SOFTWIRED

Robuste verteilte Anwendungen mit JMS

Java Spektrum Feb. 2000

Silvano Maffeis, Thomas Haas, Bill Kelly

SoftWired AG
Technoparkstrasse 1
8005 Zürich
Tel: +41-1-445 23 70, Fax: +41-1-445 23 72
http://www.softwired-inc.com
info@softwired-inc.com

Inhaltsverzeichnis

1. Zuverlässige Kommunikation Dank Messaging

Bei der Übermittlung von Geschäftsdaten kann man es nicht darauf belassen, die Finger zu kreuzen und zu hoffen, dass die Daten heil ankommen. Geschäftssysteme werden zudem je länger je verteilter und heterogener. Anwendungen, welche bis anhin einem Inseldasein frönten, müssen nun mit anderen (bestehenden und neuen) Anwendungen interagieren und kommunizieren. Zu allem Übel kommt hinzu, dass die Kommunikationsverbindungen zwischen Anwendungen ab und zu ausfallen. Um in solchen Situationen dennoch die Übermittlung von Daten zwischen verteilten Anwendungen garantieren zu können, wird seit jeher auf *Message Oriented Middleware* (MOM) zurückgegriffen.

MOM erlaubt eine lose Kopplung von verteilten Anwendungen, sowie eine hohe Zuverlässigkeit bei der Kommunikation zwischen Anwendungen. Das Programmiermodell der MOM ist Ereignis-orientiert und beruht auf dem Austausch von selbstbeschreibenden Nachrichten.

In Ausgabe 6/1999 von JavaSpektrum wurden Konzept und Funktionsweise von MOM erläutert. Der vorliegende Artikel gibt eine technische Einführung in den Java Message Service (JMS) Standard von Sun, und stellt ein JMS kompatibles MOM Produkt vor - der iBus der Firma Softwired.

2. Endlich ein Messaging Standard

MOM Produkte werden seit den siebziger Jahren entwickelt und erfolgreich eingesetzt. Bis vor kurzem hatten MOM Produkte vor allem eins gemeinsam: Jeder Hersteller kochte sein eigenes Süppchen, was die Programmierschnittstelle (API) zu seinem Produkt anging. Dies hat sich nun aber geändert. Führende MOM Hersteller haben sich unter der Schirmherrschaft von Sun zusammengetan und einen offenen Standard für Java MOM Produkte geschaffen: Der Java Message Service.

Immer weniger Unternehmen sind willens, proprietäre MOM APIs einzusetzten. Auch bei MOM besteht ein starker Trend in Richtung offener Systeme und Java. JMS ermöglicht uns verteilte Anwendungen zu entwickeln, ohne uns von proprietären messaging APIs abhängig zu machen.

2.1. Zwei Kommunikationsmodelle

JMS standardisiert zwei Kommunikationsmodelle: Punkt-zu-Punkt sowie Publish-Subscribe. Dem ersten Modell liegen *Warteschlangen* zugrunde. Diese dienen der zeitlichen und örtlichen Entkopplung von Anwendungen. Nachrichten werden produziert und in eine oder mehrere JMS Warteschlangen eingefügt. Diese bleiben dort persistent gespeichert bis sie von einem Konsumenten

abgerufen werden. Im Punkt-zu-Punkt Modell kann eine Nachricht genau einmal konsumiert werden. Das Punkt-zu-Punkt Modell wird vor allem dort eingesetzt, wo eine zeitliche Entkopplung zwischen Produzenten und Konsumenten von Nachrichten stattfinden muss, und wo jede Nachricht von nur einem Konsumenten empfangen werden darf.

Im zweiten Modell werden Nachrichten sogenannten *Themen* (engl. *topics*) zugeordnet. Produzenten erzeugen Nachrichten und publizieren (publish) diese anschliessend unter einem oder mehreren Themen. Konsumenten können diese Themen abonnieren (subscribe). Sie erhalten alle Nachrichten, welche unter dem entsprechenden Thema publiziert werden. Sind mehrere Konsumenten für ein bestimmtes Thema abonniert, so werden Nachrichten an alle Konsumenten übermittelt (multicast). Publish/Subscribe wird vor allem für Systeme eingesetzt, in denen Nachrichten in Echtzeit übermittelt werden müssen, oder in Systemen, in denen dieselbe Nachricht von mehreren Konsumenten empfangen werden muss.

2.2. Integraler Bestandteil von J2EE

JMS ist ein Bestandteil der J2EE Plattform. Zur Verwaltung von Themen und Warteschlangen greift JMS auf den *Java Naming and Directory Interface* (JNDI) Standard zu. JMS kann zudem mittels *Java Transaction Service* (JTS) mit Transaktionsmonitoren zusammenarbeiten, um das Senden und Empfangen von Nachrichten mittels verteilten Transaktionen zu steuern. So ist es zum Beispiel möglich, JMS Operationen unter dem Kontext von existierenden Datenbanktransaktionen ablaufen zu lassen. Schlägt die Transaktion fehl, so werden nicht nur allfällige Änderungen an Datenbeständen widerrufen, der Empfang der in dieser Transaktionen versendeten Nachrichten wird ebenfalls unterbunden.

EJB mit JMS ist auch ein interessantes Thema. Obschon ein EJB problemlos JMS Nachrichten erzeugen und senden kann, sieht die Version 1.1 der EJB Spezifikation noch keinen Mechanismus vor, welcher einem EJB erlaubt, sich als Konsument von Nachrichten zu registrieren. Ein EJB möchte in diesem Fall aktiviert werden, wenn eine für sie bestimmte Nachricht ankommt. Erst EJB Version 2.0 wird eine vollständige Integration von EJB und JMS bieten.

3. Nachrichten Typen

Der typische Ablauf einer JMS Produzenten-Anwendung besteht darin, dass ein JMS Nachrichtenobjekt erzeugt wird, Daten in die Nachricht eingebettet werden und die Nachricht schliesslich einer Warteschlange oder einem Thema übergeben wird. Die Konsumenten-Anwendung erhält diese Nachricht und greift auf deren Inhalt zu.

JMS Nachrichten bestehen aus einem Kopf und aus einem Inhalt. Der Kopf enthält Angaben zur Priorität der Nachricht, deren Lebensdauer, Destination usw. Der Inhalt enthält die eigentlichen Nutzdaten. Um die vielfältigsten Nutzdaten einpacken zu können, sieht JMS fünf Typen von Nachrichten vor:

- `TextMessage`: enthält eine Zeichenkette - z.B. XML Ausdrücke.
- `MapMessage`: enthält Name/Wert Paare, ähnlich wie eine JDK `Hashtable`.
- `StreamMessage`: besitzt eine Schnittstelle ähnlich zu den JDK input und output streams.
- `BytesMessage`: enthält uninterpretierte Bytes.
- `ObjectMessage`: enthält ein serialisierbares Java Objekt.

4. Flexible Dienstqualität

JMS sieht verschiedene Mechanismen vor, mit denen die Zustellung von Nachrichten, und somit die "Dienstqualität", beinflusst werden kann. Als stärkste Dienstqualität bietet JMS die *guaranteed delivery* an. Selbst bei Ausfällen von beteiligten Anwendungen oder Netzwerkverbindungen, garantiert JMS, dass Nachrichten exakt einmal zugestellt werden. Typischerweise ist bei *guaranteed delivery* mit Leistungseinbussen zu rechnen, weil jede Nachricht in einer Datenbank zwischengelagert wird und weil JMS Transaktionen zum Einsatz kommen. Folglich bietet JMS verschiedene Mechanismen an um Empfangsbestätigungen sowie Nachrichtenpersistenz an die genauen Anforderungen der Anwendungen anzupassen, sodass ein guter Durchsatz bei genügend hoher Ausfalltoleranz erzielt werden kann. Diese Mechanismen werden kurz erläutert.

4.1. Persistente Nachrichten

Im JMS Publish/Subscribe Modell wird zwischen volatilen und persistenten Nachrichten unterschieden. Ist ein Kosument auf einem bestimmten Thema nicht abonniert, so erhält er keine *volatilen* Nachrichten, welche zur Zeit publiziert werden. Klinkt sich ein Konsument in ein Thema ein, so erhält er zuerst alle *persistenten* Nachrichten, welche währen seiner Inaktivität publiziert wurden. Persistente Nachrichten werden deshalb von der JMS Middleware in einer Datenbank zwischengelagert.

Beim Punkt-zu-Punkt Modell gibt es keine volatilen Nachrichten: Alle Nachrichten sind persistent.

4.2. Empfangsbestätigungen

In beiden Modellen können Nachrichten entweder explizit oder implizit durch die Konsumenten bestätigt werden. Wählt ein Konsument die *implizite* Bestätigung, so werden Nachrichten automatisch und einzeln von der JMS Middleware bestätigt. Bei der *expliziten* Bestätigung, werden die Nachrichten

durch den Konsumenten bestätigt. Dabei können die Nachrichten einzeln oder gruppenweise bestätigt werden.

Eine persistente Nachricht gilt erst dann als empfangen, wenn diese bestätigt wurde. Fällt der Konsument aus, bevor er die Bestätigung absenden konnte, so erhält er nach einem Neustart alle nicht-bestätigten Nachrichten.

4.3. Lebenszeit von Nachrichten

Nachrichten können mit einer Lebenszeit versehen werden. Nachrichten, deren Lebenszeit abgelaufen ist, werden aus der Datenbank entfernt und folglich nicht zugestellt.

4.4. Prioritäten

Nachrichten können zudem mit Prioritäten versehen werden. Liegen Nachrichten unterschiedlicher Priorität in einer Warteschlange oder auf einem Thema vor, so werden dringendere Nachrichten zuerst ausgeliefert. Dies ist die einzige Situation in der sich Nachrichten in einer JMS Middleware zeitlich "überholen" können. Ansonsten werden Nachrichten strikte in der Reihenfolge konsumiert, in der sie auch produziert wurden.

4.5. Transaktionen

JMS bietet die Möglichkeit, die Übermittlung von Nachrichten mittels Transaktionen zu steuern. So kann eine Anwendung eine Transaktion eröffnen, Nachrichten an etliche Warteschlangen oder Themen senden, und dann das gesamte Paket bestätigen oder verwerfen. Wird die Transaktion verworfen, so ist der Effekt derselbe als seien die Nachrichten gar nie aufgegeben worden.

5. Punkt-zu-Punkt Kommunikation

Die Punkt-zu-Punkt Kommunikation wird massgeblich mittels den JMS Klassen `Queue`, `QueueReceiver` und `QueueSender` abgewickelt. Das folgende Beispiel zeigt eine JMS Anwendung, welche eine Bestellung in eine JMS Warteschlange einfügt.

5.1. Produzent

```
// Erzeuge einen JNDI Kontext:
Context ctx = new InitialContext();

// Erhalte eine QueueConnectionFactory durch JNDI:
QueueConnectionFactory factory =
  (QueueConnectionFactory)ctx.lookup("QueueConnectionFactory");

// Erzeuge eine QueueConnection:
QueueConnection connection = factory.createQueueConnection();

// Erzeuge eine nicht-transaktionale JMS session, mit
// impliziter Empfangsbestätigung:
QueueSession session
  = connection.createQueueSession(false,
  QueueSession.AUTO_ACKNOWLEDGE);

// Erhalte die Warteschlange für Bestellungen von JNDI:
Queue ordersQueue = (Queue)ctx.lookup("Orders");

// Erzeuge einen Sender für die Warteschlange:
QueueSender sender = session.createSender(ordersQueue);

// Erzeuge eine Text Nachricht:
StringBuffer orderStr = new StringBuffer("100 Barrels Crude Oil");
TextMessage order = session.createTextMessage(orderStr);

// Füge die Nachricht in die Warteschlange ein:
sender.send(order);
```

Die Initialisierung einer JMS Punkt-zu-Punkt Anwendung besteht im Anlegen eines JNDI Kontext, einer `QueueConnectionFactory`, einer `QueueConnection` und einer `QueueSession`. Diese Schritte verlaufen praktisch immer identisch und sollten deshalb wohl besser durch eine Hilfsklasse wahrgenommen werden.

`QueueConnectionFactory` und `Queue` sind *administrierte* JMS Objekte, diese werden typischerweise mittels einer Dienstanwendung vom Entwickler oder Systemadministrator angelegt und in einen JNDI Kontext eingetragen.

Eine `QueueConnection` stellt eine Verbindung zwischen einer JMS Anwendung und der JMS Middleware dar. Eine solche Verbindung kann gestartet oder gestoppt werden. Damit wird wahlweise die Zustellung von Nachrichten ermöglicht oder temporär unterbunden.

Eine `QueueSession` wird benötigt, um `QueueSender` und `QueueReceiver` Objekte zu erzeugen. Zudem stellt `QueueSession` auch Methoden zur Verwaltung von Transaktionen zur Verfügung.

Nach erfolgter Initialisierung können wir schliesslich die Warteschlange für unsere Bestellungen beziehen, einen Sender anlegen, und eine Nachricht mit einer Bestellung in die Warteschlange einfügen.

5.2. Konsument

Eine Anwendung, welche Bestellungen aus der Warteschlange entfernt, hat einen ähnlichen Aufbau wie unser Produzent von Bestellungen. Anstatt eines `QueueSenders` wird jedoch ein `QueueReceiver` angelegt. Mit dessen `receive` Methode werden die Nachrichten empfangen. Da wir nun eine explizite Empfangsbestätigung wählen (siehe `session.CLIENT_ACKNOWLEDGE` im nächsten Code Beispiel), müssen wir die empfangenen Nachrichten mittels `TextMessage.acknowledge()` bestätigen. Erst nach erfolgter Bestätigung verschwinden diese aus der Warteschlange.

```
// Erzeuge einen JNDI Kontext:
Context ctx = new InitialContext();

// Erhalte eine QueueConnectionFactory durch JNDI:
QueueConnectionFactory factory =
  (QueueConnectionFactory)ctx.lookup("QueueConnectionFactory");

// Erzeuge eine QueueConnection:
QueueConnection connection = factory.createQueueConnection();

// Erzeuge eine nicht-transaktionale JMS Session, mit
// expliziter Empfangsbestätigungen:
QueueSession session
  = connection.createQueueSession(false,
    QueueSession.CLIENT_ACKNOWLEDGE);

// Erhalte die Warteschlange für Bestellungen von JNDI:
Queue ordersQueue = (Queue)ctx.lookup("Orders");

// Erzeuge einen Empfänger für die Warteschlange:
QueueReceiver receiver = session.createReceiver(ordersQueue);

// Nachrichtenempfang zulassen:
connection.start();

// Empfange eine Nachricht aus der Warteschlange:
TextMessage order = (TextMessage)receiver.receive();

// Zeige die Bestellung an:
System.out.println(order.getText());

// Entferne die Nachricht aus der Warteschlange:
order.acknowledge();
```

6. Publish-Subscribe Kommunikation

Publish/Subscribe Anwendungen sind ähnlich aufgebaut wie Punkt-zu-Punkt Anwendungen. Es genügt weitgehend das Symbol "`Queue`" durch "`Topic`" zu ersetzen. Im folgenden zeigen wir einen Produzenten, welcher den Wechselkurs für den Euro unter dem Thema "`/reuters/fx/EUR`" publiziert. Für dasselbe Thema können sich beliebig viele Konsumenten abonnieren. Der Wechselkurs wird an *alle* Konsumenten übermittelt, diese können den Kurs dann z.B. in einem Echtzeit-Spreadsheet darstellen.

```
// Erzeuge einen JNDI Kontext:
Context ctx = new InitialContext();

// Erhalte eine TopicConnectionFactory durch JNDI:
TopicConnectionFactory factory =
  (TopicConnectionFactory)ctx.lookup("TopicConnectionFactory");

// Erzeuge eine TopicConnection:
TopicConnection connection = factory.createTopicConnection();

// Erzeuge eine nicht-transaktionale JMS Session, mit
// impliziter Empfangsbestätigung:
TopicSession session
  = connection.createTopicSession(false,
    TopicSession.AUTO_ACKNOWLEDGE);

// Erhalte das Thema für den Euro:
Topic euroTopic = (Topic)ctx.lookup("/reuters/fx/EUR");

// Erzeuge einen Sender für das Thema:
TopicPublisher publisher = session.createPublisher(euroTopic);

// Erzeuge eine Text Nachricht:
StringBuffer quoteStr = new StringBuffer("08:33:15 1.0252");
TextMessage quote = session.createTextMessage(quoteStr);

// Publiziere die Nachricht unter dem Euro Thema:
publisher.publish(quote);
```

7. Die iBus JMS Implementation

Obschon der JMS Standard noch recht jung ist, geben verschiedene Firmen bereits an, kompatible MOM Produkte liefern zu können[1]. Die angebotenen Produkte unterscheiden sich in Bezug auf Fertigstellungsgrad (manche Produkte unterstützen nur eines der beiden JMS Kommunikationsmodelle), Skalierbarkeit, Ausfalltoleranz, Dokumentation, und Support. Einige Produkte bestehen aus einer JMS "Hülle" zu einem althergebrachten MOM System, z.B. zu IBM MQSeries oder TIBCOs Informationsbus. Andere JMS Produkte sind pur in Java realisiert.

Zur letzteren Kategorie gehört auch der iBus der Firma SoftWired. iBus ist eine flexible, pur in Java realisierte JMS Middleware. Sie zeichnet sich dadurch aus, dass eine Vielzahl von Transport Protokollen in die Middleware "eingeklinkt" werden können, wie z.B. TCP/IP, SSL, HTTP, IP Multicast, aber auch Protokolle zur drahtlosen Kommunikation, z.B. SMS und WAP[2].

[1] Eine Liste der JMS Hersteller befindet sich unter http://www.java.sun.com/products/jms/vendors.html

[2] Wireless Application Protocol, http://www.wapforum.org

7.1. Eine Messaging Lösung

iBus umfasst gleich eine gesamte Messaging Produktlinie. Diese Besteht aus zwei Kernprodukten, der iBus//MessageServer und der iBus//MessageBus, sowie aus Zusatzprodukten. Die Zusatzprodukten umfassen eine JMS Bibliothek für C und C++, Software für die Nutzung von JMS in Web Anwendungen, und Software für die Übermittlung von JMS Nachrichten mittels Drahtlos-Protokollen.

Das Ziel von iBus ist, unternehmensweite Messaging Lösungen zu ermöglichen. Die wichtigen Aspekte sind dabei Skalierbarkeit, Verfügbarkeit und Sicherheit. Dazu bietet iBus Mechanismen wie Multicast-Kommunikation, Clustering, Nachrichten Routing und Zugriffskontrolle auf Themen und Warteschlangen an.

7.2. Architekturmerkmale

Sämtliche iBus Produkte beruhen auf JMS. iBus//MessageServer und iBus//MessageBus unterscheiden sich bezüglich Architektur und Dienstqualitäten. Nur iBus//MessageServer unterstützt beide JMS Modelle, sowie sämtliche von JMS geforderten Dienstqualitäten (Transaktionen, persistente Themen, Empfangsbestätigungen).

iBus//MessageServer beruht auf einem netzwerkzentrischen, ausfalltoleranten Nachrichten-Server. Der Server ist mit einer Datenbank verbunden, in diese werden Nachrichten für Warteschlangen und persistente Themen abgelegt (Abbildung 1). Nachrichten können zudem zwischen Nachrichtenserver ge-routed werden, auf diese Weise können unternehmensweite Systeme elegant und robust realisiert werden (Abbildung 3).

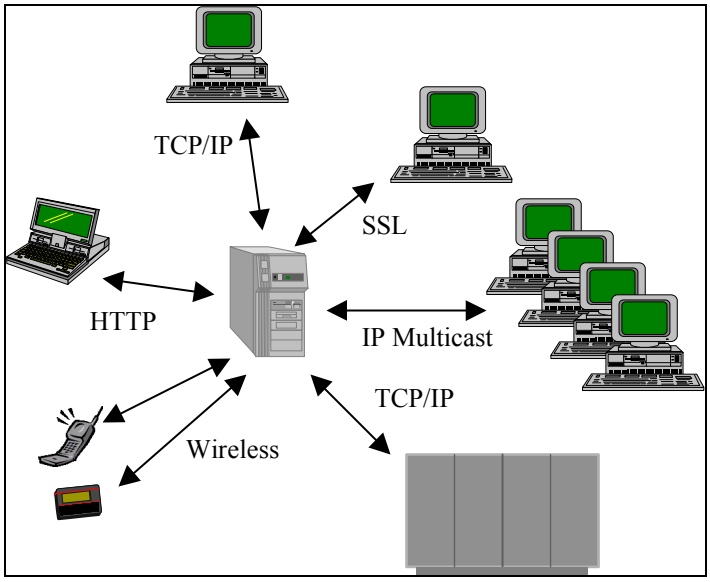

Abbildung 1: iBus//MessageServer Architektur

iBus//MessageBus wird hingegen dann eingesetzt, wenn aus technischen Gründen keine serverzentrische Lösung angestrebt werden soll. Hauptanwendungsgebiete von iBus//MessageBus sind eingebettete, verteilte Systeme, z.B. im Telekombereich, sowie Satellitennetze. In iBus//MessageBus kommunizieren Anwendungen direkt miteinander über multicast-fähige Protokolle (Abbildung 2). Die iBus//MessageBus Architektur ist vollständig verteilt und sieht keine Nachrichtenserver vor. Dieses Produkt unterstützt nur das JMS Publish/Subscribe, jedoch nicht das JMS Punkt-zu-Punkt Modell.

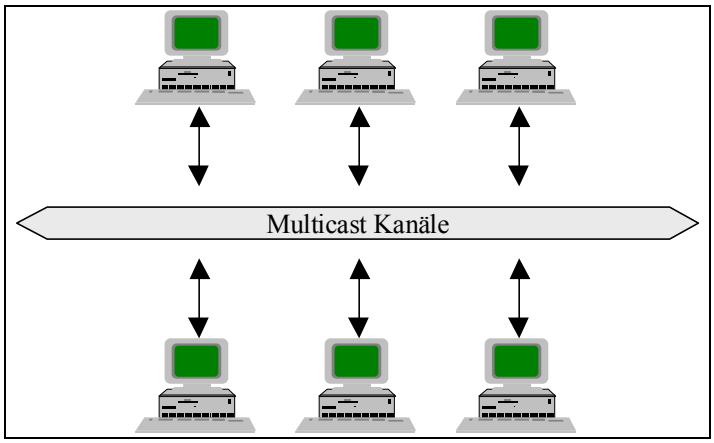

Abbildung 2: iBus//MessageBus Architektur

8. Ausblick

JMS bietet die nötigen Voraussetzung um unternehmensweite robuste Informationssysteme zu realisieren. Die beiden JMS Kommunikationsmodelle, Punkt-zu-Punkt und Publish/Subscribe, ermöglichen eine zeitlich entkoppelte Kommunikation zwischen verteilten Anwendungen, sowie die Übermittlung wichtiger Nachrichten in Echtzeit.

JMS wurde unter der Annahme Konzipiert, dass Anwendungen, Computer und Kommunikationsverbindungen zeitweise ihren Geist aufgeben werden. Selbst in solchen Fällen garantiert JMS die Zustellung von Nachrichten.

JMS Middleware wie iBus wird heute in zunehmendem Masse für verteilte Informationssysteme im Finanz- und Industriebereich eingesetzt. Nachrichtenrouting ermöglicht es, ein laufendes System zu skalieren, in dem Nachrichten vom Nachrichtenserver einer Abteilung transparent an den Nachrichtenserver einer anderen Abteilung weitergeleitet werden (Abbildung 3). Als wichtiger Bestandteil von J2EE wird JMS vermehrt dort genutzt, wo J2EE kompatible Applikationsserver zum Einsatz kommen.

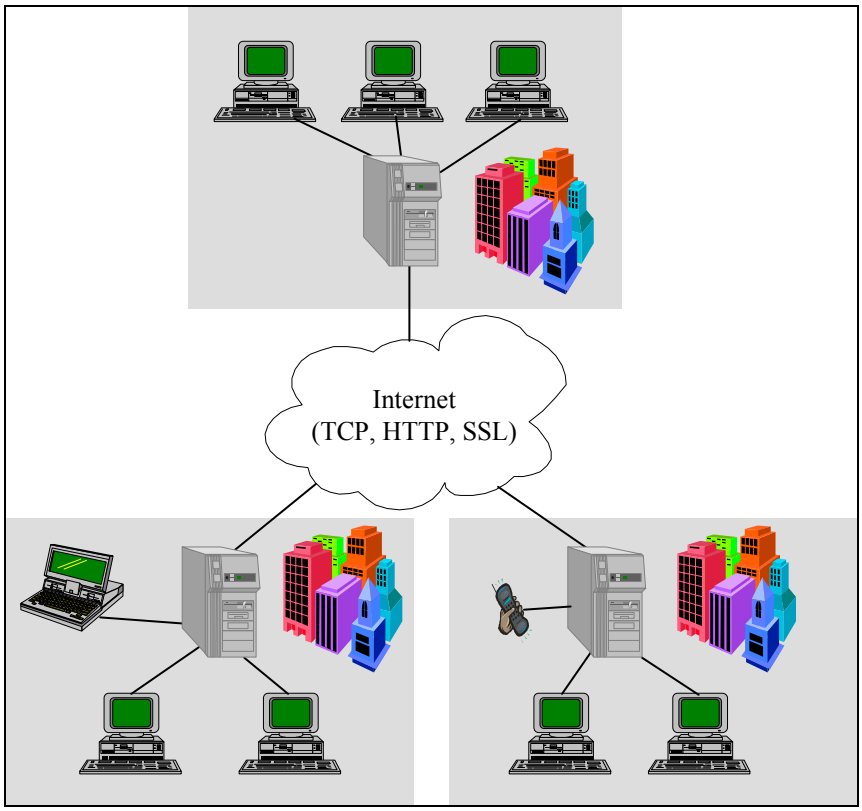

Abbildung 3: Nachrichten-routing in unternehmensweiten Systemen.

Ein Blick in die nahe Zukunft verrät, dass Mobile Endgeräte und Drahtloskommunikation neue Umgangsformen mit betrieblichen Informationssystemen ermöglichen. Geht es darum, Anwendungen auf einem Mobilgerät mit einem Informationssystem zu verbinden, so ermöglicht JMS auch hier robuste Lösungen in kurzer Zeit zu realisieren. Messaging bietet ja ein lose gekoppeltes Kommunikationsmodell an, in welchem Nachrichten an Anwendungen gesandt werden können, ohne eine stetige Netzwerkverbindung zu diesen Anwendungen vorauszusetzen.

9. Links

- JMS Home Page: http://www.java.sun.com/products/jms
- J2EE Home Page: http://java.sun.com/j2ee
- iBus Home Page: http://www.JavaMessaging.com

Silvano Maffeis ist CTO, Thomas Haas und Bill Kelly sind Senior Software Engineers bei SoftWired AG in Zürich. Die Autoren können erreicht werden unter info@softwired-inc.com.

