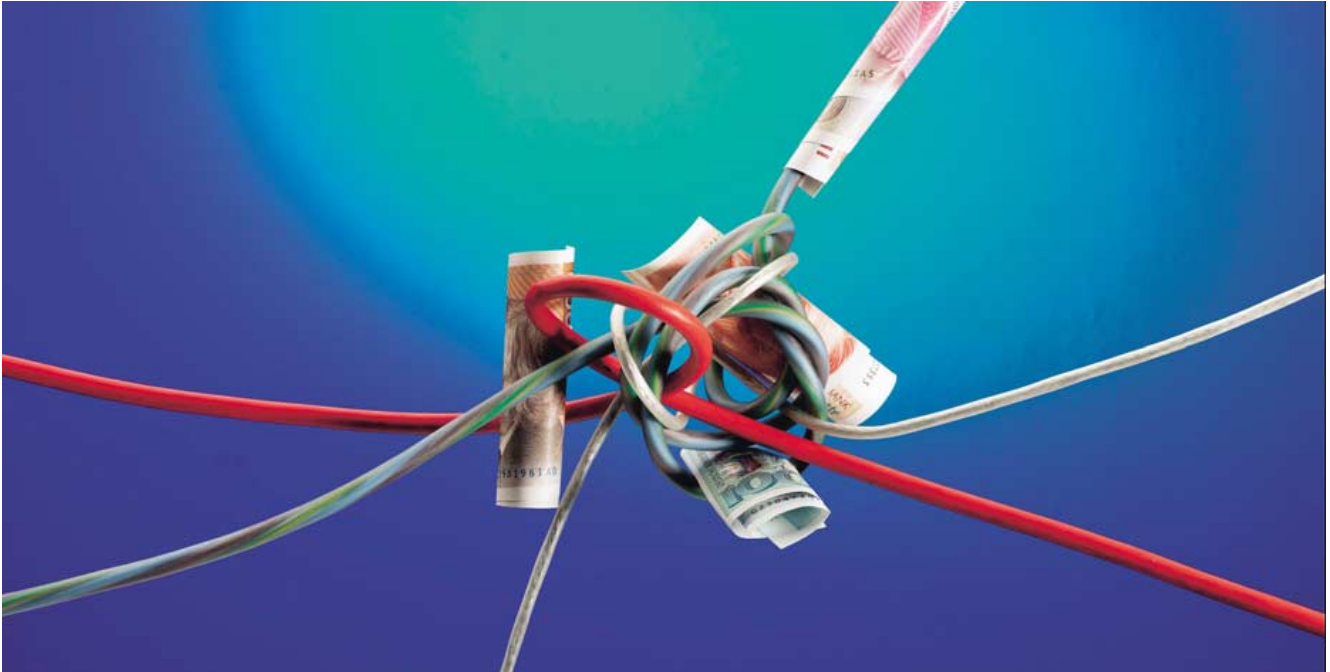


Connector Architecture: Einbindung von Legacy-Systemen

Grenzenlose Kooperation



Suns Handbuch für die J2EE-Programmierung (Java 2 Enterprise Edition) beschreibt die Integration der vorhandenen Enterprise-Information-Systeme (EIS) in E-Commerce-Anwendungen als die größte Herausforderung für die Programmierer. Die Connector Architecture, deren öffentlicher Entwurf vor kurzem im Rahmen des Java Community Process vorgestellt wurde, soll diese Arbeit erheblich erleichtern (Kasten 'Infos im Web').

Das Ziel der Spezifikation ist hoch gesteckt: Sie soll einen Standard festklopfen, der es erlaubt, beliebige IT-Systeme – etwa Enterprise-Resource Planning Software (ERP), Transaktionsmonitore und Datenbanken – skalierbar, transaktional und sicher in einen J2EE-konformen Server zu integrieren. So genannte Resource Adapter bilden den Kern der Architektur.

Ein Adapter ließe sich sowohl an einen Application

Server als auch an eine Anwendungskomponente anklippen. Seine volle Leistung wird er aber nur in Kooperation mit einem Application Server ausspielen. In diesem Fall arbeiten Adapter und Server eng zusammen, um die Systemdienste für Transaktionen, Sicherheit und Verbindungsaufbau bei der Integration des EIS bereitzustellen.

Für die Programmierung des Adapters ist der jeweilige

Stefan Krieger

Elektronische Geschäfte florieren erst, wenn auch die bewährten Altsysteme daran teilnehmen. Deren Integration ist jedoch schwierig. Mit Hilfe der J2EE-Connector-Architektur sollen Entwickler heterogene IT-Systeme standardisiert in E-Commerce-Anwendungen einbinden können.

EIS- oder Werkzeughersteller selbst zuständig. Der Lieferant des Application Server muss nur einmalig die spezifizierten Schnittstellen implementieren. Durch diese Standardisierung kann er jedes IT-System, vorausgesetzt es ist dafür ein Adapter erhältlich, in seinen Server integrieren. (Abb. 1).

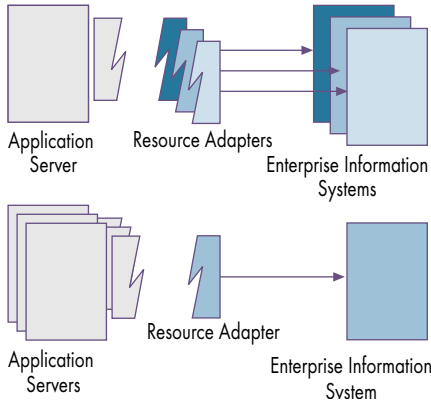
Damit die beschriebene Zusammenarbeit reibungslos funktioniert, vereinbart die Connector Architecture meh-

rere Verträge zwischen Resource Adapter und Application Server, die für beide Seiten die Schnittstellen sowie ihre Semantik festlegen. Der Adapter erfüllt dabei diese Vereinbarungen stellvertretend für das zu integrierende EIS (Abb. 2).

Verträge regeln die Kommunikation

Der *Connection Management Contract* ermöglicht dem Application Server, einen Pool für Verbindungen zum EIS aufzubauen. Diese Verbindungen sind wieder verwendbar; der oft zeitaufwändige Verbindungsaufbau lässt sich damit optimieren.

Für den transaktionssicheren Zugriff auf das EIS soll der *Transaction Management Contract* sorgen. Dazu teilt der Resource Adapter dem Application Server mit, ob, und wenn ja, welche Transaktionen er unterstützt: entwe-



Grenzlose Verbindung: Jeder Resource Adapter kann sich über definierte Schnittstellen in jeden Application Server einlinken (Abb. 1).

der verteilte und/oder lokale (siehe Kasten 'Demokratische Ressourcen').

Zur Durchführung einer verteilten Transaktion benutzt der Application Server den Transaktionsmanager des Java Transaction Service (JTS). Dadurch ist es beispielsweise möglich, dass der Zugriff über einen Adapter auf eine Mainframe-Datenbank in derselben Transaktion abläuft, die auch eine EntityBean in einer SQL-Datenbank speichert. Unterstützt das EIS keine verteilten Transaktionen, kann der Adapter die lokale Variante anbieten. Sie erlaubt, mehrere Aktionen in einer logischen Arbeitseinheit zusammenzufassen, die sich aber nur über diese eine Ressource erstreckt.

Der *Security Contract* ist, der Name legt es nahe, für die Sicherheit zuständig. Dieser Vertrag definiert Mechanismen zur Authentifizierung gegenüber dem EIS. Der Austausch von sicherheitsrelevanten Informationen zwischen

Application Server und Adapter wird über die Schnittstellen und Klassen des Java Authentication and Authorization Service (JAAS) abgewickelt.

Die notwendigen Informationen zur Installation eines Resource Adapter erhält der Server von einem speziellen Deployment Descriptor. Ist der Adapter erfolgreich im Server installiert, wird eine *ConnectionFactory* beim Java Naming and Directory Service (JNDI) registriert. Via *ConnectionFactory* bekommt die Anwendungskomponente, zum Beispiel eine Enterprise JavaBean, ihre Verbindung zum EIS.

Nicht alles lässt sich festlegen

Über den Anschluss kann die Komponente RPCs (Remote Procedure Call) an das EIS definieren, absetzen und die Ergebnisse empfangen. Die Schnittstelle, die der Re-

Demokratische Ressourcen

Eine verteilte Transaktion unterscheidet sich von einer 'normalen', lokalen dadurch, dass sie sich über mehr als eine Datenbank beziehungsweise Ressource erstrecken kann. Die Verwaltung der Transaktion übernimmt ein Transaktionsmanager, bei dem sich alle beteiligten Ressourcen anmelden.

Sind mehrere Ressourcen an einer Transaktion beteiligt, wird das Zurückschreiben von Daten (*commit*) nach dem so genannten 2-Phasen-Commit-Protokoll durchgeführt. Der Transaktionsmanager leitet in der ersten Phase eine Abstimmung unter allen beteiligten Ressourcen über den Ausgang der Transaktion ein. Jede Ressource prüft für sich, ob sie die erfolgreiche Durchführung der Transaktion garantieren kann und stimmt in die-

sem Sinne ab. Entscheiden sich alle positiv, sendet der Transaktionsmanager in der zweiten Phase die Aufforderung zur endgültigen Festschreibung der Transaktion. Im anderen Fall erhalten alle Beteiligten den Befehl zum Wiederherstellen des konsistenten Zustandes vor der Transaktion (*rollback*).

Für die Kommunikation zwischen Transaktionsmanager und Ressource ist die XA-Schnittstelle der X/Open Group der gültige Standard. Viele Datenbanken und Messaging-Systeme unterstützen ihn. Auch der Java Transaction Service folgt XA und verlangt vom Treiber der Ressource, dass er zur Durchführung einer verteilten Transaktion die Schnittstelle *javax.transaction.xa.XAResource* implementiert hat.

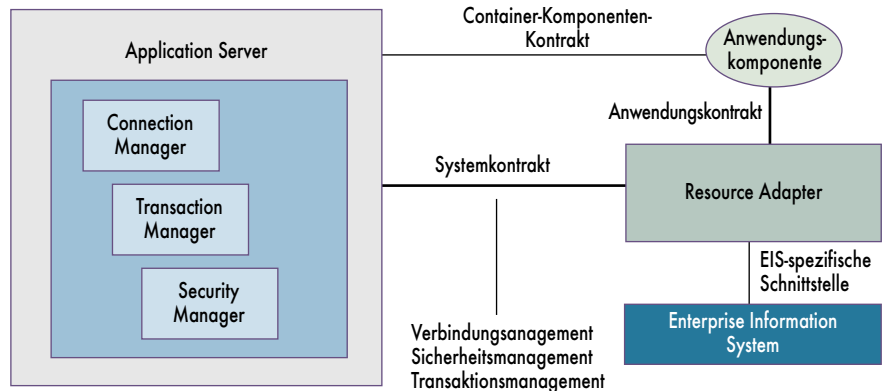
source Adapter dafür anbietet, ist in der Connector Architecture nicht spezifiziert, und zwar aus gutem Grund: Der Adapter soll einem Client in erster Linie eine einfach zu benutzende Schnittstelle zur Verfügung stellen. Weil es viele verschiedene IT-Systeme gibt, muss sich die Connector Architecture notwendigerweise auf einem hohen Abstraktionsniveau bewegen. Eine generelle Schnittstelle zu spezifizieren, gegen die sich auch noch einfach programmieren ließe, ist deshalb kaum möglich.

Daher lässt die Spezifikation dem Adapterhersteller freie Hand beim Entwurf der Schnittstelle *Connection*, um die technischen Details der Kommunikation mit dem EIS zu verbergen. Anwendungen können dadurch einfacher und schneller Zugriff auf das System erhalten, ohne dabei auf die gewohnten Services des Application Server verzichten zu müssen (Listing 1).

Eigentlich hätte sich die Spezifikationsgruppe, der neben Sun unter anderen BEA, Inprise, IBM, SAP und Oracle angehören, an diesem



- Für viele E-Commerce-Angebote ist die Beteiligung von Legacy-Systemen unerlässlich, ihre Einbindung jedoch schwierig.
- Mit einem Zusatz zur J2EE-Spezifikation, der Connector Architecture, will Sun einen Standard für die Integration beliebiger IT-Systeme etablieren.
- Ob sich Suns Vorstellungen durchsetzen, ist noch nicht gesichert, viel hängt vom guten Willen der Softwarehersteller ab.



Die Connector Architecture: Verschiedene Verträge regeln die Zusammenarbeit von Resource Adapter und Application Server (Abb. 2).

Punkt entspannt zurücklehnen können. Die Hauptarbeit ist erledigt, einer Integration von IT-Systemen in Java-Anwendungen steht theoretisch nichts mehr im Weg. Doch die Connector Architecture bietet mehr: Durch die Einführung einer zweiten, diesmal standardisierten Schnittstelle für Anwendungskomponenten, dem Common Client Interface (CCI), will man die J2EE- zur EAI-Plattform (Enterprise Application Integration) ausbauen.

Perspektiven für Java-EAI

Das CCI soll die hersteller-spezifische Schnittstelle nicht ersetzen, sondern vor allem EAI-Werkzeugen den einheitlichen Zugriff auf alle Resource Adapter ermöglichen. Wie auch über das proprietäre Interface, werden über das CCI Aktionen in Form von RPCs an das EIS abgesetzt. Die Standardisierung macht das CCI unabhängig von einem bestimmten EIS. Alle Daten wie Parameter- und Typinformationen werden in einem Metadaten-Repository abgelegt. Hier bedient sich das EAI-Werkzeug, wenn es Aufrufe an das EIS absetzen will. Das CCI arbeitet ähnlich wie ein JDBC-Treiber: Es bildet die Schnittstelle zu allen Arten von Informationssystemen und lässt sich daher als Verallgemeinerung eines JDBC-Treibers betrachten (Abb. 3).

Was der JDBC-Standard für den Zugriff auf relationale Datenbanken ist, soll CCI für

nicht relational strukturierte Daten und sonstige Informationssysteme werden. Aufgrund der hohen Abstraktion vom EIS ist das CCI nicht als primäre Schnittstelle für die Anwendungsentwicklung geeignet, sondern vor allem für Entwicklungswerkzeuge ausgelegt, die den entsprechenden Code für den Zugriff generieren.

Die Zusammenarbeit von Connector Architecture, Resource Adapter und Message Driven Beans aus der EJB-Spezifikation 2.0 [1] zur asynchronen Kommunikation eröffnet neue Perspektiven für Java-basierte Lösungen im Bereich EAI. Auf Basis von Enterprise JavaBeans lässt sich eine einheitliche, objektorientierte Sicht auf die bestehenden Informationssysteme realisieren. Wenn diese Beans zusammen innerhalb einer verteilten JTS-Transaktion Daten in verschiedenen Backend-Systemen manipulieren könnten, wäre die Integration abgeschlossen.

Bis dahin ist der Weg allerdings noch weit. Beispielsweise wird die Implementierung des CCI in der Version 1.0 der Connector Architecture nicht zwingend vorgeschrieben sein. Zunächst müssen die Entwickler also weiterhin viel Gehirnschmalz in die Integration von bestehenden Systemen investieren.

Es hängt auch vom Willen der Softwarehersteller ab, ob sich die standardisierte Integration mittels Resource Adapter überhaupt durchsetzen wird. Wünschenswert wäre dies, denn nicht nur für

große EAI-Projekte, sondern auch für die Erstellung von Web-Frontends, hinter denen Legacy-Systeme arbeiten sollen, wäre die Standardisierung ein großer Fortschritt.

Mangelndes Engagement kann man Sun in diesem Zusammenhang nicht vorwerfen. Eine entsprechende Referenzimplementierung für die Entwicklung von Resource Adapter befindet sich bereits im Betastadium. Zudem soll die Connector Architecture Bestandteil der J2EE-Spezifikation 1.3 werden. Dies könnte für viele Hersteller von Application Server Grund genug sein, die Spezifikation umzusetzen, wenn sie weiterhin mit einer J2EE-konformen Implementierung werben möchten.

Schwierig: Verteilte Transaktionen

Wenn sich ein Application Server als EAI-Plattform positionieren will, ist es mit der Umsetzung der Spezifikation allein jedoch nicht getan. Vor allem Transaktionen über mehrere Ressourcen und Datenbanken werden weiterhin Schwierigkeiten verursachen. Die entsprechenden

Standards für die Durchführung von verteilten Transaktionen existieren schon lange, aber kaum ein ERP-System unterstützt sie.

Die fehlende Transaktionsklammer kann zu Inkonsistenzen in den beteiligten EIS führen. Hier ist der Entwickler nach wie vor gefragt, denn er muss solche Probleme erkennen und geeignete Gegenmaßnahmen einleiten. Knifflig bleibt auch die Einbindung von Individualsoftware, für die es keine Resource Adapter von der Stange zu kaufen geben wird. Selbstredend ließen sich auf Basis der Connector Architecture aber auch für diese Software individuelle Resource Adapter programmieren. (jd)

STEFAN KRIEGER

ist Principal Consultant im Bereich Distributed Systems bei der Comporsys Hansa GmbH, Hamburg.

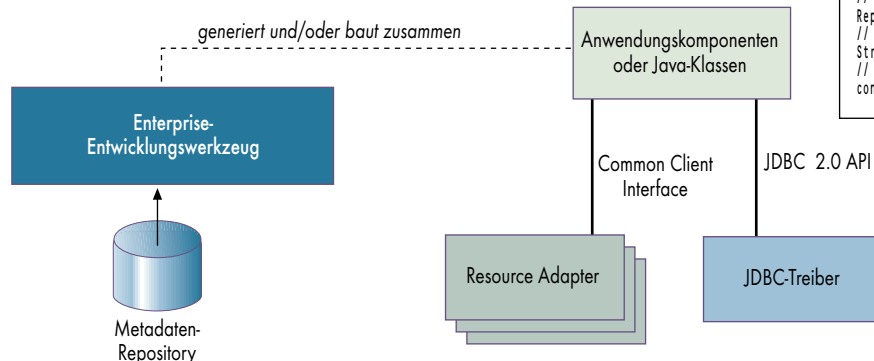
Literatur

- [1] Bernhard Merkle; EJB 2.0; Grosse Sprünge; Portable Beans und neue Query Language; *iX* 8/2000, S. 135

So könnte der Aufruf einer CICS-Transaktion über einen Resource Adapter aussehen.

LISTING 1

```
Context context = new InitialContext();
// Der Application Server konfiguriert die Factory für den Zugriff
// auf das gewünschte EIS und stellt sie über JNDI zur Verfügung.
ConnectionFactory cf = (ConnectionFactory)
context.lookup("de.comporsys.connector.cics.ConnectionFactory");
// Es wird eine Verbindung angefordert. Für die Anmeldung am EIS
// wird in diesem Fall die Benutzerinformationen explizit übergeben.
Connection connection = cf.getConnection("user", "password");
// Eine Aktion für das EIS soll aufgebaut werden
Statement statement = connection.createStatement();
// Die Aktion wird konfiguriert, hier mit
// dem Transaktionsnamen
RequestSpec requestSpec = new RequestSpec("PROGRAM");
// Die Anfrage wird erzeugt ...
Request request = statement.createRequest(requestSpec);
// ... Parameter werden gesetzt
request.setCommArea("UEBERGABEBEREICHE");
// ... und die Anfrage wird ausgeführt
Reply reply = statement.executeBrowse(request);
// Anschließend lässt sich das Ergebnis verarbeiten
String ergebnis = reply.getCommAreaAsString();
// Die Verbindung freigeben und in den Pool zurückstellen
connection.close();
```



Das Common Client Interface zur Anbindung von Anwendungskomponenten soll J2EE zur Plattform für die Enterprise Application Integration erheben (Abb. 3).