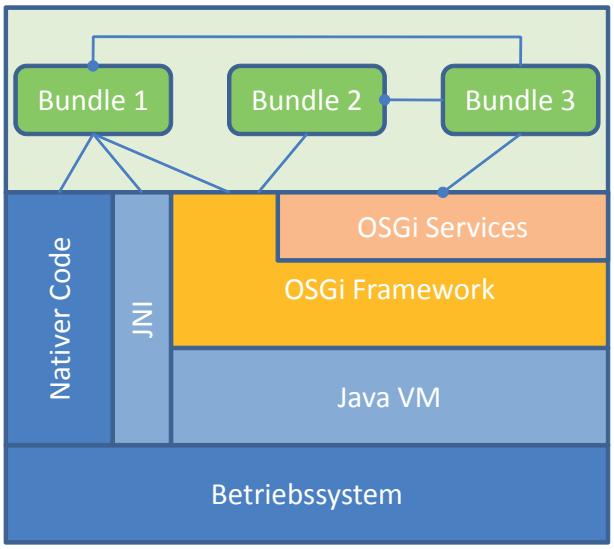

Abb. 1: Bundles nutzen die Dienste des OSGi Frameworks und solche, die von anderen Bundles registriert wurden

Dieser serviceorientierte Ansatz ist flexibel und macht eine der Stärken von OSGi aus, ist jedoch noch keine serviceorientierte Architektur im Sinne des heute oft verwendeten Begriffs SOA, da bei OSGi sämtliche Bundles in derselben virtuellen Maschine und somit auf demselben Host laufen. Zwar gibt es zahlreiche OSGi-Services, die Netzwerkkommunikation unterstützen und über ihre jeweiligen Protokolle (z. B. HTTP, FTP oder SOAP) auch aus der Ferne ansprechbar sind. Orts- und Zugriffstransparenz von Serviceschnittstellen jedoch, wie sie bei einer SOA zur Verteilung von Unternehmensanwendungen erwartet werden, sind damit noch nicht realisiert. Erst die gerade veröffentlichte OSGi-Spezifikation 4.2 sieht mit Distributed OSGi die transparente Verteilung von Diensten im Netz mithilfe einer geeigneten Middleware vor (siehe 6 »Distributed OSGi, ECF und R-OSGi«, S. 37). Eine Middleware-Plattform, die eine transparente Verteilung von OSGi-Diensten auch mit älteren OSGi-Containern vor Version 4.2 ermöglicht, ist das am Institut für Informations- und Kommunikationssysteme der ETH Zürich entwickelte R-OSGi. Dessen Funktionsweise sowie eine Einführung in die praktische Arbeit damit sollen Gegenstand dieses Artikels sein.

2 Grundkonzepte von R-OSGi

Die Zielsetzung von R-OSGi ist es, für beliebige OSGi-Implementierungen einen transparenten, verteilten Zugriff auf die Services zu ermöglichen und dabei folgende Anforderungen zu erfüllen (Rellermeyer et al. 2007):

1. Nahtlose Einbettung in OSGi: Aus Sicht eines Bundles sollen lokale und entfernte Dienste nicht unterscheidbar sein. Existierende OSGi-Anwendungen sollen ohne Anpassungen verteilbar sein.
2. Zuverlässigkeit: Anwendungsentwickler sollen nicht mit neuen Fehlermustern konfrontiert werden, sondern vom Netzwerk verursachte Fehler wie übliche Dienstausfälle behandeln können.
3. Allgemeingültigkeit: R-OSGi beschränkt sich nicht auf bestimmte Dienste. Jeder gültige OSGi-Service soll auch remote verfügbar sein können.
4. Portabilität: Die Middleware soll auch auf kleinen Embedded-Geräten, wie Mobiltelefonen, lauffähig sein. Der mit R-OSGi verbundene Ressourcenverbrauch muss deshalb bescheiden sein.
5. Adaptivität: Die Verwendung von R-OSGi ist nicht verbunden mit Rollenzuordnungen, wie Client oder Server. Die Beziehung zwischen Modulen im verteilten Kontext ist grundsätzlich symmetrisch.

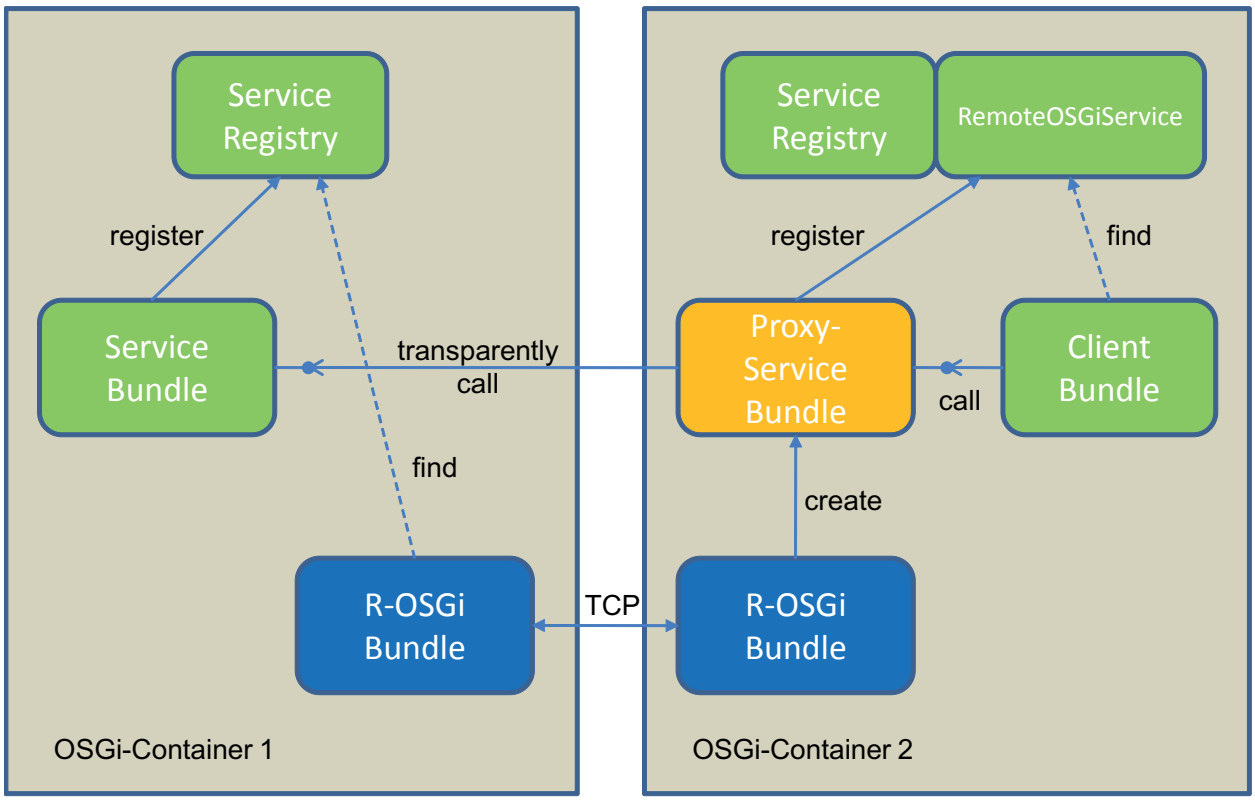

Abb. 2: Clients erhalten lokalen Zugriff auf einen von R-OSGi erzeugten Proxy-Service, der Aufrufe transparent an den eigentlichen Remote Service weiterleitet

6. Effizienz: R-OSGi soll schnell sein, vergleichbar mit der hochoptimierten RMI-Implementierung von Java 5.

Realisiert wird R-OSGi als gewöhnliches OSGi-Bundle, das auf allen OSGi-Containern gestartet werden muss, die am verteilten Szenario beteiligt sind. R-OSGi sucht jeweils im lokalen Container nach Services, für die die Property RemoteOSGIService.R_OSGi_REGISTRATION gesetzt ist und informiert den Gegenpart über das Ergebnis. Auf der Remote-Seite wird dann für jeden gefundenen Dienst von R-OSGi ein dynamisches Proxy-Bundle erzeugt und registriert, das die Schnittstelle des ursprünglichen Dienstes nachbildet. Es implementiert sie jedoch nicht selbst, sondern leitet alle Aufrufe potenzieller Clients transparent über den entsprechenden R-OSGi-Kanal an den »echten« Service und liefert den Rückgabewert zurück an den Client (siehe Abb. 2). Für den Client ist damit nicht unterscheidbar, ob der Dienst in seinem eigenen Container oder in der Ferne geleistet wird.

Die Mission, Services transparent zu verteilen, wäre damit erfüllt. Anders als bei Distributed OSGi haben sich die Entwickler von R-OSGi aber entschieden, die Clientseite nicht vollkommen agnostisch gegenüber dem Netzwerk zu gestalten. Ein Client sieht nur die Services, mit deren Container sich das Client-Bundle zuvor per R-OSGi verbunden hat. Dies kann entweder direkt durch explizite Angabe eines URI erfolgen oder mithilfe eines Discovery Service wie SLP (Service Location Protocol), der im Netz nach möglichen Anbietern sucht. Die Referenz auf den entfernten Dienst wird auch nicht – wie bei lokalem Zugriff – vom BundleContext geliefert, sondern vom RemoteOSGiService der Middleware. Dem größeren Aufwand für die Konfiguration, der mit diesem Konzept verbunden ist, steht eine stärkere Kontrolle der Verdrahtung gegenüber, die sich bei komplexen Szenarien positiv auf die Skalierbarkeit auswirken kann.

Generell stehen Anwendungen in verteilten Systemen vor größeren Herausforderungen als solche, die in nur einem Adressraum ablaufen, selbst wenn sie einem transparenten Verteilungsmodell folgen. Dies liegt an den Problemen, die durch miteinander vernetzte Rechnerknoten hinzukommen können. Dazu gehören Unterbrechungen der Netzwerkverbindung oder der Ausfall einzelner Knoten, aber auch höhere Antwortzeiten durch Latenzen im Netz, Verlust von Nachrichten oder nicht-deterministisches Verhalten des Gesamtsystems. Diese Probleme sind grundsätzlicher Natur und können auch durch R-OSGi nicht verhindert oder gelöst werden. Der Vorteil von OSGi allgemein ist aber, dass sich Bundles ohnehin nicht auf die Verfügbarkeit von Diensten verlassen können, da diese jederzeit angehalten oder sogar deinstalliert werden können. Clients müssen mit solchen unvorhersehbaren Ausfällen rechnen und umgehen können. Dabei helfen in der Regel ServiceTracker bzw. Komponentenklassen im Fall von Declarative Services. R-OSGi macht sich dies zunutze, indem es Fehler im Zusammenhang mit der Verteilung, wie beispielsweise eine Netzwerkunterbrechung, auf einen Dienstausfall abbildet und das zugehörige Proxy-Bundle deinstalliert. Das Client-Bundle muss für einen solchen Fall auch bei lokalem Zugriff eine Alternative parat haben und sollte damit klar kommen. Bewusstsein für die Serviceverteilung im Netzwerk benötigt es dafür jedoch nicht, so dass OSGi-Entwickler nicht umgeschult werden müssen.

3 Bereitstellung von Klassen im verteilten System

OSGi-Bundles können Java-Pakete aus anderen Bundles verwenden, wenn sie deren Import explizit in ihrem Bundle-Manifest deklarieren. Dies hat Konsequenzen für die Generierung von Proxy-Bundles. Es kann sein, dass die Service-Schnittstelle Klassen als Methodenparameter oder Rückgabewerte verwendet, die nicht zum Java-Standard gehören und auf der Client-Seite nicht bekannt sind. Da der Service-Proxy sie aber im Client-Container benötigt, stellt R-OSGi sie dort mithilfe von Type Injection bereit. Dafür analysiert das R-OSGi-Bundle auf der Service-Seite die Schnittstelle und trägt alle direkt oder indirekt benötigten Klassen in eine Injection-Liste ein, die mit dem Service registriert wird. Sobald ein Client den Service akquiriert, erzeugt R-OSGi die erforderlichen Klassen im Proxy-Bundle. Für die dynamische Proxy-Generierung bedient sich R-OSGi der ausgereiften Open-Source-Bibliothek ASM des ObjectWeb-Konsortiums. ASM ist ein Framework zur Manipulation, Analyse und Generierung von Java-Bytecode und wird auch in vielen anderen Bibliotheken und Frameworks verwendet.

4 Ereignisbenachrichtigung

Kaum eine Anwendung kommt ohne asynchrone Benachrichtigung aus. In Stand-alone-Applikationen wird dafür meist das Beobachtermuster verwendet. Im

OSGi-Kontext ist es nicht zu empfehlen, wenn es über Bundle-Grenzen hinweg zum Einsatz kommen soll, da es den dynamischen Lebenszyklus von Bundles nicht berücksichtigt und deshalb zu Problemen führen kann (Kriens et al. 2004). Noch schlimmer sieht es im Fall von R-OSGi (und übrigens auch Distributed OSGi) aus. Dies liegt daran, dass in einer verteilten Umgebung alle Fernaufrufe als Call-by-Value (im Gegensatz zum Java-Standard Call-by-Reference) ausgeführt werden müssen. Mit Referenzen kann die entfernte Gegenstelle wenig anfangen, deshalb müssen Kopien der Objekte übertragen werden. Würde sich nun nach dem Beobachtermuster ein Beobachter bei seinem entfernten Subjekt registrieren, würde dieses Subjekt nur eine Referenz auf einen kopierten Beobachter in seinem Adressraum speichern. Im Benachrichtigungsfall würde nur die Beobachter-Kopie informiert, nicht der ursprüngliche Beobachter im anderen Container.

Die Lösung für dieses Problem heißt EventAdmin-Service. Es handelt sich dabei um einen Standard-Service der OSGi Compendium-Spezifikation, der asynchrone und synchrone Benachrichtigungen zwischen beliebigen Bundles ermöglicht. Ein Sender von Benachrichtigungen legt eine Rubrik dafür fest, das so genannte Topic. Wer diese Nachrichten bekommen möchte, muss einen EventHandler für das entsprechende Topic registrieren. Dieses Muster funktioniert auch mit R-OSGi, sogar ohne besondere Vorkehrungen. R-OSGi setzt den EventAdmin-Service transparent im Netzwerk um, so dass Nachrichten eines Containers auch die anderen Container erreichen (siehe Abb. 3). Dafür muss lediglich auf jedem Container eine EventAdmin-Implementierung laufen.

Für die Datenübertragung im Netzwerk benutzt R-OSGi ein eigenes Protokoll über so genannte Network-Channels. Zwar kommt standardmäßig eine TCP-Implementierung (TCPChannel) zum Einsatz, diese kann jedoch leicht ausgetauscht werden. Auf diese Weise können auch mit Bluetooth oder dem Mina-Framework solche Kanäle realisiert werden, was beispielsweise die drahtlose R-OSGi-Anbindung von Mobiltelefonen ermöglicht.

5 Einsatz in der Praxis

Inzwischen setzen verschiedene kommerzielle und nicht-kommerzielle Anwendungen R-OSGi ein, wie beispielsweise die Collaborative Middleware flowSGI für mobile Endgeräte oder das Baukastensystem BUG, mit dem Embedded Devices entwickelt werden können. Die ixellence GmbH verwendet R-OSGi ebenfalls bei der Entwicklung ihrer verteilten Softwareprodukte. Im medizinischen Bereich überträgt das Telemonitoringsystem *ixTrend* hochabgetastete medizinische Echtzeitdaten via R-OSGi. Auch umfangreiche Konfigurations- und Steuerungsprozesse der verteilten medizinischen Netzwerkkomponenten werden über Remote-Services mittels R-OSGi realisiert. Im Bereich der Bildverarbeitung setzt ixellence ebenfalls R-OSGi ein. Die Videoüberwachungssoftware *ixCam* ermöglicht unter anderem die automatische Sabotagedetektion von Überwachungskameras. Die Kameras sind dabei an spezielle Kameraserver angeschlossen, auf denen die Algorithmen zur Erkennung der Kameramanipulation ausgeführt werden. Hier übernimmt R-OSGi die Ereig-

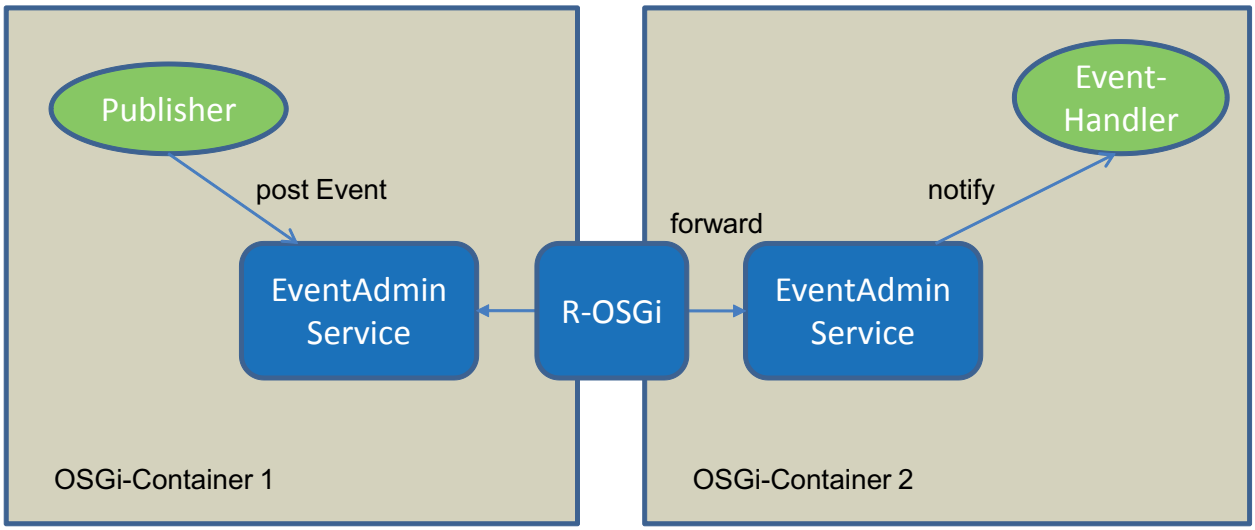

Abb. 3: Ereignisse werden von R-OSGi netzwerktransparent mithilfe der lokalen EventAdmin-Services weitergeleitet

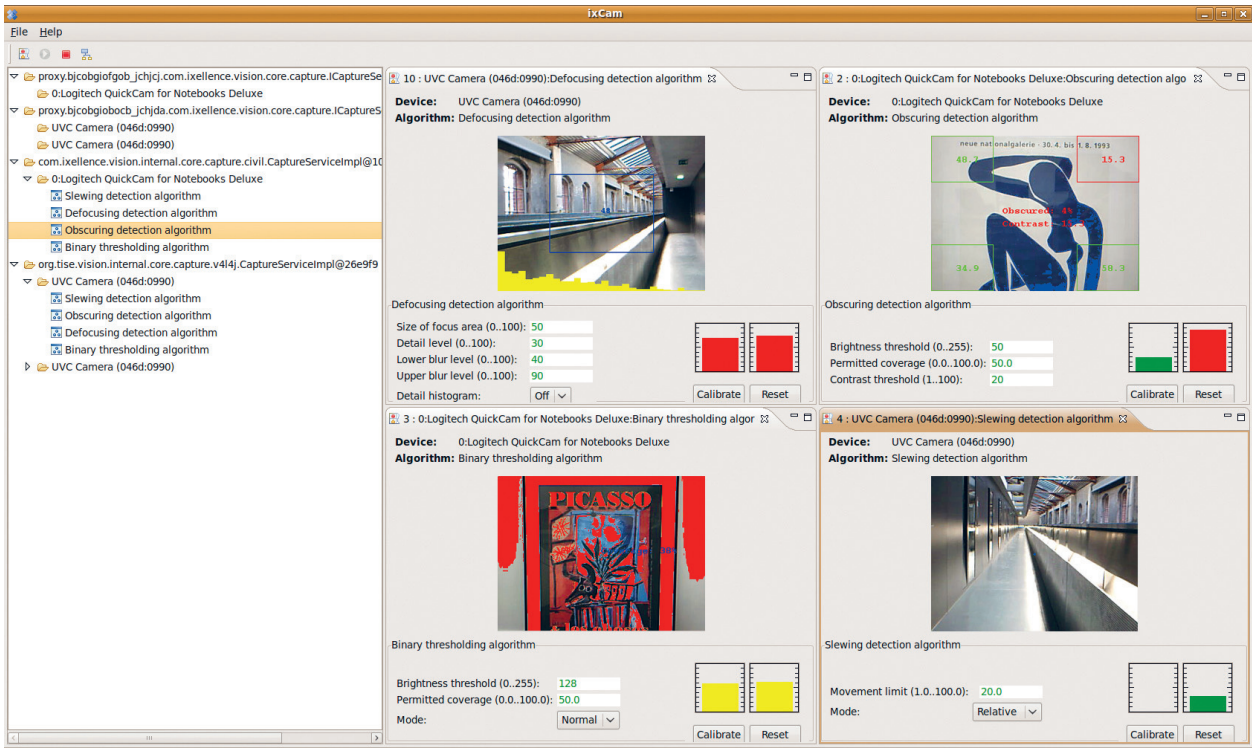

Abb. 4: Bei der Videoüberwachung mit ixCam übertragen die Kameraserver Bilddaten und Alarmereignisse via R-OSGi zur graphischen Oberfläche in der Wachzentrale.

nisbenachrichtigung im Alarmfall sowie die Übertragung der Echtzeitbilddaten von den Kameras zu den Client-Stationen (siehe Abb. 4). Die Autoren haben den aktuellen Release Candidate 4 der R-OSGi-Version 1.0.0 in diesen Projekten bereits als zuverlässig, leistungsfähig (vergleichbar mit RMI) und somit tauglich für den Produktiveinsatz erlebt. Nur die im zugehörigen Forum dokumentierten Probleme sollten bis zur endgültigen Freigabe noch gelöst werden.

6 Distributed OSGi, ECF und R-OSGi

Neben der Entwicklung von R-OSGi gab es weitere Entwicklungen im Bereich der OSGi-Technologie zur Verteilung von Anwendungen und dessen Dienste. Die jüngst im Service-Compendium der OSGi-Serviceplattform von der OSGi Alliance veröffentlichte Spezifikation zu Remote Services beschreibt eine Erweiterung des Standards für verteilte OSGi-Anwendungen (Distributed OSGi). Bei der Spezifikation von Distributed OSGi wurde bewusst darauf geachtet, keine zusätzliche Middleware-Lösung zu schaffen, da es diesbezüglich bereits eine Reihe ausgereifter Konzepte gibt. Vielmehr definiert Distributed OSGi (D-OSGi) eine Möglichkeit, existierende Middleware-Lösungen, wie SOAP, CORBA, JMS oder R-OSGi, auf standardisierte Art und Weise in das OSGi-Framework zu integrieren und dabei die bewährten Konzepte der Service-Konsumption beizubehalten. Die Aufgabe der Verteilung von Services über eine existierende Middleware übernehmen bei D-OSGi die **Distribution Provider**. Dafür stellen sie einen **Endpoint** bereit, über den Services importiert und exportiert werden können. Der Endpoint ist dabei der Zugriffspunkt der Service-Kommunikation über lokale OSGi-Grenzen hinweg. Ein OSGi-Framework kann mehrere Distribution Provider gleichzeitig nutzen, die jeweils unabhängig verschiedene Services importieren und exportieren. Mit diesem Konzept lässt sich beispielweise eine Anwendung für Buchbestellungen entwickeln, welche für den Abruf der Buchkataloge einen Distribution Provider für Webservices nutzt und für die Bestellabwicklung einen anderen Provider. Der Import und Export von Remote Services erfolgt bei D-OSGi über **Remote Service Properties**, die beim Registrieren und Referenzieren von Services benutzt werden können. Über **Intents** können bestimmte Anforderungen definiert werden, die die entfernten Dienste unterstützen müssen. Das Konzept der Intents wurde von der *SCA Policy Framework Specification* (SCA) übernommen. Hierüber können unter anderem Angaben zur Verschlüsselung oder Dienstgüte der Übertragung des Distribution Providers gemacht werden.

Neben Apache CXF (Apache CXF) stellt das Eclipse Communication Framework (ECF) eine Implementierung eines Distribution Providers für D-OSGi dar. ECF ist ein Framework für die Entwicklung von verteilten Eclipse-Anwendungen, das asynchrone Punkt-zu-Punkt- oder Publish-and-Subscribe-Kommunikation unterstützt. Die Installation von ECF in die Eclipse Entwicklungsumgebung ermöglicht Echtzeitkommunikation und stellt Teamfunktionen bereit, wie das gemeinsame Editieren von Quellcode. Grundlage dafür ist eine Reihe von APIs und Frameworks, die auf bestehenden Protokollen (XMPP, IRC, BitTorrent, MSN etc.) aufbauen. Jedes dieser Protokolle ist eine Implementierung eines Communication Containers, die bei ECF den transparenten Zugriff auf einen protokollspezifischen Kontext liefern. ECF liefert außerdem einen Communication Container für das an der ETH Zürich entwickelte R-OSGi.

R-OSGi ist keine direkte Implementierung für D-OSGi, sondern stellt eine Middleware bereit, mit der OSGi-Services ebenfalls remote zur Verfügung gestellt und abgerufen werden können. Auch bereits existierende Services können bei R-OSGi über eine *Surrogate-Registration* exportiert werden. Finden kann man Remote-Services mit einem Discovery-Service über *ServiceDiscoveryListeners*. R-OSGi kann auch mit älteren OSGi-Frameworks arbeiten, sofern auf die Ereignisbenachrichtigung durch den EventAdmin-Service verzichtet werden kann. Für OSGi R3-Implementierungen, die EventAdmin benötigen, kann auf den EventAdmin-Backport, der Teil der OSGi-Implementierung Concierge (Concierge) ist, zurückgegriffen werden.

7 Fazit

Wer bereits mit OSGi vertraut ist, erhält mit R-OSGi eine schlanke und leistungsfähige Middleware, mit der sich auch die verteilte Softwarewelt erobern lässt, ohne dass man dafür umdenken muss. Eine Einarbeitung ist dennoch erforderlich, nicht zuletzt wegen der ungewohnten Call-by-Value-Semantik. R-OSGi beeinflusst letztlich auch die Architektur der Anwendung, denn in verteilten Anwendungen wird der Grundsatz, Schnittstellen einfach und klein zu halten, zur leistungsentscheidenden Notwendigkeit. Es bleibt abzuwarten, wie sich die ersten Implementierungen der OSGi-Spezifikation 4.2 in Bezug auf Distributed OSGi bewähren und welchen Einfluss dies auf die Konsolidierung der konkurrierenden Ansätze haben wird.

Acknowledgement

Der Artikel ist das Ergebnis von Forschungsarbeiten im Bereich der Telematik im Zusammenhang mit einer an der Technischen Hochschule Wildau [FH] durchgeführten Bachelorarbeit, deren Einsatz bereits in verschiedenen Softwareprojekten erprobt wurde. Es werden die Grundkonzepte, praktische Fallstricke und Lösungsmöglichkeiten sowie die neuesten Bestrebungen zur Standardisierung dieses Softwaredesigns vorgestellt.

Literatur

Rellermeyer, Jan S.; Roscoe, Gustavo Alonso Timothy (2007): R-OSGi: Distributed Applications through Software Modularization. In: Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference, Newport Beach, CA.

Kriens, Peter; Hargrave, B. J. (2004): Listeners Considered Harmful: The Whiteboard Pattern. OSGi Alliance 2004. http://www.osgi.org/wiki/uploads/Links/whiteboard.pdf.

Konradi, Philips; Ransmayr, Viktor; Wengatz, Nicole (2009): OSGi goes Enterprise? In: JavaSpektrum 4/2009, 13-17.

Wütherich, Gerd; Hartmann, Nils; Bolb, Bernd; Lübken, Matthias (2008): Die OSGi Service Plattform; dpunkt-Verlag, Heidelberg, 1. Auflage.

Bressler, Tobias (2009): Design und Implementierung einer verteilten Client-Server-Anwendung auf Basis des OSGi Frameworks zur Analyse und Darstellung von medizinischen Echtzeitdaten. Bachelorarbeit, TFH Wildau.

Vandenhouten, Ralf; Kistel, Thomas (2009): Aus der Entfernung – Verteilte Dienste mit R-OSGi. In: iX Magazin für Professionelle Informationstechnik, 12/2009.

SCA Webseite: http://www.oasis-open.org/committees/sca-policy

Apache CXF Webseite des Projektes: http://cxf.apache.org/distributed-osgi.html.

ECF Webseite: http://www.eclipse.org/ecf.

R-OSGi Webseite: http://r-osgi.sourceforge.net.

Concierge Webseite der OSGi-Implementierung Concierge: http://concierge.sourceforge.net.

OSGi Webseite der OSGi-Alliance: http://www.osgi.org.

ASM Webseite des Projektes: http://asm.ow2.org.

BUG Webseite des Baukastensystems BUG: http://www.buglabs.net.

Autoren

Prof. Dr. rer. nat. Ralf Vandenhouten
TH Wildau [FH]
Fachbereich Ingenieurwesen/Wirtschaftsingenieurwesen
Fachgebiet Telematik
ralf.vandenhouten@tfh-wildau.de
Tel. +49 3375 508-359

Thomas Kistel, M. Eng.
TH Wildau [FH]
Fachbereich Ingenieurwesen/Wirtschaftsingenieurwesen
Fachgebiet Telematik
thomas.kistel@tfh-wildau.de
Tel. +49 3375 508-615